VA-Store: A Virtual Approximate Store Approach to Supporting Repetitive Big Data in Genome Sequence Analyses

Xianying Liu, Qiang Zhu, *Senior Member, IEEE*, Sakti Pramanik, C. Titus Brown, and Gang Qian

Abstract—In recent years, we have witnessed an increasing demand to process big data in numerous applications. It is observed that there often exist substantial amounts of repetitive data in different portions of a big data repository/dataset for applications such as genome sequence analyses. In this paper, we present a novel method, called the VA-Store, to reduce the large space requirement for repetitive data in prevailing genome sequence analysis tasks using k-mers (i.e., subsequences of length k) with multiple k values. The VA-Store maintains a physical store for one portion of the input dataset (i.e., k_0-mers) and supports multiple virtual stores for other portions of the dataset (i.e., k-mers with $k \neq k_0$). Utilizing important relationships among repetitive data, the VA-Store transforms a given query on a virtual store into one or more queries on the physical store for execution. Both precise and approximate transformations are considered. Accuracy estimation models for approximate solutions are derived. Query optimization strategies are suggested to improve query performance. Our experiments using real and synthetic datasets demonstrate that the VA-Store is quite promising in providing effective storage and efficient query processing for solving a kernel database problem on repetitive big data for genome sequence analysis applications.

Index Terms—Bioinformatics (genome or protein) databases, data storage representations, query processing, algorithms for data and knowledge management

1 Introduction

There is an increasing demand to process big data from numerous data-intensive applications such as bioinformatics, scientific experiments/simulations, e-commerce, social media, and cloud computing. Extensive research has been conducted to deal with challenges faced in all the phases (e.g., acquisition, cleaning, integration, modeling/analysis, and interpretation/visualization) of big data processing [23]. One interesting observation is that there often exist substantial amounts of repetitive data in different portions of a big data repository for many applications such as genome sequence analysis problems in bioinformatics. Techniques are required to support effective data storage and efficient query processing for such big data applications. In this paper, we propose a new virtual approximate store approach, called the VA-Store, to effectively supporting repetitive big data for genome sequence analyses. The working principle of this technique is also applicable to other application domains.

Biologists have been storing and analyzing massive volume of genomic data. The European Bioinformatics Institute (EBI) in Hinxton, UK, one of the world's largest biology-data repositories, currently stores 20 petabytes of data and backups about genes, proteins and small molecules, where genomic data accounts for 2 petabytes and grows at a rate more than double every year [36]. The rapid growth of genomic data is attributed to the vast increase in DNA sequencing capacity over the last decade. Due to the next-generation sequencing technologies, researchers can now conduct whole-genome sequencing and generate a huge volume of genome sequencing data. Efficient computational analysis of genome sequencing data becomes a fast-growing challenge.

Over the last decade, a wide variety of (genome) sequence analysis approaches have been developed that make use of fixed-length subsequences, called k-mers (where k is the length), obtained from reads (i.e., short fragments) generated by a sequencer (machine) such as Illumina [26] for a target genome. Fig. 1 illustrates how a genome sequence read is decomposed into a set of shifted k-mers with $k = 5$.

One example sequence analysis problem that can be solved by using k-mers is local alignment searching. The goal of the problem is to compare a query sequence (read) q with a set S of sequences (reads) and identify those reads in S that resemble q. The popular BLAST approach [2], [5] to solving this problem works as follows: (1) decompose q and each read in S into shifted k-mers for a fixed length k; (2) locate common k-mers between q and a (hit) read r in S; (3) extend qualified

- *X. Liu and Q. Zhu are with the Department of Computer and Information Science, University of Michigan - Dearborn, Dearborn, MI 48128. E-mail: xyliu.365@gmail.com, qzhu@umich.edu.*
- *S. Pramanik is with the Department of Computer Science and Engineering, Michigan State University, East Lansing, MI 48824. E-mail: pramanik@cse.msu.edu.*
- *C.T. Brown is with the Genome Center, University of California, Davis, CA 95616. E-mail: ctbrown@ucdavis.edu.*
- *G. Qian is with the Department of Computer Science, University of Central Oklahoma, Edmond, OK 73034. E-mail: gqian@ucok.edu.*

Manuscript received 27 May 2016; revised 29 Sept. 2018; accepted 23 Nov. 2018. Date of publication 11 Dec. 2018; date of current version 4 Feb. 2020. (Corresponding author: Qiang Zhu.)
Recommended for acceptance by Y. Zhang.
Digital Object Identifier no. 10.1109/TKDE.2018.2885952

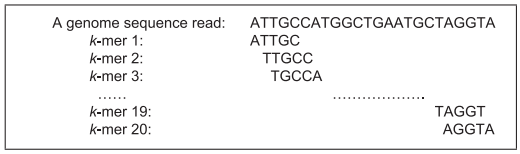


Fig. 1. Shifted k-mers for a genome sequence read.

k-mers from (2) to determine the similarity between q and r using a scoring matrix; (4) include r in the returned query result if its score exceeds a specified threshold.

Another sequence analysis problem that can be solved by using k-mers is the sequence assembly of a set S of reads for a target genome into longer contiguous (sub-) sequences, called contigs, in the genome. The basic idea of the *de Bruijn* graph-based assembly approach [12] works as follows: (1) decompose each read in S into shifted k-mers for a fixed length k; (2) construct a *de Bruijn* graph for the k-mers obtained from (1), which represents overlaps among all the k-mers; (3) traverse the graph to determine all the contigs or identify a particular contig that contains a given k-mer.

There are many other sequence analysis problems that can be solved by utilizing k-mers, such as terminus searching [10] and sequencing error correction [21]. The goal of terminus searching is to identify the genome sequences (reads) that end with a given terminus (k-mer) α, which can be realized by searching α in the set of k-mers that appear at the tail of each interesting genome sequence (read) from a given repository. The goal of sequencing error correction is to detect a possibly erroneous letter/base at a suspicious position in a read that was produced by a sequencer such as Illumina. Since each position in a target genome sequence is typically covered by a number (e.g., 20) of reads generated from the sequencing, the possibly erroneous letter at position p from read r can be verified as follows: (1) decompose all the reads into k-mers for a fixed length k; (2) count the frequencies of two k-mers α_1 and α_2, where α_1 and α_2 are the same except having different letters at position p; (3) if the frequency of α_1 is much smaller than that of α_2, we may conclude that the read containing α_1 has an erroneous letter at position p and the correct letter at position p is most likely the letter from α_2 at position p. To improve the accuracy, all the shifted k-mers overlapping with the position p could be used in (2) and a comprehensive cross-examining voting scheme could be applied in (3) [15], [16]. Once an erroneous letter is found, the correction is just a matter of replacing the erroneous letter by the correct one.

We notice that all of the above genome sequence analysis problems/applications contain the following Kernel Database Problem (KDBP$_k$): search a query k-mer q in a given set Ω_k of k-mers, where each k-mer α in Ω_k is associated with some metadata (e.g., annotations) u, and return the metadata associated with q if it is found in Ω_k.

What metadata u is depends on the underlying application. For example, u is the id(s) of the read(s) containing α for the applications of local alignment searching and terminus searching; u is the frequency of α in the reads for the application of sequencing error correction; u is the id of the node in the *de Bruijn* graph representing α and the id(s) of the associated read(s) for the application of sequence assembly. Once the metadata associated with the query k-mer(s) is returned, the corresponding application (e.g., sequencing error correction) needs to further process them (e.g., applying a voting mechanism for sequencing error correction [15]) to finish its task.

Most existing methods for genome sequence analysis applications adopt an in-memory structure (e.g., hash table, suffix tree, etc.) to solve the above KDBP problem, which requires huge memory space and does not scale well. Furthermore, most existing techniques were developed to support only one set Ω_k of k-mers for a single pre-determined length k. On the other hand, many sequence analysis problems are very sensitive to the choice of k for Ω_k. For example, for the local alignment searching problem, using k-mers with a small k might yield some useless hit reads for the final expensive alignment process, while using k-mers with a large k could cause some useful reads to be missing in the result although the processing might be more efficient. For the sequence assembly problem, using k-mers with a small k might make the resulting *de Bruijn* graph include more spurious edges and nodes, while using k-mers with a large k might make the graph become sparse and possibly disconnected. For the terminus searching problem, terminuses (k-mers) with different lengths may contain important information (e.g., age information in human genomes). For the sequencing error correction problem, using k-mers with a large k might help more accurately detect errors in the middle of a read, while using k-mers with a small k might help better detect errors near the boundaries of a read.

A straightforward approach to overcoming the limitations of a single-k technique is to store multiple sets of k-mers with different k values/lengths, e.g., Ω_m, $\Omega_{m+1}, \ldots \Omega_{m+M-1}$ ($m \geq 1$; $M > 1$). This approach would allow a user to choose a proper Ω_k ($m \leq k \leq m + M - 1$) to use for his/her application based on his/her specific requirements (e.g., efficiency versus accuracy). It would also allow one to develop techniques to utilize the results obtained from using multiple Ω_k's to improve quality of the final result [30]. However, this approach would require a large amount of space for its storage.

To overcome this difficulty, we notice that every k-mer in Ω_k is actually contained in at least one k'-mer in $\Omega_{k'}$ for $k \leq k'$, assuming Ω_k and $\Omega_{k'}$ are obtained from the same set of reads. Hence, data in Ω_k is largely repetitive if $\Omega_{k'}$ already exists. Physically storing multiple sets of k-mers clearly does not use space effectively. On the other hand, it is hard to determine what the maximum k' is and the space needed to store $\Omega_{k'}$ typically grows as k' increases.

In this paper, we propose a new virtual store approach to tackling the above problem. The key idea is to use only one physical store to physically store set Ω_{k_0} for one k_0 ($m \leq k_0 \leq m + M - 1$) and use virtual stores to logically (virtually) store the other useful sets Ω_k for all $k \neq k_0$ and $m \leq k \leq m + M - 1$ (see Fig. 2). The parameters m and M depend on the underlying sequence analysis application. We can view the union of all Ω_i's ($m \leq i \leq m + M - 1$) as the user's big dataset and various Ω_i's as different portions of the dataset.

A major challenge to realize the above approach is how to solve a given KDBP$_k$ (problem) on a virtually existing store Ω_k by using the physically existing store Ω_{k_0}. As we will see later, an KDBP$_k$ can actually be transformed into a set of KDBP$_{k_0}$'s on Ω_{k_0} that produces the same result as if KDBP$_k$ were solved by directly using Ω_k when $k \leq k_0$. However, when $k > k_0$, an KDBP$_k$ can be transformed only into a set of KDBP$_{k_0}$'s on Ω_{k_0} that usually produces an approximate result/solution for KDBP$_k$. This is

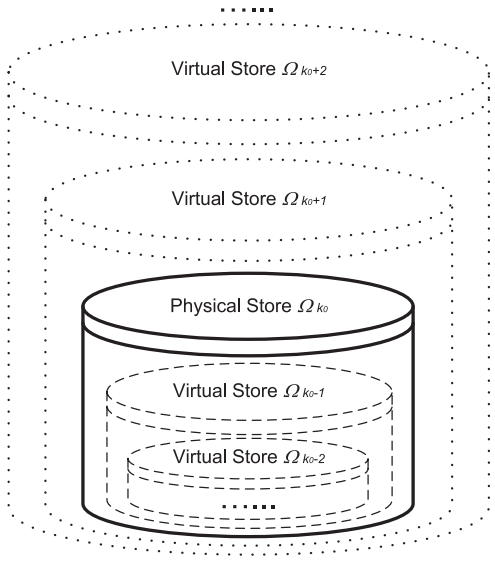


Fig. 2. Physical and virtual stores.

acceptable since genome sequence analysis problems typically seek approximate solutions in practice [12], [15], [21]. Note that an approximate solution in general may contain both false positives and false negatives, while an approximate solution produced by our method has no false negatives; that is, it can only have false positives (if any). In other words, true answers (e.g., reads matching a given query k-mer) are guaranteed to be included in our approximate solution.

In the subsequent sections of this paper, we will present the transformations in various cases, derive accuracy estimation models when approximate solutions are obtained, suggest query optimization strategies for query processing based on transformations, and present experimental evaluations. We utilize and extend the BoND-tree, which is a special index structure recently introduced by Chen et al. in [10] for processing box queries on genome sequencing data, to support efficient disk-based accesses for the physical store. Note that the focus of this work is on the higher level query transformations/optimization rather than the lower level implementation of the physical store. The BoND-tree implementation here can be replaced by any feasible method such as a Hadoop/MapReduce-based implementation.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 gives an overview of the concepts and notation needed by subsequent sections. Section 4 presents the technical details of our method including KDBP transformations, accuracy estimations, and query optimization. Section 5 reports the experimental results. Section 6 concludes the paper and highlights some future research directions.

