A Progressive Approach for Similarity Search on Matrix

Tsz Nam Chan[1], Man Lung Yiu[1], and Kien A. Hua[2]

[1] Department of Computing, Hong Kong Polytechnic University, Hong Kong,
{cstnchan,csmlyiu}@comp.polyu.edu.hk

[2] College of Engineering & Computer Science, University of Central Florida
kienhua@cs.ucf.edu

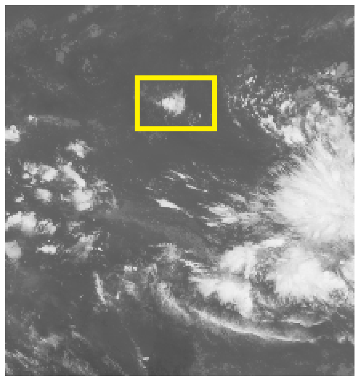
Abstract. We study a similarity search problem on a raw image by its pixel values. We call this problem as *matrix similarity search*; it has several applications, e.g., object detection, motion estimation, and super-resolution. Given a data image D and a query q, the best match refers to a sub-window of D that is the most similar to q. The state-of-the-art solution applies a sequence of lower bound functions to filter sub-windows and reduce the response time. Unfortunately, it suffers from two drawbacks: (i) its lower bound functions cannot support arbitrary query size, and (ii) it may invoke a large number of lower bound functions, which may incur high cost in the worst-case. In this paper, we propose an efficient solution that overcomes the above drawbacks. First, we present a generic approach to build lower bound functions that are applicable to arbitrary query size and enable trade-offs between bound tightness and computation time. We provide performance guarantee even in the worst-case. Second, to further reduce the number of calls to lower bound functions, we develop a lower bound function for a group of sub-windows. Experimental results on image data demonstrate the efficiency of our proposed methods.

1 Introduction

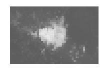
Multimedia databases [15, 14, 21] support similarity search on objects (e.g., images) by their feature vectors. In contrast, we consider a similarity search problem on a raw image by its pixel values. We call this problem as *matrix similarity search*; it has several applications, e.g., object detection [6], motion estimation [17], and super-resolution [7]. For example, we consider a satellite image in which each pixel represents a certain area on Earth (or in the sky). We illustrate a weather satellite image (obtained from [1]) in Figure 1a and a cloud pattern in Figure 1b. The matrix similarity search problem has been used for cloud motion estimation on satellite images [4]. This problem takes a data image D and a query image q as inputs (c.f. Figure 1). A candidate c refers to a sub-window (of D) with the same size as q. The matrix similarity search problem comes in two flavors [19, 22]:

- **Range search:** given a range τ_{range}, find every candidate c of D such that $dist(q,c) \leq \tau_{range}$.
- **Nearest neighbor (NN) search:** find a candidate c of D such that it has the smallest $dist(q,c)$.

The typical distance function $dist(q, c)$ is the L_p-norm distance (usually L_1 or L_2). In subsequent discussion, we let the size of D be $N_D = L_D \times W_D$, and the size of q be $N_q = L_q \times W_q$.



(a) data image D, of size $N_D = L_D \times W_D$ (best match: yellow rectangle)



(b) query image q, of size $N_q = L_q \times W_q$

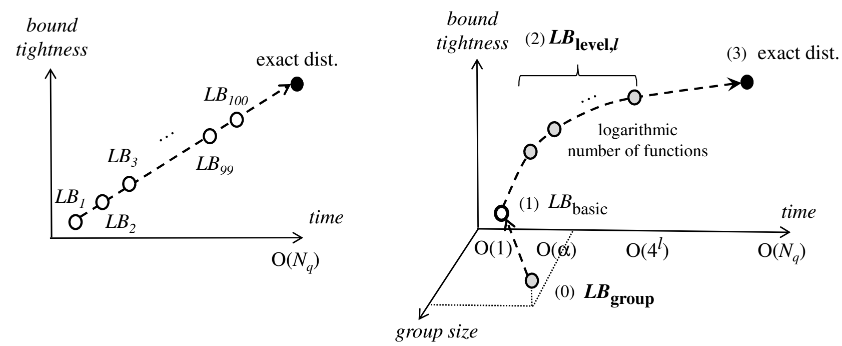
Fig. 1: The matrix similarity search problem

In this paper, we focus on the NN flavor of matrix similarity problem because some applications [17, 4] require finding the best match. Unlike the range search, the NN search has a fixed result size and does not require the user to supply a range parameter τ_{range} [22].

Schweitzer et al. [22] is the state-of-the-art NN search algorithm for the matrix similarity search problem. It applies a sequence of lower bound functions to filter candidates and reduce the response time. We illustrate this idea in Figure 2a. It starts with the cheapest lower bound function and then progressively apply tighter lower bound functions when necessary. However, this solution still suffers from two drawbacks. First, the lower bound functions in [22] are based on a Fourier transform on matrix (called the Walsh-Hadamard transform), which can only support query of the size $2^r \times 2^r$. Thus, it cannot support arbitrary query size. Second, in the worst case, it may invoke a large number of lower bound functions on a candidate, which may sum up to a high cost.

To avoid the above drawbacks on matrix similarity search, we contribute two lower bound functions $LB_{level,\ell}$ and LB_{group}, as shown in Figure 2b.

- When compared to Ref. [22], we present a generic approach to build a sequence of lower bound functions $LB_{level,\ell}$ that are applicable to arbitrary query size. As shown in Figure 2b, our approach would only call a logarithmic number of functions (in terms of N_q) in the worst-case.
- Existing lower bound functions take a single candidate as input. We develop a lower bound function LB_{group} that can take a group of candidates as input. This significantly reduces the frequency of calling lower bound functions for individual candidates.



(a) the state-of-the-art (b) our progressive approach

Fig. 2: Intuition

The rest of the paper is organized as follows. Section 2 defines our problem and introduces the background information. Section 3 presents our proposed solution. Section 4 discusses our experimental results. Section 5 elaborates on the related work. Section 6 concludes the paper with future research directions.

2 Preliminaries

2.1 Problem Definition

In this paper, we represent each image as a matrix. Let D be the data matrix (of size $N_D = L_D \times W_D$) and q be the query matrix (of size $N_q = L_q \times W_q$). A *candidate* $c_{x,y}$ is a sub-window of D with the same size as q.

$$c_{x,y}[1..L_q, 1..W_q] = D[x..x + L_q - 1, y..y + W_q - 1] \tag{1}$$

The subscript of $c_{x,y}$ denotes the start position in D; we drop it when the context is clear.

