Published online DD MM YYYY *- , YYYY, Vol. xx, No. xx* **1–8**

kmindex and ORA: indexing and real-time user-friendly queries in terabytes-sized complex genomic datasets

Téo Lemane [1,*], Nolan Lezzoche [2], Julien Lecubin [3], Eric Pelletier [4,5], Magali Lescot [2,5], Rayan Chikhi [6], and Pierre Peterlongo [1,*]

[1]Univ. Rennes, Inria, CNRS, IRISA - UMR 6074, Rennes, F-35000 France
[2]Aix-Marseille Université, Université de Toulon, IRD, CNRS, Mediterranean Institute of Oceanography (MIO), UM 110, Marseille, France
[3]SIP, OSU PYTHEAS, Marseille, France
[4]Génomique Métabolique, Genoscope, Institut de Biologie François Jacob, CEA, CNRS, Univ. Evry, Université Paris-Saclay, Evry, France
[5]Research Federation for the Study of Global Ocean Systems Ecology and Evolution, FR2022/Tara Oceans GO-SEE, CNRS, Paris, France
[6]Institut Pasteur, Université Paris Cité, G5 Sequence Bioinformatics, Paris, F-75015, France

Received January 1, 2009; Revised February 1, 2009; Accepted March 1, 2009

ABSTRACT

Despite their wealth of biological information, public sequencing databases are largely underutilized. One cannot efficiently search for a sequence of interest in these immense resources. Sophisticated computational methods such as approximate membership query data structures allow searching for fixed-length words (k-mers) in large datasets. Yet they face scalability challenges when applied to thousands of complex sequencing experiments. In this context we propose `kmindex`, a new approach that uses inverted indexes based on Bloom filters. Thanks to its algorithmic choices and its fine-tuned implementation, `kmindex` offers the possibility to index thousands of highly complex metagenomes into an index that answers sequences queries in the tenth of a second. Index construction is one order of magnitude faster than previous approaches, and query time is two orders of magnitude faster. Based on Bloom filters, `kmindex` achieves negligible false positive rates, below 0.01% on average. Its average false positive rate is four orders of magnitude lower than existing approaches, for similar index sizes. It has been successfully used to index 1,393 complex marine seawater metagenome samples of raw sequences from the Tara Oceans project, demonstrating its effectiveness on large and complex datasets. This level of scaling was previously unattainable. Building on the `kmindex` results, we provide a public web server named "Ocean Read Atlas" (ORA) at https://ocean-read-atlas.mio.osupytheas.fr/ that can answer queries against the entire Tara Oceans dataset in real-time. `kmindex` is open-source software available at https://github.com/tlemane/kmindex.

INTRODUCTION

Public genomic datasets are growing at an exponential rate (7). Their content is without a doubt one of the most valuable treasures we have at our disposal for making groundbreaking discoveries in biological domains such as agronomy, ecology, and health (9). Unfortunately, despite being publicly available in repositories such as the Sequence Read Archive (13), these resources are rarely ever reused because there is no efficient way to query their data globally. Recent years have seen many developments in search engines that can perform queries on terabyte or petabyte-sized datasets. See (16) and (5), for a review of the existing data structures.

Genomic indexing tools all adopt a similar framework. Given a database composed of genomic samples, they create an index able to associate each k-mer (word of fixed length k) that occurs in at least one of the input samples to all the samples in which it occurs. At query time, the number of k-mers shared between a queried sequence q and all the indexed samples serves as a proxy for reporting significant hits. Despite the apparent simplicity of the approach, its design is far from simple. The difficulty comes from the volume of k-mers to be indexed, which is in the order of thousands of billions, across thousands of samples.

The computational challenge is immense, as it consists in first extracting all k-mers to be indexed from sequencing samples (filtered, to partly remove erroneous k-mers), and then building a large associative data structure. The first step is based on k-mer counting and has received much attention in the last few years, resulting in very efficient solutions such as `KMC3` (14). The second step can either use exact or approximate data structures. Exact data structures are used in tools such as `MetaGraph` (12), `BiFrost` (11),

*To whom correspondence should be addressed. Email: teo.lemane@inria.fr, pierre.peterlongo@inria.fr

© YYYY The Author(s)
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.0/uk/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

or ggcat (6), and are well-suited to assembled genomes, transcriptomes, or gut microbial reads. However, as underlined by our results, these exact solutions cannot scale to more complex datasets e.g. environmental metagenomics reads, in particular as we show next for the sequencing of seawater.

In order to scale to large and complex datasets, several methods are based on inexact data structures called AMQs for "Approximate Membership Queries", the Bloom filter (4) being one of the most famous. They typically have false positives, i.e. some k-mers are wrongly associated to some samples, and such k-mers may not even exist anywhere in the indexed data. Yet Bloom filters offer the advantage to index billions of k-mers from a sample using only a few dozen of gigabytes of space. This was exploited by tools as COBS (3), SBT (20), latter improved by HowDeSBT (10), and more recently by MetaProFi (21). However, when scaling to large and complex datasets, these tools have significant limitations such as prohibitive disk usage, memory usage, computation time (either indexing time or query time), and high false positive rate. Usually, tools suffer from a combination of two or more of the aforementioned limitations.

