Lightweight compositional analysis of metagenomes with sourmash gather

Authors

- **Luiz Irber**
  [0000-0003-4371-9659](#) · [luizirber](#) · [luizirber](#)
  Graduate Group in Computer Science, UC Davis; Department of Population Health and Reproduction, UC Davis · Funded by Grant XXXXXXXX

- **C. Titus Brown**
  [0000-0001-6001-2677](#) · [ctb](#)
  Department of Population Health and Reproduction, UC Davis

Abstract

Here we describe an extension of MinHash that permits accurate compositional analysis of metagenomes with low memory and disk requirements.

Introduction

Compositional data analysis is the study of the parts of a whole using relative abundances [1]. This is a general problem with applications across many scientific fields [**???**], and examples in biology include RNA-Seq [**???**], metatranscriptomics [**???**], microbiome and metagenomics [**???**]. Taxonomic profiling is a particular instance of this general problem with the goal of finding the identity and relative abundance of microbial community elements at a specific taxonomic rank (species, genus, family), especially in metagenomic samples [**???**].

Existing taxonomic profilers use different methods to solve this problem, including aligning sequences to a reference database [**???**], using marker genes derived from known organisms from reference databases [**???**] or coupled with unknown organisms clustered from metagenomes [**???**], and exact k-mer matching using fixed k and lowest common ancestor (LCA) for resolving k-mer assignments matching multiple taxons from a reference database [**???**] or variable k and assigning multiple taxons per sequence, with an option to reduce it further to the LCA [**???**].

Once each sequence (from raw reads or assembled contigs) has a taxonomic assignment, these methods resolve the final identity and abundance for each member of the community by summarizing the assignments to a specific taxonomic rank, Taxonomic profiling is fundamentally limited by the availability of reference datasets to be used for assignments, and reporting what percentage of the sample is unassigned is important to assess results, especially in undercharacterized environments such as oceans and soil.

Results

Scaled MinHash sketches support containment operations

- scaled minhash supports similarity and containment

We define the Scaled MinHash on an input domain of k-mers, W, as follows:

$$\mathbf{SCALED}_s(W) = \left\{\, w \le \frac{H}{s} \mid \forall w \in W \,\right\}$$

where H is the largest possible value in the domain of $h(x)$ and $\frac{H}{s}$ is the value in the Scaled MinHash.

The Scaled MinHash is a mix of MinHash and ModHash ([**???**] broder). As in the former it keeps the smallest elements, and from the latter it adopts the dynamic size to allow containment estimation. However, instead of taking $0 \mod m$ elements like $\mathbf{MOD}_m(W)$, a Scaled MinHash uses a parameter s to select a subset of W.

Given a uniform hash function h and $s = m$, the cardinalities of $\mathbf{SCALED}_s(W)$ and $\mathbf{MOD}_m(W)$ converge for large $|W|$. The main difference is the range of possible values in the hash space, since

the Scaled MinHash range is contiguous and the ModHash range is not. Figure [**???**] shows an example comparing MinHash, ModHash and Scaled MinHash with the same parameter value.

Intuitively, Scaled MinHash is performing a density sampling at a rate of 1 k-mer per s k-mers seen.

Scaled MinHash permits Jaccard containment estimation between two data sets directly from the sketch …

A drawback of Scaled MinHash when compared to regular MinHash sketches is the size: the MinHash parameter s sets an upper bound on the size of the sketch, independently of the size of the original data. Scaled MinHash sketches grow proportionally to the original data cardinality, and in the worst case can have up to $\frac{H}{s}$ items.

Scaled MinHash sketches offer a fixed range of possible hash values, but with reduced sensitivity for small datasets when using larger s (scaled) values. A biological example are viruses: at $s = 2000$ many viruses are too small to consistently have a hashed value selected by the *Scaled MinHash* approach. Other *MinHash* approaches sidestep the problem by using hashing and streaming the query dataset (`Mash Screen`) or loading the query dataset into an approximate query membership data structure (`CMash`) to allow comparisons with the variable range of possible hash values, but both solutions require the original data or a more limited data representation than *Scaled MinHash*. The consistency of operating in the same data structure also allows further methods to be develop using only *Scaled MinHash* sketches and their features, especially if large collections of *Scaled MinHash* sketches are available.

Others have also applied the ModHash concept to genomic data; see, for example, Durbin's "modimizer" [2].

We provide two implementations of Scaled MinHash, `smol` and `sourmash`. `smol` is a minimal implementation of *Scaled MinHash* developed to demonstrate the method; it does not include many required features for working with real biological data, but its smaller code base makes it a more readable and concise example of the method. `sourmash` [3] implements features and functionality needed for large scale analyses of real data.

Scaled MinHash accurately estimates containment between sets of different sizes

- compares well with others
- supports large-scale sketching of genbank
- How much is missed figure; Poisson calculations?

We next compare the *Scaled MinHash* method implemented in `smol` to CMash (*Containment MinHash*) and Mash Screen (*Containment Score*) for containment queries in the Shakya dataset [4], a synthetic mock metagenomic bacterial and archaeal community where the organisms are known. Further analyses of low-coverage and contaminant genomes in this data set were described in [**???**] and [**???**].

For *Mash Screen* the ratio of hashes matched by total hashes is used instead of the *Containment Score*, since the latter uses a k-mer survival process modeled as a Poisson process first introduced in [**???**] and later used in the *Mash distance* [**???**] and *Containment score* [**???**] formulations.

