MetaGraph: Indexing and Analysing Nucleotide Archives at Petabase-scale

Mikhail Karasikov, [1,2,3,4,‡] Harun Mustafa, [1,2,3,4,‡] Daniel Danciu, [1,2] Marc Zimmermann, [1,2] Christopher Barber, [1,2] Gunnar Rätsch, [1,2,3,4,*] and André Kahles [1,2,3,4,*]

[1]Biomedical Informatics Group, Department of Computer Science, ETH Zurich, Zurich, Switzerland
[2]Biomedical Informatics Research, University Hospital Zurich, Zurich, Switzerland
[3]Swiss Institute of Bioinformatics, Zurich, Switzerland
[4]Department of Biology, ETH Zurich, Zurich, Switzerland

([‡]Equal contribution. [*]To whom correspondence should be addressed.)

{raetsch,andre.kahles}@inf.ethz.ch

Abstract

The amount of biological sequencing data available in public repositories is growing exponentially, forming an invaluable biomedical research resource. Yet, making all this sequencing data searchable and easily accessible to life science and data science researchers is an unsolved problem. We present *MetaGraph*, a versatile framework for the scalable analysis of extensive sequence repositories. *MetaGraph* efficiently indexes vast collections of sequences to enable fast search and comprehensive analysis. A wide range of underlying data structures offer different practically relevant trade-offs between the space taken by the index and its query performance. *MetaGraph* provides a flexible methodological framework allowing for index construction to be scaled from consumer laptops to distribution onto a cloud compute cluster for processing terabases to petabases of input data. Achieving compression ratios of up to 1,000-fold over the already compressed raw input data, *MetaGraph* can represent the content of large sequencing archives in the working memory of a single compute server. We demonstrate our framework's scalability by indexing over 1.4 million whole genome sequencing (WGS) records from NCBI's Sequence Read Archive, representing a total input of more than three petabases.

Besides demonstrating the utility of *MetaGraph* indexes on key applications, such as experiment discovery, sequence alignment, error correction, and differential assembly, we make a wide range of indexes available as a community resource, including those over 450,000 microbial WGS records, more than 110,000 fungi WGS records, and more than 20,000 whole metagenome sequencing records. A subset of these indexes is made available online for interactive queries. All indexes created from public data comprising in total more than 1 million records are available for download or usage in the cloud.

As an example of our indexes' integrative analysis capabilities, we introduce the concept of differential assembly, which allows for the extraction of sequences present in a foreground set of samples but absent in a given background set. We apply this technique to differentially assemble contigs to identify pathogenic agents transfected via human kidney transplants. In a second example, we indexed more than 20,000 human RNA-Seq records from the TCGA and GTEx cohorts and use them to extract transcriptome features that are hard to characterize using a classical linear reference. We discovered over 200 trans-splicing events in GTEx and found broad evidence for tissue-specific non-A-to-I RNA-editing in GTEx and TCGA.

1 Introduction

For more than a decade, continuing innovation in the area of high-throughput sequencing has propelled research in the biomedical domain and led to an exponential growth in worldwide sequencing capacity [58]. As a consequence, sequencing costs for entire human genomes have dropped well below the critical mark of 1,000 USD per sample, bringing the new critical line of 100 USD into near reach. This also results in an exponential growth of sequencing data available in public and controlled-access repositories. The number of sequenced nucleotides contained in the European Nucleotide Archive (ENA) currently doubles every 27 months, resulting in a current size of close to $1.6{\cdot}10^{16}$ nucleotide bases (16 petabases) [9] of raw read data. Transmitting the entirety of such a data set across a wire for any kind of access is clearly uneconomical and renders it currently virtually inaccessible to the broader research community for comprehensive analyses. However, even for defined subsets of samples that are collected within larger-scale studies by international consortia, such as The Cancer Genome Atlas (TCGA) [61], the Genotype Tissue Expression (GTEx) [42] project, or the MetaSUB project [19], the entire data of a single study can comprise hundreds of terabytes, making access complicated.

The classical pattern for accessing sequencing data on public repositories is to identify relevant samples using descriptive metadata and to extract a copy or a slice of the data for further processing. More recently, repositories started mirroring their contents into cloud storage, addressing the download problem, but at the same time often creating additional costs for off-the-premise compute. Today's existing infrastructure is largely adapted to this pattern of access, only indexing the metadata (e.g., sample and study ID, organism name, taxonomic information, related publications, etc.) of all samples to make it accessible and easily searchable. However, any query involving the raw sequencing data itself requires a copy of the data, complicating its analysis and reducing its accessibility for many researchers. We argue that this significantly limits the potential that sequence repositories bear for life science research. To address this issue, we propose a scalable approach to index such large repositories of sequencing data, transforming them into a highly compressed and more accessible representation for downstream analysis. A main focus of this work is thereby on scalability, allowing the processing of sequence collections on a petabase scale.

Driven by the open-data movement and the advances in high-throughput sequencing technology, it is our expectation that the number of studies aggregating large sample cohorts will further increase in the near future. In fact, we envision that making such data shareable and searchable will become a problem of high practical relevance.

As indexing large sequence collections also poses interesting algorithmic questions, different technical solutions addressing these questions have already been proposed in the recent past. Naturally, a first focus lies on making the genetic variation in large cohorts, especially in human, accessible for biomedical research and medicine. Only very recently frameworks for variation graphs, such as VG [23], and methods for compressing haplotypes [21] or paths in graphs more generally [46], have improved variation-aware alignment and variant calling in general [34, 29]. While successful for the analysis of single-species cohorts, these methods struggle to represent the large variability present in distantly related organisms or arising from metagenomic applications.

Hence, a second focus of the algorithmic work has been put on the *sequence* or *experiment discovery problem*: querying a sequence of interest (e.g., a transcript or an entire genome) against all samples available in sequence repositories. For collections of assembled genomes, the BLAST approach [7] has been in heavy use since 30 years, but lacks the scalability to allow for high throughput searches on highly diverse sequence collections or to allow for the search in raw, unassembled reads.

The methods currently available for solving the experiment discovery problem can be grouped into three main categories: i) Methods based on sketching techniques, which summarize the input data using one or multiple hashing operations and then use these summaries (sketches) to estimate distances between query and target. Examples are MASH [51, 50] and KrakenUniq [15].

ii) Methods employing Bloom filter based data structures to allow for approximate membership queries. Examples are (Split)-Sequence Bloom Tree [57, 27], Bloom Filter Trie [30], BIGSI [14], and COBS [12]. iii) Methods for the exact representation of *annotated de Bruijn graphs*, also called *colored de Bruijn graph*, storing additional metadata as labels of its nodes or edges [32]. Examples are Mantis [53, 5], VARI [45], and others [3, 39]. These methods approach experiment discovery by matching k-mers from query sequences against those stored in an index. Alongside these, there has also been growing interest in the use of de Bruijn graph-based indexes for alignment tasks as a way to accelerate alignment to repeat-prone reference genomes [41] or to unassembled read sets [40, 28]. More recent work has focused on improving the scalability of these approaches, either through strategies using more rigorous early cut-off criteria [33], or via the introduction of heuristics [55]. A major challenge faced by all existing methods is to unite the ability to efficiently operate on petabase scale input data with the capacity for fast and versatile query operations.

In addition to the more recent use in addressing the experiment discovery problem, de Bruijn graphs are also known as a classic framework for read set representation in sequence assembly in both single-sample [11, 47, 24, 25, 39] and multi-sample [32, 60] settings. Of these, the HipMER [24] and MetaHipMER [25] single-sample assemblers scale to massive data sets stored in distributed hash tables. None of these methods, however, tackles the problem of assembly from an integrative analysis perspective. Examples of such queries can include a *core genome assembly*, where sequences common to all samples in a cohort are desired [44, 43], or a *differential assembly*, where sequences shared by a foreground cohort and absent from a background cohort are sought. We introduce the latter concept as an application made tractable at scale by the framework proposed in this work.

To bridge this evident gap in the landscape of sequence analysis tools for large data collections, we present *MetaGraph*, a versatile framework for indexing and analysis of biological sequence libraries at petabase scale. While the approach is not restricted to any specific input alphabet, for the remainder of this work, we will focus on nucleotide sequences originating from DNA or RNA sequencing samples. *MetaGraph* enables building indexes of large collections of sequences, employing an annotated *de Bruijn* graph for sequence search and assembly.

The *MetaGraph* framework provides a wide range of compressed data structures for transforming very large sequencing archives into k-mer dictionaries, associating each k-mer with labels representing metadata associated with its originating sequences.

The data structures underlying *MetaGraph* are designed to balance the trade-off between the space taken by the index and the time needed for query operations. A main design goal of our framework is to allow both performing experiments on single desktop computers and scaling up to distributed compute clusters. This is achieved through a modular approach, efficient parallelization and the computation in external memory. We describe these aspects in the first part of the Results section along with our assessment of scalability.

We then outline results using *MetaGraph* to index data from a diverse collection of public sources, ranging from large RNA-Seq cohorts like TCGA [61] and GTEx [42], over vast archives of whole genome sequencing (WGS) records comprising over 1 million samples of microbial, fungal, plant, and metazoa organisms currently available in the Sequence Read Archive (SRA) [37], to large sets of highly diverse whole metagenome sequencing samples, like the MetaSUB [19] set or all available human gut metagenome samples. The total amount of sequences indexed in these graphs exceeds by far the crucial figure of one petabase and at last makes this data fully and efficiently searchable by sequence.

Finally, we use these and some smaller data sets not only to demonstrate the scalability and performance of *MetaGraph* but also to demonstrate how the graph indexes can be used for biological discovery.

2 Results

2.1 A powerful framework for efficient sequence representation

The *MetaGraph index* consists of two main components: i) a de Bruijn graph and ii) its annotating metadata (**Figure 1**, middle right). The graph is represented by a *k-mer dictionary* mapping all k-mers observed in the input sequences to unique positive integer identifiers assigned to them. These k-mers serve as elementary tokens in all operations on the *MetaGraph* index.

The metadata on the graph is a *binary relation* between k-mers and their labels representing any categorical features, such as source sample IDs or quantized expression levels. This relation is represented as a sparse binary matrix called the *annotation matrix*, with one row for each k-mer and one column for each label. The $(i, j)^{\text{th}}$ element of this matrix has value 1 if and only if the i^{th} k-mer is associated with the j^{th} label. This matrix can be of an enormous size, containing up to one trillion rows and several hundred thousand columns in the experiments we present. Yet, due to its sparsity, it can be efficiently compressed.

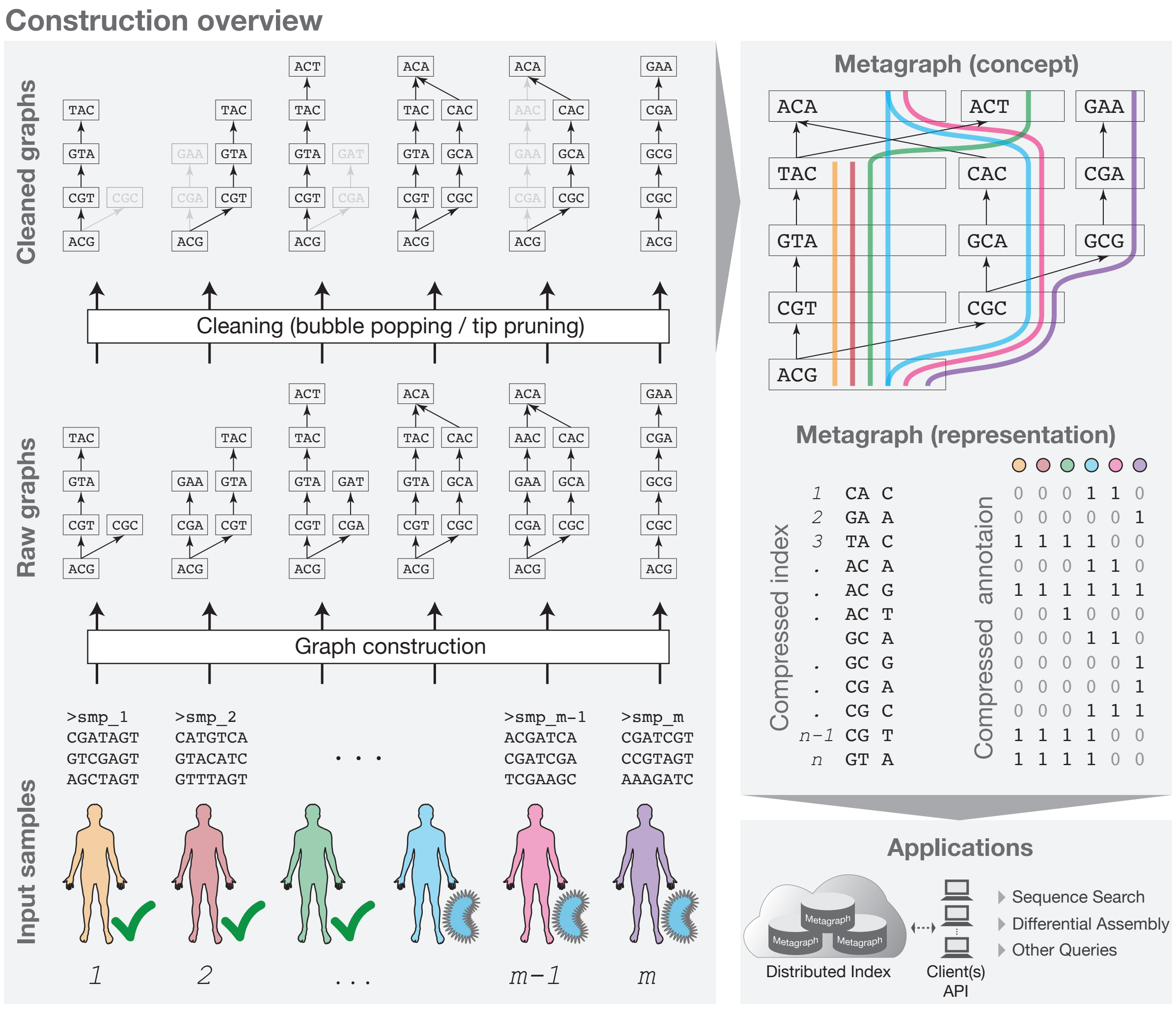


Figure 1: The *MetaGraph* framework – Schematic overview of graph construction and representation. **Left:** Individual sequencing samples are assembled into raw graphs which are then cleaned to remove noisy and erroneous paths. **Right:** Individual graphs are combined into the *MetaGraph* index (top), consisting of the compressed sequence index and a compressed annotation matrix (middle). *MetaGraph* is then used as the basis for downstream applications, such as sequence search, differential assembly, and other queries (bottom).

