Combinatorial Algorithms for Strain Level Metagenomic Microbial Detection and Quantification

Kaiyuan Zhu[1,2], Junyan Xu[2], A. Funda Ergun[1], Yuzhen Ye[1], and S. Cenk Sahinalp[2]

[1]Department of Computer Science, Indiana University, Bloomington, IN, USA,
[2]Cancer Data Science Laboratory, National Cancer Institute, National Institutes of Health, Bethesda, MD, USA

Abstract

Identifying and quantifying the microbial composition of a complex biological or environmental sample is one of the primary challenges in microbiology. Many software tools have been developed to classify metagenomic sequencing reads originating from a mixture of bacterial or viral genomes, and to estimate the microbial abundance profile of the mixture. Unfortunately the accuracy of these tools significantly degrade in the presence of large portions of shared content among the genomes in the mixture or the genomic database in use. Here we introduce `CAMMiQ`, a novel combinatorial solution to the microbial identification and abundance estimation problem, which improves all available tools with respect to the number of correctly classified reads (i.e., specificity) by an order of magnitude and resolves possible mixtures of similar genomes, possibly at the strain level. The key contribution of `CAMMiQ` is its use of arbitrary length, doubly-unique substrings, i.e. substrings that appear in exactly two genomes in the input database, instead of fixed-length, unique substrings. In order to resolve the ambiguity in the genomic origin of doubly-unique substrings, `CAMMiQ` employs a combinatorial optimization formulation, which can be solved surprisingly quickly. `CAMMiQ`'s index consists of a sparsified subset of the shortest unique and doubly-unique substrings of each genome in the database, within a user specified length range and as such it is fairly compact. In short, `CAMMiQ` offers more accurate genomic identification and abundance estimation than the best known k-mer based and marker gene based alternatives through the use of comparable computational resources.

Availability: `https://github.com/algo-cancer/CAMMiQ`

1 Introduction

Advances in high throughput sequencing (HTS) have made it possible to generate millions of short reads from a metagemonic sample in a few hours. It is many times possible to identify the microbial species present in the sample by searching each of these reads in a database of reference genomes for a significant match. In the early days of metagenomics, reads were typically searched primarily in GenBank [1] through BLAST [2]. Unfortunately, the recent growth of HTS data and reference databases has made read search and alignment using BLAST computationally infeasible. As a result, a number of novel computational methods have been developed with the goal of identifying as well as quantifying species in a metagenomic sample faster (see [3] for a summary of algorithmic approaches used for these purposes).

Among the available computational methods, some aim to use reduced size databases to achieve speed up in read search. In particular, these methods may align reads only to *marker genes*, a relatively small collection of clade-specific, single-copy genes, instead of the full reference genomes [4, 5], through the use of available read mapping techniques. Note that these methods need to obtain the collection of marker genes by employing information beyond what is offered by the database itself - which is not always possible. Furthermore, since these methods identify and use only a handful of marker genes on each genome, many of the reads in the HTS data can not be utilized, implying low specificity. As a consequence, species with low abundance within the sample may not be easily identifiable (let alone correctly quantifiable) because variation in HTS coverage may result in a few or no reads originating from the marker genes, leading to many false negatives.

Another set of metagenomic HTS analysis methods are "alignment-free". There have been alignment free methods for string comparison for a long time [6, 7, 8] and they have found applications in bioinformatics workflows before the emergence of HTS [9, 10]. Applications of alignment-free methods to metagenomic HTS data typically rely on k-mer "matches" to return a taxonomic assignment for every read. These applications either assign a read to the lowest taxonomic rank possible (determined by the specificity of the read's k-mers) [11, 12, 13, 14], or to a pre-determined taxonomic level (typically species or genus levels) [15, 16]. In contrast to marker gene based methods, k-mer based applications can usually assign the label of the relevant taxonomic rank to each given read. As the value of k becomes larger, more reads can be assigned a unique label at the desired taxonomic rank, however, with growing k, the space requirement of these methods grows very quickly - which, in the worst case, can imply a factor k increase on the space requirements of the original database. The large memory footprint maintaining the entire k-mer profile of each species can be reduced through hashing or subsampling the k-mers [17, 18]; however this would result in loss of accuracy. In addition to methods based on exact k-mer counts, it is also possible to assign metagenomic reads to bacterial genomes by employing species specific sequence features (e.g. short k-mer distribution or GC content) [19, 20, 21, 22, 23], although methods that employ this approach are typically not very accurate at species or strain level assignment.

Note that neither the alignment free, k-mer based methods, nor the marker-gene based tools take into consideration the distribution of the reads over the genome of the species they are assigned. In fact, the alignment free methods do not even take into account the length of the genomes, but are based on (unnormalized) read counts. In reality, the distribution of reads generated by current HTS techniques from a given species should be roughly uniform. This principle is fundamental to a number of *isoform abundance estimation* methods, which aim to solve a very similar problem [24, 25, 26]. Interestingly, this constraint is under utilized in the context of metagenomic analysis. One exception is the network flow based approach utilized by, e.g. [27], which (implicitly) establishes a reference guided assembly of the reads into the genomes of the species involved. As such it is quite accurate but is very slow. Another method in this direction considers the uniformity of coverage across k-mers with each genome to reduce false positive calls [28], which improves the running time at a moderate loss of accuracy.

In contrast to the metagenomic species identification and quantification methods summarized above, there are also tools to determine the likely presence of a long genomic sequence (e.g. the complete or partial genome of a bacterial species) in a given metagenomic sample [29, 30, 31, 32]. Even though these tools solve an entirely different problem, methodologically they are similar to the k-mer based species identification and quantification tools such as [12, 16] in the sense that they build a succinct index on the database (this time comprised of the the metagenomic read collection) and query this index without explicit alignment. And because of their design parameters, these tools can not perform abundance estimation for a given species.

Our Contributions. In this paper we describe `CAMMiQ` (Combinatorial Algorithms for Metagenomic Microbial Quantification), a new computational method to maintain/manage a collection of m (bacterial) genomes $\mathcal{S} = \{s_1, \ldots, s_m\}$, each assembled into one or more strings/contigs, representing a species, a particular strain of a species, or any other taxonomic rank. `CAMMiQ`'s data structure can answer queries of the following form: given a query set $\mathcal{Q}$ of HTS reads obtained from a mixture of genomes, each from $\mathcal{S}$, identify the genomes in $\mathcal{Q}$, and compute their relative abundance. Our data structure is not only very efficient in terms of its querying time, but is also shown to be very accurate, through simulations. The key novel feature of our data structure is its utilization of substrings which are present in at most c genomes ($c > 1$) in $\mathcal{S}$. There are alignment free methods that utilize unique k-mers in genomes for metagenomic analysis [12, 16, 28] already. However our data structure is the first to consider those substrings that are present in two or possibly more genomes, for increasing the proportion of reads it can utilize and thus improving sensitivity. Because of this novel feature, our data structure can accurately identify genomes at subspecies/strain level.

In order to assign each read in $\mathcal{Q}$ that includes an "almost-unique" substring (i.e. present in at most c genomes) to a genome, our data structure solves an integer linear program (ILP) - that simultaneously infers which genomes are preset in $\mathcal{Q}$ and their relative abundances. Specifically, the objective of the ILP is to identify a set of genomes, in each of which the coverage of the almost-unique substrings is (approximately) uniform.

One novel feature of our data structure is its use of *shortest* substrings (present in at most c genomes) - rather than fixed length "k-mers". This feature also increases the number of reads utilized by our data

structure since some reads may include no almost-unique k-mer but instead longer substrings that are almost-unique. It also reduces the number of substrings and thus features (not to be confused with the number of reads) to be handled by the ILP: this is because, rather than considering multiple overlapping (almost-)unique k-mers, our data structure uses a single shorter substring shared by all of them. The ILP formulation further reduces the number of almost-unique substrings it considers by maintaining a maximally sparsified set of substrings which guarantee any potential read from a genome in $\mathcal{S}$ that is almost-unique includes one such substring. In the remainder of the paper, we set the value of c to 2, so as to consider not only unique but also "doubly-unique" substrings of the genomes in $\mathcal{S}$. Even though this choice turned out to be sufficiently powerful for the datasets we experimented with, it is easy to generalize our approach to $c > 2$ (i.e. to triply-unique, etc. substrings).

As a final contribution, we provide sufficient conditions to identify and quantify genomes in a query correctly, through the use of unique substrings/k-mers, provided the reads are error free. Although this is a purely theoretical result, to the best of our knowledge it is the first of its kind for metagenomic data analysis, and is valid for `CAMMiQ` for the case $c = 1$ and other unique substring based methods such as CLARK and KrakenUniq. `CAMMiQ`'s use with $c = 2$ is primarily advised for cases where these conditions are not satisfied.

2 Algorithmic Formulation

The input to `CAMMiQ` is a set of m genomes $\mathcal{S} = \{s_i\}_{i=1}^{m}$, not necessarily all from the same taxonomic level (each genome here may be associated with a genus, species, subspecies or strain) to be indexed. Although we describe `CAMMiQ` for the case where each $s_i \in \mathcal{S}$ is a single string, we do not assume that the genomes are fully assembled into a single contig; rather the string representing a genome could simply be a concatenation of all contigs from species i and their reverse complements with a special symbol $\$_i$ between each contig. We call $\mathcal{S}$ the input *database* and $i \in \{1, \cdots, m\}$ the *genome ID* of string s_i.

A query for `CAMMiQ` involves a set of reads $\mathcal{Q} = \{r_j\}_{j=1}^{n}$ representing a metagenomic mixture. For simplicity we describe `CAMMiQ` for reads of length L, however our data structure can handle reads of varying length. Given $\mathcal{Q}$, the goal of `CAMMiQ` is to identify a set of genomes $\mathcal{A} = \{s_1, \cdots, s_a\} \subset \mathcal{S}$ and their respective abundances $p_1, \cdots, p_a$ which "best explain" $\mathcal{Q}$. This is achieved by assigning (select) reads r_j to genomes s_i such that the resulting "normalized coverage" p_i of each genome $s_i \in \mathcal{A}$ is uniform across s_i.

In its simplest form, `CAMMiQ` builds a succinct index for input database $\mathcal{S}$, so as to handle queries in the following form. Given $\mathcal{Q} = \{r_1, \ldots r_n\}$ find the smallest set of genomes $\mathcal{A}_1 = \{s_1, \cdots, s_{a_1}\} \subseteq \mathcal{S}$, such that the set of *unique* L-mers in $\mathcal{A}_1$ includes all reads in $\mathcal{Q}$ that are identical to a unique L-mer in $\mathcal{S}$. For any $s_i \in \mathcal{S}$, call an L-mer in s_i unique in $\mathcal{S}$ if it does not appear in any other $s_{i'}$. We denote by $U_{i,L}$, the set of all unique L-mers in s_i. Then $\mathcal{A}_1 = \{s_i \in \mathcal{S} \mid \mathcal{Q} \cap U_{i,L} \neq \emptyset\}$.

The above query type may not be powerful enough to identify in $\mathcal{Q}$ those genomes in $\mathcal{S}$ that are very similar (with respect to sequence composition) to other genomes and thus do not include many unique L-mers. Such a genome s_i would be especially problematic if it is low in abundance, since the chances of $\mathcal{Q}$ to include any unique L-mer from $U_{i,L}$ will be low. As a solution to this problem, `CAMMiQ` also features a second type of query, which is more general since it involves both unique and *doubly-unique* L-mers of genomes s_i. We call an L-mer doubly-unique in $\mathcal{S}$ if it appears in exactly two genomes s_i and $s_{i'}$ and denote the set of doubly-unique L-mers in s_i by $D_{i,L}$. Given query set $\mathcal{Q}$, `CAMMiQ`'s more general query asks to compute $\mathcal{A}_2$, the smallest subset of $\mathcal{S}$ which include all reads in $\mathcal{Q}$ identical to unique and doubly-unique L-mers in $\mathcal{S}$. Note that, necessarily $\mathcal{A}_1 \subseteq \mathcal{A}_2$. Also note that, by involving doubly-unique L-mers, this more general query will have a chance to capture those genomes in $\mathcal{Q}$ which have low abundance and have a few unique L-mers.

Unfortunately, the index structure necessary to support the second type of queries is larger than that for the first type of queries. Furthermore, this query type could still produce inaccurate results in the presence of read errors. For handling read errors (at noise rates commonly observed in Illumina data) `CAMMiQ` finally features a third type of query, which is even more general since it involves the "shortest" unique and doubly-unique substrings of reads in $\mathcal{Q}$. We call a substring of a genome $s_i \in \mathcal{S}$ a shortest unique substring, if it is unique to s_i, has length in the range $[L_{\min}, L_{\max}]$, and has no substring that is a shortest unique substring; this definition can be extended to shortest doubly-unique substrings as well. This query thus asks to compute the smallest subset $\mathcal{A}_3$ of $\mathcal{S}$ which include these substrings, with the constraint that the "coverage" of these substrings in each genome $s_i \in \mathcal{A}_3$ is "uniform". The query also asks to compute the relative abundance of

each genome in $\mathcal{A}_3$ as will be described later.

By considering shorter unique substrings, our most general query has a higher chance of observing them within $\mathcal{Q}$ (since a substring of length $L' < L$ is included in $L - L' + 1$ L-mers in the genome). This query may also imply that a smaller fraction of unique or doubly-unique substrings in $\mathcal{Q}$ would be subject to read errors (since it is likely that each read may include more than one such unique substring and it is likely that at least one such unique substring will be error free). Furthermore the index structure necessary to maintain the entire set of shortest unique and doubly-unique substrings of genomes in $\mathcal{S}$ is fairly large. As a result of this, we build the index on a maximally sparsified set of shortest unique and doubly-unique substrings of each $s_i \in \mathcal{S}$ which ensure that each read in $\mathcal{Q}$ that can be attributed to at most two genomes includes at least one such substring (see the Supplementary Methods for more details).

An index on this sparsified set of shortest unique and doubly-unique substrings is sufficiently powerful for `CAMMiQ` to answer all three types of queries, i.e. it can efficiently compute the sets $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}_3$. For all three query types `CAMMiQ` first identifies for each read r_j all unique and doubly-unique substrings it includes; it then assigns r_j to the one or two genomes from which these substrings can originate from. To compute $\mathcal{A}_1$ `CAMMiQ` can simply return the collection of genomes which receive at least one read assignment. To compute $\mathcal{A}_2$ `CAMMiQ` needs to solve the "set cover" problem, or more precisely, its dual, the "hitting set" problem where genomes form sets and indexed strings that appear in query reads form the items to be covered. To compute $\mathcal{A}_3$ `CAMMiQ` solves the combinatorial optimization problem that asks to minimize the variance among the number of reads assigned to each indexed substring of each genome - the solution indicates the set of genomes in $\mathcal{A}_3$ along with their respective abundances.

Details on the composition as well as the construction process for `CAMMiQ`'s index are discussed in Section 2.1. The query processing of `CAMMiQ`, which proceeds in two stages are discussed in Sections 2.2 and 2.3: The first stage assigns reads to specific genomes - which is sufficient for computing sets $\mathcal{A}_1$ and $\mathcal{A}_2$. See Section 2.2 for the criteria we use for assigning a read to a genome, based on the indexed substrings it includes. The second stage introduces the combinatorial optimization formulation to compute $\mathcal{A}_3$ as a response to the most general query type. See Section Section 2.3 for details.

