Finding the Nearest Neighbors in Biological Databases Using Less Distance Computations

Jianjun Zhou[1,2], Jörg Sander[1,3], Zhipeng Cai[1,4], Lusheng Wang[5], Guohui Lin[1,6]

[1]Department of Computing Science, University of Alberta. Edmonton, Alberta T6G 2E8, Canada.

[2]Email: jianjun@cs.ualberta.ca.

[3]To whom correspondence on the VP method should be addressed. Email: joerg@cs.ualberta.ca.

[4]Email: zhipeng@cs.ualberta.ca.

[5]Department of Computer Science, City University of Hong Kong. Kowloon, Hong Kong. Email: cswangl@cityu.edu.hk.

[6]To whom other correspondence should be addressed. Fax: (780) 492-1071. Email: ghlin@cs.ualberta.ca.

Abstract

Modern biological applications usually involve the similarity comparison between two objects, which is often computationally very expensive, such as whole genome pairwise alignment and protein three dimensional structure alignment. Nevertheless, being able to quickly identify the closest neighboring objects from very large databases for a newly obtained sequence or structure can provide timely hints to its functions and more.

This paper presents a substantial speedup technique for the well studied k-nearest neighbor (k-nn) search, based on novel concepts of virtual pivots and partial pivots, such that a significant number of the expensive distance computations can be avoided. The new method is able to dynamically locate virtual pivots, according to the query, with increasing pruning ability. Using the same or less amount of database preprocessing effort, the new method outperformed the second best method by using no more than 40% distance computations per query, on a database of 10,000 gene sequecnes, compared to several best known k-nn search methods including M-Tree, OMNI, SA-Tree, and LAESA. We demonstrated the use of this method on two biological sequence datasets, one of which is for HIV-1 viral strain computational genotyping.

Index Terms

Nearest neighbor search, metric space, triangle inequality pruning, virtual pivot, partial pivot, HIV-1 computational genotyping.

I. Introduction

The rapid growth of biological sequence and structure databases imposes a challenging problem of how to access them efficiently. For a classic instance, when scientists are to study a new biological sequence or structure, they may want to see if there is any similar sequence or structure in the existing databases so that they can use the knowledge on the existing objects to infer the properties of the new object. In the case of primary sequences only, fast homology search tools such as BLAST [1] and FASTA [2] may be used for those queries which have very similar sequences already deposited in the database. Nevertheless, though with various levels of success and further developments, neither of them is able to guarantee to locate the most similar sequences for an arbitrary query. The main cause is that these homology search tools are largely heuristics, and they typically fail on queries which do not have very similar sequences deposited in the database [3]. Finding the most similar sequences (or structures) from an existing database can be easily modeled as a similarity search problem such as the k-nearest neighbor (k-nn) search studied in the database community [4]. In k-nn search, the basic task is to efficiently retrieve from a large set of database objects the k most similar ones to a given query object, measured by a distance (or similarity) function — the smaller the distance between objects, the more similar they are.

Both the large number of database objects and the time-consuming distance computation between two objects can make the k-nn search slow, or even prohibitive, in real-life applications. How k-nn search can be done efficiently has been extensively studied in the database and theory community and numerous methods have been proposed. However, while the problem has well known solutions in low-dimensional vector space, for example, through R-trees [5], an efficient solution is still an elusive goal for k-nn search in many important real-life applications. In these real-life applications, usually the data objects are from very high dimensional vector spaces or general metric spaces in which the data objects behave, from an indexing point of view, like very high dimensional data. Such databases often have the following characteristic: each data object typically has only a few close neighbors (objects that are similar in a meaningful way), while the other objects are very far away from it at a similar distance (*i.e.*, approximately to the same degree of large dissimilarity). For example, according to our computational experience, when indexing all protein sequences using a global sequence alignment dissimilarity measure, sequences within the same protein family typically form a very tight cluster and have very small distances to each other, while sequences from different families have very large distances. This interesting characteristic of very high dimensional databases renders most indexing structures ineffective. Currently, there is no exact k-nn search method that can handle such databases efficiently [6], [4], [7], [8]. We call this kind of databases "hard"-to-index databases.

In these "hard" databases, the distance function is typically very expensive in terms of CPU time. For instance, in biological databases the distance (or similarity) functions usually involve optimization operations such as sequence or structure alignments. When using the global sequence alignments, computing the distance (or similarity) between two genomic sequences of length around 10,000 nucleotides may take seconds on a state-of-the-art PC. This property makes the naïve solution to k-nn search, that computes the distances between the query and all database objects, unacceptable for interactive applications that require quick response time or data mining applications that require very large numbers of k-nn searches. Particularly note that a modern database, such as the Protein Data Bank, may contain thousands or even millions of objects.

Consequently, the primary goal of efficient k-nn search in these "hard" databases is to reduce the number of distance computations. For that purpose, many applications have regulated their expensive distance function so that the distance function and the database together form a so-called *metric space*. A metric space is given by a set of data objects B and a distance function $d(.,.)$ that satisfies positiveness, symmetry, reflexivity, and triangle inequality (for any $x, y, z \in B, d(x,y) + d(y,z) \geq d(x,z)$). In a metric space, the triangle inequality is the fundamental mechanism to estimate the distance between two data objects, which can be done in time often negligible compared to distance computation. In many existing search methods, the distances between (all or a portion of) data objects and a set of pre-selected reference data objects are precomputed. At query time, the distance $d(q,p)$ between the query object q and a reference object p is computed first. Then, the triangle inequality on $\langle o, p, q \rangle$ can be used to derive a lower bound and an upper bound for the distance $d(q,o)$ between the query object and an arbitrary non-reference object o, by:

$$|d(q,p) - d(o,p)| \leq d(q,o) \leq d(o,p) + d(q,p). \tag{1}$$

The pre-selected reference data objects such as the above object p are called *pivots*. When there is more than one pivot, the combination of estimated bounds from all pivots in a set P using Eq. (1) leads to the largest lower bound

$$l_{d(o,q)} = \max_{p \in P} \{|d(q,p) - d(o,p)|\} \tag{2}$$

and the smallest upper bound

$$u_{d(o,q)} = \min_{p \in P} \{d(q,p) + d(o,p)\} \tag{3}$$

for $d(q,o)$. These bounds can be used to prune away objects that are not within a certain distance threshold from query object q, thus avoiding computing their distances to query object q.

There are several ways to pre-select the set P of pivots and organize them for query purpose, which can be classified into two basic approaches: (1) methods that organize the pivots in a hierarchical tree structure and (2) methods that use a non-hierarchical approach. In both approaches, the general goal is, with a reasonable amount of efforts on preprocessing the database, to answer every k-nn query using only a small number of distance computations.

A. Related Works

Non-hierarchical k-nn search methods [9], [10], [11], [12], [13], [14] select in the precomputing stage a set of pivots and compute the distance between each pivot and every database object; during query processing, all the pivots are used collectively to prune objects from the k-nn candidate list using the triangle inequality. That is, any object whose estimated lower bound is larger than the k-th smallest estimated upper bound cannot be among the k nearest neighbors of the query, and thus can be pruned away.

The naïve way to select pivots is to pick them randomly. Some researchers preferred using pivots that are far away from all the other database objects or pivots that are far away from each other [9], [10]. One can also choose not to select pivots in batch mode (*i.e.*, select all the pivots at the same time), but incrementally: Filho *et al.* [11], [15] proposed to select pivots so that they are far away from each other and at the most similar pairwise distances to each other; Bustos *et al.* [12] proposed to select pivots to maximize the mean of the pairwise distances between pivots; Rico-Juan and Micó [13] proposed a similar method LAESA, to be further described below. One recent interesting work by Digout *et al.* [14] pre-selects a large set of pivots but uses only some of them to do the actual pruning in the query stage. However, the method relies on the distances between the query object and all the pivots, which are computed directly, to determine which subset of the pivots to be used for pruning purpose. Therefore, when the distance function is very expensive, this method can not afford to use a large number of pre-selected pivots.

LAESA [13] is a nearest neighbor search algorithm, which can be easily modified to perform k-nn search (as we did in this work). In the preprocessing stage, a set of fixed pivots are selected and the distance between each pivot and every data object is computed. To select the pivots, LAESA uses a method called *maximum minimum distances* [13], to avoid selecting pivots that are close to each other. The first pivot is selected randomly. The other pivots are selected iteratively, with the next pivot always being the object that has the maximum distance to the set of already selected pivots. In the query stage, LAESA computes the distance between the query object and a fixed pivot with the lowest lower bound estimation, updates the upper bound δ of the nearest neighbor distance and prunes away objects with a lower bound estimation greater than δ. If there are no more fixed pivots to use, it computes the distances to data objects that have not been pruned yet, one at a time, until the nearest neighbor is found. During this process, each computed distance is used to update δ and to further prune objects.