2 Related Work

As mentioned earlier, most existing sequence analysis techniques were introduced based on k-mers of a single length k. For example, the BLAST-family tools [2], [5], [27], [38] were proposed for the local alignment searching application; the *de Bruijn* Graph method [12], the probabilistic *de Bruijn* Graph method [29], and the sparse bivector method [13] were suggested for the sequence assembly application; the Quake method [21] and several related k-mer counting tools such as Tallymer [34], Jellyfish [17] and DSK [18] were developed for the sequencing error correction application; and so on.

For a single-k technique, choosing an appropriate k value is crucial. Methods to choose such a k value were suggested [11], [21]. However, as mentioned earlier, using a single k value can never serve all scenarios well. To tackle the challenge for choosing a single optimal k value, researchers suggested several techniques to utilize multiple sets of k-mers for several k values [3], [30]. Unfortunately, space was not effectively used in these methods due to existence of large repetitive data. Boucher et al. [4] suggested a data structure to represent the *de Bruijn* graph $G_{K_{max}}$ for a given maximum K_{max} and presented algorithms to perform a sequence assembly task on a *de Bruijn graph* G_k for any $k \leq K_{max}$, which is the most related work that we found in the literature. The main differences between their work and ours are: (1) their technique is restricted to a specific data structure (i.e., the *de Bruijn* graph) for a specific application (i.e., the sequence assembly problem), while our work considers the generic kernel DB problem that can be applied to a number of sequence analysis applications; (2) their technique cannot support a data structure (i.e., *de Bruijn* graph) G_k for $k > K_{max}$, while our method supports a virtual store Ω_k that has a k length either smaller or larger than the k_0 length for the physical store Ω_{k_0}, i.e., both $k < k_0$ and $k > k_0$ are supported (see Fig. 2); (3) their method supports precise solutions only, while our work considers both precise and approximate solutions. To our knowledge, our work is the first of this type in the field.

Much work on indexing similar DNA sequences and string databases has been reported in the literature including [6], [7], [19], [20]. Most of them are memory based. Kahveci et al. [20] proposed a wavelet-based method to map substrings of a given database into a multidimensional integer space. They then introduced an in-memory index structure using minimum bounding rectangles for wavelet coefficients to prune a significant portion of the database from consideration when evaluating a given query. While this technique provides fast filtering and avoids false negatives, the cost for refining the final query result is still high. Furthermore, this technique was designed for a limited number of long strings/sequences since each string requires a separate index tree in their structure, which is not suitable for our sequence analysis applications that usually have to deal with a large number of reads (sequences). Cao et al. [7] proposed a distance-based sequence indexing method (DSIM) to use reference words and relative distances to compress a data sequence by representing neighbor overlapping k-mers of a fixed length in an index structure. However, this method does not support multiple lengths of k-mers. Cao et al. [6] presented a two-level (i.e., a hash table and c-trees) index for indexing DNA sequences efficiently based on k-mers to facilitate similarity searches. However, this method does not support multiple lengths of k-mers either. Huang et al. [19] proposed a scheme based on Burrows-Wheeler Transform and suffix arrays to index DNA sequences having some common segments. Although this scheme is space effective, it is only suitable for nearby overlapping reads/sequences and was not designed to handle a large number of (overlapping and non-overlapping) reads obtained for a long target DNA sequence as in our sequence analysis applications. Our work aims at supporting sequence analysis applications using k-mers of multiple lengths extracted from a large number of sequencing reads for a long target genome sequence(s). On the other hand, our work

focuses on studying higher level query transformations/optimization rather than a lower level indexing structure.

As mentioned earlier, researchers have been studying numerous issues in big data processing [23]. In particular, techniques have been developed to tackle the challenges for big data storage [24]. Chang et al. [8] introduced a distributed storage system, called the Bigtable, for managing large structured/semi-structured data. Akshay et al. [1] suggested supporting local storage in addition to networked storage in cloud systems so as to provide efficient support for Hadoop and other big data environments. Steinmaurer et al. [35] considered combining stream processing engines and big data storages for data analysis. Chen et al. [9] proposed a multilevel active storage for big data applications in high performance computing, which supports not only read-intensive operations but also write-intensive operations as well as complex operations with considerable computing demands. However, none of the techniques consider an approximate virtual storage that utilizes the relationships among repetitive data to improve the space effectiveness and query efficiency as we do here.

3 Preliminaries

Conceptually, a genome sequence η is a series of letters (bases) from alphabet $\Delta = \{A, G, T, C\}$, where letters A, G, T, C have no natural ordering among them. To obtain such a genome sequence in practice, a biologist typically uses a sequencer (e.g., Illumina) to generate reads (i.e., short fragments) of η from a target genome sample. Such a read may contain errors at some positions [21]. To reduce the effect of such sequencing errors, one usually uses multiple reads to cover each position of a genome sequence.

Reads are typically still too long to be efficiently analyzed/processed. To achieve an efficient sequence analysis, one can extract from each read subsequences with a fixed length k, called k-mers (see Fig. 1). Each k-mer α is typically associated with a piece of metadata u about the relevant read(s). Without loss of generality, in this paper, we assume that u is the list of ids for the reads containing α. Additional metadata (if any) can be accessed indirectly via the read ids in u. As mentioned earlier, numerous sequence analysis tasks can be performed by using k-mers with relevant metadata.

Note that there are two types of genome sequencing in biological sequence analyses: single-read sequencing and paired-end sequencing. Conventional single-read sequencing produces reads in one orientation (i.e., forward), while contemporary paired-end sequencing produces reads in both forward and reverse (complement) orientations. For simplicity, we assume that the input reads are given in one orientation (i.e., using reads from single-read sequencing or forward reads only from paired-end sequencing) in this paper. To handle both forward and reverse reads from paired-end sequencing, one could adopt the approach of using canonical representations of k-mers from the reads [15]. Specifically, when the shifted k-mers obtained from the input sequencing reads (in either orientation) are inserted into the physical store Ω_{k_0} of the VA-store, the canonical representation of each k-mer, which is either the k-mer itself or its reversed complement, is calculated and inserted into Ω_{k_0}. In other words, only one from each pair of (forward and reverse complement) k-mers is stored in Ω_{k_0}.

The Kernel Database Problem (KDBP$_k$) on a set Ω_k of k-mers is defined as follows: given a query k-mer q and the set Ω_k of k-mers obtained from a given set S of reads for a genome analysis application, find the following result set:

$$R(q) = \{x \mid x = r.id\ (\exists r \in S(\exists \alpha \in \Omega_k\ (r \sqsupseteq \alpha \wedge \alpha = q)))\}, \quad (1)$$

where $r.id$ is the id of read r and $r \sqsupseteq \alpha$ denotes that k-mer α is contained in read r. Informally, the above KDBP$_k$ is to perform a query, called the k-mer query,[1] to search the given k-mer q in Ω_k and return the set of ids of the reads in S that contain q if q is found in Ω_k. A challenge is how to process k-mer query q on a virtual store Ω_k (i.e., KDBP$_k$) by performing one or more k_0-mer queries on physical Ω_{k_0} (i.e., KDBP$_{k_0}$'s). This is the issue to be discussed in the next section.

As pointed by Qian et al. in [31], each k-mer can be viewed as a vector in a k-dimensional Non-ordered Discrete Data Space (NDDS): $\Gamma^k = D_1 \times D_2 \times \cdots \times D_k$, where $D_i = \{A, G, T, C\}$ $(1 \le i \le k)$ is the alphabet on the ith dimension. Hence, Ω_k is a dataset from Γ^k (i.e., $\Omega_k \subseteq \Gamma^k$). A discrete box/rectangle R in Γ^k is defined as $R = S_1 \times S_2 \times \cdots \times S_k$, where $S_i \subseteq D_i$. The area of rectangle R is defined as $|R| = \prod_{i=1}^{k} |S_i|$, where $|S_i|$ (i.e., the cardinality) is called the edge length of R along dimension i. For a set of vectors, their minimum bounding box is the smallest box that contains all the vectors. More concepts about an NDDS can be found in [31], [32], [33].

For a given box b, its corresponding box query $q(b)$ on a set Ω of vectors in an NDDS retrieves all the vectors from Ω that are within box b. As we will see, we can group multiple (exact) k-mer queries into a box query to achieve improved performance.

To efficiently process a box query on a large k-mer set on disk, we utilize and extend the BoND-tree (i.e., an index structure recently introduced by Chen et al. [10] to efficiently support box queries in an NDDS) to implement the physical store Ω_{k_0}. The structure of the BoND-tree is similar to that of the R*-tree, except that the discrete geometric concepts (e.g., discrete rectangle/box) in an NDDS are used. Special heuristics for splitting overflown nodes that utilize the characteristics of an NDDS to achieve improved performance are adopted. For our application, k_0-mers in Ω_{k_0} are saved in the leaf nodes and each k_0-mer is associated with a pointer pointing to a list of read ids that are stored in one or more linked pages.

4 The VA-Store Method

To process a k-mer query on a virtual store, we need to transform it into one or more k_0-mer queries on the physical store. In this section, we present such transformations for both $k < k_0$ and $k > k_0$, derive accuracy estimation models for approximate transformations (i.e., $k > k_0$), and discuss query optimization strategies for the transformed queries.

4.1 Query Transformations Between Virtual and Physical Stores

Assume that we want to perform a k-mer query $q^{(k)} = a_1 a_2 \ldots a_k$ on virtual store Ω_k $(k \neq k_0)$. Since Ω_k does not physically exist, we have to obtain the result of query $q^{(k)}$ by

1. In the rest of the paper, we refer a *k-mer query* to the query searching for a k-mer and refer a *query k-mer* to the k-mer representing a query. When it is not confusing, we will not distinguish a k-mer query and its corresponding k-mer.

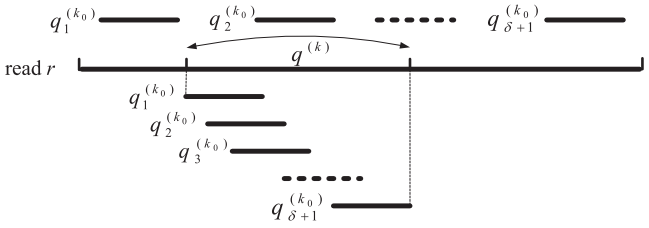


Fig. 3. Alignment scenarios for $k > k_0$.

running one or more transformed k_0-mer queries on physical store Ω_{k_0}.

4.1.1 Transformation for Case $k \leq k_0$

Let us consider $k < k_0$ first. Let $\theta = k_0 - k$. We notice that, if we use a k_0-mer $q^{(k_0)}$ that contains $q^{(k)}$ as a k_0-mer query run on Ω_{k_0}, the result set $R(q^{(k_0)})$ of query $q^{(k_0)}$ on Ω_{k_0} is contained in the result set $R(q^{(k)})$ of query $q^{(k)}$ on Ω_k. This is because, if the id of a read r is in $R(q^{(k_0)})$, it must also be in $R(q^{(k)})$, which is based on the fact that, if $r \sqsupseteq q^{(k_0)}$ is true, $r \sqsupseteq q^{(k)}$ must also be true since $q^{(k_0)} \sqsupseteq q^{(k)}$.

Let

$$Q_{k_0} = \{\ x_1x_2\dots x_sa_1a_2\dots a_kx_{s+1}\dots x_\theta \mid \theta = k_0 - k \\ \wedge\ 0 \leq s \leq \theta\ \wedge\ x_i \in \{A,G,T,C\}\ \wedge\ 1 \leq i \leq \theta\ \}. \quad (2)$$

This is the set of all the k_0-mers that contain query k-mer $q^{(k)} = a_1a_2\dots a_k$. Note that a k_0-mer in Q_{k_0} becomes $a_1a_2\dots a_kx_1\dots x_\theta$ (or $x_1x_2\dots x_\theta a_1a_2\dots a_k$) when $s = 0$ (or θ).

