Efficient Sub-Window Nearest Neighbor Search on Matrix

Tsz Nam Chan, Man Lung Yiu, Kien A. Hua, *Fellow, IEEE*

Abstract—We study a nearest neighbor search problem on a matrix by its element values. Given a data matrix D and a query matrix q, the sub-window nearest neighbor search problem finds a sub-window of D that is the most similar to q. This problem has a wide range of applications, e.g., geospatial data integration, object detection and motion estimation. In this paper, we propose an efficient progressive search solution that overcomes the drawbacks of existing solutions. First, we present a generic approach to build level-based lower bound functions on top of basic lower bound functions. Second, we develop a novel lower bound function for a group of sub-windows, in order to boost the efficiency of our solution. Furthermore, we extend our solution to support irregular-shaped queries. Experimental results on real data demonstrate the efficiency of our proposed methods.

Index Terms—nearest neighbor, similarity search

✦

1 INTRODUCTION

Multimedia databases [22], [24], [33] support similarity search on image objects by their feature vectors. In contrast, we consider a similarity search problem on a matrix (e.g., pixel value in an image) by its element values. Specifically, given a data matrix D and a query matrix q, the *Sub-Window Nearest Neighbor Search* (SWNNS) problem finds a sub-window c of D such that c is the best match of q with the same size, i.e., having the smallest distance from q [34]. The typical distance function $dist(q, c)$ is the L_p-norm distance (usually L_1 or L_2).

This problem has a wide range of applications, e.g., geospatial data integration [8], [9], [11], motion estimation [27], object detection [6], [14], [23], image editing [10] and compact encoding for scientific array databases [35]. Figure 1 illustrates the SWNNS problem on satellite images in two applications. Note that each pixel in a satellite image represents a certain area on Earth.

- Weather forecasting involves estimating cloud motion on a sequence of weather satellite images [1], [5]. In this case, D can be the latest image (cf. Figure 1a), and q is a cloud pattern (cf. Figure 1b) extracted from a previous image. SWNNS returns the best match (i.e., yellow rectangle in Figure 1a), whose location can then be utilized to estimate cloud motion.
- An example of geospatial data integration is to stitch multiple satellite map images into a single map image [8], [9], [11]. Let D be a map image (cf. Figure 1d), and q be a road junction pattern (cf. Figure 1e) extracted from another map image. The best match position of SWNNS enables us to stitch map images.

- T. N. Chan and M. L. Yiu are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong.
  E-mail: {cstnchan, csmlyiu}@comp.polyu.edu.hk
- K. A. Hua is with the College of Engineering & Computer Science, University of Central Florida.
  E-mail: kienhua@cs.ucf.edu

In subsequent discussion, the query q can either take a rectangular shape [2], [17], [18], [28], [29], [34], [38] or an irregular shape [3], [13], [30], [31], [39]. We illustrate rectangular queries in Figures 1b and e and irregular-shaped queries in Figures 1c and f, respectively.
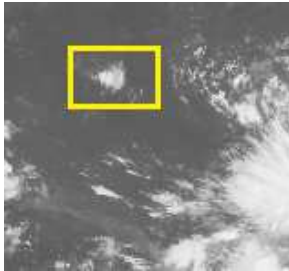
data matrix D	rectangular query	irregular-shaped query
(a) weather satellite image	(b) cloud with background	(c) cloud
(d) satellite map image	(e) junction with background	(f) junction

Fig. 1: Sub-window nearest neighbor search (SWNNS)

Dual-Bound [34] is the state-of-the-art exact method for the SWNNS problem on rectangular queries. The idea is to utilize both lower and upper distance bound functions $(LB(q, c)/UB(q, c))$ for candidates (i.e., sub-windows) such that $LB(q, c) \leq dist(q, c) \leq UB(q, c)$. This method terminates when the smallest upper bound

is less than the lower bounds of all other candidates. In addition, it iteratively refines the bounds of candidates by using a sequence of tighter lower and upper bound functions. However, this solution may invoke a large number of bounding functions per candidate in the worst case, leading to a high cost.
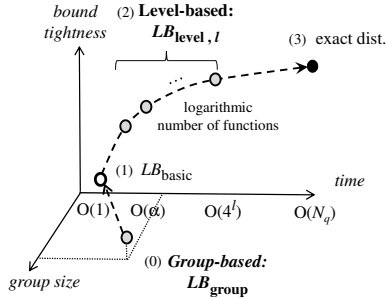

Fig. 2: Illustration of our progressive approach

Our preliminary work [7] focuses on rectangular queries. Specifically, we contribute a solution with a group-based lower bound function LB_{group} and a level-based lower bound function $LB_{level,\ell}$, as shown in Figure 2. Instead of examining candidates individually, we first gather candidates into groups and attempt pruning unpromising groups by using LB_{group}. For the surviving groups, we divide them into smaller groups and repeat the above process. When a group degenerates to a candidate, we attempt pruning it by using $LB_{level,\ell}$. Our $LB_{level,\ell}$ is designed in a fashion such that: (i) it is generic and can take any lower bound function as a building block, (ii) it limits the worst-case cost by using a logarithmic number of levels (for ℓ).

To obtain meaningful results in the aforementioned applications, it is important to ignore irrelevant pixels in the 'background' of a query (e.g., Figure 1b) and exclude them from matching. As such, it is more appropriate to model a query (e.g., cloud, road junction) by an irregular shape, as shown in Figure 1c. The state-of-the-art exact method for this problem [13] is an extended version of Dual-Bound [34]. To cope with an irregular shape, it incrementally partitions candidates into rectangles on-the-fly in order to tighten their lower and upper bounds. However, this solution needs to maintain a set of rectangles for each candidate, thus incurring high overhead on both the memory space and the response time.

Compared to our preliminary work [7], our new contribution is to develop an efficient solution for answering SWNNS on irregular-shaped queries (Section 5). To reduce the memory space for managing candidates, we adopt the same partitioning scheme for all candidates. In this approach, it is desirable to find the optimal partitioning scheme that can minimize the computation cost. We show that it is hard to find the optimal partitioning efficiently, and then propose several heuristics for this issue.

The rest of the paper is organized as follows. Section 2 elaborates on the related work. Section 3 defines our problem and introduces background information. Section 4 presents our proposed solution for rectangular queries.

Section 5 studies the SWNNS problem for queries with irregular shapes. Section 6 discusses our experimental results. Section 7 concludes the paper with future research directions.

2 RELATED WORK

2.1 Nearest Neighbor Search on Feature Vectors

The nearest neighbor (NN) search problem has been extensively studied in multimedia databases [22], [24], [33] and in time series databases [15], [32], [42].

Multimedia databases [22], [24], [33] 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 [4], [21], [33], data compression [41], and hashing [20], [37] 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 (i.e. total number of pixels in query) 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 is only known at the query time). It is infeasible to do precomputation for every possible query size as it would require $O(N_D{}^3)$ storage space, where N_D is the data image size (in pixels).

Generic NN search algorithms [25], [36] are applicable to any types of objects and distance function $dist(q, c)$. Ref. [36] requires using a lower bound function $LB(q, c)$, where $LB(q, c) \leq dist(q, c)$ always holds. Its search strategy [36] 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. [25] 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., our SWNNS problem) are not provided in [25], [36]. In this paper, we focus on developing lower bound functions like $LB_{level,\ell}, LB_{group}$ for the SWNNS problem.

2.2 Similarity Search Methods on Matrix

Similarity search methods on matrix can be classified along two dimensions: (i) whether they support rectangular queries or irregular-shaped queries, and (ii) whether they support range search or NN search.

We first discuss the works for rectangular queries on a matrix. Various lower bound functions [2], [17], [18], [28], [29], [34], [38] have been developed for similarity search problems on a matrix, in order to prune unpromising candidates efficiently and thus avoid expensive distance computations. Ouyang et al. [29] propose a unified framework that covers range search solutions [2], [17], [18], [28], [38]. The state-of-the-art NN search method is [34]. 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 $2^r \times 2^r$ but not arbitrary query size. Also, [34] has not explored our group-based lower bound function LB_{group}, which enables efficient pruning for a group of candidates.

Although the idea of multi-level pruning originates from [17], our method (in Section 4) differs from [17] in two aspects. First, our method is applicable to any rectangular query, but [17] can only handle square queries of the size $\mu^r \times \mu^r$ (where μ and r are integers). Second, as we will explain in Section 4, our multi-level pruning takes a a given lower bound function LB_{basic} as building block. Thus, our method is extensible and can benefit from future developments of LB_{basic}.

Ref. [26] assigns candidates into groups and exploits the similarities of candidates within the same group for pruning. However, it still processes candidates in each group one-by-one. In contrast, our group-based pruning technique enables pruning at the group granularity.

We then discuss the works for irregular-shaped queries on a matrix [3], [13], [30], [31], [39]. Like [3], our method partitions an irregular-shaped query into regions. While Ref. [3] allows only partitioning with square regions of sizes $2^r \times 2^r$, we allow a more flexible partitioning with rectangles. Our partitioning leads to fewer regions than [3] and thus reduces the computation overhead during pruning. Several heuristics [30], [31], [39] have been proposed to compute the results, but they do not always return the best match. The state-of-the-art method for NN search [13] is an extension of [34]. This method decomposes each candidate into a set of disjoint rectangles, and associates each rectangle R_i with a lower bound lb_i and an upper bound ub_i. When it refines the bound of a candidate, it chooses the rectangle with the largest $ub_i - lb_i$, then splits that rectangle based on an entropy idea. While this approach tends to produce a good partitioning for each individual candidate, it incurs high space and time overhead on maintaining the above partitions / rectangles. In contrast, our method eliminates such overhead by using the same partitioning scheme for all candidates.