Problem 1 (Matrix NN Search). Given a query q and a data matrix D, find the candidate c_{best} such that it has the minimum $dist_p(q, c_{best})$, where the distance is defined as:

$$dist_p(q, c) = (\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |q[i, j] - c[i, j]|^p)^{\frac{1}{p}} \tag{2}$$

Figure 3 shows a query q of size 4×4 and a data matrix D of size 8×8. There are $5 \times 5 = 25$ candidates in D. For instance, the dotted sub-window refers to the candidate $c_{3,3}$. The right-side of Figure 3 enumerates the distances from q to each candidate, assuming the L_1 distance (i.e., $p = 1$) is used. In this example, the best match is $c_{3,3}$ because it has the smallest distance $dist_1(q, c_{3,3}) = 27$ from q.

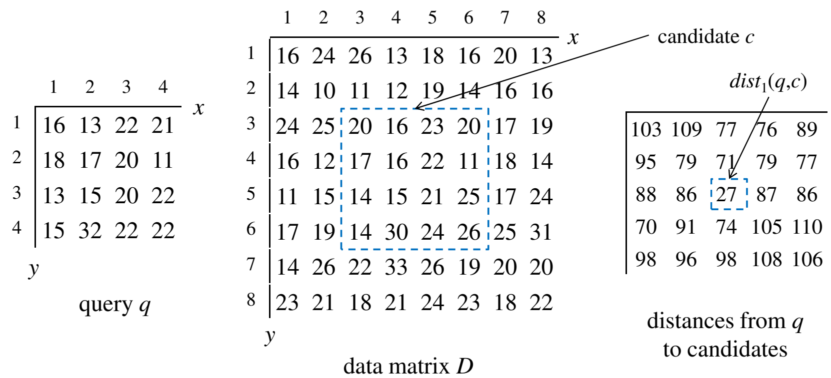


Fig. 3: Example for the problem

2.2 Background: Prefix-Sum Matrix & Basic Lower Bound Functions

In this section, we first introduce prefix-sum matrix and then discuss how they can be utilized to compute basic lower bound functions.

As we will introduce shortly, lower bound functions require summing the values in a rectangular region in a matrix. We can speedup their computation by using a prefix-sum matrix [11]. It is also called integral image [26] in the computer vision community.

Definition 1 (Prefix-sum matrix). *Given a matrix A (of size $N_A = L_A \times W_A$), we define its prefix-sum matrix P_A (of the same size) with entries:*

$$P_A[x, y] = \sum_{i=1}^{x} \sum_{j=1}^{y} A[i, j] \tag{3}$$

The prefix-sum matrix occupies $O(N_A)$ space and takes $O(N_A)$ construction time [11]. It enables us to find the sum of values of a rectangular region (say, $[x_1..x_2, y_1..y_2]$) in a matrix A in $O(1)$ time, according to Equation 4.

$$\sum A[x_1..x_2, y_1..y_2] = \begin{cases} P_A[x_2, y_2] & \text{if } x_1 = 1, y_1 = 1 \\ P_A[x_2, y_2] - P_A[x_1 - 1, y_2] & \text{if } x_1 > 1, y_1 = 1 \\ P_A[x_2, y_2] - P_A[x_2, y_1 - 1] & \text{if } x_1 = 1, y_1 > 1 \\ \quad P_A[x_2, y_2] + P_A[x_1 - 1, y_1 - 1] & \\ -P_A[x_1 - 1, y_2] - P_A[x_2, y_1 - 1] & \text{otherwise} \end{cases} \tag{4}$$

Figure 4 illustrates a data matrix D and its corresponding prefix-sum matrix P_D. The sum of values in the dotted region ([4..7,2..5]) in D can be derived from the entries (7,5), (3,1), (3,5), (7,1) in P_D.

We proceed to introduce the basic lower bound function LB_{basic} used in Figure 2. Since our solution will use LB_{basic} as a building block (cf. Section 3), we require that: (i) LB_{basic} can be computed in $O(1)$ time, (ii) $LB_{basic}(q, c) \leq dist_p(q, c)$ always holds, and (iii) LB_{basic} supports arbitrary query size.

$\Sigma D[4..7,2..5] = P_D[7,5] - P_D[3,5] - P_D[7,1] + P_D[3,1]$

data matrix D

$y \backslash x$	1	2	3	4	5	6	7	8
1	16	24	26	13	18	16	20	13
2	14	10	11	12	19	14	16	16
3	24	25	20	16	23	20	17	19
4	16	12	17	16	22	11	18	14
5	11	15	14	15	21	25	17	24
6	17	19	14	30	24	26	25	31
7	14	26	22	33	26	19	20	20
8	23	21	18	21	24	23	18	22

prefix-sum matrix P_D of D

$y \backslash x$	1	2	3	4	5	6	7	8
1	16	40	66	79	97	113	133	146
2	30	64	101	126	163	193	229	258
3	54	113	170	211	271	321	374	422
4	70	141	215	272	354	415	486	548
5	81	167	255	327	430	516	604	690
6	98	203	305	407	534	646	759	876
7	112	243	367	502	655	786	919	1056
8	135	287	429	585	762	916	1067	1226

Fig. 4: Example of a prefix-sum matrix

In this paper, we use the following lower bound functions as LB_{basic}.

$$LB_{\oplus}(q,c) = \frac{\sqrt[p]{N_q}}{N_q} \cdot \left| \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} q[i,j] - \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} c[i,j] \right| \tag{5}$$

$$LB_{\Delta}(q,c) = \left| \sqrt[p]{\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |q[i,j]|^p} - \sqrt[p]{\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |c[i,j]|^p} \right| \tag{6}$$

The first one ($LB_{\oplus}(q,c)$) is given in [28]. The second one ($LB_{\Delta}(q,c)$) is derived from the triangle inequality of the L_p distance [5, 13].

Observe that both of them can be computed in $O(1)$ time, by using a prefix-sum matrix as discussed before. Regarding the summation term for q, we can compute it once and then reuse it for every candidate c. For $LB_{\oplus}(q,c)$, the term $\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} c[i,j]$ can be derived from the prefix-sum matrix P_D (of data matrix D). For $LB_{\Delta}(q,c)$, the term $\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |c[i,j]|^p$ can be derived from the prefix-sum matrix $P_{D'}$, where the matrix D' is defined with entries: $D'[i,j] = (D[i,j])^p$.