From the previous analysis, we have

$$R(q^{(k)}) \ \supseteq \bigcup_{q^{(k_0)} \in Q_{k_0}} R(q^{(k_0)}). \quad (3)$$

On the other hand, if the id of a read r is in $R(q^{(k)})$, we have $r \sqsupseteq q^{(k)}$. Since the length of r must be at least k_0, there exists $q^{(k_0)} \in Q_{k_0}$ such that $r \sqsupseteq q^{(k_0)}$. Hence, the id of r is also contained in $R(q^{(k_0)})$. Thus, we have

$$R(q^{(k)}) = \bigcup_{q^{(k_0)} \in Q_{k_0}} R(q^{(k_0)}). \quad (4)$$

Eq. (4) indicates that the result of k-mer query $q^{(k)}$ on virtual store Ω_k can be obtained by taking a union of the results of k_0-mer queries using each $q^{(k_0)} \in Q_{k_0}$ on physical store Ω_{k_0}. In other words, the original KDBP$_k$ on Ω_k can be transformed into $|Q_{k_0}| = (\theta+1)4^\theta$ number of KDBP$_{k_0}$'s on Ω_{k_0} if $k < k_0$. Therefore, Eq. (4) is the transformation from the virtual store to the physical store when $k < k_0$.

To improve efficiency, each group of 4^θ related (exact) k_0-mer queries of the form: $x_1x_2\dots x_sa_1a_2\dots a_kx_{s+1}\dots x_\theta$, where $x_i \in \{A,G,T,C\}$, $1 \leq i \leq \theta$ and $0 \leq s \leq \theta$, can be combined into one box query $bq_s^{(k_0)}$ with the following box:

$$\underbrace{X \times X \times \dots \times X}_{s\ sets} \times \{a_1\} \times \{a_2\} \times \dots \times \{a_k\} \times \underbrace{X \times \dots \times X}_{(\theta-s)\ sets},$$

where $X = \{A,G,T,C\}$ and $0 \leq s \leq \theta$. Hence, we only need to execute $(\theta+1)$ box queries on Ω_{k_0} instead of $(\theta+1)4^\theta$ exact queries. In other words, we apply the following formula to implement the transformation given by Eq. (4) for $k < k_0$

$$R(q^{(k)}) = \bigcup_{0 \leq s \leq \theta} R(bq_s^{(k_0)}), \quad (5)$$

which can significantly reduce total query processing latency and increase shared disk accesses. The processing of box queries could be further improved by utilizing an efficient access method for box queries such as the BoND-tree.

For example, if we want to get the result of a 3-mer query $q^{(3)} = AGT$ on virtual store Ω_3 (i.e., $k = 3$), we perform three box queries with the following three boxes: $\{A\} \times \{G\} \times \{T\} \times X \times X$, $X \times \{A\} \times \{G\} \times \{T\} \times X$, and $X \times X \times \{A\} \times \{G\} \times \{T\}$, respectively, on physical store Ω_5 (i.e., $k_0 = 5$) and return the union of the results of these box queries as the result of $q^{(3)}$. If we use Eq. (4) to get the same result, we would have to perform 48 (exact) 3-mer queries.

4.1.2 Transformation for Case $k > k_0$

Now let us consider $k > k_0$. Let $\delta = k - k_0$. In this case, we decompose the given k-mer query $q^{(k)} = a_1a_2\dots a_k$ on virtual store Ω_k into $\delta+1$ shifted k_0-mer queries: $q_1^{(k_0)} = a_1a_2\dots a_{k_0}$, $q_2^{(k_0)} = a_2a_3\dots a_{k_0+1}, \dots\dots, q_{\delta+1}^{(k_0)} = a_{\delta+1}a_{\delta+2}\dots a_k$ on physical store Ω_{k_0}. If the id of read r is in the result $R(q^{(k)})$ of k-mer query $q^{(k)}$, it must be in the result $R(q_i^{(k_0)})$ of each k_0-mer query $q_i^{(k_0)}$ $(1 \leq i \leq \delta+1)$. This is because, if $r \sqsupseteq q^{(k)}$ is true, $r \sqsupseteq q_i^{(k_0)}$ must also be true for every $1 \leq i \leq \delta+1$ since $q^{(k)} \sqsupseteq q_i^{(k_0)}$ (see $q_i^{(k_0)}$'s as shown below read r in Fig. 3). Thus, we have

$$R(q^{(k)}) \ \subseteq \bigcap_{1 \leq i \leq \delta+1} R(q_i^{(k_0)}). \quad (6)$$

On the other hand, if the id of read r is in the result $R(q_i^{(k_0)})$ of $q_i^{(k_0)}$ for every $1 \leq i \leq \delta+1$, the id of r is not necessarily also in the result $R(q^{(k)})$ of $q^{(k)}$. The id of r is in $R(q^{(k)})$ only for the case when $q_i^{(k_0)}$'s are aligned in the shifted fashion as shown below read r in Fig. 3. There are many other cases (e.g., $q_i^{(k_0)}$'s as shown above read r in Fig. 3) in which the id of r is not in $R(q^{(k)})$. In other words, the two sides of Eq. (6) are not equal to each other. However, since the desired result (i.e., the left hand side) is contained in the right hand side, we use the right hand side as an approximate solution/result for given query $q^{(k)}$. Hence, our transformation for $k > k_0$ is given by the following equation:

$$R(q^{(k)}) \doteq \bigcap_{1 \leq i \leq \delta+1} R(q_i^{(k_0)}). \quad (7)$$

Since the approximate result/solution from (7) contains no false negatives, it is more useful than general approximate solutions sought by typical genome sequence analysis applications. In fact, the right hand side of Eq. (7) represents an optimal approximate solution for $q^{(k)}$ when using k_0-mer queries on Ω_{k_0}. This is because missing any $q_i^{k_0}$ might cause the approximate solution to include more unqualified read ids, resulting in a larger size and a lower accuracy.

Let us consider an example of applying Eq. (7), which also illustrates the two cases of Fig. 3. Assume that we want to get the result of a 6-mer query $q^{(6)} = GATACT$ on virtual store Ω_6 (i.e., $k = 6$). We perform the following two 5-mer queries: $q_1^{(5)} = GATAC$ and $q_2^{(5)} = ATACT$ on physical store Ω_5 (i.e., $k_0 = 5$) and return the intersection of the results of these two 5-mer queries as the approximate result of the original 6-mer query $q^{(6)}$. Let us first consider read $r_1 = ATGT\mathbf{GATACT}GGTA$. r_1 will be included in the result of query $q^{(6)}$ because it contains both the 5-mers of $q_1^{(5)}$

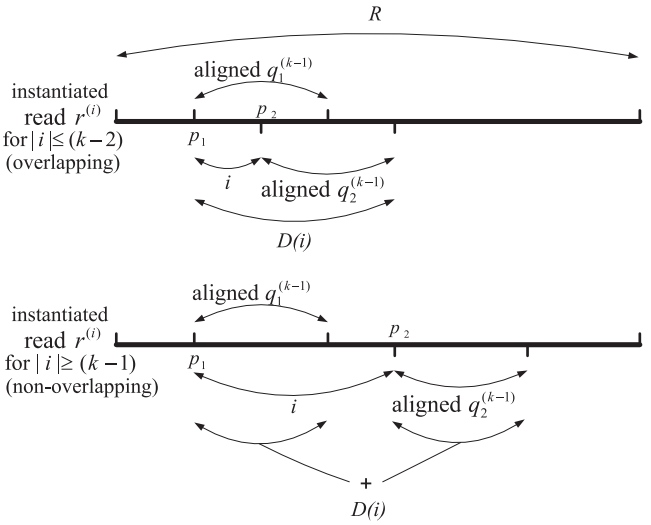


Fig. 4. Accuracy estimation parameters.

and $q_2^{(5)}$ (see the bold-faced and underlined subsequence in r_1). r_1 represents a true positive since it indeed contains the 6-mer of $q^{(6)}$ (an example of the lower shifted overlapping case in Fig. 3). Now let us consider another read $r_2 = GA\mathbf{GATAC}A\mathbf{ATACT}T$. r_2 will also be included in the result of query $q^{(6)}$ because it contains both the 5-mers of $q_1^{(5)}$ and $q_2^{(5)}$ (see the bold-faced and underlined subsequence in r_2). However, r_2 represents a false positive since it does not contain the 6-mer of $q^{(6)}$ (an example of the upper non-overlapping case in Fig. 3).

4.2 Query Accuracy Estimation

As mentioned in Section 4.1, when $k > k_0$ (≥ 1), an approximate solution to problem KDBP$_k$ on virtual store Ω_k is usually obtained when physical store Ω_{k_0} is used to solve the problem. In this case, it is desired to estimate the accuracy of such an approximate solution. In this section, we discuss how to estimate the accuracy. For simplicity, we assume that the reads in a given input genomic dataset are of the same length in the following discussion.

4.2.1 Accuracy Estimation Model for Case $k = k_0 + 1$

Let us first consider the case when $k = k_0 + 1$. Let $q^{(k)}$ be the target query k-mer on virtual store Ω_k, and $q_1^{(k-1)}$, $q_2^{(k-1)}$ be the two shifted $(k-1)$-mers (i.e., k_0-mers) of $q^{(k)}$. Let $r^{(i)}$ be a read for which there is a way to align $q_1^{(k-1)}$ and $q_2^{(k-1)}$ with $r^{(i)}$ and the distance between $q_1^{(k-1)}$ and $q_2^{(k-1)}$ in $r^{(i)}$ is $i = p_2 - p_1$, where p_1 and p_2 are the positions for the first letters of $q_1^{(k-1)}$ and $q_2^{(k-1)}$ in $r^{(i)}$, respectively. We call $r^{(i)}$ an instantiated read with distance i for $q_1^{(k-1)}$ and $q_2^{(k-1)}$. Let $D(i)$ be the number of distinct positions in read $r^{(i)}$ that are covered by $q_1^{(k-1)}$ and/or $q_2^{(k-1)}$ when $q_1^{(k-1)}$ and $q_2^{(k-1)}$ are aligned/instantiated with $r^{(i)}$. Let $T(i)$ be the number of such possible reads $r^{(i)}$. Let R ($\ge k$) be the length of each read in an input genomic dataset. Fig. 4 illustrates these parameters.

In general, for $|i| \le R - (k-1)$, we have

$$D(i) = \begin{cases} k - 1 + |i| & \text{if } |i| \le (k-2), \\ 2(k-1) & \text{if } |i| \ge (k-1). \end{cases} \tag{8}$$

Since the result of query $q^{(k)}$ on virtual store Ω_k consists of all the instantiated reads with distance 1 (i.e., $r^{(1)}$; see the case as shown below read r in Fig. 3 for $\delta = 1$) and $T(1)$ is the number of such reads, the (average) accuracy rate $\xi(k, k_0)$ of the approximate solution to KDBP$_k$ using physical store Ω_{k_0} for $k = k_0 + 1$ can be estimated as

$$\xi(k, k_0) = T(1)/T_{all}, \tag{9}$$

where

$$T_{all} = \sum_{i=-(R-(k-1))}^{R-(k-1)} T(i) = T_{separate} + T_{overlap}, \tag{10}$$

$$T_{separate} = \sum_{i=-(R-(k-1))}^{-(k-1)} T(i) + \sum_{i=(k-1)}^{R-(k-1)} T(i), \tag{11}$$

$$T_{overlap} = \sum_{i=-(k-2)}^{-1} T(i) + T(0) + \sum_{i=(k-2)}^{1} T(i). \tag{12}$$

Here $T_{separate}$ and $T_{overlap}$ are the numbers of instantiated reads with $q_1^{(k-1)}$ and $q_2^{(k-1)}$ being non-overlapped ($|i| \ge k-1$) and overlapped ($|i| \le k-2$), respectively. Our goal is to derive the estimation formulas for $T_{separate}$ and $T_{overlap}$ so that the accuracy (rate) can be estimated using Eq. (9).

To estimate $T_{separate}$ for $|i| \ge k-1$ (see Fig. 4) , we have

$$\begin{aligned} T(i) &\doteq (R - (k - 1 + |i|) + 1)4^{R-D(i)} \\ &= (R - k - |i| + 2)4^{R-D(i)}, \end{aligned} \tag{13}$$

since there are $(R - D(i))$ free positions, each of which can take any of four letters in $\Delta = \{A, G, T, C\}$, for a possible read $r^{(i)}$ that can be instantiated by $q_1^{(k-1)}$ and $q_2^{(k-1)}$ with distance i, and there are $(R - (k - 1 + |i|) + 1)$ ways to place $q_1^{(k-1)}$ and $q_2^{(k-1)}$ with distance i in $r^{(i)}$.