The above works assume that the image and the query have the same orientation. Several methods have been developed to deal with deformation or rotation of images during matching [6], [10], [23]. A representative method [10] requires solving template matching (SWNNS) as a sub-problem. As such, our proposed method can be applied to speed up the method in [10].

The similarity search on a time series [15], [32], [42] 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 (e.g., $LB_\oplus$) originate from them, our proposed level-based and group-based lower bound functions ($LB_{level,\ell}$, LB_{group}) are specifically designed for the SWNNS problem.

3 PRELIMINARIES

We first give our problem definition and provide background on prefix-sum matrices and lower bound functions.

3.1 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.

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

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

Problem 1 (Sub-window NN Search). *Given a query matrix q and a data matrix D, this problem finds the candidate c_{best} such that it has the minimum $dist(q, c_{best})$, where the distance is the L_p norm:*

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

The value of p is predefined by the application.

Figure 3 shows a query q of size 4×4 and a data matrix D of size 8×8. There are $(8-4+1)^2 = 25$ 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 $c_{3,3}$ as it has the smallest distance $dist(q, c_{3,3}) = 27$ from q.
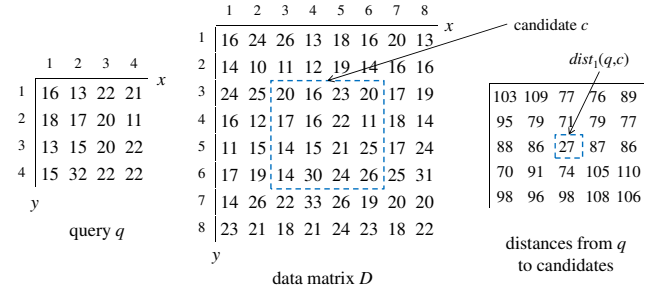

Fig. 3: Example for the problem

3.2 Prefix-Sum Matrix & Basic Lower Bounds

For convenience, we define a shorthand notation below, which will be used in later discussions.

Definition 1 (Accessing a matrix by region). *Let $R = [x_1..x_2, y_1..y_2]$ be a rectangular region and let A be a matrix. The notation $A[R]$ represents $A[x_1..x_2, y_1..y_2]$.*

As we will introduce shortly, lower bound functions require summing the values in a rectangular region in a matrix. We can speed up their computation by using a prefix-sum matrix [19], also known as an integral image [40] in the computer vision community.

Definition 2 (Prefix-sum matrix). *Given a matrix A (of size $N_A = L_A \times W_A$), we define its prefix-sum matrix P_A with entries: $P_A[x, y] = \sum_{i=1}^{x} \sum_{j=1}^{y} A[i, j]$*

The prefix-sum matrix occupies $O(N_A)$ space and takes $O(N_A)$ construction time [19]. It supports the following *region-sum* operation, i.e., finding the sum of values of a rectangular region (say, $R = [x_1..x_2, y_1..y_2]$) in a matrix A, in $O(1)$ time, according to Equation 1.

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

$$(1)$$

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.
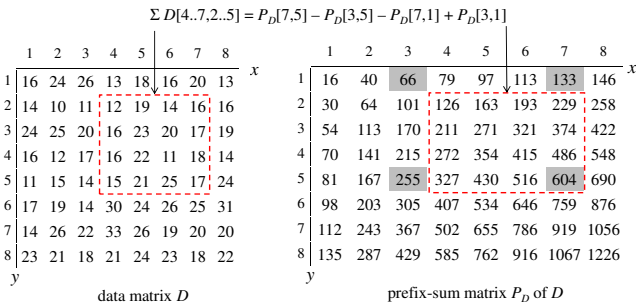
Fig. 4: Example of a prefix-sum matrix

We introduce the basic lower bound function LB_{basic}, which is used as a building block in Figure 2. We require that: (i) $LB_{basic}(q, c) \leq dist(q, c)$ always holds, and (ii) LB_{basic} supports any query size. In this paper, we introduce two functions that satisfy the above requirements of LB_{basic}. The first one ($LB_{\oplus}(q, c)$) is given in [42]. The second one ($LB_{\Delta}(q, c)$) is derived from the triangle inequality of the L_p distance [12], [21]. 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: $D'[i, j] = |D[i, j]|^p$.

$$LB_{\oplus}(q, c) = \frac{\sqrt[p]{N_q}}{N_q} \cdot \left| \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} q[i, j] - \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} c[i, j] \right| \quad (2)$$

$$LB_{\Delta}(q, c) = \left| \sqrt[p]{\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |q[i, j]|^p} - \sqrt[p]{\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |c[i, j]|^p} \right| \quad (3)$$

As a remark, we are aware of lower bound functions used in the pattern matching literature [2], [18], [28], [29], [38]. 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.

4 PROGRESSIVE SEARCH APPROACH

We first present our idea and algorithm in Section 4.1. Then, we elaborate the lower bound functions used in the algorithm in Sections 4.2, 4.3, 4.4.

4.1 The Flow of Proposed Algorithm

We illustrate the flow of our proposed NN search method in Figure 5. Like [25], [36], 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 and (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 region of candidates. Initially, H contains a group entry that represents 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 into several smaller groups G_i. For each G_i, we compute the group-based lower bound $LB_{group}(q, G_i)$ and then enheap G_i into H.
2) If it is a candidate c, then we compute the level-based 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}.
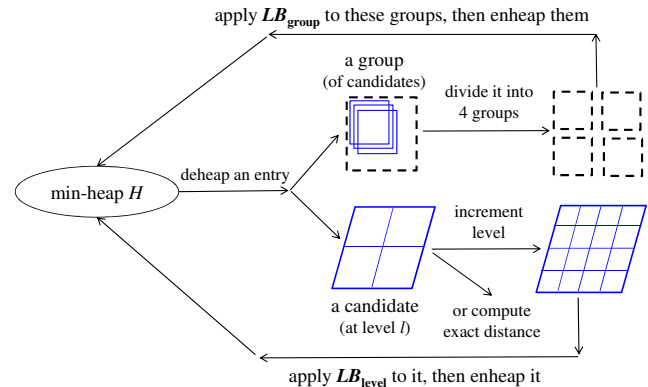
Fig. 5: The flow of our progressive search method

Table 1 lists the lower bound functions to be used in our NN search method. We measure the cost of each function as the number of region-sum operations (i.e., calls to Equation 1). We have introduced LB_{basic} (e.g., $LB_{\Delta}, LB_{\oplus}$) in Section 3.2. We will develop a level-based bound $LB_{level,\ell}$ and a group-based bound LB_{group} in Sections 4.2 and 4.3, respectively. Section 4.4 explores an efficient technique for computing LB_{group}, which involves a tunable parameter α.

We summarize our method in Algorithm 1. Like [25], [36], we employ a min-heap H in order to process entries in ascending order of their lower bound distance. We also

TABLE 1: Types of lower bound functions

Function	Apply to	Cost: # of region-sum operations
Basic: LB_{basic}	candidate	1
Level: $LB_{level,\ell}$	candidate	4^ℓ
Group: LB_{group}	group	α

maintain the best distance found thus far τ_{best} during the search. The algorithm terminates when the deheaped entry's lower bound distance is larger than τ_{best} (Line 10), as the remaining heap entries can only have the same or larger bounds than the deheaped entry. The main difference from [25], [36] is that we apply multiple lower bound functions on candidates (Line 20) and also consider lower bound function for groups of candidates (Lines 6 and 15).

As a remark, at Line 19, $\ell_{\max}$ denotes the maximum possible level, which is computed as follows:

$$\ell_{\max} = \lceil \log_2(\max\{L_q, W_q\}) \rceil \quad (4)$$

Algorithm 1 Progressive Search Algorithm for NN search

```
1: procedure PROGRESSIVE SEARCH(query q, data matrix D)
2:     τ_best ← ∞; e_best ← ∅          ▷ the best entry found so far
3:     create a min-heap H
4:     create a heap entry e_root
5:     e_root.G ← [0..L_D − 1, 0..W_D − 1]     ▷ the entire region
6:     e_root.bound ← LB_group(q, e.G)
7:     enheap e_root to H
8:     while H ≠ ∅ do
9:         e ← deheap an entry in H
10:        if e.bound ≥ τ_best then          ▷ termination condition
11:            break
12:        if |e.G| ≠ 1 then                       ▷ group entry
13:            divide e into 4 entries e_1, e_2, e_3, e_4
14:            for each e_i, i ← 1 to 4 do
15:                e_i.bound ← LB_group(q, e_i.G)
16:                e_i.ℓ ← 0
17:                if e_i.bound < τ_best then    enheap e_i to H
18:        else                                ▷ candidate entry
19:            if e.ℓ < ℓ_max then
20:                e.bound ← LB_level,ℓ(q, e)
21:                increment e.ℓ
22:                if e.bound < τ_best then    enheap e to H
23:            else                            ▷ the deepest level
24:                temp ← dist(q, e)
25:                if temp < τ_best then    τ_best ← temp; e_best ← e
```

4.2 Progressive Filtering for Candidates

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

---

1. In general, the space $[1..L_q, 1..W_q]$ may have less than 4^ℓ disjoint rectangles.

combine these 4^ℓ lower bound distances into $LB_{level,\ell}$ in Equation 5. $LB_{level,\ell}$ takes at most 4^ℓ region-sum operations, as each LB_{basic} takes one region-sum operation.

$$LB_{level,\ell}(q,c) = \Big(\sum_{v=1}^{4^\ell} LB_{basic}(q[R_v], c[R_v])^p \Big)^{1/p} \quad (5)$$

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 $LB_{level,\ell}(q,c)$.

