Memory-bound k-mer selection for large evolutionary diverse reference libraries

Ali Osman Berk Şapcı,[1] and Siavash Mirarab[1,2*]

[1] *Department of Electrical and Computer Engineering, UC San Diego, San Diego, CA 92093, USA*
[2] *Bioinformatics and Systems Biology Graduate Program, UC San Diego, San Diego, CA 92093, USA*
[*] *Correspondence: smirarab@ucsd.edu*

Abstract

Using long k-mers to find sequence matches is increasingly used in many bioinformatic applications, including metagenomic sequence classification. The accuracy of these downstream applications relies on the density of the reference databases, which, luckily, are rapidly growing. While the increased density provides hope for dramatic improvements in accuracy, scalability is a concern. The k-mers are kept in the memory during the query time, and saving all k-mers of these ever-expanding databases is fast becoming impractical. Several strategies for subsampling k-mers have been proposed, including minimizers and finding taxon-specific k-mers. However, we contend that these strategies are inadequate, especially when reference sets are taxonomically imbalanced, as are most microbial libraries. In this paper, we specifically ask the question: Given limited memory, what is the best strategy to select a subset of k-mers from an ultra-large dataset to include in a library such that the classification of reads suffers the least? We explore strategies to achieve this goal and show a set of experiments demonstrating the limitations of existing approaches, especially for novel and poorly sampled groups. We propose a library construction algorithm called KRANK (K-mer RANKer) that combines several components, including a hierarchical selection strategy with adaptive size restrictions and an equitable coverage strategy. We implement KRANK in highly optimized code and combine it with the locality-sensitive-hashing classifier CONSULT-II. Our method is able to reduce the memory consumption from roughly 140Gb down to 6, 12, or 24Gb, with only a 3.8%, 2.5%, or 0.5% loss in the F1 score. We show in extensive analyses that KRANK outperforms alternatives in both taxonomic classification and taxonomic profiling, using reasonable memory sizes.

Code availability: The implementation is available at `https://github.com/bo1929/KRANK`.

Keywords: k-mer selection, k-mer minimization, Reference library construction, Taxonomic classification, Abundance profiling, Metagenomics

1 Introduction

The number of available genomes has been growing dramatically in recent years, partly due to increased sequencing of microbial and viral species. Of the 322,193 bacterial species deposited on RefSeq since 2000, 15% have been added in 2023 alone. A major benefit of having access to all these genomes is the potential to search new *query* sequences against them. Results can be used to classify reads from a metagenomic sample and to detect contaminants. It has been long appreciated that the accuracy of these downstream applications in metagenomics relies on having access to dense reference sets; classification suffers when the query is distant from all reference genomes [6, 14, 20]. Thus, having access to more genomes gives us hope to create ultra-large reference sets [e.g., 2, 15] and immediately improve the accuracy of a several applications. This hope, however, runs into a mundane but key limitation — the memory needed to utilize these genomes as reference libraries.

Leading methods for building ultra-large reference libraries that allow classifying short reads are based on k-mers. Methods such as CLARK [12], CONSULT-II [18], and Kraken-II [21] extract k-mers for some large k (e.g., 31) from the reference set, somehow try to reduce the number of k-mers, and build a searchable data structure (e.g., a hash table) of all the k-mers and their taxonomic associations. At the time of the query, the entire data structure is loaded into the memory, k-mers from reads are extracted and searched against the data structure, and a read is classified at some taxonomic rank if certain heuristic conditions (specific to each method) are met. This approach works perfectly fine given millions or even a few billion k-mers but starts to run out of memory given much larger datasets. For example, we were able to [18] fit 8 billion 32-mers and their taxonomic labels into ≈140Gb of memory, but our moderate-size database of 10,575 genomes [24] included close to 20 billion 32-mers even after minimization from 35-mer. The 8 billion were sampled arbitrarily from the 20 billion. Such trade-offs for selecting subsets of k-mers will be increasingly felt by developers of k-mers matching methods and practitioners aiming to build ultra-large reference sets.

For the community to benefit from the ever-growing set of available genomes, we need improved methods of subsampling k-mers in a memory-bound fashion. Of course, we are not the first to note the need to subsample k-mers. Minimization is a standard technique that selects some among adjacent k-mers [17, 23] and is used by most tools. However, we argue that beyond minimization, which focuses on the redundancy of adjacent k-mers, we need to consider the evolutionary dimension. A notable attempt in this direction is seeking discriminative k-mers [5, 12] specific to a species but not found in other species. Despite the intuitive appeal of this strategy, we note that many other strategies can also be imagined. Our goal is to explore some of these alternatives.

Subsampling k-mers from ultra-large and heterogeneous databases available these days faces many challenges. On the one hand, sampling of taxonomic groups in public libraries is very imbalanced, which can become a challenge [10]. For example, RefSeq includes 35947 *Escherichia* genomes while 1164 genera have only a single genome. In the WoL-v1 database of Zhu et al. [24], despite their attempts to find representative species, the number of genomes sampled per taxon in each taxonomic rank varies three orders of magnitude (Fig. 1A). The true imbalance in evolutionary history (where some clades are much more diverse, abundant, and species-rich than others) in addition to biases in sampling (where some environments are sampled far more than others) can lead to a naive selection of k-mers missing entire branches of the evolutionary tree. The situation is made worse by the fact that for $k \gg 20$, a large number of k-mers are not shared even among genomes of the same species. Examining the WoL-v1 dataset, we see that 15% of pairs of genomes from the same species have pairwise distances above 5% (Fig. 1B), which results in 98% of 30-mers being unique to each genome in such pairs (Fig. 1C). Pairs of genomes from the same genus but different species shared more than 10% of their 30-mers in only 1% of cases. Thus, as the number of genomes N grows, the number of unique k-mers grows close to proportionally to N. Moreover, k-mers unique

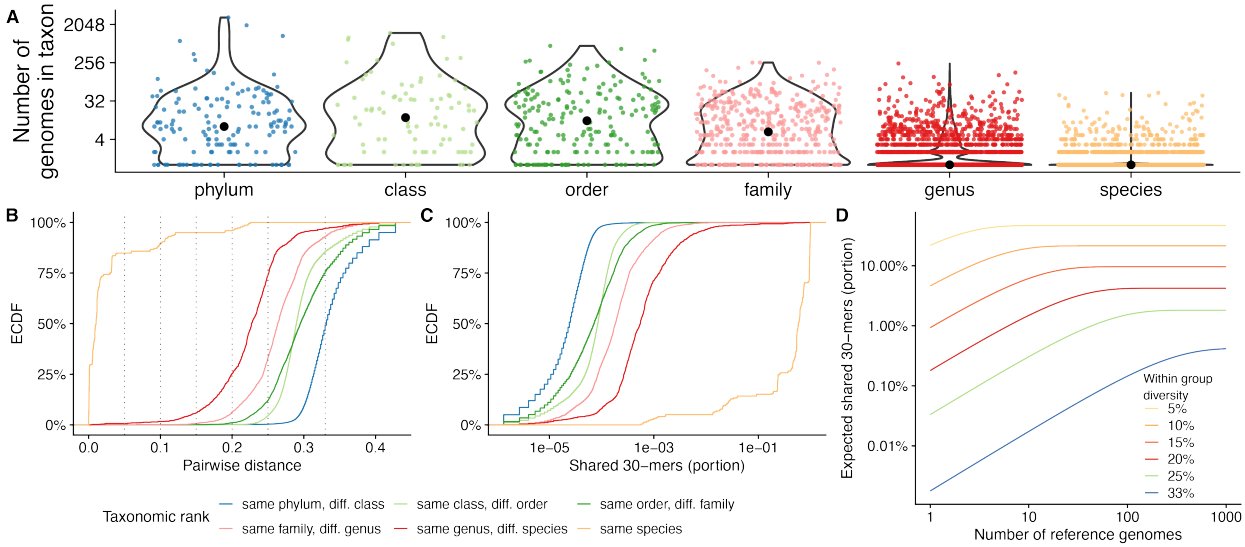

Figure 1: A) Number of genomes in WoL-v1 [24] reference set under each taxonomic node (dots), separated by ranks. B,C) The distribution of Mash [11] estimated genomic distances (B) and Jaccard similarities (C) among 500,000 randomly sampled pairs of genomes that share a taxonomic rank but are different in lower ranks. The empirical commutative distribution function (ECDF) is shown. D) The theoretical expectation for the number of 30-mers shared between a query and at least one of N sampled genomes of a reference set for a group that has within-group diversity $2d$ is shown as $(1-d)^k\left(1-(1-(1-d)^k)^N\right)$.

to ranks above species will be far outnumbered by those unique to a genome or a species at best. As a result, optimal k-mer selection is even more challenging if one tries to find a unique set of k-mers to be used at all ranks (as is often but not always [12], the goal).