As a remark, we are aware of lower bound functions used in the pattern matching literature [18, 25, 2, 10, 19]. However, since those lower bound functions take more than $O(1)$ time, we choose not to use them as LB_{basic} (the building block) in our solution.

3 Progressive Search Algorithm

We illustrate the flow of our proposed NN search method in Figure 5. Like [23, 16], we employ a min-heap H in order to process entries in ascending order of their lower bound distance. The main difference is that H contains two types of entries: (i) a candidate, (ii) a group of candidates. As discussed before, a *candidate* corresponds to a sub-window of D. On the other hand, a *group* represents a consecutive region of candidates. Initially, H contains a group entry that covers the entire D.

When we deheap an entry from H, we check whether it is a group or a candidate.

1. If it is a group G, then we divide it evenly into 4 groups G_1, G_2, G_3, G_4[3]. For each G_i, we compute the group lower bound $LB(q, G_i)$ and then enheap G_i into H.
2. If it is a candidate c, then we compute the candidate lower bound $LB_{level,\ell}(q, c)$ at the next level ℓ, and then enheap c into H again.

During this process, a group would degenerate into a candidate when it covers exactly one candidate. Similarly, when a candidate reaches the deepest level, we directly apply the exact distance function $dist(q, c)$ on it, and update the best NN distance found so far τ_{best}. The search terminates when the lower bound of a deheaped entry exceeds τ_{best}.

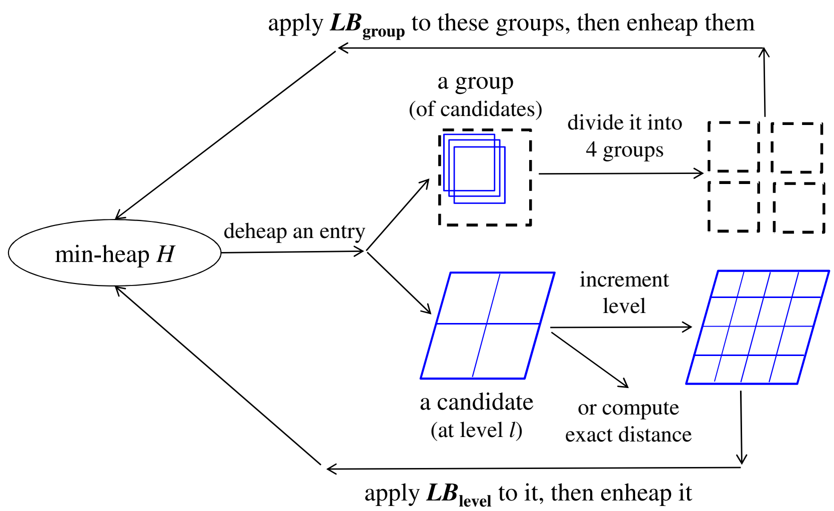


Fig. 5: The flow of our progressive search method

Table 1 lists the lower bound functions to be used in our NN search method. We have introduced LB_{basic} in Section 2.2. We will develop $LB_{level,\ell}$ and LB_{group} in Sections 3.1 and 3.2, respectively. Section 3.3 explores an efficient technique for computing LB_{group}. Finally, we summarize our proposed NN search algorithm in Section 3.4.

Function	Apply to	Cost
LB_{basic} (e.g., $LB_{\Delta}, LB_{\oplus}$)	candidate	$O(1)$
$LB_{level,\ell}$	candidate	$O(4^{\ell})$
LB_{group}	group	$O(\alpha)$

Table 1: Types of lower bound functions

[3] This is similar to the division of nodes in a quadtree.

3.1 Progressive Filtering for Candidates

As discussed in Section 1, the lower bound LB_{basic} and the exact distance $dist_p$ have a significant gap in terms of computation time and bound tightness (cf. Figure 2). In order to save expensive distance computations, we suggest to apply tighter lower bound functions progressively.

In this section, we present a generic idea to construct a parameterized lower bound function $LB_{level,\ell}$ by using LB_{basic} as a building block. The level parameter ℓ controls the trade-offs between the bound tightness and the computation time in $LB_{level,\ell}$. A small ℓ incurs small computation time whereas a large ℓ provides tighter bounds.

Intuitively, we build $LB_{level,\ell}$ by using divide-and-conquer. We can partition the space $[1..L_q, 1..W_q]$ into 4^ℓ disjoint rectangles $\{R_v : 1 \leq v \leq 4^\ell\}$, and then apply LB_{basic} (for q and c) in each rectangle R_v.[4] Then, we combine these 4^ℓ lower bound distances into $LB_{level,\ell}$ in Equation 7. The time complexity of $LB_{level,\ell}$ is $O(4^\ell)$, as each LB_{basic} takes $O(1)$ time.

$$LB_{level,\ell}(q,c) = \sqrt[p]{\sum_{v=1}^{4^\ell} LB_{basic}(q[R_v], c[R_v])^p} \tag{7}$$

For example, in Figure 6, when $\ell = 2$, both the query q and the candidate c are divided into $4^\ell = 16$ rectangles. We apply LB_{basic} on each rectangle in order to compute $LB_{level,\ell}(q,c)$. As a remark, the maximum possible level $\ell_{\max}$ (for ℓ) is:

$$\ell_{\max} = \lceil \log_2(\max\{L_q, W_q\}) \rceil \tag{8}$$

Next, we show that $LB_{level,\ell}$ satisfies the lower bound property.

Lemma 1. *For any candidate c, we have: $LB_{level,\ell}(q,c) \leq dist_p(q,c)$.*

Proof. For each region R_v, we have $LB_{basic}(q[R_v], c[R_v]) \leq \sqrt[p]{\sum_{(i,j)\in R_v} |q[i,j] - c[i,j]|^p}$, and thus $LB_{basic}(q[R_v], c[R_v])^p \leq \sum_{(i,j)\in R_v} |q[i,j] - c[i,j]|^p$. By summing it over all R_v, we obtain: $\sum_{v=1}^{4^\ell} LB_{basic}(q[R_v], c[R_v])^p \leq \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |q[i,j] - c[i,j]|^p$, because $\cup_{v=1}^{4^\ell} R_v$ covers all positions in the query matrix q. Thus we have: $LB_{level,\ell}(q,c) \leq dist_p(q,c)$. □

During search, we will apply $LB_{level,\ell}$ on a candidate in the ascending order of ℓ as shown in Figure 6. If we cannot filter c at level ℓ, then we attempt to filter it with minimal extra effort, i.e., at level $\ell + 1$. We justify this ascending ℓ order in Lemma 2.