Given the apparent challenges in overcoming simultaneously all these limitations, one might conclude that no efficient approximate data indexing strategy could ever be designed. Yet in this work we present kmindex, a new tool that performs indexing in significantly less resources than previous approaches. kmindex can index thousands of highly complex sequencing datasets, including environmental metagenomes such as seawater, which contain thousands of billions of distinct k-mers. kmindex leverages the highly efficient k-mer counter and k-mer filtration processes of our previous tool kmtricks (15). kmindex then indexes k-mers using Bloom filters. Capitalizing on recent results for lowering the false positive rate of approximate data structures (findere (18)), kmindex provides results with negligible false positive (FP) rates, approximately three orders of magnitude smaller than those obtained by coexisting tools with no impact on disk or computation time. On our tests, the final index size on disk is roughly 10 to 13% of the input gzipped fastq size with FP rates below 0.01%. Terabytes of complex metagenomic raw data can be indexed in a matter of hours. kmindex does not need to load the index in RAM to answer queries. Yet, when the indexes are stored on local SSD hard drives, query execution is instant and millions of short sequences such as reads can be queried in a minute. Also, kmindex provides a way to dynamically append new datasets to an existing index. This is critical for the useability of an index, even though most other indexing algorithms do not support it.

We developed an API, a command line interface, and a web server that can all be invoked to perform queries. Thus kmindex can be used to build a local private genomic search engine preserving data confidentiality. It can also be used as a public server, easily interacting with visualization tools as shown in this manuscript.

The technical novelties presented in this paper focus on the query model, which has been carefully designed to maximize query throughput by minimizing both cache misses and I/O operations. As a result, kmindex can perform well across a range of contexts: real-time queries of short individual sequences, and large queries consisting of millions of sequences, as presented in our benchmark results.

Results presented in this paper show that kmindex outperforms other methods dedicated to raw data indexing in terms of resources by using less memory and disk, by being several orders of magnitude faster to build and query, and by being more scalable. No other method could index the largest dataset tested in this work, with the exception of MetaProFi which requires 10x more time for indexing, 100x more time for queries, and 50x more disk space for achieving the same result quality as kmindex.

To showcase the features of kmindex on a dataset of high biological interest, we introduce a web server named "Ocean Read Atlas" (ORA) available at this URL: https://ocean-read-atlas.mio.osupytheas.fr/. ORA allows query one or several sequences across all of the *Tara* Oceans metagenomic raw datasets (22). Query results can be downloaded and an interface enables the visualization of the results on a geographic map. Furthermore, because each indexed sample is linked to a variety of environmental variables collected during the circumnavigation campaign (56 distinct measures, including pH, temperature, salinity, and so on), the interface allows users to see how the queries are related to those variables using bubble plots. In addition to demonstrating the capabilities of kmindex, ORA is the first web server capable of performing instant searches on such a large and complex dataset. It provides new perspectives on the deep exploitation of Tara oceans resources.

RESULTS

kmindex Features

kmindex, has the following features:

1. For each input dataset, k-mers are counted and filtered based on their abundance and on their co-occurrences in the various input datasets.

2. Bloom filter is built for each input dataset, storing the presence/absence of indexed k-mers. As explained in the Method section, in practice Bloom filters are *inverted*, avoiding cache misses at query time.

3. At query time, k-mers from queried sequences are grouped into batches, and Bloom filters are queried to determine the presence or absence of each k-mer in each input dataset.

4. A single command line enables the indexation of a collection of raw reads. When several distinct collections are indexed using kmindex, the resulting indexes can be registered into a single meta-index allowing users to easily query multiple indexes. This also offers the opportunity to dynamically update the index of a data collection. Moreover, where applicable (i.e. same k-mer size and Bloom filter size), compatible indexes can be merged, hence saving query time. Being able to add novel samples to an existing index is a mandatory feature when dealing with dynamic datasets, rarely offered by existing tools.

5. At query time, the kmindex design enables it to be used as a web server, using a command line interface, or via a Python API.

6. Depending on the user request, for each indexed sample, the answer to a query sequence can be either its ratio of k-mers shared or, more precisely, the positions where these shared k-mers occur. In the same spirit, when more than one sequence is queried, the answer for each indexed sample can be either a single average number of shared k-mers, or a value per input queried sequence.

7. Importantly, kmindex is highly documented and easy to use and install, supporting various package managers.

Comparative results indexing 50 metagenomic seawater samples

In this section, we evaluate the performance of kmindex together with state-of-the-art k-mer indexers MetaGraph (12), MetaProFi (21), and PAC (17).

Note that, even though MetaGraph can index k-mers, it is also intended to perform more complex tasks such as assigning additional information to each indexed k-mer or even for sequence-to-graph alignment. In the same spirit, MetaProFi is not limited to nucleotide indexing but can also index and query sequences over the amino-acid alphabet. Our method is restricted to nucleotide k-mer queries, which means the benchmarks we present are only relevant in this specific context.

Here we index raw metagenomic seawater sequencing data from 50 *Tara* Oceans samples, composed of 1.4TB of gzipped fastq files. Each sample is represented by several distinct fastq.gz read files (on average 2.5 files per sample). This dataset contains approximately 1,420 billion k-mers. Among them, approximately 394 billion are distinct, and 132 billion occur twice or more.

Table 1 shows that kmindex is the only tool able to index TB-sized datasets in less than three hours and with low memory needs. These results also show that kmindex is the only tool able to perform millions of queries on such a large database in a matter of dozens of seconds, with less than a GB of RAM.