Pivots can also be selected and organized into a hierarchical *tree* structure. These tree-based k-nn search methods [16], [17] process a query in a recursive manner starting from the root of the tree, following certain paths in the tree that cannot be excluded for the query. At a particular node in the tree, with a given query range, these methods typically compute the distance between the query object and the pivot in the tree node, and apply triangle inequality to prune certain subtrees from the search. Two of the most famous tree pivot structures are M-Tree [16] and SA-Tree [17], to be further described below. For a comprehensive survey on tree-based k-nn search methods, see Chávez *et al.* [6] and Hjaltason and Samet [4].

M-tree [16] splits the data-set into spheres represented by nodes. Each node can hold a limited number of objects, and (except the root node) is led by a pivot (called *routing object* in [16]) which is stored in the parent node. Every pivot maintains a *covering radius* that is the largest distance between itself and all the data objects in its *covering tree*, which is the subtree rooted at the node led by this pivot. During the M-tree construction stage, when a new data object arrives, its distances to pivots stored in the root node are computed and it is inserted into the most suitable node (*i.e.*, led by the closest pivot from the root node) or the node that minimizes the increase of the covering radius, and subsequently descended down to a leaf node. If the size of a node reaches the pre-specified limit, this node is split into two new nodes with their pivots elected accordingly and inserted into their parent node. In the query stage, M-tree uses the covering radius to perform triangle inequality pruning. For a node led by pivot p with covering radius $r(p)$, the distance between any data object o in this node and query q, $d(q, o)$, is lower bounded by $d(q, p) - r(p)$ [16].

SA-Tree [17] uses spatial approximation inspired by the Voronoi diagram to reduce the number of visited subtrees during the query stage. In SA-Tree, every node corresponds to a database object (unlike in M-Tree). In the SA-tree construction stage, first, a random data object is chosen as the root and a set of data objects from the whole database are selected as its neighbors. Any object in this neighbor set must be closer to the root than to any other object in this neighbor set. Each neighbor object then starts a subtree of the root and the process goes on recursively. In the query stage, besides using the covering radius to perform triangle inequality pruning the same as in M-Tree, SA-Tree also uses a property derived from the way the tree is constructed. This is, if p_1 and p_2 are a pair of parent-child nodes (serving as pivots) visited by the query process, then for any data object o in the neighbor set of node p_2, $d(o, p_1) > d(o, p_2)$ and thus $\frac{1}{2}(d(q, p_2) - d(q, p_1))$ is a lower bound of $d(q, o)$ [17].

A common problem in tree-based k-nn search methods is that close objects always have certain probability of being stored into different branches and this probability goes up quickly as the difficulty of indexing the database increases (e.g., when the dimensionality increases in vector data). These methods then have to search many subtrees to retrieve all close neighbors of the query object, which will involve a large overhead and a large number of distance computations. Another disadvantage of tree-based methods is that there is no easy way to adjust the pruning ability, when allowed, through varying the database preprocessing effort. Results in Weber *et al.* [18] and Jagadish *et al.* [19] have shown that tree-based k-nn search methods typically do not work well in very high dimensional metric data.

B. Two Novel Ideas

For non-hierarchical k-nn search methods, there are two ways to improve the pruning ability: one to increase the number of pivots, and the other to use more "effective" pivots. In those methods that precompute the distances between each pivot and every database object and subsequently compute the distances between the query object and all pivots, using a larger number of pivots implies a larger number of distance computations in not only the database preprocessing stage but also the query stage. Consequently, the distance computations will dominate the runtime when the distance function is expensive enough, and even make those k-nn search methods prohibitive on such real-life applications. Therefore, using more "effective" pivots (while keeping the number small) is the more attractive approach. The effectiveness of a pivot is measured by the accuracy of the estimated lower (and upper) bounds during the query stage. Clearly from Eq. (1), the closer a pivot to the query object, the more effective it is [17]. However, given that "traditional" pivots are selected in the database preprocessing stage, *i.e.*, they are fixed during the query stage, often there is no pivot which is close to a query. In this study, we propose to dynamically locate certain database objects, besides those fixed pivots, that are hopefully close to the query object and thus can take up the roles of effective pivots to perform triangle inequality pruning. These new pivots are called *virtual pivots*.

We also extend this novel concept of virtual pivot to use every data object as a pivot. The difference between such a pivot and the fixed pivots is that for every data object, we predict a number of its nearest neighbors using the fixed pivots together with triangle inequality, and then compute the distances between the data object and these predicted nearest neighbors. Again from Eq. (1), the idea of using such a pivot is that, since it is expected to be close to these predicted nearest neighbors, it will be an effective pivot in estimating the lower (and upper) bounds of the actual distances between the query object and these predicted nearest neighbors. All these are done in the database preprocessing stage, and we will demonstrate that these additional preprocessing efforts are paid off by the significantly reduced numbers of distance computations during the query stage.

C. Organization

In the next section, we detail our method that implements the above two novel ideas for speeding up k-nn search, that is, one to spend extra efforts in database preprocessing stage to compute the distances between each data object and a number of its predicted nearest neighbors, and the other to locate virtual pivots, which are likely effective, for a query object. We also present details on how these two new types of pivots are used in lower and upper bound estimations, by triangle inequality, for pruning purpose. We show the computational results on two biological datasets in the Results section, where several existing well-known hierarchical and non-hierarchical k-nn search methods are included for comparison. In the Discussion section, we further discuss our method and outline its key differences to the other methods.

II. METHODS

From Eq. (1), we see that an effective pivot would be one that is close to the query object, and this effectiveness is meaningful for all data objects. Alternatively, a pivot can be effective for only a portion of data objects provided that these data objects are all close to the pivot. It is often difficult to select the first kind of universally effective pivots, given that they have to be selected during the database preprocessing stage. Assuming that query objects follow the same distribution as the database objects, each database object is typically close to only a small number of other database objects. Therefore, when one is selected as a pivot, it becomes effective for also only a small number of query objects. In other words, one pivot is ineffective for the vast majority of query objects. Consequently, if an effective pivot is desired for any query object, one has to have a large pool of pivots, despite many careful considerations on how pivots should be selected [11], [12], [13].

A. Virtual Pivots

We propose to use a new type of pivots, called *virtual pivots*, in the query stage, besides the “traditional” fixed pivots. This concept enables us to use virtually any data object as a pivot to perform pruning, thus open the door to choose pivots very close to the query object, such as one of the nearest neighbors of the query. Our k-nn search method starts with the same database preprocessing as in the naïve way to randomly select a number of data objects as pivots, and precomputes the distance between each pivot and every other data object. Given a query object q, our method computes the distances between q and a (randomly chosen) subset S of the fixed pivots. These distances and the distances associated with pivots in S are used together to estimate both the lower bound and upper bound on $d(q, o)$ for those data objects o with yet unknown distances to the query, by Eqs. (2–3) [20], [8].

The non-pivot data object with the smallest estimated lower bound is taken as the first virtual pivot v, which would likely be close to the query and thus, in the absence of an effective fixed pivot, would become effective if used as a pivot. Therefore, the search method proceeds to compute $d(q, v)$ directly. Further, the triangle inequality on $\langle o, p, v\rangle$ implies that $d(o, v) \geq |d(p, o) - d(p, v)|$; and the triangle inequality on $\langle o, q, v\rangle$ implies that $d(q, o) \geq d(o, v) - d(q, v)$. Combining these two inequalities gives $d(q, o) \geq |d(p, o) - d(p, v)| - d(q, v)$, for every fixed pivot p and every data object o. Hence, virtual pivot v can be used to estimate another lower bound on $d(o, q)$ for those data objects o with yet unknown distances to the query:

$$\max_{p \in P}\{|d(p, o) - d(p, v)|\} - d(q, v) \leq d(q, o). \tag{4}$$

We note that this pruning scheme based on a virtual pivot v is significantly different from those based on fixed pivots, to be further discussed. After using $d(q, v)$ and Eq. (4) to update the estimated lower bounds and upper bounds, our k-nn search method then selects the next virtual pivot, and so on. It stops when either those k nearest neighbors are found, or a pre-specified number of virtual pivots is reached.

B. Partial Pivots

Extending the concept of virtual pivot, we use every data object as a pivot in the query stage, through extra database preprocessing efforts. Nevertheless, such a pivot would only be effective in estimating the lower (and upper) bounds of the actual distances between the query object and the nearest neighbors to this pivot, by Eq. (1). In this sense, such a pivot is only a *partial pivot*.