Experiments use $k = \{21, 31, 51\}$ (except for Mash, which only supports $k \leq 32$). For Mash and CMash they were run with $n = \{1000, 10000\}$ to evaluate the containment estimates when using larger sketches with sizes comparable to the Scaled MinHash sketches with $scaled = 1000$. The truth set is calculated using an exact k-mer counter implemented with a *HashSet* data structure in the Rust programming language [**???**].
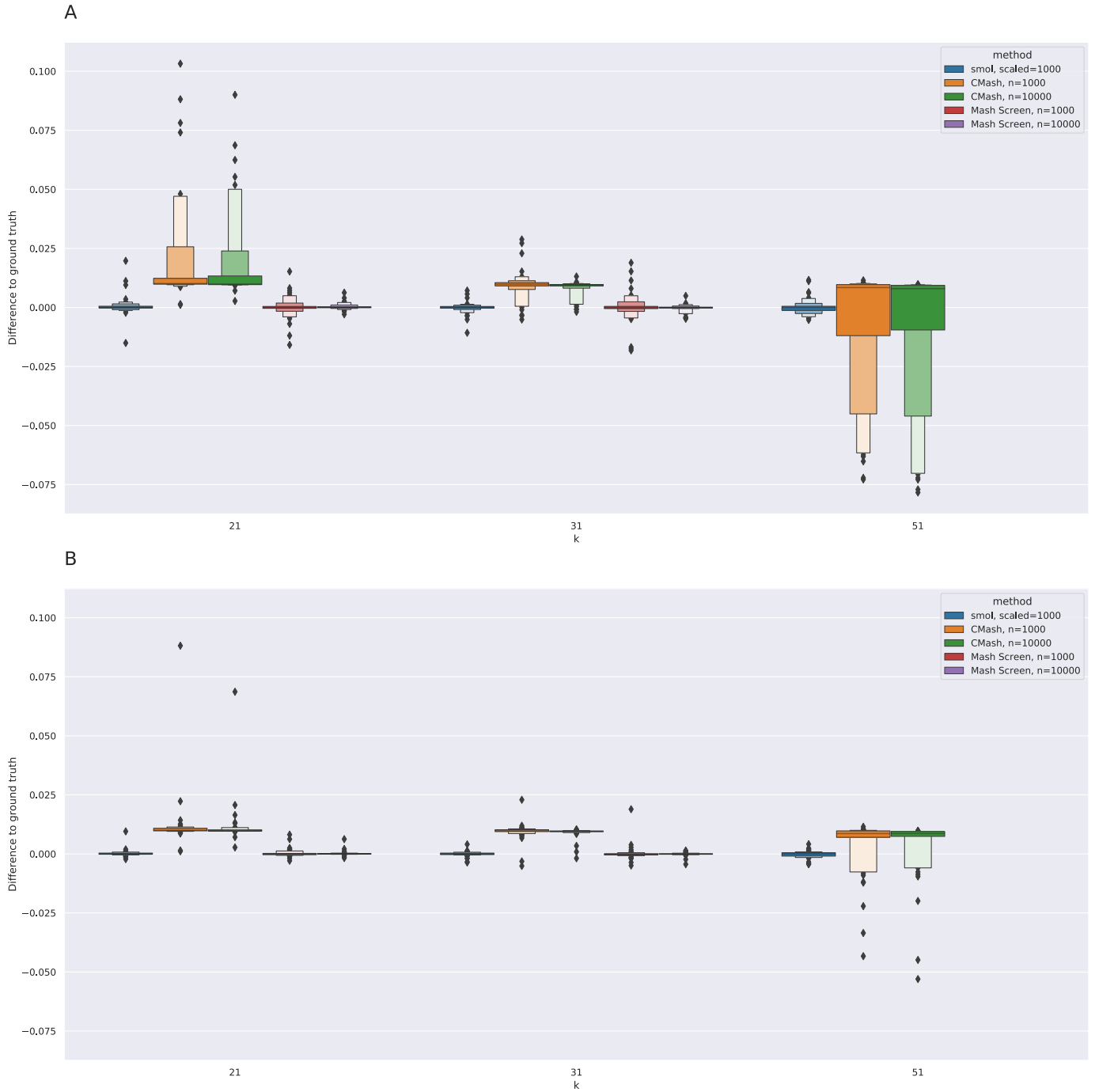
Figure 1: Letter-value plot [???] of the differences from containment estimate to ground truth (exact). Each method is evaluated for $k = \{21, 31, 51\}$, except for `Mash` with $k = 51$, since `Mash` doesn't support $k > 32$. **A**: Using all 68 reference genomes found in previous articles. **B**: Excluding low coverage genomes identified in previous articles.

All methods are within 1% of the exact containment on average (Figure ?? A), with `CMash` consistently underestimating the containment for large k and overestimating for small k. `Mash Screen` with $n = 10000$ has the smallest difference to ground truth for $k = \{21, 31\}$, followed by `smol` with `scaled=1000` and `Mash Screen` with $n = 1000$.

In order to evaluate the effect of the low-coverage and contaminant genomes previously detected in this dataset, (Figure ?? B) shows results with these genomes removed. The number of outliers is greatly reduced, with most methods within 1% absolute difference to the ground truth. `CMash` still has some outliers with up to 8% difference to the ground truth.

CTB:

- runtimes?
- Where does "Scaled MinHash sketch sizes across GenBank domains" go? (Chp 01 from Luiz thesis)

Scaled MinHash sketches support efficient indexing for large-scale containment queries

- hierarchical and inverted indices (SBT and LCA)
- supports efficient containment and similarity queries

Searching for matches in large collection of datasets is challenging when hundreds of thousands of them are available, especially if they are partitioned and the data is not all present at the same place, or too large to even be stored in a single system. Efficient methods for sequencing datasets use exact k-mer matching instead of relying on sequence alignment, but sensitivity is reduced since they can't deal with sequencing errors and biological variation as well as alignment-based methods can.

CTB: Additional points to raise: in-memory representation of sketches may be too big (!!), goal here is on disk storage/low minimum memory for "extremely large data" situation. Also/in addition, want ability to do incremental loading of things. Note we are not talking here about situations where the indices themselves are too big to download, could maybe include forward pointer to chp4.

Note, in this section we could also include distinction in performance between SBT and LCA DB, to whit: large scaled works well with LCA (small DB, ~tolerable memory, load all at once, then quite fast) but low scaled may work (much) better with SBT.

`sourmash` [3] is a software for large-scale sequence data comparisons based on MinHash sketches. Initially implementing operations for computing, comparing and plotting distance matrices for *MinHash* sketches, version 2 [**???**] introduces *Scaled MinHash* sketches and indices for this new sketch format. Indices support a common set of operations (insertion, search, and returning all signatures are the main ones), allowing them to be used interchangeably depending on the use case, performance requirements and computational resources available.

Scaled MinHash sketches can be stored in any data structure for representing the k-mer composition M of a dataset [**???**], and as a subset of M they can also be indexed by approaches for the full k-mer composition. `sourmash` [**???**] defines the MinHash Bloom Tree (*MHBT*) index, a k-mer aggregative method with explicit representation of datasets based on hierarchical indices and a specialization of the Sequence Bloom Tree [**???**], as well as the *LCA* index, a color-aggregative method with implicit representation of the datasets based on inverted indices.

The simplest index is the `LinearIndex`, a list of signatures. Search operations are executed sequentially, and insertions append new signatures to the end of the list. Internally, `sourmash` uses LinearIndex as the default index for lists of signatures provided in the command-line.

The *MinHash Bloom Tree* (*MHBT*) is a variation of the *Sequence Bloom Tree* (*SBT*) that uses Scaled MinHash sketches as leaf nodes instead of Bloom Filters as in the SBT. The search operation in SBTs is defined as a breadth-first search starting at the root of the tree, using a threshold of the original k-

mers in the query to decide when to prune the search. MHBTs use a query Scaled MinHash sketch instead, but keep the same search approach. The threshold of a query Q approach introduced in [**???**] is equivalent to the containment

$$C(Q, S) = \frac{|Q \cap S|}{|S|}$$

described in [**???**], where S is a Scaled MinHash sketch. For internal nodes n (which are Bloom Filters) the containment of the query Scaled MinHash sketch Q is

$$C(Q, n) = \frac{|\{ h \in n \mid \forall h \in Q \}|}{|Q|}$$

the same containment score defined in [**???**] for the *Containment MinHash* to *Bloom Filter* comparison.