All used command lines and links to publicly available datasets are provided in a companion GitHub website https://github.com/pierrepeterlongo/kmindex_benchmarks.

In situations involving large indexes and queries, various factors such as I/O operations or caching can impact performances. To account for these effects, we performed the benchmarks from a user's perspective. As a result, all measurements are obtained using the command line tools with particular attention to caching effects. The reported values include input parsing, query execution, and output writing. The results presented in this section demonstrate the performance in a cold cache context (the most likely and also the least favorable). Results for other scenarios, e.g. warm cache, are available in Supplementary Materials.

Executions were performed on the GenOuest platform on a node with 64 cores (128 threads) Xeon 2.2 GHz (L1 = 48KB, L2 = 1.25MB, L3 = 48MB shared) with 900 GB of memory. All computations are performed on an *xfs* filesystem allowing 1052MB/s sequential reads, 473MB/s sequential writes, and 908MB/s random reads (throughput measurements are obtained using fio (2)). All tools were parameterized to use 32 threads.

Tested tool versions: kmindex version: 0.4.0, MetaGraph version: 0.3.6, PAC commit cee1b5c (as used in the original PAC paper), MetaProFi version 0.6.0 (associated to KMC3, version 3.2.2).

Index construction results

Indexes were constructed using $k=28$. In the case of kmindex, which uses the findere approach, the 28-mers are emulated using s-mers of size $s=23$. Indexing results are presented in Table 1.

We indicate the PAC results in this table even though we were unable to collect coherent results at query time. The replies to the queries were all empty, resulting in a 100% false negative rate.

MetaProFi cannot count and filter k-mers. This imposed the preliminary usage of KMC3 for removing k-mers with abundance one, considered as erroneous. This step is mandatory as 67% of the k-mers are unique in this dataset. Indexing them would more than double the final index size for achieving the same false positive rate, and would induce sensible noise at query time. However, we consider that the KMC3 outputs are temporary and are thus not considered for computing the overall output size on disk.

As MetaGraph needed more than the available amount of RAM (900GB), it could not finish the index creation. This highlights the fact that memory usage is a bottleneck for using MetaGraph on such complex data.

Overall, during this indexing step, kmindex exhibits better performance on all considered criteria. Importantly, compared to MetaGraph and to MetaProFi, kmindex is at least one order of magnitude faster. Finally, as PAC faces a bug at query time, apart from kmindex the only tool able to index such big and complex metagenomic datasets is MetaProFi. However, and as a side note, we recall that kmindex includes the findere approach that offers a drastic reduction of the false-positive rate at query time (see next section for practical results). For obtaining false-positive rates similar to those offered by kmindex, the final index size of MetaProFi, which does not use findere, would be ≈35 times larger, thus requiring nearly 8TB only for storing the index. See (18) for details about the findere approach.

The two next sections give results about the performances at query time, either in terms of false positive rates or in terms of resources needed. Unfortunately, as mentioned earlier, PAC faces a bug at query time. Thus, the quality of its results can not be assessed. However, as we believe that this issue could be fixed by PAC authors, we still indicated its results in terms of computational needs. Also, we were unable to test queries using MetaGraph because the indexing phase had not been completed.

Query results: false positive rate

kmindex, PAC, and MetaProFi use "approximate membership query" (AMQ) data structures for indexing k-mers. This offers higher scalability at the price of the existence of False Positive (FP) calls at query time. In order to test the FP rate, we generated a random sequence (25%

chance of each nucleotide at each position, ≈50% GC) of size 10000. We used it for querying the index of the 50 *Tara* Oceans samples, successively querying the 9973 (10000-28+1) overlapping 28-mers of the query sequence. Note that we do not have a way to assess if each queried random k-mer occurs in the indexed set or not. Thus it may appear by chance (with a probability 2×10^{-6}) that such a random k-mer indeed occurs in the set. Hence the False Positive rate reported in this section is an upper bound. This detail does not impact the conclusions offered by the results.

	Average	Median	Min	Max
`MetaProFi`	11.18	9.92	6.93	21.55
`kmindex`	<0.01	0	0	0.18

Table 2. False positive rates (%). Indexed: 50 *Tara* Oceans samples. Queried: k-mers ($k=28$) from a random sequence of size 10k nucleotides. For each sample, each tool provides the ratio of the queried k-mers reported as indexed. `kmindex` indexed words of length 23 and queried 28-mers using the `findere` approach. `MetaProFi` indexed k-mers of length 28.

Table 2 shows that `kmindex` provides query answers with negligible false positive calls. In `kmindex` results, among the 50 answers (one per indexed sample) the average FP rate is 0.006% and its highest false positive rate is $\approx 0.18\%$, 18 k-mers among the 9973 queried 28-mers being FPs. With the same size of Bloom filters (and so roughly the same order of magnitude size of the final index), `MetaProFi` cannot achieve similar quality results. Over the 50 answers, its average FP rate is 11.18% and is three orders of magnitude higher than the `kmindex` FP rate. This renders the `MetaProFi` downstream analyses difficult to exploit and trust.

Query results: time and memory performance

In this section, we tested the scalability performances while querying the index with a growing number of short queries. For creating the queries, among the 50 *Tara* Oceans samples, we selected randomly and uniformly from one read to ten million reads that were then used as query sequences.

