nature computational science

Brief Communication

https://doi.org/10.1038/s43588-024-00596-6

Indexing and real-time user-friendly queries in terabyte-sized complex genomic datasets with kmindex and ORA

In the format provided by the authors and unedited

Supplementary Information

1 Benchmarks

1.1 Index construction

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 $s = 23$.

Except for `kmindex`, only two tools, `MetaProFi` and `COBS`, were able to finish building the index and perform queries. Supplementary Table 1 shows results for other tested tools that did not finish to build the index or for which the queries were not possible.

Recall that count and filter k-mers are mandatory steps for all k-mer-based indexing tools. For example, 67% of the k-mers in this dataset are unique. Indexing them would more than double the index size and produce imprecise query replies. However, these time-consuming steps are not included in `MetaProFi` and `COBS`. This is why, for these two tools, and before constructing the indexes, we used `KMC3` to count k-mers and remove those seen only once and thus considered as erroneous.

Supplementary Table 2 shows that `kmindex` is ≈ 10 times faster and uses ≈ 3 times less memory and ≈ 6.5 less disk than the only other tools able to finish the index building with 900 GB of RAM and to perform a query in less than 12h. These results highlight the fact that time and memory usage are a bottleneck for tools that require to build a compacted version of the input data. Overall, for building the index, `kmindex` exhibits better performances on all considered criteria.

	Wall clock time	Max Memory (GB)	Max temp. disk (GB)	Comment
`PAC` [7]	14h59	190	$191 + 1415^{\beta}$	Empty result to queries
`ggcat` [3]	12h59	69	$2472 + 1415^{\beta}$	⚠Max No open files: 211,648. Query of one sequence killed after 12h.
`Themisto` [1]	9h14 (killed)	> 900	4261	Killed because of RAM usage
`HIBF` [8]	16h53 (killed)	> 900	0.3	Killed because of RAM usage
`MetaGraph` [6]	50h53 (killed)	> 900	2144	Killed because of RAM usage
`Bifrost` [5]	12h57 (killed)	> 900	0	Killed because of RAM usage

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

Supplementary Table 1: Tested tools for which we were not able to build an index on the 50 *Tara* ocean samples or for which we were not able to perform a query. Tested tool versions: `PAC` commit cee1b5c (as used in the original PAC paper) and commit 940f18b (following a personal communication with the authors), `ggcat`: version v0.1.0, `Themisto`: version v3.2.0, `HIBF`: (`Raptor` version: 3.0.0, `Sharg` version: 1.0.1-rc.1, `SeqAn` version: 3.3.0-rc.1), `MetaGraph` version: 0.3.6., `Bifrost` version: 1.3.0.

	Step	Wall clock time	Max Memory (GB)	Max temp. disk (GB)	Output size on disk (GB)
MetaProFi [10]	KMC3 count	3h44	278	1019	1019
	KMC3 dump	18h11	0	5684	5684
	MetaProFi	8h20	232	226	226
	Overall	30h15	278	5684	226
COBS [2]	KMC3 count	3h44	278	1019	1019
	KMC3 dump	18h11	0	5684	5684
	COBS	4h35	160	184	184
	Overall	26h30	278	5684	184
kmindex	All	**2h56**	**107**	**878**	**164**

Supplementary Table 2: Comparing kmindex indexing performances to MetaProFi and COBS. 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. kmindex can be run with a single command line, here resumed in the "*All*" step. Tested tool versions: kmindex version: 0.4.0, MetaProFi version 0.6.0, COBS commit 1cd6df2. Both COBS and MetaProFi require filtered k-mers as inputs. We utilized KMC3 [4], version 3.2.2, to perform the counting and k-mer filtration. The k-mers that appeared two or more times in any dataset were retained.

1.2 Query performances

Supplementary Table 3 details the results provided in the main text and proposes an extended range of results. Notably, it indicates the peak RAM usage at query time, not included in the main text. This highlights the limited RAM usage of kmindex compared to MetaProFi and COBS during queries.

No. queries	1	10	100	1,000	10,000	100,000	1,000,000	10,000,000
MetaProFi Time	12s72	15s28	1m33	2m57	3m02	3m37	11m56	1h29
MetaProFi Memory peak (GB)	0.3	0.3	0.3	0.32	0.44	2.25	21	203
COBS Time	1s51	1s41	1s91	10s73	1m37	15m14	2h00	15h56
COBS Memory peak (GB)	0.012	0.018	0.036	0.28	2.66	24.58	138	295
kmindex Time	**0s06**	**0s23**	**0s87**	**3s94**	**18s03**	**58s35**	**1m13s**	**4m21s**
kmindex Memory peak (GB)	**0.005**	**0.005**	**0.006**	**0.01**	**0.05**	**0.45**	**4.9**	**46.7**

Supplementary Table 3: Query time performance of the indexes on the 50 *Tara* Ocean samples. Queries are composed of reads uniformly sampled from the 50 *Tara* Oceans datasets. Executions were performed on a cold cache.