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

Lemma 1. *Let $LB_{basic}(q,c)$ be a lower bound function for $dist(q,c)$. It holds that, $LB_{level,\ell}(q,c) \leq dist(q,c)$, for any candidate c.   [Proved in Ref. [7]]*

Note that [17], [42] have considered a similar lemma, but only for the case where $LB_{basic}(q,c) = LB_\oplus(q,c)$. In contrast, our lemma is applicable to any $LB_{basic}(q,c)$.

During search, we apply $LB_{level,\ell}$ on a candidate c 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 $\ell + 1$.
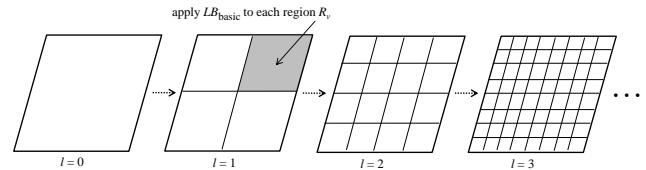

apply LB_{basic} to each region R_v

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

4.3 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. Specifically, we define G as the region $[x_{start}..x_{start} + L_g - 1, y_{start}..y_{start} + W_g - 1]$, where (i) L_g and W_g represent the size of the group, and (ii) x_{start} and y_{start} represent the start position (i.e., 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* of G as $ext_q(G) = [x_{start}..x_{end}, y_{start}..y_{end}]$, where $x_{end} = \min(x_{start} + L_g + L_q - 2, L_D)$ and $y_{end} = \min(y_{start} + W_g + W_q - 2, W_D)$. Then, $D[ext_q(G)] = D[x_{start}..x_{end}, y_{start}..y_{end}]$ represents the submatrix of D in the region $ext_q(G)$.

Our lower bound functions require the following concepts.

Definition 3 (The lowest/highest k elements in $D[ext_q(G)]$). *We define $\mathcal{L}_k(D[ext_q(G)])$ and $\mathcal{H}_k(D[ext_q(G)])$ as the **lowest and highest** k **elements** in the submatrix $D[ext_q(G)]$ **respectively**.*

Definition 4 (Summation of the lowest/highest k elements in $D[ext_q(G)]$). *We define $\mathcal{SL}_k(D[ext_q(G)])$ as **the sum of lowest** k **elements** in $D[ext_q(G)]$, and*
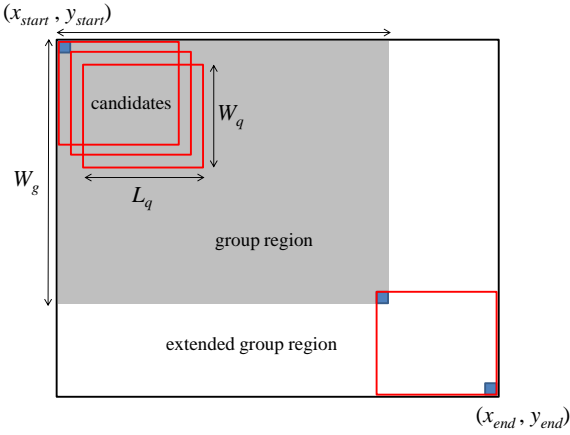
Fig. 7: A group with $L_g \times W_g$ consecutive candidates

$\mathcal{SH}_k(D[ext_q(G)])$ as **the sum of highest k elements** in $D[ext_q(G)]$.

We illustrate these concepts in Figure 8. Assume that the query size is $N_q = 2 \times 2 = 4$. Consider the group $G = [2..5, 2..5]$ (as dotted square) and the extended region $ext_q(G) = [2..6, 2..6]$ (as bolded square). In this example, the lowest N_q values in $D[ext_q(G)]$ are: $\mathcal{L}_{N_q}(D[ext_q(G)]) = \{9, 9, 10, 10\}$. Thus, $\mathcal{SL}_{N_q}(D[ext_q(G)]) = 9 + 9 + 10 + 10 = 38$.
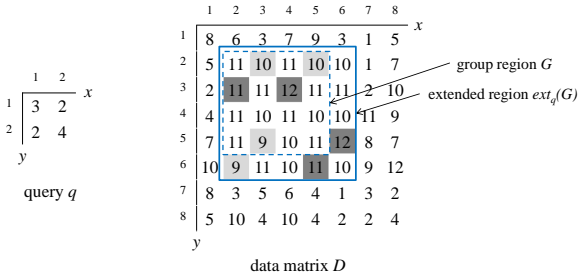

Fig. 8: Illustration of $\mathcal{L}_{N_q}(D[ext_q(G)])$ (in light color) and $\mathcal{H}_{N_q}(D[ext_q(G)])$ (in dark color)
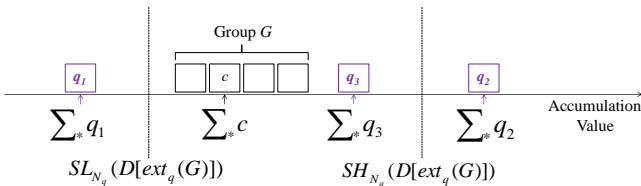

Fig. 9: Illustration of the idea in $LB_{group}^{\oplus}(q, G)$

We then extend basic lower bound functions (e.g., $LB_{\oplus}, LB_{\Delta}$) for a group G. We propose the lower bound functions $LB_{group}^{\oplus}$ and LB_{group}^{Δ} for G in Equations 6, 7. In Equation 7, the term $D^{\circ p}$ denotes the element-wise power of the matrix D with power index p, i.e., $D^{\circ p}[i, j] = (D[i, j])^p$. These functions serve as lower bounds of $LB_{\oplus}(q, c), LB_{\Delta}(q, c)$ for any candidate c in G (cf. Lemmas 2,3).

Lemma 2. *Given a group G, for any candidate c in G, we have: $LB_{group}^{\oplus}(q, G) \leq LB_{\oplus}(q, c)$.*

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

The proofs of Lemmas 2 and 3 can be found in our preliminary work [7]. Figure 9 explains why $LB_{group}^{\oplus}(q, G)$ is a lower bound function. We use three query points q_1, q_2, q_3 (with same size N_q) to illustrate the three cases in $LB_{group}^{\oplus}(q, G)$, respectively. For convenience, we drop the subscript N_q in the notations $\mathcal{SL}$ and $\mathcal{SH}$. By Equation 2, the lower bound between query q and candidate c depends on two summation terms ($\sum_* q$ and $\sum_* c = \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} c[i, j]$). The latter term $\sum_* c$ is always bounded between $\mathcal{SL}(D[ext_q(G)])$ and $\mathcal{SH}(D[ext_q(G)])$, provided that c is a member of the group G. For example, for query q_1, the lower bound distance is the difference between $\sum_* q_1$ and $\mathcal{SL}(D[ext_q(G)])$. For query q_2, it is symmetric to the above case, so the lower bound is the difference between $\sum_* q_2$ and $\mathcal{SH}(D[ext_q(G)])$. For query q_3, the lower bound distance is zero because $\sum_* q_3$ falls into the range between $\mathcal{SL}(D[ext_q(G)])$ and $\mathcal{SH}(D[ext_q(G)])$. The above idea can also be applied to $LB_{group}^{\Delta}(q, G)$.

During our search procedure (cf. Figure 5 and Algorithm 1), we apply $LB_{group}(q, G)$ on a group G. If we cannot filter G, then we partition its group region G 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. We will discuss how to compute $LB_{group}(q, G)$ efficiently in the next subsection.

4.4 Supporting Group Filtering Efficiently

The lower bound $LB_{group}(q, G)$ involves the terms $\mathcal{SL}_{N_q}(D[ext_q(G)])$ and $\mathcal{SH}_{N_q}(D[ext_q(G)])$ (Equation 6) or $\mathcal{SL}_{N_q}(D^{\circ p}[ext_q(G)])$ and $\mathcal{SH}_{N_q}(D^{\circ p}[ext_q(G)])$ (Equation 7), which require finding the lowest N_q and the highest N_q values in $D[ext_q(G)]$ or $D^{\circ p}[ext_q(G)]$.

In this section, we design a data structure called *prefix histogram matrix* to support the above operations efficiently. The parameter α allows trade-off between the running time 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 $\beta(D[i, j])$:

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

where D_{min} and D_{max} denote the minimum and maximum values in D, respectively. Consider one example using Figure 8, $D[2, 2] = 11$, $D_{min} = 1$ and $D_{max} = 12$ in this case. We can notice that $\beta(D[2, 2]) = 6$ when we set $\alpha = 6$ bins.