In our database preprocessing stage, after the fixed pivots are selected and their distances to all other data objects computed, we estimate the lower bounds on $d(o', o)$ for each non-pivot object o and those data objects o' with yet unknown distances to o, using Eq. (2). This way, for each non-pivot object o, we can predict a pre-specified number of data objects o' as its close neighbors using their estimated lower bounds. These objects o' form the *neighborhood* of object o, denoted as $N(o)$. Subsequently, we precompute these distances $d(o', o)$, for $o' \in N(o)$. In the query stage, when o is selected as a virtual pivot, $d(q, o')$ can be better bounded from below than Eq. (4) by

$$|d(o', o) - d(q, o)| \leq d(q, o') \leq d(o', o) + d(q, o), \ \forall \ o' \in N(o). \tag{5}$$

Also, when the pre-specified number of virtual pivots is reached, o can be taken advantage for possible pruning away objects in $N(o)$. That is, for any k-nn candidate o, we compute the distance $d(q, o)$, and can use Eq. (5) to possibly prune away o' if $o' \in N(o)$ was also a candidate.

C. The Complete Algorithm

Preprocessing Algorithm. Given a database B, randomly select a set of data objects as fixed pivots $P \subseteq B$ and compute the distance between each fixed pivot in P and every data object in B. Using these precomputed distances, for each data object o in B, rank all the other data objects o' in B in non-decreasing order of their estimated lower bound on $d(o', o)$ by Eq. (2). If $o \in P$, let $N(o) = B$; Otherwise, compute the distances between o and the top t objects in the ordered list, which form $N(o)$ with t pre-specified.

Query Algorithm. Given a query object q (q can be in or not in B), perform the following steps of operations:

1) (Compute the distances between q and a number of fixed pivots:) Randomly select a subset $S \subseteq P$ of the fixed pivots, and compute the distances between q and all the pivots in S.

2) (Estimate the initial k-nn distance upper bound δ:) Compute the set of upper bounds $U = \{u_{d(q,o)}\}$, where

$$u_{d(q,o)} = \begin{cases} d(q,o), & \forall\ o \in S; \\ \min_{p \in S}\{d(p,q) + d(p,o)\}, & \forall\ o \in B \setminus S. \end{cases}$$

Set δ to the k-th smallest value in U.

3) (Select the first virtual pivot:) For each data object $o \in B$ estimate its lower bound as

$$l_{d(q,o)} = \begin{cases} d(q,o), & \forall\ o \in S; \\ \max_{p \in S}\{|d(p,q) - d(p,o)|\}, & \forall\ o \in B \setminus S. \end{cases}$$

If $l_{d(q,o)} > \delta$, object o is excluded from the k-nn candidate list C, which is initialized to be B and (always) sorted in non-decreasing order of estimated distance lower bounds. Select the non-pivot object in C with the smallest lower bound as the first virtual pivot v. Add v to the set of virtual pivots V.

4) (Update δ:) Compute $d(q,v)$. If $d(q,v) > \delta$, remove v from the C. Otherwise, update both $u_{d(q,v)}$ and $l_{d(q,v)}$ as $d(q,v)$, and update δ as the k-th smallest value in U.

5) (Pruning using the virtual pivot v:) For each non-pivot k-nn candidate $o \in C \setminus N(v)$,

$$l_{d(q,o)} = \max\left\{l_{d(q,o)}, \max_{p \in P \setminus S}\{|d(p,o) - d(p,v)| - d(q,v)\}\right\};$$

for each non-pivot k-nn candidate $o \in C \cap N(v)$,

$$l_{d(q,o)} = \max\left\{l_{d(q,o)}, \max_{p \in P \setminus S}\{|d(p,o) - d(p,v)| - d(q,v)\}, |d(o,v) - d(q,v)|\right\}.$$

If $l_{d(q,o)} > \delta$, remove o from the k-nn candidate list C.

6) If there is no more virtual pivot to select, or $|C| = k$, return the list of k-nn objects;
If $|V|$ reaches the pre-specified limit, go to the next step;
Otherwise, select the non-pivot object in C with the smallest lower bound as the next virtual pivot v, add v to V, and go to Step 4).

7) (Pruning using partial pivots:) Repeat the following while $|C| > k$:
    a) Select the non-pivot object in C with the smallest lower bound as the next partial pivot o. If $l_{d(q,o)} > \delta$, return the k objects in C with the smallest distances to the query; Otherwise, compute $d(q,o)$.
    b) Update $u_{d(q,o)}$ as $d(q,o)$; For each $o' \in N(o) \cup P$, update $u_{d(q,o')}$ using Eq. (5),

$$u_{d(q,o')} = \min\left\{u_{d(q,o')}, d(o',o) + d(q,o)\right\};$$

Update δ as the k-th smallest value in U.

c) Update $l_{d(q,o)}$ as $d(q,o)$, and if $l_{d(q,o)} > \delta$, remove o from C; For each $o' \in N(o) \cup P$, update $l_{d(q,o')}$ using Eq. (5),

$$l_{d(q,o')} = \max\left\{ l_{d(q,o')}, |d(o',o) - d(q,o)| \right\},$$

and if $l_{d(q,o')} > \delta$, remove o' from C.

III. RESULTS

We tested our k-nn search method, denoted as VP, on two biological sequence datasets. The first dataset consists of 1,198 *human immunodeficiency type 1 virus* (HIV-1) complete viral sequences studied by Wu *et al.* [21]. The second dataset contains 10,000 HA gene sequences of influenza (FLU) virus (encoding *hemagglutinin*) downloaded from the NCBI GenBank [22]. For each of these two datasets, we randomly removed 100 sequences from the dataset and used them as query objects to perform k-nn search on the remaining objects. All reported performance measurements are the average values over these 100 queries.

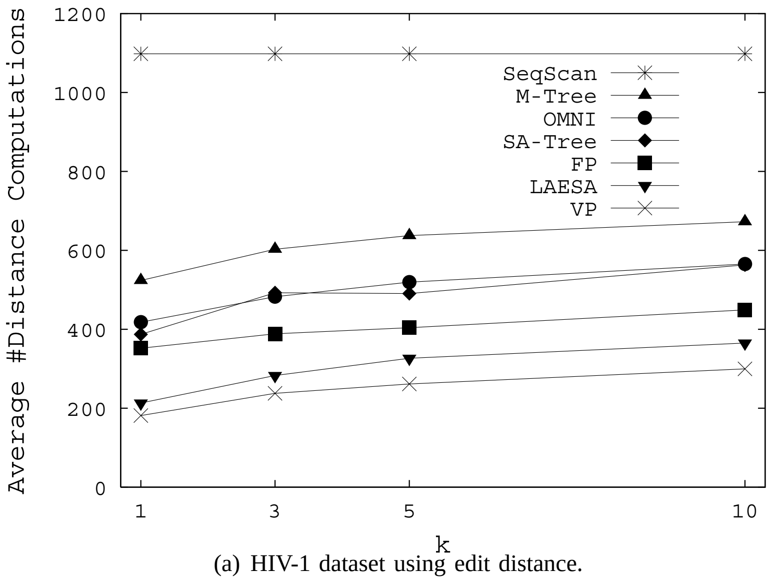
We compared our VP method with several hierarchical and non-hierarchical k-nn search methods, regarding the average number of distance computations and the average running time per query. Note that we tuned our VP method so that its preprocessing efforts, in terms of the number of precomputed distances, were less than those in the other methods. 1) The sequential scan, denoted by SeqScan, is the baseline method that computes the distances between the query and all database objects. 2) The naïve non-hierarchical pivot method uses a number of randomly selected fixed pivots, denoted by FP. In the query stage, FP computes the distances between the query objects and all the fixed pivots, and uses Eqs. (2–3) for pruning. Lastly, for each k-nn candidate, FP computes its distance to the query, updates the k-nn distance upper bound, and uses this bound to further prune objects from the candidate list. 3) The non-hierarchical pivot method by Filho *et al.* [11], denoted by OMNI, selects pivots carefully so that they are far away from each other. In the query stage, it performs the same as FP. 4) The fourth method is M-Tree [16]. 5) The fifth method is SA-Tree [17]. 6) The sixth method is LAESA [13]. Brief descriptions of the last three methods are included in the Introduction section. All these six methods are implemented in C++. The code for M-Tree was downloaded from its authors' website, and the code for LAESA was provided by its authors. All experiments were performed using an AMD Opteron 2.2GHz CPU with a 5GB RAM.