1.3 False positive rates

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 that such a random k-mer indeed occurs in the set. This happens with a probability of $\times 10^{-9}$ in the biggest set. Hence the reported False Positive rate is an upper bound. This detail does not impact the conclusions offered by the results.

Results are presented in Supplementary Table 4. As stated in the main manuscript, MetaProFi and COBS, the only tested tools with which we were able to perform queries, have an average false positive rate of 11.18% and 13.29%, respectively. In contrast, with a similar and smaller index size, kmindex shows a negligible average false positive rate of 0.006%, below the expected theoretical result thanks to the usage of the findere approach, indexing words of length 23 for querying 28-mers. Note that the difference $28-23=5$ (noted z in findere) is among the recomanded values. As hilighted in [9], the choice z value leads to robust results as long as $z>2$ and as the indexed words are bigger than 16.

	Average	Median	Min	Max
Theoretical	11.62	10.77	6.86	21.25
`MetaProFi`	11.18	9.92	6.93	21.55
`COBS`	13.29	12.30	7.07	24.60
`kmindex`	**0.006**	**0**	**0**	**0.18**

Supplementary Table 4: False positive rates (in %). Indexed: 50 *Tara* Oceans samples. Queried: k-mers ($k = 28$) from a random sequence of size 10k nucleotides. Theoretical results correspond to the usage of BFs of size 30 billion bits as used by `MetaProFi`. The `COBS` index was built using the "-f 0.25" option to set the FP rate to 25%. `kmindex` results are below the expected theoretical rates at it implements the `findere` approach [9] (see Supplementary Information, Section 1.3).

1.4 Warm and cold query results, and "fast-mode"

Supplementary Table 5 shows time and memory usage results when performing queries using `kmindex`. We evaluate 3 query scenarios from the least to the most favorable: '**c**' cold (empty cache), '**w**' warm (successive distinct queries), and '**w+**' warm+ (successive identical queries). As expected, the query times decrease drastically with the cache benefit, illustrating the I/O bounds of `kmindex`. Note that the differences between '**w**' and '**w+**' decrease with the number of queries. Indeed, running a very large number of arbitrary queries increases the probability of loading useful pages for subsequent queries.

In *fast* mode, the kernel is allowed to keep as many pages as possible in the cache resulting in significantly faster queries at the cost of higher memory usage (near to the index size for 10 million queries). Under memory pressure, the memory usage would be equivalent to the *normal* mode.

			Querying							
	cache		**1**	**10**	**100**	**1k**	**10k**	**100k**	**1M**	**10M**
kmindex	**c**	**T (s)**	0.06	0.23	1.24	4.71	19.78	53.72	93.90	261
		M (GB)	0.005	0.005	0.006	0.01	0.05	0.45	4.9	46.7
	w	**T**	0.06	0.20	1.15	4.02	10.84	16.42	40.76	225
		M	0.005	0.006	0.006	0.01	0.06	0.43	4.70	42.6
	w+	**T**	0.02	0.1	0.74	2.65	5.64	13.54	42.12	227
		M	0.005	0.006	0.006	0.01	0.05	0.44	4.48	43.70
kmindex fast	**c**	**T**	0.06	0.10	0.31	2.34	16.56	44.87	61.50	98.52
		M	0.005	0.009	0.035	0.29	2.83	25.7	133	194
	w	**T**	0.03	0.08	0.24	1.57	7.20	7.86	15.79	64.36
		M	0.005	0.009	0.035	0.29	2.84	25.7	133	194
	w+	**T**	0.06	0.05	0.07	0.18	1.04	4.63	15.18	62.33
		M	0.005	0.009	0.035	0.29	2.84	25.7	133	194

Supplementary Table 5: Time (seconds) and memory (GB) performances when querying from 1 read to 10 million reads over the 50 *Tara* Ocean samples indexed with `kmindex`. We consider the following scenarios: '**c**': cold (empty cache), '**w**' warm (successive distinct queries), and '**w+**' warm+ (successive identical queries). Queries are performed using 32 threads.