A notable feature of our framework is that its theoretical concepts, as well as their actual implementation, support the indexing of sequences over arbitrary finite alphabets. Thus, *MetaGraph* can be used for indexing biological sequences of all kinds, such as raw DNA/RNA sequencing reads, assembled genomes, but also protein sequences. The framework is modular in nature, enabling the use of a variety of interchangeable representations for storing the sequence index and the annotation matrix. These representations may be chosen to optimise the space usage for different kinds of inputs and the execution time of desired queries. *MetaGraph* is open source and its code, links to pre-compiled binaries and application examples are available at `https://github.com/ratschlab/metagraph`.

Scalable multi-sample index construction The workflow for constructing the *MetaGraph* index consists of three main stages: data pre-processing, graph construction, and annotation.

Typically, the first stage (data pre-processing) involves the construction of separate de Bruijn graphs from the raw data associated with each of the future annotation labels (e.g., the input samples in **Figure 1**, bottom left). In general, the annotation labels can be defined to represent any feature of the input sequences, such as sample ID and organism, expression quantile, or chromosome number. We apply a subsequent cleaning step to each of the graphs to remove possible sequencing errors (**Figure 1**, top left). This step is not performed on graphs constructed from data considered to be error-free, such as assembled genomes. The resulting graphs are then stored as a minimal set of linear paths, called *contigs*, covering all nodes in the graph. This set of contigs acts as a non-redundant representation of the k-mers from the original input sequences associated to that annotation label.

In the second stage of construction, all contigs obtained in the first stage are merged into a single joint de Bruijn graph (**Figure 1**, top right).

In the third construction stage, graph annotation, all contigs are mapped onto the joint graph to mark the relations of each k-mer to the annotation labels. Conceptually, each label forms a column of the annotation matrix represented as a compressed sparse binary vector (**Figure 1**, middle right). Finally, the annotation matrix is converted into a representation best suited for the target application (see Online Methods 4.6 for further details).

Fast and scalable queries on large indexes As sequence search is a key task for most biomedical analyses, we devised several efficient search algorithms to identify sequence matches in the graph and retrieve associated labels. In the first approach, input sequences are broken down into k-mers that are matched to nodes of the graph. The resulting set of paths then directly induce corresponding annotations (**Figure 2**, top left). For increased sensitivity, we also devised an algorithm for *sequence-to-graph alignment*, which identifies the closest matching path in the whole graph or in subgraphs induced by the annotation columns (**Figure 2**, bottom left; see Online Methods 4.9.2 for details on the method and Section 2.3 for results on accuracy). One important application of our search methods is *experiment discovery*, where each annotation label represents a sequencing library (SRA experiment) and the index is scanned to find all experiments with reads similar to the queried sequence.

If the query is a single sequence, both k-mer matching and alignment can be applied directly on the full annotated graph. For querying larger sets of raw reads or long sequences, we have designed an efficient batch query algorithm (schematically shown in **Figure 2**, right) that exploits the presence of k-mers shared between individual queries by forming an intermediate *query graph*, a small subgraph represented in an uncompressed format which is fast to query (see Online Methods 4.9.4 for further details). This additional step often leads to a 10 to 100 fold speedup, depending on the structure of the query data (**Supplemental Figure S-1**).

2.2 Constructing petabase-scale indexes as a community resource

Building on the powerful *MetaGraph* framework, we set out to process a significant part of all publicly available sequencing data. Consequently, we constructed indexes on real-world data sets

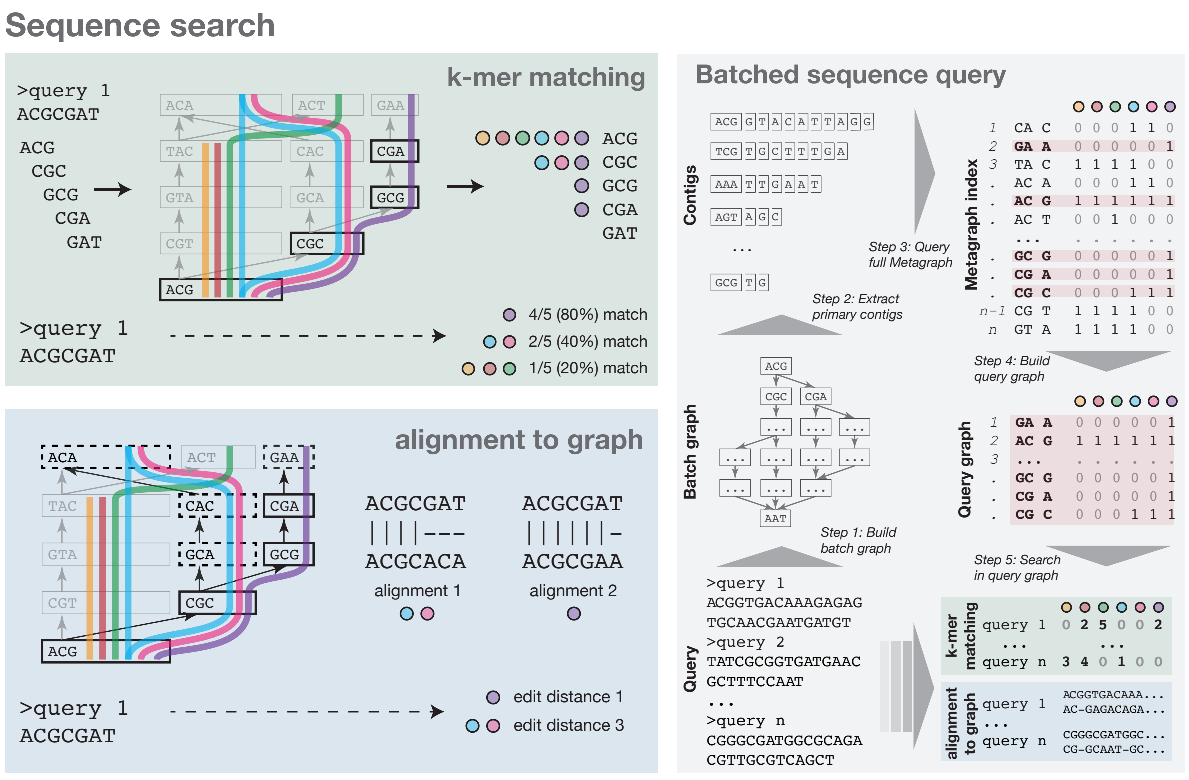


Figure 2: Graph querying approaches – **Left:** Schematic representation of the two main approaches for sequence search. **Top left:** Counting *exact k-mer matches* between query and graph. **Bottom left:** *Alignment* finds all closest paths within a given edit distance. **Right:** *Batched sequence search* retrieves a decompressed subgraph (query graph) from the full compressed annotated graph for subsequent query. All query sequences are combined into an intermediate batch graph that is then traversed to extract contigs to be queried against the full index. Hits and their corresponding annotations are aggregated to construct the final query graph, which is then searched against with the original query sequences.

of varying size and complexity, including both DNA and RNA sequencing samples. The resulting indexes form a valuable community resource, as they succinctly summarise large raw-sequence data sets, while supporting a variety of sequence queries against them. The key statistics of all data sets presented in this work and *MetaGraph* indexes constructed from them are listed in **Table 1** and visualized in **Figure 3a**. Applying only moderate cleaning on the input sequences (see Online Methods for exact parameters), a *MetaGraph* index typically requires orders-of-magnitude less storage than the original gzip-compressed inputs.

***MetaGraph* indexes a diverse range of input data** The range of input data was chosen to represent properties commonly occurring in biomedical research. On the one end stand large cohorts containing sequences sampled from a sequence pool of limited diversity. A representative of this class are the RNA-Seq experiments from the GTEx cohort [42] and the TCGA cohort [61] representing the human transcriptome. With compressed input sizes of 40 and 65 Tbp, respectively, the GTEx and TCGA cohorts can be indexed and further compressed into annotated graphs of only 132 GB and 63 GB, respectively. In the middle of the complexity spectrum reside whole genome sequencing experiments and collections of reference genomes, where similarity between samples is a function of evolutionary distance and the diversity within the data set is generally much higher than in the human transcriptome cohorts. Representatives of this class are RefSeq [48] and the UHGG gene genome catalog as well as the SRA-Microbe, SRA-Fungi, SRA-Plant, and SRA-Metazoa data sets. While RefSeq contains all sequences of assembled genomes available in version 97 of the RefSeq database, comprising 170 Gbp of input,

Table 1: Summary of constructed indexes for big data sets. (Numbers marked with $*$ represent estimates, as the computations are currently still ongoing. Entries labeled as "TBA" will be added as soon as they become available.) †: For the GTEx++ index, parts of the inputs are given as lists of mutations and reference sequences, which are negligible in the input size computation.

Data set	# bp	Input (gz)	k	# k-mers	Labels	Graph	Anno.	Ratio
GTEx [42]	$70 \cdot 10^{12}$	40 TB	41	$1.1 \cdot 10^{9}$	9,759	0.9 GB	55 GB	716x
GTEx++ [42, 36]	$70 \cdot 10^{12\dagger}$	40 TB†	41	$26 \cdot 10^{9}$	9,786	15 GB	117 GB	303x
TCGA [61]	$81 \cdot 10^{12}$	65 TB	31	$2.1 \cdot 10^{9}$	11,095	1,6 GB	63 GB	1000x
RefSeq (bact) [49]	$1.7 \cdot 10^{12}$	469 GB	31	$626 \cdot 10^{9}$	48,539	339 GB	410 GB	0.62x
UHGG catalog [4]	$11 \cdot 10^{9}$	3,3 GB	31	$9.6 \cdot 10^{9}$	4,644	5,3 GB	19 GB	0.13x
MetaSUB [19]	$7 \cdot 10^{12}$	5.5 TB	19	$71.7 \cdot 10^{9}$	4,220	49 GB	266 GB	17x
SRA MetaGut	$63 \cdot 10^{12}$	36 TB	31	$49.9 \cdot 10^{9}$	20,639	30 GB	514 GB	66x
SRA Microbe [14]	$221 \cdot 10^{12}$	170 TB	31	$39.5 \cdot 10^{9}$	446,506	30 GB	261 GB	584x
SRA Fungi	$163 \cdot 10^{12}$	81 TB	31	$277 \cdot 10^{9}$	121,907	82 GB	501 GB	139x
SRA Plants	$1.1 \cdot 10^{15}$	576 TB	31	$0.9 \cdot 10^{12}$	531,736	602 GB	TBA	TBA
SRA Metazoa	$1.5 \cdot 10^{15}$	925 TB	31	$1.5 \cdot 10^{12*}$	797,883	963 GB*	TBA	TBA

the UHGG genome catalog is a recently published catalog of human gut reference genomes [4]. The different SRA datasets comprise over 90% of currently available SRA samples for the respective groups. Only for the SRA-Microbe dataset, we rely on a previous definition: The data set contains a diverse range of samples consisting of a total of 446,506 virus, Prokaryote, and small Eukaryote genomes, and was first presented as part of the evaluation of BIGSI [14] (see Online Methods 4.13.2).

As RefSeq covers a much higher diversity of input sequences than, e.g., SRA-Microbe, its index size is correspondingly much larger. The fully searchable and annotated *MetaGraph* representation of the SRA-Microbe data set is only 291 GB (compared to 1.6 TB for the BIGSI index), while RefSeq has a total size of 1,040 GB (when annotated by species taxonomic IDs). Lastly, on the other end of the spectrum stand cohorts containing whole metagenome shotgun sequencing experiments. For this class we have selected the MetaSUB cohort [19], containing more than 4,200 environmental metagenome sequencing samples comprising 7.3 Tbp, and the SRA-MetaGut cohort, containing all human gut metagenome sequencing samples available on SRA (20,639 as of 2018-05-25), comprising approximately 60 Tbp. The input data in these cohorts are samples from very diverse populations of organisms and contain a large diversity of rare sequences. As a result, the index sizes are relatively large when compared to other data sets, for MetaSUB 315 GB and for SRA-MetaGut 544 GB.

Especially the large-input cohorts, such as the SRA-cohorts, required scaling into a commercial compute cloud for the first stage of *MetaGraph* assembly. This step is made particularly easy by the modular nature of the presented framework.

The SRA-Microbe, SRA-Fungi, SRA-Plants, SRA-Metazoa databases are built on 446,506, 531,736, 121,907, and 797,883 whole genome sequencing samples, respectively, available on SRA. All data sets listed in **Table 1** combined accumulate to a total of 3.2 petabases of input.

The complete lists of sample identifiers used in our experiments are provided in Supplemental Data Section 5.

***MetaGraph* is exceptionally scalable** The generation of indexes on petabases of input data is made possible by *MetaGraph*'s first-in-class scalability and its efficient implementation. To systematically evaluate our approach and to show why other existing methods would struggle solving tasks of a similar size, we assess both the index representation size and the query performance of *MetaGraph* and benchmark them against other state-of-the-art indexing schemes: Mantis [5], BIGSI [14], and COBS [12], using subsets of increasing size up to 25,000 samples drawn from the SRA-Microbe set.

While Mantis and *MetaGraph* provide *lossless* representations of the set of all input k-mers, BIGSI and COBS both employ probabilistic data structures that can lead to false-positive matches when the index is queried. Hence, we denote the latter two approaches as *lossy* compressors. We used BIGSI with the same parameters as in the original work [14] (3 hash functions with Bloom filters of size $25 \cdot 10^6$) and COBS with 4 hash functions and target false-positive rate equal to 0.05.