2.1 Index Construction

In order to respond to the three types of queries described above, we preprocess unique and doubly-unique substrings of genomes in $\mathcal{S}$ to form an index structure as follows. Let $\mathcal{U} = \cup_{i=1}^{m} \mathcal{U}_i$ and $\mathcal{D} = \cup_{i=1}^{m} \mathcal{D}_i$ where $\mathcal{U}_i$ and $\mathcal{D}_i$ are the substrings from genome s_i whose lengths are within the range than $[L_{\min}, L_{\max} \leq L]$, such that each $u \in \mathcal{U}$ is present in at most one genome and each $d \in \mathcal{D}$ is present in at most two genomes in $\mathcal{S}$. See below for a detailed definition for the uniqueness of a substring. If our goal is to compute $\mathcal{A}_1$ or $\mathcal{A}_2$ only (i.e. respond to the first and second type of queries), we can simply set $L_{\min} = 0, L_{\max} = L$ and maintain the corresponding substring collections $\mathcal{U}$ and $\mathcal{D}$ in our index. For computing $\mathcal{A}_3$ (i.e. responding to the third type of queries - in addition to the first and second types), we actually use the length constrained definitions of substrings for computing $\mathcal{U}$ and $\mathcal{D}$. As mentioned earlier, we then sparsify $\mathcal{U}$ and $\mathcal{D}$ as much as possible by maintaining only one representative substring among those that are in close proximity within a genome, and discarding the rest (see "Subsampling unique substrings" below for details). We start with formal definitions and some notation.

Notation and definitions. Let $s = s_1\$_1 \circ \cdots \circ s_m\$_m$ denote the string obtained by concatenating the input reference genomes $s_i \in \mathcal{S}$ and let $M = |s| = \sum_i |s_i|$ denote its length. A substring of s is a string in the form $s[l..r] = s[l]s[l+1]\cdots s[r]$. With a slight abuse of notation we denote by $s_i[l..r]$ not the actual substring of s_i including its l^{th} to r^{th} symbols, but rather the substring of s including its l^{th} to r^{th} symbols, with the provision that all these symbols are within the representation of s_i in s. We denote by an ℓ-mer a string of length ℓ. The suffix of s that starts at position i is denoted $\texttt{suf}[i] = s[i, \cdots, M]$. In what follows, we use the *generalized enhanced suffix array* of s which is composed of three parts. (i) The *suffix array* `SA` of s, which is comprised of the positions $1, 2, \cdots, M$, sorted in increasing lexicographical order of the corresponding suffixes $\texttt{suf}[i]$, $i = 1, 2, \cdots, M$. That is, $\texttt{SA}[i] = j$ indicates that $\texttt{suf}[j]$ is the i-th smallest suffix in lexicographical order. In addition, we denote $\texttt{SA}^{-1}[j] = i$ if $\texttt{SA}[i] = j$. (ii) The *longest common prefix array*, `LCP`, contains in its i-th position the length of the longest common prefix of $\texttt{suf}[\texttt{SA}[i]]$ and $\texttt{suf}[\texttt{SA}[i-1]]$, for $2 \leq i \leq M$ (and $\texttt{LCP}[1] = 0$). (iii) Finally, the *generalized suffix array* `GSA` contains the genome ID of each suffix $\texttt{suf}[\texttt{SA}[i]]$. It

is well known that all of the above arrays can be constructed in linear time. The first linear time constructed data structure that can determine whether a given substring is unique to a "document" (i.e. a genome in our context) in a collection of documents, and compute the shortest unique substring of a document that ends in a particular position, in time proportional to the substring length is the augmented suffix tree of Matias et al. [33]. Once an augmented suffix tree is computed, it can be trivially reduced to the above described enhanced suffix array in $O(M)$ time - which can also be constructed without the use of suffix trees to achieve a constant factor improvement in memory [34].

We denote by $u_i(l,r)$ the substring $s_i[l..r]$ that is *unique* to genome s_i; formally, this indicates that there exists no substring $u_j(l',r')$ on any genome $s_j \neq s_i$ such that $u_i(l,r) = u_j(l',r')$. We call $u_i(l,r)$ an shortest unconstrained unique substring, if none of its substrings are unique. Similarly, we denote by $d_i(l,r)$ the substring $s_i[l..r]$ that is *doubly-unique* to genome s_i and one other genome, say s_j; formally, this indicates that there is exactly one genome s_j which includes the substring $d_j(l',r')$, i.e., $s_i[l..r] = s_j[l'..r']$ for some l', r'. Clearly, any superstring of a unique substring is still unique and any superstring of a doubly-unique substring is either unique or doubly-unique. We call $d_i(l,r)$ a shortest unconstrained doubly-unique substring of s_i and some other genome s_j, if none of its substrings are doubly-unique.

For our purposes we need to constrain the shortest unique and doubly-unique substrings with length upper bound $L_{\max}$ and lower bound $L_{\min}$. Under these constraints, we call any shortest unconstrained unique substring $u_i(l,r)$ a shortest unique substring if $L_{\max} \geq r - l + 1 > L_{\min}$. We also call a unique substring $u_i(l,r)$ a shortest unique substring if $r-l+1 = L_{\min}$. Similarly, we call any shortest unconstrained doubly-unique substring $d_i(l,r)$ a shortest doubly-unique substring if $L_{\max} \geq r - l + 1 > L_{\min}$. Again, we call a doubly-unique substring $u_i(l,r)$ a shortest doubly-unique substring if $r - l + 1 = L_{\min}$ as well. We say an L-mer $s_i[l..l+L-1]$ includes a unique substring $s_i[l'..r']$, or, conversely, a unique substring $s_i[l'..r']$ covers an L-mer $s_i[l..l+L-1]$ if $l' \geq l$ and $r' \leq l+L-1$. As such, we call an L-mer unique if it includes a unique substring. We can generalize these definitions to the notion of an L-mer including a doubly-unique substring, or conversely, a doubly-unique substring covering an L-mer, and thus making the L-mer itself doubly-unique - provided that it is not unique.

Algorithmic framework to compute shortest unique substrings. It is quite simple to compute the shortest unique and doubly-unique substrings in $\mathcal{S}$ in $O(M)$ time by using the augmented suffix tree described in [33]. A similar running time can also be achieved through the use of a suffix array, as discussed by [35] for a single document (i.e. genome). We slightly generalize this to handle multiple genomes as follows. The key observation we use is that given a position l, the shortest unique or doubly-unique substring of s_i that starts at l (i.e. $u_i(l,r)$ or $d_i(l,r)$) is the shortest unique, or respectively doubly-unique prefix of $\texttt{suf}[l]$. In this way the problem can be reduced to searching for the longest common prefix of $\texttt{suf}[l]$ with any other suffix from another genome (i.e., any genome with ID $\neq \texttt{GSA}[\texttt{SA}^{-1}[l]]$) for each $1 \leq l \leq M$. Specifically, we define

$$\texttt{LCP}_u[i] = \max_{1 \leq j \leq M; \texttt{GSA}[i] \neq \texttt{GSA}[j]} \text{lcp}(\texttt{suf}[\texttt{SA}[i]], \texttt{suf}[\texttt{SA}[j]]) \tag{1}$$

and

$$\texttt{LCP}_d[i] = \max_{1 \leq j \leq M; \texttt{GSA}[i] \neq \texttt{GSA}[j], \texttt{GSA}[j']} \text{lcp}(\texttt{suf}[\texttt{SA}[i]], \texttt{suf}[\texttt{SA}[j]]) \tag{2}$$

where $\texttt{suf}[j']$ is any suffix with $\texttt{GSA}[i] \neq \texttt{GSA}[j']$ that maximizes $\text{lcp}(\texttt{suf}[\texttt{SA}[i]], \texttt{suf}[\texttt{SA}[j']])$ (if there is more than one solution for j' then one of them can be chosen arbitrarily), where $\text{lcp}(x,y)$ denotes the longest common prefix of two suffices x and y. CAMMiQ maintains an array SU (of length M) such that $\texttt{SU}[r] = l$ if $u_i(l,r)$ is a shortest unique substring. In order to compute SU, each of its entries is initially set to 0 and for each $i = 1, \ldots M$, one entry of SU is updated as

$$\texttt{SU}[\texttt{SA}[i] + \texttt{LCP}_u[i]] \leftarrow \max\{\texttt{SU}[\texttt{SA}[i] + \texttt{LCP}_u[i]], \texttt{SA}[i]\} \tag{3}$$

Similarly, CAMMiQ maintains an array SD (again of length M) such that $\texttt{SD}[r] = l$ if $d_i(l,r)$ is a shortest doubly-unique substring. Again each entry of SD is initially set to 0 and then for each $i = 1, \ldots M$, one entry of SD is updated as

$$\texttt{SD}[\texttt{SA}[i] + \texttt{LCP}_d[i]] \leftarrow \max\{\texttt{SD}[\texttt{SA}[i] + \texttt{LCP}_d[i]], \texttt{SA}[i]\}. \tag{4}$$

See Supplementary Section 5.1 for further details.

Computing $\texttt{LCP}_u$ and $\texttt{LCP}_d$. Given a subarray $\texttt{GSA}[i_1, \cdots, i_2]$, let $d_{\texttt{GSA}}(i_1, i_2)$ be the number of distinct genomes the entries in this subarray belong to, i.e. $d_{\texttt{GSA}}(i_1, i_2) = |\{\texttt{GSA}[i_1], \cdots, \texttt{GSA}[i_2]\}|$. We can now compute $\texttt{LCP}_u$ and $\texttt{LCP}_d$ in linear time as follows.

$$\texttt{LCP}_u[i] = \max \begin{cases} \min\limits_{i^- < x \leq i} \texttt{LCP}[x], \text{ where } i^- = \max\{1 \leq i' < i\}, \text{ s.t. } d_{\texttt{GSA}}(i', i) \geq 2 \\ \min\limits_{i < x \leq i^+} \texttt{LCP}[x], \text{ where } i^+ = \min\{i < i' \leq M\}, \text{ s.t. } d_{\texttt{GSA}}(i, i') \geq 2 \end{cases} \tag{5}$$

$$\texttt{LCP}_d[i] = \min \begin{cases} \max \begin{cases} \min\limits_{i^- < x \leq i} \texttt{LCP}[x], \text{ where } i^- = \max\{1 \leq i' < i\}, \text{ s.t. } d_{\texttt{GSA}}(i', i) \geq 2 \\ \min\limits_{i < x \leq i^{2+}} \texttt{LCP}[x], \text{ where } i^{2+} = \min\{i < i' \leq M\}, \text{ s.t. } d_{\texttt{GSA}}(i, i') \geq 3 \end{cases} \\ \max \begin{cases} \min\limits_{i^{2-} < x \leq i} \texttt{LCP}[x], \text{ where } i^{2-} = \max\{1 \leq i' < i\}, \text{ s.t. } d_{\texttt{GSA}}(i', i) \geq 3 \\ \min\limits_{i < x \leq i^+} \texttt{LCP}[x], \text{ where } i^+ = \min\{i < i' \leq M\}, \text{ s.t. } d_{\texttt{GSA}}(i, i') \geq 2 \end{cases} \end{cases} \tag{6}$$

Note that to introduce a minimum length constraint $L_{\min}$ on unique and doubly-unique substrings, each $\texttt{LCP}_u[i]$ is (re)set to $\max\{L_{\min} - 1, \texttt{LCP}_u[i]\}$ and respectively each $\texttt{LCP}_d[i]$ is (re)set to $\max\{L_{\min} - 1, \texttt{LCP}_d[i]\}$. Then, to ensure that each shortest doubly-unique substring occurs in exactly two genomes (and not one), we set $\texttt{LCP}_d[i] = \infty$ in case the above procedure ends up with $\texttt{LCP}_d[i] = \texttt{LCP}_u[i]$. See Supplementary Section 5.2 for the proof of correctness and a running time analysis for the computation of $\texttt{LCP}_u$ and $\texttt{LCP}_d$.

Subsampling unique substrings. As defined above, the array SU maintains the indices of all shortest unique substrings; some of these substrings may have large overlaps with others and thus are redundant in assessing the uniqueness of a read in the query. Let $\mathcal{U}_i$ be the collection of all unique substrings on genome s_i. Then, in order to reduce the index size, CAMMiQ aims to compute a subset $\mathcal{U}'_i$ of $\mathcal{U}_i$, consisting of the minimum number of shortest unique substrings that cover every unique L-mer on s_i. CAMMiQ also aims to compute a subset $\mathcal{D}'_i$ of $\mathcal{D}_i$, consisting of the minimum number of shortest doubly-unique substrings that cover every doubly-unique L-mer on s_i. This is all done by greedily maintaining only the rightmost shortest unique or doubly-unique substring in any L-mer of a genome in $\mathcal{S}$. In the remainder of the paper we denote the number of unique substrings in subset $\mathcal{U}'_i$ by nu_i $(= |\mathcal{U}'_i|)$ and respectively the number of doubly-unique substrings in subset $\mathcal{D}'_i$ by nd_i $(= |\mathcal{D}'_i|)$; we denote the number of unique L-mers on s_i by nu_i^L and respectively the number of doubly-unique L-mers on s_i by nd_i^L. As we prove in Supplementary Section 5.3, this greedy strategy we employ can indeed obtain the minimum number of shortest unique substrings to cover each unique L-mer, provided that each substring in $\mathcal{U}_i$ occurs only once s_i.

Index Organization. Consider the set of unique substrings Let $h = \min_{u_i \in \mathcal{U}} |u_i|$ be the minimum length of all shortest unique substrings (h is automatically set to $L_{\min}$ if the minimum length constraint is imposed). CAMMiQ maintains a hash table which maps a distinct h-mer w to a bucket containing all unique substrings u_i which have w as a prefix. Within each bucket, the remaining suffices of all unique substrings u_i, i.e. $u_i[h+1..|u_i|]$, are maintained in a trie so that each leaf contains the corresponding genome ID. For each read r_j in the query, CAMMiQ considers each substring of length h and it's reverse complement and computes its hash value (in total time linear with L through Karp-Rabin fingerprinting). If the substring has a match in the hash table, then CAMMiQ tries to extend the match until a matching unique substring is found, or finds no match. (Note that the processing for doubly-unique substrings is identical to that for unique substrings.) See Figure 1 for an overview of the index structure. Also see Section 2.2 below for the use of unique and doubly-unique substrings identified for each read to answer the query.

2.2 Query Processing Stage 1: Preprocessing the Reads

Given the index structure on the sparsified set of shortest unique and doubly-unique substrings of genomes in $\mathcal{S}$, we handle each query $\mathcal{Q}$ in two stages. In Stage 1, we preprocess each read $r_j \in \mathcal{Q}$ as follows. Consider the set of genomes associated with each unique and doubly-unique subtring in the read r_j. If the intersection of these sets are empty we discard the read since these substrings are "conflicting" (see below for a short

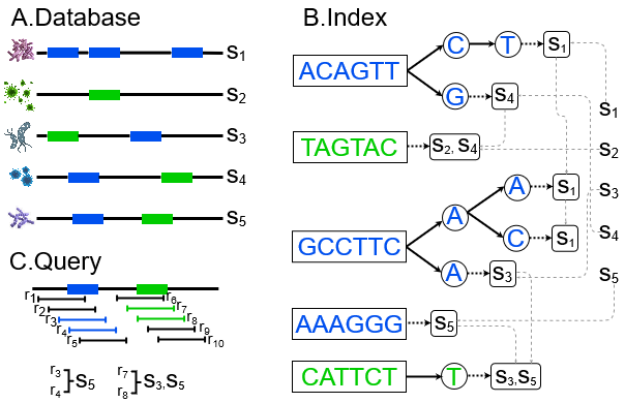

Figure 1: Overview of CAMMiQ's index structure. Strings in blue are unique substrings and those in green are doubly-unique substrings.

discussion on these conflicts). If the intersection is non-empty, for each unique substring u_i and each doubly-unique substring d_i in r_j, we increase the associated counter $c(u_i)$ and respectively $c(d_i)$ we maintain. These counters indicate the number of "conflict-free" reads that include each such substring in $\mathcal{Q}$. As described below, these counters will be essential to actual query processing in Stage 2.

1. Identify the set $U_j = \{u_{j,1}, \ldots u_{j,\ell}\} \subset \mathcal{U}$ of unique substrings in r_j; similarly identify the set $D_j = \{d_{j,1}, \ldots d_{j,\ell'}\} \subset \mathcal{D}$ of doubly-unique substrings in r_j.