A. Results on the HIV-1 Dataset

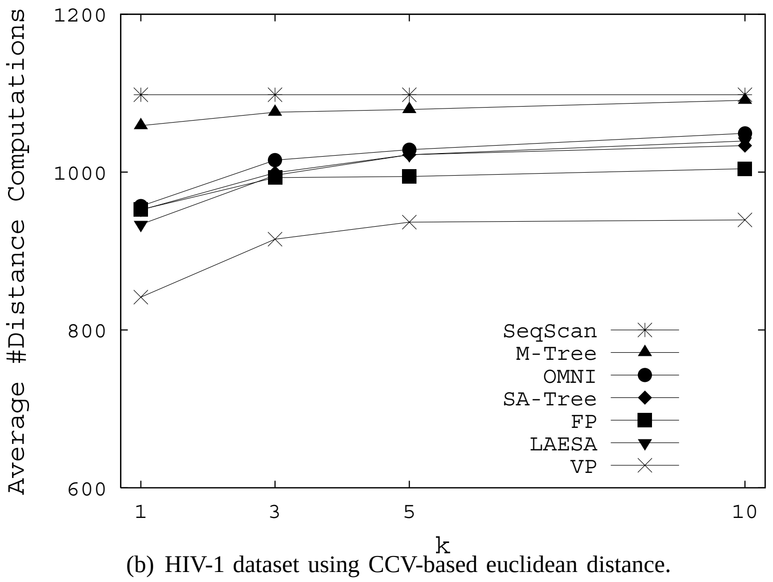
Wu *et al.* studied the HIV-1 viral strain computational genotyping problem, where 42 pure subtype HIV-1 strains were used as references and the task is to assign a subtype or a recombination to any given HIV-1 viral strain [21]. In total, there are 1,198 HIV-1 complete viral strains used in the study. The average length of these 1,198 strains is around 9,000 nucleotides. These strains form our first dataset, and they are classified into 11 pure subtypes (A, B, C, D, F, G, H, J, K, N, O) and 18 recombinations. One computational genotyping method is, given a query strain, to identify the k most similar strains in the dataset and then use their subtypes to assign a subtype to the query through a majority vote. Similarity or dissimilarity between two genomic sequences can be defined in many ways. In this work we test two of them: the general edit distance and the *Complete Composition Vector* (CCV) based euclidean distance, the latter of which was adopted in Wu *et al.* [21].

In calculating edit distances, we adopted the scoring scheme used in Mega BLAST [23], which is near optimal for aligning nucleotide sequences that differ slightly. In calculating the CCV-based euclidean distances, every HIV-1 viral strain is represented as a high dimensional vector, in which each entry records essentially the amount of evolutionary information carried by a nucleotide string. We considered all nucleotide strings of length 21 or less [21]. The distance between two HIV-1 viral strains is taken as the euclidean distance between the two corresponding high, more than two million, dimensional vectors. Both distance functions are metric. Computing the edit distance and the CCV-based euclidean distance between two HIV-1 viral strains took about 1.6 and 2.5 seconds on average, respectively.

1) Overall Performance: In FP, OMNI, and LAESA, 100 fixed pivots were selected and their distances to all data objects were computed. This way, they had equal preprocessing efforts. In our VP method, only $|P| = 80$ fixed pivots were randomly selected and similarly their distances to all data objects were computed. Afterwards, $t = 20$ close neighbors for each non-pivot object were predicted using these 80 pivots and triangle inequality, and subsequently the distance between each non-pivot object and every of its predicted close neighbors was computed. Therefore, VP had equal preprocessing efforts to FP, OMNI, and LAESA, in terms of the number of distance computations. In the query stage, for each query object, VP randomly selected and used only 5 out of the 80 fixed pivots (*i.e.*, $|S| = 5$), and the number of virtual pivots to be selected, $|V|$, was set to 10. For M-Tree, the input parameters were set according to its original paper [16]. For SeqScan and SA-Tree, there was no parameter to be set.



(a) HIV-1 dataset using edit distance.



(b) HIV-1 dataset using CCV-based euclidean distance.

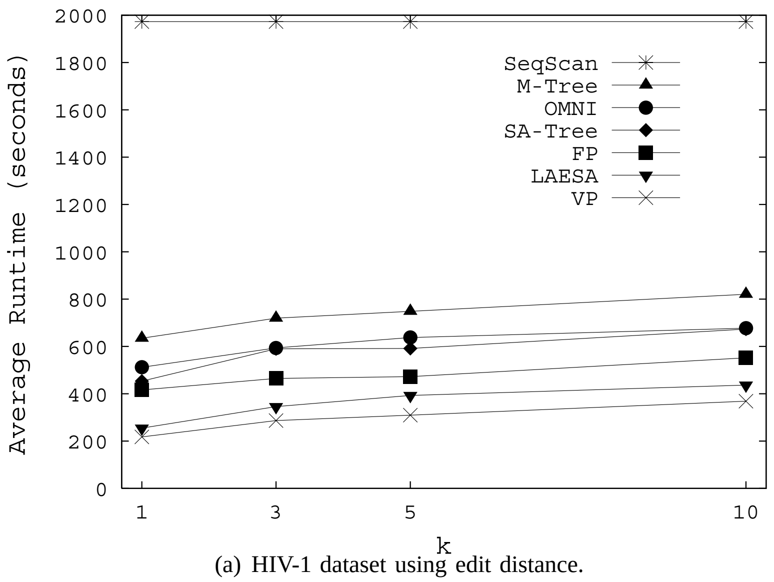
Fig. 1

THE AVERAGE NUMBERS OF DISTANCE COMPUTATIONS PER QUERY BY ALL SEVEN k-NN SEARCH METHODS ON HIV-1 DATASET, FOR $k = 1, 3, 5, 10$.

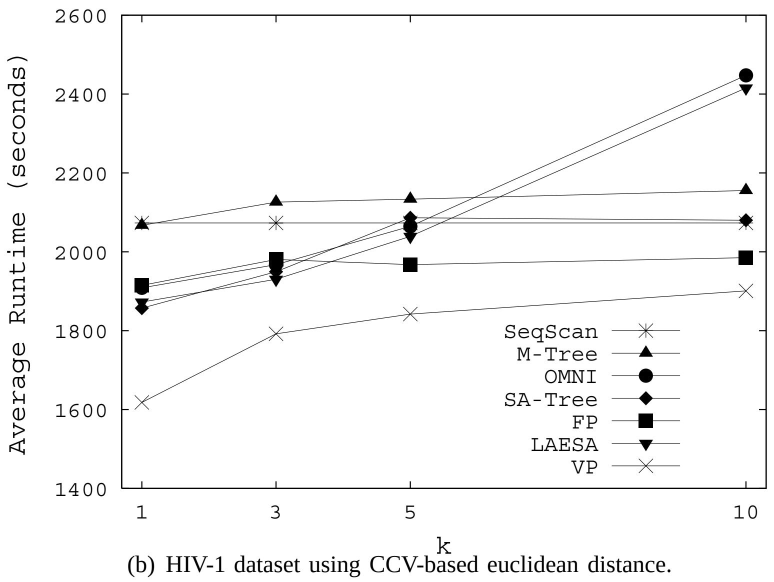
Obviously, for each query, SeqScan always made exactly 1,098 distance computations, independent of the value of k. The other six methods, which all take advantage of triangle inequality, all made less distance computations, and the detailed numbers of distance computations depended on the query HIV-1 viral strain, as well as the value of k. Figure 1(a) (Figure 1(b), respectively) plots the average numbers of edit (CCV-based euclidean, respectively) distance computations per query by these seven methods, over the 100 k-nn queries, where $k = 1, 3, 5, 10$. With respect to k, except SeqScan, each of the other six methods made more distance computations with increasing values of k. For each value of k, from the plots we can see that the general performance order is: our VP method performed the best, followed by LAESA, FP, SA-Tree/OMNI, M-Tree, and SeqScan. It was surprising to see that all these four values of k, M-Tree, OMNI, and SA-Tree made more distance computations per query than FP, which simply used 100 randomly selected fixed pivots for pruning. On the other hand, non-surprisingly, our method VP consistently outperformed all the other methods. Specifically, our VP method was at least 17.7% better than the second best method LAESA. For instance, for 1-nn search, M-Tree made 523 edit distance computations, OMNI made 418, SA-Tree made 387, FP made 351, LAESA made 213, but our VP method made only 181 edit distance computations.

The actual runtime per query by a k-nn search method includes the time for distance computations and possibly the time for triangle inequality pruning and other operations, and thus it is not necessarily proportional to the number of distance computations. Nevertheless, our collected average runtime per query by each of the seven methods, over the 100 k-nn queries, did correlate quite well with the average number of distance computations per query, as shown in Figure 2. Again, for each value of k, from the plots we can see that the general performance order is: our VP method performed the best, followed by LAESA, FP, SA-Tree/OMNI, M-Tree, and SeqScan.