In this paper, we propose a set of strategies for selecting a predefined number of k-mers from a large pool of genomes in ways that are conducive to classifying reads. Our goal is to find subsets that do not leave out poorly sampled groups, work across taxonomic ranks, and keep the required information to enable classifying relatively novel reads. While the strategies we explore are general, we implement them in the context of the locality-sensitive hashing (LSH) k-mer classifier called CONSULT-II [18]. Our method, k-mer RANKer (KRANK), takes a taxonomy and a set of genomes labeled with the taxonomy as input. It selects a subset of k-mers from these genomes based on its ranking strategy and the user-defined size. KRANK is centered around a hierarchical traversal of the taxonomy, constructing hash tables separately for each taxon, and merging these to represent the parent taxa. Thus, instead of constructing a global hash table once at the root, it builds the library gradually. This approach allows it to make choices about what k-mers to keep locally instead of globally and also helps with memory efficiency by limiting the k-mer set size in an adaptive manner for each taxon. Our highly optimized and flexible C++ implementation allowed us to compare several alternative strategies empirically. Below, we introduce steps of KRANK together with preliminary data showing their impact. We then move to more formal evaluations of the method for two tasks: taxonomic classification of metagenomic reads and taxonomic profiling of metagenomic samples. In both cases, we show that with this approach (unlike more naive selection strategies), dramatic reductions in library sizes can be sustained without substantial loss of accuracy.

2 Methods: motivation and algorithms

2.1 Preliminaries and motivation

Computational problem. We are given a taxonomic tree $\mathcal{T}$ and a set of genomes $\mathcal{G}$ labeled by the taxonomy. We use $\mathcal{T}$ to denote the set of all nodes of the taxonomy when clear by context. We

let $\mathcal{S} \subseteq \mathcal{T}$ be the set of species, and for $t \in \mathcal{T}$, we let $\mathcal{S}_t \subset \mathcal{S}$ be the set of species under node t. We can choose the leaves of the tree $\mathcal{T}$ to correspond to any desired rank, but by default, we use species ($\mathcal{S}$) as the lowest rank; this can be easily changed to any other level. Fixing a k value, we define $\mathcal{K}$ as the set all of distinct k-mers across all genomes $g \in \mathcal{G}$. $\mathcal{K}_t$ denotes the set of k-mers of all genomes labeled by a taxon $t \in \mathcal{T}$. Notice that, $\mathcal{K}_t = \cup_{t' \in \mathcal{S}_t} \mathcal{K}_{t'}$.

The goal of KRANK is to select a subset of $\mathcal{K}$ with size $M \leq |\mathcal{K}|$ such that the reference set is represented well with the selected k-mers. The choice of M makes libraries bounded in memory at a desired level. We have left the meaning of "well-represented" unspecified for now as many strategies can be considered. We will explore several strategies as ranking methods and test them empirically.

Example dataset. We demonstrate the pros and cons of selection strategies using the microbial dataset of Zhu et al. [24] composed of 10,575 genomes (WoL-v1 hereafter). We excluded 100 archaeal genomes used by Rachtman et al. [16] as queries and five genomes with IDs missing from NCBI from the reference set, leaving us with 569 archaeal and 9901 bacterial references. We chose 756 queries in total; these include 666 bacterial genomes added to RefSeq after WoL-v1 was constructed, 10 randomly selected bacteria from the reference set, and all 80 of 100 archaeal queries from Rachtman et al. [16] with Jaccard index above 0 to some reference genome. The queries span a range of distances to the closest reference genome and are binned into novelty groups as such (Fig. S5A). The query set covers 55 phyla and 396 genera, with some phyla highly represented (e.g., 253 from *Pseudomonadota*, 91 from *Bacillota*) but including several rare phyla (Fig. S5B). We generated 150bp reads using ART [4] at high coverage with default Illumina error profile and then subsampled down to 66667 reads for each query ($\approx 2\times$ coverage for these genomes). We use standard metrics F1, precision, and recall, to evaluate the accuracy with respect to NCBI (see Section A.1 for details).

CONSULT-II's hash tables. While the objective of KRANK and the set of strategies we explore are general and can be used in conjecture with any classification method (e.g., Kraken [22]), in this paper, we focus on improving k-mer selection for a particular classification method, CONSULT-II[18]. We use KRANK to build the (small) libraries used by CONSULT-II. Our k-mer selection steps adopt (and change) a hash table data structure from CONSULT-II, which incidentally helps make our approach scalable by dividing k-mers into smaller subsets based on LSH values. We keep all k-mers in a library, consisting of l separate LSH tables. The shape of each table, denoted by $\mathbf{H}$, is defined by h (number of k-mer positions used for LSH), and b (number of columns per row). $M = 2^{2h}b$ is the total k-mer budget (i.e., the total number of saved k-mers) per each of the l tables. Note that k-mers stored across l hash tables are not mutually exclusive. For each $\mathbf{H}$, we select h random but fixed positions of a k-mer as its hash index, which ensures that two k-mers with low Hamming distance (e.g., ≤ 3) share the same LSH index with high probability in at least one of the tables. Unless otherwise specified, below, we use $l = 2$, $k = 29$, $h = 13$, $b = 16$.

2.2 KRANK Algorithm

The backbone of KRANK is a post-order traversal of taxonomy $\mathcal{T}$ (Algorithm 1). At leaves, distinct k-mers are extracted (ExtractKmers) and encoded (Section 2.2.3), LSH values are computed, and each k-mer is stored in the corresponding row of the table, indexed by LSH. On an internal node (non-species taxon) t, these operations are performed in the given order to build $\mathbf{H}_t$:

1. UnionTables: takes the union of hash tables of its children, $\mathbf{H}_{t'}, t' \in \text{child}(t)$,
2. AdaptiveSizeConstraint: computes a size constraint for $\mathbf{H}_t$,
3. FilterByRank: if the size of the union exceeds the constraint, ranks and removes k-mers.

KRANK saves rows of the hash tables using vectors that allow dynamic size changes during construction, switching to static structures only at the end. At the root, each row gets pruned to

Algorithm 1 KRANK algorithm. Functions are discussed in the text.

```
procedure BuildLibrary(t)
    if t ∈ S then                                  ▷ Checks if t is a member of the bottom rank: species.
        K_t ← ExtractKmers(t)                      ▷ Extract k-mers from input genomes for species t.
        return (H_t ← InitTable(K_t))              ▷ Encodes k-mers, computes LSHs and initializes table for t
    for all t' ∈ child(t) do
        H_t' ← BuildLibrary(t')                    ▷ Recursively calls BuildLibrary for each child of t.
    H_t ← an empty table with 2^(2h) rows
    for all t' ∈ child(t) do
        H_t ← UnionTables(H_t', H_t)               ▷ Takes the union of tables with each child's hash table.
    M(t) ← AdaptiveSizeConstraint(t)               ▷ Computes the maximum allowed size for t.
    n ← |H_t| − M(t)                               ▷ Computes number of extra k-mers currently in the table
    if n > 0 then                                  ▷ Checks if there are more k-mers than the allowed size.
        FilterByRank(H_t, n)                       ▷ Ranks and removes k-mers to satisfy the constraint.
    if t = root then
        PruneThenFlatten(H_t)                      ▷ Prunes each row to b columns and projects table to an array.
    return H_t
```

b k-mers, again using the ranking. This allows converting the table to a memory-efficient flattened static 1D array of size $2^{2h}b$ (PruneThenFlatten). The final library is in the form that CONSULT-II uses. Thus, KRANK adopts the CONSULT-II algorithm for taxonomic classification and abundance profiling, after computing the soft-LCA [18] labels of k-mers. We next present each component of the algorithm, providing empirical evidence that each step improves accuracy.

2.2.1 AdaptiveSizeConstraint: Adaptive size constraints

The hierarchical construction algorithm allows us to make early decisions about what k-mers to include during the post-order traversal. We find it beneficial to preemptively filter *some number* of k-mers from *some tables* because compared to leaving the decisions to the root, gradual filtering helps reduce memory usage. This also helps with accuracy. A major challenge in selecting k-mers for diverse genome sets is the heterogeneity of diversity and sampling: There are wide differences among taxa, even at the same rank, in terms of sampling density (Fig. 1A) and levels of diversity (Fig. 1B). If we delay filtering and randomly sample k-mers at the root, the expected number of k-mers from a taxon t would be $M|\mathcal{K}_t|/|\mathcal{K}|$. This proportional contribution can lead to taxa with lower sampling having little representation, while low-diversity highly-sampled groups (e.g., *E. coli*) dominate. We partially address this challenge by imposing a size constraint on each internal node during traversal to avoid having bloated nodes that starve sisters. While this balancing can in theory be done at the root, we would need to keep a map from k-mers to genomes, which is impractical.

