Pigeonring: A Principle for Faster Thresholded Similarity Search

Pigeonring: A Principle for Faster Thresholded Similarity Search

Jianbin Qin
The University of Edinburgh
United Kingdom
jqin@inf.ed.ac.uk

Chuan Xiao *
Nagoya University
Japan
chuanx@nagoya-u.jp

ABSTRACT

The pigeonhole principle states that if n items are contained in m boxes, then at least one box has no more than n/m items. It is utilized to solve many data management problems, especially for thresholded similarity searches. Despite many pigeonhole principle-based solutions proposed in the last few decades, the condition stated by the principle is weak. It only constrains the number of items in a single box. By organizing the boxes in a ring, we propose a new principle, called the pigeonring principle, which constrains the number of items in multiple boxes and yields stronger conditions.

To utilize the new principle, we focus on problems defined in the form of identifying data objects whose similarities or distances to the query is constrained by a threshold. Many solutions to these problems utilize the pigeonhole principle to find candidates that satisfy a filtering condition. By the new principle, stronger filtering conditions can be established. We show that the pigeonhole principle is a special case of the new principle. This suggests that all the pigeonhole principle-based solutions are possible to be accelerated by the new principle. A universal filtering framework is introduced to encompass the solutions to these problems based on the new principle. Besides, we discuss how to quickly find candidates specified by the new principle. The implementation requires only minor modifications on top of existing pigeonhole principle-based algorithms. Experimental results on real datasets demonstrate the applicability of the new principle as well as the superior performance of the algorithms based on the new principle.

PVLDB Reference Format:
Jianbin Qin and Chuan Xiao. Pigeonring: A Principle for Faster Thresholded Similarity Search. *PVLDB*, 12(1): 28-42, 2018.
DOI: https://doi.org/10.14778/3275536.3275539

1. INTRODUCTION

The pigeonhole principle (a.k.a. Dirichlet's box principle or Dirichlet's drawer principle) is a simple but a powerful tool in combinatorics. It has been utilized to solve a variety of data management problems, especially for search problems involving approximate match and threshold constraints, such as Hamming distance search and set similarity search. The pigeonhole principle has many forms. For data management, the most frequently used form is stated below (though sometimes it is not explicitly claimed the principle is utilized):

If no more than n items are put into m boxes, then at least one box must contain no more than n/m items.

Many solutions to these data management problems adopt the principle to develop filtering techniques. Although the principle has become a prevalent tool for such tasks, we observe an *inherent* drawback of these solutions which lies in the principle itself: the constraint is applied on the m boxes *individually*, as shown in the following example.

Example 1 *Suppose we have $m = 5$ boxes and search for the results such that the total number of items is no more than $n = 5$. By the pigeonhole principle, the constraint for filtering is: for every result, there exists a box which contains no more than $n/m = 1$ item. Such filter is easily passed if only a box fulfills this requirement. Let b_i denote the number of items in the i-th box [1]. Consider the two box layouts $(b_0, \ldots, b_{m-1})$ in Figure 1: $(2, 1, 2, 2, 1)$ and $(2, 0, 3, 1, 2)$. Both have a total of $8 > 5$ items, but pass the filter as both have at least one $b_i \leq 1$.*

This example showcases that the constraint by the pigeonhole principle is *weak*, rendering the filtering power very limited.

In this paper, we seek stronger constraints by aggregated conditions on *multiple* boxes. By placing the m boxes $b_0, \ldots, b_{m-1}$ (without loss of generality,) clockwise in a *ring* where b_0 is next to b_{m-1}, and going clockwise on the ring, we observe:

If no more than n items are put into m boxes, then for every length l in $[1 \,..\, m]$, there exist l consecutive boxes which contain a total of no more than $l \cdot n/m$ items.

We call it the basic form of the *pigeonring principle*. Consider the above example. For every result, there must be two consecutive boxes which contain a total of no more than $2n/m = 2$ items, three consecutive boxes which contain a total of no more than $3n/m = 3$ items, and so on. For the layout $(2, 1, 2, 2, 1)$ which passes the pigeonhole principle-based filter, when $l = 2$, we have $b_0 + b_1 = 3$, $b_1 + b_2 = 3$, $b_2 + b_3 = 4$, $b_3 + b_4 = 3$, and $b_4 + b_0 = 3$. Since there are no two consecutive boxes with a sum of ≤ 2 items, it is filtered.

On the basis of the basic form of the pigeonring principle, we discover its strong form:

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 12, No. 1
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3275536.3275539

[1] For ease of computing modulo operation, the subscript i starts with 0 in this paper, unless otherwise specified. In addition, we let subscript $i = i \bmod m$ whenever $i \geq m$.

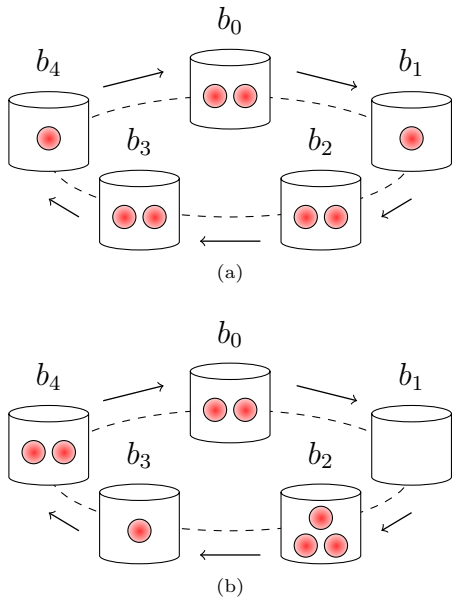


Figure 1: Pigeonring Principle ($n = 5$, $m = 5$)

If no more than n items are put into m boxes, then there exists at least one box such that for every $l \in [1\,..\,m]$, starting from this box and going clockwise, the l consecutive boxes contain a total of no more than $l \cdot n/m$ items.

In short, there exists $i \in [0\,..\,m-1]$, such that $b_i \leq n/m$, $b_i + b_{i+1} \leq 2n/m$, $b_i + b_{i+1} + b_{i+2} \leq 3n/m$, ... For the two layouts $(2, 1, 2, 2, 1)$ and $(2, 0, 3, 1, 2)$, when $l = 2$, since we cannot find any i such that $b_i \leq 1$ and $b_i + b_{i+1} \leq 2$, both are filtered. Despite being exemplified by real life objects, the new principle also holds when n is a real number. To the best of our knowledge, we are the first to discover this property.

To utilize the pigeonring principle, we focus on the problems which have the following form: f is a function that maps a pair of objects to a real number. Given a query object q, find all objects x in a database such that $f(x, q)$ is not greater (or not smaller) than a threshold τ. We call it a τ-selection problem. It covers many problems, especially for various similarity searches to cope with specific data types and similarity measures. These problems are important for numerous applications, including search and retrieval tasks, data cleaning, data integration, etc. The naïve algorithm for a τ-selection problem needs to access every object in the database, and thus cannot scale well to large datasets. For the sake of efficiency, many exact solutions [57, 132, 65, 73, 7, 30, 55, 107, 72, 28, 137] to τ-selection problems adopt the filter-and-refine strategy, and utilize the **pigeonhole principle** to find a set of candidates that satisfy necessary condition of the $f(x, q)$ constraint. Since computing $f(x, q)$ for the candidates is usually expensive, the efficiency critically depends on the filtering power measured by the number of candidates. Based on the **pigeonring principle**, stronger filtering conditions can be developed to fundamentally reduce the candidate number.

We analyze the filtering power of the pigeonring principle and show the candidates it produces are guaranteed to be a *subset* of those produced by the pigeonhole principle. It is easy to see that the pigeonring principle contains the pigeonhole principle as a special case when $l = 1$. Thus, *all* the pigeonhole principle-based methods are possible to be accelerated by the pigeonring principle. We also discuss the case when variable threshold allocation and integer reduction, two important techniques for τ-selection problems, are present, so that they can be seemlessly integrated into our principle.

We describe a *universal* filtering framework which applies to all pigeonring (and of course, pigeonhole) principle-based methods for τ-selection problems. We answer two questions: on what condition a filtering instance is complete and on what condition a filtering instance is tight. Although existing studies have developed complete and tight filtering methods for specific τ-selection problems, the two questions are yet to be answered from a general perspective. Case studies are shown for several common τ-selection problems. Moreover, we discuss the indexing and candidate generation techniques for the pigeonring principle. It only requires minor modifications on top of the existing pigeonhole principle-based methods.

To show the applicability of the new principle and the efficiency of the resulting algorithms, we conduct experiments on four τ-selection problems which cover a variety of data types and applications. The results on real datasets show that by simply applying the new principle on the existing pigeonhole principle-based methods, the search can be significantly accelerated (e.g., 15 times for Hamming distance search).

Since the pigeonring principle holds as a free extension of the pigeonhole principle, we believe that the applications of the pigeonring principle are *far beyond* the scope of τ-selection problems. We leave them as future work.

Our contributions are summarized as:

- We develop the pigeonring principle which exploits conditions on multiple boxes and hence yields inherently stronger constraints than the pigeonhole principle does. The new principle can be utilized to solve τ-selection problems efficiently as filtering conditions.
- We propose a universal filtering framework which encapsulates all the pigeonring (pigeonhole) principle-based solutions to τ-selection problems.
- We explain how to quickly find the candidates satisfying the filtering condition by the pigeonring principle with easy modifications on existing algorithms.
- We perform extensive experiments on real datasets. The results demonstrate the applicability of the pigeonring principle and the efficiency of the algorithms equipped with the pigeonring principle-based filtering.

The rest of the paper is organized as follows: Section 2 introduces the pigeonhole principle and the τ-selection problem. Section 3 presents the pigeonring principle. Section 4 describes the integration of variable threshold allocation and integer reduction to pigeonring principle. Section 5 introduces the filtering framework. Section 6 shows case studies for several τ-selection problems. Indexing and candidate generation techniques as well as cost analysis are presented in Section 7. Experimental results are reported in Section 8. Section 9 surveys related work. Section 10 concludes the paper.

2. PRELIMINARIES

Table 1 lists the frequently used notations in this paper.

2.1 Pigeonhole Principle

The simple form of the pigeonhole principle states that if $(n + 1)$ items are put into n boxes, then at least one box has two or more of the items. By generalizing to real numbers, the principle is formally stated as follows.

Table 1: Frequently Used Notations

Sym.	Description	Sym.	Description
x, y	Object	q	Query object
$\mathcal{O}$	Object universe	X	Dataset of objects
f	Selection function	τ	Selection threshold
m	#boxes	n	(bound of) #items
b_i	#items in i-th box	t_i	Threshold of b_i
B	Sequence of b_i	T	Sequence of t_i
$\|\cdot\|_1$	Sum of elements	l	Chain length
c_i^l	Chain of length l, starting from b_i		
C_B	Set of chains on B	d	Dimensionality
F	Featuring function	D	Bounding function

Theorem 1 (Pigeonhole Principle [13]) *Let $b_0, \ldots, b_{m-1}$ be m real numbers. If $b_0 + b_1 + \ldots + b_{m-1} \leq n$, then there exists at least one $b_i, i \in [0\,..\,m-1]$, such that $b_i \leq n/m$.*

2.2 τ-selection Problems

The pigeonhole principle has been utilized to solve many data management problems. Particularly, it is often used on the problems of finding objects in a database whose similarities or distances to a query object are constrained by a threshold. These problems can be generalized by the following form.

Problem 1 (τ-selection Problem) *Let $\mathcal{O}$ denote an object universe. x and y are two objects in $\mathcal{O}$. $f : \mathcal{O} \times \mathcal{O} \to \mathbb{R}$ is a function which evaluates a pair of objects. Given a collection of data objects $X \subseteq \mathcal{O}$, a query object $q \in \mathcal{O}$, and a threshold τ, the goal is to find all data objects $x \in X$ such that $f(x, q) \leq \tau$.*

We call f a selection function. It usually captures the similarity or distance between a pairs of objects. Since all these problems involve a threshold τ, we call them τ-selection problems. "$\leq$" can be replaced by "$\geq$", "$<$", or "$>$" for specific problems. Without loss of generality, we use "$\leq$" in this paper. The extension to support the other three cases is straightforward. Next we show a few examples of τ-selection problems.