We define the *prefix histogram matrix* PH_{β} as a matrix where each element $PH_{\beta}[i, j]$ is a count histogram:

$$PH_{\beta}[i, j] = \langle P_1[i, j], P_2[i, j], \cdots, P_{\alpha}[i, j] \rangle$$

where

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

$$LB^{\oplus}_{group}(q,G) = \begin{cases} \frac{\sqrt[p]{N_q}}{N_q}(\mathcal{SL}_{N_q}(D[ext_q(G)]) - \sum_* q) & \text{if } \mathcal{SL}_{N_q}(D[ext_q(G)]) > \sum_* q \\ \frac{\sqrt[p]{N_q}}{N_q}(\sum_* q - \mathcal{SH}_{N_q}(D[ext_q(G)])) & \text{if } \mathcal{SH}_{N_q}(D[ext_q(G)]) < \sum_* q \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

$$LB^{\Delta}_{group}(q,G) = \begin{cases} \sqrt[p]{\mathcal{SL}_{N_q}(D^{\circ p}[ext_q(G)])} - \sqrt[p]{\sum_* |q[i,j]|^p} & \text{if } \mathcal{SL}_{N_q}(D^{\circ p}[ext_q(G)]) > \sum_* |q[i,j]|^p \\ \sqrt[p]{\sum_* |q[i,j]|^p} - \sqrt[p]{\mathcal{SH}_{N_q}(D^{\circ p}[ext_q(G)])} & \text{if } \mathcal{SH}_{N_q}(D^{\circ p}[ext_q(G)]) < \sum_* |q[i,j]|^p \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$LB'^{\oplus}_{group}(q,G) = \begin{cases} \frac{\sqrt[p]{N_q}}{N_q}(\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) - \sum_* q) & \text{if } \mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) > \sum_* q \\ \frac{\sqrt[p]{N_q}}{N_q}(\sum_* q - \mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]})) & \text{if } \mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]}) < \sum_* q \\ 0 & \text{otherwise} \end{cases} \quad (8)$$

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

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

Figure 10a illustrates a histogram matrix PH_β in which each element $PH_\beta[i,j]$ stores a count histogram for values in region $[1..i, 1..j]$ in the data matrix D.

(a) prefix histogram matrix PH_β

(b) count histogram for $D[ext_q(G)]$
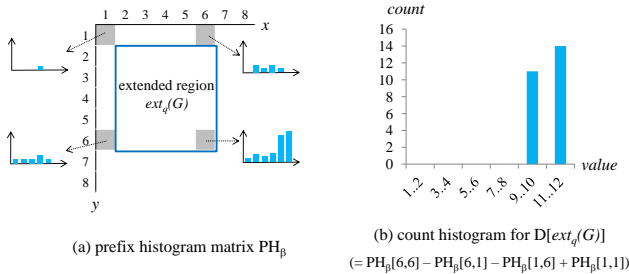$(= PH_\beta[6,6] - PH_\beta[6,1] - PH_\beta[1,6] + PH_\beta[1,1])$

Fig. 10: Prefix histogram matrix, $\alpha = 6, D_{min} = 1, D_{max} = 12$

Given an extended group region $ext_q(G)$, we first retrieve count histograms at four corners of $D[ext_q(G)]$, and then combine them into the histogram as shown in Figure 10b. With this histogram, we can derive bounds for the sum of minimum / maximum N_q values of $D[ext_q(G)]$ i.e. $\mathcal{SL}_{N_q}(D[ext_q(G)])$ and $\mathcal{SH}_{N_q}(D[ext_q(G)])$ by Definition 5.

Definition 5 (Sum of the lowest / highest N_q values in a count histogram). *Let $CH_{D[ext_q(G)]}$ be a count histogram for $D[ext_q(G)]$. We define $\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]})$ as the sum of the lowest N_q values in $CH_{D[ext_q(G)]}$, and $\mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]})$ as the sum of the highest N_q values in $CH_{D[ext_q(G)]}$.*

While scanning the bins of $CH_{D[ext_q(G)]}$ from left to right, we examine the count and the minimum bound of each bin to derive $\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]})$. A similar method can be used to derive $\mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]})$. The cost of computing a group-based lower bound equals to α region-sum operations because $CH_{D[ext_q(G)]}$ contains α bins and each bin requires 1 region-sum operation to compute.

As an example, consider the count histogram $CH_{D[ext_q(G)]}$ obtained in Figure 10b. 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: $\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) = 9 \cdot 4 = 36$. Note that $\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) = 36$ is looser than the actual value $\mathcal{SL}_{N_q}(D[ext_q(G)]) = 38$ (obtained in Figure 8). Then we propose $LB'^{\oplus}_{group}(q,G)$ in Equation 8 to replace $LB^{\oplus}_{group}(q,G)$.

Since $\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) \leq \mathcal{SL}_{N_q}(D[ext_q(G)])$ and $\mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]}) \geq \mathcal{SH}_{N_q}(D[ext_q(G)])$, $LB'^{\oplus}_{group}(q,G) \leq LB^{\oplus}_{group}(q,G)$. Similarly, we can adapt the above technique to derive a lower bound of $LB^{\Delta}_{group}(q,G)$ efficiently.

5 EXTENSION FOR IRREGULAR-SHAPED QUERIES

As discussed in the introduction, some applications may need to deal with irregular-shaped queries. For example, in geospatial data integration [8], [9], [11], the query can be a road junction which may have a T-shape. In cloud motion detection [5], the query can be an irregular cloud. Figure 11 illustrates the differences between rectangular queries and irregular-shaped queries. For each irregular-shaped query, we employ a binary mask matrix to indicate irrelevant pixels [13]. The binary mask matrix can be extracted by image segmentation methods or by application requirements [5], [9], [39].

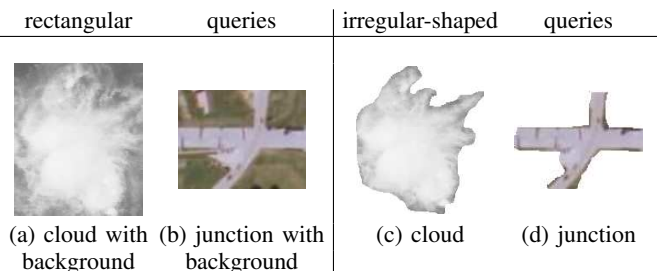| rectangular	queries	irregular-shaped	queries
(a) cloud with background	(b) junction with background	(c) cloud	(d) junction

Fig. 11: Examples of irregular-shaped queries

Problem 2 (Sub-window NN Search for Irregular-Shaped Query). *Given a query matrix q, a binary mask matrix m, and a data matrix D, this problem finds the candidate*

c_{best} *such that it has the minimum* $dist^\diamond(q, c_{best})$ *where the distance is defined as:*

$$dist^\diamond(q,c) = \Big(\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} m[i,j] \cdot |q[i,j] - c[i,j]|^p \Big)^{1/p} \quad (9)$$

We illustrate this problem in Figure 12. In the mask, relevant entries have $m[i,j] = 1$ and irrelevant entries have $m[i,j] = 0$. The best match is indicated by the candidate in a dashed square.
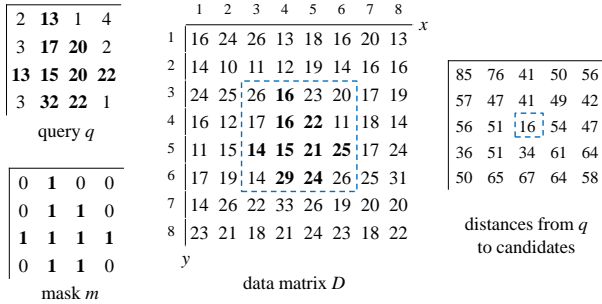

Fig. 12: Example for the irregular-shaped query

We will present two approaches in extending our progressive search method to solve the above problem. First, we propose an intuitive extension in Section 5.1. Second, we develop a more efficient extension by partitioning the mask in Section 5.2.

5.1 Is Progressive Search still applicable?

Recall that our progressive search method (Algorithm 1) applies two lower bound functions $LB_{level,\ell}$ and LB_{group}. The correctness of the algorithm depends on whether both $LB_{level,\ell}$ and LB_{group} satisfy the lower bound property. In the following, we demonstrate that, for the case of irregular-shaped query, (i) LB_{group} can be slightly modified to satisfy the lower bound property, and (ii) $LB_{level,\ell}$ violates the lower bound property.

We can modify LB_{group} (Equations 6,7) in order to satisfy the lower bound property. Intuitively, we replace N_q (i.e., the size of q) by the number of relevant entries in the mask matrix m. For this purpose, we define the set of relevant entries as

$$M_q = \{(i,j) : m[i,j] = 1\} \quad (10)$$

By using M_q, we revise the equations for LB_{group} into $LB_{group}^{\oplus,\diamond}(q,G)$ and $LB_{group}^{\triangle,\diamond}(q,G)$, in Equations 11 and 12, respectively. We omit the proofs of their lower bound property as they are similar to the proofs of Lemmas 2 and 3. Figure 13 illustrates how to compute $LB_{group}^{\oplus,\diamond}(q,G)$. Note that there are $|M_q| = 5$ relevant entries in q. For the group G, we indicate the lowest 5 and the highest 5 entries in light gray and dark gray, respectively. Then we obtain: $LB_{group}^{\oplus,\diamond}(q,G) = (|12 + 14 + 16 + 16 + 16| - |7 + 5 + 5 + 5 + 5|) = 47$.

However, it is not trivial to simply extend $LB_{level,\ell}$. We provide an example to show that $LB_{level,\ell}$ can violate the lower bound property. Consider the candidate $c_{3,3}$ (in

a dashed square) in Figure 12 and assume $p = 1$. By Equation 9, the exact distance is: $dist^\diamond(q, c_{3,3}) = 16$. For the lower bound distance, suppose that we use $LB_\oplus$ as an instance of LB_{basic}. At level $\ell = 0$, we compute: $LB_{level,0}(q, c_{3,3}) = LB_\oplus(q, c_{3,3}) = |\sum_* q - \sum_* c_{3,3}| = |190 - 319| = 129$. This violates the lower bound property as $LB_{level,0}(q, c_{3,3}) > dist^\diamond(q, c_{3,3})$. This happens because LB_{basic} considers all entries (including irrelevant entries) in a candidate. To prevent such violation, a simple solution is to disable $LB_{level,\ell}$.