2) Varying Preprocessing Efforts: We dropped SeqScan from this comparison since it performed the worst and its preprocessing effort does not affect its query performance. We mentioned above that the input parameters of M-Tree method were set according to its original paper and SA-Tree has no parameter to be tuned. For the other four methods, VP, LAESA, FP, and OMNI, we are able to tune their parameters such that they have different preprocessing efforts, which would subsequently affect their query performance. For LAESA, FP, and OMNI, their parameters are the number of fixed pivots; For our VP method, its parameters are the number of fixed pivots, the number of predicted close neighbors for each data object, the number of fixed pivots used in the query stage, and the number of virtual pivots allowed in the query stage.



(a) HIV-1 dataset using edit distance.



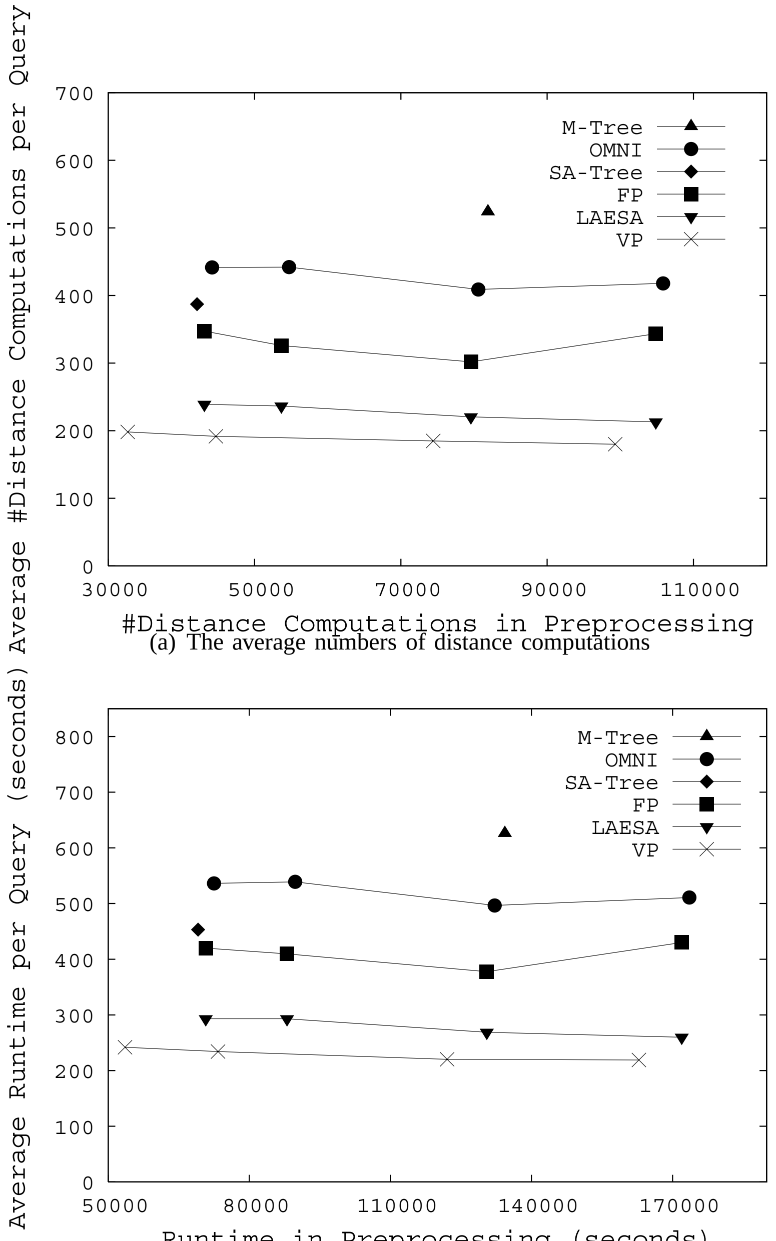
(b) HIV-1 dataset using CCV-based euclidean distance.

Fig. 2

THE AVERAGE RUNTIME PER QUERY USING BY ALL SEVEN k-NN SEARCH METHODS ON HIV-1 DATASET, FOR

$k = 1, 3, 5, 10.$

As shown in Figure 3, M-Tree always precomputed 81,910 edit distances on the database of 1,098 HIV-1 viral strains, in 134,332 seconds, and SA-Tree precomputed 42,142 edit distances



(a) The average numbers of distance computations

(b) The average runtime

Fig. 3

PERFORMANCE MEASURES PER QUERY BY THE SIX k-NN SEARCH METHODS, WITH DIFFERENT AMOUNTS OF PREPROCESSING EFFORTS ON HIV-1 DATASET USING EDIT DISTANCE.

in 69,112 seconds. We tested four different amounts of preprocessing efforts by the other four methods and their query performance in 1-nn search. We set the numbers of fixed pivots in OMNI, FP and LAESA at 40, 50, 75, and 100 such that their preprocessing efforts fell in the same range as M-Tree and SA-Tree. Correspondingly, we set the number of fixed pivots in our VP method at 10, 20, 50, and 75, and kept $(|S|, t, |V|) = (5, 20, 10)$. Figure 3 shows how varying preprocessing efforts affected the average numbers of distance computations and the average runtime per query for VP, LAESA, FP, and OMNI in 1-nn search. In these four settings, the preprocessing efforts of VP were always less than those of OMNI, FP and LAESA, yet VP performed better than them in the query stage. For instance, corresponding to the case where the numbers of fixed pivots in OMNI, FP and LAESA were set 40 (the leftmost point for each method), our VP method computed 32,669 edit distances in 53,577 seconds in the preprocessing. This effort was the smallest by comparison: LAESA computed 43,139 edit distances in 70,747 seconds, FP computed 43,140 edit distances in 70,749 seconds, and OMNI computed 44,198 edit distance in 72,484 seconds (M-Tree and SA-Tree as above). Nevertheless, VP computed only 198 edit distances per query (in 241 seconds), while LAESA computed 238 edit distance per query (in 293 seconds), FP computed 347 edit distances per query (in 419 seconds), SA-Tree computed 387 edit distances per query (in 453 seconds), OMNI computed 441 edit distances per query (in 536 seconds), and M-Tree computed 523 edit distances per query (in 625 seconds).

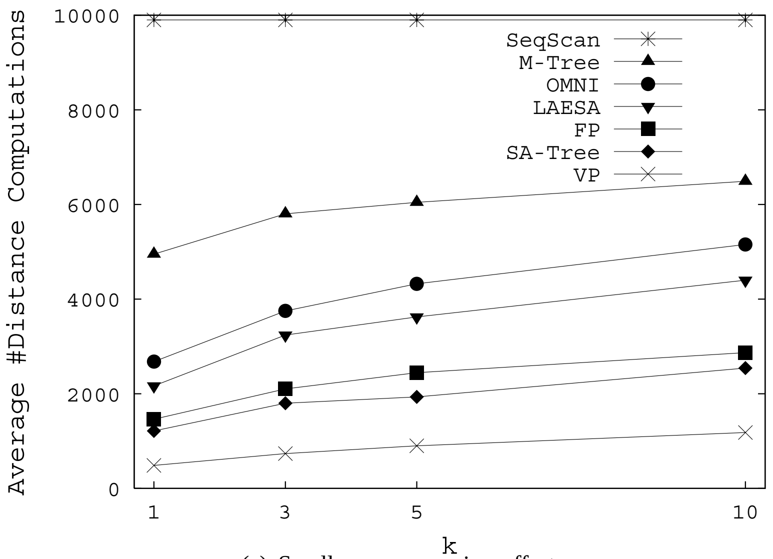
B. Results on the HA Gene Dataset

All the seven k-nn search methods were also compared on the FLU HA gene dataset, which contains 10,000 HA gene sequences. In this section we only reported the results using the CCV-based euclidean distance (the results using the edit distance were similar).

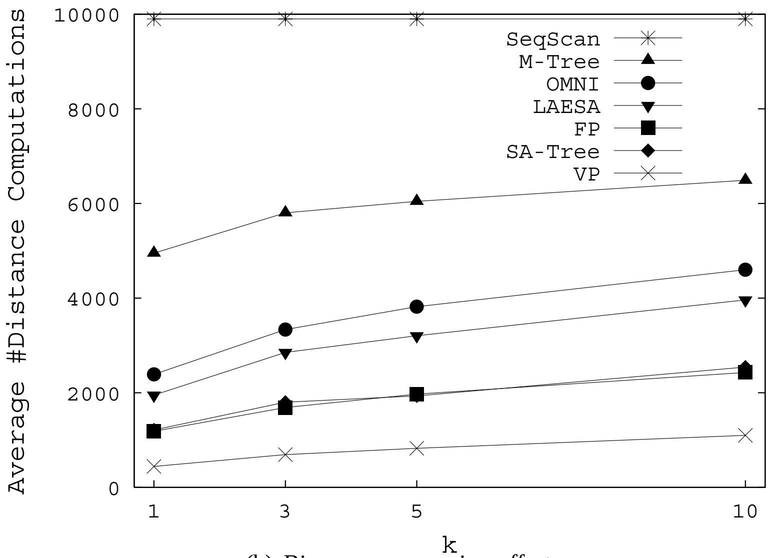
TABLE I

THE DETAILED NUMBERS OF DISTANCE COMPUTATIONS AND THE RUNTIME (IN SECONDS) BY ALL SEVEN k-NN SEARCH METHODS IN THE SMALLER PREPROCESSING, AND THEIR RESULTANT QUERY PERFORMANCE IN 1-NN SEARCH IN TERMS OF THE AVERAGE NUMBER OF DISTANCE COMPUTATIONS AND THE AVERAGE RUNTIME (IN SECONDS) PER QUERY.