Lemma 2. *Consider a candidate c that is not the nearest neighbor. The ascending level order achieves $cost_{order} \leq \frac{4}{3} \cdot cost_{opt}$, where $cost_{opt}$ is the optimal cost, and $cost_{order}$ is the cost of the order.*

[4] In general, the space $[1..L_q, 1..W_q]$ may have less than $O(4^\ell)$ disjoint rectangles.

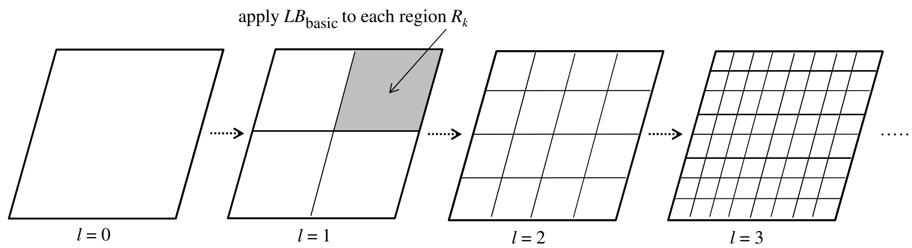


Fig. 6: $LB_{level,\ell}$ at different levels

Proof. Recall that the cost of $LB_{level,\ell}(q, c)$ is 4^ℓ. Let ℓ^* be the smallest level such that $LB_{level,\ell^*}(q, c) > dist_{NN}$, where $dist_{NN}$ is the best match distance.

In order to discard c, the optimal way (which knows ℓ^*) is to apply LB_{level,ℓ^*}. Thus, we have: $cost_{opt} = 4^{\ell^*}$.

For the ascending level order, we have: $cost_{order} = \sum_{i=0}^{\ell^*} 4^i = \frac{4^{\ell^*+1}-1}{3}$. Thus, we have: $cost_{order}/cost_{opt} \leq \frac{4^{\ell^*+1}-1}{3\times 4^{\ell^*}} \leq \frac{4}{3}$. □

3.2 Progressive Filtering for Groups

We first introduce the concept of a group and then propose a lower bound function for it. A *group* G represents a consecutive region of candidates as shown in Figure 7. It contains the following attributes: (i) L_g and W_g represent the size of the group, and (ii) x_{start} and y_{start} represent the start position (top-left corner) of the group. In order to cover all candidates in the group (e.g., those at bottom-right corner), we define the *extended region* as $G.R^{ext} = [x_{start}..x_{end}^{ext}, y_{start}..y_{end}^{ext}]$, where $x_{end}^{ext} = \min(x_{start} + L_g + L_q - 1, L_D)$ and $y_{end}^{ext} = \min(y_{start} + L_g + W_q - 1, W_D)$.

Our lower bound functions require the following concepts.

Definition 2 (The smallest / largest N_q values). *We define $N_q \min(G.R^{ext})$ as the multi-set of the smallest N_q values in the submatrix $D[G.R^{ext}]$, i.e., it satisfies:*

$$\max\{v : v \in N_q \min(G.R^{ext})\} \leq \min\{v : v \in D[G.R^{ext}] - N_q \min(G.R^{ext})\}$$

Then we define the following aggregates:
$\phi_{\min}(G.R^{ext}) = \sum_{v \in N_q \min(G.R^{ext})} v, \quad \phi^p_{\min}(G.R^{ext}) = \sum_{v \in N_q \min(G.R^{ext})} |v|^p.$

We define the max *versions (i.e., $N_q \max(G.R^{ext}), \phi_{\max}(G.R^{ext}), \phi^p_{\max}(G.R^{ext})$) in a similar way.*

We illustrate these concepts in Figure 8. Assume that $p = 2$ and the query size is $N_q = 2\times 2 = 4$. Consider the group G with region $G.R = [2..5, 2..5]$ (as dotted square) and the extended region $G.R^{ext} = [2..6, 2..6]$ (as bolded square). In this example, the smallest N_q values $G.R^{ext}$ are: 9, 9, 10, 10. Thus, we have: $\phi_{\min}(G.R^{ext}) = 9 + 9 + 10 + 10 = 38$, $\phi^2_{\min}(G.R^{ext}) = 2 \cdot 9^2 + 2 \cdot 10^2 = 362$.

We then extend basic lower bound functions (e.g., $LB_\oplus, LB_\Delta$) for a group G. We propose the lower bound functions $LB^\oplus_{group}$ and LB^Δ_{group} for G in Equations 9,10.

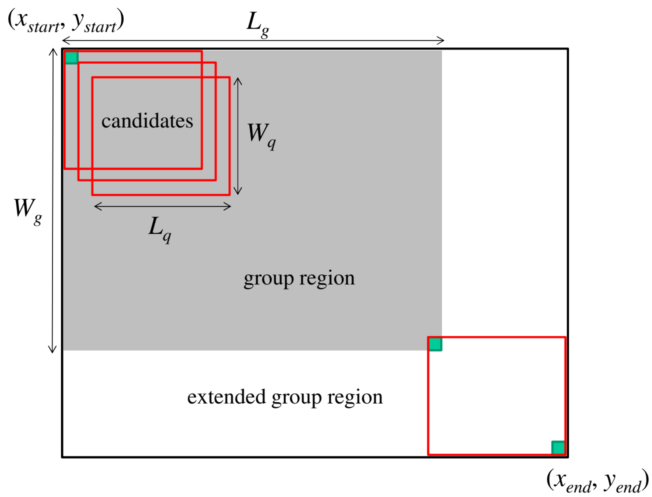


Fig. 7: Illustration of a group with $L_g \times W_g$ consecutive candidates

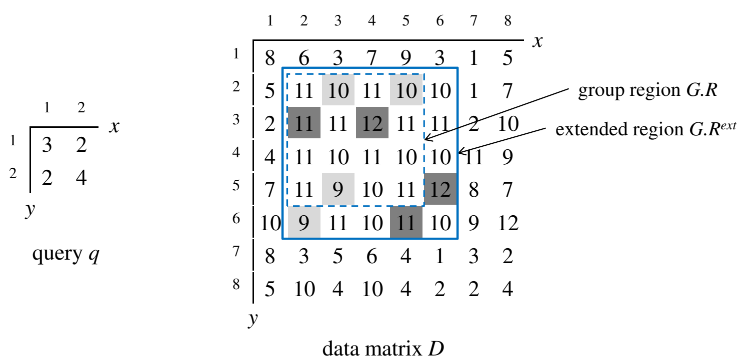


Fig. 8: Illustration of $N_q \min(G.R^{ext})$ and $N_q \max(G.R^{ext})$

They serve as lower bounds of $LB_{\oplus}(q, c), LB_{\Delta}(q, c)$ for any candidate c in G (cf. Lemmas 3,4).