MHBTs support both containment and similarity queries. For internal nodes the containment $C(Q, n)$ is used as an upper-bound of the similarity $J(Q, n)$:

C(Q, n) &\ge J(Q, n) \\
\frac{\vert Q \cap n \vert }{\vert Q \vert} \ge \frac{\vert Q \cap n \vert }{\vert Q \cup n \vert}

since $|Q \cup n| \ge |Q|$. When a leaf node is reached then the similarity $J(Q, S)$ is calculated for the Scaled MinHash sketch S and declared a match if it is above the threshold t. Because the upper-bound is being used, this can lead to extra nodes being checked, but it simplifies implementation and provides better correctness guarantees.

The LCA index in sourmash is an inverted index that stores a mapping from hashes in a collection of signatures to a list of IDs for signatures containing the hash. Despite the name, the list of signature IDs is not collapsed to the lowest common ancestor (as in kraken), and is calculated as needed by downstream methods using the taxonomy information that is also stored separately in the LCA index.

The mapping from hashes to signature IDs in the LCA index is an implicit representation of the original signatures used to build the index, and so returning the signatures is implemented by rebuilding the original signatures on-the-fly. Search in an LCA index matches the k-mers in the query to the list of signatures IDs containing them, using a counter data structure to sort results by number of hashes per signature ID. The rebuilt signatures are then returned as matches based on the signature ID, with containment or similarity to the query calculated against the rebuilt signatures.

mash screen [**???**] has a similar index, but it is constructed on-the-fly using the distinct hashes in a sketch collection as keys, and values are counters initially set to zero. As the query is processed, matching hashes have their counts incremented, and after all hashes in the query are processed then all the sketches in the collection are checked in the counters to quantify the containment/similarity of each sketch in the query. The LCA index uses the opposite approach, opting to reconstruct the sketches on-the-fly.

In order to evaluate MHBT and LCA indices construction a GenBank snapshot from July 18, 2020, containing 725,331 assembled genomes (5,282 Archaea, 673,414 Bacteria, 6,601 Fungi 933 Protozoa and 39,101 Viral) was used to measure runtime, memory consumption and final index size. MHBT indices were built with $scaled = 1000$, and LCA indices used $scaled = 10000$. Table 1 shows the results for the LCA index, and Table 2 for the MHBT index.

Table 1: Results for LCA indexing, with $scaled = 10000$ and $k = 21$.

Domain	Runtime (s)	Memory (MB)	Size (MB)
Viral	57	33	2
Archaea	58	30	5

Domain	Runtime (s)	Memory (MB)	Size (MB)
Protozoa	231	3	17
Fungi	999	3	65
Bacteria	12,717	857	446

Table 2: Results for MHBT indexing, with $scaled = 1000$, $k = 21$ and internal nodes (Bloom Filters) using 10000 slots for storage.

Domain	Runtime (s)	Memory (MB)	Size (MB)
Viral	126	326	77
Archaea	111	217	100
Protozoa	206	753	302
Fungi	1,161	3,364	1,585
Bacteria	32,576	47,445	24,639

Index sizes are more affected by the number of genomes inserted than the individual *Scaled MinHash* sizes. Despite Protozoan and Fungal *Scaled MinHash* sketches being larger individually, the Bacterial indices are an order of magnitude larger for both indices since they contain two orders of magnitude more genomes.

Comparing between LCA and MHBT index sizes must account for their different scaled parameters, but as shown in Chapter 1 a *Scaled MinHash* with $scaled = 1000$ when downsampled to $scaled = 10000$ is expected to be ten times smaller. Even so, MHBT indices are more than ten times larger than their LCA counterparts, since they store extra caching information (the internal nodes) to avoid loading all the data to memory during search. LCA indices also contain extra data (the list of datasets containing a hash), but this is lower than the storage requirements for the MHBT internal nodes.

For the purpose of evaluating the performance characteristics of MHBT and LCA indices when performing searches, each of the previously described indices generated from GenBank domains was used to execute similarity searches (finding datasets in a collection that are similar to a query) using appropriate queries for each domain. All queries were selected from the relevant domain and queried against both MHBT ($scaled = 1000$) and LCA ($scaled = 10000$), for $k = 21$.

Table 3: Running time in seconds for similarity search using LCA ($scaled = 10000$) and MHBT ($scaled = 1000$) indices.

	Viral	Archaea	Protozoa	Fungi	Bacteria
LCA	1.06	1.42	5.40	26.92	231.26
SBT	1.32	3.77	43.51	244.77	3,185.88

Table 4: Memory consumption in megabytes for similarity search using LCA ($scaled = 10000$) and MHBT ($scaled = 1000$) indices.

	Viral	Archaea	Protozoa	Fungi	Bacteria
LCA	223	240	798	3,274	20,926
SBT	163	125	332	1,656	2,290

Table [3] shows running time for both indices. For small indices (Viral and Archaea) the LCA running time is dominated by loading the index in memory, but for larger indices the cost is amortized due to the faster running times. This situation is clearer for the Bacteria indices, where the LCA search completes in 3 minutes and 51 seconds, while the SBT search takes 54 minutes.

When comparing memory consumption, the situation is reversed. Table [4] shows how the LCA index consistently uses twice the memory for all domains, but for larger indices like Bacteria it uses as much as 10 times the memory as the MHBT index for the same data.

For both runtime and memory consumption, it is worth pointing that the LCA index is a tenth of the data indexed by the MHBT. This highlights the trade-off between speed and memory consumption for both approaches, especially for larger indices.

Metagenome sketches can be accurately decomposed into constituent genomes by a greedy algorithm, 'gather'

- outline algorithm
- compare conceptually vs least/lowest common ancestor approaches; combinatorial
- showcase some examples on synthetic data

Methods summarizing taxonomic assignments from sequences in the query metagenome to calculate the profile for the community follow a bottom-up approach. `gather` is a new method following a top-down approach: starting from the k-mer composition of the query, it iteratively finds a match in a collection of datasets with the largest *containment* of the query (most elements in common), and create a new query by *removing elements* in the match from the original query. The process stops when the new query doesn't have any more matches in the collection, or a user-provided minimum detection threshold is reached. This approach differs from previous methods because the co-occurrence of k-mers in a match is considered a strong signal that they are coming from the same organism in the original sample, and is used instead of the LCA-based methods to resolve ambiguities in the taxonomic assignment of a sequence (or its k-mers).