We ask how *many* k-mers should be kept at each taxonomic node. We define $M(t) \leq M$ to be the upper bound (i.e., the size constraint) for the number of k-mers kept for each table at each node (i.e., AdaptiveSizeConstraint in Algorithm 1). Let $r : \mathcal{T} \to (0, 1]$ assign a portion of the full budget M to each node t; any function r is valid as long as it takes the value 1 at the root (i.e., $r(\text{root}) = 1$) and increases as we move up the tree (i.e., $r(t) \geq r(t')$ if $t' \in \text{child}(t)$). We set

$$M(t) = \begin{cases} |\mathcal{K}_t| & t \in \mathcal{S}, \\ \sqrt{r(t)}M & \text{otherwise} \end{cases} \tag{1}$$

noting that $\sum_{t' \in \text{child}(t)} M(t') \geq M(t)$ due to the concavity of the square root. We propose two

options for $r(t)$:

$$r(t) = \begin{cases} \dfrac{\sum_{s \in \mathcal{S}_t} |\mathcal{K}_s|}{\sum_{s \in \mathcal{S}} |\mathcal{K}_s|} & \text{total } k\text{-mer count constraint,} \\ \dfrac{|\mathcal{S}_t|}{|\mathcal{S}|} & \text{number of species constraint.} \end{cases} \tag{2}$$

Going up the tree, the k-mer budget assigned to each taxon increases compared to each child, but it never exceeds the total budget assigned to its children. The two options for $r(t)$ would be identical if all species have equal-sized genomes and each species is sampled the same number of times ($\mathcal{K}_s$ can be the union over multiple genomes). Their main difference is that the first option allows highly sampled species (or very large genomes) to contribute more, while the latter does not. With either definition, the concavity of $M(t)$ makes sure that the size constraint becomes more restrictive as we move up the tree. Close to the leaves, very few k-mers would be removed (e.g., a single species out of 10^4 still is assigned 1% of the total budget when visiting this node), while close to the root, taxon budgets get closer to the full budget (e.g., a phyla with $2/3$ of the species is assigned 82% of the budget). Note that the k-mers from a genome not removed at one taxon can still be removed further up the tree.

We empirically evaluate these two heuristics against the baseline of enforcing no adaptive constraint; i.e., $M(t) = |\mathcal{K}_t|$ if $t \neq$ root and $M(root) = M$, with random k-mer selection. Imposing the adaptive size constraints can make a substantial difference for taxonomic groups with low levels of sampling (Fig. 2A). For example for phyla with $[0, 50]$ and $(50, 500]$ reference genomes sampled, both definitions from (2) improve F1 accuracy over enforcing no adaptive constraint for up to 35% and 12%, respectively, with similar improvements obtained at other ranks. Thus, the adaptive size constraints achieve their stated goal of ensuring groups with lower sampling are not left out. This equitable spreading of the k-mer budget inevitably results in reduced representation for highly sampled groups. For example, for the few phyla with more than 500 reference genomes, we observe a considerable drop in the performance (e.g., F1 score declines are 14% and 20% for k-mer count and the number of species heuristics, resp.). Thus, it may appear that better sampling of low-representation groups has to come at the expense of the high-representation groups; however, this tradeoff can be avoided.

Since we have two different tables ($l = 2$) in the library, we can have a mixed strategy: We enforce adaptive size constraint for one table using the k-mer count constraint of (2) and no adaptive constraint for the other. This mixed strategy is almost as good as the adaptive version for low-representation groups, and almost as good as the no-adaptive-constraint strategy for highly sampled groups (Fig. 2A). Overall, this mixed strategy outperforms having no adaptive constraints, both in terms of precision and recall (Fig. 2B). The improvements are most visible for moderately novel queries (distance to the closest $(0.025, 0.1]$). Across queries, the mixed strategy improves the recall by 6% – 10% and precision by 2% – 5% above the genus rank. Thus, we adopt this mixed mode as the default.

2.2.2 FilterByRank: Which k-mers to select?

Faced with the question of which k-mers to select given the limited budget, three options immediately present themselves: selecting randomly, selecting *discriminative* k-mers unique to some taxa, or conversely, selecting common k-mers present in many taxa. We explore these options and offer an alternative that seeks to maximize the minimum coverage of taxa (min-max strategy). To choose among these ranking approaches, besides intuitive arguments, we rely on empirical data. We compare them with the adaptive size constraint approach described above using k-mer count constraints for both tables. For each hash taxon t, FilterByRank removes k-mers when the size

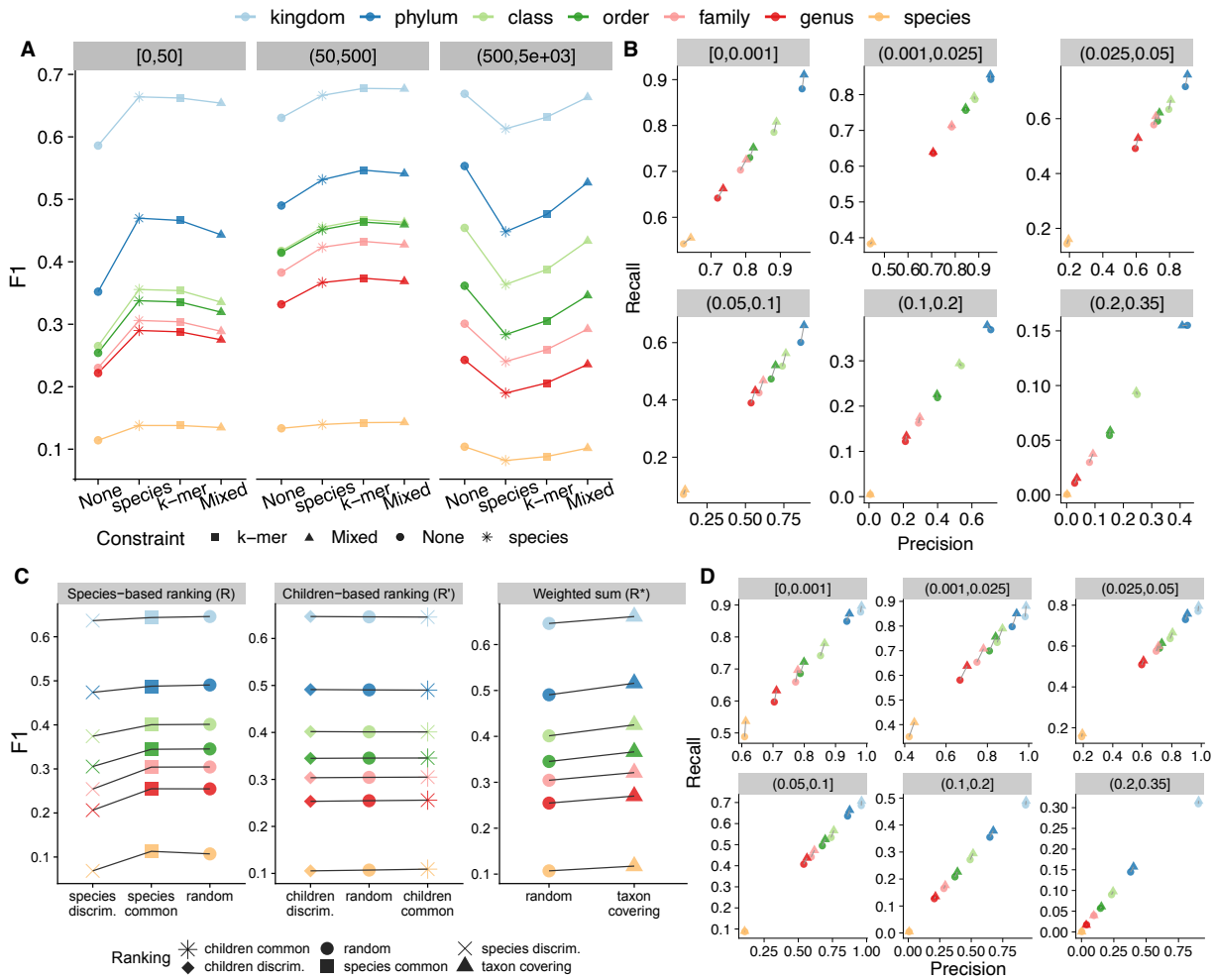

Figure 2: A) Comparison of adaptive size-constraint approaches with random k-mer selection in terms of average F1 scores; None: enforcing no constraint, species: number of species constraint, k-mer: total k-mer count constraint, Mixed: no constraint for one table and k-mer count constraint for the other table. We divide queries into three bins based on the number of reference genomes in the phylum of the query (panels). B) Precision vs recall of None vs Mixed (default) for bins of query novelty. C) Evaluation of different ranking strategies (discriminative versus common) using different measures (species-based R and children-based R′) and using our heuristic (weighted-sum R*). D) Precision vs recall of Random vs weighted-sum heuristic for different bins of query novelty. In A) and C) x-axes are ordered by the average F1 score of all queries. All results use: $w = 35$, $k = 32$, $l = 2$, $h = 12$, and $b = 16$. In B) and C), the distance to the closest reference, defining query novelty, is estimated by Mash [11].

of the table $\mathbf{H}_t$ exceeds $M(t)$. First, each row is pruned to b k-mers according to the strategy. If the table size is still greater than $M(t)$, we continue removing k-mers, one at a time from each row in the descending order of their sizes, and iterating until the size constraint is satisfied.