Problem 2 (Hamming Distance Search) *Given a collection of d-dimensional binary vectors X, a query vector q, find all $x \in X$ such that $H(x, q) \leq \tau$. $H(\cdot, \cdot)$ measures the Hamming distance between two binary vectors: $H(x, y) = \sum_{i=0}^{d-1} \Delta(x[i], y[i])$. $x[i]$ denotes the value of the i-th dimension of x. $\Delta(x[i], y[i]) = 0$, if $x[i] = y[i]$; or 1, otherwise.*

Problem 3 (Set Similarity Search) *An object is a set of tokens drawn from a finite universe $\mathcal{U}$. Given a collection of objects X, a query set q, find all $x \in X$ such that $sim(x, q) \geq \tau$. $sim(\cdot, \cdot)$ is a set similarity function, e.g., the overlap similarity $O(x, y) = |x \cap y|$.*

Problem 4 (String Edit Distance Search) *Given a collection of strings X, a query string q, find all $x \in X$ such that $ed(x, q) \leq \tau$. $ed(\cdot, \cdot)$ is the edit distance between two strings. It is the minimum number of operations (insertion, deletion, or substitution of a symbol) to transform a string to another.*

Problem 5 (Graph Edit Distance Search) *Given a collection of graphs X, a query graph q, find all $x \in X$ such that $ged(x, q) \leq \tau$. $ged(\cdot, \cdot)$ is the graph edit distance between two graphs. It is the minimum number of operations to transform one graph to another. The operations include: inserting an isolated labeled vertex, deleting an isolated vertex, changing the label of a vertex, inserting a labeled edge, deleting an edge, and changing the label of an edge.*

The above problems [2] collectively cover a variety of data types and applications such as image retrieval, near-duplicate detection, entity resolution, and structure search. For instance, in image retrieval, images are converted to binary vectors and the vectors whose Hamming distances to the query are within a threshold of 16 are identified for further image-level verification [131]. In entity resolution, the same entity may differ in spellings or formats, e.g., `al-Qaeda`, `al-Qaida`, and `al-Qa'ida`. A string similarity search with an edit distance threshold of 2 can capture these alternative spellings [108].

Computing $f(x, q)$ for every data and query object is prohibitive for large datasets. To avoid this, many exact solutions [3] to τ-selection problems are based on the filter-and-refine strategy to generate a set of candidates. They first extract a bag of *features* from each object, e.g., a partition for Hamming distance search [57, 132, 65, 73], q-grams for string edit distance search [72, 28, 53, 55, 107], trees, paths, or a partition for graph edit distance search [128, 101, 109, 136, 138, 56, 137]. The intuition is that if two objects are similar, there must be a pair of similar or identical features from the two objects. By the pigeonhole principle, the constraint $f(x, q) \leq \tau$ is thus converted to a necessary condition on pairs of features, called *filtering condition*. The data objects that satisfy this condition are called *candidates*. It is much more efficient to check whether a pair of features satisfies the filtering condition than to compute $f(x, q)$; and with the help of an index, one may quickly identify all the candidates. They are eventually verified by comparing $f(x, q)$ with τ. Since computing $f(x, q)$ for the candidates is time-consuming, the search performance depends heavily on the candidate number.

Example 2 *Consider an instance of Hamming distance search. $d = 10$. $\tau = 5$. Table 2 shows four data objects and a query object. They are vertically partitioned into 5 equi-width disjoint parts. Let x_i denote the i-th part of x. Because the parts are disjoint, the sum of distances in the five parts $\sum_{i=0}^{4} H(x_i, q_i) = H(x, q)$. Let each box b_i represent a part. By Theorem 1, if $H(x, q) \leq \tau$, there exists at least one box such that $H(x_i, q_i) \leq \tau/5 = 1$. This becomes the filtering condition. x^1, x^2, and x^3 are candidates because $H(x_1^1, q_1) = H(\mathsf{11}, \mathsf{10}) = 1$, $H(x_0^2, q_0) = H(\mathsf{00}, \mathsf{00}) = 0$, and $H(x_0^3, q_0) = H(\mathsf{01}, \mathsf{00}) = 1$ [4]. $H(x^1, q) = 8$. $H(x^2, q) = 5$. $H(x^3, q) = 7$. Only x^2 is a result.*

Table 2: Hamming Distance Search

	b_0	b_1	b_2	b_3	b_4
$x^1 =$	11	11	10	11	10
$x^2 =$	00	01	01	11	10
$x^3 =$	01	01	10	01	10
$x^4 =$	11	01	10	11	00
$q =$	00	10	01	00	11

[2] Another common τ-selection problem is L^p distance search. However, the pigeonhole principle is hardly adopted by prevalent methods for L^p distance search ($p > 0$). For this reason, we choose not to speed up L^p distance search in this paper.
[3] In this paper, we focus on exact solutions and single-core, in-memory, and stand-alone implementations of algorithms.
[4] Despite other parts satisfying the condition for the three data objects, they are not reported here since the objects have already become candidates by checking the first two parts.

3. PIGEONRING PRINCIPLE

In the pigeonhole principle, the threshold of a box can be regarded as a quota. To generate candidates, only individual boxes are considered. Even if $f(x,q)$ exceeds τ by a large margin, a data object becomes a candidate if only it has a box within the quota. E.g., consider x^1 in Example 2. The distances in the five boxes are $(2,1,2,2,1)$. b_1 and b_4 satisfy the filtering condition. x^1 becomes a candidate, but $f(x^1,q)=8>\tau$. This case is common for real datasets, and consequently the filtering power is rather weak. To address this issue, our idea is to examine multiple boxes and compare the accumulated distance with the quota.

Example 3 *For x^1 in Example 2, we organize the boxes in a ring in which b_0 succeeds b_4, as shown in Figure 1(a), where a ball indicates a Hamming distance of 1. Now we find candidates by checking every two adjacent boxes: b_0b_1, b_1b_2, b_2b_3, b_3b_4, and b_4b_5, each with a quota of $2\cdot\tau/m=2$. Since the m parts are disjoint, we can sum up the distances in individual boxes to obtain the distances in multiple boxes, which are 3, 3, 4, 3, and 3, respectively. Since all of them exceed the quota, x^1 is filtered.*

The idea in Example 3 can be extended to combinations of any size, which becomes the intuition of our pigeonring principle. We investigate in the following direction: if the sum of m numbers is *bounded* by a value, what is the property for the sum of a subset of these numbers? E.g., in Example 3, there must be two consecutive boxes whose sum of distances does not exceed $2\cdot f(x,q)/m$, thus $2\tau/m$ for every result.

Let B be a sequence of m real numbers $(b_0,\ \ldots,\ b_{m-1})$. Each b_i is called a *box* (for brevity, we abuse the term to denote the number of items in it). Let $\|B\|_1$ denote the sum of all elements in B; i.e., $\|B\|_1=\sum_{i=0}^{m-1}b_i$. We place the boxes in a *ring*, in which b_{m-1} is adjacent to b_0. Let a chain c_i^l be a sequence of l consecutive boxes starting from b_i:

$$c_i^l=(b_i,\ldots,b_{i+l-1}).$$

Recall that we let subscript $i=i \bmod m$, if $i\geq m$. $\|c_i^l\|_1$ denotes the sum of elements in c_i^l; i.e., $\|c_i^l\|_1=\sum_{j=i}^{i+l-1}b_i$.

When $l=1$, c_i^l contains a single element b_i. $\forall l'\in[1\,..\,l]$, $c_i^{l'}$ is an l'-*prefix* of c_i^l, and $c_{i+l-l'}^{l'}$ is an l'-*suffix* of c_i^l. $c_j^{l'}$ is a *subchain* of c_i^l if $j\geq i$ and $j+l'\leq i+l$. c_i^m is called a *complete chain* because every box in B appears exactly once in c_i^m. $\|c_i^m\|_1=\|B\|_1$. We restrict the length of a chain to not exceeding m. Let C_B be the set of all chains based on B, i.e., $C_B=\{\,c_i^l\mid i\in[0\,..\,m-1], l\in[1\,..\,m]\,\}$.

Example 4 *Consider Figure 1(a). $c_3^4=(b_3,b_4,b_0,b_1)$. $\|c_3^4\|_1=2+1+2+1=6$. c_3^2 is a 2-prefix of c_3^4. c_4^3 is a 3-suffix of c_3^4. c_4^2 is a subchain of c_3^4. c_3^5 is a complete chain.*

Theorem 2 (Pigeonring Principle – Basic Form) [5] *B is a sequence of m real numbers. If $\|B\|_1\leq n$, then $\forall l\in[1\,..\,m]$, there exists at least one chain $c_i^l\in C_B$ such that $\|c_i^l\|_1\leq l\cdot n/m$.*

To utilize the pigeonring principle for τ-selection problems, we may regard each box as the output of a function taking x and q as input, e.g., $b_i(x,q)=H(x_i,q_i)$ for Hamming distance search, so that $\|B(x,q)\|_1=f(x,q)$ is guaranteed [6]. Then for a result object x, because $f(x,q)\leq\tau$, by Theorem 2, we can always find a chain $c_i^l\in C_{B(x,q)}$ such that $\|c_i^l\|_1\leq l\cdot\tau/m$. As a result, a data object becomes a candidate only if it meets this condition. It is noteworthy to mention that when $l=1$, the pigeonring principle becomes exactly the pigeonhole principle. As a result, the pigeonhole principle is a special case of the pigeonring principle. Since a candidate produced by the pigeonring principle must have a chain satisfying $\|c_i^l\|_1\leq l\cdot\tau/m$, by Theorem 1, there exists at least one box in c_i^l such that its value is less than or equal to τ/m. This implies:

Lemma 1 *Given X, q, τ, and $B(x,q)$, the candidates produced by Theorem 2 are a subset of those produced by Theorem 1.*

One may notice that when $\|B(x,q)\|_1=f(x,q)$ and $l=m$, all the candidates are exactly the results, meaning that in this case the candidate generation subsumes the verification.

Example 5 *Consider Example 2. For the four data objects, the values of boxes $b_0,\ldots,b_{m-1}$ are:*

$$B(x^1,q)=(2,1,2,2,1).$$
$$B(x^2,q)=(0,2,0,2,1).$$
$$B(x^3,q)=(1,2,2,1,1).$$
$$B(x^4,q)=(2,2,2,2,2).$$

Since the five parts are disjoint, $\|B(x,q)\|_1=f(x,q)$. We use the above method to generate candidates. Suppose $l=2$. We represent in a sequence the $\|c_i^l\|_1$ values for $i=0,\ldots,m-1$. $(\|c_i^l(x^1,q)\|_1)=(3,3,4,3,3)$. $(\|c_i^l(x^2,q)\|_1)=(2,2,2,3,1)$. $(\|c_i^l(x^3,q)\|_1)=(3,4,3,2,2)$. $(\|c_i^l(x^4,q)\|_1)=(4,4,4,4,4)$. Since objects x^2 and x^3 have at least one chain whose value is within $l\cdot\tau/m$, they become candidates. x^1 and x^4 are filtered.

Next we will see the condition stated by Theorem 2 can be further strengthened. A chain c_i^l is called *viable* if it satisfies the condition in Theorem 2: $\|c_i^l\|_1\leq l\cdot n/m$. Otherwise, it is called *non-viable*. We may also call a box viable or non-viable since it can be regarded as a chain of length 1. Given a viable chain c_i^l, if all of its prefixes are also viable, i.e., $\forall l'\in[1\,..\,l]$, $\|c_i^{l'}\|_1\leq l'\cdot n/m$, then c_i^l is called *prefix-viable*.

Theorem 3 (Pigeonring Principle – Strong Form) *B is a sequence of m real numbers. If $\|B\|_1\leq n$, then $\forall l\in[1\,..\,m]$, there exists at least one prefix-viable chain $c_i^l\in C_B$.*

By the strong form of the pigeonring principle, a stronger filtering condition is delivered: Assume $\|B(x,q)\|_1=f(x,q)$. A candidate object must have a chain such that each of the chain's prefixes c_i^l satisfies $\|c_i^l\|_1\leq l\cdot\tau/m$.

Lemma 2 *Given X, q, τ, and $B(x,q)$, the candidates produced by Theorem 3 are a subset of those produced by Theorem 2.*

Example 6 *Assume $\tau=5$, $m=5$, and $B=(2,0,3,1,2)$ for a data object. When $l=2$, $(\|c_i^l\|_1)=(2,3,4,3,4)$. c_0^l is the only chain of length 2 satisfying $\|c_i^l\|_1\leq l\cdot\tau/m$. This object is not filtered by the basic form of the pigeonring principle. However, its 1-prefix $\|c_0^1\|_1>1\cdot\tau/m$. By the strong form of the pigeonring principle, this object is filtered.*

In the rest of the paper, when context is clear, we mean the strong form when the pigeonring principle is mentioned.