From (8), (11) and (13), we have

$$\begin{aligned} T_{separate} &\doteq 2 \sum_{(k-1)}^{R-(k-1)} (R - k - i + 2)4^{R-2(k-1)} \\ &= (R - 2k - 1)(R - 2k + 4)4^{R-2(k-1)}. \end{aligned} \tag{14}$$

To estimate $T_{overlap}$ for $|i| \le k-2$ (see Fig. 4), we notice that $T(i)$ may be 0 for many i's within the range. This is because $q_1^{(k-1)}$ and $q_2^{(k-1)}$ are taken from the same given query k-mer $q^{(k)}$. Assume $q^{(k)} = a_1a_2a_3 \dots a_{k-1}a_k$, then $q_1^{(k-1)} = a_1a_2a_3 \dots a_{k-1}$ and $q_2^{(k-1)} = a_2a_3 \dots a_{k-1}a_k$. The natural distance between $q_1^{(k-1)}$ and $q_2^{(k-1)}$ is $i = 1$. Unless $q^{(k)}$ contains a special periodic pattern, it is usually impossible to align/instantiate $q_1^{(k-1)}$ and $q_2^{(k-1)}$ with a read $r^{(i)}$ for distance $i \ne 1$ ($|i| \le k-2$).

For example, to have a feasible instantiation for $i = 2$, we must have $a_2 = a_3 = a_4 = \dots = a_{k-1} = b_1$, i.e., $q^{(k)} = a_1b_1b_1 \dots b_1a_k$. In this case, we say that $q^{(k)}$ has a period length 1. To have a feasible instantiation for $i = 3$, we must have $a_2 = a_4 = b_1$, $a_3 = a_5 = b_2$, $a_4 = a_6 = b_1$, $a_5 = a_7 = b_2, \dots$, $a_{k-1} = b_1$ (or b_2), i.e., $q^{(k)} = a_1b_1b_2b_1b_2 \dots (b_1|b_2)a_k$, where $(b_1|b_2)$ indicates either b_1 or b_2. In this case, we say that $q^{(k)}$ has a period length 2. In general, to have a feasible instantiation for $i = c + 1$, we must have $a_2 = a_{c+2} = b_1$, $a_3 = a_{c+3} = b_2, \dots, a_{c+1} = a_{2c+1} = b_c, a_{c+2} = a_{2c+2} = b_1 \dots$, $a_{k-1} = (b_1|b_2|\cdots|b_c)$, i.e., $q^{(k)} = a_1(b_1b_2 \dots b_c)^m b_1b_2 \dots b_t a_k$, where $(b_1b_2 \dots b_c)^m$ indicates that period $b_1b_1 \dots b_c$ is repeated

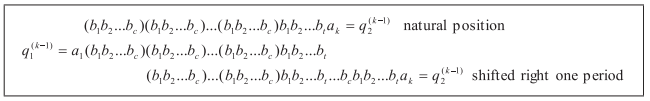


Fig. 5. Alignable positions for $q_1^{(k-1)}$ and $q_2^{(k-1)}$ in Scenario 1.

m (≥ 2) times, $m = \lfloor (k-2)/c \rfloor$, and $t = (k-2)\ mod\ c$. In this case, we say that $q^{(k)}$ has a period length c. Note that $b_1b_2 \ldots b_t$ is empty if $t = 0$.

We have three observations. First, periodic patterns can be found in real genomic datasets. Hence, estimation formulas need to be developed to handle the situations in which a periodic pattern is contained in a query k-mer. Second, $q^{(k)}$ may contain nested periods of multiple lengths. For example, a period "AG" of length 2 is nested in a period "$AGAG$" of length 4. To avoid double counting, we consider the minimum period only for a given query k-mer if nested periods exist. Third, $q_1^{(k-1)}$ and $q_2^{(k-1)}$ of a periodic $q^{(k)}$ may be aligned with each other multiple times by shifting $q_2^{(k-1)}$ one or more periods to the left or to the right with respect to $q_1^{(k-1)}$. It is noted that a right shifting is always feasible. However, it may not be true for the left shifting. Different shiftabilities have different effects on the accuracy estimation formulas. We consider various scenarios as follows.

Scenario 1. Right shiftable only and having a stop letter at the right end, i.e., $q^{(k)} = a_1(b_1b_2 \ldots b_c)^m b_1b_2 \ldots b_ta_k$, where $c \geq 1, c > t, m \geq 2$, and stop letter $a_k \neq b_{t+1}$.

Since there is a stop letter a_k at the right end of $q^{(k)}$, $q_2^{(k-1)}$ is generally not alignable with $q_1^{(k-1)}$ when it is shifted to the left for any distance $i \in [-(k-2), 0]$ from its natural position with distance i = 1 (see Fig. 5). Hence, we have $T(i) = 0$ for $i \in [-(k-2), 0]$. On the other hand, if $q_2^{(k-1)}$ is shifted to the right for a distance that is not a multiple of period length c, i.e., $i > 1$ and $i \neq 1 + j * c$ ($1 \leq j \leq m$), it is usually not alignable with $q_1^{(k-1)}$. Thus, we also have $T(i) = 0$ for such i's. $T(i) \neq 0$ only if $q_2^{(k-1)}$ is shifted to the right for a distance $j * c$ from its natural position with $i = 1$. Since stop letter a_k is at the right end of $q_2^{(k-1)}$, instantiated reads $r^{(1+j_1*c)}$ and $r^{(1+j_2*c)}$ are different if $j_1 \neq j_2$. Therefore, we have

$$T_{overlap} \doteq T(1) + \sum_{j=1}^{m} T(1 + j * c) = \sum_{j=0}^{m} T(1 + j * c). \quad (15)$$

Note that estimation formula (13) can still be used for $i = 1 + j * c$ ($0 \leq j \leq m$). From (8), (9), (10), (13), (14) and (15), the query accuracy can be estimated as follows:

$$\begin{aligned}\xi(k, k_0) &\doteq (R-k+1)4^{R-k}/ \\ &\quad \Bigg[(R-2k-1)(R-2k+4)4^{R-2(k-1)} \\ &\quad + \sum_{j=0}^{m}(R-k-j*c+1)4^{R-(k+j*c)}\Bigg] \\ &\doteq (R-k+1)(1-4^{-c})/ \\ &\quad [(R-2k-1)(R-2k+4)4^{-k+2}(1-4^{-c}) \\ &\quad + (R-k+1)(1-4^{-k+2-c}) - \\ &\quad c * 4^{-c}(1-4^{-k+2})/(1-4^{-c}) \\ &\quad + (k-2) * c * 4^{-k+2-c}].\end{aligned} \quad (16)$$

Scenario 2. Right shiftable only and having a stop letter at the left end, i.e., $q^{(k)} = a_1(b_1b_2 \ldots b_c)^m b_1b_2 \ldots b_ta_k$, where $c \geq 1, c > t, m \geq 2$, $a_k = b_{t+1}$ and stop letter $a_1 \neq b_c$.

Since there is a stop letter a_1 at the left end of $q^{(k)}$, $q_2^{(k-1)}$ is generally not alignable with $q_1^{(k-1)}$ when it is shifted to the left for any distance $i \in [-(k-2), 0]$ from its natural position with distance i = 1. Hence, we have $T(i) = 0$ for $i \in [-(k-2), 0]$. On the other hand, if $q_2^{(k-1)}$ is shifted to the right for a distance that is not a multiple of period length c, i.e., $i > 1$ and $i \neq 1 + j * c$ ($1 \leq j \leq m$), it is usually not alignable with $q_1^{(k-1)}$. Thus, we also have $T(i) = 0$ for such i's. The difference between this and Scenario 1 is that the letter at the right end of $q_2^{(k-1)}$ follows the periodic pattern. In other words, there is no stop at the right end. Although it is also true that $T(i) \neq 0$ if $q_2^{(k-1)}$ is shifted to the right for a distance $j * c$ from its natural position with $i = 1$, any instantiated read $r^{(1+j*c)}$ ($1 \leq j \leq m$) is a special case of $r^{(1)}$. Thus, $T(1)$ has already included $T(1 + j * c)$ for $1 \leq j \leq m$. Therefore, we have $T_{overlap} \doteq T(1)$. From (8), (9), (10), (13) and (14), the query accuracy can be estimated as follows:

$$\begin{aligned}\xi(k, k_0) &\doteq (R-k+1)4^{R-k}/ \\ &\quad [(R-2k-1)(R-2k+4)4^{R-2(k-1)} \\ &\quad + (R-k+1)4^{R-k}] \\ &= (R-k+1)/ \\ &\quad [(R-2k-1)(R-2k+4)4^{-k+2} \\ &\quad + (R-k+1)].\end{aligned} \quad (17)$$

Scenario 3. Left shiftable[2] with $c > 1$, i.e., $q^{(k)} = a_1(b_1b_2 \ldots b_c)^m b_1b_2 \ldots b_ta_k$, where $c > t, m \geq 2$, $a_k = b_{t+1}$ and $a_1 = b_c$.

In this case, $q_2^{(k-1)}$ is shiftable to both the left and the right for any distance that is a multiple of period length c from its natural position with $i = 1$ (see Fig. 6). The analysis for the right shifting situation is the same as the one we had above for Scenario 2. In other words, $T(i) = 0$ for $i > 1$ and $i \neq 1 + j * c$ ($1 \leq j \leq m$); and $T(i)$ is contained in $T(1)$ for $i = 1 + j * c$. For the left shifting, we notice that $q^{(k)} = b_c(b_1b_2 \ldots b_c)^m b_1b_2 \ldots b_tb_{t+1} = (b_cb_1b_2 \ldots b_{c-1})^m b_cb_1 \ldots b_tb_{t+1}$. From a similar reason for the right shifting situation, we have $T(i) = 0$ for $i < 1$ and $i \neq 1 + j * c$ ($-m \leq j \leq -1$). When $i = 1 + j * c$ ($-m \leq j \leq -1$), from a similar analysis for the right shifting, we find that $T(1 + j * c)$ is contained in $T(1 - c)$ for $-m \leq j \leq -2$. Therefore, we have $T_{overlap} \doteq T(1 - c) + T(1)$. From (8), (9), (10), (13), and (14), the query accuracy can be estimated as follows:

$$\begin{aligned}\xi(k, k_0) &\doteq (R-k+1)4^{R-k}/ \\ &\quad [(R-2k-1)(R-2k+4)4^{R-2(k-1)} \\ &\quad + (R-k-c+3)4^{R-(k+c-2)} + (R-k+1)4^{R-k}] \\ &= (R-k+1)/ \\ &\quad [(R-2k-1)(R-2k+4)4^{-k+2} \\ &\quad + (R-k-c+3)4^{-c+2} + (R-k+1)].\end{aligned} \quad (18)$$

2. Note that the right shiftable is always true.

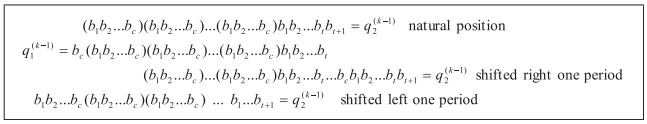


Fig. 6. Alignable positions for $q_1^{(k-1)}$ and $q_2^{(k-1)}$ in Scenario 3.

Scenario 4. Left shiftable with $c = 1$, i.e., $q^{(k)} = a_1b_1b_1 \dots b_1a_k$, where $m \geq 2$ and $a_k = a_1 = b_1$.

In this case, $q_2^{(k-1)}$ is also shiftable to both the left and the right. But the shifting can be done for any distance $|i| \leq k-2$. However, we notice that each $T(i)$ is included in $T(0)$ for $|i| \leq k-2$. Thus, $T_{overlap} = T(0)$ in this case. Therefore, from (8), (9), (10), (13) and (14), the query accuracy can be estimated as follows:

$$\begin{aligned}\xi(k,k_0) &\doteq (R-k+1)4^{R-k}/\\ &\quad [(R-2k-1)(R-2k+4)4^{R-2(k-1)}\\ &\quad + (R-k+2)4^{R-k+1}]\\ &= (R-k+1)/[(R-2k-1)(R-2k+4)4^{-k+2}\\ &\quad + 4(R-k+2)].\end{aligned} \tag{19}$$

Scenario 5. Other situations that do not belong to the above Scenarios 1, 2, 3, and 4.