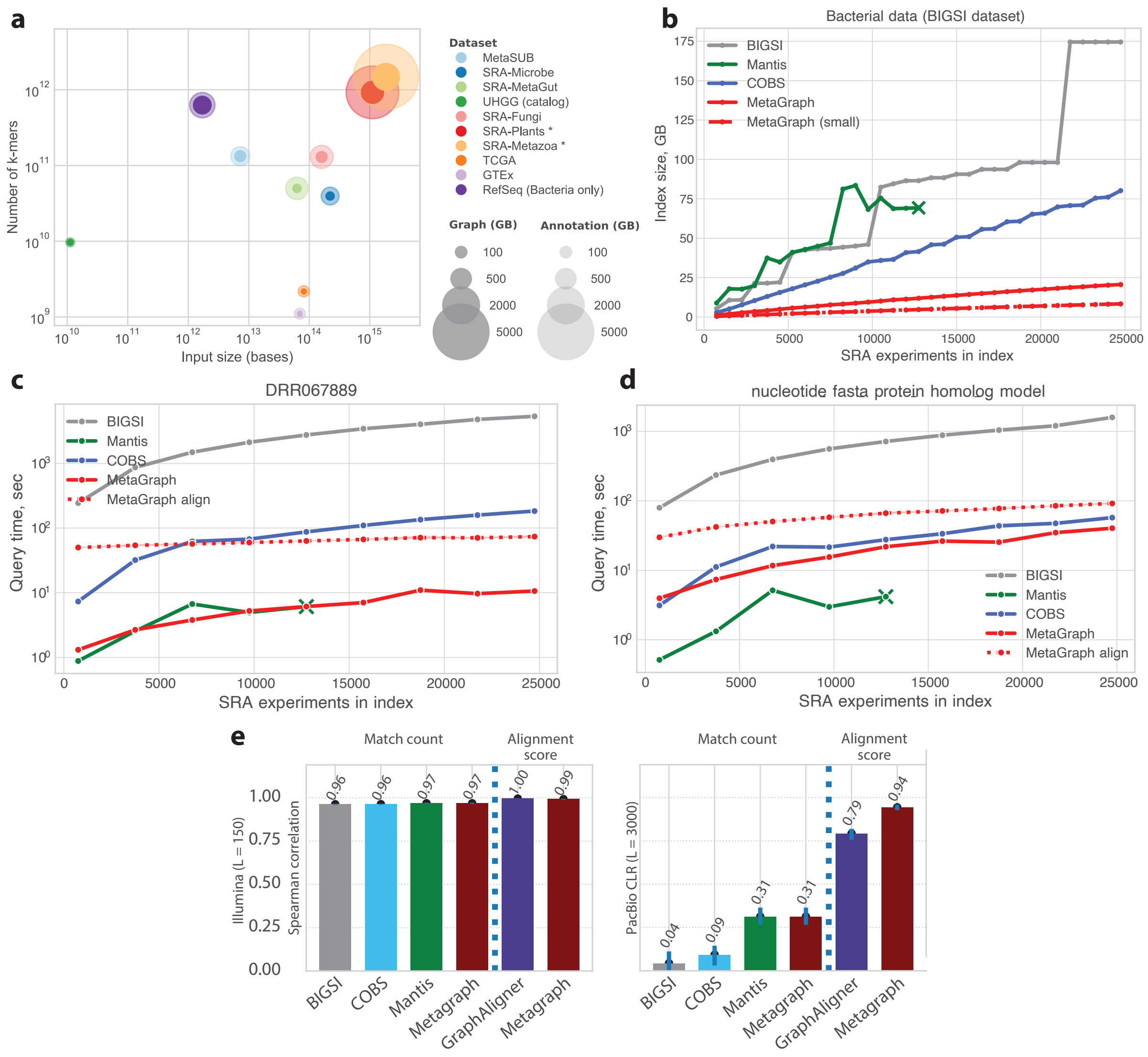


Figure 3: Scalability and performance – **a)** Overview of all *MetaGraph* indexes presented in this work, showing total number of input bases on the x-axis and index size in total number of unique k-mers on the y-axis. Marker size represents the size of the index. The solid portion of each marker represents the fraction of total size taken by the graph and the translucent portion represents the fraction taken by the annotation (**Table 1**). **b)** Size of evaluated index data structures for representing a set of microbial WGS experiments of increasing size, shown for both lossy indexing methods BIGSI and COBS and lossless Mantis and *MetaGraph*. Construction of the Mantis index for the 13,500 sample subset was only 43.6% complete after reaching our 240 h run time limit. **c)** Times for querying gut microbiome sequencing data. Color scheme as in a. **d)** Same as in b, querying AMR gene DNA sequences from the CARD database [1]. **e)** Accuracy of sequence search approaches for different sequencing technologies (left: Illumina, right: PacBio CLR). The graph is constructed from an *Escherichia coli* reference genome and the query reads are simulated from 12 different *E. coli* genomes. Accuracy is measured as the Spearman correlation between search scores (k-mer match count or alignment score) and ground truth sequence alignment scores. Error bars represent 95% CI's across 100 bootstrap samples from 3480 and 874 query reads of Illumina- and PacBio-type, respectively, corresponding to a coverage of 1×.

While all representations grow approximately linearly with the input size, the slope and growth behavior are drastically different (**Figure 3b**). For Mantis, the full index grows irregularly with an increasing number of experiments, but is always significantly larger than *MetaGraph* (see also **Supplemental Figure S-6** for further details). Notably, *MetaGraph* also uses significantly less memory than BIGSI and COBS, despite their use of a lossy compression approach.

Note that the memory footprint of an index is not only relevant for its construction, but also for any query, as the index needs to be loaded into memory for performing query operations on it. Thus, it becomes especially relevant for petabase scale indexes, where the total memory available on servers is limited and a constant factor can determine whether an index can be hosted.

To meet different efficiency needs when serving queries, *MetaGraph* provides a variety of alternative annotation representations with different trade-offs between size of the index and query performance. However, even when using the larger annotation representations optimized for maximal query speed, *MetaGraph* needs significantly less memory than its competitors (**Supplemental Figure S-6**).

2.3 *MetaGraph* shows superior search performance

In addition to scaling index construction to large input sizes, a second focus of *MetaGraph* lies on providing efficient sequence search. With the batched query strategy, *MetaGraph* demonstrates query times competitive to all other evaluated methods, and for some queried data, surpasses BIGSI and COBS in query speed by up to several orders of magnitude (**Figure 3c** and **d**).

De Bruijn graph-based methods typically perform sequence search via exact k-mer matching (e.g., [12, 14, 53]), which can be seen as an approximation of semi-global alignment of a query sequence against each label's corresponding subgraph. However, this poses a sensitivity-specificity trade-off, where reducing k increases sensitivity and reduces match specificity, and vice-versa.

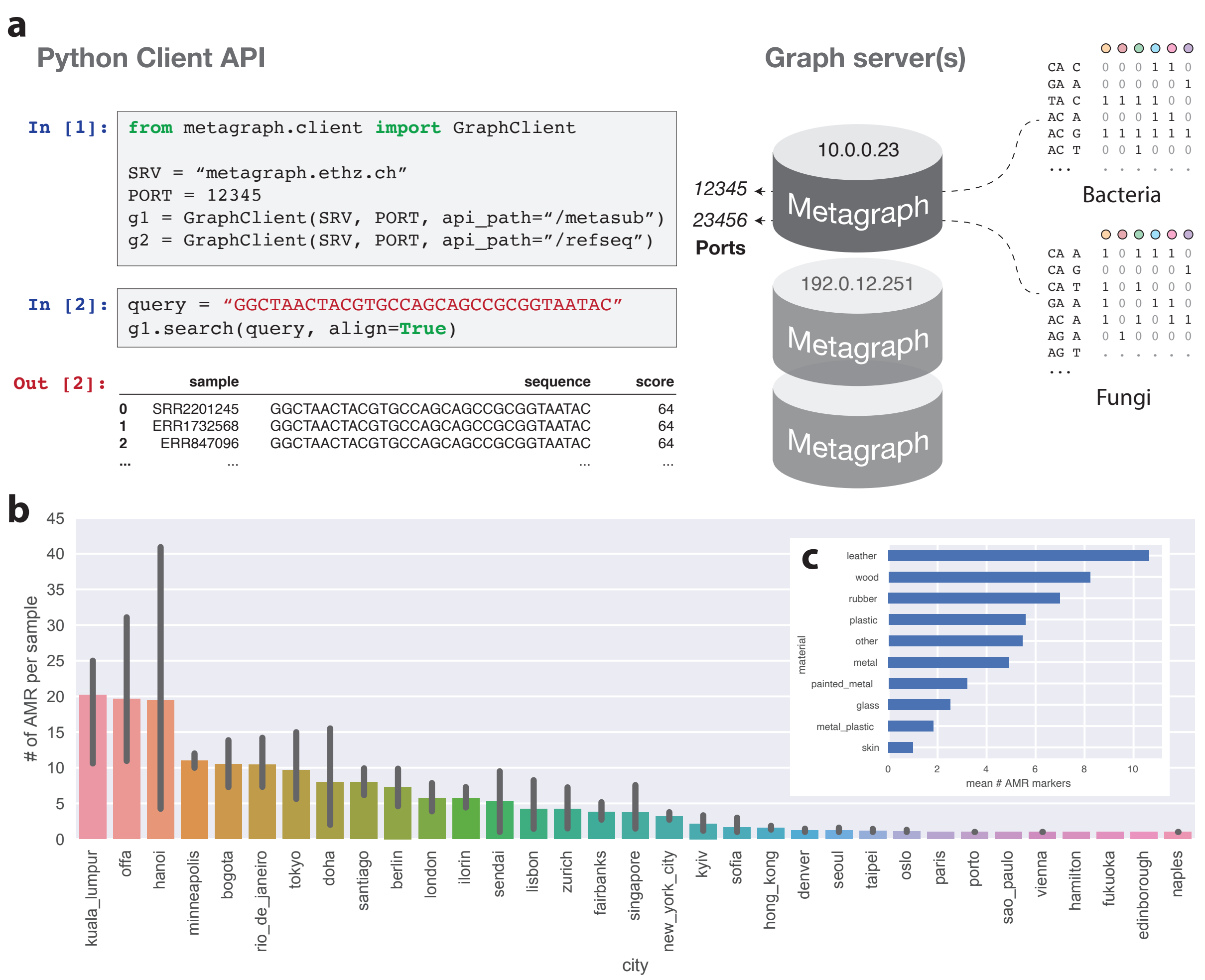
To address this in *MetaGraph*, sequence search can be performed not only via exact k-mer matching but also with alignment to graph. Notably, the sensitive alignment strategy in *MetaGraph* performs faster than some of the other exact k-mer matching methods (**Supplemental Figure S-1**).

***MetaGraph* alignment improves search accuracy** Alignment to an annotated de Bruijn graph can be approached either via alignment to the entire graph, or via alignment to any of its subgraphs, for instance, induced by any of the annotation labels. The latter case, as mentioned above, more closely mirrors the classical problem of aligning sequences against a reference genome.

Since searches of a query sequence against each label are independent, we evaluated the accuracy of querying against single-label, read set-derived genome graphs as a proxy for alignment against a reference genome (**Figure 3e** and **Supplemental Figure S-2**). For this, we simulated Illumina HiSeq read sets from an *Escherichia coli K-12, MG1655* reference genome (GenBank accession ID `NC_000913.3`) with ART [31] at varying coverage levels and indexed them with Mantis, BIGSI, COBS, and *MetaGraph* (see Online Methods 4.13.1). We then measured the accuracy of both exact k-mer matching and alignment queries (where applicable). In addition, we also measured the accuracy of the sequence-to-graph alignment tool GraphAligner [55] on a GFA representation of the *MetaGraph* index. For testing, we simulated Illumina HiSeq and PacBio read sets from 12 *E. coli* genomes and aligned them to the `NC_000913.3` reference genome with the Parasail aligner [18] to compute ground-truth alignment scores. We found that the accuracy of alignment and k-mer matching with *MetaGraph* outperforms all other tools in this experiment. Although exact k-mer matching did not perform as well for all methods, the alignment methods in *MetaGraph* substantially improve the accuracy of match scores for the simulated PacBio reads. In addition, we observed that alignments to graphs constructed from read sets of coverage 20× were as accurate as alignments to graphs constructed from the original reference sequence (**Supplemental Figure S-2**). These results show that if *MetaGraph* indexes are built on sufficiently deeply sequenced inputs, they can replace *bona fide* reference genome databases as backends for sequence search. We further discuss the generalizability of this conclusion in Section 3.

2.4 *MetaGraph* allows for distributed and interactive use

In addition to the single-machine use case, where the graph index is built and queried locally, *MetaGraph* also supports the distribution of indexes for query via a client-server architecture. Using this concept, a set of graphs and annotations can be easily spread across multiple machines. Each machine runs *MetaGraph* in *server mode*, offering one or multiple indexes, awaiting queries on a pre-defined port. This modularity makes it straightforward to integrate one or many queries across a whole set of graphs served on multiple servers. For easy integration of results and coordination of different *MetaGraph* instances, we provide interfaces to popular scripting languages, such as Python, allowing for the interactive usage of one or several (remote) *MetaGraph* index instances (**Figure 4a**).



```
In [1]: from metagraph.client import GraphClient

        SRV = “metagraph.ethz.ch”
        PORT = 12345
        g1 = GraphClient(SRV, PORT, api_path=“/metasub”)
        g2 = GraphClient(SRV, PORT, api_path=”/refseq”)

In [2]: query = “GGCTAACTACGTGCCAGCAGCCGCGGTAATAC”
        g1.search(query, align=True)

Out [2]:
         sample                          sequence   score
   0  SRR2201245  GGCTAACTACGTGCCAGCAGCCGCGGTAATAC      64
   1  ERR1732568  GGCTAACTACGTGCCAGCAGCCGCGGTAATAC      64
   2   ERR847096  GGCTAACTACGTGCCAGCAGCCGCGGTAATAC      64
   ...       ...                               ...     ...
```

Figure 4: Utility and usability – **a)** *MetaGraph* is designed to support a client-server infrastructure as exemplified here with a script in Python. In a few steps, several remote (or local) graph instances can be created and queried interactively. Results are returned as a data frame that can be used for further analyses. **b)** Number of antimicrobial resistance (AMR) markers per sample for different cities in the MetaSUB study. Bars represent $\pm\sigma$. **c)** Distribution of the mean number of AMR markers grouped by surface material based on all samples of the MetaSUB data set.