Because we can go either clockwise or counterclockwise on the ring to collect chains, "prefix" can be replaced with

[5] Please refer to the full version of this paper [74] for the proofs of the theorems and the lemmata in this paper.

[6] We assume this setting throughout this section. The general case will be discussed in Section 5.

"suffix" in Theorem 3, and the principle still holds. We may also replace "$\leq$" with "$>$" and prove the principle for the non-viable case. We call a chain whose suffixes are all viable a suffix-viable chain. If a chain's prefixes/suffixes are all non-viable, we say it is a prefix/suffix-non-viable chain. The following corollaries are obtained.

Corollary 1 *B is a sequence of m real numbers. If $\|B\|_1 \leq n$, then $\forall l \in [1\,..\,m]$, there exist at least one prefix-viable chain $c_i^l \in C_B$ and at least one suffix-viable chain $c_{i'}^l \in C_B$. If $\|B\|_1 > n$, then $\forall l \in [1\,..\,m]$, there exist at least one prefix-non-viable chain $c_i^l \in C_B$ and at least one suffix-non-viable chain $c_{i'}^l \in C_B$.*

Corollary 2 *Consider four types of chains: prefix-viable, suffix-viable, prefix-non-viable, and suffix-non-viable. If two contiguous chains c_i^l and $c_{i+l-1}^{l'}$ are the same type, then $c_i^{l+l'}$ is also this type.*

3.1 Filtering Performance Analysis

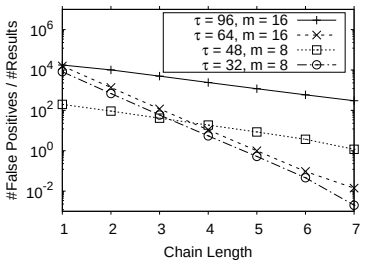
We first analyze the probability that a data object is a candidate for a chain length l (denoted by $\Pr(CAND_l)$), and then estimate the ratio of false positive number and result number in the candidate set of the pigeonring principle-based filtering. We assume $\|B(x,q)\|_1 = f(x,q)$. All the m boxes are assumed independent random variables in $(-\infty, +\infty)$, having the same probability density function (PDF) [7], denoted by p. Let $n = \tau$. Then a viable chain c_i^l must satisfy $\|c_i^l\|_1 \leq l \cdot \tau/m$. Our idea is to construct by recurrence all the rings in which there is no prefix-viable chain of length $l \in [1\,..\,m]$, hence to obtain $1 - \Pr(CAND_l)$. By the pigeonring principle, for such rings, $\|B\|_1 > \tau$. By Corollary 1, there exists at least one suffix-non-viable chain of length m. Although c_0^m might not be a suffix-non-viable chain, we will discuss this scenario later, and assume c_0^m is suffix-non-viable first. Obviously, c_0^m does not have any prefix-viable chain of length $l \in [1\,..\,m]$ as its subchain. We call such suffix-non-viable chain a target chain.

A target chain can be constructed by concatenation of chains drawn from a set. E.g., when $m = 3$ and $l = 2$, there are only three cases for the boxes in a target chain: `NNN`, `NVN`, and `VNN`. `V` and `N` denote viable and non-viable boxes, respectively. Thus, it can be constructed by concatenating chains in $\{\,\texttt{N}, \texttt{VN}\,\}$, where `VN` is non-viable. We call a set of chains a *word set* if concatenating any number of chains in it always yields a suffix-non-viable chain. Each chain in it is called a *word*.

Because a target chain contains no prefix-viable chain of length l, we consider the word set W which consists of (1) non-viable chain of length 1, and (2) suffix-non-viable chains of length l', $l' \in [2\,..\,l]$, whose $(l'-1)$-prefixes are prefix-viable. The set $\{\,\texttt{N}, \texttt{VN}\,\}$ (`VN` is non-viable) in the above example is such kind of word set when $l = 2$. Given a word $w_i \in W$ whose length is $|w_i|$, let $\Pr(w_i)$ denote the probability that a chain of length $|w_i|$ is w_i. Consider a chain c constructed by concatenation of words $w_0, \ldots, w_k \in W$. The probability that a chain of length $|w_0| + |w_1| + \ldots + |w_k|$ is c is the product of the words' probabilities: $\prod_{i=0}^{k} \Pr(w_i)$. Let $M(x)$ be the probability that a chain of length x is a target chain.

$$M(x) = \begin{cases} 1 & , \text{ if } x = 0; \\ \sum_{i=1}^{\min(x,l)} M(x-i)\Pr(w^i) & , \text{ if } x > 0. \end{cases}$$

[7] If the thresholds or PDFs of boxes differ or dependency exists, e.g., by joint PDFs, $\Pr(CAND_l)$ can be computed by dynamic programming, extending the method in this subsection.



(a) Hamming Distance Search ($d = 256$)

Figure 2: Filtering Performance Analysis

w^i denotes a word in W whose length is i. The probability that a chain of length i is w^i is computed as follows.

$$\text{When } i = 1, \Pr(w^i) = \int_{\tau/m}^{+\infty} p(x)dx.$$

$$\text{When } i = 2, \Pr(w^i) = \int_{-\infty}^{\tau/m} p(x)dx \int_{2\tau/m-x}^{+\infty} p(y)dy.$$

$$\text{When } i > 2, \Pr(w^i) = \int_{-\infty}^{\tau/m} p(x_0)dx_0 \int_{-\infty}^{2\tau/m-x_0} p(x_1)dx_1 \ldots$$
$$\int_{-\infty}^{i\tau/m-\sum_{j=0}^{i-2} x_j} p(x_{i-1})dx_{i-1} \int_{(i+1)\tau/m-\sum_{j=0}^{i-1} x_j}^{+\infty} p(y)dy.$$

Let $N(x)$ be the probability that there is no prefix-viable chain of length l in a ring of x boxes, i.e., $1 - \Pr(CAND_l)$. Since we assume c_0^m is the target chain, b_{m-1} always ends with the last box of a word in W. To compute $N(x)$, we also need to consider the case when b_{m-1} ends with the other positions in a word. This can be done by shifting the starting position of a target chain of length $(x - l')$ for every $l' \in [2\,..\,l]$, to $b_1, \ldots, b_{l-1}$, and then append a word of length l' in W. Thus,

$$N(x) = \begin{cases} M(x) & , \text{ if x = 1;} \\ M(x) + \sum_{i=2}^{\min(x,l)} M(x-i)(i-1)\Pr(w^i), & \text{if } x > 1. \end{cases}$$

Then $\Pr(CAND_l) = 1 - N(m)$.

Next we analyze the expected ratio of false positive number and result number in a candidate set. The probability that an object is a result is

$$\Pr(RES) = \int_{-\infty}^{+\infty} p(x_0)dx_0 \ldots$$
$$\int_{-\infty}^{+\infty} p(x_{m-2})dx_{m-2} \int_{-\infty}^{\tau-\sum_{i=0}^{m-2} x_i} dx_{m-1}.$$

The ratio is $\Pr(CAND_l)/\Pr(RES)$. For the assumption $\|B(x,q)\|_1 = f(x,q)$, when $l = m$, $\Pr(RES) = \Pr(CAND_l)$.

Based on the analysis, we plot in Figure 2 the ratio of false positive number and result number for Hamming distance search on a synthetic dataset with uniform distribution (please see Section 8.2 for results on real datasets). It can be observed that the estimated ratio keeps decreasing with the growth of chain length l. The ratio is smaller than 1 for some parameter settings, meaning most candidates are results.

4. VARIABLE THRESHOLD ALLOCATION AND INTEGER REDUCTION

The pigeonhole principle has many variants. We discuss two variants that have been utilized to solve τ-selection problems: variable threshold allocation and integer reduction.

Instead of using a fixed threshold n/m, we may assign different thresholds for $b_0, \ldots, b_{m-1}$.

Theorem 4 (Pigeonhole Principle - Variable Threshold Allocation [73]) *Given two sequences of real numbers: $(b_0, \ldots, b_{m-1})$ and $(t_0, \ldots, t_{m-1})$. If $b_0 + b_1 + \ldots + b_{m-1} \leq n$ and $t_0 + t_1 + \ldots + t_{m-1} = n$, then there exists at least one $b_i, i \in [0\,..\,m-1]$, such that $b_i \leq t_i$.*

If $b_0, \ldots, b_{m-1}$ are limited to integers, the thresholds do not have to sum up to n, but $n - m + 1$, as stated below.

Theorem 5 (Pigeonhole Principle - Integer Reduction [73]) *Given two sequences of integers $(b_0, \ldots, b_{m-1})$ and $(t_0, \ldots, t_{m-1})$. If $b_0 + b_1 + \ldots + b_{m-1} \leq n$ and $t_0 + t_1 + \ldots + t_{m-1} = n - m + 1$, then there exists at least one $b_i, i \in [0\,..\,m-1]$, such that $b_i \leq t_i$.*

To see this is correct, assume $b_i > t_i$ for every $i \in [0\,..\,m-1]$. Then $b_0 + b_1 + \ldots + b_{m-1} = \sum_{i=0}^{m-1}(t_i + 1) = n - m + 1 + m = n + 1$. It contradicts $b_0 + b_1 + \ldots + b_{m-1} \leq n$.

The pigeonring principle applies to these variants as well. Let $T = (t_0, \ldots, t_{m-1})$. We have the following theorem.

Theorem 6 (Pigeonring Principle - Variable Threshold Allocation) *Consider two sequences of real numbers B and T. If $\|B\|_1 \leq n$ and $\|T\|_1 = n$, then $\forall l \in [1\,..\,m]$, there exists at least one $c_i^l \in C_B$ such that each of its prefixes $c_i^{l'}$ satisfies $\|c_i^{l'}\|_1 \leq \sum_{j=i}^{j+l'-1} t_j$.*

Assume $\|B(x,q)\|_1 = f(x,q)$. We can distribute the threshold τ with a sequence T such that $\|T\|_1 = \tau$. By Theorem 6, a data object becomes a candidate only if it yields a chain such that each of its prefixes c_i^l satisfies $\|c_i^l\|_1 \leq \sum_{j=i}^{j+l-1} t_j$.

Example 7 *Consider x^1 in Example 5. Suppose $T = [1, 2, 0, 1, 1]$. $\|T\|_1 = 5 = \tau$. When $l = 2$, $\|c_0^l\|_1 = 3 \leq t_0 + t_1$. It is the only chain of length 2 satisfying $\|c_i^l\|_1 \leq \sum_{j=i}^{j+l-1} t_j$. However, its 1-prefix $\|c_0^1\|_1 = 2 > t_0$. x^1 is filtered.*

It can be seen that Theorem 3 is a special case of Theorem 6 when $t_i = n/m$ for every $i \in [0\,..\,m-1]$. If we regard the boxes in B as variables, then with an assumption on these variables, the condition of T in Theorem 6, $\|T\|_1 = n$, is tight:

Lemma 3 *Assume the m boxes in B are independent variables and $\forall i \in [0\,..\,m-1]$, the range of b_i is an interval $[u_i, v_i]$. If $\sum_0^{m-1} u_i \leq n \leq \sum_0^{m-1} v_i$, then $\nexists T = (t_0, \ldots, t_{m-1})$ such that (1) $\|T\|_1 < n$; and (2) $\forall \|B\|_1 \leq n$ and $l \in [1\,..\,m]$, there exists at least one $c_i^l \in C_B$ such that each of its prefixes $c_i^{l'}$ satisfies $\|c_i^{l'}\|_1 \leq \sum_{j=i}^{j+l'-1} t_j$.*

Intuitively, this lemma means that when the m boxes are independent and every box is a real number in a continuous range, if we use Theorem 6 for filtering, the thresholds of boxes cannot be reduced while we are still guaranteed to find all the results, which is necessary for an exact algorithm.

If the m boxes are limited to integers, we may use integer reduction to reduce the thresholds, like in Theorem 5.