	Method	VP	SA-Tree	FP	LAESA	OMNI	M-Tree
Preprocessing	#dist comp's	960,705	1,682,030	985,049	985,049	985,049	1,280,537
	runtime	263,079	460,383	269,614	269,614	269,614	350,492
Per query	#dist comp's	485	1,213	1,462	2,166	2,388	4,950
	runtime	194	373	468	716	743	2,178



(a) Smaller preprocessing effort

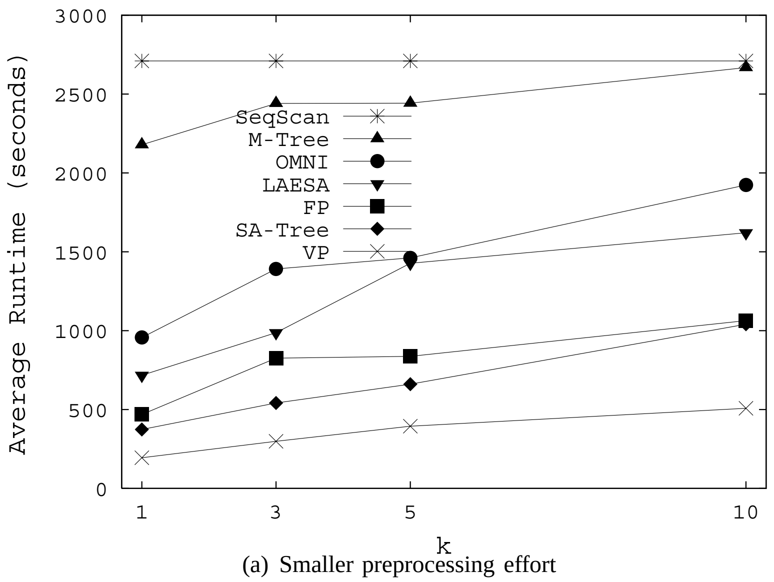


(b) Bigger preprocessing effort

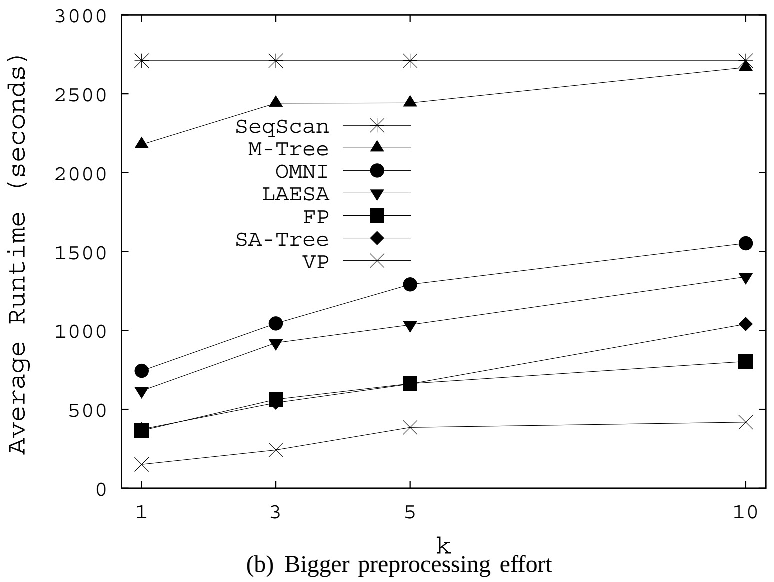
Fig. 4

THE AVERAGE NUMBERS OF DISTANCE COMPUTATIONS PER QUERY BY ALL SEVEN k-NN SEARCH METHODS WITH TWO DIFFERENT AMOUNTS OF PREPROCESSING EFFORTS, ON HA DATASET, FOR $k = 1, 3, 5, 10$.

Figures 4 and 5 show the query performance of the seven k-nn search methods on HA dataset, for $k = 1, 3, 5, 10$. We only plotted two different amounts of preprocessing efforts



(a) Smaller preprocessing effort



(b) Bigger preprocessing effort

Fig. 5

THE AVERAGE RUNTIME PER QUERY BY ALL SEVEN k-NN SEARCH METHODS WITH TWO DIFFERENT AMOUNTS OF PREPROCESSING EFFORTS, ON HA DATASET, FOR $k = 1, 3, 5, 10$.

for those k-nn search methods whose preprocessing efforts can be tuned. For M-Tree, we used its default parameter setting; For OMNI, FP, and LAESA, the numbers of fixed pivots were set at 100 and 200, respectively; Correspondingly, for our VP method, the numbers of fixed pivots were set at 75 and 150, and $(|S|, t, |V|) = (5, 20, 10)$. We distinguish these two settings as the *smaller* and the *bigger* preprocessing efforts. Table I shows that in the smaller preprocessing settings, our VP method spent the least efforts among the six methods, in both the number of distance computations and the actual runtime (SeqScan was excluded since it doesn't have the preprocessing stage). Nevertheless, both the table and Figures 4(a) and 5(a) show that our VP method outperformed the other five methods (and SeqScan) in the query stage, in both the average number of distance computations and the actual runtime per query. Furthermore, the results show a slightly different performance tendency compared to that on the HIV-1 dataset: for each value of k, our VP method performed the best, followed by SA-Tree/FP, LAESA, OMNI, M-Tree, and SeqScan. That is, LAESA performed worse than SA-Tree and FP on this FLU HA gene dataset. Our VP method can be 150% better than the second best method SA-Tree. For instance, for 1-nn search with the smaller preprocessing efforts (cf. Table I), the second best method SA-Tree made $1,213$ CCV-based distance computations, but our VP method made only 485 per query.

IV. Discussion

A. *Query Performance Dependence on the Effectiveness of Pivots*

We discussed earlier that the level of effectiveness of pivots is extremely important for the success of a query. Essentially, one effective pivot, fixed or virtual, would have the ability to estimate the distances between the query object and the non-pivot objects accurately such that a large portion of database can be pruned away to avoid distance computations.

The reported performance measurements on the two biological sequence datasets for the six k-nn search methods (excluding SeqScan) that use triangle inequality pruning were on the 100 randomly pre-selected query objects. Certainly, whether or not there exists an effective pivot for each of them would affect the query performance of these six methods. In fact, the most effective pivots for these 100 randomly pre-selected query objects have various levels of effectiveness. To study the relation between the query performance of different methods and the level of effectiveness of pivots, we calculated the nearest neighbor (*i.e.*, 1-nn) distance for each of the 100 query objects, and divided them into 5 bins of equally wide distance intervals. The nearest neighbor distance associated with the query object measures the level of effectiveness of the most

TABLE II

THE DISTANCE INTERVALS ASSOCIATED WITH THE FIVE BINS AND THE NUMBERS OF QUERY OBJECTS THEREIN.

	Bin	1	2	3	4	5
HIV-1-edit	Distance interval	[0, 0.08)	[0.08, 0.16)	[0.16, 0.24)	[0.24, 0.32)	[0.32, 0.4)
	#Objects	25	11	43	20	1
HIV-1-CCV	Distance interval	[0, 50)	[50, 100)	[100, 150)	[150, 200)	[200, 250)
	#Objects	8	9	9	12	62
HA-CCV	Distance interval	[0, 20)	[20, 40)	[40, 60)	[60, 80)	[80, 100)
	#Objects	51	34	10	4	1

effective pivot for the query. Let HIV-1-edit denote the HIV-1 dataset using the edit distance; Likewise, HIV-1-CCV and HA-CCV denote the HIV-1 and HA gene datasets using the CCV-based euclidean distance, respectively. Table II collects the distance intervals associated with the five bins and the numbers of query objects therein, for HIV-1-edit, HIV-1-CCV, and HA-CCV. Separately for each bin, we collected the average numbers of distance computations per query by all six methods in 1-nn search, and plotted in Figure 6 separately. In this experiment, the input parameters of M-Tree method were set according to its original paper, FP, OMNI, and LAESA were set to select 100 fixed pivots, and VP was set to use a less preprocessing effort than all the other five methods, with $(|P|, |S|, t, |V|) = (80, 5, 20, 10)$ for HIV-1-edit and HIV-1-CCV, $(|P|, |S|, t, |V|) = (75, 30, 20, 10)$ for HA-CCV. From these plots, one can see that the performance of all six methods declined as the average nearest neighbor distance associated with the query objects increases. Also, our VP method almost always, except for the middle bin of HIV-1-CCV, outperformed the other five methods.