We demonstrate this usability in publicly available example scripts[1]. In this analysis, the full CARD AMR database [1] is queried against a *MetaGraph* index containing more than 4,400 whole metagenome sequencing samples from the MetaSUB cohort. We use the data to generate a ranking of cities based on the average number of AMR-markers in a sample (**Figure 4b**), largely consistent with the analysis performed on the raw data using orthogonal strategies [19]. Further, the script is used for exploratory analyses, linking sample metadata, such as surface material at the sampling location, to the query results (**Figure 4c**). We invite readers to run the script themselves, reproduce the plots interactively and further explore the available data.

2.5 Differential sequence assembly identifies pathogens in kidney transplant patients

In additional to sequence search, *MetaGraph* also supports integrative analysis on the samples in an annotated graph, assembling sequence markers satisfying user-provided criteria. For instance, given a set of whole metagenome sequencing samples of two patient populations that are distinguished by a certain phenotype (e.g., resistance to treatment), one can categorise the patient samples according to this phenotype and select the corresponding distinguishing k-mers (**Figure 5a**, top). These can then be assembled into *differential sequences* (**Figure 5a**, bottom), which act as markers for further study. We refer to this process as *differential assembly*.

Classical approaches to solve this problem would involve aligning the reads from each sample and selecting a subset to act as markers. This analysis, however, requires the raw sequencing data for each sample to be available. With *MetaGraph* indexes, this analysis can be performed directly on the compressed index.

Columns of the annotation matrix in a *MetaGraph* index *mask* nodes of the graph and induce subgraphs from which sequences can be assembled. Thus, one can construct a desired subgraph by performing a series of operations on the annotation columns to derive a corresponding node mask (see Online Methods 4.10 for more details).

To demonstrate the concept of differential assembly on a real-world application, we performed an analysis on whole metagenome sequencing samples collected from the urine of kidney transplant patients to test for the transfection of *JC polyomavirus* and other pathogens [56]. Using the pre-transplantation samples of recipients as the background sets and the intersection of the donor and post-transplantation samples of the recipients as foreground sets, we used the *MetaGraph* representation of all samples to assemble sequences present in the foreground but absent in the background for each pair. Given the assembly parameters, the differential assembly resulted in sequences identified as *JC polyomavirus* (genome length 5032 bp) having average lengths from 43 bp to 242 bp and NG50 values ranging from 0 (two samples) to 59. Consistent with the original publication [56], we detected *JC polyomavirus* in 6 out of 15 samples as an agent transmitted with transplantation. The detection was performed by querying the assembled sequences against the *MetaGraph* SRA-Microbe index and the virus portion of the RefSeq index (**Figure 5b**). Of the 16,913 differential sequences assembled from the patients (5502 of them from those with positive detections), 906 were identified as containing *JC polyomavirus* k-mers, among which 838 contained exclusively *JC polyomavirus* k-mers (see **Supplemental Figure S-4** and **Supplemental Table S-12**).

2.6 Uncovering unexpected transcriptome features in GTEx and TCGA

The Genotype Tissue Expression (GTEx) project has become a *de facto* reference set for human RNA-Seq expression and is widely used in the community [42]. While gene expression values and other summary statistics of the more than 10,000 samples are easily accessible, the whole set of raw sequence RNA-Seq files comprises more than 40 TB even in compressed state. Similarly, The Cancer Genome Atlas (TCGA) has collected more than 10,000 RNA-Seq samples from

[1]Available at `https://github.com/ratschlab/metagraph_paper_resources/tree/master/notebooks`

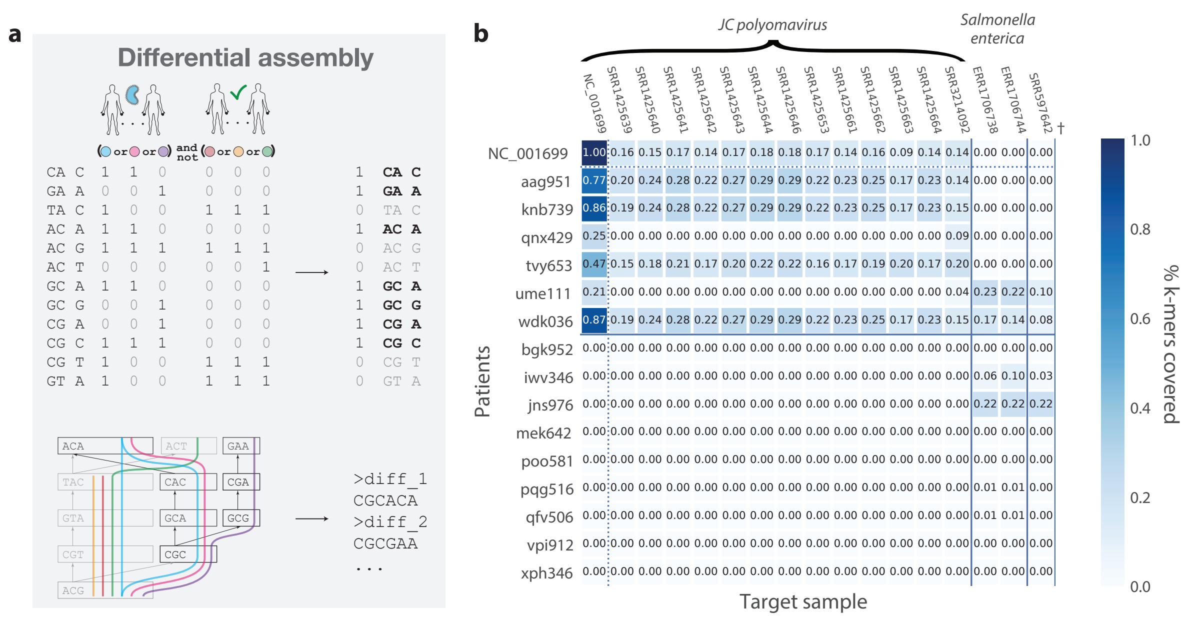


Figure 5: Differential graph assembly – a) Differential assembly schema. Logical operations applied on columns form a logical mask inducing a subgraph of the full graph. The example shows the subgraph implied by using nodes present in the first three columns, but not present in any of the last three columns. **b)** Assignment of sequences resulting from differential assembly of kidney transplant patients to target genomes and samples (see Online Methods 4.10). Rows represent patient metagenome samples, columns represent SRA samples in which the assembled sequences were discovered. Shading indicates fraction of target genome covered. The SRR IDs represent samples annotated as *JC polyomavirus*, *Salmonella enterica*, and uncultured bacteria (†) in the SRA-Microbe index, while `NC_001699` represents the *JC polyomavirus* reference genome.

primary tumors, spanning across more than 30 cancer types, constituting a central resource for cancer research [61]. This cohort amasses 65 TB in compressed sequences for RNA-Seq samples alone.

For either cohort, tasks that depend on access to the whole data set require an extraordinary effort. For instance, in order to search for the presence of previously unobserved sequence variants in the cohort (e.g., alternative splice forms or somatic variants) the entire data set would need to be re-analyzed.

We compressed 9,759 RNA-Seq samples from GTEx [42], a total of 70 Tbp, into an annotated *MetaGraph* index of only 56 GB in size. In addition, we constructed an extended transcriptome index containing, in addition to all variation detected in the GTEx raw data, the human reference genome sequence (version 38), the GENCODE reference transcriptome (version 32), and all variants from the gnomAD cohort (release 3.0) [36]. We refer to the latter index as *GTEx++*. Although the genome and gnomAD variation could also be represented using frameworks like VG [46], the integration of transcriptome and raw sequencing data is currently not straightforward. Adding annotations by sample and expression quantiles increased the index size to 114 GB and 230 GB, for GTEx and GTEX++ indexes, respectively, which is still an over 100-fold reduction in size to the original data. The compression of the TCGA cohort data shows a similar ratio, reducing 81 Tbp of input data into an annotated graph of just 65 GB.

Although *MetaGraph* compresses the GTEx input sequences into a de Bruijn graph of only 15 GB through removing redundancy and sanitizing the (noisy) input, the graph index still shows a remarkable sensitivity when queried with the original data, showing that indeed only a small fraction of likely noisy unitigs was removed during the initial cleaning phase of the graphs. To check how well *MetaGraph* preserves the sequence information present in the raw data and how this affects alignment sensitivity relative to classical linear alignment tools, we re-aligned raw reads from 10 randomly picked GTEx samples back to the GTEx++ index. When we compared the number of aligned reads between *MetaGraph* and a spliced alignment to a linear reference genome [20], we found on-par results in all cases and superior sensitivity in 8 out of 10 query samples (**Supplemental Figure S-3a**). When looking specifically into the fraction of reads unmappable by the linear aligner, we found that generally more than half of these reads can still be aligned by *MetaGraph* and find support in almost all GTEx samples (**Supplemental Figure S-3b**).

Detection of trans-splice junctions Exploiting the efficient query of the graph index, we investigated different transcriptome features. First, we assayed the use of exon combinations arising from trans-splicing. In this atypical form of splicing the donor of an exon is not connected to the acceptor of the subsequent exon but to the acceptor of the preceding exon (**Figure 6a**, schema at the top). Due to their non-contiguity, classical linear RNA-Seq aligners are unable to find such alignments, as we illustrate for genome alignments of a GTEx sample using a standard spliced aligner (**Figure 6a**, bottom). We created short sequences of length 81 bp spanning all theoretically possible trans-junctions in the GENCODE (version 30) annotation, using the GRCh38 reference genome. This resulted in a total of 3,457,560 candidate trans-junctions. All sequences not matching to the reference genome were aligned to the *MetaGraph* GTEx index, which resulted in 472 trans-junction candidates, perfectly matching against at least one path in a subgraph induced by a single sample. Interestingly, the occurrence of these junctions is not uniformly distributed across tissues (**Figure 6b**), nor does it correlate with the number of samples available per tissue (**Supplemental Figure S-7**). Even though the expression of trans-junctions is lower than that of the flanking regular junctions, we find sufficient read coverage to support these alignments, making an artefactual discovery unlikely (**Figure 6c**).

Expression evidence for somatic variants in TCGA We were interested to collect the RNA-Seq expression evidence for a large set of known somatic mutations in the TCGA cohort. Based on all single nucleotide variants of the COSMIC database (version 82) [59], we generated query sequences from the GRCh37 reference genome spanning the variant with additional 20 bp of sequence context both upstream and downstream. All sequences that did not map to the GRCh37 reference, were then aligned to the *MetaGraph* TCGA index. We matched the corresponding variants against the somatic variant calls of the MC3 project, performed on a large set of TCGA samples [22]. For all samples with both expression and MC3 evidence available, we asked which COSMIC variants were both detected in the whole exome sequencing (WXS) based variant calling and supported by RNA-Seq expression evidence. While over half of the positions were confirmed both by RNA-Seq and WXS, showing a Jaccard index of larger than 0.5 (**Figure 6d**), about 30% of the positions were mainly supported by RNA-seq, resulting in a Jaccard index of 0. Especially those positions that were solely found in RNA piqued our interest. One such position is COSM6336980, that expresses the variant allele exclusively in TCGA samples of the thyroid cancer sub-cohort (**Figure 6e**). Our first suspicion of a cancer specific variant was not confirmed, when we found that all of the normal samples in the Thyroid Cancer (THCA) sub-cohort expressed the variant allele as well (**Supplemental Figure S-8**). Interestingly, when confirming in the GTEx *MetaGraph* index, we found that also all GTEx thyroid tissue samples exclusively express the variant allele (**Figure 6f**), which led us to hypothesize tissue-specific RNA-editing as the possible source of this alteration. Interestingly, the observed variant is not a classical A-to-I editing, but instead represents a silent G-to-A alteration, which shows an astonishing pervasiveness across all thyroid tissue samples.