Theorem 7 (Pigeonring Principle - Integer Reduction) *Consider two sequences of integers B and T. If $\|B\|_1 \leq n$ and $\|T\|_1 = n - m + 1$, then $\forall l \in [1\,..\,m]$, there exists at least one $c_i^l \in C_B$ such that that each of its prefixes $c_i^{l'}$ satisfies $\|c_i^{l'}\|_1 \leq l' - 1 + \sum_{j=i}^{j+l'-1} t_j$.*

This theorem suggests that if $f(x,q)$ and τ are limited to integers, we may distribute τ with a sequence of m integers $T = (t_0, \ldots, t_{m-1})$ such that $\|T\|_1 = \tau - m + 1$. Assume $\|B(x,q)\|_1 = f(x,q)$. A data object becomes a candidate only if it yields a chain such that each of its prefixes c_i^l satisfies $\|c_i^l\|_1 \leq l - 1 + \sum_{j=i}^{j+l-1} t_j$.

Example 8 *Consider x^3 in Example 5. Suppose $T = (1, 0, 0, 0, 0)$. $\|T\|_1 = 1 = \tau - m + 1$. When $l = 2$, $\|c_4^l\|_1 = 2 \leq l - 1 + t_4 + t_0$. It is the only chain of length 2 satisfying $\|c_i^l\|_1 \leq l - 1 + \sum_{j=i}^{j+l-1} t_j$. However, its 1-prefix $\|c_4^1\|_1 = 1 > 1 - 1 + t_4$. x^3 is filtered.*

If the m boxes are independent and the range of every box is an integer interval, then the condition of T in Theorem 7, $\|T\|_1 = n - m + 1$, is tight.

We may replace "$\leq$" with "$\geq$" in Theorem 6 and the theorem still holds. If we use "$\geq$" instead of "$\leq$" in Theorem 7, we need to replace "$n - m + 1$" with "$n + m - 1$" and "$l' - 1 + \sum_{j=i}^{j+l'-1} t_j$" with "$1 - l' + \sum_{j=i}^{j+l'-1} t_j$" to make the theorem hold.

5. FILTERING FRAMEWORK

Based on the pigeonring principle, we describe a universal filtering framework for τ-selection problems. Although this framework has been materialized as many (pigeonhole principle-based) solutions to τ-selection problems, it is yet to be formulated generally. By this framework, we may decide the completeness and the tightness of any pigeonring principle-based filtering instance from a general perspective.

The pigeonring principle-based filtering in essence leverages the relation between f and the sum of a set of functions' outputs. It consists of three components: extract, box, and bound. The extract component draws a bag of features from an object, such as projections, histograms, and substrings. A common method is to partition an object, and each part is regarded as a feature. To be more general, features are not necessarily disjoint, nor their union has to be an entire object. The box component distributes the bag of features into m subbags (overlap may exist), and then returns m values for m pairs of subbags, one from a data object and the other from a query object. The bound component bounds the sum of the m values returned by the box component. Next we define the framework and the components formally.

A (pigeonring principle-based) filtering instance is a triplet $\langle F, B, D \rangle$ composed of a featuring function F, a sequence of boxes B, and a bounding function D.

The feature extraction is implemented by a function F which maps an object to a bag of features: $F(x) = \{ x_0, x_1, \ldots \}$. In general, we use the same feature extraction as state-of-the-art pigeonhole principle-based methods do.

Each box $b_i(x,q)$ is a function which selects subbags of features from $F(x)$ and $F(q)$ and returns a real number. The design of b_i depends on the problem and the extracted features. In general, it captures the similarity or distance of features, or tells if features match or not. E.g., for Hamming distance search, a box returns the Hamming distance between a data and a query object over a part. Let $B(x,q)$ be a sequence of m boxes: $B(x,q) = (b_0(x,q), \ldots, b_{m-1}(x,q))$. We construct a ring on $B(x,q)$ and collect a set of chains $C_{B(x,q)}$.

D is a function which maps a threshold τ to a real number. The most common case is an identity function $D(\tau) = \tau$, e.g., for Hamming distance search. In other cases, especially when lower bounding techniques are used, $D(\tau)$ may be other values,

e.g., 2τ for the content-based filter of string edit distance search [115]. The filtering instance works on condition that $\|B(x,q)\|_1$ be bounded by $D(\tau)$ for every result of the query; i.e., $\|B(x,q)\|_1 \leq D(\tau)$.

By regarding $D(\tau)$ as n, we may use the pigeonring principle to establish a filtering condition on $C_{B(x,q)}$. The pigeonring principle guarantees that the set of candidates satisfying this condition is a superset of $\{\, x \mid \|B(x,q)\|_1 \leq D(\tau) \,\}$. When $l = m$, the candidates are exactly $\{\, x \mid \|B(x,q)\|_1 \leq D(\tau) \,\}$. To make the filtering instance work for the constraint $f(x,q) \leq \tau$, we define the *completeness* of a filtering instance. Let $\mathcal{R}$ denote the range of f, $\mathcal{R} \subseteq \mathbb{R}$. $\tau \in \mathcal{R}$.

Definition 1 *A filtering instance $\langle F, B, D \rangle$ is complete iff $\forall x, q \in \mathcal{O}$ and $\tau \in \mathcal{R}$, $\|B(x,q)\|_1 \leq D(\tau)$ is a necessary condition of $f(x,q) \leq \tau$.*

Intuitively, the completeness shows the condition on which we can safely use the pigeonring principle so that no result will be missed for any input. A sufficient and necessary condition of the completeness is stated below.

Lemma 4 *A filtering instance $\langle F, B, D \rangle$ is complete, iff (1) $\forall x, q \in \mathcal{O}$, $\|B(x,q)\|_1 \leq D(f(x,q))$, and (2) $\nexists x_1, q_1, x_2, q_2 \in \mathcal{O}$, such that $f(x_1,q_1) < f(x_2,q_2)$ and $\|B(x_1,q_1)\|_1 > D(f(x_2,q_2))$.*

We may judge if a filtering instance is complete with this lemma. Further, we can prove that if a filtering instance satisfies Condition 1 of Lemma 4 and D is monotonically increasing over $\mathcal{R}$, then it is complete.

Complete filtering instance exists for all τ-selection problems: $m = 1$, $b_0 = -1$, and $D(\tau) = 0$. However, it is trivial as all the data objects are candidates. Besides completeness, a filtering instance is also supposed to deliver good filtering power. In the ideal case, $f(x,q) \leq \tau$ is equivalent to $\|B(x,q)\|_1 \leq D(\tau)$, as captured by the definition of *tightness*:

Definition 2 *A filtering instance $\langle F, B, D \rangle$ is tight iff $\forall x, q \in \mathcal{O}$ and $\tau \in \mathcal{R}$, $\|B(x,q)\|_1 \leq D(\tau)$ is a necessary and sufficient condition of $f(x,q) \leq \tau$.*

Intuitively, the tightness shows that $\|B\|_1$ can be tightly bounded using f. It also implies the completeness, and we are guaranteed that when using pigeonring principle with $l = m$, the candidates are exactly the results. A sufficient and necessary condition of the tightness is stated below.

Lemma 5 *A filtering instance $\langle F, B, D \rangle$ is tight, iff (1) $\forall x, q \in \mathcal{O}$, $\|B(x,q)\|_1 \leq D(f(x,q))$, and (2) $\nexists x_1, q_1, x_2, q_2 \in \mathcal{O}$, such that $f(x_1,q_1) < f(x_2,q_2)$ and $D(f(x_1,q_1)) \geq \|B(x_2,q_2)\|_1$.*

We can also prove that if a filtering instance is tight, then $\|B(x,q)\|_1$ and $D(f(x,q))$ must be strictly increasing with respect to $f(x,q)$.

6. CASE STUDIES

In this section, we discuss how to utilize the pigeonring principle to improve the pigeonhole principle-based algorithms for τ-selection problems.

The general rules are discussed first. Although the search performance depends heavily on the candidate number, a small candidate number does not always lead to fast search speed because the filter itself also poses overhead. As we will see in Section 8.3, some methods reduce candidates by expensive operations, and eventually spend too much time on filtering. It is difficult to accurately estimate the search time, but in general, the filter should be light-weight. As a result, to apply the new principle on pigeonhole principle-based algorithms, it is crucial that we work out an efficient way to compute the value of each box in a chain in order to check if it is prefix-viable. Another key is to choose proper chain length l to strike a balance between filtering time and candidate number. This will be investigated empirically in Section 8.2.

Next we delve into the τ-selection problems listed in Section 2.2 and show how to leverage the filtering framework and the new principle to solve them. Since our methods are devised on top of existing pigeonhole principle-based methods, for each problem, we briefly review the existing method and describe our filtering instance. Since our methods are devised on top of existing pigeonhole principle-based methods, for each problem, we briefly review the existing method and describe our filtering instance [8]. We include a remark on implementation for efficient computation of the box values.

6.1 Hamming Distance Search

Existing algorithm. Our method is based on the GPH algorithm [73]. It divides d dimensions into m disjoint parts, and utilizes variable threshold allocation and integer reduction for filtering. The threshold of each part is computed by a cost model. By the pigeonhole principle, given a sequence of thresholds $T = (t_0, \ldots, t_{m-1})$ such that $\|T\|_1 = \tau - m + 1$, a candidate must have at least one part such that the Hamming distance to the query object over this part does not exceed t_i.

Filtering instance by the pigeonring principle.

- Extract: d dimensions are partitioned into m parts. Each part of an object x is a feature, denoted by x_i.
- Box: m is tunable. $b_i(x,q) = H(x_i, q_i)$.
- Bound: $D(\tau) = \tau$.

Because $\|B(x,q)\|_1 = f(x,q)$, by Lemma 5, the filtering instance is complete and tight. Theorem 7 is used for filtering.

Remark on implementation. To compute $b_i(x,q)$, we count the number of bits set to 1 in x_i bitwise XOR q_i. This can be done by a built-in popcount operation supported by most modern CPUs. We may also exploit the popcount to compute the sum of multiple boxes at a time.

6.2 Set Similarity Search

Suppose we use overlap similarity.

Existing algorithm. Our method is based on the pkwise algorithm [105]. Although it was developed for local similarity search to identify similar sliding windows represented by sets, it is also competitive on set similarity search (as shown in Section 8.3, it is even faster than the algorithms dedicated to set similarity queries). It extends the prefix filtering [19, 8]. The tokens in each object are sorted by a global order (e.g., increasing frequency). A k-wise signature is a combination of k tokens. By the pigeonhole principle, a candidate shares with a query object at least one k-wise signature in their first few tokens, called prefixes.

The token universe is partitioned into $(m-1)$ disjoint parts, each part called a class, numbered from 1 to $m-1$. Let the p-prefix/p-suffix of an object x be the first/last p tokens of x by the global order. If $|x \cap y| \geq \tau$, then $\exists k \in [1\,..\,m-1]$, such that the p_x-prefix of x and the p_y-prefix of y share at least k tokens

[8] Please see the full version of this paper [74] for running examples.

(a k-wise signature) in class k. The prefix length p_x is the smallest integer such that $\sum_{k=1}^{m-1} \max(0, cnt(x, p_x, k) - k + 1) = |x| - \tau + 1$. $cnt(x, p_x, k)$ is the number of class k tokens in the p_x-prefix of x. p_y is computed in the same way.

Filtering instance by the pigeonring principle.

- Extract: An object x is divided into p_x-prefix and $(|x| - p_x)$-suffix. The first two features are x and the $(|x| - p_x)$-suffix (denoted by x_0). Each of the other $(m-1)$ features (denoted by x_i) consists of class i tokens in the p_x-prefix.
- Box: m is tunable. $b_0(x, q) = |x_0 \cap q|$, if the last token of the p_x-prefix of x precedes the last token of the p_q-prefix of q in the global order; or $|x \cap q_0|$, otherwise. For $i \in [1 \,..\, m-1]$, $b_i(x, q) = |x_i \cap q_i|$.
- Bound: $D(\tau) = \tau$.

As b_0 computes the overlap in the suffix and the other boxes computes the overlap in the prefix, $\|B(x, q)\|_1$ is exactly the overlap of x and q; i.e., $\|B(x, q)\|_1 = f(x, q)$. By Lemma 5, the filtering instance is tight and complete. Based on the pkwise algorithm, we use variable threshold allocation and integer reduction: (1) (suffix) When $i = 0$, $t_i = |q| - p_q + 1$. (2) (prefix, if the number of tokens is adequate to create an i-wise signature) When $1 \le i \le m-1$, $t_i = i$, if $cnt(q, p_q, i) \ge i$; (3) (prefix, the other case) $t_i = cnt(q, p_q, i) + 1$, if $cnt(q, p_q, i) < i$. It can be proved $t_0 + t_1 + \ldots + t_{m-1} = \tau + m - 1$. Hence Theorem 7 (the $\ge$ case) is used for filtering. When $m = 2$ and $l = 1$, our method exactly becomes the prefix filtering.