In this case, $q_2^{(k-1)}$ generally cannot be shifted to the left or the right for any distance other than $i = 1$. In other words, $T(i) = 0$ for $i \neq 1$ and $|i| \leq k-2$. Thus, $T_{overlap} = T(1)$ in this case. From (8), (9), (10), (13) and (14), the query accuracy can be estimated as follows:

$$\begin{aligned}\xi(k,k_0) &\doteq (R-k+1)4^{R-k}/\\ &\quad [(R-2k-1)(R-2k+4)4^{R-2(k-1)}\\ &\quad + (R-k+1)4^{R-k}]\\ &= (R-k+1)/[(R-2k-1)(R-2k+4)4^{-k+2}\\ &\quad + (R-k+1)].\end{aligned} \tag{20}$$

Note that the above estimation derivations are based on an assumption that the letters for each position in reads are uniformly distributed and different positions in a read are independent. Our results can be summarized as follows.

Theorem 4.1. *Assume that the letters for each position in reads from a given dataset are uniformly distributed and the positions of a read are independent. The shifted k_0-mers for the input reads are stored in physical store Ω_{k_0}. If $k = k_0 + 1$, the (average) accuracy rate $\xi(k, k_0)$ of a k-mer query on a virtual store Ω_k (i.e., solving $KDBP_k$) when using Ω_{k_0} can be estimated by Eqs. (16), (17), (18), (19) and (20) for Scenarios 1, 2, 3, 4 and 5, respectively.*

4.2.2 Extension to General Case

Using the above analytical approach to deriving an accuracy estimation model for the general case when $k = k_0 + \delta$ with $\delta \geq 1$ is difficult. Following Eq. (7), a given k-mer query $q^{(k)}$ on virtual store Ω_k is transformed into $(\delta+1)$ k_0-mer queries on physical store Ω_{k_0}. When $\delta > 1$, there are too many cases to consider on how these $(\delta+1)$ k_0-mer queries are overlapped and aligned/instantiated with a given read r. Hence, we use a regression approach to deriving an accuracy estimation model based on sampling, rather than applying an analytical approach, for the general case.

There are two forces that affect the accuracy of the approximate solution for $q^{(k)}$ from Eq. (7). One force is the similarity between $q^{(k)}$ and its transformed k_0-mer queries. When δ increases, the degree of such similarity decreases. Using such transformed queries to represent the original query has a negative impact on the accuracy of the result. The other force is the uniqueness of instantiating the transformed query k_0-mers with a given read r. When δ increases, the chance for all $(\delta+1)$ transformed k_0-mers to be instantiated with r in other ways besides the desired $q^{(k)}$ (i.e., the natural shifted instantiation shown as the case below r in Fig. 3) becomes smaller, which has a positive impact on the accuracy of the result. Fig. 13 shows a typical pattern for the accuracy of an approximate solution observed from experiments. Initially, the first force is dominant, which makes the accuracy decrease. Later on, the second force becomes gradually dominant, which makes the accuracy increase. As a result, a "V" shape pattern is observed in the initial segment of an accuracy curve. Since an approximate solution usually has a high accuracy when δ is large, whose estimation is unimportant, the challenge is how to derive a regression estimation model to capture the accuracy for the initial "V" shape pattern. Clearly, the normal linear or polynomial regression models do not work well here.

Fortunately, the analytical approach discussed in Section 4.2.1 provides a way to find a good formulation for regression in our situation. Let us consider the two extreme cases (i.e., the lower and upper ones) shown in Fig. 3 for the $(\delta+1)$ k_0-mers being instantiated/aligned with read r. The lower instantiation yields the correct result for k-mer query $q^{(k)}$, while the upper instantiation represents an incorrect result. Let R be the length of read r, and assume $R \geq (\delta+1) * k_0$ (i.e., all the $(\delta+1)$ k_0-mers can be instantiated with r in a non-overlapping way). Note that any position in r that is not covered by the instantiation can take any of the four letters in $\Delta = \{A, G, T, C\}$.

Let T_{low} and T_{up} be the numbers of reads instantiated as the lower and upper cases shown in Fig. 3, respectively. If there were only the lower and upper instantiations, similar to Eq. (9), the accuracy can be estimated as

$$\begin{aligned}\xi(k,k_0) &\doteq T_{low}/[T_{low}+T_{up}]\\ &= (R-k+1)*4^{R-k} \Big/ \Bigg[(R-k+1)*4^{R-k}\\ &\quad + \binom{R-(\delta+1)*k_0+(\delta+1)}{\delta+1} * (\delta+1)! * 4^{R-(\delta+1)k_0}\Bigg]\\ &= 1/[1+H_{k_0}(\delta)*4^{-(k_0-1)\delta}],\end{aligned} \tag{21}$$

where $H_{k_0}(\delta) = [\prod_{i=1}^{\delta+1}(R-(\delta+1)*k_0+i)]/(R-(k_0+\delta)+1)$. Note that $H_{k_0}(\delta)$ in the above equation increases as δ increases, while term $4^{-(k_0-1)\delta}$ decreases as δ increases. The former is dominant initially, while the latter is dominant later on. As a result, a "V" shape pattern can be observed. However, there are many other cases besides the upper instantiation case shown in Fig. 3 that can lead to incorrect results. For example, some of the $(\delta+1)$ k_0-mers may be partially overlapped. Hence, more terms that are similar to the second term of the denominator in Eq. (21) may be

included in the denominator. To capture such effects, we adopt the following regression model:

$$f_{k_0}(\delta) = 1/[1 + a * H_{k_0}(\delta) * 4^{-b*(k_0-1)*\delta}]. \quad (22)$$

We view δ as the explanatory variable, k_0 as a parameter, and a and b are the regression coefficients.

We employ the least squares method to determine the optimal a and b coefficients as follows. We first perform a transformation on Eq. (22) to get

$$\begin{aligned} g_{k_0}(\delta) &= ln(1/f_{k_0}(\delta) - 1) = ln(a * H_{k_0}(\delta) * 4^{-b*(k_0-1)*\delta}) \\ &= ln(a) + ln(H_{k_0}(\delta)) - b * (k_0 - 1) * \delta * ln(4). \end{aligned} \quad (23)$$

Let $S = \sum_{i=0}^{n} [g_{k_0}(\delta_i) - g_i]^2$, where $< \delta_i, g_i >$ $(i = 1, 2, \ldots, n)$ are n sample observation pairs. Solving equations

$$\begin{aligned} \frac{\partial S}{\partial a} &= \sum_{i=1}^{n} 2 * [ln(a) + ln(H_{k_0}(\delta_i)) \\ &\quad - b * (k_0 - 1) * \delta_i * ln(4) - g_i] * \frac{1}{a} = 0, \\ \frac{\partial S}{\partial b} &= \sum_{i=1}^{n} -2 * [ln(a) + ln(H_{k_0}(\delta_i)) \\ &\quad - b * (k_0 - 1) * \delta_i * ln(4) - g_i] * (k_0 - 1) * \delta_i * ln(4) = 0, \end{aligned}$$

we get

$$\begin{aligned} ln(a) = &\left[\sum_{i=1}^{n} g_i * \sum_{i=1}^{n} \delta_i^2 - \sum_{i=1}^{n} ln(H_{k_0}(\delta_i)) * \sum_{i=1}^{n} \delta_i^2 \right. \\ &\left. - \sum_{i=1}^{n} \delta_i * \sum_{i=1}^{n} g_i \delta_i + \sum_{i=1}^{n} \delta_i * \sum_{i=1}^{n} \delta_i * ln(H_{k_0}(\delta_i)) \right] \\ &\Bigg/ \left[n * \sum_{i=1}^{n} \delta_i^2 - \left(\sum_{i=1}^{n} \delta_i \right)^2 \right], \end{aligned} \quad (24)$$

$$\begin{aligned} b = &\left[\sum_{i=1}^{n} g_i \sum_{i=1}^{n} \delta_i - \sum_{i=1}^{n} ln(H_{k_0}(\delta_i)) \sum_{i=1}^{n} \delta_i \right. \\ &\left. + n * \sum_{i=1}^{n} \delta_i * ln(H_{k_0}(\delta_i)) + n * \sum_{i=1}^{n} g_i \delta_i \right] \\ &\Bigg/ \left[(k_0 - 1) * ln(4) * \left[n * \sum_{i=1}^{n} \delta_i^2 - \left(\sum_{i=1}^{n} \delta_i \right)^2 \right] \right]. \end{aligned} \quad (25)$$

Theorem 4.2. *Assume that the k_0-mers for the input reads are stored in physical store Ω_{k_0}. Let $k = k_0 + \delta$ $(\delta \geq 1)$. The (average) accuracy $\xi(k, k_0)$ of a k-mer query $q^{(k)}$ on a virtual store Ω_k (i.e., solving KDBP$_k$) when using Ω_{k_0} can be estimated by Eq. (22) with $\xi(k, k_0) = f_{k_0}(k - k_0)$, where coefficients a and b can be determined by Eqs. (24) and (25).*

4.3 Accuracy-Guided Adaptive Query Processing

As discussed in Section 4.1, when $k < k_0$, we can improve query processing efficiency by running fewer box queries as specified in Eq. (5) instead of running many exact queries from the direct transformation given in Eq. (4). However, when $k > k_0$, query optimization for the transformation given in Eq. (7) is more complicated. In this section, we present an accuracy-guided adaptive query processing algorithm to improve efficiency for computing an approximate solution for a given k-mer query $q^{(k)}$ when $k > k_0$.

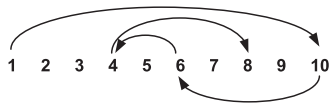


Fig. 7. Maximum distance principle.

Following the transformation given in Eq. (7), to compute an approximate solution to $q^{(k)}$ on virtual store Ω_k, we need to execute $\delta + 1$ $(\delta = k - k_0)$ shifted k_0-mer queries $q_1^{(k_0)}$, $q_2^{(k_0)}, \ldots, q_{\delta+1}^{(k_0)}$ (see the lower case as shown below read r in Fig. 3) on physical store Ω_{k_0} and use the intersection of their results as an approximate solution to $q^{(k)}$. Since these $(\delta + 1)$ k_0-mer queries are largely overlapped, it is possible that using only a subset of these queries is sufficient to produce an approximate solution with a similar accuracy. In such a case, the computing efficiency is improved since only a subset of the queries are actually executed.

Two questions need to be answered to develop such an optimization technique. First, which subset of queries is to be executed and in what order? Second, what is the stop condition to terminate the query processing?

To answer the first question, we realize that it is desirable to choose as diverse queries as possible because we should use a small number of queries to capture as many scenarios as possible. We notice that the distance between the indexes of two queries represents their similarity. For example, queries $q_3^{(k_0)}$ and $q_5^{(k_0)}$ are more similar than queries $q_1^{(k_0)}$ and $q_6^{(k_0)}$ since the distance between the first pair $(5 - 3 = 2)$ is smaller than the distance between the second pair $(6 - 1 = 5)$. Therefore, we adopt a so-called "maximum distance principle" to choose the next query for execution, namely, choosing the next query whose minimum distance to all the previously chosen queries is the largest. For example, given 10 k_0-mer queries $q_1^{(k_0)}$, $q_2^{(k_0)}, \ldots, q_{10}^{(k_0)}$, if we first choose $q_1^{(k_0)}$, the next chosen query should be $q_{10}^{(k_0)}$ since it has the largest (minimum) distance to $q_1^{(k_0)}$ among the unchosen queries. We then choose $q_6^{(k_0)}$ (or $q_5^{(k_0)}$) since its minimum distance to $q_1^{(k_0)}$ and $q_{10}^{(k_0)}$ is 4, which is the largest among the unchosen queries. The next query can be $q_4^{(k_0)}$ (or $q_3^{(k_0)}$ or $q_8^{(k_0)}$). This process continues like the binary search (see Fig. 7).

As for the second question, it is noticed that the accuracy of the approximate solution is monotonically increasing as more queries are executed. This is because the result (solution) of the target k-mer query is contained in the intersection of the results of all the $(\delta + 1)$ shifted k_0-mer queries (see Eq. (6)). One possible stop condition is to end the query processing if the difference between the accuracies of two consecutive approximate solutions is small. However, checking the accuracy of an approximate solution usually involves high overhead (e.g., performing expensive sequence alignments). Hence, we adopt the following easy-checking stop condition, namely, the query processing ends if the percentage of the size reduction for the approximate solution is small, i.e., $(s' - s'')/s' \leq \epsilon$ where s'' and s' are the sizes of two consecutive approximate solutions, respectively, and ϵ is a given tolerance. Every time when a new k_0-mer query is executed, its result is used to intersects with the preceding approximate result, which usually removes more unqualified reads from the approximate solution, resulting in a reduced size approximate solution with an improved accuracy. Therefore, this

stop condition is similar to the first stop condition except that it is more efficient to carry out.