We then summarize how to extend our progressive search algorithm (Algorithm 1) for irregular-shaped queries. First, we disable $LB_{level,\ell}$ by removing Lines 18–21. Second, we replace LB_{group} by $LB_{group}^\diamond$ at Lines 6 and 15. Third, we replace $dist(q,c)$ by $dist^\diamond(q,c)$ at Line 23, and compute it efficiently by Equation 13.

$$dist^\diamond(q,c) = \Big(\sum_{(i,j)\in M_q} |q[i,j] - c[i,j]|^p \Big)^{1/p} \quad (13)$$

5.2 Extension for $LB_{level,\ell}$ based on Partitioning

To achieve efficient extension of Algorithm 1, it is important to develop a replacement for the level-based lower bound $LB_{level,\ell}$.

We plan to decompose the mask m into a set of disjoint rectangles. To enable the lower bound property, we should use a partition that covers no '0'-entry of m. We formally define a valid partition as follows.

Definition 6 (Valid partition). *Let* Γ *be a set of disjoint rectangles, where each rectangle* $R \in \Gamma$ *can be described by* $[R.x_{start}..R.x_{end}, R.y_{start}..R.y_{end}]$. *Given a mask matrix* m, *we call* Γ *a valid partition if,* $\forall R \in \Gamma$, $\forall (i,j) \in R$, $m(i,j) = 1$.

Figure 14 illustrates a mask m and a valid partition of three rectangles: $\Gamma = \{[1..1, 3..3], [2..3, 2..4], [4..4, 3..3]\}$. Note that a valid partition cannot cover any '0'-entry of m.

Given a valid partition Γ of a mask m, we define the lower bound function $LB_\Gamma(q,c)$ in Equation 14.

$$LB_\Gamma(q,c) = \Big(\sum_{R\in\Gamma} LB_{basic}(q[R], c[R])^p \Big)^{1/p} \quad (14)$$

Since each term $LB_{basic}(q[R], c[R])$ takes one region-sum operation, the cost of computing $LB_\Gamma(q,c)$ equals to $|\Gamma|$ region-sum operations.

Then we prove that $LB_\Gamma(q,c)$ satisfies the lower bound property (cf. Lemma 4).

Lemma 4. $LB_\Gamma(q,c) \leq dist^\diamond(q,c)$.

Proof:
$$
\begin{aligned}
dist^\diamond(q,c)^p &= \sum_{i=1}^{L_q}\sum_{j=1}^{W_q} m[i,j]|q[i,j]-c[i,j]|^p \\
&\geq \sum_{R\in\Gamma}\sum_{(i,j)\in R} m[i,j]|q[i,j]-c[i,j]|^p \\
&\geq \sum_{R\in\Gamma} LB_{basic}(q[R],c[R])^p \\
&= LB_\Gamma(q,c)^p
\end{aligned}
$$

$$LB_{group}^{\oplus,\diamond}(q,G) = \begin{cases} \frac{\sqrt[p]{|M_q|}}{|M_q|}(\mathcal{SL}_{|M_q|}(D^{\circ 1}[ext_q(G)]) - \sum_{M_q} q) & \text{if } \mathcal{SL}_{|M_q|}(D^{\circ 1}[ext_q(G)]) > \sum_{M_q} q \\ \frac{\sqrt[p]{|M_q|}}{|M_q|}(\sum_{M_q} q - \mathcal{SH}_{|M_q|}(D^{\circ 1}[ext_q(G)])) & \text{if } \mathcal{SH}_{|M_q|}(D^{\circ 1}[ext_q(G)]) < \sum_{M_q} q \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$LB_{group}^{\Delta,\diamond}(q,G) = \begin{cases} \sqrt[p]{\mathcal{SL}_{|M_q|}(D^{\circ p}[ext_q(G)])} - \sqrt[p]{\sum_{M_q} |q[i,j]|^p} & \text{if } \mathcal{SL}_{|M_q|}(D^{\circ p}[ext_q(G)]) > \sum_{M_q} |q[i,j]|^p \\ \sqrt[p]{\sum_{M_q} |q[i,j]|^p} - \sqrt[p]{\mathcal{SH}_{|M_q|}(D^{\circ p}[ext_q(G)])} & \text{if } \mathcal{SH}_{|M_q|}(D^{\circ p}[ext_q(G)]) < \sum_{M_q} |q[i,j]|^p \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\text{where } \mathcal{L}_{|M_q|}(D[G.R^{ext}]) \text{ is the lowest } |M_q| \text{ values in the submatrix } D[G.R^{ext}] \tag{13}$$

$$\mathcal{SL}_{|M_q|}(D^{\circ \omega}[ext_q(G)]) = \sum_{v \in \mathcal{L}_{|M_q|}(D[G.R^{ext}])} v^{\omega}$$
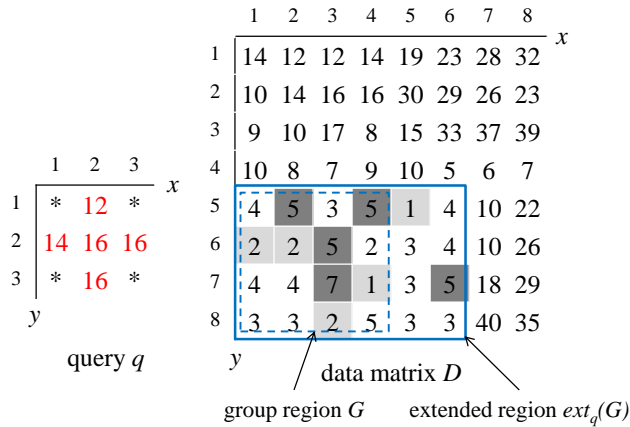

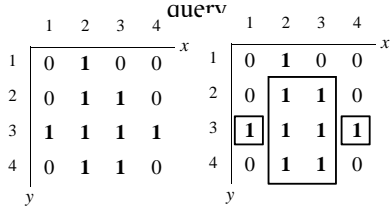Fig. 13: Group-based lower bound for irregular-shaped query

Fig. 14: A valid partition Γ of a mask m

In general, we may employ a sequence of valid partitions $\langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell_{max}^{\diamond}-1} \rangle$ with an increasing number of rectangles, where $\ell_{max}^{\diamond}$ denotes the number of levels. During search, we apply LB_{Γ_ℓ} on a candidate c in the ascending order of ℓ as shown in Figure 15. If we cannot filter c at level ℓ, then we attempt to filter it with minimal extra effort, i.e., at level $\ell + 1$.
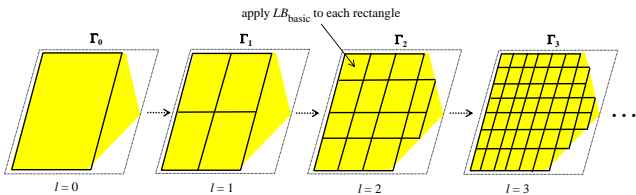

Fig. 15: Level ℓ in irregular partition plan

We propose to apply the same sequence Γ_{seq} for all candidates. This would eliminate the overhead of on-the-fly partitioning and allow us to manage each candidate with

$O(1)$ space only (i.e., the current lower bound and level of the candidate).

We then discuss the extension to the progressive search algorithm. First, before Line 1, we construct a sequence of valid partitions $\Gamma_{seq} = \langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell_{max}^{\diamond}-1} \rangle$ from the mask m, by using heuristics to be discussed in Section 5.2.2. Second, at Line 19, we replace $LB_{level,\ell}(q,e)$ by $LB_{\Gamma_\ell}(q,e)$.

5.2.1 Cost Formulation and Hardness

We first formulate the computation cost of our algorithm. The cost depends on a query matrix q, a binary mask matrix m, a data matrix D, and a sequence of partitions $\Gamma_{seq} = \langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell_{max}^{\diamond}-1} \rangle$. To simplify our analysis, we disable group-based pruning and measure the computation cost as the total number of rectangles used in calling $LB_{\Gamma_\ell}(q,c), dist^{\diamond}(q,c)$ only.

Given a candidate c of D, we have:

$$cost(LB_{\Gamma_\ell}(q,c)) = |\Gamma_\ell|$$
$$cost(dist^{\diamond}(q,c)) = |M_q|$$

where M_q was defined in Equation 10.

We denote the NN distance by $\tau_{opt} = \min_c dist^{\diamond}(q,c)$. order of lower bound distance until reaching τ_{opt}. For simplicity, we can assume that τ_{opt} is known in advance in this model. If a candidate c can be pruned before or at level $\ell_{max}^{\diamond} - 1$, then it incurs cost $\sum_{\ell=0}^{F(q,c)} |\Gamma_\ell|$ only, where $F(q,c)$ denotes the last level for computing the lower bound $LB_{\Gamma_\ell}(q,c)$:

$$F(q,c) = \min(\{\ell : LB_{\Gamma_\ell}(q,c) \geq \tau_{opt}\} \cup \{\ell_{max}^{\diamond} - 1\})$$

In addition, we must compute the exact distance $dist^{\diamond}(q,c)$ for the following subset of candidates:

$$C_{exact} = \{c : F(q,c) = \ell_{max}^{\diamond}-1, LB_{\Gamma_{\ell_{max}^{\diamond}-1}}(q,c) < \tau_{opt}\}$$

In summary, the total cost of Γ_{seq} is:

$$cost(\Gamma_{seq}, q, m, D) = \sum_c \sum_{\ell=0}^{F(q,c)} |\Gamma_\ell| + |C_{exact}||M_q|$$

Generally, we wish to find the best sequence $\langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell_{max}^{\diamond}-1} \rangle$ that minimizes $cost(\Gamma_{seq}, q, m, D)$. Although our solution computes

Γ_{seq} only once, we cannot afford to spend too much time (e.g., more than $O(N_D N_q)$ which is the time complexity of brute force method) to compute Γ_{seq}. Unfortunately, we will show that this problem is NP-hard, let alone to solve it in $O(N_D N_q)$ time.