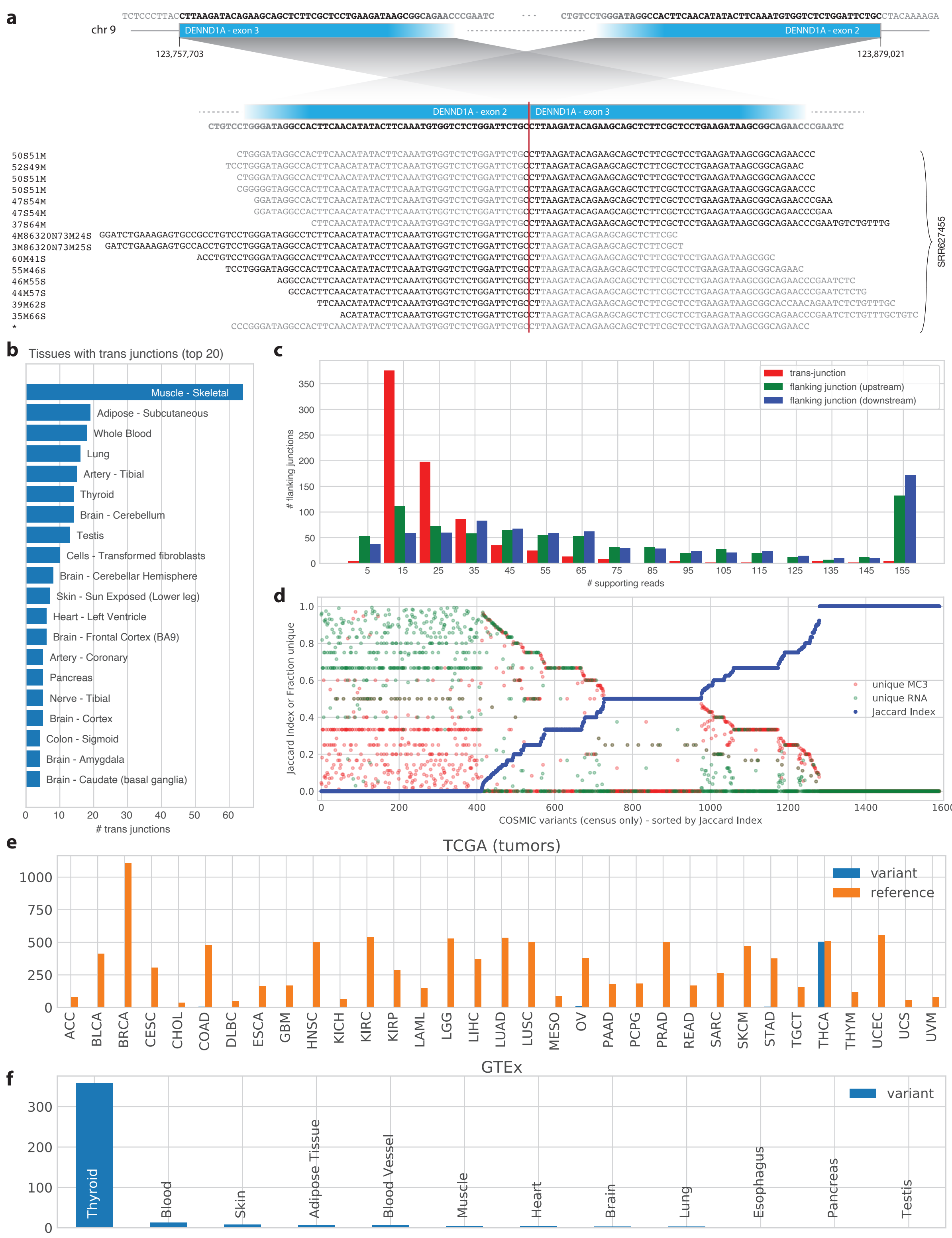


Figure 6: Analysis of transcriptome graphs – a) Schematic of the trans-splicing principle illustrated using the human gene DENND1A (chr 9) as an example. Top: Schematic representation of exon re-arrangement into a trans-junctions. Bottom: Spliced alignments to the linear reference genome for GTEx sample SRR627455. **b)** Top 20 GTEx tissues sorted by number of detected trans-junctions. **c)** Distribution of the number of supporting reads for all trans-junctions (red), compared to the distribution of read-support for the acceptor of the upstream junction (green) and the donor of the downstream junction (blue). **d)** COSMIC variants in cancer census genes found in the *MetaGraph* index. Each value on the x-axis corresponds to a single COSMIC variant with indices sorted by Jaccard index (blue), describing how well RNA and DNA samples agree on presence of the variant. Number of samples uniquely supporting RNA and DNA is shown in green and red, respectively. **e)** Expression distribution for COSMIC variant COSM6336980 across TCGA tumor samples, grouped by tumor type. Reference allele is shown in orange, variant allele in blue. **f)** Expression distribution for COSMIC variant COSM6336980 across GTEx samples, grouped by tissue.

3 Discussion

Recent advances in DNA sequencing technology have led to massive growth in the amount of high-throughput sequencing data available to the scientific community. However, a lack of standardized approaches for optimal representation and indexing of sequencing data at petabase scale severely limits the interactive exploration of this data and complicates large-scale genomic analysis efforts.

We have presented *MetaGraph*, a scalable framework designed for indexing and analysis of large collections of biological sequence data. We have shown that *MetaGraph* scales with the size of the input data exceptionally well and, in most cases, constructs index representations significantly smaller than those produced by current state-of-the-art methods. In addition, unlike BIGSI and other databases designed for approximate membership queries, *MetaGraph* constructs exact k-mer index representations that make no false-positive errors when querying.

We demonstrated the scalability of our approach by constructing queryable indexes for almost the entire collection of microbial, fungi, plant, and metazoa whole genome sequencing data sets present in NCBI's Sequence Read Archive, comprising a total of more than 3.2 petabases of input and representing more than 1.4 million individual samples. We know of no other available method that was applied to such large amounts of input data. Based on our scalability experiments, we expect that the framework can also keep up with future increases in data growth. However, when indexing the available data, we had to restrict ourselves to data generated on platforms with modest error rate. It is part of our future work, to adapt our raw data cleaning and error correction protocols to also be applicable to sequencing platforms with higher error rates, such as PacBio's SMRT or Oxford Nanopore Technologies long reads.

For the query task of experiment discovery, we have shown that the *MetaGraph* index achieves superior query performance to existing methods. In addition, its sequence-to-graph alignment method allows for increased search sensitivity, especially when querying more divergent sequences, while still maintaining competitive query times. While the current search methods return either a k-mer count or a disjoint sequence of alignments covering the query sequence, these results do not take into account different notions of node ordering on the graph. Recently, this has been addressed via indexing coordinates [60] or graph traversal distances [16] on genome graphs. With these additions, *MetaGraph* indexes have the potential to act as a faster replacement for database search methods such as BLAST.

We have further explored the utility of *MetaGraph* on a number of diverse computational biology applications. Based on a joint cohort of over 20,000 RNA-Seq samples from TCGA and GTEx, we used *MetaGraph* to explore the human transcriptome for interesting features. Thereby our focus was on properties that are hard to detect using linear reference genome alignments or require the integration of many samples to be seen. Specifically, we have evaluated the occurrence of trans-splicing and found over 200 cases commonly occurring within samples of the GTEx cohort. So far our analysis was restricted to trans-junctions occurring within the same gene but could be easily extended into a whole genome assay. A second focus of our transcriptome exploration was on the expression of somatic variants from the COSMIC catalog in different tissue and cancer types. We found that a large fraction of variants has evidence in RNA, but is not reliably called using methods based on whole exome sequencing. One mechanism we suggest contributes to this discrepancy is the occurrence of RNA editing. Thereby we put a special focus on tissue-specific RNA-editing, which seems to occupy a regulatory role. Further investigations in this direction are needed but are out of scope for this work.

Another innovative feature implemented in *MetaGraph* is a generalized framework for sequence assembly from subgraphs. This framework enables the user to fetch biological sequences specific to certain properties or groups of interest (e.g., individual samples, patients subgroups, or any set function of them). As a consequence, *MetaGraph* can answer such queries as "get all sequences found in samples x and y but not present in sample z", or "get all sequences shared by all organisms in this taxonomic group". This generalized view on assembly allows for new kinds of integrative analyses. We demonstrated the concept of differential assembly using whole metagenome sequencing of urine samples taken from kidney transplant patients. Using *MetaGraph*, we could reproduce findings on the data in a few minutes and could generalize the analysis. While our sequence assembly methods mirror those of other metagenomics assemblers by extracting unitigs [11, 47, 39] (non-branching stretches of the de Bruijn graph), many of these methods rely on iterative cleaning procedures to increase the lengths of the assembled contigs [11, 47]. As an alternative approach, the problem of chaining together unitigs to form longer contigs and scaffolds is a clear target for future work [10]. In particular, the use of the annotation matrix to motivate the path traversal strategy at branching points could potentially lead to the assembly of longer contigs without compromising assembly quality with heavy cleaning procedures.

A main goal of *MetaGraph* is to make large data sets accessible for interactive exploration. We facilitate this via providing a server-mode for the *MetaGraph*-backend that enables interactive queries from a client via the *MetaGraph* API. Several of the display panels in this work can be reproduced using the public *MetaGraph* instance and a Jupyter Notebook (Supplemental Data, Section 5). The interfaces can be easily extended to also support querying from other languages such as R or Julia.

We envision *MetaGraph* not only to provide a scalable framework for indexing highly diverse sequence databases but also to serve as a versatile tool that enables researchers to perform large-scale comparative analyses in genomics and medicine on typical academic compute clusters. It makes public data sets interactively accessible that are otherwise too big to handle or hard to retrieve. Along with the functionality currently provided, the scalability of many other pipelines could also be improved by translating string matching procedures into equivalent graph operations and supporting such operations using our framework.

4 Online Methods

4.1 Graph representations

MetaGraph uses k-mers (short contiguous sequences of length k) as elementary indexing tokens. Every two k-mers with overlap of at least k-1 characters are connected with a directed edge, which induces a node-centric *de Bruijn* graph, or more generally, an abstract sequence graph, with its nodes representing sequences and its edges representing overlap between them or adjacency in the input sequences.

MetaGraph employs several data structures for storing k-mer sets: i) a simple hash table, ii) a compressed indicator vector (e.g., for DNA sequences, a binary vector of size 4^k indicating which k-mers are present in the set) represented as a bitmap [17], iii) the succinct BOSS table [13] storing the set of k-mers succinctly. All of these data structures support exact membership queries, map k-mers to their positive indexes (or zero, if the queried k-mer does not belong to the set), and are used as a basis to implement different representations of the *de Bruijn* graph abstraction. We call these representations i) *HashDBG*, ii) *BitmapDBG*, and iii) *SuccinctDBG*, respectively. HashDBG and SuccinctDBG also feature dynamic versions of the representations that support online updates of the graph (e.g., insertion of a k-mer or a path in the graph).

In all experiments presented in this work, we used the static version of SuccinctDBG to represent full compressed graphs and the fast HashDBG to represent uncompressed query graphs (see Batched sequence search, **Figure 2**). The BOSS table used as a base data structure in SuccinctDBG often achieves the best compression of the k-mer dictionary and requires only about 6 bits per k-mer (in contrast, HashDBG requires at least ten times more than that for $k = 31$), which becomes a game-changer when indexing data at petabase scale. However, the lower memory footprint of the BOSS table comes at the cost of slower k-mer membership queries and mapping to retrieve their respective node indexes in the graph, requiring k internal traversal steps [13]. We alleviate this issue by augmenting the BOSS table with an auxiliary table mapping the k-mer suffixes (usually of length 10) to their respective ranges of rows in the BOSS table. With this optimization and the careful implementation in general, SuccinctDBG achieves performance sufficient to make the overall query times competitive to other methods while keeping the memory footprint at least an order of magnitude lower (see benchmarks, **Figure 3a-c**).

4.2 Graph construction

For graph construction, all k-mers are extracted from the given input sequences. For each k-mer, the number of times it occurred in the input is counted and the duplicate k-mers are discarded. *MetaGraph* uses the `SortedSet` approach to generate the so-called *k-mer spectrum* of the input: the input k-mers are appended to a list, which is sorted and de-duplicated every time it reaches the allocated space limit or after all input k-mers have been processed. During de-duplication, the k-mer counts are summed up to maintain the total count of each unique k-mer. In addition, *MetaGraph* offers the `SortedSetDisk` approach, which implements a similar algorithm in external memory. It imposes limits to memory usage and allows constructing arbitrarily large k-mer spectra, but requires a larger amount of disk I/O. Lastly, *MetaGraph* supports passing pre-computed outputs from the k-mer counter KMC3 [38] as input to make use of its extremely efficient counting algorithm and filters. Once the full spectrum is constructed and all k-mers are ordered, they are converted into the final data structure to construct the target graph representation.

4.3 Graph traversal and sequence extraction

All sequence information stored in the graph (or a defined subgraph) can be extracted and stored in FASTA format via graph traversal. Starting at all nodes with no incoming edges, the graph is fully traversed, and its paths formed by consecutive overlapping k-mers are converted into sequences (contigs) that are returned as a result of the operation. Each k-mer of the graph (or subgraph) appears in the assembled contigs exactly once. That is, the resulting set of sequences is a disjoint node cover of the traversed graph.

MetaGraph distinguishes two main types of traversal: i) traversal in *contig* mode extends a traversed path until no further outgoing edge is present or if all the next outgoing edges have already been traversed, while ii) traversal in *unitig* mode only extends a path if its last node has a single outgoing edge, and this edge is the single edge incoming to its target node. This is similar to the definition of unitig by [32]. For all traversal modes, we assemble sequences in parallel.

4.4 Graph cleaning and refinement

After a sample graph is constructed from raw data, paths likely originating from noisy inputs are pruned off. Then, the remaining sequences are extracted from the cleaned graph and output in FASTA format. To identify potentially erroneous k-mers, we use an algorithm originally developed by Iqbal and colleagues [32], adapted and scaled up in *MetaGraph* to work not only for small but also for large graphs (up to trillions of nodes). Briefly, the *k-mer spectrum* is used to calculate an abundance threshold for solid (non-noise) k-mers. Then, the graph is traversed and all unitigs with the median read support for k-mers below the pre-computed threshold are pruned off of the graph. In addition, all tips (unitigs where the last node has no outgoing edges) shorter than a given length are pruned off as well.

When indexing cohorts of large raw sequencing files, we optionally performed an additional step of pre-filtering based on their k-mer spectra. In this step, before building initial sample graphs, we discarded all k-mers occurring only once in the sequencing experiment, if the average k-mer count for that experiment exceeded 5 (see respective sections for each index in Online Methods and descriptions of experiments on SRA data sets 4.13).

4.5 Primary sequence graphs

When indexing sequencing reads from unknown strands, we supplement each read with its reverse complement, which is then indexed along with its originating read. As a result, the de Bruijn graph accumulates each k-mer in both orientations. Such graphs (which we call *canonical*) can be represented by storing only one orientation of each k-mer and simulating the full canonical graph on-the-fly (e.g., for querying outgoing edges, return not only the edges outgoing from the source k-mer, but also all edges incoming to its reverse complement).

Storing only *canonical k-mers* (the lexicographically smallest among the k-mer and its reverse complement) effectively reduces the size of the graph by up to two times. However, this cannot be efficiently used with the succinct graph representation based on the BOSS table. The BOSS table by design requires that each k-mer in it has other k-mers overlapping its prefix and suffix of length $k-1$ (at least one incoming and one outgoing edge in the de Bruijn graph). However, among any two consecutive k-mers in a read, only one is likely to be canonical. In other words, storing only canonical k-mers in the BOSS table would usually require adding at least two extra dummy k-mers for each canonical one, which makes this approach memory-inefficient. We overcome this issue by constructing so called *primary graphs*, where the word "primary" reflects the traversal order, as described below.