2. (a) If $U_j = D_j = \emptyset$ then discard r_j.
   (b) If $U_j \neq \emptyset$ but it includes a pair of unique substrings $u_{j,i}$ and $u_{j,i'}$ which originate from different genomes, then discard r_j.
   (c) If $U_j \neq \emptyset$ and all its unique substrings originate from the same genome s_k, however D_j includes a substring $d_{j,i}$ which can not originate from s_k, then again discard r_j.
   (d) If $U_j = \emptyset$ and the intersection of the set of genomes from where the substrings in D_j can originate is empty, then also discard r_j.
   (e) If on the other hand,
      i. $U_j \neq \emptyset$, all its unique substrings originate from the same genome s_k, and each doubly-unique substring $d_{j,i'} \in D_j$ can originate from s_k, or
      ii. $U_j = \emptyset$, however the intersection between the genomes where the doubly-unique substrings of r_j can originate from is comprised of only one genome, s_k, or
      iii. $U_j = \emptyset$ and d_j is comprised of doubly-unique substrings that can only originate from the same pair of genomes s_k and $s_{k'}$, then

   then increase $c(u_{j,i})$ by 1 for each $u_{j,i} \in U_j$ and $c(d_{j,i})$ by 1 for each $d_{j,i} \in D_j$.

The above counters are sufficient to compute the set $\mathcal{A}_1$ as well as $\mathcal{A}_3$, the answer to our most general query type. For computing $\mathcal{A}_2$, CAMMiQ additionally maintains a counter $d(s_k, s_{k'})$ for each pair of genomes $s_k, s_{k'}$, indicating the number of reads in $\mathcal{Q}$ that can originate both from s_k and $s_{k'}$; the value of this counter needs to be increased for each pair of involved genomes by 1 in case (iii) above.

The preprocessing stage described above eliminates those reads that include conflicting unique or doubly-unique substrings - conflicting in the sense that they are associated with different genomes. There are two main reasons for observing such conflicts: (i) read errors, (ii) the presence of genomes in the query that are not in $\mathcal{S}$. By eliminating these conflicting reads, we reduce the chances of mis-identifying the genomes they may originate from.

In short, the read preprocessing stage produces two vectors $\mathbf{c}_i^u = (c(u_{i,1}), \cdots, c(u_{i,nu_i}))$ and $\mathbf{c}_i^d = (c(d_{i,1}), \cdots, c(d_{i,nd_i}))$ which indicate the number of (conflict-free) reads that include each unique and doubly-unique substring on each genome s_i. One can use these vectors to compute $\mathcal{A}_1 = \{s_i : \sum_{l=1}^{nu_i} c(u_{i,l}) > 0\}$.

Additionally, through the use of the counter $d(s_k, s_{k'})$ maintained for each pair of genomes $s_k, s_{k'}$ one can compute $\mathcal{A}_2 = \arg\min |\mathcal{A}' \subset \mathcal{S}|$ such that (i) $s_i \in \mathcal{A}'$ if $\sum_{l=1}^{nu_i} c(u_{i,l}) > 0$ and (ii) $\exists s_i \in \mathcal{A}'$, if $d(s_k, s_{k'}) > 0$ then either $i = k$ or $i = k'$. This is basically the solution to the hitting set problem we mentioned earlier, whose formulation as an integer linear program (ILP) is well known [36]. From this point on, our main focus will be computing $\mathcal{A}_3$ (as well as the relative abundances of the genomes in $\mathcal{A}_3$) for which we introduce an ILP formulation described below.

2.3 Query Processing Stage 2: ILP Formulation

`CAMMiQ` computes the list of genomes in the query as well as their abundances through an integer linear program (ILP) described below. Let $\delta_i = 0/1$ be the indicator for the absence or presence of the genome s_i in $\mathcal{Q}$. The ILP formulation assigns a value to each δ_i and also computes for each s_i its abundance p_i in the range $[p_{\min}, p_{\max}]$.

Minimize

$$\sum_i \left(\frac{1}{nu_i}\sum_{l=1}^{nu_i} |c(u_{i,l}) - e(u_{i,l})| + \frac{1}{nd_i}\sum_{l=1}^{nd_i} |c(d_{i,l}) - e(d_{i,l})|\right)$$

Subject to

$$e(u_{i,l}) = (L - |u_{i,l}|) \cdot p_i \cdot \frac{1}{L} \cdot (1 - \hat{\text{err}})^{|u_{i,l}|} \qquad \forall i, l, \text{ s.t. } 1 \le l \le nu_i \tag{7}$$

$$e(d_{i,l}) = (L - |d_{i,l}|) \cdot (p_i + p_j) \cdot \frac{1}{L} \cdot (1 - \hat{\text{err}})^{|d_{i,l}|} \qquad \forall i, l \text{ s.t. } 1 \le l \le nd_i \tag{8}$$

$$p_i \le \delta_i \cdot p_{\max} \qquad \forall i \tag{9}$$

$$\delta_i = 0 \qquad \forall i, \text{ s.t. } s_i \notin E(\mathcal{Q}) \tag{10}$$

$$p_i \ge \delta_i \cdot \min\{L \cdot \sum_{l=1}^{nu_i} c(u_{i,l}) \cdot \frac{1}{nu_i^L}, L \cdot \sum_{l=1}^{nd_i} c(d_{i,l}) \cdot \frac{1}{nd_i^L}\} \cdot (1 - \epsilon) \qquad \forall i, \text{ s.t. } s_i \in E(\mathcal{Q}) \tag{11}$$

$$\sum_i |s_i| \cdot p_i \le n \cdot L \tag{12}$$

The objective of the ILP is to minimize the sum of absolute difference between the expected and the actual number of read to cover a unique or doubly-unique substring. Since each genome may have different number of unique and doubly-unique substrings, this difference is normalized w.r.t. nu_i or nd_i. Constraints (7) and (8) define the expected number of reads to cover a particular unique substring $u_{i,l}$ or doubly-unique substring $d_{i,l}$ - here p_j is the abundance of genome s_j which also includes $d_{i,l}$. Here we denote by $\hat{\text{err}}$ the estimated read error (specifically substitution) rate per nucleotide, and denote by $|w|$ the length of a substring w. Constraint (9) ensures that the abundance p_i of a genome is 0 if $\delta_i = 0$. Constraint (10) ensures that the solution to the above ILP excludes those genomes whose counters for unique and doubly-unique substrings add up to a value below a threshold - so as to reduce the size of the solution space. More specifically, given a threshold value α, the constraint excludes those genomes s_i which are not in the set of genomes $E(\mathcal{Q})$ whose counters for its unique substrings add up to a value above $\alpha \cdot nu_i^L$, and doubly unique substrings add up to a value above $\alpha \cdot nd_i^L$. More formally, $E(\mathcal{Q}) = \{s_i \in \mathcal{S} \mid \sum_{l=1}^{nu_i} c(u_{i,l}) \ge \alpha \cdot nu_i^L\} \cap \{s_i \in \mathcal{S} \mid \sum_{l=1}^{nd_i} c(d_{i,l}) \ge \alpha \cdot nd_i^L\}$. Constraint (11) enforces a lower bound on the coverage (and thus the abundance) of each genome s_i in the solution to the above ILP (namely, with $\delta_i = 1$), which must match the coverage ($L \cdot \sum_{l=1}^{nu_i} c(u_{i,l}) \cdot \frac{1}{nu_i^L}$ and $\sum_{l=1}^{nd_i} c(d_{i,l}) \cdot \frac{1}{nd_i^L}$) resulted from the number of reads in $\mathcal{Q}$ that include a unique and doubly-unique substring respectively, i.e., it must be at least $(1 - \epsilon)$ times the smaller one above for a user defined ϵ. Constraint (12) enforces an upper bound on the coverage (and thus the abundance) of each genome s_i in the solution to the above ILP, through making the sum (over each s_i) of the number of reads produced on s_i based on p_i not exceed the total number of reads n. Collectively, the last two constraints ensure that the abundance p_i computed from the ILP matches what is (i.e., the coverage based on read counts)

given by $\mathcal{Q}$. Note that the absolute values in the objective can be removed by introducing a new variable $d(u_{i,l}) \geq \max\{c(u_{i,l}) - e(u_{i,l}), e(u_{i,l}) - c(u_{i,l})\}$.

2.4 When to use Unique Substrings - the Error Free Case

We now provide a set of sufficient conditions to guarantee the (approximate) performance that can be obtained (with high probability) in metagenomic identification and quantification by the use of unique substrings only. These conditions apply to CAMMiQ when $c = 1$, as well as CLARK, KrakenUniq and other similar approaches. In case these conditions are NOT met, it is advisable to use CAMMiQ with $c \geq 2$.

Suppose that we are given a query $\mathcal{Q}$ composed of n error free reads of length L, sampled independently and uniformly at random from a collection of genomes $\mathcal{A} = \{s_1, \cdots, s_a\}$ according to their abundances $p_1, \cdots, p_a$. More specifically, suppose that our goal is to answer query $\mathcal{Q}$ by computing $\mathcal{A}_1$, along with an estimate for the abundance value p_i for each $s_i \in \mathcal{A}_1$, calculated as the weighted number of reads assigned to s_i according to the procedure described in Section 2.2. Then, the L1 distance between the true abundance values and this estimate will not exceed a value determined by n (number of reads), a, and $q_{\min}$, the minimum (normalized) proportion of unique L-mers among these genomes. For a given failure probability δ and an upper bound on L1 distance ϵ, this translates into sufficient conditions on the values of n, a and $q_{\min}$ to ensure acceptable performance by the computational method in use.

Theorem 1. *Let $\mathcal{Q} = \{r_1, \cdots, r_n\}$ be a set of n error-free reads of length L, each sampled independently and uniformly at random from all positions on a genome $s_i \in \mathcal{A} = \{s_1, \cdots, s_a\}$, where $s_1, \cdots, s_a$ is distributed according to their abundances $p_1, \cdots, p_a > 0$. Let $p'_i = \frac{p_i \cdot n_i^L}{\sum_{i'=1}^{a} p'_{i'} \cdot n_{i'}^L}$ be the corresponding "unnormalized" abundance of p_i for $i = 1, \cdots, a$, where n_i^L denotes the total number of L-mers on s_i. Let $q_1, \cdots, q_a > 0$ be the proportion of unique L-mers on $s_1, \cdots, s_a$ respectively; $p_{\min} = \min\{p_1, \cdots, p_a\}$; $q_{\min} = \min\{q_1, \cdots, q_a\}$. Then,*

- *(i) With probability at least $1 - \delta$, each s_i can be identified through querying $\mathcal{Q}$ if $n \geq \frac{2(a+1)+\ln(1/\delta)}{(p_{\min} q_{\min})^2}$*
- *(ii) With probability at least $1 - \delta$, the L1 distance between the predicted abundances $\hat{p}_1, \cdots, \hat{p}_a$ by setting $\hat{p}_i = \frac{c_i/q_i}{n}$ and the true (unnormalized) abundances $p'_1, \cdots, p'_a$ is at most ϵ if $n \geq \frac{2(a+1)+\ln(1/\delta)}{(\epsilon q_{\min})^2}$.*
- *(iii) Given n such reads in a query, with probability at least $1-\delta$, the L1 distance between the predicted abundances $\hat{p}_1, \cdots, \hat{p}_a$ by setting $\hat{p}_i = \frac{c_i/q_i}{n}$ and the true (unnormalized) abundances $p'_1, \cdots, p'_a$ is bounded by $\sqrt{\frac{2[\ln(1/\delta)+(a+1)]}{n q_{\min}^2}}$.*

where c_i denotes the number of reads assigned to s_i.

See Supplementary Methods for a proof.

3 Results

In order to benchmark the overall running time performance, memory utilization and accuracy of CAMMiQ, we have created a large, "species-level" dataset consisting of all complete bacterial genomes in NCBI's RefSeq [37] Database (downloaded on 06/16/2019). We (randomly) selected one representative genome per species out of 13737 reference genomes representing 4122 distinct species. This resulted in a total of $m = 4122$ genomes with a total length of $M = 3.4 * 10^{10}$ (including the reverse complement contigs). On this bacterial genome dataset, CAMMiQ was compared against the state of the art k-mer based and marker-gene based metagenomics analysis tools, specifically Kraken2 [38] (the latest version of Kraken [12]), KrakenUniq [28], CLARK [16] and MetaPhlAn2 [39] (these four tools provide very similar functionality to CAMMiQ such as read level classification and abundance estimation).

In addition, we composed a smaller, "strain-level" dataset consisting only of 614 Human Gut related genomes according to [40] to evaluate CAMMiQ's strain level identification and quantification performance. All of our experiments were run on a Linux server equipped with 40 Intel Xeon E7-8891 2.80 GHz processors,

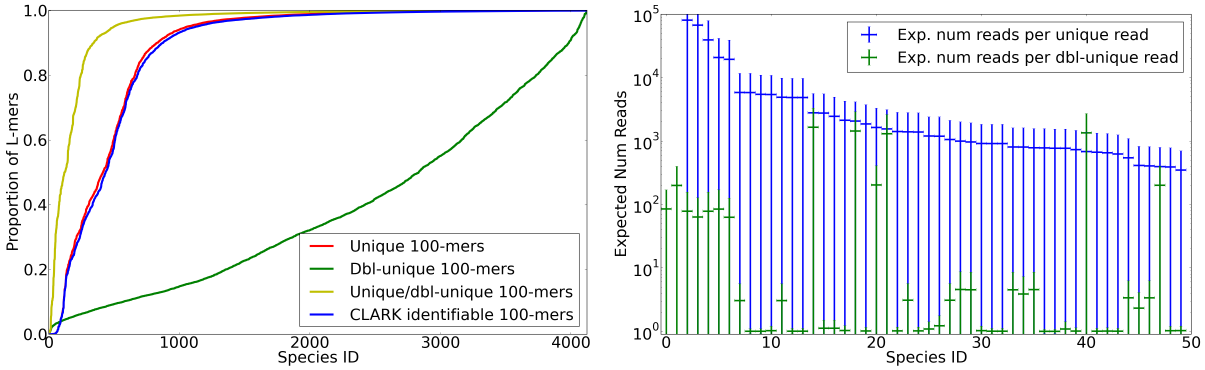

Figure 2: (A) Proportion of L-mers (for $L = 100$) that include a unique substring (plotted in red), a doubly-unique substring (plotted in green), or either a unique or a doubly-unique substring (plotted in yellow) in the large dataset of 4122 bacterial genomes from NCBI RefSeq database; these L-mers, when presented as reads are utilizable by `CAMMiQ` (when $L_{\max} = 100$). Also included are the proportion of L-mers that include a unique k-mer ($k = 30$) and thus are utilized by CLARK. For each plot, the genomes are independently sorted with respect to the corresponding proportionality in ascending order. (B) The expected number of reads (of length $L = 100$) needed to capture one read containing a unique substring (in blue) as well as one read containing a doubly-unique substring (in green) in the 50 genomes with the lowest proportion of unique L-mers. Also included are the range for each value within the corresponding standard deviation. Note that for this figure $L_{\max} = 50$.

with 2.5 TB of physical memory and 30 TB of disk space. The ILP solver used by `CAMMiQ` is IBM ILOG CPLEX 12.9.0.

Below, Section 3.1 demonstrates the advantage `CAMMiQ` can offer by utilizing doubly-unique, in addition to unique substrings, and considering shortest such substrings instead of fixed length k-mers as per CLARK. Section 3.2 gives an overview of the sets of queries we used to benchmark `CAMMiQ`'s performance. Section 3.3 and 3.4 then demonstrate `CAMMiQ`'s performance on these query sets on both our species-level and strain-level datasets, compared with the state of the art metagenomics analysis tools. Finally Section 3.5 compares the computational resources required by these tools.

3.1 Comparative Utility of Variable-Length and Doubly-Unique Substrings

First, we demonstrate the theoretical advantage offered by `CAMMiQ` in comparison to other tools, due to its unique utilization of not only unique k-mers (as per CLARK), but substrings of any length, which are unique or doubly-unique - within the species-level dataset we constructed. For that we compared the proportion of L-mers from each genome s_i in this dataset (for $L = 100$) that are unique or doubly-unique (and thus can be utilized by `CAMMiQ`) with the proportion of L-mers that include a unique k-mer (that can be utilized by CLARK) for $k = 30$.