In Theorem 1, we will show that the decision version of our problem (in Definition 7) is NP-hard via reduction from a known NP-complete problem called the *Rectilinear Picture Compression* (RPC) decision problem [16] (p.232) (in Definition 8).

Definition 7 (Γ-decision problem).
INSTANCE: $\langle q, D, m, \ell_{max}^{\diamond}, LB_{basic}, K \rangle$, *where* q, D *are matrices*, m *is a binary matrix*, $\ell_{max}^{\diamond}, K$ *are integers, and* LB_{basic} *is a basic lower bound function.*
PROBLEM: *Is there any sequence* $\Gamma_{seq} = \langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell_{max}^{\diamond}-1} \rangle$ *such that it satisfies Definition 6 and* $cost(\Gamma_{seq}, q, m, D) \leq K$?

Definition 8 (RPC-decision problem).
INSTANCE: $\langle K', m'[1..n, 1..n] \rangle$, *where* K', n *are integers, and* m' *is a* $n \times n$ *binary matrix.*
PROBLEM: *Is there any set* S *of disjoint rectangles* $\{R_1, R_2, ...\}$ *that satisfies both conditions below?*
- $|S| \leq K'$
- $\bigcup_{R_z \in S} R_z = \{(i, j) : m'[i, j] = 1\}$

Theorem 1. *The Γ-decision problem is NP-hard.*

Proof:
First, we present the **reduction scheme** from the RPC-decision problem to our Γ-decision problem.
- Set $m[1..n, 1..n]$ to $m'[1..n, 1..n]$
- Set $q[1..n, 1..n]$ with all '0' entries
- Set $D[1..n, 1..n]$ with all '1' entries
- Set K to K', set $\ell_{max}^{\diamond}$ to 1, and set LB_{basic} to $LB_{\oplus}$

The above reduction scheme takes polynomial time.

We proceed to show that the RPC-decision instance returns true *if and only if* the Γ-decision instance returns true. For convenience, we define the notation $M_q = \{(i, j) : m[i, j] = 1\}$. Since D has only one candidate c, we obtain:
$$\tau_{opt} = dist^{\diamond}(q, c) = \left(\sum_{(i,j) \in M_q} |0 - 1|^p \right)^{\frac{1}{p}} = |M_q|^{\frac{1}{p}}.$$

If the RPC-decision instance returns true, then there exists a set S of disjoint rectangles such that:
- $|S| \leq K' = K$
- $\bigcup_{R_z \in S} R_z = M_q$ (since $m = m'$)

Thus, S also satisfies the valid partition condition in Definition 6. We then set $\Gamma_0 = S$ and plan to show that Γ-decision instance returns true. By Equation 14, we derive:

$$
\begin{aligned}
(LB_{\Gamma_0}(q, c))^p &= \sum_{R_z \in \Gamma_0} LB_{\oplus}(q[R_z], c[R_z])^p \\
&= \sum_{R_z \in \Gamma_0} \left(\frac{\sqrt[p]{|R_z|}}{|R_z|} \Big| \sum_{(i,j) \in R_z} q[i,j] - c[i,j] \Big| \right)^p \\
&= \sum_{R_z \in \Gamma_0} \left(\frac{\sqrt[p]{|R_z|}}{|R_z|} |R_z| \right)^p = \sum_{R_z \in \Gamma_0} |R_z| = |M_q|
\end{aligned}
$$

We get $LB_{\Gamma_0}(q, c) \geq \tau_{opt}$ and thus $F(q, c) = 0$. Then we obtain: $cost(\Gamma_{seq}, q, m, D) = |\Gamma_0| + 0 \leq K$. Therefore, the Γ-decision instance returns true.

If the Γ-decision instance returns true, then there exists $\langle \Gamma_0 \rangle$ such that $cost(\Gamma_{seq}, q, m, D) \leq K$. Since D has only one candidate, we have two cases to consider:

Case when $|C_{exact}| = 1$
We have: $cost(\Gamma_{seq}, q, m, D) = |\Gamma_0| + |M_q| \leq K$. Since $K = K'$ and $|\Gamma_0| \geq 0$, we get $|M_q| \leq K'$. We then set $S = \{[i..i, j..j] : m[i, j] = 1\}$. Since $|M_q| \leq K'$ and $m = m'$, we infer that S covers m' exactly and thus the RPC-decision instance returns true.

Case when $|C_{exact}| = 0$
We have: $cost(\Gamma_{seq}, q, m, D) = |\Gamma_0| \leq K = K'$. Since $|C_{exact}| = 0$, we derive: $LB_{\Gamma_0}(q, c) \geq \tau_{opt}$. By the lower bound property of LB_{Γ_0}, we get $\tau_{opt} = dist^{\diamond}(q, c) \geq LB_{\Gamma_0}(q, c)$. Thus, we obtain $LB_{\Gamma_0}(q, c) = \tau_{opt}$. By substituting q, c, τ_{opt} into the above equation, we get:
$$\sum_{R_z \in \Gamma_0} |R_z| = |M_q|.$$

By Definition 6, Γ_0 contains disjoint rectangles that cover only '1'-entries. Combining this fact with the above equation, we infer that Γ_0 covers all '1'-entries in M_q (i.e., in m').

Finally, we set $S = \Gamma_0$. The RPC-decision instance returns true because we have shown that: (i) $|\Gamma_0| \leq K'$, and (ii) Γ_0 covers all '1'-entries in m'. □

5.2.2 Split-and-Mend Partitioning

In this section, we present several $O(N_q)$-time heuristics for partitioning a mask m at level ℓ. We propose a *split-and-mend* strategy to obtain good partitioning heuristics. First, we apply 'split' to divide m into at most 4^{ℓ} rectangles. Second, we apply 'mend' to ensure that each rectangle is valid (cf. Definition 6).

We consider two 'split' heuristics based on tree structures for 2D points:
- **Quad-tree split**: We build a level-ℓ quad-tree on m, and then output each leaf node as a rectangle.
- **KD-tree split**: We build a level-2ℓ KD-tree on '1'-entries of m. Note that the KD-tree divides m by the x-axis and the y-axis in an alternate manner. Then, we output each leaf node as a rectangle.

We show the results of Quad-tree split at level $\ell = 1$ in Figure 16a and KD-tree split at level $\ell = 2$ in Figure 16b, respectively.

Suppose that, after splitting, we obtain the rectangles in Figure 17a. However, the top-right and the bottom-left rectangles are invalid because they cover some '0'-entries in m. We then suggest 'mend' heuristics on the above rectangles in order to generate a valid partition (cf. Definition 6).
- **Drop**: We simply drop invalid rectangles, as shown in Figure 17b.
- **Grow**: Consider the bottom-left invalid rectangle in Figure 17c. We choose a '1'-entry (in gray color) and then find the maximal rectangle containing it.

While 'Drop' returns fewer rectangles, 'Grow' tends to produce rectangles that lead to tighter bounds. We will compare them in the experimental study.
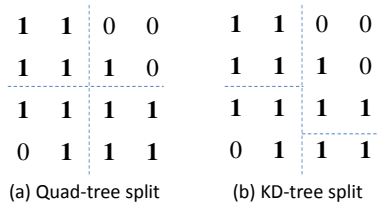
$$\begin{array}{cc|cc}
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1
\end{array}
\qquad
\begin{array}{cc|cc}
1 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1
\end{array}$$

(a) Quad-tree split   (b) KD-tree split

Fig. 16: Examples on split

(a) after splitting   (b) refine by dropping   (c) refine by growing
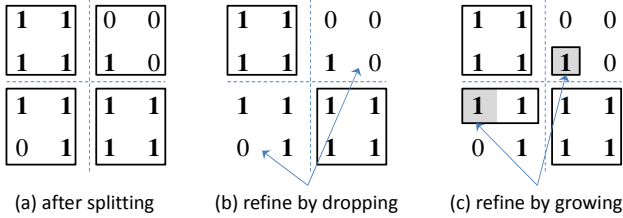
Fig. 17: Examples on mend

Based on the split-and-mend strategy, we obtain four combinations of heuristics for partitioning the mask: Quad-Drop, Quad-Grow, KD-Drop and KD-Grow.

6 EXPERIMENTAL EVALUATION

We present our experiments for rectangular queries and irregular-shaped queries in Sections 6.1 and 6.2, respectively. We implemented all algorithms in C++ and conducted experiments on an Intel i7 3.4GHz PC running Ubuntu.

6.1 Experiments for Rectangular Queries

6.1.1 Experimental Setting

We summarize our methods and the state-of-the-art [34] (denoted as Dual) in Table 2a. We label our *progressive search* methods with the same prefix PS. Their suffixes represent which techniques are used.

- PSL applies LB_{level} only, and
- PSLG applies both LB_{level} and LB_{group}.

The subscripts (e.g., $\oplus$ or Δ) indicate whether their lower bound functions are built on top of $LB_{\oplus}$ or LB_{Δ}.