When traversing a canonical de Bruijn graph, we can additionally apply the constraint that only one of the orientations of a given k-mer is output. More precisely, the traversal algorithm works as usual, but never visits a k-mer if its reverse complement had already been visited. Whichever orientation of the forward or reverse complement k-mer is visited first, is considered to be the primary k-mer of the pair. This results in a set of sequences, which we call *primary* (*primary contigs* or *primary unitigs*). Note that the traversal order of the graph may change the set of primary sequences extracted from it, but it may never change the total number of k-mers in these sequences (*primary k-mers*). This is relevant when extracting primary contigs in multiple threads, since the node traversal order may differ between runs. We call graphs constructed from primary sequences *primary graphs*. Unlike the common approach where only canonical k-mers are stored, primary de Bruijn graphs can be efficiently represented succinctly using the BOSS table, and effectively allow us to reduce the size of the graph part of the *MetaGraph* index by up to two times.

4.6 Graph annotation and its construction

Independent of the choice of graph representation, a variety of methods are provided in *MetaGraph* for compressing the annotation matrix to accommodate for different query types. For fast sequence search queries, we provide implementations of the matrix compression techniques from VARI [45] (*RowFlat*) and *Rainbowfish* [6]. For query graphs (see Batched sequence search, **Figure 2**), we developed a compressed row-major sparse matrix representation *RowCompressed* which additionally supports dynamic operations. For differential assembly and high compression performance, we employ the Multiary Binary Relation Wavelet Tree (*Multi-BRWT*) compression technique [35]. Finally, a hybrid dynamic matrix compressor *ColumnCompressed* is provided for memory-efficient construction of the annotation matrix.

The typical workflow for constructing an annotation matrix for a large input set consists of the following steps. After the joint sequence graph for the input set has been constructed, we iterate over the input samples in parallel and map all k-mers of each sample to the graph, generating a single annotation column. To avoid the mapping of identical k-mers multiple times and prevent searching for noise k-mers, we typically use the unitig FASTA files of the cleaned sample graphs instead of the raw sequences when annotating a graph, which greatly reduces the annotation construction time. Once all columns have been constructed, *MetaGraph* performs an agglomerative hierarchical clustering on them to generate a *guide tree*. This guide tree is then used to construct a joint Multi-BRWT compressed representation of the matrix of all annotation columns. Finally, the Multi-BRWT scheme is relaxed to apply local optimizations of its structure improving the compression performance [35].

4.7 Index distribution scheme

Large data sets can be partitioned by input samples into multiple parts and indexed separately. These indexes can then be independently hosted on different servers. For greater flexibility, the graph representation can also be separated from the annotation representation. This distribution approach enables virtually unlimited scalability. For search, each query is sent to each part of the full index, followed by aggregation of the partial results to generate a final result of the query returned back to the user.

4.8 Dynamic index augmentation and batch updates

We support three strategies for extending an existing *MetaGraph* index. First, the new batch can be indexed in a separate annotated de Bruijn graph hosted on the same or a different server. Then, both can be queried simultaneously as it is done for a distributed *MetaGraph* index. Second, a graph can be directly updated if it is represented using dynamic data structures (e.g., SuccinctDBG in the dynamic form), which support dynamic updates. This approach allows making instant changes, however, does not enable large updates because of the limited performance of dynamic data structures [2]. Finally, for large updates, the existing index can be completely reconstructed. For reconstruction, the index is first decomposed into contig buckets, where each bucket stores contigs extracted from the subgraph induced by its respective annotation column. Then these buckets are extended with the new data and a new index is constructed from them. Notably, this approach uses the non-redundant set of contigs and does not require processing raw data from scratch again. Thus, the construction is performed on data up to 100 times smaller than the original raw reads.

4.9 Sequence search

The goal of sequence search is, given a query sequence, to retrieve the associated annotation labels from the *MetaGraph* index. We distinguish two types of sequence search: i) exact k-mer matching, intersecting the query k-mers with the set of k-mers of the index and ii) sequence-to-graph alignment, finding the set of paths in the graph that spell a sequence within a given edit distance to the query. Both approaches will be described in further detail in the following subsections.

4.9.1 Exact k-mer matching

The simplest approach for sequence search is exact k-mer matching. Each query sequence is decomposed into k-mers that are mapped onto the k-mer dictionary and their respective rows of the annotation matrix are then decoded to answer the query.

4.9.2 Sequence-to-graph alignment

To achieve greater sequence search sensitivity in cases where exact k-mer matching is not sufficient, we developed a sequence-to-graph alignment algorithm for the *MetaGraph* index. The algorithm takes a classic seed-and-extend approach [7], using several heuristics in both stages to improve query time.

Seeding Given an input sequence, the first step is finding exact k-mer matches in the graph. From these, maximal exact matches (MEMs, i.e., contiguous stretches of matching k-mers) are used as seeds. However, since the strategy of greedily taking exact nucleotide matches at graph bifurcation nodes may lead to the extension algorithm starting from a suboptimal node, we impose an additional constraint that MEMs must be contained within a single unitig (called Uni-MEMs [41]). By default, this seeding algorithm restricts seeds to be at least of length k. If the k-mer dictionary supports matching sequences of length less than k, this restriction may be relaxed to allow for shorter seed lengths. For instance, the BOSS table in SuccinctDBG forms a self index that can be searched by iterating through a given pattern, resulting in a range of row indices pointing to k-mers prefixed by the query [13]. If multiple nodes in the graph match a prefix of length $k' < k$, then all of them are considered as seeds.

Iterative seed extension Each seed is extended forwards and backwards in the graph to produce a local alignment. The extension algorithm is a generalization of the Smith-Waterman-Gotoh pairwise local alignment algorithm [26] to de Bruijn graphs. It extends the bit-parallel sequence-to-graph alignment algorithms introduced in [54, 55] to support affine gap penalties and introduces heuristics which improve alignment accuracy in the context of long error-prone reads, or graphs derived from low-coverage samples (see Section 2.3).

We now describe the extension algorithm in more detail. Given a seed, let $s = s_1 \cdots s_\ell$ denote the suffix of the query sequence beyond the end of the seed. In the subsequent extension algorithm, we use a dynamic programming table to represent the scores of the best partial alignments. More precisely, each node v has three corresponding integer score vectors S_v, E_v, and F_v of size ℓ. $S_v[i]$ stores the best alignment score of the prefix $s_1 \cdots s_i$ ending at node v. $E_v[i]$ and $F_v[i]$ represent the best alignment scores of $s_1 \cdots s_i$ ending with an insertion and deletion at node v, respectively.

Since the graph may be cyclic, multiple passes are made over the elements of the dynamic programming table until their values converge [54]. For this, we maintain a priority queue storing graph nodes whose corresponding score vectors have not yet converged and iteratively update them until the queue has been exhausted. Each node is prioritized by the greatest score among all elements updated in the last iteration [54]. To restrict the alignment search space, we employ the X-drop criterion [8, 62], skipping an element if it is more than X units lower than the current best computed alignment score. In addition, we apply a restriction on the total number of nodes which can be explored as a constant factor of ℓ.

Chaining local alignments The algorithm described above results in a set of local alignments of the query sequence. From this, a set of non-overlapping local alignments is constructed via a weighted job scheduling algorithm (see Algorithm 1). The local alignments are first sorted in non-decreasing order by their positions in the query sequence. Then, *chains* of local alignments are constructed iteratively. When two alignments cover overlapping regions in the query sequence, we trim the alignments to remove the overlapping region and maximize the sum of their scores.

Algorithm 1 Chaining local alignments

Precondition: $q = q_1 \cdots q_n$ is a query sequence, $A_1, \ldots, A_m$ are local alignments of q to the graph sorted by their ending coordinates in q.

Postcondition: $\mathbf{A}$, a sequence of disjoint alignments of q to the graph.

```
function ChainLocalAlignments($[A_1, \ldots, A_m]$)
    $\mathbf{S} \leftarrow [-\infty, \ldots, -\infty]$              ▷ Initialize the vector of $n$ scores
    $\mathbf{A}^* \leftarrow []$                                  ▷ Initialize an empty alignment chain
    for $j \leftarrow 0$ to $m - 1$ do
        $\mathbf{A} \leftarrow$ Chain($[A_j], j + 1$)          ▷ Find the best alignment chain starting with $A_j$
        if Score($\mathbf{A}$) > Score($\mathbf{A}^*$) then
            $\mathbf{A}^* \leftarrow \mathbf{A}$
        end if
    end for
    return $\mathbf{A}^*$
end function

function Chain($\mathbf{A}$, $i$)
    if $i = m$ then
        return $\mathbf{A}$
    end if
    if $\mathbf{S}$[EndPos(Last($\mathbf{A}$))] ≥ Score($\mathbf{A}$) then
        return []          ▷ return nothing if this chain scores lower than previous chains
    end if
    $\mathbf{S}$[EndPos(Last($\mathbf{A}$))] ← Score($\mathbf{A}$)      ▷ store best score at the current end point
    $\mathbf{A}^* \leftarrow []$
    for $j \leftarrow i$ to $m - 1$ do                                ▷ append an alignment
        if BeginPos($A_j$) ≥ EndPos(Last($\mathbf{A}$)) then          ▷ check for overlap with $A_j$
            $\mathbf{A}' \leftarrow$ Chain(Append($\mathbf{A}, A_j$), $j + 1$)
        else
            $\mathbf{A}' \leftarrow$ DisjoinAppend($\mathbf{A}, A_j$)       ▷ Trim last alignment and append
            $\mathbf{A}' \leftarrow$ Chain($\mathbf{A}'$, $j + 1$)
        end if
        if Score($\mathbf{A}'$) > Score($\mathbf{A}^*$) then
            $\mathbf{A}^* \leftarrow \mathbf{A}'$
        end if
    end for
    return $\mathbf{A}^*$
end function
```

4.9.3 Seed-and-extend on primary graphs

While primary graphs act as an efficient representation of canonical de Bruijn graphs, special considerations need to be made when aligning to these graphs to ensure that all paths which are present in the corresponding canonical graph are still reachable. For this, we introduce a further extension of the alignment algorithm to allow for alignment to an implicit canonical graph while only keeping a primary graph in memory. During seed extension, the children of a given node are determined simply by finding the children of that node in the primary graph, along with the parents of its reverse complement node. Finding exact matching seeds of length $\geq k$ can be achieved in a similar fashion, searching for both the forward and reverse complement of each k-mer in the primary graph.

To find seeds of length $< k$, matching to the suffixes of nodes in the canonical graph, a three step approach is taken. First, seeds corresponding to the forward orientation of the query are found according to the algorithm described in 4.9.2. The next two steps then retrieve suffix matches which are represented in their reverse complement form in the graph. In the second step, the reverse complements of the query k-mers are searched to find node ranges corresponding to suffix matches of length k'. Finally, these ranges are traversed forwards $k - k'$ steps in the graph to make the prefixes of these nodes correspond to the sequence matched. The reverse complements of these nodes are then returned as the remaining suffix matches.

4.9.4 Batched sequence search and alignment

To increase sequence search performance, *MetaGraph* processes query sequences in batches and queries each batch against a small uncompressed *query graph* extracted from the full compressed index (see **Figure 2**, right). Given a batch of query sequences, they are transformed into a transient *batch graph*, which is then traversed to extract a non-redundant set of contigs. These contigs are then queried against the full *MetaGraph* index via exact k-mer match to select the respective subset of annotations. The resulting small annotated graph is called *query graph* and represents the intersection of the batch graph with the full *MetaGraph* index. All inputs are then searched against the query graph.

For alignment queries, the query graph is augmented with additional neighbouring contigs from the full index to include the alignment paths that would be present if the query sequences were aligned to the full graph. We refer to this extension as the *hull* of the initial query graph. The size of the hull is set such that all k-mers which would potentially be explored by alignment to the full index are added to the query graph.

4.10 Sequence extraction from subgraphs

Another type of query on a *MetaGraph* index consists in the extraction of sequences from its given subgraph (see **Figure 5** bottom middle – Sequence extraction). Depending on how such a subgraph is defined, this may be used for filtering low coverage unitigs, constructing the core genome of a cohort, or assembling differential sequences. Extraction of sequences from subgraphs defined by masks is done by marking all nodes excluded from the subgraph as visited before starting traversal. Thus, only nodes in the subgraph are visited during traversal and assembled into target sequences.

In particular, constructing a node mask that can be used for differential assembly or core genome assembly proceeds in three stages. In the first stage, the annotation columns for each label of interest are counted in a bit-packed integer count vector. Then, all nodes corresponding to non-zero counts are included in the target subgraph. In the second stage, unitigs from this subgraph are evaluated according to user-defined inclusion criteria based on the values in the count vector. Those not passing these criteria are discarded from the subgraph. Finally, short tips are removed and the remaining subgraph is assembled into the resulting unitigs.

4.11 API

For interactively querying large graph indexes, *MetaGraph* offers a Python API that enables programmatic access to a single or multiple *MetaGraph* servers. Started in server mode, the *MetaGraph* index will be persistently present in memory of the server machine and accept queries on a pre-defined port. The API then allows sending search or alignment queries to the index and returns the result as a Pandas data frame for further downstream analysis.

4.11.1 *MetaGraph* online search engine

We make a subset of the indexes constructed and presented in this work available for online search queries at `https://metagraph.ethz.ch`. The list of indexes currently hosted includes SRA-Fungi, SRA-Microbe, MetaSUB and will further grow in the future.

4.12 Data sets

We have generated a wide range of different *MetaGraph* indexes on very diverse input data sets. In this section we describe the sources and compositions of those data.

4.12.1 Sequence Read Archive (SRA)

Most of the publicly available sequencing data generated in the past decades is collected in the databases of NCBI's Sequence Read Archive (SRA) [37]. Either using data set definitions of prior work [14] or defining own sub-sets using the metadata provided through the database, separating the data into different groups of related samples. For each such sub-set we have then constructed a separate *MetaGraph*. In the following, we will provide details for each group separately.