Remark on implementation. For $i \in [1 \,..\, m-1]$, $b_i(x, q)$ is computed by set intersection. The computation of $b_0(x, q)$ is expensive. As a result, whenever we are to compute the value of $b_0(x, q)$, x becomes a candidate and the verification is invoked directly. In doing so, the filtering instance becomes not tight but gains in efficiency. Orthogonal filtering techniques such as length filter [8], position filter [116], and index-level skipping [111] are available to speed up the search process.

6.3 String Edit Distance Search

Existing algorithm. Our method is based on the Pivotal algorithm [28] which utilizes prefix filtering on q-grams. It sorts the q-grams of each string by a global order (e.g., increasing frequency) and picks the first $(\kappa\tau + 1)$ (can be reduced by location-based filter [115]) q-grams, called prefix. Let κ denote the q-gram length. Let P_x and P_q denote x's and q's prefixes, respectively. If $ed(x, q) \le \tau$, then:

- Pivotal prefix filter (based on the pigeonhole principle): if the last token in P_x precedes the last token in P_q in the global order, any $(\tau + 1)$ disjoint q-grams (called pivotal q-grams) in P_x must have at least one exact match in P_q; otherwise, any $(\tau + 1)$ pivotal q-grams in P_q must have at least one exact match in P_x.
- Alignment filter: the sum of the $(\tau + 1)$ minimum edit distances from each pivotal q-gram to a substring whose starting position differs by no more than τ in the other string must be within τ.

Filtering instance by the pigeonring principle.

- Extract: We sort the q-grams of each string by a global order and take the first $(\kappa\tau + 1)$ q-grams as features. Another feature is the whole string.
- Box: $m = \tau + 1$. If the last feature q-gram of x precedes the last feature q-gram of q in the global order, $b_i(x, q) = \min\{\, ed(x_i, q[u \,..\, v]) \mid 0 \le v - u \le \kappa + \tau - 1 \wedge u, v \in [\max(0, x_i.p - \tau) \,..\, \min(x_i.p + \kappa - 1 + \tau, |q| - 1)] \,\}$; otherwise, $b_i(x, q) = \min\{\, ed(q_i, x[u \,..\, v]) \mid 0 \le v - u \le \kappa + \tau - 1 \wedge u, v \in [\max(0, q_i.p - \tau) \,..\, \min(q_i.p + \kappa - 1 + \tau, |x| - 1)] \,\}$. x_i is the i-th pivotal q-gram of x. $x_i.p$ is its starting position in x. $x[u \,..\, v]$ is x's substring from positions u to v. The notations with respect to q are defined analogously.
- Bound: $D(\tau) = \tau$.

$\|B(x, q)\|_1$ is the sum of minimum edit distances in the alignment filter. As shown in [28], $\|B(x, q)\|_1 \le f(x, q)$. By Lemma 4, the filtering instance is complete. It is not tight due to violation of Condition 2 of Lemma 5. We use Theorem 3 for filtering. The first box of a prefix-viable chain must be zero, i.e., an exact match. By the pivotal prefix filter, for the first box of a chain, we only consider if it matches a q-gram in the prefix of the other side.

Remark on implementation. The alignment filter is essentially a special case ($l = m$) of the basic form of the pigeonring principle (Theorem 2). The time complexity of computing edit distance for a pivotal q-gram is $O(\kappa^2 + \kappa\tau)$. Seeing its expense, rather than computing the exact values of the edit distances for pivotal q-grams, we compute their lower bounds by content-based filter [115]: Given two strings x and y and a threshold t, $ed(x, y) \le t$ only if $H(h_x, h_y) \le 2t$. $H(\cdot, \cdot)$ is the Hamming distance. h_x (h_y) is a bit vector hashed from x (y): If x has a symbol σ, the corresponding bit is 1; otherwise, the bit is 0. In doing so, we may also limit the length of the substring $x[u \,..\, v]$ ($q[u \,..\, v]$) to κ and the completeness still holds. By a fast popcount algorithm with constant number of arithmetic operations, the complexity is reduced to $O(\kappa + \tau)$.

6.4 Graph Edit Distance Search

Existing algorithm. Our method is based on the Pars algorithm [137]. It divides each data graph into $(\tau + 1)$ disjoint subgraphs (may contain half-edges). By the pigeonhole principle, a candidate has at least one subgraph which is subgraph isomorphic (including half-edges) to the query graph.

Filtering instance by the pigeonring principle.

- Extract: We partition a graph to $(\tau + 1)$ disjoint subgraphs as features. Another feature is the whole graph.
- Box: $m = \tau + 1$. $b_i(x, q)$ is the minimum graph edit distance from a feature x_i to any subgraph of q: $b_i(x, q) = \min\{\, ged(x_i, q') \mid q' \sqsubseteq q \,\}$. $\sqsubseteq$ denotes a subgraph relation.
- Bound: $D(\tau) = \tau$.

As shown in [137], $\|B(x, q)\|_1 \le f(x, q)$. By Lemma 4, the filtering instance is complete. It is not tight due to violation of Condition 2 of Lemma 5. We use Theorem 3 for filtering. The first box of a prefix-viable chain must be zero, meaning x_i is a subgraph of q.

Remark on implementation. We check if $ged(x_i, q') \le t$ using its necessary condition by a subgraph isomorphism test to q from the deletion neighborhood [63, 108] of x_i by $\lfloor t \rfloor$ operations (deleting an edge or an isolated vertex, or changing a vertex label to wildcard), so as to circumvent the expensive subgraph enumeration and edit distance computation. $ged(x_i, q') \le t$ only if a subgraph isomorphism is found.

7. INDEXING, CANDIDATE GENERATION, AND COST ANALYSIS

Since a candidate always yields a prefix-viable chain of length l, to find candidates, we begin with searching for a viable single box (we call it the first step of candidate generation). This can be done efficiently with an index. By the case studies in Section 6, this step is the exactly same as the candidate

generation of existing methods [73, 105, 28, 137]. Hence we use the *same indexes* as these methods do to find viable single boxes. We refer readers to these studies for details.

With a viable single box, we check if the chains of lengths $2, \ldots, l$ starting from this box are all viable (we call it the second step of candidate generation). This can be done on the fly. An object is a candidate only if it passes this check. A speedup is that if the check of a chain c_i^l fails at some length, say l', i.e., $c_i^{l'}$ is not prefix-viable, then we do not need to check the chain starting from any position in $[i \,..\, i + l' - 1]$, because by Corollary 2, none of them is prefix-viable.

To find candidates by the pigeonring principle, only minor modifications are needed for the second step. E.g., for Hamming distance search we only need to add a few bit operations. Because the prefix-viable check is done incrementally, we believe that optimizations are available for indexing and candidate generation; e.g., a specialized index to share computation. Nonetheless, in order to evaluate the effectiveness of the pigeonring principle itself, we choose not to apply such optimizations in this paper and leave them as future work.

We analyze the search cost. Since the cost of feature extraction is irrespective of which principle is used, we only consider candidate generation and verification. For the pigeonhole principle-based method, the cost $C_{PH} = C_{C1} + |A_{PH}| \cdot c_V$. C_{C1}, $|A_{PH}|$, and c_V are the cost of (the first step of) candidate generation, the number of candidates by the pigeonhole principle, and the average cost of verifying a candidate, respectively. For the pigeonring principle-based method, the cost $C_{PR} = C_{C1} + C_{C2} + |A_{PR}| \cdot c_V$. C_{C2} and $|A_{PR}|$ are the cost of the second step of candidate generation and the number of candidates by the pigeonring principle, respectively. C_{C2} is upper-bounded by $(l-1) \cdot |V| \cdot c_B$. $|V|$ is the number of viable boxes identified in the first step. c_B is the average cost of checking a box in the second step. Since $|A_{PR}| \leq |A_{PH}|$, there is a tradeoff between C_{C2} and $|A_{PR}| \cdot c_V$, which we are going to evaluate through experiments.

8. EXPERIMENTS

8.1 Experiment Setup

We select eight datasets, two for each of the four τ-selection problems listed in Section 2. 1,000 objects are randomly sampled from each dataset as queries.

- **GIST** is a set of 80 million GIST descriptors for tiny images [99]. We convert them to 256-dimensional binary vectors by spectral hashing [114].
- **SIFT** is a set of 1 billion SIFT features from the BIGANN dataset [44]. We follow the method in [65] to convert them to 512-dimensional binary vectors.
- **Enron** is a set of 517,386 emails by employees of the Enron Corporation, each tokenized by white space and punctuation. The average number of tokens is 142.
- **DBLP** is a set of 860,751 bibliography records from the DBLP website. Each object is a concatenation of author name(s) and a publication title, tokenized by white space and punctuation. The average number of tokens is 14.
- **IMDB** is a set of 1 million actor/actress names from the IMDB website. The average string length is 16.
- **PubMed** is a set of 4 million publication titles from MEDLINE. The average string length is 101.
- **AIDS** is a set of 42,687 antivirus screen chemical compounds from the Developmental Therapeutics Program at NCI/NIH. The average numbers of vertices/edges are 26/28. The numbers of vertex/edge labels are 62/3.
- **Protein** is a set of 6,000 protein structures from the Protein Data Bank [80]. The original dataset has only 600 graphs. We make up our dataset by duplication and randomly applying minor errors. The average numbers of vertices/edges are 33/56. The numbers of vertex/edge labels are 3/5.

The following state-of-the-art methods are compared.

- Hamming distance search: GPH is a partition-based algorithm [73] for Hamming distance search. We set partition size $m = \lfloor d/16 \rfloor$ for best overall search time.
- Set similarity search: We use Jaccard similarity $J(x, y) = |x \cap y|/|x \cup y|$. It can be converted to an equivalent overlap similarity: $J(x, y) \geq \tau \iff |x \cap y| \geq (|x| + |y|)\tau/(1 + \tau)$. We consider three algorithms: (1) pkwise is a prefix filter-based algorithm [105] for local similarity search. It can be easily adapted for set similarity search and achieves good performance. We set the partition size of token universe to 4 (equivalent to $m = 5$), as suggested by [105]. (2) AdaptSearch [102] is a prefix filter-based algorithm dedicated to set similarity search. As its join version is shown to be slower than the AllPairs [8] and the PPJoin [116] algorithms in a few cases [61], we disable its extension of prefixes (and apply the position filter [116] if necessary) to make it the same as AllPairs' or PPJoin's search version, whenever either of the two is faster. (3) PartAlloc [30] is a partition filter-based algorithm for set similarity join. We adapt it for search. All the competitors are equipped with fast verification [61].
- String edit distance search: Pivotal is a q-gram-based algorithm [28]. The number of pivotal q-grams is $m = \tau + 1$. The q-gram length κ is set to $3, 2, 2, 2$ for $\tau = 1, 2, 3, 4$ on IMDB and $8, 6, 6, 4, 4$ for $\tau = 4, 6, 8, 10, 12$ on PubMed.
- Graph edit distance search: Pars is a partition-based algorithm [137]. The partition size $m = \tau + 1$.

Our pigeonring principle-based algorithms are denoted by Ring. When $l = 1$, Ring exactly becomes the above competitors (pkwise for set similarity search). We choose the same settings for Ring and its pigeonhole principle-based counterparts. The source codes were either from our previous work or received from the original authors of the aforementioned work.

The methods in [65, 132, 7, 52, 115, 72, 109, 136, 138] are not compared since prior work [73, 102, 61, 28, 137] showed they are outperformed by the above selected ones. Approximate methods are not considered because we focus on exact solutions and the main purpose of our experiments is to show the speedup on top of the pigeonhole principle-based methods.

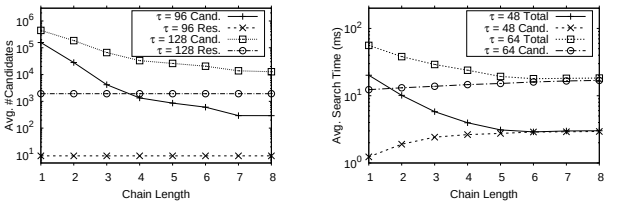
The experiments were carried out on a server with an Octa-Core Intel(R) Xeon(R) CPU @3.2GHz Processor and 256GB RAM, running Ubuntu 16.04. All the algorithms were implemented in C++ in a single thread main memory fashion.

Since we use exactly the same indexes and parameters for index construction as the pigeonhole principle-based counterparts do, the index sizes and the construction times are the same as theirs, thus not repeatedly evaluated.