$$LB^{\oplus}_{group}(q, G) = \begin{cases} \frac{\sqrt[p]{N_q}}{N_q}(\phi_{\min}(G.R^{ext}) - \sum_* q) & \text{if } \phi_{\min}(G.R^{ext}) > \sum_* q \\ \frac{\sqrt[p]{N_q}}{N_q}(\sum_* q - \phi_{\max}(G.R^{ext})) & \text{if } \phi_{\max}(G.R^{ext}) < \sum_* q \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$LB^{\Delta}_{group}(q, G) = \begin{cases} \sqrt[p]{\phi^p_{\min}(G.R^{ext})} - \sqrt[p]{\sum_* |q[i,j]|^p} & \text{if } \phi^p_{\min}(G.R^{ext}) > \sum_* |q[i,j]|^p \\ \sqrt[p]{\sum_* |q[i,j]|^p} - \sqrt[p]{\phi^p_{\max}(G.R^{ext})} & \text{if } \phi^p_{\max}(G.R^{ext}) < \sum_* |q[i,j]|^p \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where $\sum_* q = \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} q[i,j]$ and $\sum_* |q[i,j]|^p = \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |q[i,j]|^p$.

Lemma 3. *Given a group G, for any candidate c in G, we have: $LB^{\oplus}_{group}(q,G) \leq LB_{\oplus}(q,c)$.*

Proof. First, we focus on the first case of $LB^{\oplus}_{group}(q,G)$, i.e., when $\phi_{\min}(G.R^{ext}) > \sum_* q$.

Consider a candidate c in the group region of G. Since $N_q \min(G.R^{ext})$ contains the least N_q values in the group, we have: $\sum_* c \geq \phi_{\min}(G.R^{ext})$. Combining it with the condition in the first case, i.e., $\phi_{\min}(G.R^{ext}) > \sum_* q$), we have $\sum_* c \geq \phi_{\min}(G.R^{ext}) > \sum_* q$.

Then we apply the above inequality on $LB_{\oplus}(q,c)$ and derive: $LB_{\oplus}(q,c) = \frac{\sqrt[p]{N_q}}{N_q} \cdot (\sum_* c - \sum_* q) \geq \frac{\sqrt[p]{N_q}}{N_q}(\phi_{\min}(G.R^{ext}) - \sum_* q) = LB^{\oplus}_{group}(q,G)$.

We omit the proof for the second case as it is similar to the above argument. The proof for the third case (i.e., $LB^{\oplus}_{group}(q,G) = 0$) is trivial. □

Lemma 4. *Given a group G, for any candidate c in G, we have: $LB^{\Delta}_{group}(q,G) \leq LB_{\Delta}(q,c)$.*

Proof. First, we focus on the first case of $LB^{\Delta}_{group}(q,G)$, i.e., when $\phi^p_{\min}(G.R^{ext}) > \sum_* |q[i,j]|^p$.

Consider a candidate c in the group region of G. Since $N_q \min(G.R^{ext})$ contains the least N_q values in the group, we have: $\sum_* |c[i,j]|^p \geq \phi^p_{\min}(G.R^{ext})$. Combining it with the condition in the first case, i.e., $\phi^p_{\min}(G.R^{ext}) > \sum_* |q[i,j]|^p$, we have $\sum_* |c[i,j]|^p \geq \phi^p_{\min}(G.R^{ext}) > \sum_* |q[i,j]|^p$.

Then we apply the above inequality on $LB_{\Delta}(q,c)$ and derive: $LB_{\Delta}(q,c) = \sqrt[p]{\sum_* |c[i,j]|^p} - \sqrt[p]{\sum_* |q[i,j]|^p} \geq \sqrt[p]{\phi^p_{\min}(G.R^{ext})} - \sqrt[p]{\sum_* |q[i,j]|^p} = LB^{\Delta}_{group}(q,G)$.

We omit the proof for the second case as it is similar to the above argument. The proof for the third case (i.e., $LB^{\Delta}_{group}(q,G) = 0$) is trivial. □

We will discuss how to compute $LB_{group}(q,G)$ efficiently in the next subsection.

During our search procedure (cf. Figure 5), we will apply $LB_{group}(q,G)$ on a group G. If we cannot filter G, then we partition its group region $G.R$ into four sub-groups G_1, G_2, G_3, G_4 accordingly, and apply $LB_{group}(q,G_i)$ on each sub-group G_i.

3.3 Supporting Group Filtering Efficiently

The lower bound $LB_{group}(q,G)$ involves the terms $\phi_{\min}(G.R^{ext})$, $\phi_{\max}(G.R^{ext})$, $\phi^p_{\min}(G.R^{ext})$, $\phi^p_{\max}(G.R^{ext})$, which require finding the smallest N_q and the largest N_q values in $G.R^{ext}$.

In this section, we design a data structure called *prefix histogram matrix* to support the above operations efficiently, namely in $O(\alpha)$ time. The parameter α allows trade-off between the time complexity and the bound tightness. A larger α tends to provide tighter bounds, but it incurs more computation time.

We proceed to elaborate on how to construct the prefix histogram matrix for a data matrix D. First, we partition the values in matrix D into α bins and convert each value $D[i, j]$ to the following bin number $D'[i, j]$:

$$D'[i,j] = \left\lfloor \alpha \cdot \frac{D[i,j] - D_{min}}{D_{max} - D_{min} + 1} \right\rfloor + 1$$

where D_{min} and D_{max} denote the minimum and maximum values in D, respectively.

We define the *prefix histogram matrix* PH_D as a matrix where each entry $PH_D[i, j]$ is a vector:

$$PH_D[i,j] = \langle P_1[i,j], P_2[i,j], \cdots, P_\alpha[i,j] \rangle$$

where

$$P_v[i,j] = \texttt{count}_{(x,y)\in[1..i,1..j]}(D'[x,y] = v)$$

As a remark, the prefix histogram matrix occupies $O(\alpha N_D)$ space.