Algorithm 1. Accuracy-Guided Adaptive Query Processing

Input: (1) physical store Ω_{k_0}; (2) $\delta+1$ shifted k_0-mer queries $q_1^{(k_0)}, q_2^{(k_0)}, \ldots, q_{\delta+1}^{(k_0)}$ ($\delta \geq 1$) for a given k-mer query $q^{(k)}$; (3) stop tolerance ϵ.

Output: An approximate solution $R(q^{(k)})$ to given $q^{(k)}$ and the set QS of executed k_0-mer queries.

1 initialize $QS = \emptyset$, $queue$ = $\emptyset$;
2 execute $q_1^{(k_0)}$ and $q_{\delta+1}^{(k_0)}$ on Ω_{k_0} to get results $R(q_1^{(k_0)})$ and $R(q_{\delta+1}^{(k_0)})$;
3 let $R(q^{(k)}) = R(q_1^{(k_0)})$;
4 let s' = size of $R(q^{(k)})$;
5 let $R(q^{(k)}) = R(q^{(k)}) \cap (R(q_{\delta+1}^{(k_0)})$;
6 let s'' = size of $R(q^{(k)})$;
7 let $QS = \{q_1^{(k_0)}, q_{\delta+1}^{(k_0)}\}$;
8 **if** δ == 1 **or** $(s'-s'')/s' \leq \epsilon$ **then**
9 return $R(q^{(k)})$ and QS;
10 **else**
11 let [$node$.min, $node$.max] = [2, δ];
12 $queue$.append($node$);
13 **while** $queue$ is not empty **and** $(s'-s'')/s' > \epsilon$ **do**
14 $node$ = $queue$.pop();
15 let $m = \lceil$ ($node$.min + $node$.max)/ 2 $\rceil$;
16 execute $q_m^{(k_0)}$ on Ω_{k_0} to get its result $R(q_m^{(k_0)})$;
17 let $R(q^{(k)}) = R(q^{(k)}) \cap (R(q_m^{(k_0)})$;
18 let $s' = s''$;
19 let s'' = size of $R(q^{(k)})$;
20 let $QS = QS \cup \{q_m^{(k_0)}\}$;
21 **if** $node$.min $< m$ **then**
22 let [$node'$.min, $node'$.max]=[$node$.min, $m-1$];
23 $queue$.append($node'$);
24 **end if**
25 **if** $m <$ $node$.max **then**
26 let [$node'$.min, $node'$.max]=[m+1, $node$.max];
27 $queue$.append($node'$);
28 **end if**
29 **end while**
30 return $R(q^{(k)})$ and QS;
31 **end if**

The above ideas are incorporated into Algorithm 1, which utilizes a binary tree with a breadth-first traversal to realize the maximum distance principle for choosing the next k_0-mer query for execution. From the description, we can see that the algorithm executes at least two k_0-mer queries, i.e., $q_1^{(k_0)}$ and $q_{\delta+1}^{(k_0)}$ (steps 1-9). If the stop condition is not met and there are more un-executed k_0-mer queries, the algorithm repeatedly selects a new k_0-mer query for execution based on the maximum distance principle until the stop condition is met or there is no more un-executed k_0-mer query left (steps 10-31). Each node of the binary search tree that realizes the maximum distance principle contains a range of indexes for un-executed k_0-mer queries (steps 11, 22 and 26). A queue is used to implement the breadth-first traversal of the binary search tree (steps 12, 14, 23, and 27). When the leading node is popped from the queue, the query having the middle index from the range associated with the node is chosen for execution (step 16). The remaining left and right un-executed subranges (if any) are associated with the left and right child nodes, respectively, which are then appended to the end of the queue (steps 21-28). The process may end in the middle without generating the whole tree when the stop condition is satisfied. It is also noted that the algorithm returns $R(q^{(k)}) = \emptyset$ if s' or s'' is found to be 0 at steps 4, 6 and 19. Such an error checking code is not included in the algorithm description for conciseness.

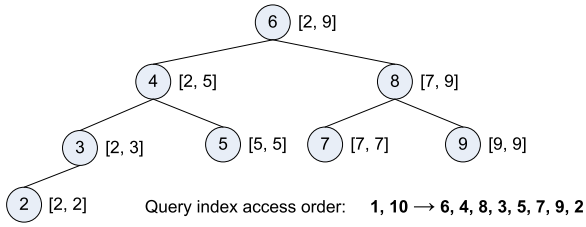


Fig. 8. Query index access order determination.

Fig. 8 shows an example of the binary search tree for a list of 10 k_0-mer queries $q_1^{(k_0)}, q_2^{(k_0)}, \ldots, q_{10}^{(k_0)}$. Indexes 1 and 10 are not included in the index range associated with the root node since $q_1^{(k_0)}$ and $q_{10}^{(k_0)}$ are executed at the beginning of Algorithm 1 and the tree is used to select k_0-mer queries with indexes ranging from 2 to 9. The number inside each node is the middle index from the associated index range, which is the index of the k_0-mer query selected for execution when the node is popped from the queue. The nodes (representing their relevant k_0-mer queries) are visited in a breadth-first fashion, which follows the maximum distance principle. The traversal may end in the middle if the stop condition is met before all the queries are considered.

Let n be the number of given shifted k_0-mer queries. In the worst case, Algorithm 1 still needs to execute all n k_0-mer queries and store $(n-1)/2$ nodes in the queue. In other words, the worst-case complexity of Algorithm 1 is $O(n)$ for both time and space. However, in practice, the algorithm typically only needs to execute a subset (e.g., as low as 20 percent) of given k_0-mer queries to achieve a high accuracy, as shown in our experiments in Section 5.

5 Experiments

To evaluate the performance of our VA-Store approach, we conducted extensive experiments. The experiment programs were implemented in C++ and Python. The physical store in the VA-Store was implemented using the BoND-tree outlined in Section 3. All the experiments were conducted on a Dell PC with a 3.2 GHz Intel Core i7-4790 CPU, 12 GB RAM, 5 TB Hard Drive, and Linux 3.16.0 OS. One synthetic random read dataset RAND with 308,846 reads and two real genome read datasets SR and BJ with 635,036 reads and 479,105 reads, respectively, were used in the experiments. SR and BJ were extracted from Streptomyces rapamycinicus genome NRRL 5491 CP006567.1[3] and Bradyrhizobium japonicum genome NZ_CP007569,[4] respectively. The length of each read was 100 bases (letters). Unless stated otherwise, the reads in the experiments were obtained by cutting up the assembled genome sequences. Various parameter values (e.g., k and δ) were considered in the experiments.

3. http://www.ncbi.nlm.nih.gov/nuccore/CP006567.1/.
4. http://www.ncbi.nlm.nih.gov/nuccore/627779227/.

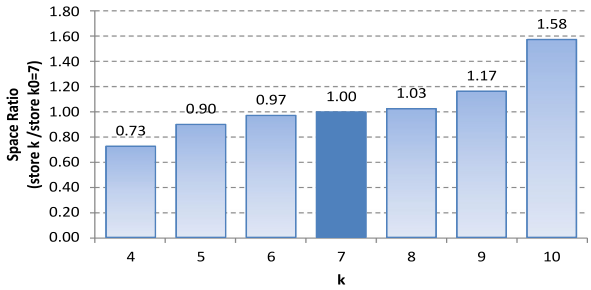


Fig. 9. Space requirements for stores of k-mers of various lengths for dataset SR.

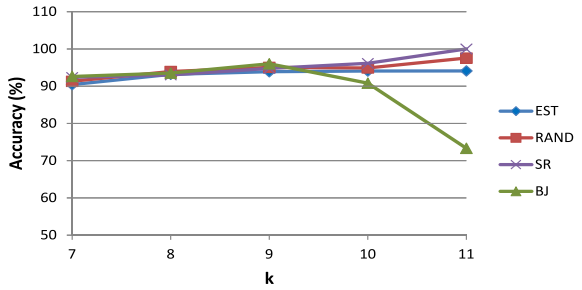


Fig. 11. Estimated and observed accuracies for Scenario 1 with period length c = 2.

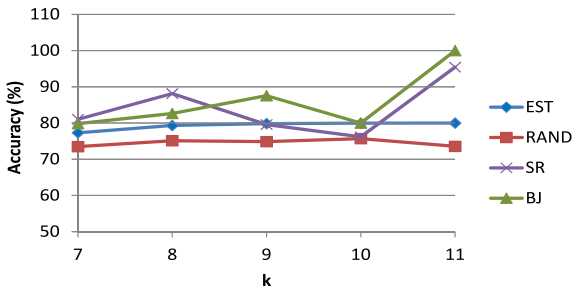


Fig. 10. Estimated and observed accuracies for Scenario 1 with period length c = 1.

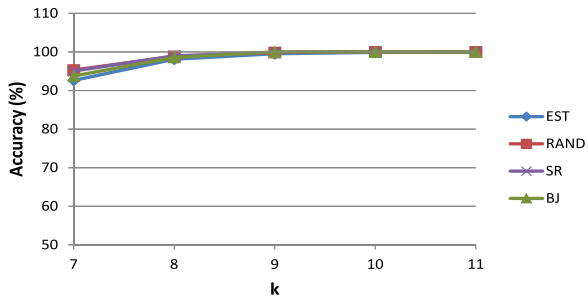


Fig. 12. Estimated and observed accuracies for Scenario 5.

The VA-Store is mainly designed to save storage space for repetitive genome sequence data. If a user wants to use k-mers with M different values/lengths for k, we employ one physical store Ω_{k_0} to keep k_0-mers of one length $k = k_0$ (e.g., the middle length) and support k-mers of other lengths logically/virtually, which would require roughly only $1/M$ of the space that would be needed to physically store all k-mers with M lengths. In fact, the space requirements to physically store the sets (stores) of k-mers with different values of k may not be the same. A larger k value leads to longer k-mers and also usually produces more distinct k-mers from a given set of reads, which would demand more space. On the other hand, for a smaller k value, although the set of distinct k-mers produced from the given set of reads is usually smaller, each k-mer may require more space to keep the relevant metadata (e.g., the ids of reads containing a given k-mer) since a smaller k-mer has a higher chance to appear in more reads. However, since the space required for metadata is relatively small, the demand for space increase caused by a larger k value usually dominates the overall space requirement for each store. The actual space requirements depend on the underlying dataset and the implementation of the stores. Fig. 9 shows the space requirements for store Ω_k ($4 \leq k \leq 10$; $k_0 = 7$) for dataset SR if each Ω_k is physically stored. When the VA-Store is adopted to keep k-mers ($4 \leq k \leq 10$) for dataset SR (i.e., only Ω_{k_0} is physically stored), we would save about 86.4 percent of the space that would be needed to physically store all k-mers ($4 \leq k \leq 10$).

One set of experiments was performed to examine how well the analytical accuracy estimation model for $k = k_0 + 1$ from Section 4.2.1 can predict the k-mer query accuracy when using physical store Ω_{k_0}. Figs. 10, 11, and 12 show the comparisons between the estimated accuracies (EST) by the model and the observed average accuracies for k-mer queries in Scenarios 1 and 5 using Ω_{k_0} for three experimental datasets RAND, SR and BJ. From the figures, we can see that the derived estimation model captures the query accuracy behaviors quite well in general. However, the estimation became worse when k became large for Scenario 1 on real datasets SR and BJ. This was because the approximate solution/result size for a k-mer query with a special pattern defined in Scenario 1 was small when k was large — there were not many reads containing such a special k-mer pattern when k was large in SR and BJ. As a result, any false positive read contained in the result would have a big impact on the accuracy. For example, a query result consisting of 1 correct read and 1 incorrect read would have an accuracy of 50 percent, while the accuracy becomes 100 percent if the incorrect read is not included. It is difficult to capture such a vulnerability in accuracy using a statistical model in such a case. On the other hand, the model captures the accuracy behavior for the random dataset RAND relatively well since such a dataset contains more reads with the searching patterns even when k is large. Similar observations were obtained for Scenarios 2, 3, and 4, which are not shown here due to the space limit. Note that the majority of k-mers fall into Scenario 5 and require no special patterns. The accuracy behavior in this scenario is captured well by our model for all k values.