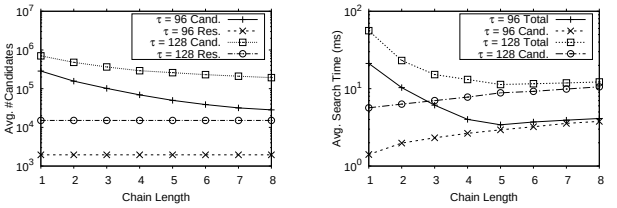
8.2 Effect of Chain Length

We study how the performance of Ring changes with chain length l.

The average numbers of candidates per query and the corresponding search times are plotted in Figures 3 – 6 ((a) and (c)). Two τ settings are shown for each dataset. We also plot the number of results. It can be observed the candidate

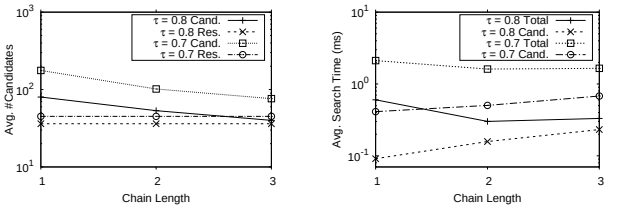


(a) GIST, Candidate (b) GIST, Time

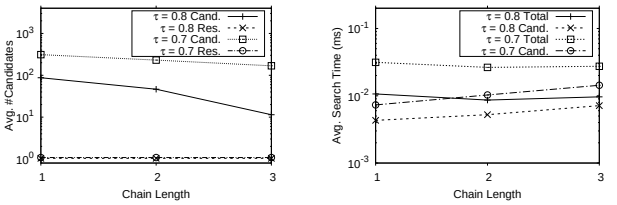


(c) SIFT, Candidate (d) SIFT, Time

Figure 3: Effect of Chain Length on Hamming Distance Search

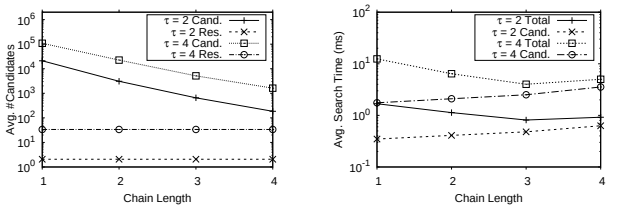


(a) Enron, Candidate (b) Enron, Time

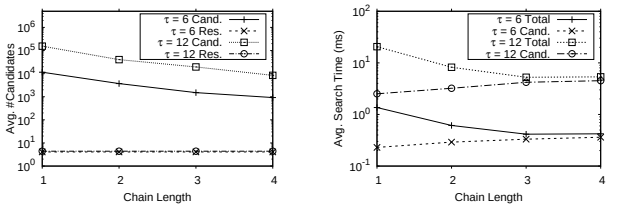


(c) DBLP, Candidate (d) DBLP, Time

Figure 4: Effect of Chain Length on Set Similarity Search

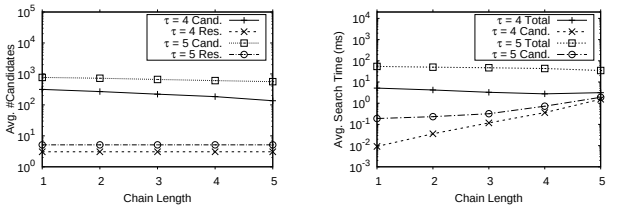


(a) IMDB, Candidate (b) IMDB, Time

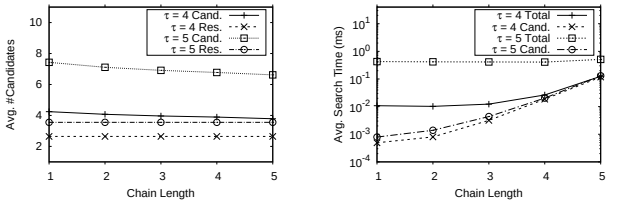


(c) PubMed, Candidate (d) PubMed, Time

Figure 5: Effect of Chain Length on String Edit Distance Search



(a) AIDS, Candidate (b) AIDS, Time



(c) Protein, Candidate (d) Protein, Time

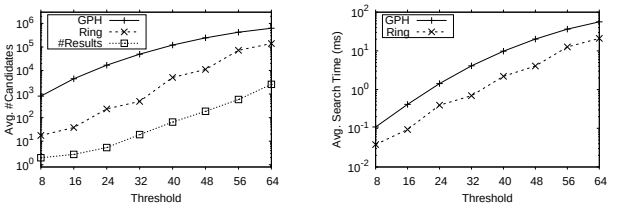
Figure 6: Effect of Chain Length on Graph Edit Distance Search

numbers decrease with the growth of chain length. This is expected: when l increases, because we look for prefix-viable chains from existing ones, the candidates are always reduced to a subset.

In Figures 3 – 6 ((b) and (d)), we plot the candidate generation time and the total search time. Their difference is the verification time. Feature extraction time is negligible and thus we make it subsumed by candidate generation. We observe: when the chain length increases, the candidate generation time keeps increasing, while the general trend of the total search time is to decrease and rebound. According to the analysis in Section 7, there is a tradeoff: with longer chains, we spend more time looking for prefix-viable chains, while the candidate number is reduced and the verification time is saved. The following settings achieve the overall best search time: (1) Hamming distance search: $l = 5$ or 6. (2) Set similarity search: $l = 2$. (3) String edit distance search: $l = \min(3, \tau+1)$. (4) Graph edit distance search: $l = [\tau - 2 \,..\, \tau]$. We use these settings in the rest of the experiments.

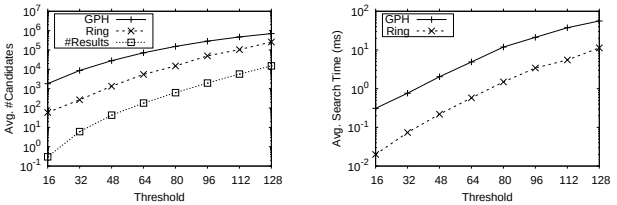
8.3 Comparison with Alternative Methods

Figures 7(a) – 7(d) show the average candidate number and search time on the two datasets for Hamming distance search. With the new principle, candidates and search time are significantly reduced. The speedup over GPH is up to 5.9 times on GIST and 15.5 times on SIFT. SIFT's dimensionality is twice as much as GIST's. This results in more expensive verification per candidate on SIFT. The reduction in candidates is thus more converted to search time. We also notice that the speedup on the two datasets comes from not only Hamming distance computation but also the union of candidate sets before verification (e.g., both b_1 and b_4 produce x^1 as a candidate in Example 2, and a union is required to avoid duplicate verification). This is attributed to two factors:



(a) Candidate, GIST

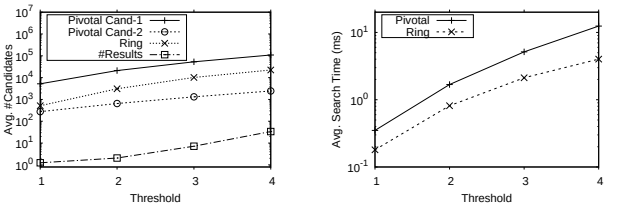
(b) Time, GIST



(c) Candidate, SIFT

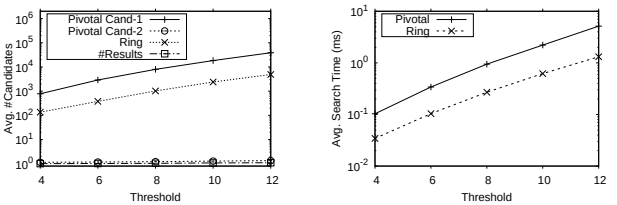
(d) Time, SIFT

Figure 7: Comparison on Hamming Distance Search



(a) Candidate, IMDB

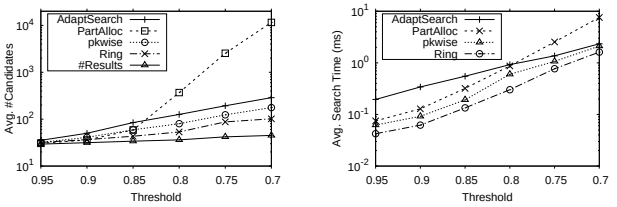
(b) Time, IMDB



(c) Candidate, PubMed

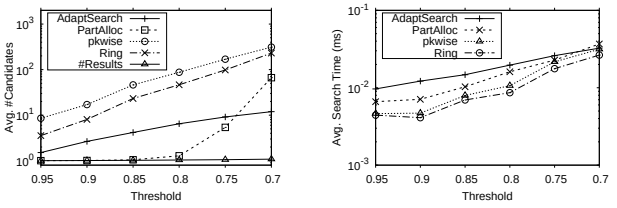
(d) Time, PubMed

Figure 9: Comparison on String Edit Distance Search



(a) Candidate, Enron

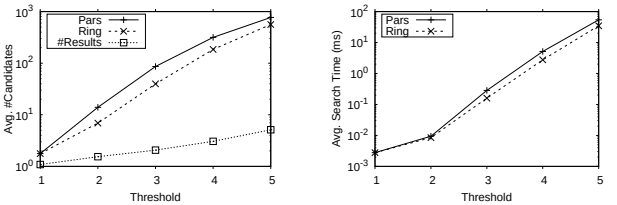
(b) Time, Enron



(c) Candidate, DBLP

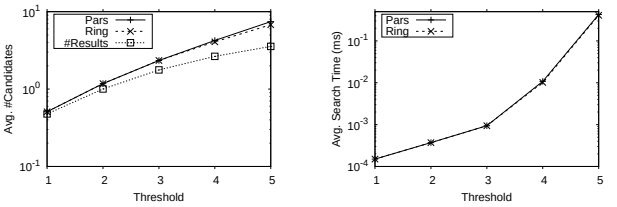
(d) Time, DBLP

Figure 8: Comparison on Set Similarity Search



(a) Candidate, AIDS

(b) Time, AIDS



(c) Candidate, Protein

(d) Time, Protein

Figure 10: Comparison on Graph Edit Distance Search

(1) The size of the input to the union is reduced by the new principle. (2) The prefix-viable check for Hamming distance search is faster than the hash table lookup used for the union.

The results for set similarity search are plotted in Figures 8(a) – 8(d). Note that this is a $f(x, q) \geq \tau$ case. The smaller the threshold is, the looser the constraint we have. The fastest competitor is Ring, followed by pkwise. Although PartAlloc has small candidate number (especially on DBLP), it spends too much time on candidate generation and thus become less efficient. It finds candidates by selecting signatures with a cost model. Due to the fast verification [61] on all the competitors, PartAlloc's advantage on candidate number is compromised. This is in accord with the results of a recent study on set similarity join [111], suggesting that we need not only small candidate number but also light-weight filtering. Both pkwise and AdaptSearch extend prefix lengths to find objects that share multiple tokens in prefixes. pkwise is faster because (1) pkwise uses token combinations to check the number of shared tokens, as opposed to AdaptSearch's merging long lists; and (2) AdaptSearch computes prefix lengths by a cost model, which incurs considerable overhead, despite reporting a smaller candidate number in a few cases. Ring exploits the advantage of pkwise and successfully reduces candidates from pkwise at a tiny additional cost by counting the overlap of same class tokens in prefixes (merging two very short lists), thereby becoming the fastest. The speedup over the runner-up, pkwise, is up to 2.0 times on Enron and 1.2 times on DBLP.

We provide the results for string edit distance search in Figures 9(a) – 9(d). We divide Pivotal's candidate number into two parts: the candidates that pass the pivotal prefix filter (denoted by Cand-1s) and the Cand-1s that pass the alignment filter (denoted by Cand-2s). Ring reduces candidates on the basis of Pivotal's Cand-1. By the alignment filter, Pivotal's Cand-2 number becomes less than Ring's candidate number, and even close to the result number on PubMed. However, since the filter involves expensive edit distance computation between q-grams and substrings, the small Cand-2 number does

not always pay off. Ring is always faster than Pivotal, by up to 3.1 times on IMDB and 3.9 times on PubMed. The reasons are: (1) Ring is able to early stop whenever the prefix-viable check fails at some length $l' \leq l \leq m$, whereas the alignment filter has to check m boxes. (2) Instead of computing the exact edit distance between a q-gram and a substring, Ring obtains a lower bound using bit vectors. This achieves good filtering power at the cost of only a few bit operations. The speedup of Ring is more significant on PubMed, where long q-grams are chosen to filter. This is in accord with the reduction in the time complexity of a box check from $O(\kappa^2 + \kappa\tau)$ to $O(\kappa + \tau)$.