Figure 9a illustrates a histogram matrix PH_D in which each entry $PH_D[i, j]$ stores a count histogram for values in region $[1..i, 1..j]$ in the data matrix D.

Given an extended group region $G.R^{ext}$, we first retrieve count histograms at four corners of $G.R^{ext}$, and then combine them into the histogram as shown in Figure 9b. With this histogram, we can derive bounds for the minimum / maximum N_q values of $G.R^{ext}$ in D by Definition 3.

Definition 3 (Sum of the smallest / largest N_q values in a count histogram). *Let CH be a count histogram for $G.R^{ext}$. We define $\phi'_{\min}(CH)$ as the sum of the smallest N_q values in CH, and $\phi'_{\max}(CH)$ as the sum of the largest N_q values in CH.*

While scanning the bins of CH from left to right, we examine the count and the minimum bound of each bin to derive $\phi'_{\min}(CH)$. A similar way can be used to derive $\phi'_{\max}(CH)$. The time complexity is $O(\alpha)$ as CH contains α bins.

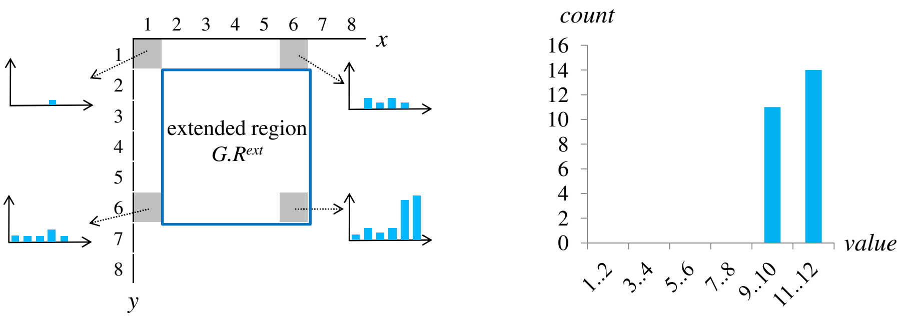
As an example, consider the count histogram CH obtained in Figure 9b. Assume that $\alpha = 6$ and $N_q = 4$. Thus, the width of each bin is $\frac{D_{max}-D_{min}+1}{\alpha} = \frac{12}{6} = 2$. Since the count of bin 9..10 is above N_q, we derive: $\phi'_{\min}(CH) = 9 \cdot 4 = 36$. Note that $\phi'_{\min}(CH) = 36$ is looser than the actual value $\phi_{\min}(G.R^{ext}) = 38$ (obtained in Figure 8).

Then we replace $LB^{\oplus}_{group}$ by the following function $LB'^{\oplus}_{group}$:

$$LB'^{\oplus}_{group}(q,G) = \begin{cases} \frac{\sqrt[p]{N_q}}{N_q}(\phi'_{\min}(CH) - \sum_* q) & \text{if } \phi'_{\min}(CH) > \sum_* q \\ \frac{\sqrt[p]{N_q}}{N_q}(\sum_* q - \phi'_{\max}(CH)) & \text{if } \phi'_{\max}(CH) < \sum_* q \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

Since $\phi'_{\min}(CH) \leq \phi_{\min}(G.R^{ext})$ and $\phi'_{\max}(CH) \geq \phi_{\max}(G.R^{ext})$, $LB'^{\oplus}_{group} \leq LB^{\oplus}_{group}$.

Similarly, we can adapt the above technique to derive a lower bound of LB^{Δ}_{group} efficiently.



(a) prefix histogram matrix $\mathrm{PH_D}$

(b) histogram for $G.R^{ext}$

(= $\mathrm{PH_D}[6,6] - \mathrm{PH_D}[6,1] - \mathrm{PH_D}[1,6] + \mathrm{PH_D}[1,1]$)

Fig. 9: prefix histogram matrix, $\alpha = 6, D_{min} = 1, D_{max} = 12$

3.4 Algorithm for NN Search

In this section, we summarize our techniques in Algorithm 1. Like [23, 16], we employ a min-heap H in order to process entries in ascending order of their lower bound distance. Also, we maintain the best distance found-so-far τ_{best} during the search. The main difference from [23, 16] is that we apply multiple lower bound functions on candidates and also consider lower bound function for groups of candidates.

Initially, we create an entry e_{root} to represent the group of all candidates. In each iteration, we deheap an entry e and check whether it is a group entry or a candidate entry. When e is a group entry, we divide it into four group entries and enheap them into H. Otherwise, e is a candidate entry, and then we examine the level of e. If e has not reached the maximum level ℓ_{max}, we compute $LB_{level,\ell}(q, e)$, advance it to the next level, and enheap it into H. Otherwise, we compute the exact distance of e from q, and update τ_{best} if necessary. The loop terminates when H becomes empty or the lower bound of the current entry exceeds τ_{best}.

4 Experimental Evaluation

In this section, we compare the efficiency of our methods with the state-of-the-art method [22] called Dual-Bound (DB). Table 2 shows the bounding functions used in these methods. Our *progressive search* methods share the same prefix PS:

– PSL stands for progressive search with LB_{level} only, and
– PSLG stands for progressive search with both LB_{level} and LB_{group}.

The subscripts of our methods (e.g., $\oplus$ or Δ) indicate whether they use lower bound functions built on top of $LB_{\oplus}$ or LB_{Δ}. We implemented all algorithms in C/C++ and conducted experiments on an Intel i7 3.4GHz PC running Ubuntu.

Algorithm 1 Progressive Search Algorithm for NN search

procedure PROGRESSIVE SEARCH(query matrix q, data matrix D)
  $\tau_{best} \leftarrow \infty$ ▷ best NN distance found so far
  create a min-heap H
  create a heap entry e_{root}
  $e_{root}.G \leftarrow [0..L_D - 1, 0..W_D - 1]$ ▷ the region covered by the group
  $e_{root}.bound \leftarrow LB_{group}(q, e.G)$
  enheap e_{root} to H
  while $H \neq \emptyset$ **do**
    $e \leftarrow$ deheap an entry in H
    if $e.bound \geq \tau_{best}$ **then** ▷ termination condition
      break
    if $|e.G| \neq 1$ **then** ▷ group entry
      divide e into 4 entries e_1, e_2, e_3, e_4
      for each e_i, $i \leftarrow 1$ to 4 **do**
        $e_i.bound \leftarrow LB_{group}(q, e_i.G)$
        enheap e_i to H **if** $e_i.bound \leq \tau_{best}$
    else ▷ candidate entry
      if $e.\ell < \ell_{max}$ **then** ▷ not at the deepest level
        $e.bound \leftarrow LB_{level,\ell}(q, e)$
        increment $e.\ell$
        enheap e to H **if** $e.bound \leq \tau_{best}$
      else
        compute $dist_p(q, c)$

Note that each method (in Table 2) requires a preprocessing step — scan a data image D to compute its prefix-sum matrix. This step is done only once before queries arrive. It is negligible compared to the query response time.