Any data structure supporting both the *containment* $C(A, B) = \frac{|A \cap B|}{|A|}$ and *remove elements* operations can be used as a query with `gather`. For example, a *set* of the k-mer composition of the query supports element removal, and calculating containment can be done with regular set operations. Approximate membership query (AMQ) sketches like the *Counting Quotient Filter* [???] can also be used, with the benefit of reduced storage and memory usage. Moreover, the collection of datasets can be implemented with any data structure that can do containment comparisons with the query data structure, including implicit representations like an inverted index from hashed k-mers to dataset IDs (as detailed in subsection [2.1.2]).

With *Scaled MinHash* sketches, the same hash w' will be present in sketches for other datasets also containing the k-mer m, as long as they have the same parameter s or can be downsampled to the same s. This is not guaranteed for regular *MinHash* sketches, and is what allows removing elements from the *Scaled MinHash* sketch of a query once a match is found. Since the containment of two datasets can also be estimated directly from their *Scaled MinHash* sketches, they are viable data structures for `gather`, especially since they are only a small fraction of the original dataset size and easier to store, manipulate and share.

Compared to previous taxonomic profiling methods, *Scaled MinHash* can also be seen as a mix of two other approaches: It uses exact k-mer matching and assignment, and the k-mers selected by the MinHashing process are equivalent to implicitly-defined markers. It differs from previous approaches

because only a subset of the k-mer composition is used for matching, and traditional gene markers are explicitly chosen due to sequence conservation and low mutation rates, while MinHashing k-mers generates a randomized, but consistent across datasets, set of marker k-mers.

CTB note: gather can be done on exact k-mers as well.

Algorithm 1 describes the `gather` method using a generic operation `FindBestContainment`. An implementation for `FindBestContainment` for a list of datasets is presented in Algorithm 2. Appendix A has a minimal implementation in the Rust programming language [**???**] for both algorithms, including a *Scaled MinHash* sketch implementation using a *set* data structure from the Rust standard library (`HashSet`).

`gather` is implemented as a method of the `Index` abstract base class (ABC) in `sourmash`. This ABC declares methods that any index in `sourmash` has to implement, but each index is allowed to implement it in the most efficient or performant way:

1. For `Linear` indices, the `FindBestContainment` operation is implemented as a linear scan over the list of signatures (Algorithm 2);

2. For `MHBT` indices, `FindBestContainment` is implemented as a depth-first search that tracks the best containment found, and prunes the search if it the current estimated containment in an internal node is lower than the current best containment.

3. `LCA` indices can implement `gather` by building a counter of how many hashes of the query are present in each signature, and then using the signature with the largest count as a match. As matches are found, the count for the hashes in the match are decreased in the counter, and then the new signature with the largest count is the next match.

`sourmash gather`, the command-line interface that adds further user experience improvements to the API level, also allows passing multiple indices to be searched, providing incremental support for rerunning with additional data without having to recompute, merge or update the original databases.

Taxonomic profiling based on 'gather' is accurate

- CAMI results
- suggests gather/greedy decomposition is pretty good

Taxonomic profiling in `sourmash` is built as an extra step on top of the `gather` algorithm. `gather` returns assignments to a dataset in a collection, and based on that assignment the extra step associates a taxonomic ID (based on some dataset identifier) and a taxonomic lineage (a path from root to taxonomic ID) derived from a specific taxonomy. After a lineage is available, each taxonomic rank is summarized based on the abundances under it.

The Critical Assessment of Metagenome Intepretation (CAMI) [**???**] is a community-driven initiative bringing together tool developers and users to create standards for reproducibly benchmarking metagenomic methods. Challenges are organized around datasets representing microbial communities of interest in metagenomics, like marine, high-strain and rhizosphere datasets. Sequencing data is generated by CAMISIM [**???**], a microbial community and metagenome simulator using a gold standard with a known community composition to model different aspects (diversity levels, abundances and sequencing technologies features) of these datasets.

Each challenge typically includes three tasks: assembly, taxonomic profiling and binning (at taxon or genome levels). Since there is a standard output format that tools need to implement, performance comparisons can be streamlined. CAMI provides a set of tools for computing performance metrics for each group: MetaQUAST for assembly, AMBER for binning, and OPAL [**???**] for taxonomic profiling evaluation.

`gather` can be used for the taxonomic profiling task, where the goal is finding what organisms are present in a metagenome sampled from a microbial community, and what are their relative abundances. Taxonomic profiling is based on a predetermined taxonomy of known organisms, as well as a collection of genomes for each organism. It differs from taxonomic classification, where each read or sequence in the metagenome is given a taxonomic assignment, and from binning, which aims to cluster reads or sequences into bins, possibly representing unknown organisms.

The first set of CAMI challenges happened in 2015 and results were published in 2017. Since then more tools were developed and improved, as well as reference databases growing in size and diversity. Reproducing the running environment used by the original tools is challenging, even with all the focus on reproducibility by the organizers and community.

The CAMI initiative released new challenges in 2019 (marine, high-strain and pathogen detection) and 2020 (rhizosphere), with updated processes for submission, evaluation and participation. In addition to short-read sequencing data matching Illumina profiles, it also includes long-read sequencing data with PacBio and Nanopore profiles, allowing further benchmarks and comparisons. CAMI also provides a snapshot of the RefSeq reference genomes for building specialized databases for each tool, as well with an NCBI Taxonomy to minimize differences in taxonomic reports. Since challenges only release the gold standard after they are concluded and published, results for comparison with new methods are still pending.

The CAMI II mouse gut metagenome [**???**] is a toy dataset, used for preparing and calibrating tools for other CAMI II challenges. Similar to the concluded challenges from CAMI, it provides gold standards for expected microbial community composition, including presence and relative abundance of organisms. The simulated mouse gut metagenome (*MGM*) was derived from 791 bacterial and archaeal genomes, representing 8 phyla, 18 classes, 26 orders, 50 families, 157 genera, and 549 species. 64 samples were generated with *CAMISIM*, with 91.8 genomes present on each sample on average. Each sample is 5 GB in size, and both short-read (Illumina) and long-read (PacBio) sequencing data is available.