For a k-mer $x \in \mathcal{K}$ and a set of taxa $\mathcal{T}' \subset \mathcal{T}$, we define $\mathrm{R}, \mathrm{R}' : \mathcal{K} \to [0, 2^{|\mathcal{T}'|})$ as

$$\mathrm{R}(x,t) = |\{y \ : \ x \in \mathcal{K}_y, \, y \in \mathcal{S}_t\}|$$

and

$$\mathrm{R}'(x,t) = \sum\nolimits_{t' \in \mathrm{child}(t)} \mathbb{1}\left\{|\mathrm{R}(x,t')| > 0\right\} \ .$$

R denotes the cardinality of the subset of species under t which includes x, and R′ counts the children of t that have x at least once. We can express each strategy in terms of rankings of each k-mer $x \in \mathcal{K}_t$ using either function and removing low-ranked k-mers accordingly. In the case of ties, which can happen frequently, especially near leaves, we simply break them randomly.

The case against discriminative k-mers. Keeping discriminative k-mers found only in specific target species, an approach that has been proposed by others [12, 13], would correspond to filtering out k-mers where $\mathrm{R}(x,t) > 1$ (this could be easily adapted to other levels). Going one step further, we emulate the discriminative k-mer strategy by ranking k-mers inversely to $\mathrm{R}(x,t)$ or $\mathrm{R}'(x,t)$, thus removing common k-mers. Empirically, using either version is worse than simply using random selection (Fig. 2C). The reduction is negligible with children-based $\mathrm{R}'(x,t)$ but is quite dramatic when used with species counts $\mathrm{R}(x,t)$; for example, the declines in F1 are 36% and 19% for species and genus ranks, respectively. Thus, we caution against using discriminative k-mers in a memory-bound setting and offer some insights into the possible reasons for this poor performance.

Very few k-mers are shared even among different species of the same genus. For example, in the WoL-v1 data set [24], a random sample of genomes that are from the same genus, but different species share only 0.05% of their 30-mers (Fig. 1C) and have Mash distances that typically exceed 20% (Fig. 1B). With simplifying assumptions, we can approximate the probability that a k-mer would be shared between a query genome and at least one of N genomes sampled from a taxonomic group, assuming each pair of genomes in this group has distance d to parent as $(1-d)^k\left(1-(1-(1-d)^k)^N\right)$. Plotting this equation shows that using discriminative k-mers leads to very few matches between a query and the reference (Fig. 1D). For instance, in a group with $d = 20\%$ diversity (which is less than most genera; see Fig. 1B), only 0.7% of query 30-mers are expected to be found in at least one among $N = 5$ references genomes and only 4.2% when $N \to \infty$ (an infinite number of genomes sampled from that genus). Since most k-mers are unique and subsampling is inevitable in a memory-bound setting, removing common k-mers makes it difficult to match any read. This only gets worse for more diverse groups at higher ranks and is not much better within the same species, where the pairwise Mash distance is 5% or more in 15% of cases.

Selecting common k-mers alone does not help. We can emulate common k-mer selection by ranking k-mers proportionally to $\mathrm{R}(x,t)$ or $\mathrm{R}'(x,t)$. Given the argument against discriminative k-mers, common k-mers seem appealing, especially at lower ranks, because they can represent each taxon using conserved sequences present across different genomes. Empirically, this strategy used with either $\mathrm{R}(x,t)$ or $\mathrm{R}'(x,t)$ is better than using discriminative k-mers but is not different than random sampling. Just like randomly selecting k-mers, this strategy focuses the budget on larger and densely sampled taxa. For instance, the WoL-v1 reference set includes 2975 *Pseudomonadota* genomes and only a single genome from many other phyla. While some unevenness is undoubtedly due to genuine differences among the diversity of phyla, some must be due to uneven sampling.

Covering all children improves accuracy. Instead of maximizing total coverage, we can ensure no species is left uncovered. While this can be imposed as a complex set covering problem, in the interest of running time, we opt for a simple approach using weighted sums. We set

$$\mathrm{R}^*(x,t) = \sum\nolimits_{t' \in \mathrm{child}(t)} w_{t'}(x) \cdot \mathrm{R}(x,t')$$

where $w_{t'}$ down-weights groups that are highly sampled among surviving k-mers. A natural choice of weights is $w_t = 1/|\mathbf{H}_t|$. However, this approach consistently down-weights the same children and can remove a child altogether. Thus, we define the weight of a k-mer x locally for its row by setting it to be inversely proportional to the *coverage* of each child taxon among k-mers of that row:

$$w_{t'}(x) = \left(\left|\left\{x' \,:\, x' \in \mathbf{H}_{t'},\, \mathrm{LSH}(x) = \mathrm{LSH}(x')\right\}\right|\right)^{-1}$$

These weights ensure that children covered with fewer surviving k-mers get covered by the parent.

Using this approach improves k-mer selection consistently across all taxonomic levels compared to random strategy, which is similar to using common k-mers (Fig. 2C). Further dividing queries

by their novelty, we observe that this strategy improves both precision and recall, particularly for less novel queries; recall increases as much as %15 increase at the species level (Fig. 2D).

2.2.3 Other components: compact k-mer encodings and hash table operations

In extracting k-mers from all input genomes (ExtractKmers), we use minimizers to perform an initial subsampling by choosing the k-mer whose encoding has a MurmurHash3 [1] value that is the smallest in a local window of size $w = k + 3$. We adopt the CONSULT [16] left/right encodings of k-mers to allow fast calculation of Hamming distance using a `pop-count`, an XOR, an OR, and a shift instruction. For $k > 16$, these encodings require 64 bits. However, h many k-mers positions are already given by the hash index and do not need to be saved as part of the encoding. We save memory by dropping these positions; this reduces the encoding size to a 32bit number when $k - h \geq 16$, which is the case in our default configurations ($k = 30$ and $h = 14$ or $k = 29$ and $h = 13$). Since each k-mer encoding is only 32 bits, we also update the CONSULT-II design so that it saves the encodings in the hash table, as opposed to pointers to a separate array.

For each leaf t, InitTable creates a table with 2^{2h} rows, computes LSH values of each k-mer $x \in \mathcal{K}_t$ based on h positions, and stores k-mer's compact encoding in the row indexed by the LSH value. We keep a counter initialized to 1 associated with each k-mer to enable computing $\mathrm{R}(x, t)$ later. For non-leaf taxa, UnionTables combines hash table rows by taking the union of k-mers. Luckily, the union operation can be defined on a per-row basis as the same k-mers share LSH value; independent union operations are performed in parallel across rows. Union also updates $\mathrm{R}(x, t)$. Furthermore, KRANK uses some of the available threads to perform post-order traversal in parallel.

One final trick is needed: Since we need to merge multiple tables at internal nodes, the required memory may exceed what is available. To tackle this issue, we used a batching approach. Except for the AdaptiveSizeConstraint, all components of our algorithm can be performed independently on a per-row basis. Therefore, we divide rows of the tables into batches and build them separately and in parallel. KRANK has a parameter to set the number of batches, and each batch's size constraint is simply updated as $M(t)$ divided by the number of batches.

3 Results

3.1 Taxonomic classification of the controlled novelty queries

We compared KRANK against CONSULT-II [18], where k-mer selection is arbitrary, Kraken-II (with default settings), and CLARK [13] (with default settings, k=31; species rank), which uses discriminative k-mers. For KRANK, the classification algorithm is identical to CONSULT-II and involves soft-LCA k-mer labeling and a voting mechanism based on hamming distance. We tested KRANK with two memory levels: KRANK-hs (high-sensitivity) sets $k = 30$, $h = 14$, $b = 16$ and uses 51.2Gb of memory while KRANK-lw (lightweight) sets $k = 29$, $h = 13$, $b = 16$ and 12.8Gb of memory. We examined the same query set we have been using so far (Section 2.1).