Results are presented in Table 3. Despite providing only negative answers, we indicate the `PAC` performances. These results show that `kmindex` outperforms `MetaProFi` and `PAC`, both in terms of memory resources and computation time, again with several orders of magnitude.

In particular, these results demonstrate that `kmindex`, which can query millions of reads in a matter of four to five minutes, can be used to compare separately each read from a full read set to all data indexed in large projects, such as *Tara* Oceans. This opens the doors to novel usages for analyzing raw read sets as queries, such as determining the diversity of a complex queried read set or for clustering reads based on their similarity to each indexed dataset.

Notably, as shown in Supplementary Materials, `kmindex` disposes from a so-called "fast-mode", that uses more RAM (depending on the memory pressure) and achieves even faster queries. For instance, the 10 million reads can be queried in 1m33 instead of 4m21, using ≈194G of RAM instead of ≈47G.

Furthermore, these results demonstrate that `kmindex` can query instantaneously (less than one-tenth of a second) a sequence over an index as the one indexing large and complex metagenomic seawater datasets. This offers the possibility to provide public servers querying in real-time such large datasets, as presented in the following section.

Ocean Gene Atlas web server example

Thanks to these novel possibilities offered by `kmindex`, we built and made available a public web interface able to perform queries on a dataset composed of 1,393 samples (distinct locations and distinct fraction sizes) of the *Tara* Oceans project (22). These samples are divided into six distinct groups, determined by the size fraction of the sequenced species. Based on this clustering we built six distinct indexes (all with the same parameters). At query time, as all the six indexes are registered in a unique meta-index, the whole set of samples is queried. A description of the dataset is available in Table 4. The size of the final uncompressed index is approximately 13% of the size of the raw fastq.gz files, which is 36.7 TB.

This interface is built upon the so-called `kmindex-server` module. The user can use one or

	Step	Wall clock time	Max Memory (GB)	Max temp. disk (GB)	Output size on disk (GB)
`MetaGraph`	Build	23h30	459	2634	442
	Primary contigs	22h56	727	215	652
	Graph Primary$^{\alpha}$	NA	>900	NA	NA
	Overall	NA (>60h)	NA	NA	NA
`MetaProFi`	`KMC3` count	3h44	278	1019	1019
	`KMC3` dump	18h11	0	5684	5684
	`MetaProFi`	8h20	232	226	226
	Overall	30h15	278	5684	226
`PAC`	All	15h59	190	191 + 1415$^{\beta}$	184
`kmindex`	All	**2h56**	**107**	**878**	**164**

$^{\alpha}$ The "Graph Primary" step from `MetaGraph` did not finish because it requires more than 900GB of RAM.

$^{\beta}$ in order to consider multiple files per sample, the original input file has to be concatenated and so doubled using `PAC`.

Table 1. Comparing `kmindex` indexing performances to those from `MetaGraph`, `MetaProFi` and `PAC`. The indexed dataset is composed of 50 *Tara* Oceans metagenomes datasets (total size 1.4TB). "Wall clock time" corresponds to the *user* time. "*Max temp. disk*" indicates the maximal disk used at runtime, this can be a temporary usage as for `kmindex`. "*Output size on disk*" indicates the size of the created files (the final indexes but also the size of intermediate results in the case of `MetaGraph` and `MetaProFi`. `kmindex` and `PAC` can be run with a single command line, here resumed in the "*All*" step.

No. queries	1	10	100	1,000	10,000	100,000	1,000,000	10,000,000
`MetaProFi` Time	0m12	0m15	1m33	2m57	3m02	3m37	11m56	1h29m12
`MetaProFi` Memory peak (GB)	0.3	0.3	0.3	0.32	0.44	2.25	21	203
`PAC` Time	5m30	16m48	34m31	38m58	36m06	36m03	39m54	36m35
`PAC` RAM (GB)	89	90	90	90	90	90	92	104
`kmindex` Time	**0s06**	**0s23**	**1s24**	**4s71**	**19s78**	**53s72**	**1m13s**	**4m21s**
`kmindex` Memory peak (GB)	**0.005**	**0.005**	**0.006**	**0.01**	**0.05**	**0.45**	**4.9**	**46.7**

Table 3. Query time performance of the indexes on the 50 Tara Ocean samples. The queries are composed of random reads uniformly sampled from the 50 Tara Oceans datasets. Executions were performed on a cold cache (full results are proposed in the Supplementary Materials). `PAC` uses partitioning and only loads into memory the partitions that are relevant for the queries resolution. The query times seem to point out that the k-mer diversity results in the loading of almost all partitions once the number of queries reaches 100. `kmindex` and `MetaProFi` rely on memory-mapped files, thus the memory usages do not reflect the minimum amount of memory required for the query but also include page caching.

Fraction	No. samples	k-mers/sample
Integrated ($0.8-2000\mu m$)	193	14.5e9
Meso ($180-2000\mu m$)	208	13.2e9
Micro ($20-280\mu m$)	195	15e9
Nano ($3-20\mu m$)	213	16.1e9
Pico ($0.2-5\mu m$)	425	11.1e9
Virus ($<0.2\mu m$)	159	2.3e8

Table 4. Description of the indexed dataset organized by size fraction. The column "k-mers/sample" is the average number of distinct k-mers per sample.

more sequences to query the index and determine the similarity of each sequence to the 1,393 indexed samples. A world map depicts the resulting biogeography, as well as the environmental parameters associated with the sequences. Finally, results can be downloaded in the form of tables or under several image formats.