Table 3 summarizes the details of our image data and queries. We collect image datasets from [1, 19]. *Photo* [19] contains 30 images of the size 2560×1920. *Weather* [1] contains 30 weather satellite images of the size 1800×1800; the timestamps of these images are from 00:00 on 1/4/2014 to 06:00 on 2/4/2014. For each image, we generate 10 random starting positions by the uniform distribution to extract queries from that image.

In each experiment, we execute the methods on 300 queries (= 30 images $\times$ 10 queries) and then report the average response time.

Method	Bounding functions used in the method
DB	[22]
PSL$_\oplus$	$LB_\oplus, LB_{level}$
PSL$_\Delta$	LB_Δ, LB_{level}
PSLG$_\oplus$	$LB_\oplus, LB_{level}, LB_{group}$

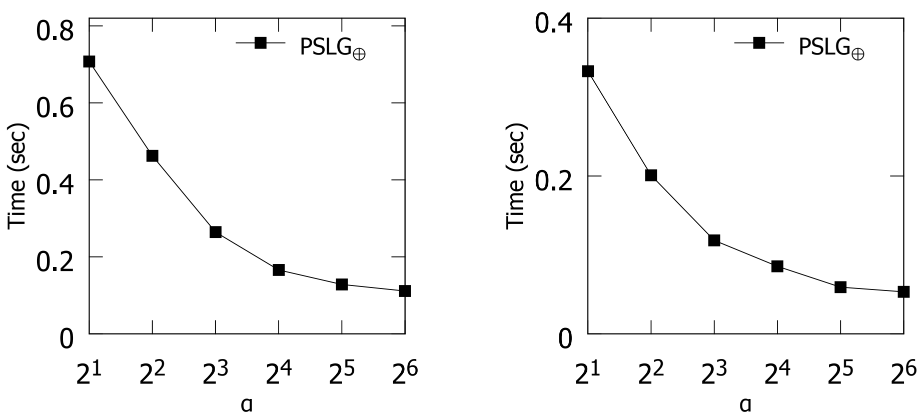
Table 2: The list of our methods and the competitor

Dataset	Image size	Number of images	Number of queries per image
Photo	2560×1920	30	10
Weather	1800×1800	30	10

Table 3: Our datasets and queries

4.1 Experimental Results

First, we study the effect of the number of bins α on the response time of our method PSLG$_\oplus$. Figure 10 plots the running time as a function of α. When α increases, the group-based lower bound LB_{group} becomes tighter (i.e., higher pruning power) so the response time drops. Nevertheless, when α is too large, it incurs high overhead to compute LB_{group} so the response time rises slightly. In subsequent experiments, we set $\alpha = 16$ by default.



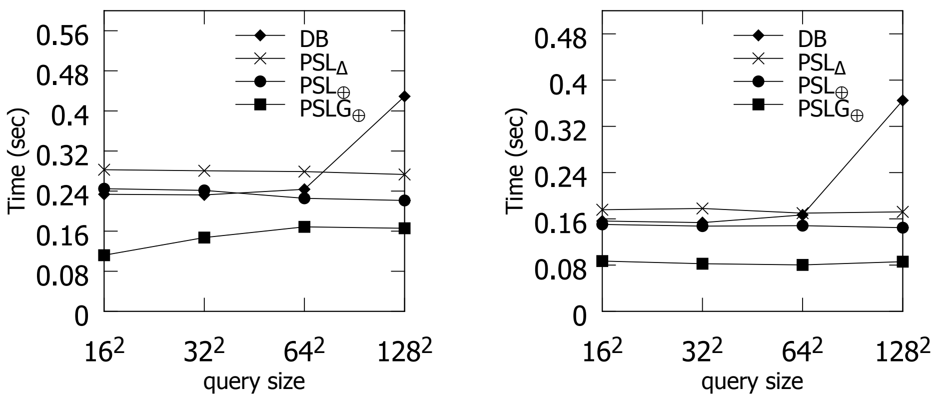
(a) Photo (b) Weather

Fig. 10: Response time vs. the number of bins α

Next, we evaluate the scalability of methods with respect to the query size N_q. Figure 11 shows the response time of methods versus the query size N_q. Since DB [22] can only support query size of the form $2^r \times 2^r$, we use query sizes like $32^2, 64^2, \cdots$ in this experiment. Thanks to the group lower bound function, PSLG$_\oplus$ outperforms all other methods and scales better with respect to N_q. On the other hand, DB, PSL$_\Delta$ and PSL$_\oplus$ need to obtain candidates one-by-one and incur higher overhead on maintaining the min-heap.

Since PSL$_\oplus$ performs better than PSL$_\Delta$, we omit PSL$_\Delta$ in the next experiment.

Following [22], we then test the robustness of methods by adding noise to queries. As in [22], Gaussian noise with a standard deviation σ is added into each query image.



(a) Photo (b) Weather

Fig. 11: Response time vs. vary query size N_q

The query size is fixed to 128×128 in this experiment. Figure 12 shows the response time of methods as a function of σ. The performance gap between our methods and DB widens as σ increases. At a high σ, the pruning power of all lower bound functions becomes weaker. For each worst-case candidate (that cannot be pruned), DB may invoke a long sequence of bounding functions on it, whereas our methods invoke only a logarithmic number of LB_{level} (in terms of N_q) on it. In summary, our methods are more robust against noise.