Because the official challenges don't have gold standards published yet, it is currently the only alternative for using the CAMI benchmarking tools to evaluate new methods with updated datasets. Curated metadata for multiple tools is also available, and users can submit their tools for inclusion. All tools currently in the curated metadata repository use the short-read samples.
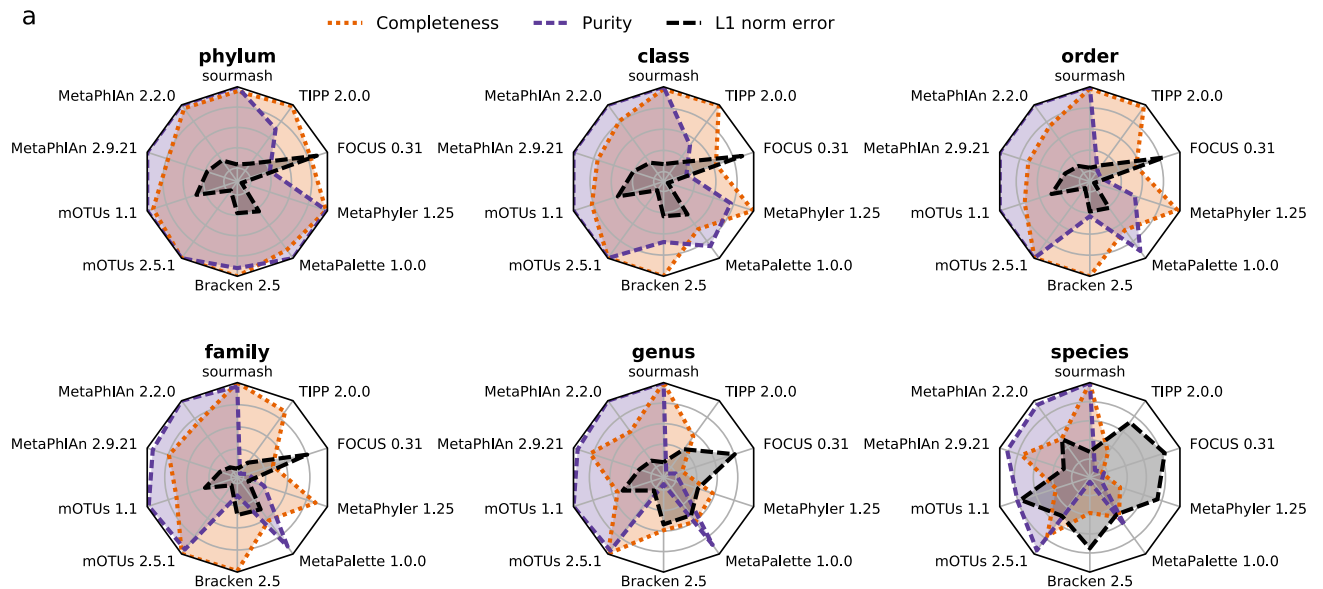
Figure 2: Comparison per taxonomic rank of methods in terms of completeness, purity (1% filtered), and L1 norm.
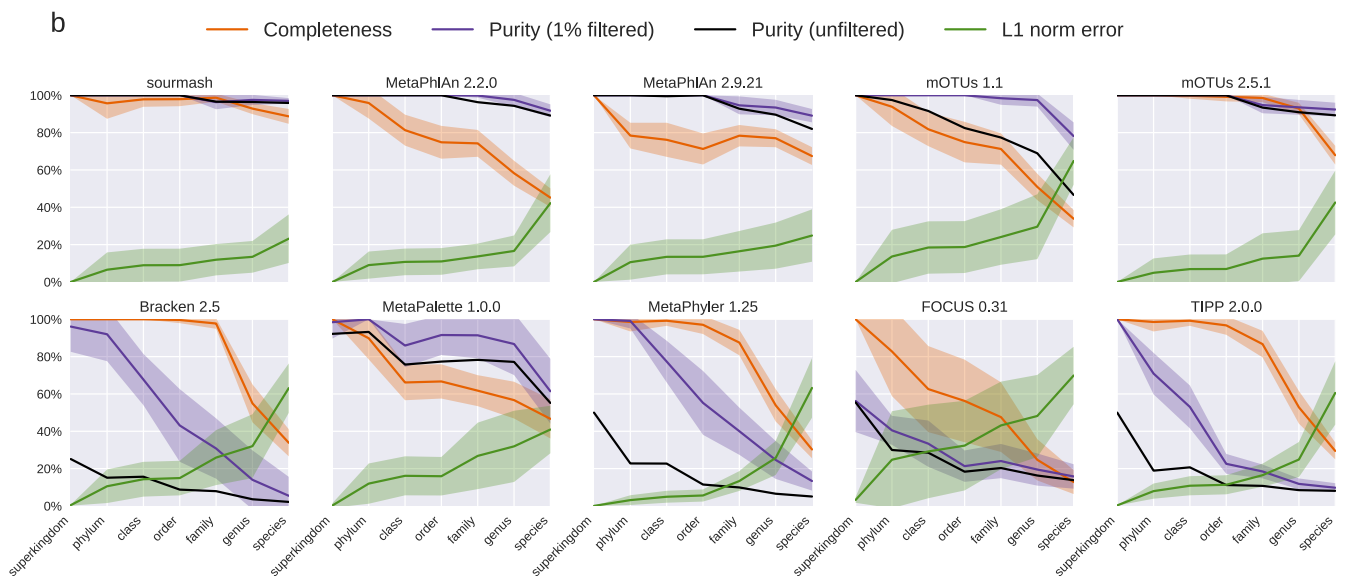
Figure 3: Performance per method at all major taxonomic ranks, with the shaded bands showing the standard deviation of a metric. In **a** and **b**, completeness, purity, and L1 norm error range between 0 and 1. The L1 norm error is normalized to this range and is also known as Bray-Curtis distance. The higher the completeness and purity, and the lower the L1 norm, the better the profiling performance.

c	Completeness	Purity (1% filtered)	L1 norm error	Sum of scores
1st	sourmash (247)	sourmash (179)	mOTUs 2.5.1 (789)	sourmash (1262)
2nd	mOTUs 2.5.1 (416)	MetaPhlAn 2.2.0 (241)	sourmash (836)	mOTUs 2.5.1 (1887)
3rd	Bracken 2.5 (1008)	mOTUs 1.1 (631)	MetaPhlAn 2.9.21 (1401)	MetaPhlAn 2.2.0 (3527)
4th	MetaPhyler 1.25 (1298)	mOTUs 2.5.1 (682)	MetaPhlAn 2.2.0 (1497)	MetaPhlAn 2.9.21 (4349)
5th	TIPP 2.0.0 (1424)	MetaPhlAn 2.9.21 (789)	MetaPhyler 1.25 (1586)	MetaPhyler 1.25 (5148)
6th	MetaPhlAn 2.2.0 (1789)	MetaPalette 1.0.0 (1182)	mOTUs 1.1 (2317)	mOTUs 1.1 (5253)
7th	MetaPhlAn 2.9.21 (2159)	MetaPhyler 1.25 (2264)	TIPP 2.0.0 (2361)	MetaPalette 1.0.0 (5989)
8th	mOTUs 1.1 (2305)	Bracken 2.5 (2881)	MetaPalette 1.0.0 (2390)	Bracken 2.5 (6574)
9th	MetaPalette 1.0.0 (2417)	TIPP 2.0.0 (3361)	Bracken 2.5 (2685)	TIPP 2.0.0 (7146)
10th	FOCUS 0.31 (3424)	FOCUS 0.31 (3764)	FOCUS 0.31 (3894)	FOCUS 0.31 (11082)

Figure 4: Methods rankings and scores obtained for the different metrics over all samples and taxonomic ranks. For score calculation, all metrics were weighted equally.