For reasons of robustness and continuity of service, the index is deployed on a networked and redundant filesystem with lower performance than the benchmark environment. As a result, the query times are higher although suitable for this type of service.

This server, named "Ocean Read Atlas" (`ORA`), whose representation is provided in Figure 1, extends the "Ocean Gene Atlas" server (OGA) (23, 24) that is able to query assembled genes from *Tara* Oceans (22) and Malaspina (1).

Details about indexed read sets, and more information about the server architecture and setup are provided in Supplementary Materials

We believe this server will be of great importance to the *Tara* Oceans consortium as a whole, and more broadly to anybody interested in marine genetic data.

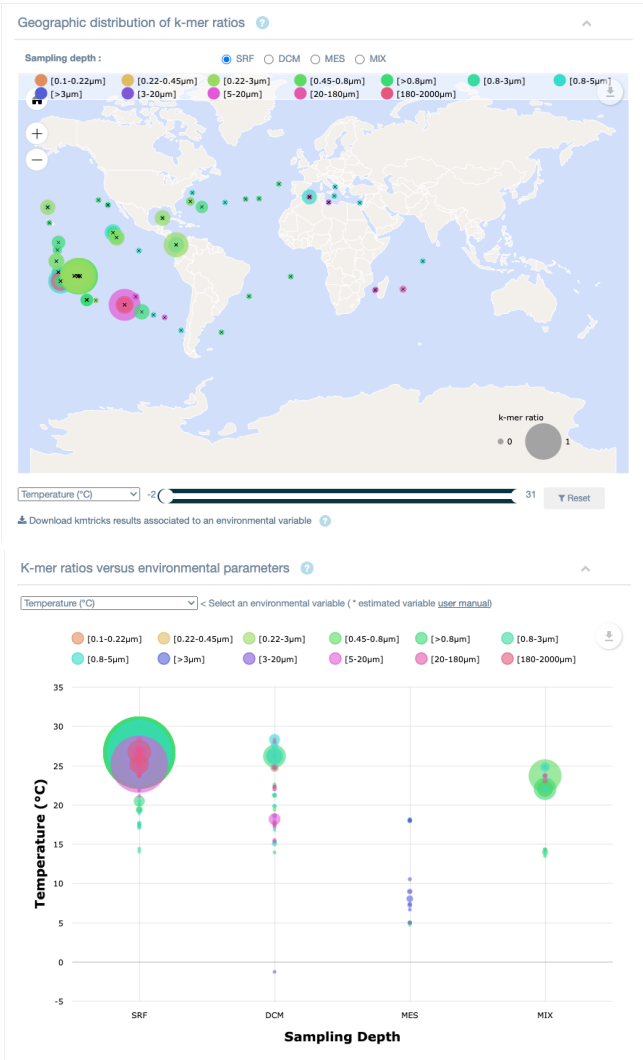

Figure 1. Capitalizing on the `kmindex` features, we propose "Ocean Read Atlas", a web server able to query one or several sequences provided by the user against the whole *Tara* Oceans metagenomic dataset in a matter of few seconds, and online. Left: the result interface in which the biogeography distribution of the sequence similarity ratio is shown among all data samples whose answers are higher than a threshold of 0.01. The size of the point depicts the similarity of the queried sequences. Right: in a second frame, a bubble plot represents these results according to the environmental variables. On the website, users can select a subset of selected samples according to a particular environmental variable (e.g. a particular size fraction or depth) using a slider below the world map.

METHODS

Overview

Conceptually, the presence of each indexed k-mer is stored in one Bloom filter per input read set. At query time, the existence of each k-mer of the query is provided by the Bloom filters. The k-mer counting and filtering processes used for creating the Bloom filters are adapted from `kmtricks` modules (15) and are not recalled in this manuscript.

The main `kmindex` conceptual novelties are how Bloom filters are represented and organized in memory, and how queries are performed, as described in the next sections.