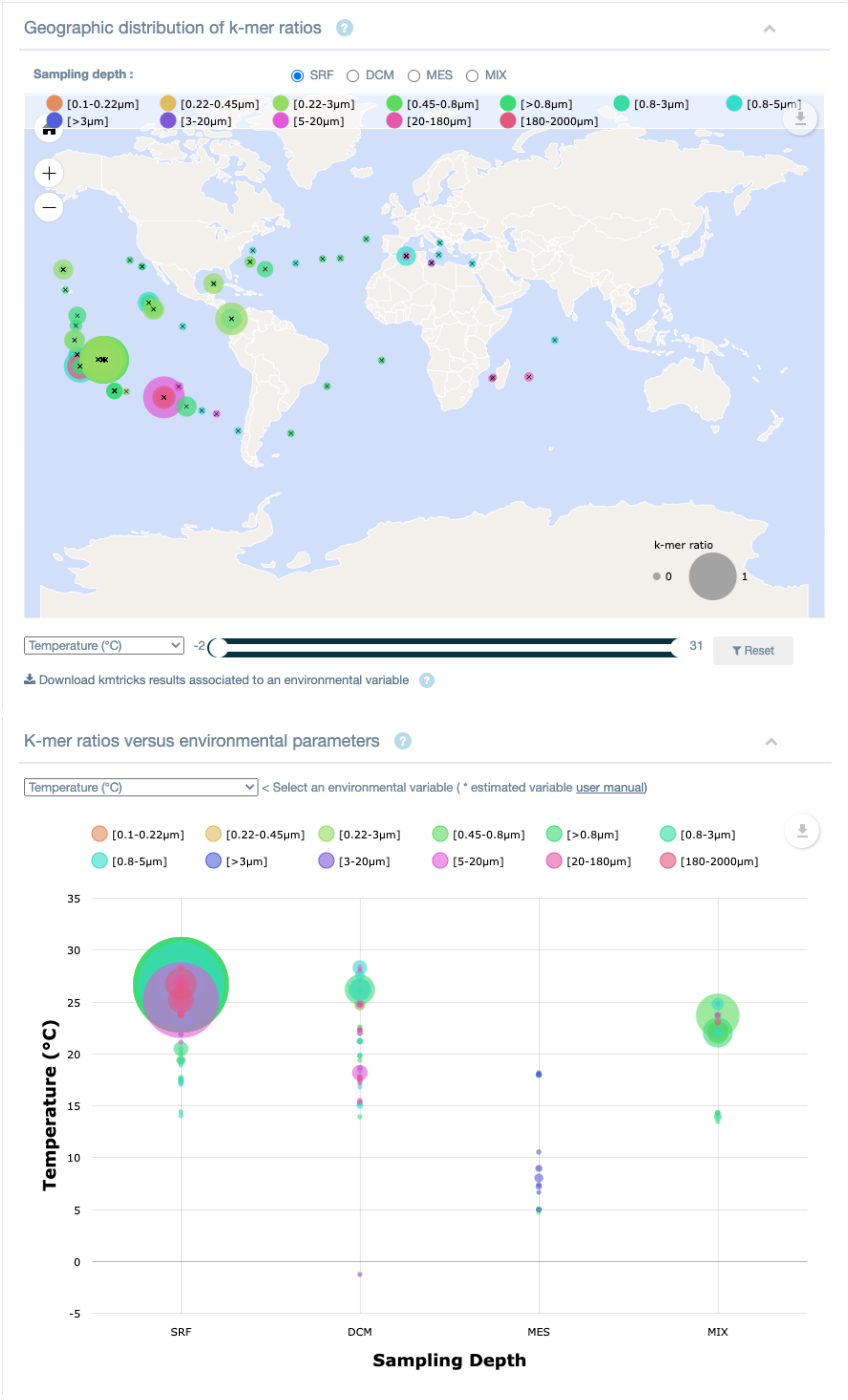
1.5 `kmindex` dynamicity performances

`kmindex` disposes of two distinct ways to add novel samples to an existing index. The indexing time does not depend on the chosen approach. As presented in Supplementary Table 6, when merging indexes together, the query time is optimal, equivalent to the one obtained from the same index built directly on the full dataset. However, this approach has the constraint that all the merged indexes have to be built using the same parameters (hash function, number of partitions, bloom filter sizes). On the other hand, when distinct indexes are *registered* together, each index is individually queried increasing the running time. However, as the indexing parameters are independent, this solution is more flexible. It is well adapted when indexing highly diverse samples such as samples from other *Tara* missions or distinct phylogenetic groups for instance.