Figure [2], [3], [4] is an updated version of Figure 6 from [**???**] including `sourmash`, comparing 10 different methods for taxonomic profiling and their characteristics at each taxonomic rank. While previous methods show reduced completeness, the ratio of taxa correctly identified in the ground truth, below the genus level, `sourmash` can reach 88.7% completeness at the species level with the highest purity (the ratio of correctly predicted taxa over all predicted taxa) across all methods: 95.9% when filtering predictions below 1% abundance, and 97% for unfiltered results. `sourmash` also has the lowest L1-norm error (the sum of the absolute difference between the true and predicted abundances at a specific taxonomic rank), the highest number of true positives and the lowest number of false positives.

Table 5: Updated Supplementary Table 12 from [**???**]. Elapsed (wall clock) time (h:mm) and maximum resident set size (kbytes) of taxonomic profiling methods on the 64 short read samples of the CAMI II mouse gut data set. The best results are shown in bold. Bracken requires to run Kraken, hence the times required to run Bracken and both tools are shown. The taxonomic profilers were run on a computer with an Intel Xeon E5-4650 v4 CPU (virtualized to 16 CPU cores, 1 thread per core) and 512 GB (536.870.912 kbytes) of main memory.

Taxonomic binner	Time (hh:mm)	Memory (kbytes)
MetaPhlAn 2.9.21	18:44	5,139,172
MetaPhlAn 2.2.0	12:30	1,741,304
Bracken 2.5 (only Bracken)	**0:01**	**24,472**
Bracken 2.5 (Kraken and Bracken)	**3:03**	39,439,796
FOCUS 0.31	13:27	5,236,199
CAMIARKQuikr 1.0.0	16:19	27,391,555
mOTUs 1.1	19:50	**1,251,296**
mOTUs 2.5.1	14:29	3,922,448
MetaPalette 1.0.0	76:49	27,297,132
TIPP 2.0.0	151:01	70,789,939
MetaPhyler 1.25	119:30	2,684,720
sourmash 3.4.0	16:41	5,760,922

When considering resource consumption and running times, `sourmash` used 5.62 GB of memory with an *LCA index* built from the RefSeq snapshot (141,677 genomes) with $scaled = 10000$ and $k = 51$. Each sample took 597 seconds to run (on average), totalling 10 hours and 37 minutes for 64 samples. MetaPhlan 2.9.21 was also executed in the same machine, a workstation with an AMD Ryzen 9 3900X 12-Core CPU running at 3.80 GHz, 64 GB DDR4 2133 MHz of RAM and loading data from an NVMe SSD, in order to compare to previously reported times in Table [5]. MetaPhlan took 11 hours and 25 minutes to run for all samples, compared to 18 hours and 44 minutes previously reported, and correcting the `sourmash` running time by this factor it would likely take 16 hours and 41 minutes in the machine used in the original comparison. After correction, `sourmash` has similar runtime and memory consumption to the other best performing tools (*mOTUs* and *MetaPhlAn*), both gene marker and alignment based tools.

Additional points are that `sourmash` is a single-threaded program, so it didn't benefit from the 16 available CPU cores, and it is the only tool that could use the full RefSeq snapshot, while the other tools can only scale to a smaller fraction of it (or need custom databases). The CAMI II RefSeq

snapshot for reference genomes also doesn't include viruses; this benefits `sourmash` because viral *Scaled MinHash* sketches are usually not well supported for containment estimation, since viral sequences require small scaled values to have enough hashes to be reliable.

Discussion

Scaled MinHash offers benefits, drawbacks vs regular MinHash

Combine theoretical discussion with practical discussion of benefits/drawbacks.

Good:

- supports containment on sketches directly, vs CMash and mash screen
- supports other operations
- supports downsampling

Bad:

- size grows with size of input data
- does not have sensitivity at small sizes

Scaled MinHash supports operating directly on sketches

(CTB: Portions of this may belong in Results; and/or combine with previous section)

Once a Scaled MinHash is calculated there are many operation that can be applied without depending on the original data, saving storage space and allowing scaling analysis to thousands of datasets. Most of these operations are also possible with MinHash and ModHash, with caveats. One example of these operations is : the contiguous value range for Scaled MinHash sketches allow deriving $\mathbf{SCALED}_{s'}(W)$ sketches for any $s' \geq s$ using only $\mathbf{SCALED}_s(W)$. MinHash and ModHash can also support this operation, as long as $n' \leq n$ and m' is a multiple of m.

Because Scaled MinHash sketches collect any value below a threshold this also guarantees that once a value is selected it is never discarded. This is useful in streaming contexts: any operations that used a previously selected value can be cached and updated with new arriving values. $\mathbf{MOD}_m(W)$ has similar properties, but this is not the case for $\mathbf{MIN}_n(W)$, since after n values are selected any displacement caused by new data can invalidate previous calculations.

Abundance tracking is another extension to MinHash sketches, keeping a count of how many times a value appeared in the original data. This allows filtering for low-abundance values, as implemented in Finch [**???**], another MinHash sketching software for genomics. Filtering values that only appeared once was implemented before in Mash by using a Bloom Filter and only adding values after they were seen once, with later versions also implementing an extra counter array to keep track of counts for each value in the MinHash.

TODO: discuss here how abundance tracking in MinHash is not "correct", because it is not a proper weighted subsample of the data? Note that Scaled MinHash is a proper weighted subsample.

Other operations are adding and subtracting hash values from a Scaled MinHash sketch, allowing post-processing and filtering. Although possible for $\mathbf{MIN}_n(W)$, in practice this requires

oversampling (using a larger n) to account for possibly having less than n values after filtering (the approach taken by Finch [**???**]).

Gather works surprisingly well and matches simple data structures

gather is a straightforward algorithm.

easy to take advantage of other data structures b/c "just k-mers".

SBT, LCA implementations. Talk about k-mer and colro aggregative methods?

xx can we guess at places where gather would break?

`gather` is a new method for decomposing datasets into its components that outperforms current method when using synthetic datasets with known composition. By leveraging *Scaled MinHash* sketches and efficient indexing data structures it can scale the number of reference datasets used by over an order of magnitude when compared to existing methods.