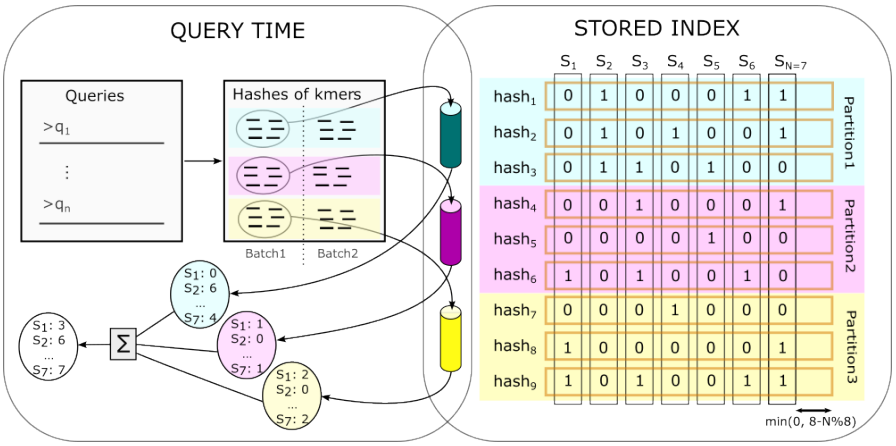

Figure 2. `kmindex`: an overview of the data structure and the query process. **Right: Data structure showing the stored index on disk**. Vertical black rectangles represent the N Bloom filters (one per input sample). The orange horizontal rectangles represent the actual data structure saved on disk, storing consecutively the 0/1 values of distinct Bloom filters for the same hash value. The three colored horizontal rectangles represent three partitions, each saved in a distinct file. **Left: Query example**. Hashes of k-mers are symbolized by small horizontal lines, divided into two batches, and grouped by partitions. Each group is sorted. Each cylinder represents the streaming of a set of hashed k-mers, querying lines of Bloom filters mapped into memory. For simplicity, the image shows only queries from hashed k-mers from one of the two represented batches. In practice, for each partition, all batches are queried. For each sample, the results from each partition and each batch are finally summed up (as symbolized by the "Σ" symbol in this figure). Also for simplicity, this figure does not represent the use case in which the distribution of hits along the query sequences is reported as a binary vector.

Importantly, k-mers are processed according to their sorted hash values during indexing and querying processes. As so, cache misses are significantly minimized when feeding or reading a Bloom filter, hence loading the entire indexed collection in memory is not needed. This results in orders of magnitude faster indexing, and query, and lower memory usage than other k-mer indexing tools.

In addition to these features, the query process employs the `findere` (18) approach, which allows for an embarrassingly low Bloom filter false positive rate, slightly faster query execution, and no impact on disk or RAM usage.

Index construction

The index construction is entirely delegated to `kmtricks`, which allows partitioned construction of one-hash Bloom filter matrices. Each sub-matrix indexes a subset of k-mers matching a specific set of minimizers. As represented in Figure 2 (right), the resulting index consists of P distinct matrices (with P being the number of partitions, equal to 3 in the figure). In order to save indexing and query time, the index is "inverted": given a k-mer, the N bits indicating its presence/absence in the N indexed datasets are consecutive in the index. This allows for fast queries across numerous datasets. Hence, in practice, in a matrix, each row is a bit vector representing the presence or absence of a hash value in each indexed sample. Note that the rows are not packed to save construction and query time. This results in the fact that each row is composed of $\left\lceil \frac{N}{8} \right\rceil \times 8$ bits. Doing so, $\min(0, 8 - N\%8)$ bits are unused for each row, as represented by a double arrow in Figure 2. This is up to 7 bits per row. These few lost bits may appear as a drawback, but this is negligible in regard to the N value that is meant to be in the order of a few hundred or thousands, and, importantly, this enables us to efficiently append novel indexed samples to an existing index.

By default, the resulting index is not compressed. Although requiring more space, this ensures optimal access time (both for writing and for reading), and it offers the possibility to dynamically add new datasets to an index.

Index query

The query process is also sketched in Figure 2. Batch processing is used for queries. This allows maximum throughput while maintaining control over memory usage. The user can specify the batch size and the maximum number of parallel batches accordingly to the system's capabilities.

The resolution of a batch proceeds as follows.

1. **Bucketing**. The index is organized by partition, each corresponding to a set of minimizers. The first step consists in splitting query sequences into k-mers, which are then hashed and inserted into the right partition according to their minimizers. Each k-mer partition of the batch can then be solved by querying the corresponding index partition.

2. **Sorting**. Each partition is sorted to enable its resolution in a single sequential pass on the corresponding index partition.

3. **k-mer level resolution**. Querying a specific k-mer consists in fetching the row that corresponds to its hash value in the index to retrieve the bit vector corresponding to its presence or absence in each sample. For each query, the response vectors are aggregated by summation, resulting in an integer vector that represents the number of positive hits in each indexed sample.

   Obtaining the response vector for each k-mer is the current bottleneck of our method because of I/O operations. For this reason, instead of loading the index into memory, index partitions are read through memory-mapped files. This allows reading only the parts of the index that are relevant to the batch resolution, which is particularly beneficial in the case of small queries.

The memory-mapped files can be managed in two different ways: **Normal mode**: Each batch manages its own mappings of index partitions. The mapping of a partition is closed as soon as all k-mer belonging to the partition is resolved. The cached pages are then marked as available for eviction, resulting in lower memory usage (see Supplementary Materials). **Fast mode**: All batches share the same mappings. This way, a larger number of pages are kept in the cache when conditions are favorable, i.e. without memory pressure, avoiding possible new I/O operations when solving the remaining batches. The memory usage may therefore seem high due to important page caching, up to the size of the index in the context of large query sets. Note that both modes require the same minimum amount of memory, the other part of the memory usage corresponds only to page caching which is automatically managed by the kernel. In other words, under memory pressure, both modes show the same memory usage.

4. **Sequence level resolution**. Finally, a result file is generated in either json or tsv format depending on the user's choice. Query results are filtrated based on the threshold specified by the user. The user can also request the distribution of hits along the query sequences, represented as a binary vector.

DISCUSSION

Additional features

One of the additional features of `kmindex` is that it has no limit regarding the k value used during indexing. This provides greater flexibility and customization for users.