Across different ranks and query novelty bins, KRANK was able to match the best performance despite using far fewer k-mers and less memory (Fig. 3A). All methods had similar F1 accuracy for queries that were not novel (≤ 0.025), except CLARK which had slightly lower accuracy (e.g., 10% F1 decline for phylum level for the $[0, 0.001]$ bin and 9% decline for the class level for the $(0.001, 0.025]$ bin). For these less novel queries, KRANK-lw using 12.8Gb was indistinguishable from CONSULT-II using 140.7Gb. As queries become more novel (> 0.025), accuracy drops for all methods but there are large differences between them. For example, for the $(0.05, 0.1]$ bin, both versions of KRANK outperform Kraken-II and CLARK above species ranks with increasing margins as the rank goes up. At the phylum rank, KRANK-lw achieves 16% and 35% higher F1 scores compared to Kraken-II and CLARK, respectively. Although the KRANK-hs maintains

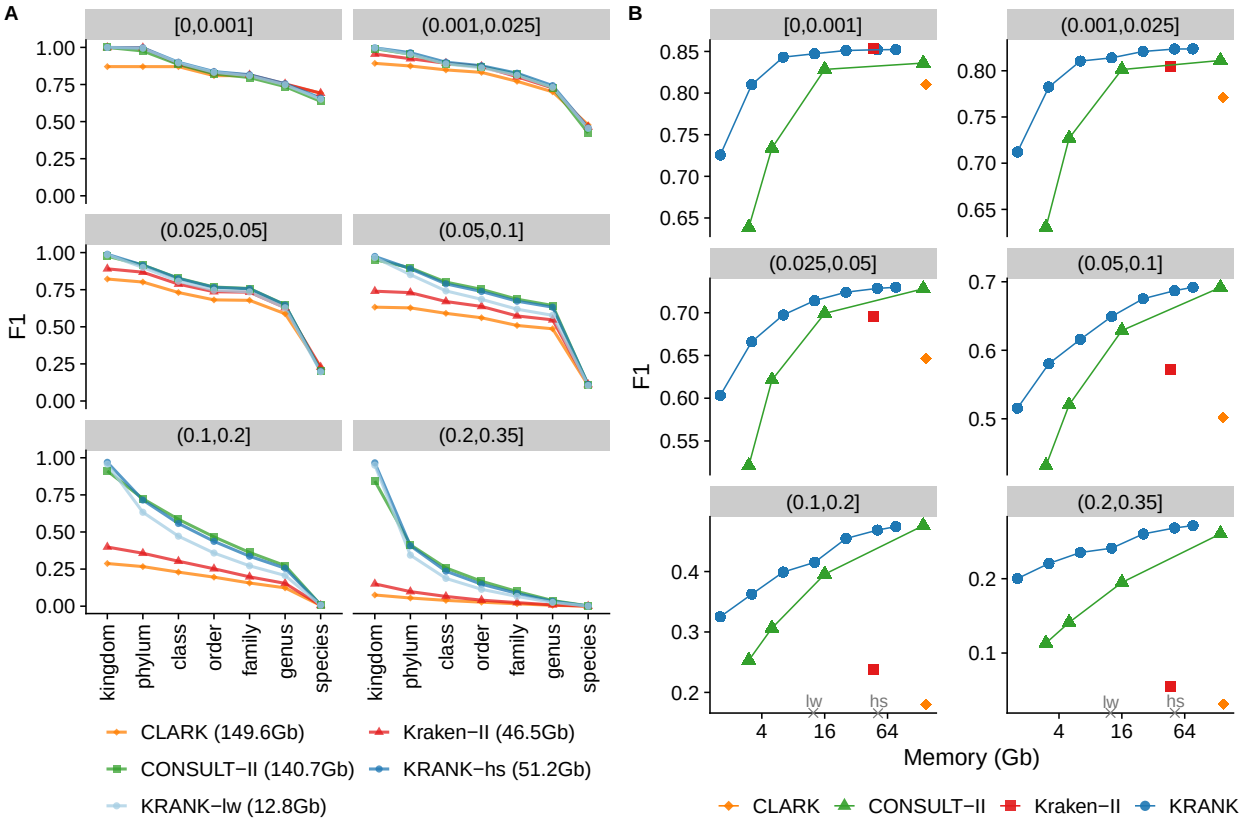

Figure 3: A) Detailed comparison of two default memory levels of KRANK with other methods across different ranks on the WoL-v1 dataset with 66667 simulated 150bp reads from each of 756 query genomes. F1 scores are averages of all queries in each novelty bin. B) Classification performance with different memory levels for KRANK and CONSULT-II. Kraken-II and CLARK are shown as single data points at fixed memory usage. Each panel is a novelty bin, corresponding to the distance to the closest reference genome, measured by Mash [11]. F1 scores are averages of all queries in each novelty bin across ranks. In the x-axis, lw (lightweight) and hs (high-sensitivity) memory levels of KRANK are marked.

almost identical performance with CONSULT-II for the bins $(0.025, 0.05]$ and $(0.05, 0.1]$, KRANK-lw experiences a small decline at some ranks (e.g., 8%–9% at the order rank). The trend accelerates for the most novel queries: KRANK-hs preserves the same performance as CONSULT-II, and KRANK-lw degrades by moderate levels (e.g., 12% decline at the phylum for the $(0.1, 0.2]$ bin). Nevertheless, KRANK-lw outperforms Kraken-II (e.g., 77% and 250% higher F1 at the phylum rank for the bins $(0.1, 0.2]$ and $(0.2, 0.35]$, respectively). Somewhat surprisingly, both versions of KRANK are slightly better than CONSULT-II in terms of kingdom-level classification for novel genomes; this is likely due to better representation of archaea with careful k-mer selection instead of arbitrary selection.

For our two methods, KRANK and CONSULT-II, we can adjust the consumed memory and observe the impact. Varying the memory between 1.6Gb to 140Gb, we saw that KRANK clearly and dramatically outperforms CONSULT-II in terms of average F1 score in memory-bound situations (Fig. 3B). As memory decreases, CONSULT-II's F1 scores decrease much faster than KRANK. For instance, given less than 4Gb, KRANK can be up to 26% more accurate for the least novel bin; it also remains better as the memory increases. For the most novel bin, KRANK with 1.6Gb performs better than CONSULT-II with 16Gb. KRANK with 1.6Gb also outperforms CLARK and Kraken-II for novel queries > 0.1, giving 89% and 160% higher F1 scores than Kraken-II and CLARK, respectively. Note that CONSULT-II and KRANK store the same number of k-mers (268M in a single table) when memory usages are 5Gb and 3.2Gb, respectively, but KRANK's F1 scores remain

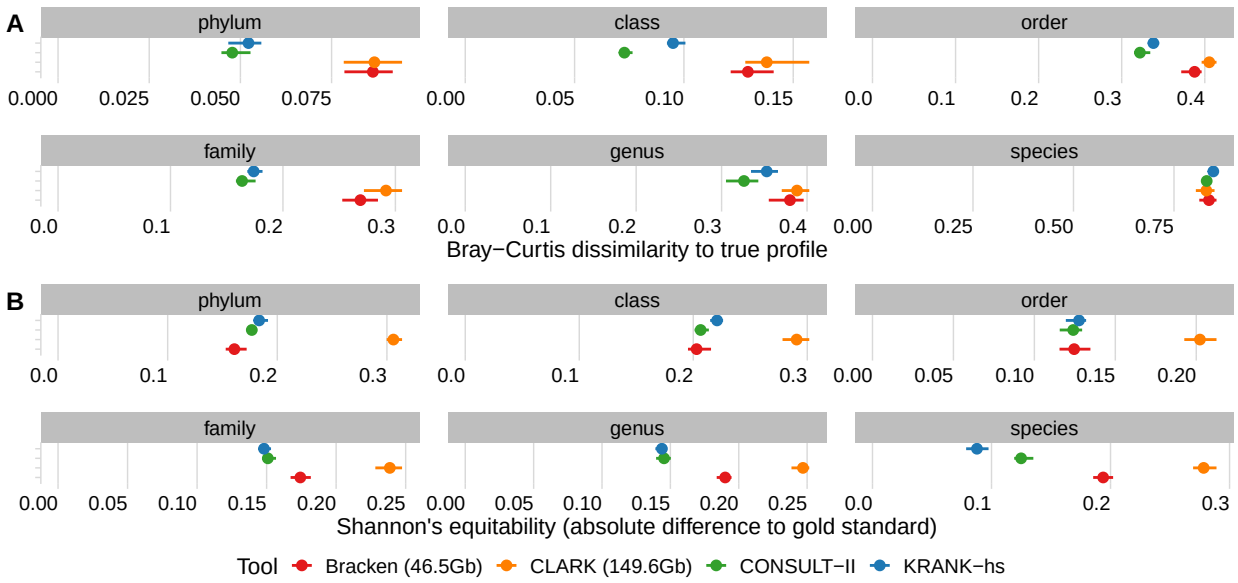

Figure 4: Results of taxonomic profiling on CAM-I challenge. A) The Bray-Curtis metric measures the dissimilarity between the estimated and true profiles. B) We show the difference between Shannon's equitability of the estimated profile and of the gold standard, measuring how well each tool reflects the alpha diversity in terms of the evenness of the taxa abundances. The points are mean values across five samples, and the horizontal bars are for standard error.

higher (Fig. S7). Since the only difference between methods is the selected k-mers, these results clearly demonstrate the effectiveness of the selection algorithm. Interestingly, as memory increases, the F1 score reaches a point of diminishing returns where it improves only slowly, especially for the less novel queries (≤ 0.05).