Figure 2A summarizes our findings: on the horizontal axis, the genomes are sorted with respect to the proportion of unique and doubly-unique L-mers they have; the vertical axis depicts this proportionality (from 0.0 to 1.0). The four plots we have are for the proportion of unique L-mers, doubly-unique L-mers, the combination of unique and doubly-unique L-mers (all utilized by `CAMMiQ`), and the L-mers that include a unique k-mer (utilized by CLARK). As can be seen, roughly three quarters of all genomes in this dataset (and thus many of the bacterial species) are easily distinguishable since a large fraction of their L-mers include a unique k-mer. However, roughly a quarter of the genomes in this dataset can benefit from the consideration of doubly-unique substrings, especially when their abundances are low. In particular, 66 of these 4122 genomes/species have extremely low proportion of ($\leq 1\%$) unique 100-mers. In fact, the species *Francisella sp. MA06-7296* does not have a single unique 100-mer and the species *Rhizobium sp. N6212* does not have any 100-mer that include a unique 30-mer (in fact any substring of length $\leq L_{\max} = 50$). These two species cannot be identified by CLARK in any microbial mixture, independent of the abundance values.

Query Set	Related Dataset	Read Length (L)	Num. Reads (n)	Err Rate
Least-20-uniform-e.f.		100	4.8M	0
Least-20-uniform		100	4.8M	0.01
Least-quantifiable-20-uniform-e.f.		100	5.0M	0
Least-quantifiable-20-uniform	Selected	100	5.0M	0.01
Least-20-genera-uniform-e.f.	complete	100	4.0M	0
Least-20-genera-uniform	bacteria	100	4.0M	0.01
Least-20-genera-lognormal	genomes	100	4.0M	0.01
Random-20-uniform		100	4.4M	0.01
Random-20-lognormal		100	5.0M	0.01
Random-20-lognormal-a.g.		100	1.1M	0.01
Random-100-uniform		100	21.5M	0.006
HumanGut-least-25	Human Gut	100	2.0M	0.01
HumanGut-random-100-1	related	100	8.0M	0.01
HumanGut-random-100-2	bacteria	125	8.0M	0.01
HumanGut-all	genomes	100	20.0M	0.01

Table 1: Synthetic metagenomes generated to benchmark `CAMMiQ`'s performance against the best available metagenomic classification and abundance estimation tools. The top collection of queries involve genomes from our species-level dataset consisting of 4122 distinct bacterial species. The bottom collection were sampled from our strain-level dataset of 617 gastrointestinal associated bacteria (possibly incompletely assembled) provided in [40].

Figure 2B depicts the inverse proportionality of doubly-unique L-mers in comparison to unique L-mers among 50 genomes that have the lowest proportion of unique L-mers - for $L = 100$. The inverse-proportionality of unique or doubly-unique L-mers for a genome corresponds to the number of reads to be sampled (on average) from that genome to guarantee that the sample includes one read that is guaranteed to be assigned to the correct genome. In the absence of read errors, this guarantees the correct identification of the corresponding genome in the query. Note that, in half of these 50 genomes, almost all L-mers are doubly-unique. This implies that any query involving one or more of these genomes would unlikely to be resolved with CLARK. Yet this query could be handled by `CAMMiQ` since if a read contains a unique or doubly-unique substring, then it can be correctly assigned to the corresponding species as described in Section 2.2.

Note that 3296 of the 4122 species in our species-level dataset have $\geq 90\%$ of their 100-mers as unique. Only a few of these unique substrings do not include a unique 30-mer and thus will be missed by CLARK. This implies that from the accuracy point of view on these genomes, CLARK's use of k-mers instead of the shortest substrings does not put it at a disadvantage on these genomes - when $k = 30$ and $L = 100$. However, as it will become clear later (see Table 5) `CAMMiQ`'s use of shortest substrings, combined with its subsampling strategy gives it an advantage over CLARK (as well as KrakenUniq) with respect to the size of the index structure - despite the fact that `CAMMiQ` needs to index not only unique but also doubly-unique substrings from each genome.

3.2 Simulated Metagenomes for Querying `CAMMiQ`

In order to systematically compare the performance of `CAMMiQ` and other metagenomic profiling tools, we generated several simulated metagenomes as query sets, summarized in Table 1. The upper part of the table corresponds to simulated data from our species-level dataset and the lower part corresponds to our strain-level dataset.

The first set of simulated metagenomes aim to assess how well `CAMMiQ` identifies species in a query. For that we simulated a metagenome consisting of the 20 genomes that have the lowest number of unique L-mers in our species-level dataset. Each genome in the mixture was simulated to have similar read coverage. The very first query we generated from this mixture (denoted Least-20-uniform-e.f.) had no read errors. The second query (denoted Least-20-uniform) had i.i.d. substitution errors occurring at a rate of 1%. Note that

the 20 genomes we used in this mixture are intrinsically difficult to be identified by CLARK and other tools we compared. However since these genomes have many doubly-unique L-mers - which are sometimes shared with multiple other genomes, they could be identified by `CAMMiQ` (see Supplementary Methods and Figure 4 for a more detailed explanation).

The second set of metagenomes we simulated aim to assess the species-level quantification performance of `CAMMiQ`. This simulated metagenome consisted of the 20 genomes which are among the 50 genomes in our species-level dataset with the lowest proportion of unique L-mers, but had the highest proportion of doubly-unique L-mers, making them somewhat easier to identify in comparison to the simulated metagenome above, but difficult to quantify by tools other than `CAMMiQ`. We again generated two simulated read collections from this mixture, one with no sequencing errors (denoted Least-quantifiable-20-uniform-e.f.) and another with i.i.d. substitution rate of 1% (denoted Least-quantifiable-20-uniform). The ability of `CAMMiQ`'s ILP formulation to simultaneously determine the presence and abundance of genomes in these queries help it outperform the alternatives. (See Supplementary Methods and Figure 4 for a more detailed explanation.)

Even though RefSeq identifies each genome in our species-level dataset to represent a distinct species, a few of them have "unclassified" lineages at the species level. (Some of the genomes in the above queries are among them; see Figure 4 in the Supplementary Methods.) For example, *Rhizobium sp. N1314* with Taxonomy ID: 1703961, NCBI BLAST name: *a-proteobacteria*, has Rank: species; however its Lineage is noted as *unclassified Rhizobium*. Because of this ambiguity, we generated a third set of metagenomes, again consisting 20 of those 50 genomes with the lowest proportion of unique L-mers, this time making sure that each of these 20 genomes represent a distinct genus. We generated 3 queries from this set of genomes. The first one had uniform coverage and had no sequencing errors (denoted Least-20-genera-uniform-e.f.). The second and third both had i.i.d. substitutions at a rate of 1%; the second had a uniform read coverage (denoted Least-20-genera-uniform), while the third had lognormal distribution (denoted Least-20-genera-lognormal).

In addition to the above particularly challenging queries, we simulated a number of additional read collections from 20 to 100 randomly chosen genomes from our species-level dataset. Unlike the above queries, all these read collections had i.i.d. substitution errors; the first three queries at a rate of 1% and the last query at a rate of 0.6%. Among them, the first simulated query (denoted Random-20-uniform) included reads from 20 genomes, each with similar read coverage. The second (denoted Random-20-lognormal) again included reads from 20 genomes, this time with coverages obeying a log-normal distribution. The third (denoted Random-20-lognormal-a.g.) included reads from 20 genomes, again with log-normal coverage distribution; what makes this query unique is that 10% of the reads were from an additional genome (denoted in the dataset name as a.g.) not included in our species-level dataset and thus is not part of `CAMMiQ`'s index. The fourth (denoted Random-100-uniform) included reads from 100 randomly chosen genomes from our species-level dataset, all with similar coverage.

In order to assess `CAMMiQ`'s performance in strain level identification and quantification, we simulated queries involving genomes from a database of 614 strains of human gastrointestinal bacteria [40][1] from 409 species. We again simulated multiple queries, the first one involving reads from 25 strains with the smallest number of unique L-mers (denoted HumanGut-least-25), next two involving reads from randomly selected 100 strains, the first with $L = 100$ as per the remainder of the queries (denoted HumanGut-random-100-1), and the second with $L = 125$ (denoted HumanGut-random-100-2), and the final involving reads sampled from 409 randomly picked strain level genomes (denoted HumanGut-all), each representing a distinct species in the dataset. Note that none of these queries included more than one strain per species since two distinct strains from a species are not likely to be simultaneously present in a metagenomic sample.

The index we built to respond to these queries consisted of all the 614 strain level genomes described above. The majority of these genomes are not complete and is comprised of multiple contigs; we filtered out any contig with length < 10KB and built the index on the remaining contigs. This resulted in seven strains without a single unique 100-mer and one strain without a single unique or doubly unique 100-mer. This last genome of *Bacillus andreraoultii* was excluded from our queries since it contains no indexable substring.

[1]The complete set of genomes in this database is 617 but only 614 can be downloaded from RefSeq.

Performance Measure	Query Set	CAMMiQ	Kraken2	KrakenUniq	CLARK	MetaPhlAn2
A. Precision	Least-20-uniform-e.f.	**1.0**	0.014	0.952	**1.0**	-
	Least-20-uniform	**0.974**	0.001	0.009	0.006	-
	Least-quantifiable-20-uniform-e.f.	**1.0**	0.043	0.936	**1.0**	-
	Least-quantifiable-20-uniform	**0.989**	0.012	0.086	0.064	-
	Least-20-genera-uniform-e.f.	**1.0**	0.027	0.978	**1.0**	-
	Least-20-genera-uniform	**0.993**	0.005	0.047	0.028	-
	Least-20-genera-lognormal	**0.993**	0.005	0.049	0.033	-
	Random-20-uniform	**0.997**	0.783	0.981	0.994	-
	Random-20-lognormal	**0.998**	0.933	0.991	0.995	-
	Random-20-lognormal-a.g.*	**0.998**	0.900	**0.998**	0.997	-
	Random-100-uniform	**0.998**	0.968	0.965	0.997	-
B. Num. Assigned Reads	Least-20-uniform-e.f.	**1.72M**	25226	664	675	76483
	Least-20-uniform	**1.57M**	220721	62189	94632	70605
	Least-quantifiable-20-uniform-e.f.	**3.18M**	79408	6307	5844	120199
	Least-quantifiable-20-uniform	**2.81M**	246877	62617	81575	115959
	Least-20-genera-uniform-e.f.	**2.89M**	33523	1844	1763	67625
	Least-20-genera-uniform	**2.58M**	170506	36286	57764	64204
	Least-20-genera-lognormal	**2.53M**	183709	41681	59679	73433
	Random-20-uniform	3.84M	3.33M	**3.88M**	3.81M	80633
	Random-20-lognormal	**4.48M**	3.84M	4.43M	4.39M	19484
	Random-20-lognormal-a.g.	0.91M	0.93M	**0.97M**	**0.97M**	28328
	Random-100-uniform	**19.5M**	17.0M	19.4M	18.7M	0.4M
C. Num. Correctly Identified Genomes	Least-20-uniform-e.f.	20/20	16	17	18	13
	Least-20-uniform	20/28	16	17	18	13
	Least-quantifiable-20-uniform-e.f.	20/20	19	19	20	18
	Least-quantifiable-20-uniform	20/27	19	19	20	18
	Least-20-genera-uniform-e.f.	20/20	18	18	18	12
	Least-20-genera-uniform	20/24	18	18	18	12
	Least-20-genera-lognormal	20/33	17	17	18	12
	Random-20-uniform	20/20	18	20	20	11
	Random-20-lognormal	20/21	20	20	20	9
	Random-20-lognormal-a.g.*	20/20	19	20	20	13
	Random-100-uniform	100/100	100	100	100	76
D. L1 Err.	Least-20-uniform-e.f.	**0.0790**	-	-	0.8846	0.8171
	Least-20-uniform	**0.0929**	-	-	0.9124	0.8180
	Least-quantifiable-20-uniform-e.f.	**0.0375**	-	-	0.5774	0.5722
	Least-quantifiable-20-uniform	**0.0278**	-	-	0.5659	0.5956
	Least-20-genera-uniform-e.f.	**0.0626**	-	-	0.7156	0.9153
	Least-20-genera-uniform	**0.0591**	-	-	0.7067	0.9533
	Least-20-genera-lognormal	**0.0439**	-	-	0.5701	1.0083
	Random-20-uniform	**0.0113**	0.4139	0.1446	0.2042	0.8257
	Random-20-lognormal	**0.0038**	0.2843	0.1217	0.1496	1.7367
	Random-20-lognormal-a.g.*	0.1262	0.2412	**0.1173**	0.1861	0.5578
	Random-100-uniform	**0.0096**	0.2364	0.1327	0.2063	0.7801

Table 2: Performance evaluation of CAMMiQ, Kraken2, KrakenUniq, CLARK and MetaPhlAn2 on queries from the species level dataset. Precision: the proportion of reads correctly assigned to a genome among the set of reads assigned to some genome (correctly or incorrectly). Number of assigned reads: the total number of reads assigned to some genome. Number of correctly identified genomes: for CAMMiQ we report both the number of correctly identified genomes (true positives) and the total number of genomes returned by its ILP formulation (false positive); for Kraken2, KrakenUniq, CLARK and MetaPhlAn2 we only report the number of correctly identified genomes (true positives). Note that we consider MetaPhlAn2 to have a correct identification even if it reports the genus that this genome belongs to. L1 error: the L1 distance between the true relative abundance values (between 0 and 1) and the predicted abundance values for each genome in the query (i.e. positives). We made an exception for MetaPhlAn2, where we measured the genus level L1 distance. Note that we converted the true abundance values reported by Kraken2, KrakenUniq and CLARK by dividing the predicted abundance value for each genome by its length and then normalizing these values by the total abundance value of all genomes. In each of the four measures, the best performing tool's results are highlighted.
*: 10% reads in the query Random-20-lognormal-a.g. are from a genome not in the index; any assignment of these reads are necessarily incorrect by all tools except MetaPhlAn2 - which uses its own index, that happens to include this genome.

3.3 Classification and Quantification Performance - Species Level

We tested CAMMiQ on read collections sampled from the species-level dataset and compared its practical performance with the select metagenomic analysis tools. Perhaps the most widely-used performance measures to benchmark metagenomic classifiers are the proportion of reads correctly assigned to a genome among (i) the set of reads assigned to some genome, i.e. *precision*, and (ii) the full set of reads in the query, i.e. *sensitivity* [16]. In Table 2A we report the precision values for all tools we benchmarked with the exception of MetaPhlAn2, since it does not report individual read assignments. In Table 2B, instead of reporting sensitivity, we report the total number of reads assigned to a genome, since the main goal of Kraken2, KrakenUniq and CLARK are to classify as many reads as possible, correctly. The number of reads assigned to a genome (part B of the Table) multiplied by precision (part A of the table) indeed gives this number, i.e. that of correctly classified reads. Note that any read assigned to a taxonomic level higher than (and not including) species level by Kraken2 or KrakenUniq are considered to be not assigned. (In a way, this increases their reported precision but decreases their number of assigned reads.) Also note that CAMMiQ's primary goal is not to classify each read but rather identify and quantify genomes in a query. Nonetheless, we still report its intermediate output as follows. CAMMiQ considers certain reads as conflicting; here we consider them as not assigned. It assigns certain reads to a single genome; we consider each such read assigned, and if the assignment is correct, also correctly assigned. CAMMiQ then assigns each remaining read ambiguously to two potential genomes.[2] We consider this read assigned, and in case one of these two genomes are correct, also correctly assigned. Finally since MetaPhlAn2 has an index based on a predetermined database, it is not easy to evaluate this tool's precision, since the IDs of the genomes indexed by the other four tools do not always correspond to MetaPhlAn2's genome IDs. As a result we only report its number of assigned reads but not its precision.