Plots in Figure 6 also show that the performance of some methods deteriorates faster than the others. For instance, on HA-CCV dataset (Figure 6(c)), LAESA outperformed M-Tree and OMNI when the average nearest neighbor distance associated with the query objects is small (the first two bins), but LAESA gradually lagged behind as the average nearest neighbor distance increases (the last two bins). Table II and Figure 6(b) together explain why all six methods performed relatively poor on HIV-1-CCV. On HIV-1-CCV, most query objects (62 out of 100) were in Bin 5, that is, they have large nearest neighbor distances and consequently low levels of effectiveness of the pivots, fixed or virtual. Nevertheless, our VP method still outperformed the other five methods on this bin.

B. Advantages of Using Virtual Pivots

Given a virtual pivot v, Eq. (4) gives a new lower bound estimation $|d(p,o)-d(p,v)|-d(q,v)$ for $d(q,o)$. This estimation is derived from two adjacent triangles on four objects, and thus might be looser than a single triangle inequality as Eq. (1). Nevertheless, such a pruning scheme based on a virtual pivot v is significantly different from the traditional methods using many fixed pivots (*i.e.*, Eq. (2)), where a much larger number of distances to the query have to be computed in the query stage. Additionally, in the traditional methods, Eq. (3) gives the estimated upper bounds which might all be large; while in our VP method, if we are able to select the virtual pivot v to be close to the query q, then $d(q,v)$ will be small and serve as a tighter upper bound for the nearest neighbor distance. To summarize, there are several advantages for using a virtual pivot:

- One can freely choose from the non-pivot data objects as virtual pivots during querying time, based on the given query. In our VP method, we choose the one with the smallest estimated lower bound. This query-dependent and dynamic pivot selection will likely enable us to select virtual pivots that are close to the query object.
- The distances between the close virtual pivots and the query will provide a much tighter k-nn upper bound, δ, than the other methods. This upper bound and the new lower bound estimation by Eq. (4) can more effectively prune away distant data objects.
- In our VP method, the selection of virtual pivots in the query stage relies on only a small number of fixed pivots whose distances to the query are computed. The subsequent lower bound estimation using Eq. (4) and a larger number of fixed pivots saves a substantial number of distance computations during the query stage. Consequently, one can opt to precompute a large set of fixed pivots to improve the effectiveness of virtual pivot pruning by Eq. (4) (see also Figure 3).

C. Time and Space Complexity

Given a set of data objects B, a set of randomly selected fixed pivots P, and a constant t specifying the number of predicted close neighbors for each data object, in the preprocessing stage, our VP method computes ($|P|\cdot|B|+t\cdot|B|$) distances with an overhead of $O(|P|\cdot|B|+t\cdot$

$|B| \cdot \log |B|)$ storing, sorting, and processing the computed distances, when the prediction of the t close neighbors for each data object is done by a frontier search [24]. The space complexity is $O(|P| \cdot |B| + t \cdot |B|)$.

In the query stage with $|V|$ virtual pivots allowed, the time complexity is $O(|V| \cdot |P| \cdot |B| + t \cdot |B| + |B| \cdot \log |B|)$ distance lower and upper bound estimations and sorting the candidate list, in addition to the time for distance computations. The space complexity is $O(|P| \cdot |B| + t \cdot |B|)$, mostly for storing those precomputed distances for query processing.

Similarly as pointed out in [17], there are extreme datasets (such as the query objects in Bin 5 of HIV-1-CCV dataset) on which no pivot can provide any distance information on data objects whose distances to the query are unknown yet. Our VP method would also fail to work efficiently on them, and in the worst case the distances between the query and all data objects have to be computed. The precise theoretical analysis for our VP method on reasonable distance distributions such as the one in [17] is one of our focuses in the future work.

D. Distance Distributions

Based on a "homogeneous" distance distribution model, Ciaccia *et al.* [25] provided a theoretical analysis on the CPU cost spent on distance computations and the I/O overhead. Navarro [17] started from Delaunay graphs and proposed a simplified uniformly distributed distance model to analyze SA-Tree. It should be noted that both assumptions are on the relative distances, not the spatial distribution of the data objects. For the complete HIV-1 dataset of 1,198 viral strains, we have calculated all pairwise edit distances and CCV-based euclidean distances. These distances are plotted in Figure 7 separately. It is difficult to determine which distribution these two distances follow.

E. HIV-1 Genotyping Accuracy

The $1,198$ HIV-1 strains include 867 pure subtype strains: 70 A, 268 B, 419 C, 54 D, 10 F, 13 G, 3 H, 2 J, 2 K, 5 N, and 21 O, and 331 recombinants: 196 CRF01AE, 52 CRF02AG, 3 CRF03AB, 3 CRF04CPX, 3 CRF05DF, 8 CRF06CPX, 7 CRF07BC, 4 CRF08BC, 5 CRF09CPX, 3 CRF10CD, 10 CRF11CPX, 10 CRF12BF, 6 CRF13CPX, 7 CRF14BG, 5 CRF1501B, 2 CRF16A2D, 4 CRF18CPX, and 3 CRF19CPX. Using the majority vote, we examined the subtyping accuracies by k-nn search with $k = 1, 3, 5$. In more details, when $k = 1$, each of the

$1,198$ strains was used as a query against the other $1,197$ strains. The query strain was assigned the subtype of its nearest neighbor. When the edit distance was used, $1,188$ queries, or 99.17%, were assigned the correct subtype or recombination; when the CCV-based euclidean distance was used, 8 more queries, summing up to 99.83%, were assigned subtype or recombination correctly. The only two strains genotyped incorrectly were `DQ207943`, from pure subtype B to AB recombination, and `AY771588`, from BF recombination to pure subtype B.

When $k = 3$ (5, respectively), the subtypes and recombinations consisting of less than or equal to 3 (5, respectively) strains were removed from consideration. That is, only 8 (7, respectively) pure subtypes and 12 (8, respectively) recombinations were used, which include $1,174$ ($1,151$, respectively) strains. When the edit distance was used, $1,160$ ($1,136$, respectively) queries, or 98.81% (98.70%, respectively), were assigned the correct subtype or recombination; when the CCV-based euclidean distance was used, 6 (11, respectively) more queries, summing up to 99.32% (99.65%, respectively), were assigned subtype correctly.

TABLE III

THE HIV-1 COMPUTATIONAL GENOTYPING ACCURACIES BY k-NN SEARCH AND MAJORITY VOTE, $k = 1, 2, 3, 4, 5$.

Distance function	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
HIV-1-edit	99.165%	99.328%	98.807%	98.799%	98.696%
HIV-1-CCV	99.833%	99.832%	99.318%	99.313%	99.652%

Table III summarizes the HIV-1 computational genotyping accuracies by k-nn search coupled with majority vote, for $k = 1, 2, 3, 4$, and 5. From these five pairs of genotyping accuracies, one can see that the accuracy through the CCV-based euclidean distance was always higher than the corresponding one using the edit distance. In this sense, the CCV representation of the whole genomic sequences and the CCV-based euclidean distance seem to capture better the evolutionary information embedded in the whole genomic sequences.

ACKNOWLEDGEMENT

The authors are grateful to Juan Ramón Rico-Juan for providing the LAESA source code, and to three reviewers for many helpful suggestions and comments which improved the presentation of the paper. This research is supported by APRI, CFI, NSERC, and PrioNet.

REFERENCES

[1] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Research*, vol. 25, pp. 3389–3402, 1997.

[2] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proceedings of National Academy of Sciences of USA*, vol. 85, pp. 2444–2448, 1988.

[3] B. Ma, J. Tromp, and M. Li, "PatternHunter: Faster and more sensitive homology search," *Bioinformatics*, pp. 440–445, 2002.

[4] G. R. Hjaltason and H. Samet, "Index-driven similarity search in metric spaces," *ACM Transactions on Database Systems*, vol. 28, pp. 517–580, 2003.

[5] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD'84)*, pp. 47–57, 1984.

[6] E. Chávez, G. Navarro, R. A. Baeza-Yates, and J. L. Marroquín, "Searching in metric spaces," *ACM Computing Surveys*, vol. 33, pp. 273–321, 2001.