3.2 Abundance profiling of metagenomic samples of the CAMI-I challenge

We compared the methods in terms of taxonomic profiling accuracy on the dataset from the Critical Assessment of Metagenome Interpretation (CAMI-I) [19] benchmarking challenge. This dataset contains five different high-complexity samples, each of size 75Gbp, which are simulated by mimicking the abundance distribution of the underlying microbial communities. To measure the performances of tools, we used the main two metrics singled out by the open-community profiling assessment tool (OPAL) [9]: the Bray-Curtis dissimilarity between the estimated profile and the true profile and Shannon's equitability as a measure of alpha diversity. We tested the Bracken [7] extension of Kraken-II which combines its results with Bayesian priors for better profiling.

On the CAMI-I high complexity dataset, abundance profiles estimated by CONSULT-II are the most similar to the true profile in terms of Bray-Curtis dissimilarity across all 5 samples (Fig. 4). At the species level, all methods have high errors and are comparable. KRANK performs close to CONSULT-II despite the much-reduced memory, except for high ranks, where performance decline is relatively high (e.g., 16% drop in F1 score at the class). KRANK performs considerably better than both Bracken and CLARK, with 21%, 12%, 3%, 22%, and 3% lower errors than Bracken, across phylum to genus ranks, respectively. Since the largest three phyla in our reference set constitute 80% – 85% of each sample, this dataset presents a case where the KRANK strategy of covering low-sampled groups is not needed and sampling of highly covered phyla by CONSULT is sufficient.

All methods underestimate Shannon's equitability, which is a measure of the variety and distribution of taxa present in a sample. Bracken has closer values to the gold standard at the phylum, class, and order ranks (15% – 20%), whereas below genus, CONSULT-II and KRANK become

better than Bracken (21% at the species). Except for species rank, CONSULT-II's and KRANK's estimates are very close, and the difference never exceeds 6%. Interestingly, KRANK is able to estimate Shannon's equitability 16% more accurately than CONSULT-II at the species level.

3.3 Resource usage in controlled novelty experiment

KRANK does not introduce any extra computation during the classification stage, but smaller libraries often result in finding k-mer matches faster. Thus, KRANK is considerably faster than CONSULT-II. KRANK-hs had 1501 seconds running time, whereas CONSULT-II (140.7Gb version) took 2914 seconds. After the expensive but one-time k-mer selection stage with KRANK, smaller libraries only help in terms of performance due to many reasons such as shorter library loading time and better cache performance. Nevertheless, both CONSULT-II and KRANK are slower than CLARK and Kraken-II, which had 676 seconds and 88 seconds running time, respectively. Our measurements were performed using 96 threads on a machine with 2.2 GHz AMD EPYC 7742 processors. Note that the relatively longer running time of CONSULT-II (and KRANK) is mostly due to the fact that CONSULT-II splits taxonomic identification into two independent stages for the sake of backward compatibility with CONSULT. One subprogram finds k-mer matches and writes them to the disk, then, another one reads matching information and performs prediction.

Even though there is extra computation introduced due to the hierarchical algorithm, KRANK libraries can be constructed faster than CONSULT-II due to its efficient implementation. Moreover, splitting LSH range, i.e., rows of the table, enables having independent jobs that can be run on different cluster nodes, if resources are available. For instance, for high-sensitivity mode, we used 512 batches to split 2^{28} rows. After preprocessing and k-mer extraction (45 minutes with 14 threads), each batch was run as a separate job and took 10 minutes on average with 14 threads. Increasing the number of threads for each batch can further decrease the running time but results in requiring more memory as some threads are used to perform tree traversal in parallel. Using this configuration, the peak memory usage for each batch to build its part of the table was less than 32 GB. In total, KRANK-hs used around 2400 CPU hours 10,575 genomes (2 tables and 512 batches, 10 minutes with 14 threads). Kraken-II was again the fastest tool, and needed ≈3.5 hours with 96 threads, followed by CLARK (≈10 hours with 96 threads). CONSULT-II took ≈28 hours (with 128 threads) to construct, excluding the k-mer extraction step which was performed by Jellyfish[8].

4 Conclusion and future work

We introduced KRANK for selecting a memory-bound and representative k-mer set from a large genome collection for taxonomic classification and abundance profiling. Our approach is based on a hierarchical k-mer filtering algorithm to build an efficient library with high predictive power. We explored various strategies for enforcing size constraint and k-mer ranking and chose among them empirically. We targeted robustness for challenging cases such as novel queries and groups with low sampling. While we paired KRANK here with the CONSULT-II classification algorithm, its strategies are general and its selected k-mers can be used by any method. Future work should make KRANK compatible with other popular k-mer-based tools such as Kraken-II and adopt the standardized k-mer format of Dufresne et al. [3].

We note that library construction is relatively expensive. Nevertheless, the creation of the reference library is a one-time operation, amortized over many subsequent uses. Furthermore, it is possible to extend KRANK to allow incremental updates to the library. Here, we used a moderately large dataset to demonstrate the ability of KRANK to reduce the size without sacrificing accuracy compared. Immediate future work should focus on building libraries with varying sizes, ranging from a couple of gigabytes to hundreds of gigabytes, from super-large databases such as all of

the RefSeq database, GTDB [15], and WoL-v3 [2], which the community will find useful. Finally, while in this paper we used the taxonomic tree to represent the evolutionary history, this is not a requirement of the algorithm. KRANK can use a more accurate phylogenetic tree as a reference, and our classification algorithm can be extended to allow phylogenetic placement in the future.

References

[1] Appleby, A. (2009). MurmurHash3.

[2] Balaban, M., Jiang, Y., Zhu, Q., McDonald, D., Knight, R., and Mirarab, S. (2023). Generation of accurate, expandable phylogenomic trees with uDance. *Nature Biotechnology*.

[3] Dufresne, Y., Lemane, T., Marijon, P., Peterlongo, P., Rahman, A., Kokot, M., Medvedev, P., Deorowicz, S., and Chikhi, R. (2022). The K-mer File Format: a standardized and compact disk representation of sets of k -mers. *Bioinformatics*, 38(18):4423–4425.

[4] Huang, W., Li, L., Myers, J. R., and Marth, G. T. (2012). ART: A next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594.

[5] Lee, D., Karchin, R., and Beer, M. A. (2011). Discriminative prediction of mammalian enhancers from DNA sequence. *Genome Research*, 21(12):2167–2180.

[6] Liang, Q., Bible, P. W., Liu, Y., Zou, B., and Wei, L. (2020). DeepMicrobes: taxonomic classification for metagenomics with deep learning. *NAR Genomics and Bioinformatics*, 2(1).

[7] Lu, J., Breitwieser, F. P., Thielen, P., and Salzberg, S. L. (2017). Bracken: estimating species abundance in metagenomics data. *PeerJ Computer Science*, 3:e104.

[8] Marçais, G. and Kingsford, C. (2011). A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics*, 27(6):764–770.

[9] Meyer, F., Bremges, A., Belmann, P., Janssen, S., McHardy, A. C., and Koslicki, D. (2019). Assessing taxonomic metagenome profilers with OPAL. *Genome Biology*, 20(1):51.

[10] Nasko, D. J., Koren, S., Phillippy, A. M., and Treangen, T. J. (2018). RefSeq database growth influences the accuracy of k-mer-based lowest common ancestor species identification. *Genome Biology*, 19(1):165.

[11] Ondov, B. D., Treangen, T. J., Melsted, P., Mallonee, A. B., Bergman, N. H., Koren, S., and Phillippy, A. M. (2016). Mash: fast genome and metagenome distance estimation using MinHash. *Genome Biology*, 17(1):132.

[12] Ounit, R. and Lonardi, S. (2016). Higher classification sensitivity of short metagenomic reads with CLARK-S. *Bioinformatics (Oxford, England)*, 32(24):3823–3825.

[13] Ounit, R., Wanamaker, S., Close, T. J., and Lonardi, S. (2015). CLARK: fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers. *BMC Genomics*, 16(1):236.