Other containment estimation methods described in Chapter [1](), *CMash* [**???**] and *mash screen* [**???**], can also implement `gather`. Running a search requires access to the original dataset (*mash screen*) for the query, or a Bloom Filter derived from the original dataset (*CMash*), and when the collection of reference sketches is updated the Bloom Filter from *CMash* can be reused, but *mash screen* needs access to the original dataset again.

Since *Scaled MinHash* sketches allow using the sketch directly for `gather`, which are a fraction of the original data in size and also allow enumerating all the elements, an operation not possible with Bloom Filters, they can be stored and reused for large collections of sequencing datasets, including public databases like the Sequence Read Archive [**???**]. A service that calculate these *Scaled MinHash* sketches and make them available can improve discoverability of these large collections, as well as support future use cases derived from other *Scaled MinHash* features.

Taxonomic profiling is fundamentally limited by the availability of reference datasets, even if new reference datasets can be derived from clustering possible organisms based on sequence data in metagenomes [**???**]. `gather` as implemented in `sourmash` is a method that can scale to increasingly larger collections of datasets due to multiple reasons:

- containment and similarity estimation with *Scaled MinHash* sketches has lower computational requirements than alignment over all reads of a dataset;

- since *Scaled MinHash* sketches use a subset of the k-mer composition, they also scale better than full k-mer composition representations, requiring less space and reducing the number of elements to be computed;

- querying multiple databases can be done independently, avoiding the need to merge, update or reprocess databases when new datasets are available. A new database with the new datasets can be constructed and queried together with previous ones.

Taxonomic profiling in `sourmash` is implemented as an extra step on top of `gather` results. Because these steps are independent of the dataset assignment that `gather` generates, updates to the taxonomy don't require re-executing `gather`, since the taxonomic information can be derived

from the same dataset identifier (but potentially with a new associated taxonomic ID). This allows using new taxonomies derived from the same underlying datasets [**???**], as well as updates to the original taxonomy used before.

Despite improvements to standardization and reproducibility of previous analysis, benchmarking taxonomic profiling tools is still challenging, since tools can generate their reference databases from multiple sources and choosing only one source can bias or make it impossible to evaluate them properly. This is especially true for real metagenomic datasets derived from samples collected from soil and marine environments, where the number of unknown organisms is frequently larger than those contained in reference databases. With the advent of metagenome-assembled genomes (MAGs) there are more resources available for usage as reference datasets, even if they are usually incomplete or draft quality. `sourmash` is well positioned to include these new references to taxonomic profiling given the minimal requirements (a *Scaled MinHash* sketch of the original dataset) and support for indexing hundreds of thousands of datasets.

Limitations of gather

`gather` as implemented in `sourmash` has the same limitations as *Scaled MinHash* sketches, including reduced sensitivity to small genomes/sequences such as viruses. *Scaled MinHash* sketches don't preserve information about individual sequences, and short sequences using large scaled values have increasingly smaller chances of having any of its k-mers (represented as hashes) contained in the sketch. Because it favors the best containment, larger genomes are also more likely to be chosen first due to their sketches have more elements, and further improvements can take the size of the match in consideration too. Note that this is not necessarily the *similarity* $J(A, B)$ (which takes the size of both A and B), but a different calculation that normalizes the containment considering the size of the match.

`gather` is also a greedy algorithm, choosing the best containment match at each step. Situations where multiple matches are equally well contained or many datasets are very similar to each other can complicate this approach, and additional steps must be taken to disambiguate matches. The availability of abundance counts for each element in the *Scaled MinHash* is not well explored, since the process of *removing elements* from the query doesn't account for them (the element is removed even if the count is much higher than the count in the match). Both the multiple match as well as the abundance counts issues can benefit from existing solutions taken by other methods, like the *species score* (for disambiguation) and *Expectation-Maximization* (for abundance analysis) approaches from Centrifuge [**???**].

Future directions

In this chapter `gather` is described in terms of taxonomic profiling of metagenomes. That is one application of the algorithm, but it can applied to other biological problems too. If the query is a genome instead of a metagenome, `gather` can be used to detect possible contamination in the assembled genome by using a collection of genomes and removing the query genome from it (if it is present). This allows finding matches that contain the query genome and evaluating if they agree at specific taxonomic rank, and in case of large divergence (two different phyla are found, for example) it is likely to indicative that the query genome contains sequences from different organisms. This is especially useful for quality control and validation of metagenome-assembled genomes (MAGs), genomes assembled from reads binned and clustered from metagenomes, as well as a verification during submission of new assembled genomes to public genomic databases like GenBank.

Improvements to the calculation of *Scaled MinHash* sketches can also improve the taxonomic profiling use case. Exact k-mer matching is limited in phylogenetically distant organisms, since small

nucleotide differences lead to distinct k-mers, breaking homology assumptions. Different approaches for converting the datasets into a set to be hashed (*shingling*) than computing the nucleotide k-mer composition, such as spaced k-mers [**???**] and minimizers [**???**] and alternative encodings for the nucleotides using 6-frame translation to amino acid [**???**] or other reduced alphabets [**???**], can allow comparisons on longer evolutionary distances and so improve taxonomic profiling by increasing the sensitivity of the containment estimation. These improvements don't fundamentally change the `gather` method, since it would still be based on the same *containment* and *remove element* operations, but show how `gather` works as a more general method that can leverage characteristics from different building blocks and explore new or improved use cases.

`gather` is a new method for decomposing datasets into its components with application in biological sequencing data analysis (taxonomic profiling) that can scale to hundreds of thousands of reference datasets with computational resources requirements that are accessible to a large number of users when used in conjunction with *Scaled MinHash* sketches and efficient indices such as *LCA* and *MHBT*. It outperforms current methods in community-develop benchmarks, and opens the way for new methods that explore a top-down approach for profiling microbial communities, including further refinements that can resolve larger evolutionary distances and also speed up the method computationally.

XXX SBT and LCA indices

Scaled MinHash sketches are fundamentally a subset of the k-mer composition of a dataset, and so any of the techniques described in [**???**] are potential candidates for improving current indices or implementing new ones. The MHBT index can be improved by using more efficient representations for the internal nodes [**???**] and constructing the MHBT by clustering [**???**], and the LCA index can use more efficient storage of the list of signatures IDs by representing the list as colors [**???**]. The memory consumption of the LCA index can also be tackled by implementing it in external memory using memory-mapped files, letting the operating system cache and unload pages as needed.

Current indices are also single-threaded, and don't benefit from multicore systems. Both indices can be used in parallel by loading as read-only and sharing for multiple searches, but is is also possible to explore parallelization for single queries by partitioning the LCA and assigning each partition to a thread, as well as using a work-stealing thread pool for expanding the search frontier in the MHBT in parallel. In any case, the current implementations serve as a baseline for future scalability and can be used to guide optimization and avoid extraneous overhead and common failings of such projects [**???**].