We benchmarked CAMMiQ using its default parameter setting of $L_{\min} = 26$ and $L_{\max} = 50$, against Kraken2, KrakenUniq and CLARK, with all three using k-mer length of 26. All four of these tools used the same genomes for establishing their index. As can be seen in Table 2, CAMMiQ demonstrated the best precision for read classification in all of the 11 simulated query sets. With respect to the total number of assigned reads (correctly or incorrectly) on the first 7 queries, i.e. those involving the 20 genomes with the least number of unique L-mers (Least-20-uniform-e.f. and Least-20-uniform), those that are the least quantifiable (Least-quantifiable-20-uniform-e.f. and and Least-quantifiable-20-uniform), and those are composed of genomes each from a distinct genus (Least-20-genera-uniform-e.f., Least-20-genera-uniform and Least-20-genera-lognormal), CAMMiQ improves over the alternatives not only in terms of precision but also the number of reads assigned, sometimes by an order of magnitude or more. The only exceptions are on those "hypothetical" error-free queries, on which not only CAMMiQ and but also CLARK achieves 100% precision. For the above 7 queries Kraken2 has the lowest precision; KrakenUniq has improved precision but only to a degree. In return, the number of assigned reads by KrakenUniq are typically the lowest.

On the next four queries, which are easier to identify and quantify, CAMMiQ's peformance is still the best overall. Its precision is the best for all four of these queries while the number of assigned reads are slightly worse than KrakenUniq in two queries. This is likely due to the fact that KrakenUniq produces some incorrect assignments, especially for the Random-20-uniform query on which KrakenUniq has a lower precision. The precision of CAMMiQ is identical to KrakenUniq on Random-20-lognormal-a.g. query, where 10% of the reads are sampled from an additional genome not indexed. This is despite that the number of assigned reads from this query are higher for KrakenUniq and CLARK, demonstrating that the introduction of un-indexed species impacts the performance of CAMMiQ similarly to the other tools.

We next evaluated the number of correctly identified genomes (for MetaPhlAn2, correctly identified species) in each query, as well as the L1 distance between the true abundance profile and the predicted abundance profile by all five tools on the 11 queries involving our species level dataset. The results can be found in Table 2, parts C and D and Figure 3. Note that CAMMiQ and MetaPhlAn2 automatically incorporate a normalization with respect to genome lengths, while Kraken2, KrakenUniq and CLARK simply report the number of reads assigned to each taxonomy rank (for our queries, species) as their abundance profile. In order to compute L1 distances correctly, we converted the true abundance profiles of Kraken2, KrakenUniq and CLARK by dividing the predicted abundance value of each genome by its length and then normalizing

[2]This happens if the read includes one or more doubly unique substrings from the same pair of genomes but no unique substring.

Query Set	$L_{\min}$	Precision	Num. Assigned Reads	Num. Identified Species		L1 Err.	
				$\alpha = 0.001$	$\alpha = 0.0001$	$\alpha = 0.001$	$\alpha = 0.0001$
Least-20-uniform	21	0.887	1.69M	20/21	31/32	0.0835	0.0836
	26	0.974	1.57M	20/21	28/29	0.0929	0.0929
	31	0.980	1.49M	20/20	22/23	0.1064	0.1060
Least-quantifiable-20-uniform	21	0.967	2.85M	20/20	29/31	0.0423	0.0400
	26	0.989	2.81M	20/20	27/29	0.0294	0.0278
	31	0.992	2.70M	20/20	24/26	0.0344	0.0326
Least-20-genera-uniform	21	0.974	2.61M	20/20	25/26	0.0706	0.0715
	26	0.993	2.58M	20/20	24/26	0.0585	0.0591
	31	0.995	2.48M	20/20	23/24	0.0688	0.0683
Least-20 genera-lognormal	21	0.974	2.56M	19/19	34/35	0.0394	0.0418
	26	0.993	2.53M	19/19	33/34	0.0416	0.0439
	31	0.995	2.43M	19/19	32/33	0.0355	0.0374
Random-20-uniform	21	0.993	3.83M	20/20	20/20	0.0220	0.0220
	26	0.997	3.84M	20/20	20/20	0.0113	0.0113
	31	0.998	3.69M	20/20	20/20	0.0294	0.0294
Random-20-lognormal	21	0.996	4.50M	20/20	21/21	0.0432	0.0431
	26	0.998	4.48M	20/20	21/21	0.0039	0.0038
	31	0.998	4.32M	20/20	21/21	0.0062	0.0058
Random-20-lognormal-a.g.*	21	0.989	0.92M	17/17	20/20	0.1492	0.1268
	26	0.998	0.91M	16/16	20/20	0.1631	0.1262
	31	0.999	0.87M	16/16	20/20	0.1658	0.1298
Random-100-uniform	21	0.996	19.5M	100/100	100/100	0.0176	0.0176
	26	0.998	19.5M	100/100	100/100	0.0096	0.0096
	31	0.999	19.0M	100/100	100/100	0.0104	0.0104

Table 3: Performance of `CAMMiQ` as a function of minimum unique/doubly-unique substring length $L_{\min} = 21, 26, 31$, and minimum relative read count threshold $\alpha = 0.001, 0.0001$ to report a genome. Precision: the proportion of reads correctly assigned to a genome among the set of reads assigned to some genome, correctly or incorrectly. Number of assigned reads: the total number of reads assigned to some genome. Number of identified genomes: the number of genomes returned by `CAMMiQ`'s ILP formulation v.s. the number of genomes that have sufficient read assignments (determined by α). L1 error: the L1 distance between the true relative abundance values (between 0 and 1) and the predicted abundance values for each genome in the corresponding query.
*: 10% reads in the query Random-20-lognormal-a.g. are from a genome not in the index; any assignment of such a read to a genome is necessarily incorrect.

these values by the total abundance value of all genomes.

As can be seen in Table 2 panels C and D, as well as Figure 3, CAMMiQ clearly offers the best performance in both identification and quantification. It correctly identified all genomes present in each one of the 11 queries and was not impacted by the additional genome we introduced in the Random-20-lognormal-a.g. query. As importantly, it only returned very few false positive genomes for the most challenging Least-20-uniform, Least-quantifiable-20-uniform and Least-20-genera-uniform as well as Least-20-genera-lognormal queries, and at most one false positive genome for the remaining 7 queries. Other tools had varying levels of false negatives in these 11 queries. Among them CLARK offered the best false negative performance: it only reported false negatives in the queries involving a genome without a single unique k-mer. As can be expected, MetaPhlAn2 had the worst performance with respect to false negatives, very likely due to the incompleteness of its marker gene list[3]. This also led to a larger L1 distance than the other tools, even for the relatively easy queries. Kraken2 and KrakenUniq were also prone to have a few false negatives, though less than MetaPhlAn2. Furthermore their predicted abundances are smaller than the true abundance values, because they may assign reads to higher taxonomic ranks than the species level; see Figure 3. This is especially evident in the in the first 7 (difficult) queries: even though Kraken2 and KrakenUniq identified the majority of the genomes correctly, the abundance values they reported on these genomes were all close to 0 and thus the L1 distances turned out to be very close to 1 (see Figure 3). As a result, these values are not reported in Table 2D. Note that we do not report the number of false positive genomes returned by each tool on these 11 queries. This is primarily because of the fact that CLARK, Kraken2 and KrakenUniq are not designed to identify genomes in a query mixture and thus do not "care" whether the false positive assignments are distributed across many genomes (which would result in a large number of false positives) or are concentrated in a few genomes (resulting in a few false positives). MetaPhlAn2 aims to identify genomes however it can do so in any taxonomic level. As such, all these four tools return a very large number of false positives, especially for the first 7 queries, but since we felt that this could be unfair due to the above reasons, we decided to report only the false positives reported by CAMMiQ, noting that its genome identification performance is significantly better than the other tools. We ignore the false positive calls also in the L1 measure since it is calculated only on the true positive genomes. This explains the single query and measure for which CAMMiQ seems to have performed worse than an alternative, namely KrakenUniq: on Random-20-lognormal-a.g. KrakenUniq's L1 distance is slightly better than CAMMiQ. We remind that in this query, 10% of the reads are sampled from a genome not in the index. Since these reads can not be assigned through any means, the relative abundances of the other genomes will be overestimated by $\sim 10\%$, provided that there are no false positive genomes as per CAMMiQ. However, KrakenUniq's false positive calls (there are several) reduces its relative abundance estimates for the true positive genomes, and gives a seemingly better L1 distance. This can be observed in Figure 3, which depicts for each query, relative abundance estimate for each genome by each one of the tools we benchmarked. As can be clearly seen, CAMMiQ's individual abundance estimates are just on the mark for each one of the genomes even for the most difficult queries.

We finally evaluated the impact of two important parameters for CAMMiQ: α, the minimum relative read count threshold for reporting a genome, and $L_{\min}$, the minimum unique or double-unique substring length (values larger than the default value of 50 for $L_{\max}$ did not have a big impact and thus are not reported here). In Table 3 we report the results for each possible combination of $L_{\min} = 21, 26, 31$ and $\alpha = 0.001, 0.0001$, on 8 of the 11 queries, omitting the 3 error free queries (on which the impact on precision is minimal). As we increase $L_{\min}$, CAMMiQ's precision improves, however its read assignment performance deteriorates. Interestingly, its predicted abundance values did not change much with increasing $L_{\min}$. As a result we set the default $L_{\min}$ to 26 in CAMMiQ. On the other hand, increasing the value of α, decreased the number of false positives in CAMMiQ's output, particularly in the most difficult queries. However, as a result of this, for the queries Least-20-genera-lognormal and Random-20-lognormal-a.g., those genomes with low abundance values were disregarded by CAMMiQ, leading to false negatives. CAMMiQ allows the user to set the parameter α with prior knowledge on the reads to be queried (e.g., the expected read coverage or the number of genomes in the query); we set its default value to 0.0001.

[3]Here we used the latest set of marker genes mpa_v20_m200 in MetaPhlAn2.

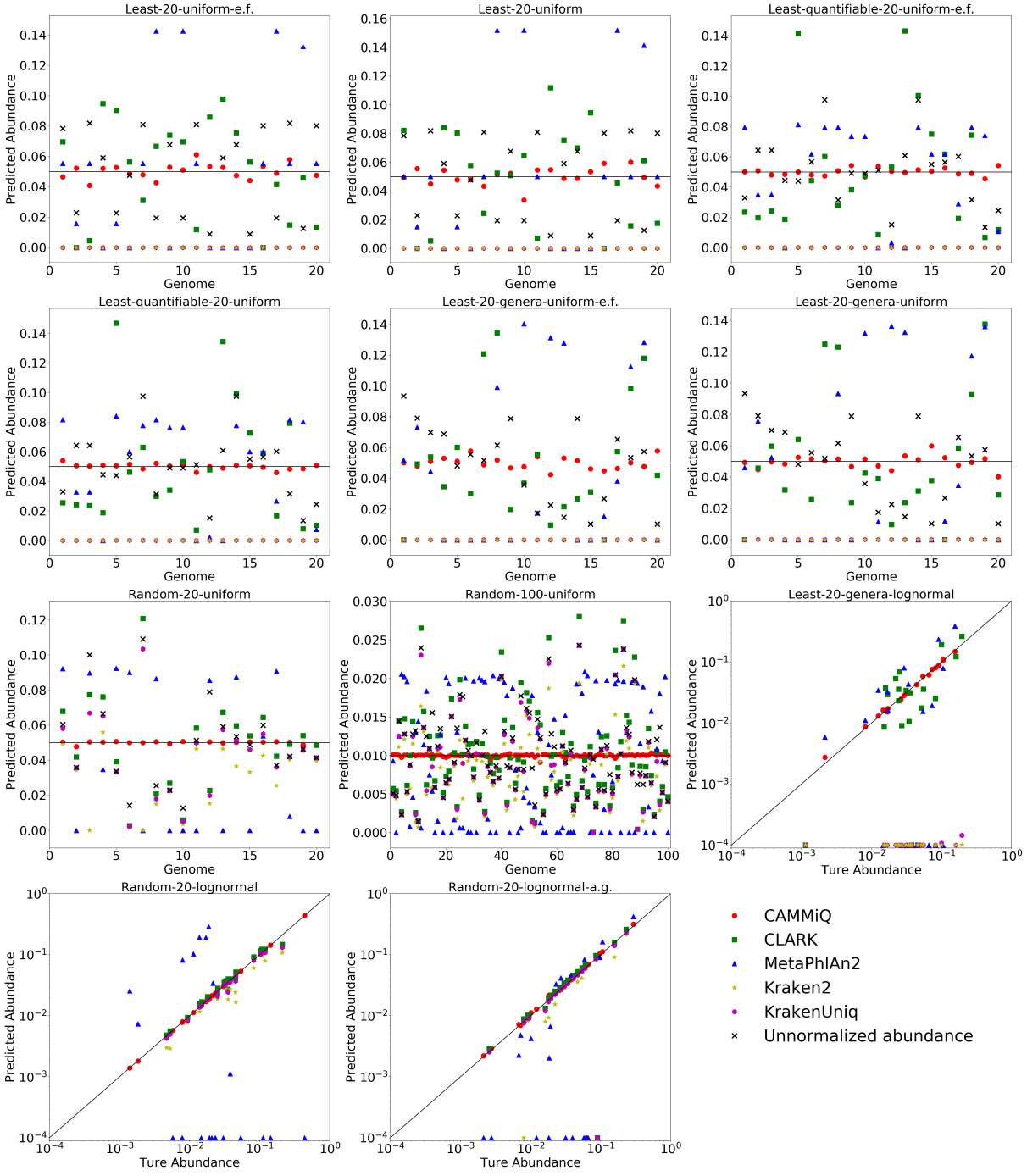

Figure 3: Comparison of CAMMiQ's relative abundance estimates to that of the existing methods for all 11 queries involving the species level dataset. The horizontal axis in the top 8 plots represent the species (i.e., genomes) in an arbitrary order and the vertical axis represent the relative abundance values. The bottom 3 plots are for the queries with log-normal distributions, where the horizontal axis represent the true abundance values while the vertical axis represent the estimated abundance values, both in log-scale.

Query Set	CAMMiQ		MetaPhlAn2	
	Num. Identified Strains	L1 Err.	Num. Identified Strains	L1 Err.
HumanGut-least-25	24/26	0.1130	$\geq$ 18/19	0.2910
HumanGut-random-100-1	100/101	0.0209	$\geq$ 74/83	0.3260
HumanGut-random-100-2	100/102	0.0256	$\geq$ 67/72	0.5066
HumanGut-all	404/407	0.0517	$\geq$ 279/305	0.4439

Table 4: CAMMiQ's quantification performance at strain level, compared to MetaPhlAn2. Number of identified strains: the number of true positive strains/the total number of strains identified. L1 error: the L1 distance between the true relative abundance values and the predicted abundance values, across all strains in the query.

3.4 Quantification Performance - Strain Level

We finally evaluated CAMMiQ's performance on queries composed from our strain-level dataset that consists of 614 Human Gut related genomes (strains) from 409 species [40] as described in Section 3.2. Three of the tools we evaluated, namely CLARK, Kraken2 and KrakenUniq, simply aim to perform read classification, which we evaluated in Section 3.3. They are not designed to identify or quantify genomes, especially at the strain level. On the other hand MetaPhlAn2's index have strain level information and through that it attempts to identify and quantify strains. As a result we report on the performances of both CAMMiQ and MetaPhlAn2 in Table 4. As can be seen, CAMMiQ managed to identify and accurately quantify all strains in the queries HumanGut-random-100-1 and HumanGut-random-100-2, and $>$ 98% strains in the query HumanGut-all, with almost no false positives. Here we only report the strain level calls made by MetaPhlAn2 (it made additional calls at the species level or higher), based on our best attempt to match its calls to the strain IDs available to us. It may be possible to slightly increase the true positive values (the first value) reported here.