Another key feature of `kmindex` is its index merging and index registering capabilities. Users can add new samples to an existing index in two different ways. A novel and independent index can be *registered* with a previous one. In this case, at query time, each registered index is queried independently. This also gives the user the option to query only a subset of the registered indexes. Alternatively, users can extend an existing index, and the parameters of the previous index (such as the *ad-hoc* hash function or the Bloom filter sizes) are automatically reused. This second choice is less flexible, but it provides better performances at query time as fewer data structures are interrogated (see results presented in Supplementary Materials). The first choice is well adapted when indexing sets of samples with distinct characteristics such as their size of variability.

Convenient k-mer filtration is also included as a feature in `kmindex`. This feature is based on the `kmtricks` results (15). It enables the filtration of erroneous k-mers, not only relying on their abundance in a dataset but also thanks to their co-abundances in all indexed datasets. To the best of our knowledge, this feature cannot be proposed by any other indexing tool.

Query results with `kmindex` can be shown with various degrees of precision. For each indexed sample and given a set of queried sequences, users can obtain the average similarity of all queried sequences or they can obtain a similarity value per queried sequence. Finally, `kmindex` can provide a result format showing per position of each queried sequence if the k-mer occurring at this position is indexed or not. This enables highlighting some regions of interest among the queried sequences.

The results presented in this manuscript involve metagenomic datasets. However, it is worth mentioning that `kmindex` is technically adapted for indexing any kind of dataset represented using the nucleic alphabet, including barcodes and metatranscriptomics.

Finally, `kmindex` is designed to be flexible and accessible to users. Once indexes are built thanks to a command-line interface (CLI), queries can be performed also via a CLI, via an API, or as an HTTP server. This gives users multiple options for interacting with the software, making it easier to integrate into their workflows.

Main limitations and future work

As mentioned earlier, `kmindex` does not compress the created Bloom filters, this ensures, among others, extremely fast queries. However, this means that the size of the index grows linearly with the number of samples it contains. This feature is limiting in terms of storage capacity, especially for datasets containing orders of magnitude more complex samples than tested here (say, hundreds of thousands).

A limitation of `kmindex` is that its current scope does not include association with third-party information such as variants or genes. However, similar to what is proposed in the context of estimating transcript expression by the `Needle` tool (8), in the near future, `kmindex` will integrate the `fimpera` approach (19) to estimate the abundance of each k-mer in each indexed sample, using counting Bloom filters.

CONCLUSION

We propose `kmindex`, a tool for creating k-mer indexes from terabyte-sized raw sequencing datasets. It is the only tool able to index highly complex data such as thousands of seawater metagenomic samples, and it is the only tool able to provide instant query answers, with a non-null but negligible false positive rate, below 0.01% in our tests.

Practical features offered by `kmindex` also include 1/ a user-friendly interface for creating indexes through a unique and simple command line, 2/ the `kmindex` tool is easy to install *via* Conda, docker, nix, or portable binary releases in addition to the installation from sources. The `kmindex` repository is highly documented.

We believe that by its performance and its usage simplicity, `kmindex` makes indexing k-mers from large and complex genomic projects practically possible for the first time. Future work includes the compression of the provided indexes, the ability to store each indexed k-mer with its abundance in each dataset, as well as with third-party pieces of information such as known variants, or annotations.

The optimized performance of `kmindex` opens up a new channel for leveraging genetic data, removing the obstacles that often isolate studies from each other. The Ocean Read Atlas that we propose is a forerunner in this significant breakthrough. Furthermore, this instance provides a remarkable new tool to fully utilize the wealth of data provided by the *Tara* Oceans project.

DATA AVAILABILITY

A list of publicly available data used in this work is proposed in the https://github.com/pierrepeterlongo/kmindex_benchmarks repository.

FUNDING

The work was funded by ANR SeqDigger (ANR-19-CE45-0008), the IPL Inria Neuromarkers, and received some support from the French government under the France 2030 investment plan, as part of the Initiative d'Excellence d'Aix-Marseille Université - A*MIDEX - Institute of Ocean Sciences (AMX-19-IET-016). Inria has taken up the publication charges for this paper in the context of its open science policy. This work is part of the ALPACA project that has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 956229.

ACKNOWLEDGEMENTS

We acknowledge the GenOuest bioinformatics core facility (https://www.genouest.org) and the TGCC (https://www-hpc.cea.fr/index-en.html) for providing the computing infrastructure, as well as France Génomique for funding of the TGCC computing resources used to process data used in this article. The authors thank Jean-Marc Aury for his help regarding the usage of the *Tara* Oceans data sets. *Tara* Oceans (which includes both the *Tara* Oceans and *Tara* Oceans Polar Circle expeditions) would not exist without the leadership of the *Tara* Ocean Foundation and the continuous support of *Tara* Oceans consortium members. The authors also thank Kahles Andre and Mustafa Harun for their help regarding the usage of `MetaGraph`, and Camille Marchet and Antoine Limasset for their support using `PAC`. The web server is hosted by the OSU Pythéas cluster with the help of Cyrille Blanpain and SIP members. Adrien Malgoyre from SIP is thanked for the development of the OSU Pythéas GitLab.