Another set of experiments was conducted to show the accuracies of approximate solutions for k-mer queries using physical store Ω_{k_0} for the general case $k = k_0 + \delta$ ($\delta \geq 1$) and examine how well the regression accuracy estimation model can capture the "V" shape accuracy behavior as discussed in Section 4.2.2. Fig. 13 shows the accuracy changing patterns as δ increases (with $k_0 = 7$) for the three experimental datasets. From the figure, we can see that, as δ increases, the accuracy decreases first, then increases, and finally stays at a high accurate level. The reason behind this phenomenon has been explained in Section 4.2.2. To capture the "V" shape behavior, we suggested a regression accuracy estimation model in Section 4.2.2. Since a synthetic dataset is readily available in practice, in the experiment, we used the observed values for k-mer queries on synthetic dataset RAND to train the coefficients in the regression model. The

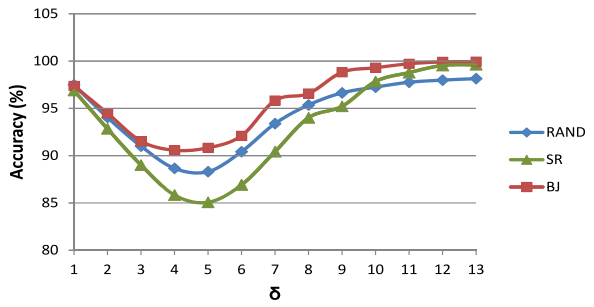


Fig. 13. Accuracy changing pattern in the general case.

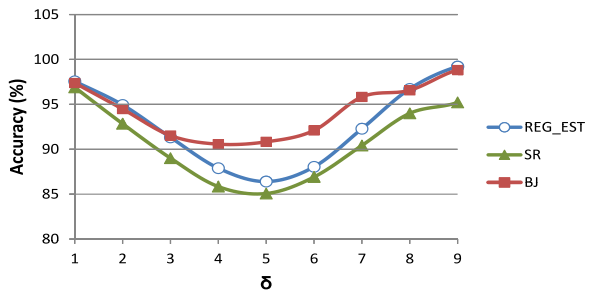


Fig. 14. Estimated accuracies using regression model.

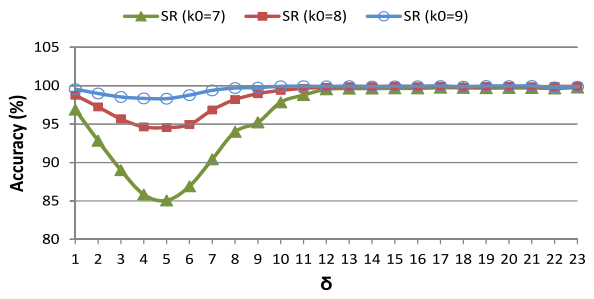


Fig. 15. Effect of physical store length k_0 on accuracy.

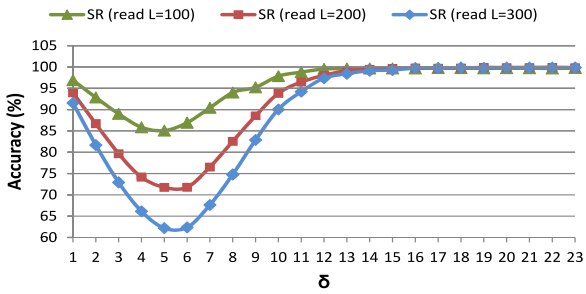


Fig. 16. Effect of read length L on accuracy.

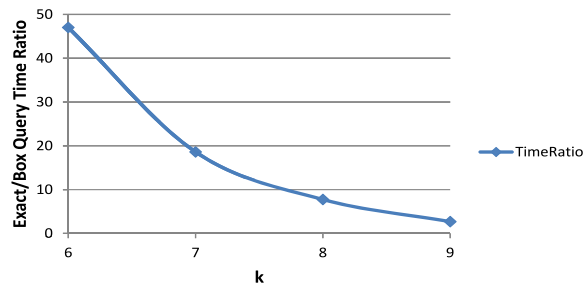


Fig. 17. Query optimization via box queries.

derived regression model was then used to estimate the accuracies (i.e., capturing the "V" shape pattern) for k-mer queries on real datasets SR and BJ. Fig. 14 shows the comparison of the estimated and observed query accuracies during the "V" shape phase. From the figure, we can see that the obtained regression model can indeed capture the "V" shape pattern and give reasonable accuracy estimates for k-mer queries on SR and BJ.

It is observed from Fig. 13 that, when k (= $k_0 + \delta$) becomes large (e.g., ≥ 20), the VA-Store method can provide a highly accurate solution for a k-mer query (with an accuracy approaching 100 percent), which is consistent with the discussion in Section 4.2.2. The low accuracy solutions occur around the "V" shape when k is relatively small (e.g., $8 \leq k \leq 16$). Furthermore, the degree of inaccuracy around the "V" shape decreases when k_0 becomes larger, as shown in Fig. 15. This is because the upper scenario in Fig. 3 that causes false positives has a less chance to occur when longer transformed k_0-mers (i.e., larger k_0) for a k-mer query are instantiated with a read. On the other hand, the degree of inaccuracy around the "V" shape increases when the length of a read becomes larger, as shown in Fig. 16 ($k_0 = 7$), since there is a higher chance for the upper scenario in Fig. 3 to occur for a larger read.

Note that, although our VA-Store method can support k-mer queries of any length in principle, it should aim to efficiently support k-mer queries of desired lengths for a given application in practice. When the length k for an input k-mer query is too far from the length k_0 chosen for the physical store Ω_{k_0} of the underlying VA-store, the efficiency will suffer since many k_0-mer queries have to be executed to answer the k-mer query. In biological sequence analysis, there are applications that use k-mers with small lengths (e.g., $6 \sim 16$) [2], [14], [22], [25], [28], [37], especially those like BLAST that apply a pre-filtering step with a low sensitivity to allow more candidates to be considered for subsequent processing. The performance study of our VA-Store method for k-mers with small lengths is particularly useful for such applications.

The next set of experiments was conducted to evaluate the performance of our query optimization strategies for processing transformed queries. The performance was measured based on the average of processing 1,000 random k-mer queries on k-mer datasets generated from dataset SR. Fig. 17 shows the savings on query processing time when applying the box query optimization strategy specified by Eq. (5) with $k_0 = 10$. From the figure, we can see that using box queries from the optimized transformation (Eq. (5)) can significantly reduce the processing time, comparing to using exact queries from the original transformation (Eq. (4)). For example, the query processing time for using exact queries is about 47.0 times of that for using box queries for $k = 6$, and the former is about 2.7 times of that for using box queries for $k = 9$. Note that a smaller k requires more transformed exact queries to process a k-mer query, in which case using box queries is more advantageous. To evaluate the performance of our Accuracy-Guided Adaptive Query Processing algorithm (AGAQ), we compared it with three direct methods for executing a k-mer query $q^{(k)}$ on Ω_{k_0} ($k > k_0$) based on Eq. (7): (1) the full method (FullM) that executes all $(\delta + 1)$ shifted k_0-mer queries for $q^{(k)}$; (2) the random method (RandM) that randomly selects a subset of shifted k_0-mer queries for $q^{(k)}$ to execute; (3) the sequential method (SeqM) that sequentially selects a subset of shifted k_0-mer queries for $q^{(k)}$ to execute. RandM and SeqM apply the same stop condition used by AGAQ to stop selecting the next k_0-mer query

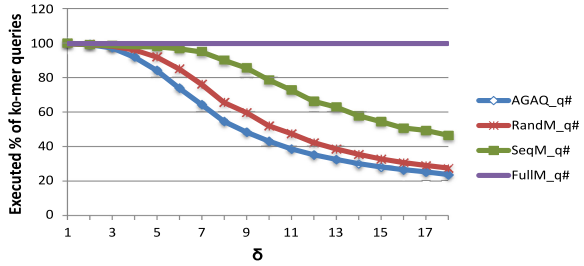


Fig. 18. Comparison of number of k_0-mer queries executed.

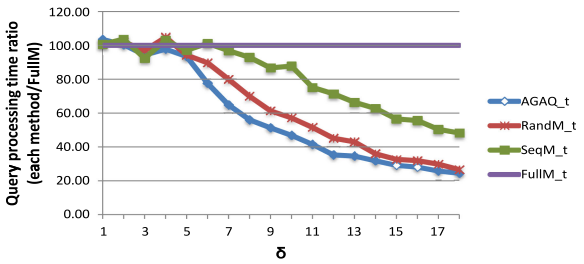


Fig. 19. Comparison of query processing time.

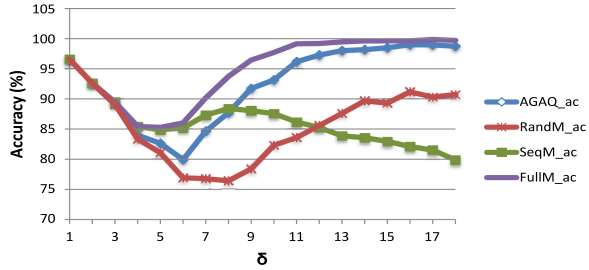


Fig. 20. Comparison of accuracy achieved.

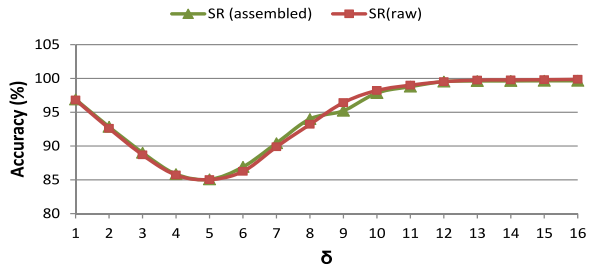


Fig. 21. Effect of using assembled and raw reads.

to execute. Fig. 18 shows the number (percentage) of k_0-mer queries executed by each of the compared methods when processing $q^{(k)}$. From the figure, we can see that AGAQ executed the least number of k_0-mer queries during the query processing. As a result, it usually has the minimum query processing time (see Fig. 19). Fig. 20 shows the query accuracy achieved by each of the compared methods. From the figure, we can see that AGAQ can generally achieve a better accuracy, comparing to SeqM and RandM except a few cases in which SeqM is better. For these exceptions, we noticed that SeqM executed almost all the k_0-mer queries just like FullM, which incurred higher query processing cost. AGAQ is increasingly better when δ increases. To examine the scalability of AGAQ with respect to the dataset size, we also tested it for k-mer datasets generated from a larger dataset RN for Rattus norvegicus genome with 6,886,157 reads (i.e., about 10.8 times larger than dataset SR) and observed similar results. For example, the executed percentage of k_0-mer queries, the query processing time ratio, and the accuracy of AGAQ for $\delta = 9$ on RN are 46.6 percent (versus 48.3 percent for SR in Fig. 18), 51.8 percent (versus 51.3 percent for SR in Fig. 19), and 93.2 percent (versus 91.7 percent for SR in Fig. 20), respectively. In summary, AGAQ uses a minimum processing time to achieve a very high accuracy.

Lastly, Fig. 21 demonstrates that the performance difference in accuracy between using raw reads for the genomic data and using reads obtained from the assembled genome sequence in our experiments is very small for the evaluation of the VA-Store method. Hence, comparable performance is expected when the VA-Store method is used by applications with raw reads.