SRA - Microbe This data set was first used in [14] and presented as a data basis for the BIGSI index. Consisting of 446,506 microbial genome sequences, this set originally posted the largest indexed genome dataset. Although representing only a snapshot of ENA at that time, we decided to keep the same sequence set for this work, to enable direct comparison and benchmarking. A full list of SRA ids contained in this set is available as file `TableS1_SRA_Microbe.tsv` (with further information available in `TableS10_SRA_Microbe_McCortex_logs.tsv.gz` and `TableS11_SRA_Microbe_no_logs.tsv`) in the Supplemental Resources described in Section 5. For details on how the set of genomes was selected, we refer to the original publication of [14].

SRA - Fungi This data set contains all whole genome sequencing samples from NCBI SRA assigned to the taxonomic ID 4751 (fungi) specifying the library source GENOMIC and excluding samples using platforms PACBIO_SMRT or OXFORD_NANOPORE. In total this amounts to 125,585 samples processed for cleaning. Out of these 114,839 (91.44%) could be successfully cleaned and were used to assemble the final *MetaGraph* index. All sample metadata was requested from SRA using the BigQuery tool on the Google Cloud Platform. The complete list of all samples (including information which ones were successfully cleaned) is available as file `TableS2_SRA_Fungi.tsv` in the Supplemental Resources described in Section 5.

SRA - Plants This data set contains all whole genome sequencing samples from NCBI SRA assigned to the taxonomic ID 4751 (fungi) specifying the library source GENOMIC and excluding samples using platforms PACBIO_SMRT or OXFORD_NANOPORE. In total this amounts to 576,226 samples processed for cleaning. Out of these 531,736 (92.28%) could be successfully cleaned and were used to assemble the final *MetaGraph* index. All sample metadata was requested from SRA using the BigQuery tool on the Google Cloud Platform. The complete list of all samples (including information which ones were successfully cleaned) is available as file `TableS3_SRA_Plant.tsv` in the Supplemental Resources described in Section 5.

SRA - Metazoa This data set contains all whole genome sequencing samples from NCBI SRA assigned to the taxonomic ID 33208 (metazoa) specifying the library source GENOMIC and excluding samples using platforms PACBIO_SMRT or OXFORD_NANOPORE. In total this amounts to 899,045 samples processed for cleaning. Out of these 797'883 (88.75%) could be successfully cleaned and were used to assemble the final *MetaGraph* index. All sample metadata was requested from SRA using the BigQuery tool on the Google Cloud Platform. The complete list of all samples (including information which ones were successfully cleaned) is available as file `TableS4_SRA_Metazoa.tsv` in the Supplemental Resources described in Section 5.

SRA - Human Gut Microbiome (MetaGut) This data set contains all sequencing samples of the assay type `WGS` and `AMPLICON` from NCBI SRA that were annotated with the `organism` label "human gut metagenome", excluding the subset of experiments using platforms PACBIO_SMRT and OXFORD_NANOPORE. In total this amounts to 242,619 samples, where 177,759 (73.3%) were AMPLICON and 64,860 (26.7%) were WGS samples. Using a more lenient cleaning strategy, no automatic coverage threshold was detected for these samples and all processed records were included into the final *MetaGraph* index. All sample metadata was requested from SRA using the BigQuery tool on the Google Cloud Platform. The complete list of all samples processed for this index is available as file `TableS5_MetaGut.tsv` n the Supplemental Resources described in Section 5.

4.12.2 MetaSUB

This data set contains 4,220 whole metagenome sequencing samples collected from the built environment through the MetaSUB consortium [19]. The raw data can be downloaded using the MetaSUB utils[2]. A list of all sample IDs used in this study is available as file `TableS6_MetaSUB.csv.gz` in the Supplemental Resources described in Section 5.

4.12.3 GTEx

The GTEx data set contains 9,759 raw RNA-seq files from the GTEx project [42] (version 7). The data was downloaded from SRA via dbGaP and processed on the Leonhard Med compute cluster of ETH Zurich. A list of all sample IDs used together with accompanying metadata is available as file `TableS7_GTEX.txt` in the Supplemental Resources described in Section 5.

4.12.4 TCGA

The TCGA database has been built using RNA-Seq samples collected by The Cancer Genome Atlas (TCGA) project [61]. In total, the index contains 11,095 individual records spanning over all available TCGA cancer type. The data was downloaded from the Genomic Data Commons Portal of the NCI. A list of all sample IDs used in this study is available as file `TableS8_TCGA.tsv.gz` in the Supplemental Resources described in Section 5.

4.12.5 RefSeq

This data set contains all assembled reference genome sequences present in the version 97 release of the NCBI Refseq database. In total the sequences for 103,302 taxonomic IDs are integrated into the *MetaGraph* index, each taxonomic ID forming a label in the annotation. The full list of taxonomic IDs included in the index is available as file `TableS13_RefSeq97_taxIds.tsv` in the Supplemental Resources described in Section 5.

4.13 Experiments

This section summarizes the experimental setup for the different results presented in this work.

4.13.1 Measuring the accuracy of sequence search

Given the reference genome for *Escherichia coli K-12 MG1655* (RefSeq accession `NC_000913.3` [48]), we simulated Illumina HiSeq-type reads with ART [31] with coverage $C \in \{1, 10, 20\}$ and constructed cleaned *MetaGraph* indexes with $k = 31$ from the read sets. For cleaning, we trimmed short tips of length less than $2k$ and discarded low-coverage unitigs, using unitig abundance fallback values of 5 and 2 for input read sets of coverage > 5 and ≤ 5, respectively when an appropriate cutoff could not be determined. Then, we constructed *MetaGraph*, BIGSI, and COBS

[2]`https://github.com/MetaSUB/metasub_utils`

indexes from these cleaned unitigs. To generate an index for GraphAligner, we exported the *MetaGraph* index in GFA format. For evaluation, we simulated both Illumina HiSeq and PacBio CLR-type reads (the latter using PBSIM [52]) from 12 *E. coli* reference genomes (see Supplementary Table S9) and computed their optimal semi-global alignment scores to the `NC_000913.3` reference using the Parasail aligner [18].

4.13.2 Construction of the SRA-Microbe index

The SRA-Microbe graph was constructed from the same samples used to construct the BIGSI index [14]. A merged canonical graph with $k = 31$ was constructed from cleaned contigs obtained from the European Bioinformatics Institute FTP file server. The graph was then serialized into primary contigs, then reconstructed and annotated by sample ID to form the final graph.

4.13.3 Construction of the SRA-Fungi, SRA-Plants, and SRA-Metazoa indexes

Briefly, each sample was either transferred and decompressed from NCBI's cloud mirror or downloaded from ENA (if not available on SRA) onto a cloud compute server and subjected to k-mer counting using KMC3 [38] to generate the k-mer spectrum. If the median k-mer count on the spectrum was less than 2, the sample was further processed without cleaning. Otherwise, the sample was subjected to cleaning, using *MetaGraph*'s `clean` mode, pruning tips shorter than $2k$, and using an automatically detected coverage threshold for unitig removal, with a fallback value of 3. After cleaning, for each sample a canonical graph with $k = 31$ was created and serialized into primary contigs. For each cohort (Fungi, Plants, and Metazoa) the serialized samples were joined into a merged graph representation. Further details are available in the Supplemental Methods.

The general procedure to construct the *MetaGraph* index was as follows. A sample was downloaded from SRA and then subjected to k-mer counting using KMC3 [38]. Based on the count-spectrum, we used *MetaGraph*'s cleaning procedure, with a fallback value of 2. If the median k-mer count in the spectrum was less than 2, we did not attempt cleaning. All samples that were successfully cleaned or were not subjected to cleaning due to low coverage, were then used for *MetaGraph* index construction in `canonical` mode, using the build strategy based on external disk memory. Subsequently, we extracted primary contigs from the graph and rebuild a primary graph from the extracted contigs.

After graph construction, we used the cleaned input sequences to annotate the joint graph. Each input sample thereby formed an individual annotation column. All annotation columns were then transformed in to the Multi-BRWT representation for improved compression and higher query performance.

4.13.4 Construction of the SRA-MetaGut index

The construction of this index mainly follows the same procedure as previously described for the SRA-Fungi, SRA-Plants, and SRA-Metazoa indexes. The only difference is in input sample cleaning. As the automatic threshold estimation is not particularly well suited for metagenomics data, we have switched off the singleton filtering on the k-mer spectrum and used a constant cleaning threshold of 2 during graph cleaning. All remaining steps of construction and annotation remained unaltered.

4.13.5 Construction of the MetaSUB graph

All input samples were directly assembled into canonical de Bruijn graphs using *MetaGraph* with $k = 41$. All graphs were then cleaned using the *MetaGraph* cleaning mode, pruning tips shorter than $2k$ and removing unitigs depending on coverage (automatically detected based on k-mer spectrum). If no threshold could be detected, we used 3 as fallback value. The cleaned graphs were then transformed into primary contigs and stored on disk. Based on the primary contigs a joint graph was built, using external memory.

The joint primary graph was then annotated with each individual sample, where one sample was transformed into a single annotation column. The columns were then aggregated into a joint Multi-BRWT index, using the `transform_anno` mode of *MetaGraph*.

4.13.6 Construction and differential assembly of kidney transplant graphs

Graphs were constructed from each raw sequencing sample with $k = 31$. Each graph was then cleaned by trimming short tips of length $< 2k$ and pruning low coverage unitigs. A fallback of 2 was used when an appropriate coverage cutoff could not be determined. Joint annotated graphs were then constructed for all samples of each donor-recipient pair. Given a joint graph, the post-transplant recipient sample and the donor sample were considered to be the foreground set, while the pre-transplant sample of the recipient was considered to be the background. To define the differential subgraph, each graph unitig was kept if at least 80% of its constituent k-mers were present in both foreground patients and at most 20% of its constituent k-mers were present in the background set.

4.13.7 Construction and query of GTEx indexes

All available RNA-Seq samples that were part of the version 7 release of GTEx [42] were downloaded via dbGaP. A list of all samples used is available in the Supplemental Data Section 5. Each sample was individually transformed into a graph using $k = 41$ and then cleaned using *MetaGraph*'s `clean` module, trimming tips shorter than $2k$ and using an automatically detected coverage threshold with fallback 3 for removing noisy unitigs, and then serialised to disk into fasta format. All resulting fasta file were then assembled into a joint graph and then serialized to disk again into primary contigs. From these primary contigs a final graph was assembled. The primary merged graph was then annotated using the cleaned fasta file of each sample, generating one label per sample. All individual annotation columns were then collected into a joint matrix, that was transformed into relaxed Multi-BRWT representation.

Re-alignment experiments against GTEx index We randomly selected a subset of 10 samples from the GTEx cohort (available as file `TableS14_GTExSubset.tsv` in supplemental resources given in Section 5) to evaluate the re-alignment of samples against the graph. For each sample, we considered the first 250,000 read pairs. The reads were re-aligned to the human reference genome (version hg38.p12) using the STAR aligner [20] (version 2.7.0f). Similarly, we used the *MetaGraph* sequence-to-graph alignment to align the reads back to the GTEx index. In either setting, we used sensitive alignment criteria, utilizing `--outFilterMatchNmin 21` for STAR and `--align-min-seed-length 21` for *MetaGraph*. The latter setting does not apply for *MetaGraph* exact k-mer matching, where always a full k-mer is mapped.

4.13.8 Construction and query of TCGA index

All available TCGA RNA-Seq samples availabe at the Genomic Data commons were downloaded. A list containing all processed samples is available as file `TableS8_TCGA.tsv` in the Supplemental Data Section 5. The same assembly and annotation strategy as for GTEx samples was used, with the only difference that k was chosen as 31.

To generate the list of candidate trans-junctions, we iterated over all genes present in the GENCODE annotation (version 32) and formed for all transcripts

5 Supplemental Data

Additional resources for this project, including sample metadata, interactive notebooks and analysis scripts are available in GitHub at `https://github.com/ratschlab/metagraph_paper_` `resources`. The source code of the *MetaGraph* software is available under GPLv3 License at `https://github.com/ratschlab/metagraph`.

Acknowledgements

This work was supported through funds of ETH Zurich, MK and HM are funded by the Swiss National Science Foundation Grant No. 407540_167331 "Scalable Genome Graph Data Structures for Metagenomics and Genome Annotation" as part of Swiss National Research Programme (NRP) 75 "Big Data." The authors would like to thank the members of the MetaSUB international consortium for early access to raw sequencing data and their feedback on early versions of the geolocation DNA sequence search. We also would like to acknowledge the NCBI staff working on maintaining and developing the Sequence Read Archive for their support and interest in our project. The authors thank the members of the Biomedical Informatics Group at ETH Zurich for constant feedback and input on the project. The authors also would like to thank Google Inc. for providing a package of free compute credits on the Google Cloud Platform. We further would like to thank the donors and their families who contributed data to the GTEx and TCGA projects. This study contains data gathered by the Genotype-Tissue Expression (GTEx) Project available through dbGaP at http://www.ncbi.nlm.nih.gov/gap through dbGaP accession number phs000424.v7.p1. This study further contains data gathered by The Cancer Genome Atlas (TCGA) project available through dbGaP at http://www.ncbi.nlm.nih.gov/gap through dbGaP accession number phs000178.v1.p1. The authors are also grateful for the ability to use the publicly available data gathered by the gnomAD project.

References

[1] Brian P Alcock, Amogelang R Raphenya, Tammy TY Lau, Kara K Tsang, Mégane Bouchard, Arman Edalatmand, William Huynh, Anna-Lisa V Nguyen, Annie A Cheng, Sihan Liu, et al. Card 2020: antibiotic resistome surveillance with the comprehensive antibiotic resistance database. *Nucleic acids research*, 48(D1):D517–D525, 2020.

[2] Bahar Alipanahi, Alan Kuhnle, Simon J Puglisi, Leena Salmela, and Christina Boucher. Succinct Dynamic de Bruijn Graphs. *Bioinformatics*, 05 2020. btaa546.

[3] Bahar Alipanahi, Martin D Muggli, Musa Jundi, Noelle R Noyes, and Christina Boucher. Metagenome snp calling via read colored de bruijn graphs. *Bioinformatics*, 2020.