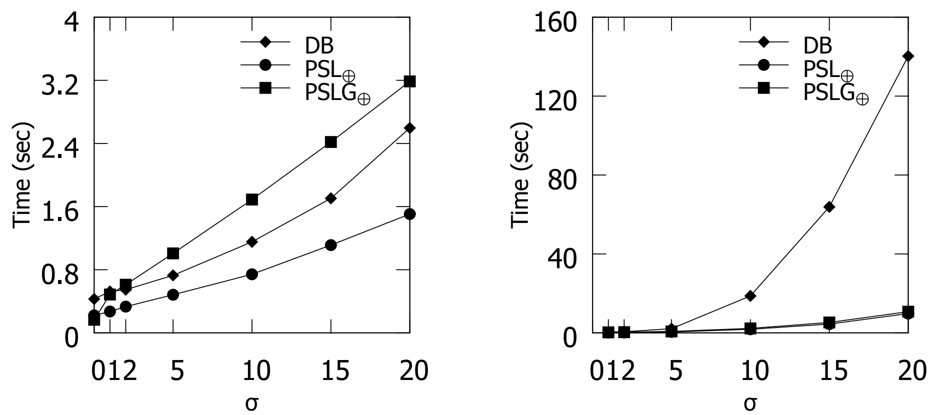
5 Related Work

5.1 Nearest neighbor search

The nearest neighbor (NN) search problem has been extensively studied in multimedia databases [15, 14, 21] and in time series databases [28, 8, 20].

Multimedia databases [15, 14, 21] usually conduct similarity search (i.e., NN search) on *feature vectors* of images (e.g., their color / texture histograms) rather than on raw pixel values in images. Various techniques on indexing [3, 13, 21], data compression [27], and hashing [24, 12] have been developed to process NN search efficiently. Recall that those multimedia techniques require knowing feature vectors in advance. Those techniques are applicable to our problem context, when the query size N_q is fixed, as we can convert each candidate (sub-window) $c_{x,y}$ to a N_q-dimensional feature vector offline. However, those techniques become inapplicable if we need to support arbitrary query size (i.e., N_q only known at the query time). It is infeasible to do precomputation for every possible query size as it would blow up the storage space by a huge factor (${N_D}^2$), where $N_D = L_D \times W_D$ is the size of the data image.

Generic NN search algorithms [23, 16] are applicable to any types of objects and distance function $dist(q, c)$. Ref. [23] requires using a lower bound function $LB(q, c)$,



(a) Photo (b) Weather

Fig. 12: Response time vs. the noise σ

where $LB(q, c) \leq dist(q, c)$ always holds. Its search strategy [23] is to examine candidates in ascending order of $LB(q, c)$ and then compute their exact distances to q, until the current $LB(q, c)$ exceeds the best NN distance found so far. Ref. [16] takes an additional upper bound function $UB(q, c)$ as input and utilizes it to further reduce the searching time. Observe that the lower bound functions for a specific problem (e.g., matrix similarity search problem) are not provided in [23, 16]. In this paper, we focus on developing lower bound functions like LB_{level}, LB_{group} for matrix similarity search.

The NN search on a time series [28, 8, 20] can be considered as a special case of our problem, where both the data image D and the query q are modeled as vectors instead of matrices. While some simple lower bound functions originate from them, our proposed lower bound functions (LB_{level}, LB_{group}) are new and specific to matrix similarity search. Specifically, our LB_{level} is a generic function that can be built on top of any given LB_{basic}, and our LB_{group} can take a group of candidates as input.

5.2 Matrix similarity search methods

Various lower bound functions [18, 25, 2, 10, 19, 9, 22] have been developed for the matrix similarity search problem, in order to prune unpromising candidates efficiently and thus avoid expensive exact distance computations. Most solutions focus on range search and a few study on NN search. Ouyang et al. [19] proposes a unified framework that covers range search solutions [18, 25, 2, 10, 9]. The state-of-the-art NN search method is [22]. It applies both lower and upper bound functions to accelerate NN search. Its lower / upper bound functions are based on a Fourier transform on matrix (called the Walsh-Hadamard transform), which can only support query of the size $2^r \times 2^r$. Thus, it cannot support arbitrary query size. Also, [22] has not explored our group-based lower bound function LB_{group}, which applies to a group of candidates instead of a single candidate.

6 Conclusion

We have developed a progressive NN search method for the matrix similarity search problem. It includes a generic lower bound function LB_{level} for candidates, and a group-based lower bound function LB_{group} for a group of candidates. Our proposed solution performs much better than the state-of-the-art method.

In the future, we plan to investigate approximate NN search methods for matrix similarity search. Sampling techniques may be adapted to address this problem.

References

1. Weather datasets. http://weather.is.kochi-u.ac.jp/sat/GAME/.
2. G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or. The gray-code filter kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):382–393, 2007.
3. C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, 2001.
4. R. Brad and I. A. Letia. Extracting cloud motion from satellite image sequences. In *ICARCV*, pages 1303–1307, 2002.
5. 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.
6. R. M. Dufour, E. L. Miller, and N. P. Galatsanos. Template matching based object recognition with unknown geometric parameters. *IEEE Transactions on Image Processing*, 11(12):1385–1396, 2002.
7. W. T. Freeman, T. R. Jones, and E. C. Pasztor. Example-based super-resolution. *IEEE Computer Graphics and Applications*, 22(2):56–65, 2002.
8. 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.
9. M. Gharavi-Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *IEEE Transactions on Image Processing*, 10(4):526–533, 2001.
10. Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(9):1430–1445, 2005.
11. C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *SIGMOD*, pages 73–88, 1997.
12. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.
13. H. V. Jagadish, B. C. Ooi, K. 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.
14. D. A. Keim and B. Bustos. Similarity search in multimedia databases. In *ICDE*, page 873, 2004.
15. F. Korn, N. Sidiropoulos, C. Faloutsos, E. L. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB*, pages 215–226, 1996.
16. H. Kriegel, P. Kröger, P. Kunath, and M. Renz. Generalizing the optimality of multi-step *k* -nearest neighbor query processing. In *SSTD*, pages 75–92, 2007.
17. Y. Moshe and H. Hel-Or. Video block motion estimation based on gray-code kernels. *IEEE Transactions on Image Processing*, 18(10):2243–2254, 2009.
18. W. Ouyang and W. Cham. Fast algorithm for walsh hadamard transform on sliding windows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):165–171, 2010.

19. W. Ouyang, F. Tombari, S. Mattoccia, L. di Stefano, and W. Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1):127–143, 2012.
20. 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.
21. H. Samet. Techniques for similarity searching in multimedia databases. *PVLDB*, 3(2):1649–1650, 2010.
22. H. Schweitzer, R. A. Deng, and R. F. Anderson. A dual-bound algorithm for very fast and exact template matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(3):459–470, 2011.
23. T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.
24. Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.
25. F. Tombari, S. Mattoccia, and L. di Stefano. Full-search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(1):129–141, 2009.
26. P. A. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.
27. R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.
28. B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB*, pages 385–394, 2000.