6 Conclusions

There often exist substantial amounts of repetitive data in different portions of a big data repository for applications in domains such as bioinformatics. Directly storing repetitive data on disk would waste much space and limit the scalability of the techniques using the data. We notice that there are some useful relationships among the repetitive data in important applications such as genome sequence analysis problems. Although such relationships (e.g., containment, implications) may not be a simple duplication with a clear boundary, we can utilize them to transform queries on one portion of the dataset into queries on another related (repetitive) portion of the dataset. The transformations may be precise or approximate. Based on this observation, in this paper, we present a virtual approximate store approach, called the VA-Store, to supporting genome sequence analysis applications on repetitive big sequencing data. The main contributions of our work are summarized as follows:

- A kernel database problem (KDBP$_k$) on a set of k-mers for many sequence analysis applications such as local alignment searching, sequence assembly, terminus searching, and error correction is identified.
- A method using one physical store Ω_{k_0} and multiple virtual stores Ω_k ($k \neq k_0$) to support sequence analysis applications with repetitive big data is suggested. In particular, for $k < k_0$, a precise transformation rule to convert a k-mer query (i.e., problem KDBP$_k$) on Ω_k into an equivalent set of k_0-mer queries (i.e., problems KDBP$_{k_0}$'s) on Ω_{k_0} is given. For $k > k_0$, an approximate transformation rule to convert a k-mer query on Ω_k into an optimal set of k_0-mer queries on Ω_{k_0} is provided.
- Accuracy estimation models for approximate solutions obtained from the approximate transformation rule when $k > k_0$ are derived. In particular, an analytical accuracy estimation model dealing with various query patterns for $k = k_0 + 1$ is derived. A general regression accuracy estimation model for $k \geq k_0$ is developed.
- An optimization strategy to combine multiple transformed (exact) k_0-mer queries into one box k_0-mer query for $k < k_0$ is suggested. For $k > k_0$, an efficient and effective accuracy-guided adaptive query processing algorithm is developed so as to execute as few queries as possible while keeping a high accuracy. In particular, a maximum distance principle is adopted to choose as diverse queries as possible for execution to cover a broad range of scenarios with a small number of executed queries. A binary search tree strategy

is used to efficiently realize the maximum distance principle. In addition, an easy-checking effective stop condition is suggested.
- Extensive experiments were conducted, which demonstrate that the proposed VA-Store approach is quite promising in efficiently and effectively solving a kernel database problem on repetitive big data in bioinformatics.

The VA-Store showcases a feasible way to utilize important relationships among repetitive data to develop effective storage methods and efficient processing algorithms for useful analytics on repetitive big data. It provides a new query-based approach to facilitating genome sequence analyses for large datasets on disk. It represents a novel contribution to the areas of bioinformatics, database query processing and optimization, and big data processing.

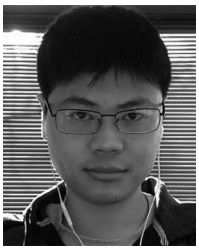
Our work is just the beginning of further research that needs to be done in order to completely solve the problems associated with supporting virtual approximate stores. Our future work includes further improving the accuracy estimation model (e.g., incorporating other combinatorial terms and more parameters in the regression model), studying other adaptive query processing algorithms (e.g., utilizing the patterns contained in the given k-mer query to decide the next best k_0-mer query to execution), determining the optimal k_0 for the physical store based on the underlying applications, supporting multiple physical stores to avoid the worst accuracy occurring at the bottom of the "V" shape, and developing an implementation framework for the physical store using the Hadoop/MapReduce environment.

Acknowledgments

Research was supported by the US National Science Foundation (NSF) (under Grants #IIS-1320078 and #IIS-1319909), the University of Michigan, and the Michigan State University.

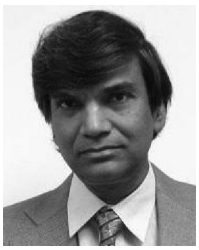
References

[1] M. S. Akshay, S. Mohan, et al., "Efficient support of big data storage systems on the cloud," in *Proc. Int. Workshop Cloud Comput. Appl.*, 2013, pp. 1–5.
[2] S. F. Altschul, W. Gish, et al., "Basic local alignment search tool," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, 1990.
[3] A. Bankevich, S. Nurk, et al., "SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing," *J. Comput. Biol.*, vol. 19, no. 5, pp. 455–77, 2012.
[4] C. Boucher, A. Bowe, et al., "Variable-order *de Bruijn* graphs," CoRR, 2014. [Online]. Available: http://arxiv.org/abs/1411.2718
[5] C. Camacho, G. Coulouris, et al., "BLAST+: Architecture and applications," *BMC Bioinf.*, vol. 10, 2009, Art. no. 421.
[6] X. Cao, B. C. Ooi, et al., "DSIM: A distance-based indexing method for genomic sequences," in *Proc. 5th IEEE Symp. Bioinf. Bioeng.*, 2005, pp. 97–104.
[7] X. Cao, S. C. Li, and A. K. H. Tung, "Indexing DNA sequences using q-grams," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2015, pp. 4–16.
[8] J. Chang, J. Dean, et al., "Bigtable: A distributed storage system for structured data," *ACM Trans. Comput. Syst.*, vol. 26, no. 2, pp. 4:1–4:26, 2008.
[9] C. Chen, M. Lang, and Y. Chen, "Multilevel active storage for big data applications in high performance computing," in *Proc. IEEE Int. Conf. Big Data*, 2013, pp. 169–174.
[10] C. Chen, A. Watve, S. Pramanik, and Q. Zhu, "The BoND-tree: An efficient indexing method for box queries in non-ordered discrete data spaces," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 11, pp. 2629–2643, Nov. 2013.
[11] R. Chikhi and P. Medvedev, "Informed and automated k-mer size selection for genome assembly," *Bioinf.*, vol. 30, no. 1, pp. 31–37, 2014.
[12] P. Compeau, P. Pevzner, and G. Tesler, "How to apply *de Bruijn* graphs to Genome assembly," *Nat. Biotechnol.*, vol. 29, no. 11, pp. 987–991, 2011.
[13] T. C. Conway and A. J. Bromage, "Succinct data structures for assembling large Genomes," *Bioinf.*, vol. 27, no. 4, pp. 479–486, 2011.
[14] H. Cui and X. Zhang, "Alignment-free supervised classification of metagenomes by recursive SVM," *BMC Genomics*, vol. 14, 2013, Art. no. 641.
[15] Y. Gu, Q. Zhu, X. Liu, Y. Dong, C. T. Brown, and S. Pramanik, "Using disk based index and box queries for Genome sequencing error correction," in *Proc. Int. Conf. Bioinf. Comput. Biol.*, 2016, pp. 69–76.
[16] Y. Gu, X. Liu, Q. Zhu, Y. Dong, C. T. Brown, and S. Pramanik, "A new method for DNA sequencing error verification and correction via an on-disk index tree," in *Proc. ACM Conf. Bioinf. Comput. Biol. Health Informat.*, 2015, pp. 503–504.
[17] M. Guillaume and C. Kingsford, "A fast, lock-free approach for efficient parallel counting of occurrences of k-mers," *Bioinf.*, vol. 27, no. 6, pp. 764–770, 2011.
[18] R. Guillaume, D. Lavenier, and R. Chikhi, "DSK: k-mer counting with very low memory usage," *Bioinf.*, vol. 29, no. 5, pp. 652–653, 2013.
[19] S. Huang, T. W. Lam, et al., "Indexing similar DNA sequences," in *Proc. Int. Conf. Algorithmic Appl. Manage.*, 2010, pp. 180–190.
[20] T. Kahveci and A. Singh, "Efficient index structures for string databases," in *Proc. Int. Conf. Very Large Data Bases*, 2001, pp. 351–360.
[21] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, "Quake: Quality-aware detection and correction of sequencing errors," *Genome Biol.*, vol. 11, no. 11, 2010, Art. no. R116.
[22] W. J. Kent, "BLAT - the BLAST-like alignment tool," *Genome Res.*, vol. 12, no. 4, pp. 656–664, 2002.
[23] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," *Proc. VLDB Endowment*, vol. 5, no. 12, pp. 2032–2033, 2012.
[24] J. Li, Z. Xu, Y. Jiang, and R. Zhang, "The overview of big data storage and management," in *Proc. IEEE 13th Int. Conf. Cogn. Inform. Cogn. Comput.*, 2014, pp. 510–513.
[25] W. Liao, J. Ren, et al., "Alignment-free transcriptomic and metatranscriptomic comparison using sequencing signatures with variable length Markov chains," *Sci. Rep.*, vol. 6, 2016, Art. no. 37243.
[26] M. Metzker, "Sequencing technologies - the next generation," *Nature Rev. Genetics*, vol. 11, pp. 31–46, 2010.
[27] C. S. Oehmen and D. J. Baxter, "ScalaBLAST 2.0: Rapid and robust BLAST calculations on multiprocessor systems," *Bioinf.*, vol. 29, no. 6, pp. 797–798, 2013.
[28] Y. Orenstein, D. Pellow, et al., "Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing," *Comput. Biol.*, vol. 13, no. 10, 2017, Art. no. e1005777.
[29] J. Pell, A. Hintze, et al., "Scaling metagenome sequence assembly with probabilistic *de Bruijn* graphs," *Proc. Nat. Academy Sci. United States America*, vol. 109, no. 33, pp. 13272–13277, 2012.
[30] Y. Peng, H. Leung, S.-M. Yiu, and F. Y. Chin, "IDBA — A practical iterative *de Bruijn* graph de novo assembler," in *Proc. Annu. Int. Conf. Res. Comput. Mol. Biol.*, 2010, pp. 426–440.
[31] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik, "The ND-tree: A dynamic indexing technique for multidimensional non-ordered discrete data spaces," in *Proc. Int. Conf. Very Large Data Bases*, 2003, pp. 620–631.
[32] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik, "Dynamic indexing for multidimensional non-ordered discrete data spaces using a data-partitioning approach," *ACM Trans. Database Syst.*, vol. 31, no. 2, pp. 439–484, 2006.
[33] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik, "A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces," *ACM Trans. Inf. Syst.*, vol. 23, no. 1, pp. 79–110, 2006.
[34] K. Stefan, A. Narechania, and J. C. Stein, "A new method to compute k-mer frequencies and its application to annotate large repetitive plant genomes," *BMC Genomics*, vol. 9, 2008, Art. no. 517.
[35] T. Steinmaurer, P. Traxler, et al., "Combining stream processing engines and big data storages for data analysis," in *Proc. Int. Symp. Methodologies Intell. Syst.*, 2014, pp. 476–485.
[36] M. Vivien, "Biology: The big challenges of big data," *Nature*, vol. 498, pp. 255–260, 2013.
[37] D. M. Winget and K. E. Wommack, "Randomly amplified polymorphic DNA PCR as a tool for assessment of marine viral richness," *Appl. Environ. Micorbiol.*, vol. 74, no. 9, pp. 2612–2618, 2008.
[38] K. Zhao and X. Chu, "G-BLASTN: Accelerating nucleotide alignment by graphics processors," *Bioinf.*, vol. 30, no. 10, pp. 1384–1391, 2014.



Xianying Liu received the BS degree in computer science from the Xi'an Jiaotong University, China, in 2012, and the MS degree in computer and information science from the University of Michigan - Dearborn, in 2015. He is currently a software engineer with Facebook. Before that, he worked at IBM Watson Health, IBM Almaden Research Center. His research interests include: Query processing and optimization for database systems, big data processing for bioinformatics, genome sequence analysis, and data analytics for life sciences.

Qiang Zhu received the PhD degree in computer science from the University of Waterloo, Canada, in 1995. He is currently the William E. Stirton professor and the chair of the Department of Computer and Information Science, University of Michigan - Dearborn. He received numerous distinguished research awards. His research interests include query processing and optimization for database systems, big data processing, high-dimensional indexing, streaming data processing, self-managing databases, and web information systems. He is also an ACM distinguished scientist, an IBM CAS faculty fellow, and a senior member of the IEEE.



Sakti Pramanik received the BE degree in electrical engineering from the Calcutta University, the MS degree in electrical engineering from the University of Alberta, Edmonton, Canada, and the PhD degree in computer science from the Yale University. He was awarded the University's gold medal for securing the highest grade among all branches of engineering from Calcutta University. He is currently a professor with the Department of Computer Science and Engineering, Michigan State University. His research interests include high-dimensional indexing, genome sequence analysis, and multimedia databases.

C. Titus Brown received the BA degree in mathematics from Reed College, in 1997, and the PhD degree in developmental biology from the California Institute of Technology, in 2007. He is currently an associate professor with the Genome Center and the School of Veterinary Medicine, University of California, Davis. He was also a faculty member with the Department of Computer Science and Engineering, Michigan State University. His research interests include bioinformatics, genomics, developmental biology, and next-generation sequencing data.

Gang Qian received the BS and MS degrees both in computer science from Shanghai Jiao Tong University, China, respectively, and the PhD degree in computer science from Michigan State University, in 2004. He is currently a professor and the chair of the Department of Computer Science, University of Central Oklahoma. His research interests include high-dimensional indexing, bioinformatics, multimedia databases, information retrieval for text and document databases, and machine learning.