3.5 Computational Resource

We compared the running time and memory use of CAMMiQ, Kraken2, KrakenUniq, CLARK and MetaPhlAn2 in responding to the queries. We do not report the time for building the index or loading it into memory, since this is performed only once - all tools roughly need a couple hours to construct the index on the species level dataset. When it comes to querying time, CAMMiQ is outperformed only by Kraken2, typically by a factor of 3. This is primarily due to the fact that Kraken2's index is much smaller than that of CAMMiQ since it consists of a small subset of unique k-mers. This compact index structure substantially impacts its performance, which is improved by KrakenUniq, through its consideration of all unique k-mers. The resulting index size of KrakenUniq is comparable to that of CAMMiQ (CAMMiQ's is slightly higher as it also includes doubly-unique substrings), however its running time is almost twice as much, even though it does not solve an ILP. CAMMiQ owes its superior identification and quantification performance to its ILP formulation, however it is not time-wise penalized by it. The best memory footprint is achieved by MetaPhlAn2 through the use of its own index, however both its run time and its identification/quantification performance is below the others.

4 Discussion

We have introduced CAMMiQ, a new computational approach to solve a computational problem that has not been exactly addressed by any available method: given a set $\mathcal{S}$ of distinct genomic sequences (of any taxonomic rank), build a data structure so as to identify and quantify genomes in a any query, composed of a mixture of reads from a subset of genomes from $\mathcal{S}$. CAMMiQ is particularly designed to handle genomes that lack unique features; for that, it reduces the identification and quantification problems to a combinatorial optimization problem that assigns substrings with limited ambiguity (i.e. doubly unique substrings) to genomes so that each genome is "uniformly covered". Because each such substring has limited ambiguity, the resulting combinatorial optimization problem can be very efficiently solved through existing integer program solvers such as IBM CPLEX (and thus are branch-and-bound based, whose run-time performance

Performance Measure	Query Set	CAMMiQ	Kraken2	KrakenUniq	CLARK	Metaphlan2
A. Read Assignment Time	Least-20-uniform	151.7s	**52.2s**	286.5s	402.5s	608.8s
	Least-quantifiable-20-uniform	180.5s	**50.8s**	284.1s	416.0s	569.6s
	Least-20-genera-uniform	143.5s	**53.0s**	273.4s	285.2s	438.9s
	Least-20-genera-lognormal	153.6s	**61.8s**	268.8s	289.4s	442.7s
	Random-20-uniform	198.4s	**48.1s**	284.0s	315.1s	516.7s
	Random-20-lognormal	167.8s	**48.8s**	294.5s	367.6s	569.3s
	Random-20-lognormal-a.g.	49.8s	**14.7s**	78.9s	75.2s	188.9s
	Random-100-uniform	906.0s	**215.4s**	1261.5s	1536.2s	2401.4s
B. Memory	Least-20-uniform	94.6G	11.4G	67.9G	81.3G	**1.3G**
	Least-quantifiable-20-uniform	93.3G	11.4G	77.1G	81.3G	**1.3G**
	Least-20-genera-uniform	92.6G	11.4G	81.4G	81.0G	**1.3G**
	Least-20-genera-lognormal	93.3G	11.4G	77.8G	81.0G	**1.3G**
	Random-20-uniform	95.4G	11.4G	92.7G	81.3G	**1.3G**
	Random-20-lognormal	95.7G	11.4G	89.3G	81.3G	**1.3G**
	Random-20-lognormal-a.g.	94.3G	11.3G	51.9G	80.3G	**1.3G**
	Random-100-uniform	111.9G	11.9G	143.3G	85.7G	**1.4G**
C. Index Size	-	8.94G	11.20G	155.22G	71.85G	**1.11G**

Table 5: Comparison of the computational resources required by CAMMiQ, Kraken2, KrakenUniq, CLARK and MetaPhlAn2, when run on a single thread.

is determined by the fan-out at each decision point). Provided that the doubly-unique substrings of a given genome are not all shared with one other genome, the use of doubly-unique substrings increases CAMMiQ's ability to identify and quantify this genome within a query. As mentioned earlier, in case the dataset to be indexed involves several genomes with high level of similarity, CAMMiQ's data structure and its combinatorial optimization formulation can easily be generalized to include triply, quadruply ... unique substrings, without much computational overhead.

References

[1] Huson, D. H., Auch, A. F., Qi, J. & Schuster, S. C. Megan analysis of metagenomic data. *Genome research* **17**, 377–386 (2007).

[2] Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J. Basic local alignment search tool. *Journal of molecular biology* **215**, 403–410 (1990).

[3] Elworth, R. *et al.* To petabytes and beyond: recent advances in probabilistic and signal processing algorithms and their application to metagenomics. *Nucleic Acids Research* (2020).

[4] Liu, B., Gibbons, T., Ghodsi, M., Treangen, T. & Pop, M. Accurate and fast estimation of taxonomic profiles from metagenomic shotgun sequences. *Genome biology* **12 (Suppl 2)**, S4 (2011).

[5] Segata, N. *et al.* Metagenomic microbial community profiling using unique clade-specific marker genes. *Nature methods* **9**, 811 (2012).

[6] Lopresti, D. & Tomkins, A. Block edit models for approximate string matching. *Theoretical computer science* **181**, 159–179 (1997).

[7] Cormode, G., Paterson, M., Sahinalp, S. C. & Vishkin, U. Communication complexity of document exchange. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, 197–206 (Society for Industrial and Applied Mathematics, 2000).

[8] Li, M., Chen, X., Li, X., Ma, B. & Vitányi, P. M. The similarity metric. *IEEE transactions on Information Theory* **50**, 3250–3264 (2004).

[9] Vinga, S. & Almeida, J. Alignment-free sequence comparison—a review. *Bioinformatics* **19**, 513–523 (2003).

[10] Leimeister, C.-A. & Morgenstern, B. Kmacs: the k-mismatch average common substring approach to alignment-free sequence comparison. *Bioinformatics* **30**, 2000–2008 (2014).

[11] Ames, S. K. *et al.* Scalable metagenomic taxonomy classification using a reference genome database. *Bioinformatics* **29**, 2253–2260 (2013).

[12] Wood, D. E. & Salzberg, S. L. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome biology* **15**, R46 (2014).

[13] Břinda, K., Sykulski, M. & Kucherov, G. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics* **31**, 3584–3592 (2015).

[14] Kawulok, J. & Deorowicz, S. Cometa: classification of metagenomes using k-mers. *PloS one* **10**, e0121453 (2015).

[15] Tu, Q., He, Z. & Zhou, J. Strain/species identification in metagenomes using genome-specific markers. *Nucleic acids research* **42**, e67–e67 (2014).

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

[17] Luo, Y., Zeng, J., Berger, B. & Peng, J. Low-density locality-sensitive hashing boosts metagenomic binning. In *International Conference on Research in Computational Molecular Biology*, 255 (Springer, 2016).

[18] Ondov, B. D. *et al.* Mash: fast genome and metagenome distance estimation using minhash. *Genome biology* **17**, 132 (2016).

[19] McHardy, A. C., Martín, H. G., Tsirigos, A., Hugenholtz, P. & Rigoutsos, I. Accurate phylogenetic classification of variable-length dna fragments. *Nature methods* **4**, 63 (2007).

[20] Rosen, G., Garbarine, E., Caseiro, D., Polikar, R. & Sokhansanj, B. Metagenome fragment classification using n-mer frequency profiles. *Advances in bioinformatics* **2008** (2008).

[21] Brady, A. & Salzberg, S. L. Phymm and phymmbl: metagenomic phylogenetic classification with interpolated markov models. *Nature methods* **6**, 673 (2009).

[22] Rosen, G. L., Reichenberger, E. R. & Rosenfeld, A. M. Nbc: the naive bayes classification tool webserver for taxonomic classification of metagenomic reads. *Bioinformatics* **27**, 127–129 (2010).

[23] Vervier, K., Mahé, P., Tournoud, M., Veyrieras, J.-B. & Vert, J.-P. Large-scale machine learning for metagenomics sequence classification. *Bioinformatics* **32**, 1023–1032 (2015).

[24] Lin, Y.-Y. *et al.* Cliiq: Accurate comparative detection and quantification of expressed isoforms in a population. In *International Workshop on Algorithms in Bioinformatics*, 178–189 (Springer, 2012).

[25] Li, W., Feng, J. & Jiang, T. Isolasso: a lasso regression approach to rna-seq based transcriptome assembly. *Journal of Computational Biology* **18**, 1693–1707 (2011).

[26] Dao, P. *et al.* Orman: optimal resolution of ambiguous rna-seq multimappings in the presence of novel isoforms. *Bioinformatics* **30**, 644–651 (2014).

[27] Sobih, A., Tomescu, A. I. & Mäkinen, V. Metaflow: Metagenomic profiling based on whole-genome coverage analysis with min-cost flows. In *International Conference on Research in Computational Molecular Biology*, 111–121 (Springer, 2016).

[28] Breitwieser, F., Baker, D. & Salzberg, S. L. Krakenuniq: confident and fast metagenomics classification using unique k-mer counts. *Genome biology* **19**, 1–10 (2018).

[29] Solomon, B. & Kingsford, C. Fast search of thousands of short-read sequencing experiments. *Nature biotechnology* **34**, 300 (2016).

[30] Solomon, B. & Kingsford, C. Improved search of large transcriptomic sequencing databases using split sequence bloom trees. In *International Conference on Research in Computational Molecular Biology*, 257–271 (Springer, 2017).

[31] Sun, C., Harris, R. S., Chikhi, R. & Medvedev, P. Allsome sequence bloom trees. In *International Conference on Research in Computational Molecular Biology*, 272–286 (Springer, 2017).

[32] Ferdman, M., Johnson, R. & Patro, R. Mantis: A fast, small, and exact large-scale sequence-search index. In *Research in Computational Molecular Biology*, 271 (Springer, 2018).

[33] Matias, Y., Muthukrishnan, S., Sahinalp, S. C. & Ziv, J. Augmenting suffix trees, with applications. In *European Symposium on Algorithms*, 67–78 (Springer, 1998).

[34] Kasai, T., Lee, G., Arimura, H., Arikawa, S. & Park, K. Linear-time longest-common-prefix computation in suffix arrays and its applications. In *Annual Symposium on Combinatorial Pattern Matching*, 181–192 (Springer, 2001).

[35] Ilie, L. & Smyth, W. F. Minimum unique substrings and maximum repeats. *Fundamenta Informaticae* **110**, 183–195 (2011).

[36] Vazirani, V. V. *Approximation algorithms* (Springer Science & Business Media, 2013).

[37] Pruitt, K. D., Tatusova, T. & Maglott, D. R. Ncbi reference sequences (refseq): a curated non-redundant sequence database of genomes, transcripts and proteins. *Nucleic acids research* **35**, D61–D65 (2007).

[38] Wood, D. E., Lu, J. & Langmead, B. Improved metagenomic analysis with kraken 2. *Genome biology* **20**, 257 (2019).

[39] Truong, D. T. *et al.* Metaphlan2 for enhanced metagenomic taxonomic profiling. *Nature methods* **12**, 902 (2015).

[40] Forster, S. C. *et al.* A human gut bacterial genome and culture collection for improved metagenomic analyses. *Nature biotechnology* **37**, 186 (2019).

[41] Weissman, T., Ordentlich, E., Seroussi, G., Verdu, S. & Weinberger, M. J. Inequalities for the l1 deviation of the empirical distribution. *Hewlett-Packard Labs, Tech. Rep* (2003).

5 Supplementary Methods

5.1 Unique substrings from $\texttt{LCP}_u$ or $\texttt{LCP}_d$

The pseudocode for the algorithm to compute arrays SU and SD according to equations (3) and (4) respectively, given SA, $\texttt{LCP}_u$ and $\texttt{LCP}_d$ is given below. As can be seen, the algorithm runs in $O(M)$ time.

Algorithm 1 ShortestUniqueFromLCP(SA, $\texttt{LCP}_u$, $\texttt{LCP}_d$, $L_{\max}$)

for i **from** 1 **to** M **do** //Initialize SU
  $\texttt{SU}[i] \leftarrow 0$
  $\texttt{SD}[i] \leftarrow 0$
end for
for i **from** 1 **to** M **do** //Update SU according to (3) and SD according to (4)
  if $\texttt{LCP}_u[i] < L_{\max}$ **then**
    $\texttt{SU}[\texttt{SA}[i] + \texttt{LCP}_u[i]] \leftarrow \max\{\texttt{SU}[\texttt{SA}[i] + \texttt{LCP}_u[i]], \texttt{SA}[i]\}$
  end if
  if $\texttt{LCP}_d[i] < L_{\max}$ **then**
    $\texttt{SD}[\texttt{SA}[i] + \texttt{LCP}_d[i]] \leftarrow \max\{\texttt{SD}[\texttt{SA}[i] + \texttt{LCP}_d[i]], \texttt{SA}[i]\}$
  end if
end for
return SU, SD

5.2 Computing $\texttt{LCP}_u$ and $\texttt{LCP}_d$

In this section we show that SU and SD can be correctly constructed in $O(M)$ time. We start by showing that the definition of $\texttt{LCP}_u$ and $\texttt{LCP}_d$ in Equation (1) and Equation (2) can respectively lead to the shortest substrings occurring in at most one genome or two genomes. Then we give CAMMiQ's detailed implementation of Equation (5) and Equation (6) to compute the $\texttt{LCP}_u$ and $\texttt{LCP}_d$ arrays. Finally we give a running time analysis of this implementation.

First consider the content of SU at the end of procedure ShortestUniqueFromLCP. To see the substring $s[l..r]$ corresponds to the r-th entry $\texttt{SU}[r] = l$ (where $l \neq 0$) in SU is unique, meaning it only occurs in genome with ID $\texttt{GSA}[\texttt{SA}^{-1}[l]]$, assume that there is another genome $s_{i'}$ having the same substring $s[l', r'] = s[l..r]$ - this leads to a contradiction, since it implies that $\text{lcp}(\texttt{suf}[l], \texttt{suf}[l']) \geq r - l + 1$. However, due to the update rule of SU and the definition of $\texttt{LCP}_u$, $\text{lcp}(\texttt{suf}[l], \texttt{suf}[l']) \leq r - l$ for any $1 \leq l' \leq M$ satisfying $\texttt{suf}[l]$ and $\texttt{suf}[l']$ start on different genomes, namely $\texttt{GSA}[\texttt{SA}^{-1}[l]] \neq \texttt{GSA}[\texttt{SA}^{-1}[l']]$, which is a contradiction. Now, to see $s[l..r]$ is a shortest unique substring, i.e. no substring of $s[l..r]$ is unique to genome $\texttt{GSA}[\texttt{SA}^{-1}[l]]$, we show that any $s[l..r' < r]$ and $s[l' > l, r]$ occurs in one other genome $s_{i'}$. The former case is due to the definition of $\texttt{LCP}_u$ - there exists $\texttt{suf}[l']$ on genome $i' \neq \texttt{GSA}[\texttt{SA}^{-1}[l]]$ such that $\text{lcp}(\texttt{suf}[l], \texttt{suf}[l']) \geq r - l$, implying a substring $s[l'..l' + (r - l) - 1]$ identical to $s[l, r - 1]$; the later case is due to the update rule of SU - if $s[l' > l, r]$ is also unique to genome $\texttt{GSA}[\texttt{SA}^{-1}[l]]$, then $\texttt{SU}[r]$ must be set to l' instead of l. Therefore, $s[l..r]$ is a shortest unique substring (to genome with ID $\texttt{GSA}[\texttt{SA}^{-1}[l]]$); on the other hand, if $s[l..r]$ is a shortest unique substring, then $\texttt{SU}[r]$ will maintain l after SU is completely updated.