For graph edit distance search, Figures 10(a) – 10(d) show the results on the two datasets. The reduction in candidate number and search time is not as significant as on the other problems. The main reason is, for the other problems, boxes are exclusive or almost exclusive, e.g., disjoint parts for Hamming distance search and disjoint token classes for set similarity search. For graph edit distance search, though the feature subgraphs are disjoint, their vertex mappings in the query graph via subgraph isomorphism may highly overlap. This fact showcases the hardness of complex structures like graphs. Nonetheless, Ring outperforms Pars by up to 1.9 times on AIDS. Their performances on Protein are close. Ring wins by a small margin of 1.04 times speedup. There are two factors for why the gap is more remarkable on AIDS: (1) There is still plenty of room between the numbers of candidates and results on AIDS. Ring is able to reduce candidates by more than 40% and thus have remarkable gain in search time. On Protein, since the two numbers are already close, the room for speedup is small. (2) Protein has much fewer labels than AIDS. This makes feature subgraphs less selective and more likely to be contained by the query graph, meaning the data graphs are more likely to pass the pigeonring principle-based filter.

9. RELATED WORK

Pigeonhole principle. The pigeonhole principle is a theorem in combinatorics. It has several forms in which the numbers of items and boxes differ [13]. The simple form discusses the case of $(n + 1)$ items in n boxes. The strong form discusses the case of $(\sum_{i=1}^{m} q_i - m + 1)$ items in m boxes, where q_i are positive integers. It is easy to extend these forms to real numbers. In set theory, it is formulated by Dirichlet drawer principle [25] using functions on finite sets. It can be also applied to infinite sets where n and m are described by cardinal numbers. Apart from these formulations, it has applications in various fields of mathematics. E.g., in number theory, Dirichlet's approximation theorem is a consequence of the pigeonhole principle [6]. It has also been used to bound the gaps between primes [95], which are steadily improved over the years towards proving the twin prime conjecture. The principle is also studied in theoretical computer science, especially for its provability and proof complexity [41, 14, 69, 71, 2, 50, 15, 78, 77]. In the area of databases, the principle has been extensively utilized to solve thresholded similarity searches [57, 132, 65, 73, 7, 30, 55, 107, 72, 28, 56, 137] which can be formalized as τ-selection problems, as well as other important problems such as association rule mining [85].

τ-selection problem. The study on τ-selection problems has received much attention in the last few decades. A multitude of solutions have been devised to handle different representations of objects and selection functions. A common scenario is to deal with objects in multi-dimensional space. Efficient solutions were proposed for binary vectors and Hamming distance [93, 57, 65, 132, 66, 73]. More investigations were towards vectors with real-valued dimensions and L^p distance. Notable approaches are tree-based indexing [24, 10], lower bounding [42], transformation (including dimension reduction) [9, 67, 130, 43, 114, 91], and locality sensitive hashing (LSH) [38, 26, 59, 96, 37]. Some of them targeted k-NN queries rather than thresholded queries. However, the pigeonhole principle is barely utilized for L^p distance, and approximate solutions are more popular than exact ones. We refer readers to a book on multi-dimensional indexes [82], a survey on dimension reduction [1], and a survey on the widely studied hashing-based approaches [104]. The searches with other similarity measures such as Bregman divergence [135] and earth mover's distance [118, 94] have also been investigated. Recently, much work was devoted to set similarity search and its variant of batch processing (similarity join). Most solutions were developed for overlap, Jaccard, or cosine similarities. Prevalent exact approaches are based on prefix filter [19, 8, 116, 79, 11, 102, 4, 60, 111]. Experimental evaluation can be found for set similarity join [61]. Other exact approaches include partition filter [7, 30], enumeration [27, 31], tree indexing [133], and postings list merge [83, 40]. Approximate approaches have also been developed, such as minhash and other LSH [12, 129, 84, 5, 22, 23]. The research on string similarity search (join) received tantamount attention. Most work adopted edit distance constraints. The methods are based on overlapping substrings (q-grams) [39, 52, 115, 109, 110, 113], non-overlapping substrings [53, 121, 107, 54, 72, 122, 28, 123], or tree indexes [134, 34, 33, 29, 58, 127]. Experimental evaluation for the join case was reported in [45]. We also recommend a survey [126]. Some work proposed to use fuzzy match on tokens [18, 103, 27]. A recent study targeted Jaro-Winkler distance [112]. Another line of methods cope with biosequence alignment, including BLAST [3], the Smith-Waterman algorithm [90], the BWT improvement [51], and those from the database area [62, 16, 68, 120]. For complex data types such as graphs, solutions have been developed for maximum common subgraph [119, 87, 86, 46] and graph edit distance [128, 101, 109, 136, 138, 56, 137] constraints. Another common data type is time series, including trajectories. Existing studies considered dynamic time warping [124, 49, 17, 139, 81, 47, 36, 48, 75, 125], edit distance [20, 21, 76, 64], longest common subsequence [100], other similarity measures [17, 35, 97, 98, 70, 88, 117, 92, 106], and systems for multiple similarity measures [89]. An experimental evaluation appeared in [32].

10. CONCLUSION

In this paper, we proposed the pigeonring principle, an extension of the pigeonhole principle with stronger constraints. We utilized the pigeonring principle to develop filtering methods for τ-selection problems. We showed that the resulting filtering condition always produces less or equal number of candidates than the pigeonhole principle does. Thus, all the pigeonhole principle-based solutions are possible to be accelerated by the new principle. A filtering framework was proposed to cover the pigeonring principle-based solutions. Based on the framework, we showed case studies for several common τ-selection problems. The pigeonring principle-based algorithms were implemented on top of existing pigeonhole principle-based solutions to these problems with minor modifications. The superiority of the pigeonring principle-based algorithms were demonstrated through experiments on real datasets.

11. REFERENCES

[1] N. Ailon and B. Chazelle. Faster dimension reduction. *Commun. ACM*, 53(2):97–104, 2010.

[2] M. Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14(4):417–433, 1994.

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[4] D. C. Anastasiu and G. Karypis. L2AP: fast cosine similarity search with prefix L-2 norm bounds. In *ICDE*, pages 784–795, 2014.

[5] A. Andoni, P. Indyk, T. Laarhoven, I. P. Razenshteyn, and L. Schmidt. Practical and optimal LSH for angular distance. In *NIPS*, pages 1225–1233, 2015.

[6] T. Apostol. *Modular Functions and Dirichlet Series in Number Theory*. Graduate Texts in Mathematics. Springer New York, 1997.

[7] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, 2006.

[8] R. J. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW*, 2007.

[9] S. Berchtold, C. Böhm, and H. Kriegel. The pyramid-technique: Towards breaking the curse of dimensionality. In *SIGMOD*, pages 142–153, 1998.

[10] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.

[11] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[12] A. Z. Broder. On the resemblance and containment of documents. In *SEQS*, 1997.

[13] R. Brualdi. *Introductory Combinatorics*. Math Classics. Pearson, 2017.

[14] S. R. Buss. Polynomial size proofs of the propositional pigeonhole principle. *The Journal of Symbolic Logic*, 52(4):916–927, 1987.

[15] S. R. Buss, R. Impagliazzo, J. Krajícek, P. Pudlák, A. A. Razborov, and J. Sgall. Proof complexity in algebraic systems and bounded depth frege systems with modular counting. *Computational Complexity*, 6(3):256–298, 1997.

[16] X. Cao, S. C. Li, B. C. Ooi, and A. K. H. Tung. Piers: An efficient model for similarity search in DNA sequence databases. *SIGMOD Record*, 33(2):39–44, 2004.

[17] K. Chan, A. W. Fu, and C. T. Yu. Haar wavelets for efficient similarity search of time-series: With and without time warping. *IEEE Trans. Knowl. Data Eng.*, 15(3):686–705, 2003.

[18] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313–324, 2003.

[19] S. Chaudhuri, V. Ganti, and R. Kaushik. A primitive operator for similarity joins in data cleaning. In *ICDE*, 2006.

[20] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[21] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[22] T. Christiani and R. Pagh. Set similarity search beyond minhash. In *STOC*, pages 1094–1107, 2017.

[23] T. Christiani, R. Pagh, and J. Sivertsen. Scalable and robust set similarity join. *CoRR*, abs/1707.06814, 2017.

[24] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.

[25] U. Daepp and P. Gorkin. *Reading, Writing, and Proving: A Closer Look at Mathematics*. Undergraduate Texts in Mathematics. Springer New York, 2003.

[26] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[27] D. Deng, A. Kim, S. Madden, and M. Stonebraker. Silkmoth: An efficient method for finding related sets with maximum matching constraints. *PVLDB*, 10(10):1082–1093, 2017.

[28] D. Deng, G. Li, and J. Feng. A pivotal prefix based filtering algorithm for string similarity search. In *SIGMOD*, pages 673–684, 2014.

[29] D. Deng, G. Li, J. Feng, and W. Li. Top-k string similarity search with edit-distance constraints. In *ICDE*, pages 925–936, 2013.

[30] D. Deng, G. Li, H. Wen, and J. Feng. An efficient partition based method for exact set similarity joins. *PVLDB*, 9(4):360–371, 2015.

[31] D. Deng, Y. Tao, and G. Li. Overlap set similarity joins with theoretical guarantees. In *SIGMOD*, pages 905–920, 2018.

[32] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB*, 1(2):1542–1552, 2008.

[33] J. Feng, J. Wang, and G. Li. Trie-join: a trie-based method for efficient string similarity joins. *VLDB J.*, 21(4):437–461, 2012.

[34] D. Fenz, D. Lange, A. Rheinländer, F. Naumann, and U. Leser. Efficient similarity search in very large string sets. In *SSDBM*, pages 262–279, 2012.

[35] E. Frentzos, K. Gratsias, and Y. Theodoridis. Index-based most similar trajectory search. In *ICDE*, pages 816–825, 2007.

[36] A. W. Fu, E. J. Keogh, L. Y. H. Lau, C. A. Ratanamahatana, and R. C. Wong. Scaling and time warping in time series querying. *VLDB J.*, 17(4):899–921, 2008.

[37] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.

[38] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

[39] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.

[40] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. In *ICDE*, pages 267–276, 2008.

[41] A. Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[42] Y. Hwang, B. Han, and H. Ahn. A fast nearest neighbor search algorithm by nonlinear embedding. In *CVPR*, pages 3053–3060, 2012.

[43] 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 Trans. Database Syst.*, 30(2):364–397, 2005.

[44] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: re-rank with source coding. *CoRR*, abs/1102.3828, 2011.

[45] Y. Jiang, G. Li, J. Feng, and W. Li. String similarity joins: An experimental evaluation. *PVLDB*, 7(8):625–636, 2014.

[46] C. Jin, S. S. Bhowmick, B. Choi, and S. Zhou. PRAGUE: towards blending practical visual subgraph query formulation and query processing. In *ICDE*, pages 222–233, 2012.

[47] E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005.

[48] E. J. Keogh, L. Wei, X. Xi, M. Vlachos, S. Lee, and P. Protopapas. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under euclidean and warping distance measures. *VLDB J.*, 18(3):611–630, 2009.

[49] S. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *ICDE*, pages 607–614, 2001.

[50] J. Krajícek, P. Pudlák, and A. R. Woods. An exponenetioal lower bound to the size of bounded depth frege proofs of the pigeonhole principle. *Random Struct. Algorithms*, 7(1):15–40, 1995.

[51] T. W. Lam, W. Sung, S. Tam, C. Wong, and S. Yiu. Compressed indexing and local alignment of DNA. *Bioinformatics*, 24(6):791–797, 2008.

[52] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, 2008.

[53] C. Li, B. Wang, and X. Yang. VGRAM: Improving performance of approximate queries on string collections using variable-length grams. In *VLDB*, 2007.

[54] G. Li, D. Deng, and J. Feng. A partition-based method for string similarity joins with edit-distance constraints. *ACM Trans. Database Syst.*, 38(2):9:1–9:33, 2013.

[55] G. Li, D. Deng, J. Wang, and J. Feng. Pass-Join: A partition-based method for similarity joins. *PVLDB*, 5(1):253–264, 2012.

[56] Y. Liang and P. Zhao. Similarity search in graph databases: A multi-layered indexing approach. In *ICDE*, pages 783–794, 2017.