TABLE 2: The list of our methods and the competitors

Method	Techniques used
Dual [34]	[34]
$PSL_{\oplus}$	Section 4.2
PSL_{Δ}	Section 4.2
$PSLG_{\oplus}$	Sections 4.2 and 4.3

(a) methods for rectangular queries

Method	Techniques used
iDual [13]	[13]
iPSG	Section 5.1
$iPSL_{quad,drop}$, $iPSL_{quad,grow}$ $iPSL_{kd,drop}$, $iPSL_{kd,grow}$	Section 5.2
$iPSLG_{quad,drop}$, $iPSLG_{quad,grow}$ $iPSLG_{kd,drop}$, $iPSLG_{kd,grow}$	Sections 5.1, 5.2

(b) methods for irregular-shaped queries

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.

For example, the preprocessing time is only 0.22s per image for the Weather dataset in Table 3.

Table 3a lists the details of our datasets and queries. We collect these datasets from [1], [29]. *Photo640, Photo1280* and *Photo2560* [29] contain 30 images of the size 640×480, 1280×960 and 2560×1920 respectively. *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. Since our competitors only support the L_2 norm, we use the L_2 norm in all experiments.

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

TABLE 3: Our datasets and queries

| Dataset	Image size	Number of images	Number of queries per image
Photo2560	2560×1920	30	10
Photo1280	1280×960	30	10
Photo640	640×480	30	10
Weather	1800×1800	30	10

(a) the setting for rectangular queries

Dataset	Number of images	Number of queries per image	Query extraction method
Photo2560	30	10	Matlab segmentation
Weather	30	1	Manual extraction

(b) the setting for irregular-shaped queries

6.1.2 Results

First, we study the effect of the number of bins α on the response time of our method $PSLG_{\oplus}$. Figure 18 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) Photo2560   (b) Weather
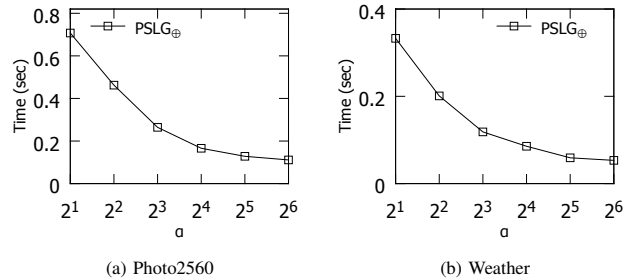
Fig. 18: Effect of the number of bins α

We have also collected measurements to study the effectiveness of techniques in $PSLG_{\oplus}$, at the default setting ($\alpha = 16$). First, the exact distance calculation incurs only 5% of the running time, whereas the computation of bounds incurs 95% of the running time. Second, the majority of candidates (99%) are pruned at the group level and the remaining candidates are pruned at the candidate level.

Next, we evaluate the scalability of methods with respect to the query size N_q. Figure 19 shows the response time

of methods versus the query size N_q. Since Dual [34] 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, $\text{PSLG}_\oplus$ outperforms all other methods and scales better with respect to N_q. On the other hand, Dual, $\text{PSL}_\triangle$ and $\text{PSL}_\oplus$ need to obtain candidates one-by-one and incur higher overhead on maintaining the min-heap. Since $\text{PSL}_\oplus$ performs better than $\text{PSL}_\triangle$, we omit $\text{PSL}_\triangle$ in the next experiment.
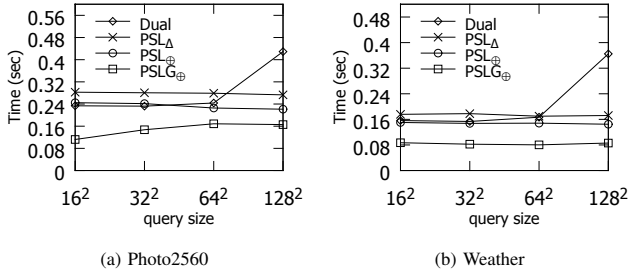

(a) Photo2560　　　　　　　　(b) Weather

Fig. 19: Effect of the query size N_q

Then, we test the scalability of methods with different data sizes (by using three datasets Photo640, Photo1280 and Photo2560), while fixing the query size to 64×64. Figure 20 shows the response time of methods with respect to the data size. Our methods perform better than the competitor Dual. When the data size increases, our group-based pruning technique becomes more powerful and thus the gap between $\text{PSLG}_\oplus$ and the other methods widens.
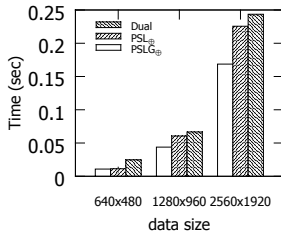

Fig. 20: Effect of the data size N_D

To test the robustness of methods, we follow [34] and add Gaussian noise into each query image. The query size is fixed to 128×128 in this experiment. Figure 21 shows the response time of methods as a function of the noise (in standard deviation). The performance gap between our methods and Dual widens as the noise increases. At a high noise, the pruning power of all lower bound functions becomes weaker. In the worst-case, Dual may invoke a long sequence of bounding functions per candidate, whereas our methods invoke at most a logarithmic number of LB_{level} (in terms of N_q) per candidate. In summary, our methods are more robust than Dual against noise.

6.2 Experiments for Irregular-Shaped Queries

6.2.1 Experimental Setting

We summarize our methods and the state-of-the-art [13] (denoted as iDual) in Table 2b. We label our methods with the same prefix iPS. Their suffixes (G or L or both) represent which lower bound techniques are used. Their subscripts represent which partitioning techniques are used.
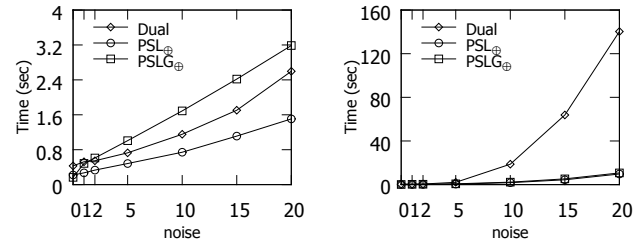
(a) Photo2560　　　　　　　　(b) Weather

Fig. 21: Effect of the noise

Table 3b lists the details of our datasets and queries. The experiments in [13] have tested with three synthetic query shapes only. In contrast, we test with a wider variety of query shapes in our experiments. For the Photo2560 dataset [29], we apply the Matlab segmentation function on each rectangular query in order to obtain an irregular-shaped query. For the Weather dataset [1], we follow the same approach as in [39] and manually extract a cloud pattern from each data image.

The response time of our iPSLG methods includes the partitioning time. In all of our experiments, the partitioning time is at most 1.2% of the response time only, implying that our partitioning heuristics incur very low overhead.

6.2.2 Results

We first compare the effectiveness of our partitioning heuristics and name these methods as $\text{iPSL}_{kd,drop}$, $\text{iPSL}_{quad,drop}$ $\text{iPSL}_{kd,grow}$ and $\text{iPSL}_{quad,grow}$. Figures 22a and b plot the response time of these methods with respect to the Gaussian noise as described in Section 6.1.2. In general, 'grow' is better than 'drop' because 'grow' can produce more rectangles and thus provide tighter bounds. On the other hand, 'quad' performs slightly better than 'kd'. The best method $\text{iPSL}_{quad,grow}$ is faster than others up to 20% and 40%, on the Photo2560 and the Weather datasets, respectively.

In Section 5.2, we have formulated a cost equation to express the computation cost of our method. We then measure this cost in the above experiment and show it in Figures 22c and d. We observe that the trends are similar to those in Figures 22a and b. Again, $\text{iPSL}_{quad,grow}$ achieves the lowest cost, and it performs better than other methods by up to 37% and 53%, on the Photo2560 and the Weather datasets, respectively.

In the next experiment, we compare the competitor (iDual) with three variants of our methods: one using group-based lower bound (iPSG), one using level-based lower bound ($\text{iPSL}_{quad,grow}$), and one using both types of lower bounds ($\text{iPSLG}_{quad,grow}$). Figure 23a and b show the response time of these methods as a function of the noise. iDual is the worst since it incurs high overhead on maintaining a set of rectangles for each candidate. Both $\text{iPSL}_{quad,grow}$ and $\text{iPSLG}_{quad,grow}$ outperform iPSG, implying that our lower bounds in Section 5.2 are more powerful than the simple bound in Section 5.1. We then plot the maximum heap space of these methods, in terms of the

(a) response time on Photo2560

(b) response time on Weather
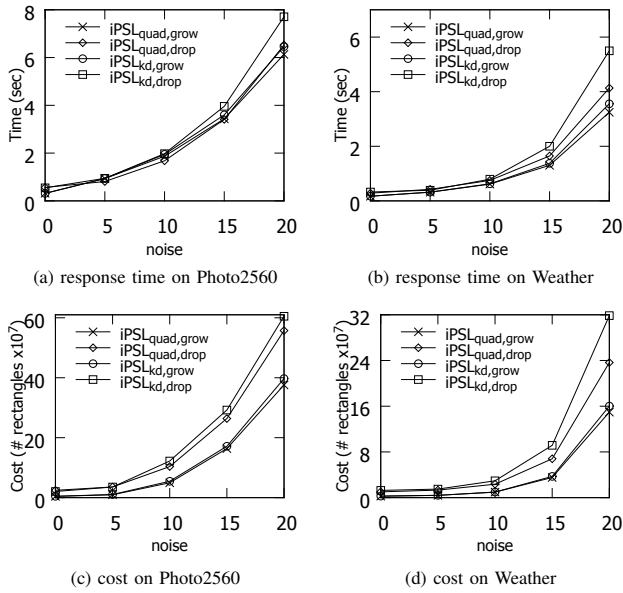
(c) cost on Photo2560

(d) cost on Weather

Fig. 22: Comparisons of partitioning heuristics, varying the noise

number of rectangles, in Figure 23c and d. iDual occupies considerable amount of space on maintaining rectangles for candidates. $iPSL_{quad,grow}$ requires only $O(1)$ space per candidate. Since $iPSLG_{quad,grow}$ can perform group-based pruning, it consumes the smallest amount of space.