Now consider the content of SD at the end of procedure ShortestUniqueFromLCP. We follow the above proof to show $s[l..r]$ is a shortest doubly-unique substring (to genome ID $\texttt{GSA}[\texttt{SA}^{-1}[l]]$ and possibly another genome i'). To see the substring $s[l..r]$ corresponds to the r-th entry $\texttt{SD}[r] = l$ (where $l \neq 0$) in SD occurs in at most two genomes, with ID $\texttt{GSA}[\texttt{SA}^{-1}[l]]$ (and possibly i', any genome that $\texttt{suf}[l']$ belongs to, giving the largest $\text{lcp}(\texttt{suf}[l], \texttt{suf}[l'])$), we can assume there exists a thrid genome $s_{i''}$ having the same substring $s[l'', r''] = s[l, r] = s[l', r']$ and similarly obtain a contradiction. Note that according to (2), it is possible to have $\texttt{LCP}_d[\texttt{SA}^{-1}[l]] \geq \texttt{LCP}_u[\texttt{SA}^{-1}[l]]$ and in this case $s[l..r]$ is a unique substring which only occurs in genome $\texttt{GSA}[\texttt{SA}^{-1}[l]]$. If $\texttt{LCP}_d[\texttt{SA}^{-1}[l]] < \texttt{LCP}_u[\texttt{SA}^{-1}[l]]$ on the other hand, then $s[l..r]$ must occur in exactly two genomes, since SD is updated according to $\texttt{LCP}_d$ and we can find another suffix of s whose length-$(r - l + 1)$ $(r - l + 1 \leq \texttt{LCP}_d[\texttt{SA}^{-1}[l]])$ prefix is identical to $s[l..r]$. In addition, to see $s[l..r]$ is a shortest doubly-unique substring, meaning no substring of $s[l..r]$ occurs only in genome $\texttt{GSA}[\texttt{SA}^{-1}[l]]$ and i', we can

similarly show that any $s[l..r' < r]$ and $s[l' > l, r]$ can be found in a third genome $s_{i''}$, regardless whether $\texttt{LCP}_d[\texttt{SA}^{-1}[l]] = \texttt{LCP}_u[\texttt{SA}^{-1}[l]]$ or $\texttt{LCP}_d[\texttt{SA}^{-1}[l]] < \texttt{LCP}_u[\texttt{SA}^{-1}[l]]$.

As a result of the above observations we can now formally state the following lemma.

Lemma 2. *(i)After updating* $\texttt{SU}$ *according to (3),* $\texttt{SU}[r] = l \neq 0$ *implies that* $s[l..r]$ *is a shortest unique substring to genome* $\texttt{GSA}[\texttt{SA}^{-1}[l]]$*;*
(ii)After updating $\texttt{SD}$ *according to (4)* $\texttt{SD}[r] = l \neq 0$ *implies that* $s[l..r]$ *is a shortest doubly-unique substring to genome* $\texttt{GSA}[\texttt{SA}^{-1}[l]]$ *and* $i' = \texttt{GSA}[\texttt{SA}^{-1}[l']]$ *where* $\texttt{suf}[l']$ *gives the largest* $lcp(\texttt{suf}[l], \texttt{suf}[l']))$*.*

Furthermore, it's also clear that all unique substrings $s[l..r]$ are stored in $\texttt{SU}$ and all doubly-unique substrings $s[l..r]$ are stored in $\texttt{SD}$, as we have considered the suffix of s starting with every possible l.

Both $\texttt{LCP}_u$ and $\texttt{LCP}_d$ can be modified to incorporate the minimum length constraint $L_{\min}$ on unique/doubly-unique substrings. By setting $\texttt{LCP}_u[i]$ to $\max\{L_{\min} - 1, \texttt{LCP}_u[i]\}$ for each entry $1 \leq i \leq M$, the corresponding substrings maintained in $\texttt{SU}$ should also be unique, and with minimum length $L_{\min}$. One should be careful however when dealing with $\texttt{LCP}_d$: if $\texttt{LCP}_u[i] \leq L_{\min} - 1$, then the corresponding substring $s[\texttt{SA}[i], \texttt{SA}[i] + \texttt{LCP}_u[i]]$ occurs in only one genome. Therefore we set $\texttt{LCP}_d[i]$ to ∞ (meaning it's not considered) if $\texttt{LCP}_u[i] \leq L_{\min} - 1$ or $\texttt{LCP}_d[i] \geq \texttt{LCP}_u[i]$; and to $\max\{L_{\min} - 1, \texttt{LCP}_d[i]\}$ otherwise (this can be done by first set each $\texttt{LCP}_u[i]$ to $\max\{L_{\min} - 1, \texttt{LCP}_u[i]\}$ and $\texttt{LCP}_d[i]$ to $\max\{L_{\min} - 1, \texttt{LCP}_d[i]\}$, and then set each $\texttt{LCP}_d[i]$ to ∞ if $\texttt{LCP}_d[i] \geq \texttt{LCP}_u[i]$), which ensures the corresponding substrings maintained in $\texttt{SD}$ are doubly-unique, and with minimum length $L_{\min}$.

With the correctness of (3) and (4) in mind, our next concern is how to actually compute $\texttt{LCP}_u$ and $\texttt{LCP}_d$ based on their definitions. In the following we show that (5) and (6) correctly implement (1) and (2), without considering the borderline cases (i.e., for $i = 1$ or $i = M$; to handle these cases we can set $\texttt{GSA}[0] = \texttt{GSA}[M + 1] = 0$ and ignore i^{2-} when $i = 1$ and i^{2+} when $i = M$).

Lemma 3. *For any* $1 \leq i \leq M$*,*

$$(i)\texttt{LCP}_u[i] = \max \begin{cases} \min\limits_{i^- < x \leq i} \texttt{LCP}[x], \ \textit{where } i^- = \max\{1 \leq i' < i\}, \ \textit{s.t. } d_{\texttt{GSA}}(i', i) \geq 2 \\ \min\limits_{i < x \leq i^+} \texttt{LCP}[x], \ \textit{where } i^+ = \min\{i < i' \leq M\}, \ \textit{s.t. } d_{\texttt{GSA}}(i, i') \geq 2 \end{cases}$$

$$(ii)\texttt{LCP}_d[i] = \min \begin{cases} \max \begin{cases} \min\limits_{i^- < x \leq i} \texttt{LCP}[x], \ \textit{where } i^- = \max\{1 \leq i' < i\}, \ \textit{s.t. } d_{\texttt{GSA}}(i', i) \geq 2 \\ \min\limits_{i < x \leq i^{2+}} \texttt{LCP}[x], \ \textit{where } i^{2+} = \min\{i < i' \leq M\}, \ \textit{s.t. } d_{\texttt{GSA}}(i, i') \geq 3 \end{cases} \\ \max \begin{cases} \min\limits_{i^{2-} < x \leq i} \texttt{LCP}[x], \ \textit{where } i^{2-} = \max\{1 \leq i' < i\}, \ \textit{s.t. } d_{\texttt{GSA}}(i', i) \geq 3 \\ \min\limits_{i < x \leq i^+} \texttt{LCP}[x], \ \textit{where } i^+ = \min\{i < i' \leq M\}, \ \textit{s.t. } d_{\texttt{GSA}}(i, i') \geq 2 \end{cases} \end{cases}$$

where $d_{\texttt{GSA}}(i_1, i_2) = |\{\texttt{GSA}[i_1], \cdots, \texttt{GSA}[i_2]\}|$.

Proof. Let $\text{lcp}(i, j)$ denote $\text{lcp}(\texttt{suf}[i], \texttt{suf}[j])$ for short. We utilize the properties of $\texttt{SA}$ and $\texttt{LCP}$ array: (a) the longest common prefix of two suffices $\texttt{suf}[i]$ and $\texttt{suf}[j]$ (assume $\texttt{suf}[i]$ is lexicographically smaller than $\texttt{suf}[j]$) is $\text{lcp}(i, j) = \min\{\texttt{LCP}[x] \mid x \in [\texttt{SA}^{-1}[i]+1, \texttt{SA}^{-1}[j]]\}$; also we have (b) $\text{lcp}(i, j) \geq \text{lcp}(\texttt{SA}[\texttt{SA}^{-1}[i]-1], j)$ and $\text{lcp}(i, j) \geq \text{lcp}(i, \texttt{SA}[\texttt{SA}^{-1}[j] + 1])$. (i) follows immediately from these properties:

$$\texttt{LCP}_u[i] = \max\{\text{lcp}(\texttt{SA}[i], \texttt{SA}[i^+]), \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i])\}.$$

To see (ii), we consider three cases:

- If $\text{lcp}(\texttt{SA}[i], \texttt{SA}[i^+]) = \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i])$, then $\texttt{LCP}_d[i] = \text{lcp}(\texttt{SA}[i], \texttt{SA}[i^+]) = \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i])$, due to (b). (ii) holds in this case by applying (a).
- If $\text{lcp}(\texttt{SA}[i], \texttt{SA}[i^+]) < \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i])$, then $\texttt{LCP}_d[i] = \max\{\text{lcp}(\texttt{SA}[i], \texttt{SA}[i^+]), \text{lcp}(\texttt{SA}[i^{2-}], \texttt{SA}[i])\}$ due to (b). Also $\texttt{LCP}_d[i] \leq \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i]) = \max\{\text{lcp}(\texttt{SA}[i], \texttt{SA}[i^{2+}]), \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i])\}$ since $\text{lcp}(\texttt{SA}[i], \texttt{SA}[i^{2+}]) \leq \text{lcp}(\texttt{SA}[i], \texttt{SA}[i^+]) < \text{lcp}(\texttt{SA}[i^-], \texttt{SA}[i])$. (ii) therefore holds by applying (a).

- If $\mathrm{lcp}(\mathtt{SA}[i], \mathtt{SA}[i^+]) > \mathrm{lcp}(\mathtt{SA}[i^-], \mathtt{SA}[i])$, then we have similarly $\mathtt{LCP}_d[i] = \max\{\mathrm{lcp}(\mathtt{SA}[i], \mathtt{SA}[i^{2+}]), \mathrm{lcp}(\mathtt{SA}[i^-], \mathtt{SA}[i])\}$ due to (b) and $\mathtt{LCP}_d[i] \leq \mathrm{lcp}(\mathtt{SA}[i], \mathtt{SA}[i^+]) = \max\{\mathrm{lcp}(\mathtt{SA}[i], \mathtt{SA}[i^+]), \mathrm{lcp}(\mathtt{SA}[i^{2-}], \mathtt{SA}[i])\}$ since $\mathrm{lcp}(\mathtt{SA}[i^{2-}], \mathtt{SA}[i]) \leq \mathrm{lcp}(\mathtt{SA}[i^-], \mathtt{SA}[i]) < \mathrm{lcp}(\mathtt{SA}[i], \mathtt{SA}[i^+])$. Again we can see (ii) by applying (a), which complete the proof.

□

Next we give the pseudocode for the algorithm to update $\mathtt{LCP}_u$ and $\mathtt{LCP}_d$ based on (5) and (6) respectively, and show that they actually run in linear time.

Algorithm 2 LCP_uFromLCP($\mathtt{GSA}$, $\mathtt{LCP}$, $L_{\min}$)

```
LCP_u ← array of length M
i ← 1, minlcp ← M, next ← 0
while i < M do
    while i < M and GSA[i + next] = GSA[i + next + 1] do
        next ← next + 1
    end while
    for j from next to 0 do
        minlcp ← min{minlcp, LCP[i + j + 1]}
        LCP_u[i + j] ← minlcp
    end for
    i ← i + next + 1
    minlcp ← M, next ← 0
end while
i ← M, minlcp ← M, next ← 0
while i > 1 do
    while i > 1 and GSA[i − next] = GSA[i − next − 1] do
        next ← next + 1
    end while
    for j from next to 0 do
        minlcp ← min{minlcp, LCP[i − j]}
        LCP_u[i − j] ← max{LCP_u[i − j], minlcp}
    end for
    i ← i − next − 1
    minlcp ← M, next ← 0
end while
for i from 1 to M do
    LCP_u[i] ← max{LCP_u[i], L_min}
end for
return LCP_u
```

Lemma 4. *Both LCP_uFromLCP and LCP_dFromLCP run in $O(M)$ time.*

Proof. Through a simple aggregate analysis, we can see that LCP_uFromLCP visits each entry of $\mathtt{GSA}$, $\mathtt{LCP}$ and $\mathtt{LCP}_u$ 2 times; LCP_dFromLCP* visits each entry of $\mathtt{GSA}$, $\mathtt{LCP}$ 3 times and each entry of $\mathtt{LCP}^*_d$ 2 times for either focus $= +/-$. □

Combining the above lemmata, we concluse that

Theorem 5. *Both $\mathtt{SU}$ and $\mathtt{SD}$ can be computed in in $O(M)$ time.*

5.3 Sampling unique substrings

Recall that $\mathcal{U}_i = \{u_{i,1}(l_1, r_1), u_{i,nu_i}(l_{nu_i}, r_{nu_i})\}$ defines either the collection of all shortest unique substrings or unique substrings with minimum length $L_{\min}$ on a given genome s_i (sorted by l_x, namely $l_1 \leq \cdots \leq l_{nu_i}$). In fact, the list of left indices $l_1, \cdots, l_{nu_i}$ are stored in the corresponding $r_1, \cdots, r_{nu_i}$ entries in $\mathtt{SU}$ array. Due to the minimum length constraint, no substring $u_{i,x} \in \mathcal{U}_i$ can be a substring of any other $u_{i,y} \in \mathcal{U}_i$ if they are not identical (in fact, there could be some $\{u_{i,x}(l_x, r_x) = \{u_{i,y}(l_y, r_y) \in \mathcal{U}_i$ for $x \neq y$). This makes $l_1 < \cdots < l_{nu_i}$ and $r_1 < \cdots < r_{nu_i}$. The goal of sampling unique substrings from $\mathcal{U}_i$ is to identify and maintain the smallest number of unique substrings such that they cover the same set of unique L-mers on s_i as $\mathcal{U}_i$ (if $L_{\max} = L$ then the sampled unique substrings should cover all unique L-mers).

Here we present the greedy sampling strategy implemented by `CAMMiQ` to sample unique substrings from $\mathcal{U}_i$. Denote by begin_i the beginning position of s_i in s and by $\mathcal{U}'_i$ the unique substrings already sampled (initially $\mathcal{U}'_i$ is empty). Starting with begin_i, consider every L-mer of s_i from left to right; if it does not include any unique substring, then ignore this L-mer; otherwise add its rightmost unique substring into $\mathcal{U}'_i$ and move to the next L-mer which does not include this substring until reaching the L-mer that ends at

Algorithm 3 LCP$_d$FromLCP(GSA, LCP, $L_{\min}$)

$\texttt{LCP}_d \leftarrow$ array of length M
$\texttt{LCP}_u \leftarrow \text{LCP}_u\text{FromLCP}(\texttt{GSA}, \texttt{LCP}, M, L_{\min})$
$\texttt{LCP}_{d'} \leftarrow \text{LCP}_d\text{FromLCP*}(\texttt{GSA}, \texttt{LCP}, +)$
$\texttt{LCP}_{d''} \leftarrow \text{LCP}_d\text{FromLCP*}(\texttt{GSA}, \texttt{LCP}, -)$
for i **from** 1 **to** M **do**
  $\texttt{LCP}_d[i] \leftarrow \max\{\min\{\texttt{LCP}_{d'}[i], \texttt{LCP}_{d''}[i]\}, L_{\min}\}$
  if $\texttt{LCP}_d[i] \geq \texttt{LCP}_u[i]$ **then**
    $\texttt{LCP}_d[i] \leftarrow \infty$
  end if
end for
return $\texttt{LCP}_d$
function LCP$_d$FromLCP*(GSA, LCP, focus= $\{+, -\}$)
  $\texttt{LCP}^*_d \leftarrow$ array of length M
  $i \leftarrow 1$, minlcp $\leftarrow M$, next1 $\leftarrow 0$
  if focus $= +$ **then**
    next2 $\leftarrow 1$
  else
    next2 $\leftarrow 0$
  end if
  while $i < M$ **do**
    while $i < M$ and $\texttt{GSA}[i + \text{next1}] = \texttt{GSA}[i + \text{next1} + 1]$ **do**
      next1 $\leftarrow$ next1 $+ 1$
    end while
    while focus $= +$ and $i < M$ and $\texttt{GSA}[i + \text{next1} + \text{next2}] = \texttt{GSA}[i + \text{next1} + \text{next2} + 1]$ **do**
      next2 $\leftarrow$ next2 $+ 1$
    end while
    for j **from** next1 $+$ next2 **to** 0 **do**
      minlcp $\leftarrow \min\{\text{minlcp}, \texttt{LCP}[i + j + 1]\}$
      if $j \leq$ next1 **then**
        $\texttt{LCP}^*_d[i + j] \leftarrow$ minlcp
      end if
    end for
    $i \leftarrow i +$ next1 $+ 1$
    minlcp $\leftarrow M$, next1 $\leftarrow 0$
    if focus $= +$ **then**
      next2 $\leftarrow 1$
    else
      next2 $\leftarrow 0$
    end if
  end while
  $i \leftarrow M$, minlcp $\leftarrow M$, next1 $\leftarrow 0$
  if focus $= -$ **then**
    next2 $\leftarrow 1$
  else
    next2 $\leftarrow 0$
  end if
  while $i > 1$ **do**
    while $i > 1$ and $\texttt{GSA}[i - \text{next1}] = \texttt{GSA}[i - \text{next1} - 1]$ **do**
      next1 $\leftarrow$ next1 $+ 1$
    end while
    while focus $= -$ and $i > 1$ and $\texttt{GSA}[i - \text{next1} - \text{next2}] = \texttt{GSA}[i - \text{next1} - \text{next2} - 1]$ **do**
      next2 $\leftarrow$ next2 $+ 1$
    end while
    for j **from** next1 $+$ next2 **to** 0 **do**
      minlcp $\leftarrow \min\{\text{minlcp}, \texttt{LCP}[i - j]\}$
      if $j \geq$ next2 **then**
        $\texttt{LCP}^*_d[i - j] \leftarrow \max\{\texttt{LCP}^*_d[i - j], \text{minlcp}\}$
      end if
    end for
    $i \leftarrow i -$ next1 $- 1$
    minlcp $\leftarrow M$, next1 $\leftarrow 0$
    if focus $= -$ **then**
      next2 $\leftarrow 1$
    else
      next2 $\leftarrow 0$
    end if
  end while
  return $\texttt{LCP}^*_d$
end function

$\text{begin}_i + |s_i| - 1$. At the and of this, add the sampled unique substrings in $\mathcal{U}'_i$ to the hash table described in Section 2.1.