[57] A. X. Liu, K. Shen, and E. Torng. Large scale hamming distance query processing. In *ICDE*, pages 553–564, 2011.

[58] W. Lu, X. Du, M. Hadjieleftheriou, and B. C. Ooi. Efficiently supporting edit distance based string similarity search using B^+-trees. *IEEE Trans. Knowl. Data Eng.*, 26(12):2983–2996, 2014.

[59] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[60] W. Mann and N. Augsten. PEL: position-enhanced length filter for set similarity joins. In *Proc. GvD (Foundations of Databases)*, pages 89–94, 2014.

[61] W. Mann, N. Augsten, and P. Bouros. An empirical evaluation of set similarity join techniques. *PVLDB*, 9(9):636–647, 2016.

[62] C. Meek, J. M. Patel, and S. Kasetty. OASIS: an online and accurate technique for local-alignment searches on biological sequences. In *VLDB*, pages 910–921, 2003.

[63] M. Mor and A. S. Fraenkel. A hash code method for detecting and correcting spelling errors. *Commun. ACM*, 25(12):935–938, 1982.

[64] R. Neamtu, R. Ahsan, E. A. Rundensteiner, and G. N. Sárközy. Interactive time series exploration powered by the marriage of similarity distances. *PVLDB*, 10(3):169–180, 2016.

[65] M. Norouzi, A. Punjani, and D. J. Fleet. Fast search in hamming space with multi-index hashing. In *CVPR*, pages 3108–3115, 2012.

[66] E. Ong and M. Bober. Improved hamming distance search using variable length hashing. In *CVPR*, pages 2000–2008, 2016.

[67] B. C. Ooi, K. Tan, C. Yu, and S. Bressan. Indexing the edges - A simple and yet efficient approach to high-dimensional indexing. In *PODS*, pages 166–174, 2000.

[68] P. Papapetrou, V. Athitsos, G. Kollios, and D. Gunopulos. Reference-based alignment in large sequence databases. *PVLDB*, 2(1):205–216, 2009.

[69] J. B. Paris, A. J. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *The Journal of Symbolic Logic*, 53(4):1235–1244, 1988.

[70] J. Peng, H. Wang, J. Li, and H. Gao. Set-based similarity search for time series. In *SIGMOD*, pages 2039–2052, 2016.

[71] T. Pitassi, P. Beame, and R. Impagliazzo. Exponential lower bounds for the pigeonhole principle. *Computational Complexity*, 3:97–140, 1993.

[72] J. Qin, W. Wang, C. Xiao, Y. Lu, X. Lin, and H. Wang. Asymmetric signature schemes for efficient exact edit similarity query processing. *ACM Trans. Database Syst.*, 38(3):16:1–16:44, 2013.

[73] J. Qin, Y. Wang, C. Xiao, W. Wang, X. Lin, and Y. Ishikawa. GPH: Similarity search in hamming space. In *ICDE*, pages 29–40, 2018.

[74] J. Qin and C. Xiao. Pigeonring: A principle for faster thresholded similarity search. *CoRR*, abs/1804.01614, 2018.

[75] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, pages 262–270, 2012.

[76] S. Ranu, D. P, A. D. Telang, P. Deshpande, and S. Raghavan. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*, pages 999–1010, 2015.

[77] R. Raz. Resolution lower bounds for the weak pigeonhole principle. *J. ACM*, 51(2):115–138, 2004.

[78] A. A. Razborov. Proof complexity of pigeonhole principles. In *DLT*, pages 100–116, 2001.

[79] L. A. Ribeiro and T. Härder. Generalizing prefix filtering to improve set similarity joins. *Inf. Syst.*, 36(1):62–78, 2011.

[80] K. Riesen and H. Bunke. IAM graph database repository for graph based pattern recognition and machine learning. In *SSPR & SPR*, pages 287–297, 2008.

[81] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. FTW: fast similarity search under the time warping distance. In *PODS*, pages 326–337, 2005.

[82] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[83] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *SIGMOD*, 2004.

[84] V. Satuluri and S. Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *PVLDB*, 5(5):430–441, 2012.

[85] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *VLDB*, pages 432–444, 1995.

[86] H. Shang, X. Lin, Y. Zhang, J. X. Yu, and W. Wang. Connected substructure similarity search. In *SIGMOD*, pages 903–914, 2010.

[87] H. Shang, K. Zhu, X. Lin, Y. Zhang, and R. Ichise. Similarity search on supergraph containment. In *ICDE*, pages 637–648, 2010.

[88] S. Shang, L. Chen, Z. Wei, C. S. Jensen, K. Zheng, and P. Kalnis. Trajectory similarity join in spatial networks. *PVLDB*, 10(11):1178–1189, 2017.

[89] Z. Shang, G. Li, and Z. Bao. DITA: distributed in-memory trajectory analytics. In *SIGMOD*, pages 725–740, 2018.

[90] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. In *Journal of Molecular Biology*, volume 147(1), pages 195–197, 1981.

[91] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *PVLDB*, 8(1):1–12, 2014.

[92] N. Ta, G. Li, Y. Xie, C. Li, S. Hao, and J. Feng. Signature-based trajectory similarity join. *IEEE Trans. Knowl. Data Eng.*, 29(4):870–883, 2017.

[93] Y. Tabei, T. Uno, M. Sugiyama, and K. Tsuda. Single versus multiple sorting in all pairs similarity search. *Journal of Machine Learning Research - Proceedings Track*, 13:145–160, 2010.

[94] Y. Tang, L. H. U, Y. Cai, N. Mamoulis, and R. Cheng. Earth mover's distance based similarity search at scale. *PVLDB*, 7(4):313–324, 2013.

[95] T. Tao. Small and large gaps in the primes. *Latinos in the Mathematical Sciences Conference*, April 2015.

[96] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst.*, 35(3), 2010.

[97] E. Tiakas, A. Papadopoulos, A. Nanopoulos, Y. Manolopoulos, D. Stojanovic, and S. Djordjevic-Kajan. Searching for similar trajectories in spatial networks. *Journal of Systems and Software*, 82(5):772–788, 2009.

[98] E. Tiakas and D. Rafailidis. Scalable trajectory similarity search based on locations in spatial networks. In *MEDI*, pages 213–224, 2015.

[99] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

[100] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[101] G. Wang, B. Wang, X. Yang, and G. Yu. Efficiently indexing large sparse graphs for similarity search. *IEEE Trans. Knowl. Data Eng.*, 24(3):440–451, 2012.

[102] J. Wang, G. Li, and J. Feng. Can we beat the prefix filtering?: an adaptive framework for similarity join and search. In *SIGMOD*, pages 85–96, 2012.

[103] J. Wang, G. Li, and J. Feng. Extending string similarity join to tolerant fuzzy token matching. *ACM Trans. Database Syst.*, 39(1):7:1–7:45, 2014.

[104] J. Wang, H. T. Shen, J. Song, and J. Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[105] P. Wang, C. Xiao, J. Qin, W. Wang, X. Zhang, and Y. Ishikawa. Local similarity search for unstructured text. In *SIGMOD*, pages 1991–2005, 2016.

[106] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu, and X. Qin. Torch: A search engine for trajectory data. In *SIGIR*, pages 535–544, 2018.

[107] W. Wang, J. Qin, C. Xiao, X. Lin, and H. T. Shen. Vchunkjoin: An efficient algorithm for edit similarity joins. *IEEE Trans. Knowl. Data Eng.*, 25(8):1916–1929, 2013.

[108] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit constraints. In *SIMGOD*, 2009.

[109] X. Wang, X. Ding, A. K. H. Tung, S. Ying, and H. Jin. An efficient graph indexing method. In *ICDE*, pages 210–221, 2012.

[110] X. Wang, X. Ding, A. K. H. Tung, and Z. Zhang. Efficient and effective KNN sequence search with approximate n-grams. *PVLDB*, 7(1):1–12, 2013.

[111] X. Wang, L. Qin, X. Lin, Y. Zhang, and L. Chang. Leveraging set relations in exact set similarity join. *PVLDB*, 10(9):925–936, 2017.

[112] Y. Wang, J. Qin, and W. Wang. Efficient approximate entity matching using jaro-winkler distance. In *WISE*, pages 231–239, 2017.

[113] H. Wei, J. X. Yu, and C. Lu. String similarity search: A hash-based approach. *IEEE Trans. Knowl. Data Eng.*, 30(1):170–184, 2018.

[114] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[115] C. Xiao, W. Wang, and X. Lin. Ed-Join: an efficient algorithm for similarity joins with edit distance constraints. *PVLDB*, 1(1):933–944, 2008.

[116] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang. Efficient similarity joins for near-duplicate detection. *ACM Trans. Database Syst.*, 36(3):15:1–15:41, 2011.

[117] D. Xie, F. Li, and J. M. Phillips. Distributed trajectory similarity search. *PVLDB*, 10(11):1478–1489, 2017.

[118] J. Xu, Z. Zhang, A. K. H. Tung, and G. Yu. Efficient and effective similarity search over probabilistic data based on earth mover's distance. *PVLDB*, 3(1):758–769, 2010.

[119] X. Yan, P. S. Yu, and J. Han. Substructure similarity search in graph databases. In *SIGMOD*, pages 766–777, 2005.

[120] X. Yang, H. Liu, and B. Wang. ALAE: accelerating local alignment with affine gap exactly in biosequence databases. *PVLDB*, 5(11):1507–1518, 2012.

[121] X. Yang, B. Wang, and C. Li. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In *SIGMOD*, pages 353–364, 2008.

[122] X. Yang, B. Wang, C. Li, J. Wang, and X. Xie. Efficient direct search on compressed genomic data. In *ICDE*, pages 961–972, 2013.

[123] X. Yang, Y. Wang, B. Wang, and W. Wang. Local filtering: Improving the performance of approximate queries on string collections. In *SIGMOD*, pages 377–392, 2015.

[124] B. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *ICDE*, pages 201–208, 1998.

[125] R. Ying, J. Pan, K. Fox, and P. K. Agarwal. A simple efficient approximation algorithm for dynamic time warping. In *SIGSPATIAL GIS*, pages 21:1–21:10, 2016.

[126] M. Yu, G. Li, D. Deng, and J. Feng. String similarity search and join: a survey. *Frontiers Comput. Sci.*, 10(3):399–417, 2016.

[127] M. Yu, J. Wang, G. Li, Y. Zhang, D. Deng, and J. Feng. A unified framework for string similarity search with edit-distance constraint. *VLDB J.*, 26(2):249–274, 2017.

[128] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[129] J. Zhai, Y. Lou, and J. Gehrke. Atlas: a probabilistic algorithm for high dimensional similarity search. In *SIGMOD*, pages 997–1008, 2011.

[130] R. Zhang, B. C. Ooi, and K.-L. Tan. Making the pyramid technique robust to query types and workloads. In *ICDE*, pages 313–324, 2004.

[131] W. Zhang, K. Gao, Y. Zhang, and J. Li. Efficient approximate nearest neighbor search with integrated binary codes. In *ACM Multimedia*, pages 1189–1192, 2011.

[132] X. Zhang, J. Qin, W. Wang, Y. Sun, and J. Lu. Hmsearch: an efficient hamming distance query processing algorithm. In *SSDBM*, page 19, 2013.

[133] Y. Zhang, X. Li, J. Wang, Y. Zhang, C. Xing, and X. Yuan. An efficient framework for exact set similarity search using tree structure indexes. In *ICDE*, pages 759–770, 2017.

[134] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. B^{ed}-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD*, pages 915–926, 2010.

[135] Z. Zhang, B. C. Ooi, S. Parthasarathy, and A. K. H. Tung. Similarity search on bregman divergence: Towards non-metric indexing. *PVLDB*, 2(1):13–24, 2009.

[136] X. Zhao, C. Xiao, X. Lin, W. Wang, and Y. Ishikawa. Efficient processing of graph similarity queries with edit distance constraints. *VLDB J.*, 22(6):727–752, 2013.

[137] X. Zhao, C. Xiao, X. Lin, W. Zhang, and Y. Wang. Efficient structure similarity searches: a partition-based approach. *VLDB J.*, 27(1):53–78, 2018.

[138] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Efficient graph similarity search over large graph databases. *IEEE Trans. Knowl. Data Eng.*, 27(4):964–978, 2015.

[139] Y. Zhu and D. E. Shasha. Warping indexes with envelope transforms for query by humming. In *SIGMOD*, pages 181–192, 2003.