(a) response time on Photo2560

(b) response time on Weather

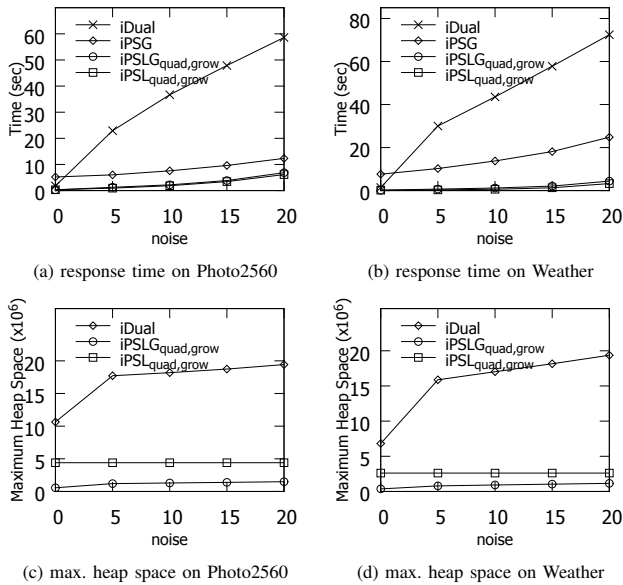(c) max. heap space on Photo2560

(d) max. heap space on Weather

Fig. 23: Comparisons of methods, varying the noise

Finally, we test the effect of the query size on the response time of our method $iPSL_{quad,grow}$ and the competitor iDual, without noise. We measure the query size as the number of '1'-entries in the mask. The sizes of queries range from 960 to 16384 in the Photo2560 dataset and from 5400 to 84710 in the Weather dataset. We plot the results in Figure 24. Both the average response time and the worst-case response time of $iPSL_{quad,grow}$ outperform iDual significantly.

(a) $iPSL_{quad,grow}$ on Photo2560

(b) iDual on Photo2560

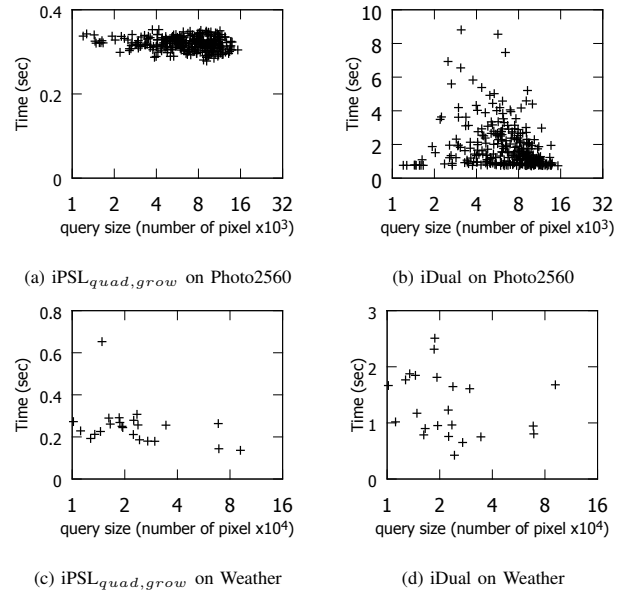(c) $iPSL_{quad,grow}$ on Weather

(d) iDual on Weather

Fig. 24: Effect of query size, fixing $\sigma = 0$

7 CONCLUSION

The contribution of our work is twofold. First, the proposed technique can support irregular-shaped queries. This new flexibility makes the new solution much more effective. Second, this new advantage is achieved with substantially less computation in comparison with the current state of the art, about 20 times faster when the noise level is low to medium and at least 9 times faster when the noise level is high. Our experiments on real datasets indicate that the proposed method is capable of real-time computation and therefore enables a wide range of new applications not possible before. In the future, we plan to investigate approximation algorithms to further reduce the running time with theoretical guarantee.

ACKNOWLEDGEMENT

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]   M. Ben-Yehuda, L. Cadany, and H. Hel-Or. Irregular pattern matching using projections. In *ICIP*, pages 834–837, 2005.
[4]   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.
[5]   R. Brad and I. A. Letia. Extracting cloud motion from satellite image sequences. In *ICARCV*, pages 1303–1307, 2002.
[6]   Y. Cai and G. Baciu. Detecting, grouping, and structure inference for invariant repetitive patterns in images. *IEEE Trans. Image Processing*, 22(6):2343–2355, 2013.
[7]   T. N. Chan, M. L. Yiu, and K. A. Hua. A progressive approach for similarity search on matrix. In *SSTD*, pages 373–390, 2015.
[8]   C. Chen, C. A. Knoblock, and C. Shahabi. Automatically conflating road vector data with orthoimagery. *GeoInformatica*, 10(4):495–530, 2006.
[9]   C. Chen, C. Shahabi, and C. A. Knoblock. Utilizing road network data for automatic identification of road intersections from high resolution color orthoimagery. In *STDBM 04*, pages 17–24, 2004.

[10] M. Cheng, F. Zhang, N. J. Mitra, X. Huang, and S. Hu. Repfinder: finding approximately repeated scene elements for image editing. *ACM Trans. Graph.*, 29(4):83:1–83:8, 2010.

[11] Y. Chiang, C. A. Knoblock, C. Shahabi, and C. Chen. Automatic and accurate extraction of road intersections from raster maps. *GeoInformatica*, 13(2):121–157, 2009.

[12] 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.

[13] R. Deng. *Fast Matching Techniques Utilizing Integral Images*. PhD thesis, University of Texas at Dallas, 2011.

[14] 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.

[15] 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.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[17] 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.

[18] 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.

[19] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *SIGMOD*, pages 73–88, 1997.

[20] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.

[21] 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.

[22] D. A. Keim and B. Bustos. Similarity search in multimedia databases. In *ICDE*, page 873, 2004.

[23] Y. Kong, W. Dong, X. Mei, X. Zhang, and J. Paul. Simlocator: robust locator of similar objects in images. *The Visual Computer*, 29(9):861–870, 2013.

[24] 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.

[25] 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.

[26] A. Mahmood and S. Khan. Exploiting transitivity of correlation for fast template matching. *IEEE Trans. Image Processing*, 19(8):2190–2200, 2010.

[27] 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.

[28] 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.

[29] 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.

[30] O. Pele and M. Werman. Accelerating pattern matching or how much can you slide? In *ACCV*, pages 435–446, 2007.

[31] O. Pele and M. Werman. Robust real-time pattern matching using bayesian sequential hypothesis testing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(8):1427–1443, 2008.

[32] 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.

[33] H. Samet. Techniques for similarity searching in multimedia databases. *PVLDB*, 3(2):1649–1650, 2010.

[34] 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.

[35] A. Seering, P. Cudré-Mauroux, S. Madden, and M. Stonebraker. Efficient versioning for scientific array databases. In *ICDE*, pages 1013–1024, 2012.

[36] T. Seidl and H. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.

[37] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[38] 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.

[39] D. Truong, V. Nguyen, A. D. Duong, C. N. Ngoc, and M. Tran. Realtime arbitrary-shaped template matching process. In *ICARCV*, pages 1407–1412, 2012.

[40] P. A. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[41] 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.

[42] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB*, pages 385–394, 2000.

Tsz Nam Chan received the bachelor's degree in 2014 from Hong Kong Polytechnic University. He is currently a PhD student in Hong Kong Polytechnic University, under the supervision of Dr. Man Lung Yiu. His research interests include multidimensional similarity search and pattern matching.

Man Lung Yiu received the bachelor's degree in computer engineering and the PhD degree in computer science from the University of Hong Kong in 2002 and 2006, respectively. Prior to his current post, he worked at Aalborg University for three years starting in the Fall of 2006. He is now an associate professor in the Department of Computing, The Hong Kong Polytechnic University. His research focuses on the management of complex data, in particular query processing topics on spatiotemporal data and multidimensional data.

Kien A. Hua is a Professor of Computer Science, and Director of the Data Systems Lab at the University of Central Florida. He served as the Associate Dean for Research of the College of Engineering and Computer Science at UCF. Prior to joining the university, he was a lead architect at IBM Mid-Hudson Lab, where he helped develop a highly parallel computer system, the precursor to the highly successful commercial parallel computer known as SP2. Dr.Hua received his B.S. in Computer Science, and M.S. and Ph.D. in Electrical Engineering, all from the University of Illinois at Urbana-Champaign. His diverse expertise includes network and wireless communications, image/video computing, sensor networks, medical imaging, databases, mobile computing, and intelligent transportation systems. He has published widely, with over 10 papers recognized as best/top papers at conferences and a journal. Many of his research have had significant impact on society. His Chaining technique started the peer-to-peer video streaming paradigm. His Skyscraper Broadcasting, Patching, and Zigzag techniques have been heavily cited in the literature, and have inspired many commercial systems in use today. Dr. Hua has served as a Conference Chair, an Associate Chair, and a Technical Program Committee Member of numerous international conferences, as well as on the editorial boards of a number of professional journals. Dr. Hua is a Fellow of IEEE.