[4] Alexandre Almeida, Stephen Nayfach, Miguel Boland, Francesco Strozzi, Martin Beracochea, Zhou Jason Shi, Katherine S Pollard, Ekaterina Sakharova, Donovan H Parks, Philip Hugenholtz, et al. A unified catalog of 204,938 reference genomes from the human gut microbiome. *Nature Biotechnology*, pages 1–10, 2020.

[5] Fatemeh Almodaresi, Prashant Pandey, Michael Ferdman, Rob Johnson, and Rob Patro. An efficient, scalable and exact representation of high-dimensional color information enabled via de bruijn graph search. In *International Conference on Research in Computational Molecular Biology*, pages 1–18. Springer, 2019.

[6] Fatemeh Almodaresi, Prashant Pandey, and Rob Patro. Rainbowfish: A Succinct Colored de Bruijn Graph Representation. In Russell Schwartz and Knut Reinert, editors, *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*, volume 88 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[7] 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.

[8] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[9] European Nucleotide Archive. Ena statistics – assembled/annotated sequence growth. `https://www.ebi.ac.uk/ena/about/statistics`. Accessed: 2020-05-26.

[10] Jasmijn A Baaijens, Leen Stougie, and Alexander Schönhuth. Strain-aware assembly of genomes from mixed samples using flow variation graphs. In *International Conference on Research in Computational Molecular Biology*, pages 221–222. Springer, 2020.

[11] Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A Gurevich, Mikhail Dvorkin, Alexander S Kulikov, Valery M Lesin, Sergey I Nikolenko, Son Pham, Andrey D Prjibelski, et al. Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477, 2012.

[12] Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. Cobs: a compact bit-sliced signature index. In *International Symposium on String Processing and Information Retrieval*, pages 285–303. Springer, 2019.

[13] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2012.

[14] Phelim Bradley, Henk C den Bakker, Eduardo PC Rocha, Gil McVean, and Zamin Iqbal. Ultrafast search of all deposited bacterial and viral genomic data. *Nature biotechnology*, 37(2):152, 2019.

[15] FP Breitwieser, DN Baker, and Steven L Salzberg. Krakenuniq: confident and fast metagenomics classification using unique k-mer counts. *Genome biology*, 19(1):198, 2018.

[16] Xian Chang, Jordan Eizenga, Adam M Novak, Jouni Sirén, and Benedict Paten. Distance indexing and seed clustering in sequence graphs. *Bioinformatics*, 36(Supplement_1):i146–i153, 2020.

[17] Thomas C. Conway and Andrew J. Bromage. Succinct data structures for assembling large genomes. *Bioinformatics*, 27(4):479–486, feb 2011.

[18] Jeff Daily. Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments. *BMC bioinformatics*, 17(1):81, 2016.

[19] David Danko, Daniela Bezdan, Ebrahim Afshinnekoo, Sofia Ahsanuddin, Chandrima Bhattacharya, Daniel J Butler, Kern Rei Chng, Francesca De Filippis, Jochen Hecht, Andre Kahles, et al. Global genetic cartography of urban metagenomes and anti-microbial resistance. *BioRxiv*, page 724526, 2019.

[20] Alexander Dobin, Carrie A Davis, Felix Schlesinger, Jorg Drenkow, Chris Zaleski, Sonali Jha, Philippe Batut, Mark Chaisson, and Thomas R Gingeras. Star: ultrafast universal rna-seq aligner. *Bioinformatics*, 29(1):15–21, 2013.

[21] Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.

[22] Kyle Ellrott, Matthew H Bailey, Gordon Saksena, Kyle R Covington, Cyriac Kandoth, Chip Stewart, Julian Hess, Singer Ma, Kami E Chiotti, Michael McLellan, et al. Scalable open science approach for mutation calling of tumor exomes using multiple genomic pipelines. *Cell systems*, 6(3):271–281, 2018.

[23] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al. Variation graph toolkit improves read mapping by representing genetic variation in the reference. *Nature biotechnology*, 36(9):875–879, 2018.

[24] Evangelos Georganas, Aydın Buluç, Jarrod Chapman, Steven Hofmeyr, Chaitanya Aluru, Rob Egan, Leonid Oliker, Daniel Rokhsar, and Katherine Yelick. Hipmer: an extreme-scale de novo genome assembler. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2015.

[25] Evangelos Georganas, Rob Egan, Steven Hofmeyr, Eugene Goltsman, Bill Arndt, Andrew Tritt, Aydin Buluç, Leonid Oliker, and Katherine Yelick. Extreme scale de novo metagenome assembly. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 122–134. IEEE, 2018.

[26] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[27] Robert S Harris and Paul Medvedev. Improved representation of sequence bloom trees. *Bioinformatics*, 36(3):721–727, 2020.

[28] Mahdi Heydari, Giles Miclotte, Yves Van de Peer, and Jan Fostier. Browniealigner: accurate alignment of illumina sequencing data to de bruijn graphs. *BMC bioinformatics*, 19(1):1–10, 2018.

[29] Glenn Hickey, David Heller, Jean Monlong, Jonas A Sibbesen, Jouni Sirén, Jordan Eizenga, Eric T Dawson, Erik Garrison, Adam M Novak, and Benedict Paten. Genotyping structural variants in pangenome graphs using the vg toolkit. *Genome biology*, 21(1):1–17, 2020.

[30] Guillaume Holley, Roland Wittler, and Jens Stoye. Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage. *Algorithms for Molecular Biology*, 11(1):3, 12 2016.

[31] Weichun Huang, Leping Li, Jason R Myers, and Gabor T Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

[32] Zamin Iqbal, Mario Caccamo, Isaac Turner, Paul Flicek, and Gil McVean. De novo assembly and genotyping of variants using colored de Bruijn graphs. *Nature Genetics*, 2012.

[33] Pesho Ivanov, Benjamin Bichsel, Harun Mustafa, André Kahles, Gunnar Rätsch, and Martin Vechev. Astarix: Fast and optimal sequence-to-graph alignment. In *International Conference on Research in Computational Molecular Biology*, pages 104–119. Springer, 2020.

[34] Chirag Jain, Sanchit Misra, Haowen Zhang, Alexander Dilthey, and Srinivas Aluru. Accelerating sequence alignment to graphs. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 451–461. IEEE, 2019.

[35] Mikhail Karasikov, Harun Mustafa, Amir Joudaki, Sara Javadzadeh-No, Gunnar Rätsch, and André Kahles. Sparse binary relation representations for genome graph annotation. In *International Conference on Research in Computational Molecular Biology*, pages 120–135. Springer, 2019.

[36] Konrad J Karczewski, Laurent C Francioli, Grace Tiao, Beryl B Cummings, Jessica Alföldi, Qingbo Wang, Ryan L Collins, Kristen M Laricchia, Andrea Ganna, Daniel P Birnbaum, et al. The mutational constraint spectrum quantified from variation in 141,456 humans. *Nature*, 581(7809):434–443, 2020.

[37] Yuichi Kodama, Martin Shumway, and Rasko Leinonen. The sequence read archive: explosive growth of sequencing data. *Nucleic acids research*, 40(D1):D54–D56, 2012.

[38] Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. KMC 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.

[39] Dinghua Li, Chi-Man Liu, Ruibang Luo, Kunihiko Sadakane, and Tak-Wah Lam. Megahit: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de bruijn graph. *Bioinformatics*, 31(10):1674–1676, 2015.

[40] Antoine Limasset, Bastien Cazaux, Eric Rivals, and Pierre Peterlongo. Read mapping on de bruijn graphs. *BMC bioinformatics*, 17(1):1–12, 2016.

[41] Bo Liu, Hongzhe Guo, Michael Brudno, and Yadong Wang. debga: read alignment with de bruijn graph-based seed and extension. *Bioinformatics*, 32(21):3224–3232, 2016.

[42] John Lonsdale, Jeffrey Thomas, Mike Salvatore, Rebecca Phillips, Edmund Lo, Saboor Shad, Richard Hasz, Gary Walters, Fernando Garcia, Nancy Young, et al. The genotype-tissue expression (gtex) project. *Nature genetics*, 45(6):580, 2013.

[43] Charley GP McCarthy and David A Fitzpatrick. Pan-genome analyses of model fungal species. *Microbial genomics*, 5(2), 2019.

[44] Duccio Medini, Claudio Donati, Hervé Tettelin, Vega Masignani, and Rino Rappuoli. The microbial pan-genome. *Current opinion in genetics & development*, 15(6):589–594, 2005.

[45] Martin D. Muggli, Alexander Bowe, Noelle R. Noyes, Paul S. Morley, Keith E. Belk, Robert Raymond, Travis Gagie, Simon J. Puglisi, and Christina Boucher. Succinct colored de Bruijn graphs. *Bioinformatics*, 2017.

[46] Adam M Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional burrows–wheeler transform and its applications. *Algorithms for Molecular Biology*, 12(1):18, 2017.

[47] Sergey Nurk, Dmitry Meleshko, Anton Korobeynikov, and Pavel A Pevzner. metaspades: a new versatile metagenomic assembler. *Genome research*, 27(5):824–834, 2017.

[48] Nuala A. O'Leary, Mathew W. Wright, J. Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, Alexander Astashyn, Azat Badretdin, Yiming Bao, Olga Blinkova, Vyacheslav Brover, Vyacheslav Chetvernin, Jinna Choi, Eric Cox, Olga Ermolaeva, Catherine M. Farrell, Tamara Goldfarb, Tripti Gupta, Daniel Haft, Eneida Hatcher, Wratko Hlavina, Vinita S. Joardar, Vamsi K. Kodali, Wenjun Li, Donna Maglott, Patrick Masterson, Kelly M. McGarvey, Michael R. Murphy, Kathleen O'Neill, Shashikant Pujar, Sanjida H. Rangwala, Daniel Rausch, Lillian D. Riddick, Conrad Schoch, Andrei Shkeda, Susan S. Storz, Hanzhen Sun, Francoise Thibaud-Nissen, Igor Tolstoy, Raymond E. Tully, Anjana R. Vatsan, Craig Wallin, David Webb, Wendy Wu, Melissa J. Landrum, Avi Kimchi, Tatiana Tatusova, Michael DiCuccio, Paul Kitts, Terence D. Murphy, and Kim D. Pruitt. Reference sequence (RefSeq) database at NCBI: Current status, taxonomic expansion, and functional annotation. *Nucleic Acids Research*, 2016.

[49] Nuala A O'Leary, Mathew W Wright, J Rodney Brister, Stacy Ciufo, Diana Haddad, Rich McVeigh, Bhanu Rajput, Barbara Robbertse, Brian Smith-White, Danso Ako-Adjei, et al. Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, 44(D1):D733–D745, 2016.

[50] Brian D Ondov, Gabriel J Starrett, Anna Sappington, Aleksandra Kostic, Sergey Koren, Christopher B Buck, and Adam M Phillippy. Mash screen: High-throughput sequence containment estimation for genome discovery. *Genome biology*, 20(1):232, 2019.

[51] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. Mash: fast genome and metagenome distance estimation using minhash. *Genome biology*, 17(1):132, 2016.

[52] Yukiteru Ono, Kiyoshi Asai, and Michiaki Hamada. Pbsim: Pacbio reads simulator—toward accurate genome assembly. *Bioinformatics*, 29(1):119–121, 2013.

[53] Prashant Pandey, Fatemeh Almodaresi, Michael A Bender, Michael Ferdman, Rob Johnson, and Rob Patro. Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index. *Cell Systems*, 7 2018.

[54] Mikko Rautiainen, Veli Mäkinen, and Tobias Marschall. Bit-parallel sequence-to-graph alignment. *Bioinformatics*, 35(19):3599–3607, 2019.

[55] Mikko Rautiainen and Tobias Marschall. Graphaligner: Rapid and versatile sequence-to-graph alignment. *Genome Biology*, 21(1):1–28, 2020.

[56] Peter W Schreiber, Verena Kufner, Kerstin Hübel, Stefan Schmutz, Osvaldo Zagordi, Amandeep Kaur, Cornelia Bayard, Michael Greiner, Andrea Zbinden, Riccarda Capaul, et al. Metagenomic virome sequencing in living donor and recipient kidney transplant pairs revealed jc polyomavirus transmission. *Clinical Infectious Diseases*, 69(6):987–994, 2019.

[57] Brad Solomon and Carl Kingsford. Improved Search of Large Transcriptomic Sequencing Databases Using Split Sequence Bloom Trees. *Journal of Computational Biology*, 25(7):755–765, 7 2018.

[58] Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. Big data: Astronomical or genomical? *PLoS Biology*, 2015.

[59] John G Tate, Sally Bamford, Harry C Jubb, Zbyslaw Sondka, David M Beare, Nidhi Bindal, Harry Boutselakis, Charlotte G Cole, Celestino Creatore, Elisabeth Dawson, Peter Fish, Bhavana Harsha, Charlie Hathaway, Steve C Jupe, Chai Yin Kok, Kate Noble, Laura Ponting, Christopher C Ramshaw, Claire E Rye, Helen E Speedy, Ray Stefancsik, Sam L Thompson, Shicai Wang, Sari Ward, Peter J Campbell, and Simon A Forbes. COSMIC: the Catalogue Of Somatic Mutations In Cancer. *Nucleic Acids Research*, 47(D1):D941–D947, 10 2018.

[60] Isaac Turner, Kiran V Garimella, Zamin Iqbal, and Gil McVean. Integrating long-range connectivity information into de bruijn graphs. *Bioinformatics*, 34(15):2556–2565, 2018.

[61] John N Weinstein, Eric A Collisson, Gordon B Mills, Kenna R Mills Shaw, Brad A Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, Joshua M Stuart, Cancer Genome Atlas Research Network, et al. The cancer genome atlas pan-cancer analysis project. *Nature genetics*, 45(10):1113, 2013.

[62] Zheng Zhang, Scott Schwartz, Lukas Wagner, and Webb Miller. A greedy algorithm for aligning dna sequences. *Journal of Computational biology*, 7(1-2):203–214, 2000.