[14] Pachiadaki, M. G., Brown, J. M., Brown, J., Bezuidt, O., Berube, P. M., Biller, S. J., Poulton, N. J., Burkart, M. D., La Clair, J. J., Chisholm, S. W., and Stepanauskas, R. (2019). Charting the Complexity of the Marine Microbiome through Single-Cell Genomics. *Cell*, 179(7):1623–1635.e11.

[15] Parks, D. H., Chuvochina, M., Waite, D. W., Rinke, C., Skarshewski, A., Chaumeil, P.-A., and Hugenholtz, P. (2018). A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life. *Nature Biotechnology*, 36(10):996–1004.

[16] Rachtman, E., Bafna, V., and Mirarab, S. (2021). CONSULT: accurate contamination removal using locality-sensitive hashing. *NAR Genomics and Bioinformatics*, 3(3):10.1101/2021.03.18.436035.

[17] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M., and Yorke, J. A. (2004). Reducing storage requirements for biological sequence comparison. *Bioinformatics (Oxford, England)*, 20(18):3363–3369.

[18] Şapcı, A. O. B., Rachtman, E., and Mirarab, S. (2024). CONSULT-II: Accurate taxonomic identification and profiling using locality-sensitive hashing. *bioRxiv*.

[19] Sczyrba, A., Hofmann, P., Belmann, P., Koslicki, D., Janssen, S., Dröge, J., Gregor, I., Majda, S., Fiedler, J., Dahms, E., Bremges, A., Fritz, A., Garrido-Oter, R., Jørgensen, T. S., Shapiro, N., Blood, P. D., Gurevich, A., Bai, Y., Turaev, D., DeMaere, M. Z., Chikhi, R., Nagarajan, N., Quince, C., Meyer, F., Balvočiūtė, M., Hansen, L. H., Sørensen, S. J., Chia, B. K. H., Denis, B., Froula, J. L., Wang, Z., Egan, R., Don Kang, D., Cook, J. J., Deltel, C., Beckstette, M., Lemaitre, C., Peterlongo, P., Rizk, G., Lavenier, D., Wu, Y.-W., Singer, S. W., Jain, C., Strous, M., Klingenberg, H., Meinicke, P., Barton, M. D., Lingner, T., Lin, H.-H., Liao, Y.-C., Silva, G. G. Z., Cuevas, D. A., Edwards, R. A., Saha, S., Piro, V. C., Renard, B. Y., Pop, M., Klenk, H.-P., Göker, M., Kyrpides, N. C., Woyke, T., Vorholt, J. A., Schulze-Lefert, P., Rubin, E. M., Darling, A. E., Rattei, T., and McHardy, A. C. (2017). Critical Assessment of Metagenome Interpretation—a benchmark of metagenomics software. *Nature Methods*, 14(11):1063–1071.

[20] von Meijenfeldt, F. A. B., Arkhipova, K., Cambuy, D. D., Coutinho, F. H., and Dutilh, B. E. (2019). Robust taxonomic classification of uncharted microbial sequences and bins with CAT and BAT. *Genome Biology*, 20(1):217.

[21] Wood, D. E., Lu, J., and Langmead, B. (2019). Improved metagenomic analysis with Kraken 2. *Genome Biology*, 20(1):257.

[22] Wood, D. E. and Salzberg, S. L. (2014). Kraken: Ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3).

[23] Zheng, H., Marçais, G., and Kingsford, C. (2023). Creating and Using Minimizer Sketches in Computational Genomics. *Journal of Computational Biology*, page cmb.2023.0094.

[24] Zhu, Q., Mai, U., Pfeiffer, W., Janssen, S., Asnicar, F., Sanders, J. G., Belda-Ferre, P., Al-Ghalith, G. A., Kopylova, E., McDonald, D., Kosciolek, T., Yin, J. B., Huang, S., Salam, N., Jiao, J.-Y., Wu, Z., Xu, Z. Z., Cantrell, K., Yang, Y., Sayyari, E., Rabiee, M., Morton, J. T., Podell1, S., Knights, D., Li1, W.-J., Huttenhower, C., Segata, N., Smarr, L., Mirarab, S., and Knight, R. (2019). WoL: reference phylogeny for microbes (data pre-release).

Supplementary Information

Appendix A Supplementary methods

A.1 Evaluation metrics

We evaluated the predictions of each tool with respect to the NCBI taxonomy. Each prediction is separately evaluated across each taxonomic rank. When the reference set had at least one genome matching the query taxon at a particular rank, we call it a *positive*: TP if it is the correct taxon, FP if it is the incorrect taxon, and FN if the read is not classified at that rank. Similarly, at a particular rank, when the reference set lacks a genome from the query taxon, we call it a *negative*: TN if the read is not classified at that rank, FP if it is classified, which would necessarily be false. We ignored queries at ranks where the true taxonomic ID given by NCBI is a missing rank.

Appendix B Supplementary Figures

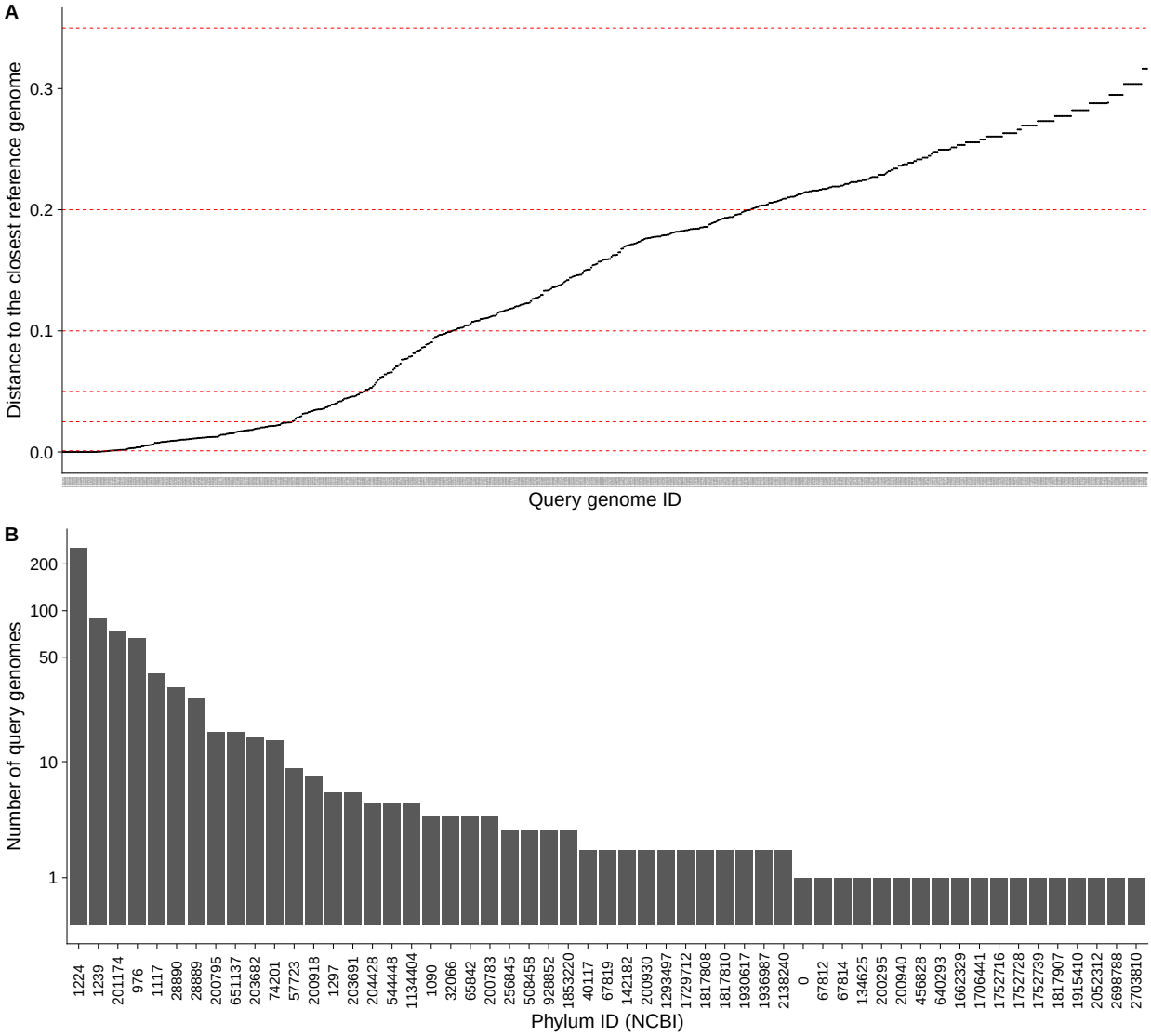

Figure S5: A) The distance to the closest reference genome, as estimated by Mash, of each query genome. Each data point is for a query genome (x-axis). B) Distribution of 765 queries across different phyla. The number of query genomes from the corresponding phylum is given the y-axis, and the x-axis is the NCBI ID of the phylum. 0 refers to genomes with unknown/unassigned at the phylum level.