REFERENCES

1. S.G Acinas, P Sánchez, G Salazar, F.M. Cornejo-Castillo, M. Sebastián, R. Logares, M Royo-Llonch, L Paoli, S. Sunagawa, P. Hingamp, and et al. Deep ocean metagenomes provide insight into the metabolic architecture of bathypelagic microbial communities. *Communications Biology*, 4(604):1–15, 2022.
2. Jens Axboe. Flexible I/O Tester, 2022.
3. Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. Cobs: a compact bit-sliced signature index. In *String Processing and Information Retrieval: 26th International Symposium, SPIRE 2019, Segovia, Spain, October 7–9, 2019, Proceedings 26*, pages 285–303. Springer, 2019.
4. Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
5. Rayan Chikhi, Jan Holub, and Paul Medvedev. Data structures to represent a set of k-long dna sequences. *ACM Computing Surveys (CSUR)*, 54(1):1–22, 2021.
6. Andrea Cracco and Alexandru I Tomescu. Extremely-fast construction and querying of compacted and colored de bruijn graphs with ggcat. *bioRxiv*, pages 2022–10, 2022.
7. Carla Cummins, Alisha Ahamed, Raheela Aslam, Josephine Burgin, Rajkumar Devraj, Ossama Edbali, Dipayan Gupta, Peter W Harrison, Muhammad Haseeb, Sam Holt, et al. The european nucleotide archive in 2021. *Nucleic Acids Research*, 50(D1):D106–D110, 2022.
8. Mitra Darvish, Enrico Seiler, Svenja Mehringer, René Rahn, and Knut Reinert. Needle: a fast and space-efficient prefilter for estimating the quantification of very large collections of expression experiments. *Bioinformatics*, 38(17):4100–4108, 2022.
9. Robert C Edgar, Jeff Taylor, Victor Lin, Tomer Altman, Pierre Barbera, Dmitry Meleshko, Dan Lohr, Gherman Novakovsky, Benjamin Buchfink, Basem Al-Shayeb, et al. Petabase-scale sequence alignment catalyses viral discovery. *Nature*, 602(7895):142–147, 2022.
10. Robert S Harris and Paul Medvedev. Improved representation of sequence bloom trees. *Bioinformatics*, 36(3):721–727, 2020.
11. Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. *Genome biology*, 21(1):1–20, 2020.
12. Mikhail Karasikov, Harun Mustafa, Daniel Danciu, Marc Zimmermann, Christopher Barber, Gunnar Rätsch, and André Kahles. Metagraph: Indexing and analysing nucleotide archives at petabase-scale. *BioRxiv*, 2020.
13. Kenneth Katz, Oleg Shutov, Richard Lapoint, Michael Kimelman, J Rodney Brister, and Christopher O'Sullivan. The sequence read archive: a decade more of explosive growth. *Nucleic acids research*, 50(D1):D387–D390, 2022.
14. Marek Kokot, Maciej Długosz, and Sebastian Deorowicz. Kmc 3: counting and manipulating k-mer statistics. *Bioinformatics*, 33(17):2759–2761, 2017.
15. Téo Lemane, Paul Medvedev, Rayan Chikhi, and Pierre Peterlongo. kmtricks: Efficient and flexible construction of bloom filters for large sequencing data collections. *Bioinformatics Advances*, 2022.
16. Camille Marchet, Christina Boucher, Simon J Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, 2021.
17. Camille Marchet and Antoine Limasset. Scalable sequence database search using Partitioned Aggregated Bloom Comb-Trees. In *Recomb 2022- 26th Annual International Conference on Research in Computational Molecular Biology*, La jolla, United States, May 2022.
18. Lucas Robidou and Pierre Peterlongo. findere: fast and precise approximate membership query. In *International Symposium on String Processing and Information Retrieval*, pages 151–163. Springer, 2021.
19. Lucas Robidou and Pierre Peterlongo. fimpera: drastic improvement of approximate membership query data-structures with counts. *Bioinformatics*, 39(5):btad305, 2023.
20. 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, 2018.
21. Sanjay K Srikakulam, Sebastian Keller, Fawaz Dabbaghie, Robert Bals, and Olga V Kalinina. Metaprofi: an ultrafast chunked bloom filter for storing and querying protein and nucleotide sequence data for accurate identification of functionally relevant genetic variants. *Bioinformatics*, 39(3):btad101, 2023.
22. Shinichi Sunagawa, Silvia G Acinas, Peer Bork, Chris Bowler, Tara Oceans Coordinators, Damien Eveillard, Gabriel Gorsky, Lionel Guidi, Daniele Iudicone, Eric Karsenti, Fabien Lombard, Hiroyuki Ogata, Stephane Pesant, Matthew B Sullivan, Patrick Wincker, and Colomban de Vargas. Tara oceans: towards global ocean ecosystems biology. *Nat Rev Microbiol*, 18(8):428–445, 2020.
23. Caroline Vernette, Julien Lecubin, Pablo Sánchez, Shinichi Sunagawa, Tom O Delmont, Silvia G Acinas, Eric Pelletier, Pascal Hingamp, and Magali Lescot. The ocean gene atlas v2. 0: online exploration of the biogeography and phylogeny of plankton genes. *Nucleic Acids Research*, 50(W1):W516–W526, 2022.
24. Emilie Villar, Thomas Vannier, Caroline Vernette, Magali Lescot, Miguelangel Cuenca, Aurélien Alexandre, Paul Bachelerie, Thomas Rosnet, Eric Pelletier, Shinichi Sunagawa, et al. The ocean gene atlas: exploring the biogeography of plankton genes online. *Nucleic Acids Research*, 46(W1):W289–W295, 2018.