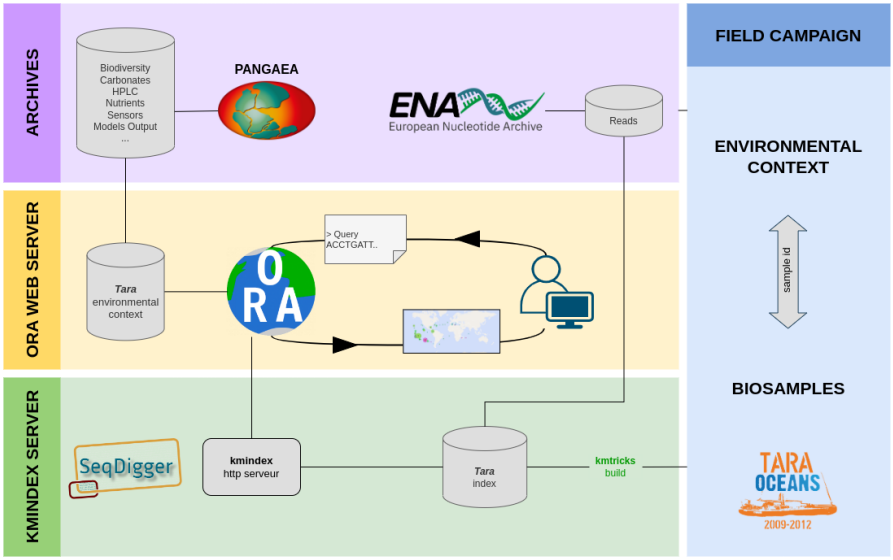
No. queries	1	10	100	1,000	10,000	100,000	1,000,000	10,000,000
`kmindex` original (50 samples at once)	0.13	0.13	0.40	2.46	16.52	41.44	54.87	100.36
`kmindex` merged (5×10 samples merged)	0.14	0.15	0.40	2.47	16.54	43.92	53.81	92.26
`kmindex` register (5×10 samples registered)	0.37	0.64	1.91	10.76	45.68	72.68	82.43	259.98

Supplementary Table 6: Time performances (seconds) when querying from 1 read to 10 million reads over the 50 *Tara* Ocean samples indexed with `kmindex`. The index is built either as "**original**", "**merged**", or "**register**". With the "**original**" approach, the 50 samples are indexed in a unique process. With the "**merged**" and the "**register**" approaches, the 50 samples are separated into 5 groups of 10 samples each. The "**merged**" approach consists of physically extending an existing index, thus ending up with a unique index with the same performances as in the "**original**" approach. The "**register**" approach consists of registering independent indexes together.

2 Illustrations of the Ocean Read Atlas



Supplementary Figure 1: Screenshot of the "Ocean Read Atlas" result interface. Top: the biogeography distribution of the queried sequence is shown among all data samples. The size of the point depicts the similarity of the queried sequences with the corresponding sample. Bottom: a bubble plot representing the correlation between the query presence and the environmental variables of the samples in which it occurs.



Supplementary Figure 2: The ORA web service is organized in 3 distinct entities: **1)** a database containing the environmental parameters and the biosample information of the campaign, **2)** the `kmindex` server allowing index request, and **3)** the ORA server making the link between the 2 previous entities and allowing visualization of the results *via* a web interface.

References

[1] Jarno N Alanko, Jaakko Vuohtoniemi, Tommi Mäklin, and Simon J Puglisi. Themisto: a scalable colored k-mer index for sensitive pseudoalignment against hundreds of thousands of bacterial genomes. *bioRxiv*, pages 2023–02, 2023.

[2] 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.

[3] Andrea Cracco and Alexandru I Tomescu. Extremely fast construction and querying of compacted and colored de bruijn graphs with ggcat. *Genome Research*, pages gr–277615, 2023.

[4] Sebastian Deorowicz, Marek Kokot, Szymon Grabowski, and Agnieszka Debudaj-Grabysz. Kmc 2: fast and resource-frugal k-mer counting. *Bioinformatics*, 31(10):1569–1576, 2015.

[5] 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.

[6] 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.

[7] 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.

[8] Svenja Mehringer, Enrico Seiler, Felix Droop, Mitra Darvish, René Rahn, Martin Vingron, and Knut Reinert. Hierarchical interleaved bloom filter: enabling ultrafast, approximate sequence queries. *Genome Biology*, 24(1):1–25, 2023.

[9] 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.

[10] 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.