In the following we show that the above greedy strategy obtains the smallest number of unique substrings that cover the same set of unique L-mers as $\mathcal{U}_i$, provided that each unique substring in $\mathcal{U}_i$ occurs only one time (i.e., any $u_{i,x} \in \mathcal{U}_i$ is not identical to another $u_{i,y} \in \mathcal{U}_i$ if $x \neq y$). As a result, the total number of unique substrings in $\mathcal{U}' = \cup_{i=1}^{m} \mathcal{U}'_i$ included in CAMMiQ index is also as small as possible.

Theorem 6. *If $u_{i,x} \in \mathcal{U}_i \neq u_{i,y} \in \mathcal{U}_i$ for $x \neq y$, then GreedySampling returns the smallest $\mathcal{U}'_i$ such that if an L-mer includes some $u_{i,x} \in \mathcal{U}_i$, then it also includes at least one $u_{i,x'} \in \mathcal{U}'_i$.*

Proof. Consider $\mathcal{U}'_i = \{u'_{i,1}(l'_1, r'_1), \cdots, u'_{i,|\mathcal{U}'_i|}(l'_{|\mathcal{U}'_i|}, r'_{|\mathcal{U}'_i|})\}$ that GreedySampling returns; also consider an alternative sample $\mathcal{U}''_i = \{u''_{i,1}(l''_1, r''_1), \cdots, u''_{i,|\mathcal{U}''_i|}(l''_{|\mathcal{U}''_i|}, r''_{|\mathcal{U}''_i|})\}$ of $\mathcal{U}_i$ that covers the same set of L-mers; assume both sets are sorted by the left indices ($l'_1 < \cdots < l'_{|\mathcal{U}'_i|}$; $l''_1 < \cdots < l''_{|\mathcal{U}''_i|}$). First, observe that $l'_1 \geq l''_1$

Algorithm 4 GreedySampling($\mathcal{U}_i = \{u_{i,1}(l_1, r_1), u_{i,nu_i}(l_{nu_i}, r_{nu_i})\}$, L)

$\mathcal{U}'_i \leftarrow \emptyset$
cur $\leftarrow$ begin$_i$ // begin$_i$: the beginning position of s_i in s
$u_{i,\text{last}}(l_{\text{last}}, r_{\text{last}}) \leftarrow$ NIL
for x **from** 1 to nu_i **do**
  if $u_{i,\text{last}} \neq$ NIL and $r_x \geq$ cur $+ L$ **then**
    $\mathcal{U}'_i \leftarrow \mathcal{U}'_i \cup \{u_{i,\text{last}}(l_{\text{last}}, r_{\text{last}})\}$
    cur $\leftarrow l_{\text{last}} + 1$
  end if
  $u_{i,\text{last}}(l_{\text{last}}, r_{\text{last}}) \leftarrow u_{i,x}(l_x, r_x)$
end for
return $\mathcal{U}'_i$

since $u'_{i,1}$ is the rightmost unique substring in $\mathcal{U}_i$ which is fully included in the leftmost unique L-mer. As a consequence we must have $l'_2 \geq l''_2$ (and so on). Otherwise, if $l'_1 = l''_1$ then $u'_{i,2}(l'_2, r'_2)$ is not the rightmost unique substring in the next unique L-mer whose left index is greater than l'_1, and if $l'_1 > l''_1$ then there is some unique L-mer not covered by any unique substring in $\mathcal{U}''_i$. Therefore $|\mathcal{U}'_i| \leq |\mathcal{U}''_i|$ since there is a injection between the elements in $\mathcal{U}'_i$ and $\mathcal{U}''_i$ until reaching the last unique L-mer on s_i. □

Corollary 7. *If $u_{i,x} \in \mathcal{U}_i \neq u_{i,y} \in \mathcal{U}_i$ for $x \neq y$ on each s_i, then $\mathcal{U}' = \cup_{i=1}^{m} \mathcal{U}'_i$ is the smallest set of unique substrings such that if an L-mer on s_i includes some $u_{i,x} \in \mathcal{U}_i$, then it also includes at least one $u_{i,x'} \in \mathcal{U}'_i$.*

We note that the greedy sampling strategy works in practice even if there are actually unique substrings occurring more than once in a given genome, meaning $u_{i,x} = u_{i,y} \in \mathcal{U}_i$ for some $y > x$, leading to $\mathcal{U}'_i$ (and thus $\mathcal{U}'$) being close to optimality, since these unique substrings would constitute a very small proportion ($\leq 0.1\%$) with the default minimum length $L_{\min} = 26$ of unique substrings. This gives significantly smaller indices than alternative k-mer based tools, and results in an integer program with a small number of constraints.

We applied the above strategy to sample doubly-unique substrings in $\mathcal{D}_i$, to obtain the minimum size $\mathcal{D}'_i$ for each genome s_i so that the aggregate set of doubly-unique substrings $\mathcal{D}'$ is at most twice the optimal, provided that each doubly-unique substring occurs once in each of the corresponding genomes.

Corollary 8. *If $d_{i,x} \in \mathcal{D}_i \neq d_{i,y} \in \mathcal{D}_i$ for $x \neq y$ on each s_i, then $\mathcal{D}' = \cup_{i=1}^{m} \mathcal{D}'_i$ is at most twice as large as the smallest set of doubly-unique substrings such that if an L-mer on s_i includes some $d_{i,x} \in \mathcal{D}_i$, then it also includes at least one $d_{i,x'} \in \mathcal{D}'_i$.*

5.4 Proof of Theorem 1

We begin with the following theorem from Weissman et al. [41] that bounds the L1 distance between the empirical distribution of a sequence of independent, identically distributed random variables and the true distribution.

Theorem 9. *Let P be a probability distribution on the set $\mathcal{A} = \{1, \cdots, a\}$. Let $X_1, X_2, \cdots, X_n$ be i.i.d. random variables distributed according to P. Then, for any given $\epsilon > 0$,*

$$Pr[||P - \bar{P}||_1 \geq \epsilon] \leq (2^a - 2)\exp(-n\epsilon^2/2)$$

where $\bar{P}$ is the empirical estimation of P defined as $\bar{P}(i) = \frac{\sum_{j=1}^{n} \Delta(X_i = j)}{n}$, where $\Delta(e) = 1$ if and only if e is true and $\Delta(e) = 0$ otherwise.

Now consider a collection of genomes $\mathcal{A} = \{s_1, \cdots, s_a\}$ with relative abundances $p_1, \cdots, p_a$ and the set $\mathcal{Q} = \{r_1, \cdots, r_n\}$ of n reads (i.e., L-mers) sampled independently and uniformly at random from $\mathcal{A}$ according to $p_1, \cdots, p_a$. On each genome s_i let n_i^L denote the total number of L-mers and $q_i = \frac{nu_i^L}{n_i^L}$ be the proportion of unique L-mers; then the probability of a read $r_j \in \mathcal{Q}$ corresponds to a unique L-mer on s_i is $\frac{p_i n_i^L}{\sum_{i'=1}^{a} p_{i'} n_{i'}^L} \cdot q_i$, and the probability of a read $r_j \in \mathcal{Q}$ does not correspond to any unique L-mers on s_i is $\frac{p_i n_i^L}{\sum_{i'=1}^{a} p_{i'} n_{i'}^L} \cdot (1 - q_i)$.

Therefore $r_1, \cdots, r_j$ are i.i.d. distributed according to $(\frac{p_1 n_1^L}{\sum_{i'=1}^a p_{i'} n_{i'}^L} \cdot q_1, \cdots, \frac{p_a n_a^L}{\sum_{i'=1}^a p_{i'} n_{i'}^L} \cdot q_a, \sum_{i=1}^a \frac{p_i n_i^L}{\sum_{i'=1}^a p_{i'} n_{i'}^L} \cdot (1-q_i)) = (p'_1 q_1, \cdots, p'_a q_a, \sum_{i=1}^a p'_i(1-q_i))$ where the last term corresponds to the reads that are not unique to any $s_i \in \mathcal{A}$.

Proof of Theorem 1. Let c_i be the number of reads assigned to s_i. Also let $p' = (p'_1, \cdots, p'_a)$. We set $\hat{p} = (\hat{p_1}, \cdots, \hat{p_a})$, by defining $\hat{p_i} = \frac{c_i/q_i}{n}$ to be the predicted abundance of s_i based on the number of reads assigned to it. Consider $P = (p'_1 q_1, \cdots, p'_a q_a, \sum_{i=1}^a p'_i(1-q_i))$ and $\hat{P} = (\frac{c_1}{n}, \cdots, \frac{c_a}{n}, 1 - \sum_{i=1}^a \frac{c_i}{n}) = (\hat{p_1} q_1, \cdots, \hat{p_a} q_a, 1 - \sum_{i=1}^a \frac{c_i}{n})$; by definition, we have $||P - \hat{P}||_1 \geq \sum_{i=1}^a |p'_i - \hat{p_i}| q_i \geq q_{\min} \cdot \sum_{i=1}^a |p'_i - \hat{p_i}| = q_{\min} \cdot ||p' - \hat{p}||_1$. Then the following three theorem statements hold.

- (i) Given that $n \geq \frac{2(a+1)+\ln\frac{1}{\delta}}{(p_{\min} q_{\min})^2}$, we have

$$\begin{aligned}
\Pr\left[||p' - \hat{p}||_1 \geq p_{\min}\right] &= \Pr\left[q_{\min}||p' - \hat{p}||_1 \geq p_{\min} q_{\min}\right] \\
&\leq \Pr\left[||P - \hat{P}||_1 \geq p_{\min} q_{\min}\right] \\
&\leq 2^{a+1} \exp(-n(p_{\min} q_{\min})^2/2) \\
&\leq 2^{a+1} \exp(-\frac{2(a+1)+\ln\frac{1}{\delta}}{(p_{\min} q_{\min})^2}(p_{\min} q_{\min})^2/2) \\
&= \frac{2^{a+1}}{e^{a+1}}\delta \\
&\leq \delta.
\end{aligned}$$

  This implies that with probability $\geq 1-\delta$ the L1 distance between p' and $\hat{p}$ is upper bounded by $p_{\min}$. As a result we have $\hat{p_i} > 0$ for each $\hat{p_i}$, i.e. $c_i \geq 0$.

- (ii) The proof follows by simply replacing $p_{\min}$ with ϵ in the proof of (i).

- (iii) The proof follows by simply replacing $p_{\min}$ with $\sqrt{\frac{2(\ln\frac{1}{\delta}+(a+1))}{n q_{\min}^2}}$ in the proof of (i). Specifically,

$$\begin{aligned}
\Pr\left[||p' - \hat{p}||_1 \geq \sqrt{\frac{2(\ln\frac{1}{\delta}+(a+1))}{n q_{\min}^2}}\right] &= \Pr\left[q_{\min}||p' - \hat{p}||_1 \geq q_{\min} \cdot \sqrt{\frac{2(\ln\frac{1}{\delta}+(a+1))}{n q_{\min}^2}}\right] \\
&\leq \Pr\left[||P - \hat{P}||_1 \geq \sqrt{\frac{2(\ln\frac{1}{\delta}+(a+1))}{n}}\right] \\
&\leq 2^{a+1} \exp(-\frac{n}{2} \cdot \frac{2(\ln\frac{1}{\delta}+(a+1))}{n}) \\
&= \frac{2^{a+1}}{e^{a+1}}\delta \\
&\leq \delta.
\end{aligned}$$

6 Supplementary Figures

Figure 4 presents 50 genomes in our species-level dataset with the least number of unique L-mers. In this graph each node represents one such genome; each edge connects two nodes if they share a doubly-unique substring. Solid black edges indicate a pair of nodes that share at least 30 doubly-unique substrings; the remaining edges in grey indicate node pairs with fewer number of shared doubly-unique substrings. Notice that there is a special node in the center, representing the union of all genomes not included in these 50-genome subset. Any node connected to this special node by a single edge, or by a path, is identifiable and quantifiable by `CAMMiQ`, provided that all edges in this path are black (22 of these 50 nodes are as such) or all nodes in this path have sufficient abundance. Note that 20 of the genomes here are connected to this special node by a black edge: these are the genomes that form the least-quantifiable-20 dataset in our experiments.

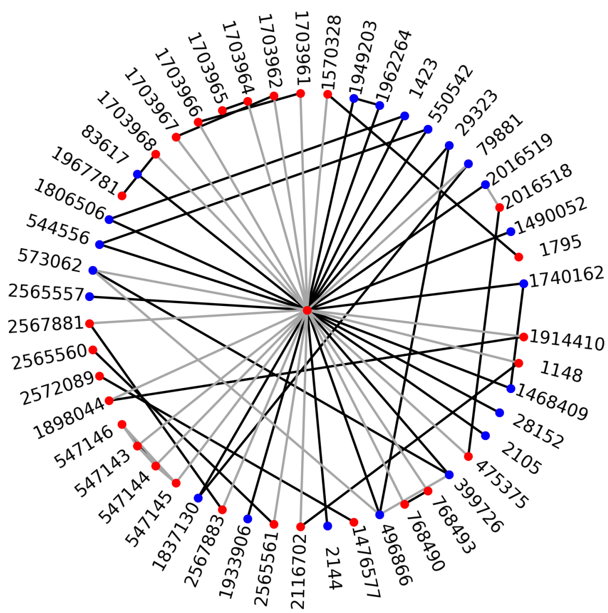

Figure 4: Shared doubly-unique substrings among the 50 genomes with the least number of unique L-mers in our species-level dataset consisting of 4122 RefSeq bacteria genomes. Each node represents one of these 50 genomes, labeled with its NCBI taxID at species level. The central node specially represents the remaining 4072 genomes. An edge connecting two nodes indicate at least one doubly-unique substring shared between them. A black edge indicates ≥ 30 doubly-unique substrings in `CAMMiQ`'s index shared between the two corresponding genomes. All other edges in grey imply < 30 shared doubly-unique substrings. A blue-colored node indicates one that is connected to the central node through a path of black edges. As such, they are relatively easy to identify and quantify; 22 of these 50 nodes are blue. The remaining (red) nodes can be identified by `CAMMiQ` provided they have "sufficient abundance" in the query.