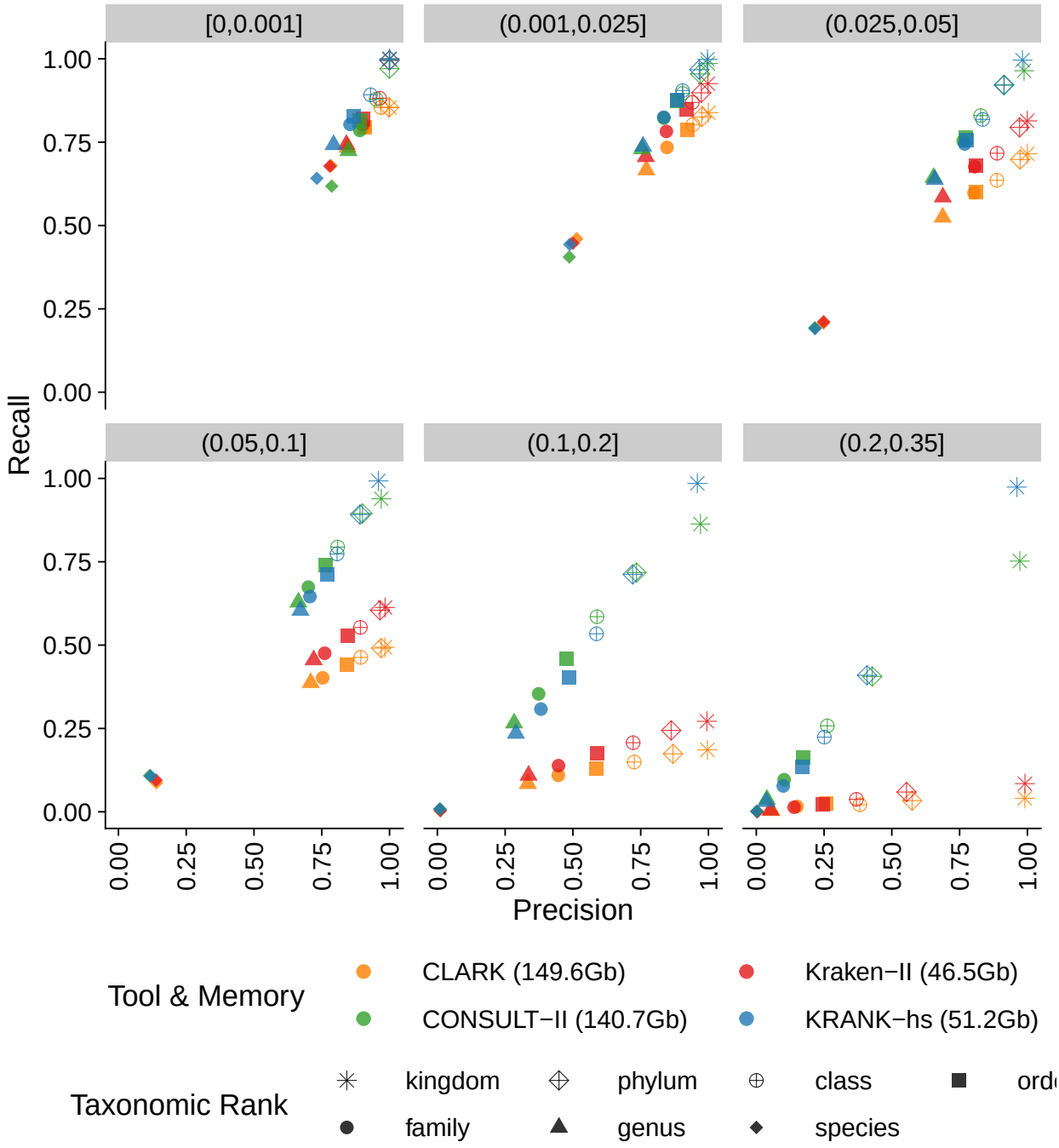

Figure S6: A more fine-grained comparison of the high-sensitivity mode of KRANK with other methods across different ranks on the WoL-v1 dataset with 66667 simulated 150bp reads from each of 756 query genomes. Results are averaged over all queries and reported across different novelty bins and ranks in terms of precision (x-axis) and recall (y-axis) scores. Shapes of the data points stand for the taxonomic rank.

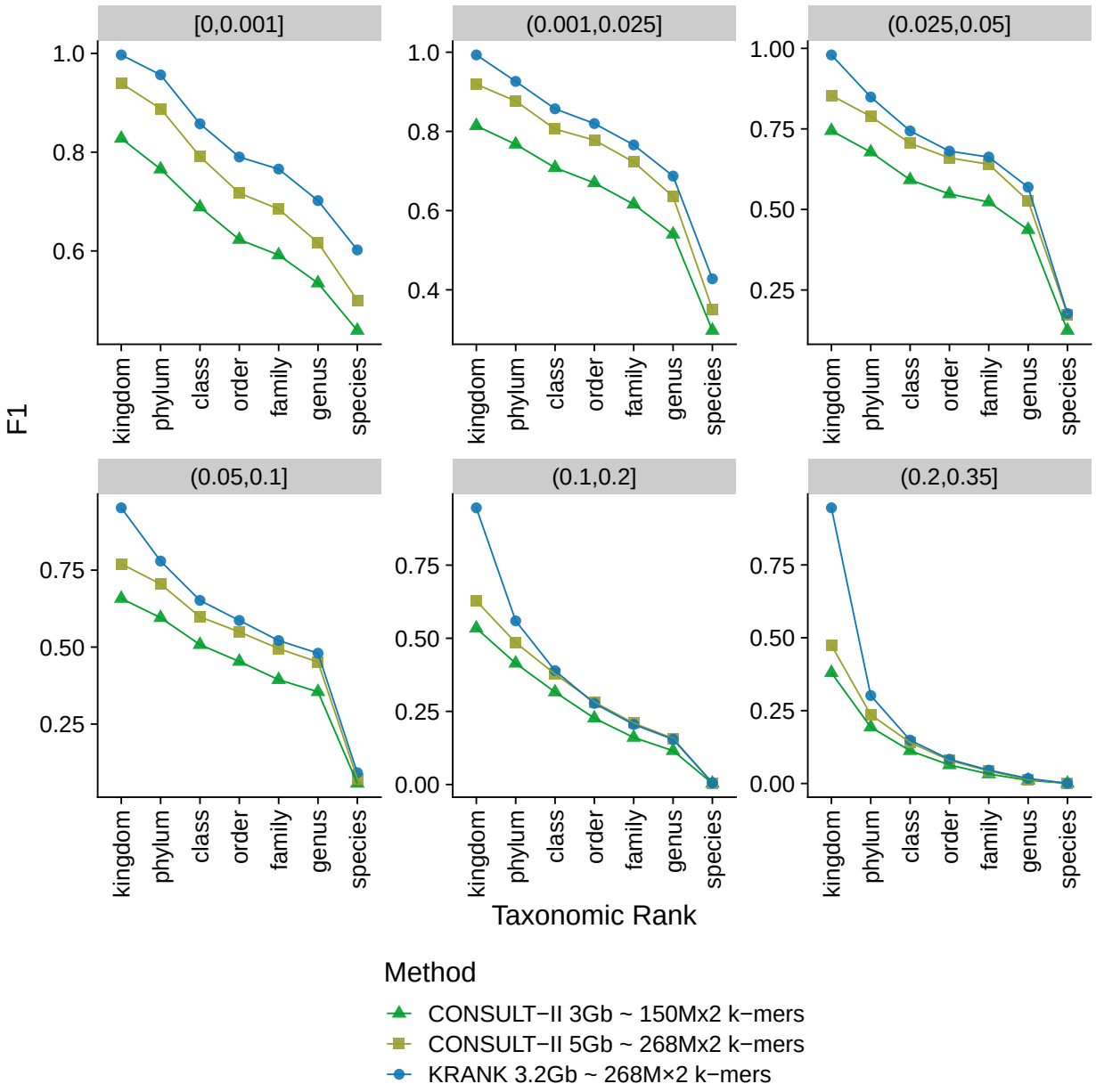

Figure S7: Comparison of CONSULT-II and KRANK in cases with the same memory level and the same number of k-mers in the library. KRANK can store 268M k-mer in a single table using less memory (1.8), but the selected k-mer set further improves the performance regardless of the memory. F1 scores are reported across all ranks as averages of all queries in each novelty bin.