Conclusion

Scaled MinHash sketches allow scaling analysis to thousands of datasets, but efficiently searching and sharing them can benefit from data structures that index and optimize these use cases. This chapter introduces an index abstraction that can be trivially implementing using a list of sketches (*Linear index*) and more advanced implementations based on inverted indices (*LCA index*) and hierarchical indices (*MHBT*) providing options for fast and memory-efficient operations, as well as making it easier to share and analyze collections of sketches. All these functionalities are implemented in

Taxonomy results are excellent.

Discuss vs LCA.

reference the LCA-has-limits/k-mers saturate paper

mix and match taxonomies is easy b/c we anchor to genomes.

Algorithm is simple, computational performance is great

Performant implementation in sourmasha Python API for data exploration and methods prototyping.

Database types work well

"online" approaches

Some limitations of gather and database types (equal results can be hard to detect efficiently with current SBT implementation)

The Linear index is appropriate for operations that must check every signature, since it doesn't have any indexing overhead. An example is building a distance matrix for comparing signatures all-against-all. Search operations greatly benefit from extra indexing structure. The MHBT index and k-mer aggregative methods in general are appropriate for searches with query thresholds, like searching for similarity or containment of a query in a collection of datasets. The LCA index and color aggregative methods are appropriate for querying which datasets contain a specific query k-mer.

As implemented in sourmash, the MHBT index is more memory efficient because the data can stay in external memory and only the tree structure for the index need to be loaded in main memory, and data for the datasets and internal nodes can be loaded and unloaded on demand. The LCA index must be loaded in main memory before it can be used, but once it is loaded it is faster, especially for operations that need to summarize k-mer assignments or require repeated searches.

Due to these characteristics, and if memory usage is not a concern, then the LCA index is the most appropriate choice since it is faster. The MHBT index is currently recommended for situations where memory is limited, such as with smaller scaled values ($s \leq 2000$) that increase the size of signatures, or when there are a large number (hundreds of thousands or more) of datasets to index.

Converting between indices

Both MHBT and LCA index can recover the original sketch collection. In the MHBT case, it outputs all the leaf nodes. In the LCA index, it reconstruct each sketch from the hash-to-dataset-ID mapping. This allows trade-offs between storage efficiency, distribution, updating and query performance.

Because both are able to return the original sketch collection, it is also possible to convert one index into the other.

Limitations and future directions

(From David Koslicki) Gotchas:

- Lack of sensitivity for small queries
- Potentially large sketch sizes

And a couple other that I've tentatively/mathematically observed:

- The variance of the estimate of C(A,B)=|AB| / |A| appears to also depend on |A|, which was somewhat surprising
- The "fixed k-size" problem (which might be able to be overcome with the prefix-lookup data structure, if one sacrifices some accuracy)

Scaled MinHash sketches are fundamentally a subset of the k-mer composition of a dataset, and so any of the techniques described in [**???**] are potential candidates for improving current indices or implementing new ones. The MHBT index can be improved by using more efficient representations for the internal nodes [**???**] and constructing the MHBT by clustering [**???**], and the LCA index can use more efficient storage of the list of signatures IDs by representing the list as colors [**???**]. The memory consumption of the LCA index can also be tackled by implementing it in external memory using memory-mapped files, letting the operating system cache and unload pages as needed.

Current indices are also single-threaded, and don't benefit from multicore systems. Both indices can be used in parallel by loading as read-only and sharing for multiple searches, but is is also possible to explore parallelization for single queries by partitioning the LCA and assigning each partition to a thread, as well as using a work-stealing thread pool for expanding the search frontier in the MHBT in parallel. In any case, the current implementations serve as a baseline for future scalability and can be used to guide optimization and avoid extraneous overhead and common failings of such projects [**???**].

Scaled MinHash sketches allow scaling analysis to thousands of datasets, but efficiently searching and sharing them can benefit from data structures that index and optimize these use cases. This chapter introduces an index abstraction that can be trivially implementing using a list of sketches (*Linear index*) and more advanced implementations based on inverted indices (*LCA index*) and hierarchical indices (*MHBT*) providing options for fast and memory-efficient operations, as well as making it easier to share and analyze collections of sketches. All these functionalities are implemented in `sourmash`, a software package exposing these features as a command-line program as well as a Python API for data exploration and methods prototyping.

These indices also serve as another set of building blocks for constructing more advanced methods for solving other relevant biological problems like taxonomic profiling, described in Chapter 3, and approaches for increasing the resilience and shareability of biological sequencing data, described in Chapter 5.

Scaled minhash has limitations vs regular minhash

virus, etc. (could go in first discussion section, but also deserves to be highlighted)

Conclusion

Scaled MinHash sketches are simple to implement and analyze, with consistent guarantees for the range of values and subsetting properties when applied to datasets. Containment and similarity operations between *Scaled MinHash* sketches avoid the need to access the original data or more limited representations that only allow membership query, and serve as a proxy for large scale comparisons between hundreds or thousands of datasets.

Small genomes require low scaled values in order to properly estimate containment and similarity, and exact k-mer matching is brittle when considering evolutionarily-diverged organisms. While some of these problems can be overcome in future work, *Scaled MinHash* sketches can serve as a prefilter

for more accurate and computationally expensive applications, allowing these methods to be used in larger scales by avoiding processing data that is unlikely to return usable results.

Scaled MinHash sketches are effective basic building blocks for creating a software ecosystem that allow practical applications, including taxonomic classification in metagenomes and large scale indexing and searching in public genomic databases.

References

1. **The Statistical Analysis of Compositional Data**
   J. Aitchison
   Journal of the Royal Statistical Society: Series B (Methodological) (1982-01) https://doi.org/gfs7gn
   DOI: 10.1111/j.2517-6161.1982.tb01195.x

2. **richarddurbin/modimizer**
   Richard Durbin
   (2020-09-03) https://github.com/richarddurbin/modimizer

3. **sourmash: a library for MinHash sketching of DNA**
   C. Titus Brown, Luiz Irber
   The Journal of Open Source Software (2016-09-14) https://doi.org/ghdrk5
   DOI: 10.21105/joss.00027

4. **Comparative metagenomic and rRNA microbial diversity characterization using archaeal and bacterial synthetic communities**
   Migun Shakya, Christopher Quince, James H. Campbell, Zamin K. Yang, Christopher W. Schadt, Mircea Podar
   Environmental Microbiology (2013-06) https://doi.org/f42ccr
   DOI: 10.1111/1462-2920.12086 · PMID: 23387867 · PMCID: PMC3665634