[7] S.-A. Berrani, L. Amsaleg, and P. Gros, "Approximate searches: k-neighbors + precision," in *Proceedings of the 2003 Conference on Information and Knowledge Management (CIKM'03)*, pp. 24–31, 2003.

[8] V. Athitsos, M. Hadjieleftheriou, G. Kollios, and S. Sclaroff, "Query-sensitive embeddings," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*, pp. 706–717, 2005.

[9] M. Shapiro, "The choice of reference points in best-match file searching," *Communications of the ACM*, vol. 20, pp. 339–343, 1977.

[10] M. L. Mico, J. Oncina, and E. Vidal, "A new version of the nearest-neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing time and memory requirements," *Pattern Recognition Letters*, vol. 15, pp. 9–17, 1994.

[11] R. F. S. Filho, A. J. M. Traina, C. Traina Jr., and C. Faloutsos, "Similarity search without tears: The OMNI family of all-purpose access methods," in *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, pp. 623–630, 2001.

[12] B. Bustos, G. Navarro, and E. Chávez, "Pivot selection techniques for proximity searching in metric spaces," *Pattern Recognition Letters*, vol. 24, pp. 2357–2366, 2003.

[13] J. R. Rico-Juan and L. Micó, "Comparison of AESA and LAESA search algorithms using string and tree-edit-distances," *Pattern Recognition Letters*, vol. 24, pp. 1417–1426, 2003.

[14] C. Digout, M. A. Nascimento, and A. Coman, "Similarity search and dimensionality reduction: Not all dimensions are equally useful," in *Proceedings of the 9th International Conference on Database Systems for Advances Applications (DASFAA'04)*, pp. 831–842, 2004.

[15] C. Traina Jr., R. F. S. Filho, A. J. M. Traina, M. R. Vieira, and C. Faloutsos, "The Omni-family of all-purpose access methods: a simple and effective way to make similarity search more efficient," *VLDB Journal*, vol. 16, pp. 483–505, 2007.

[16] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, pp. 426–435, 1997.

[17] G. Navarro, "Searching in metric spaces by spatial approximation," *The VLDB Journal*, vol. 11, pp. 28–46, 2002.

[18] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-

dimensional spaces," in *Proceedings of 24th International Conference on Very Large Data Bases (VLDB'98)*, pp. 194–205, 1998.

[19] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "iDistance: An adaptive b$^+$-tree based indexing method for nearest neighbor search," *ACM Transactions on Database Systems*, vol. 30, pp. 364–397, 2005.

[20] J. Vleugels and R. C. Veltkamp, "Efficient image retrieval through vantage objects," *Pattern Recognition*, vol. 35, pp. 69–80, 2002.

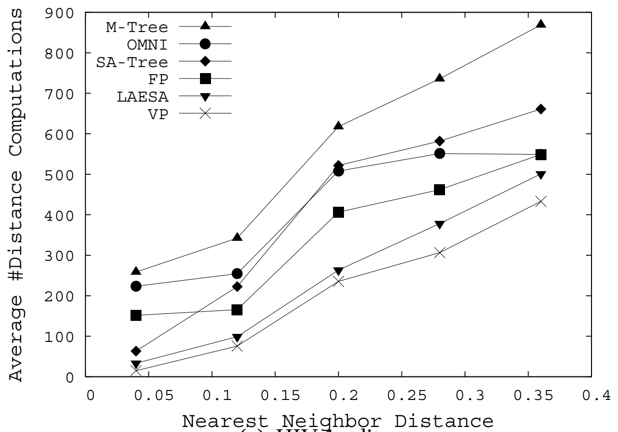
[21] X. Wu, Z. Cai, X.-F. Wan, T. Hoang, R. Goebel, and G.-H. Lin, "Nucleotide composition string selection in HIV-1 subtyping using whole genomes," *Bioinformatics*, vol. 23, pp. 1744–1752, 2007.

[22] `http://www.ncbi.nlm.nih.gov/genomes/FLU/`.

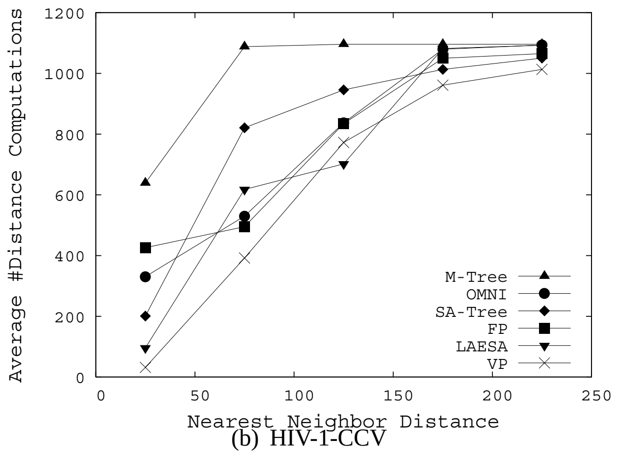
[23] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller, "A greedy algorithm for aligning DNA sequences," *Journal of Computational Biology*, vol. 7, pp. 203–214, 2000.

[24] J. Zhou and J. Sander, "Speedup clustering with hierarchical ranking," in *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM'06)*, pp. 1205–1210, 2006. A longer version appears as `http://www.cs.ualberta.ca/TechReports/2008/TR08-09/TR08-09.pdf`.

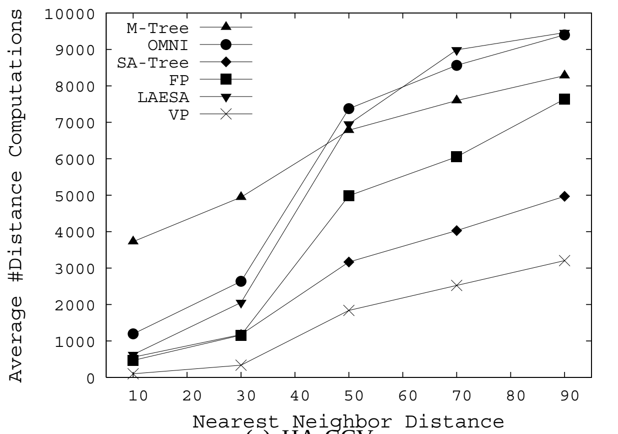
[25] P. Ciaccia, M. Patella, and P. Zezula, "A cost model for similarity queries in metric spaces," in *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98)*, pp. 59–68, 1998.



(a) HIV-1-edit



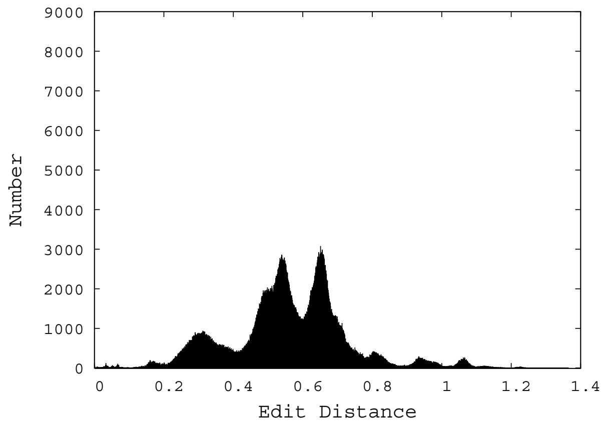
(b) HIV-1-CCV



(c) HA-CCV

Fig. 6

THE AVERAGE NUMBERS OF DISTANCE COMPUTATIONS PER QUERY BY ALL SIX METHODS IN 1-NN SEARCH FOR FIVE BINS OF QUERY OBJECTS WITH DIFFERENT NEAREST NEIGHBOR DISTANCE RANGES, ON THREE DATASETS.



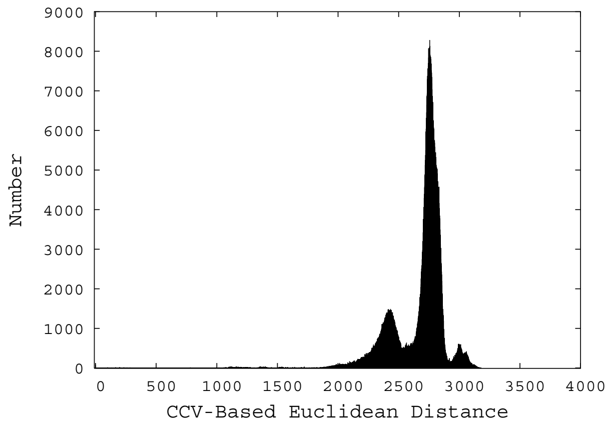


Fig. 7

THE DISTRIBUTIONS OF ALL PAIRWISE EDIT DISTANCES AND ALL PAIRWISE CCV-BASED EUCLIDEAN DISTANCES ON THE COMPLETE HIV-1 DATASET OF 1,198 VIRAL STRAINS.