THE HONG KONG
POLYTECHNIC UNIVERSITY
香港理工大學
Pao Yue-kong Library
包玉剛圖書館

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

Pao Yue-kong Library, The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

http://www.lib.polyu.edu.hk

SIMILARITY MEASURES: ALGORITHMS AND APPLICATIONS

CHAN TSZ NAM

PhD
The Hong Kong Polytechnic University

2019

The Hong Kong Polytechnic University

Department of Computing

Similarity Measures: Algorithms and Applications

Tsz Nam CHAN

A thesis submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

July 2018

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

. . . .

Tsz Nam CHAN

July 2018

Abstract

Similarity measures are the basic components for various problems such as image processing, computer vision, pattern recognition and machine learning problems. However, evaluating the similarity measures is normally the bottleneck for many applications. In this thesis, we highlight three computational intensive applications and propose efficient algorithms in these scenarios.

The first application is object detection in images. Given a query image, this problem finds the most similar sub-image within a given target image. The problem can be formulated as the nearest neighbor search problem. In the context of computer vision, we also call this the template matching problem. The Euclidean distance is used to measure the dissimilarity between the query image and a sub-image. However, the time complexity of object detection for each query is the product of the sizes of sub-image and image, which is prohibited for fast object detection scenario. We propose two solutions which can significantly outperform the state-of-the-art method by 9-20 times faster.

The second application is image retrieval. Existing image retrieval systems extract the feature histograms for all images. During the online phase, image retrieval systems return the k most similar images for each online image-query from the user. One robust similarity measure between two histograms is based on the Earth Mover's Distance (EMD). However, due to the cubic time complexity for evaluating EMD, it restricts the applicability to small-scale datasets. We present the approximation framework that leverages on lower and upper bound functions to compute approximate EMD with error guarantee. Under this framework, we present two solutions which can significantly outperform the existing exact or heuristic solutions. Our experimental studies demonstrate that our best solution can outperform the existing method by 2.38x to 7.26x times faster.

The third application is (kernel) classification. In machine learning context, kernel function is the similarity measure between two multidimensional vectors, which are extracted by different feature extraction methods, based on different scenarios. Many machine learning models need to compute the weighted aggregation of kernel function values with respect to a set of multidimensional vectors and the query vector, using different types of kernel functions, for example: Gaussian, Polynomial or Sigmoid kernels. However, computing the online kernel aggregation function is normally expensive which limits its applicability for some real-time (e.g. network anomaly detection) or large-scale (e.g. density estimation/ classification for physical modeling) applications. We propose novel and effective bounding techniques to speed up the computation of kernel aggregation. We further boost the efficiency by leveraging index structures and exploiting index tuning opportunities. Experimental studies on many real datasets reveal that our proposed method achieves speedups of 2.5-738x over the state-of-the-art.

Acknowledgements

First, I would like to express my deepest sense of gratitude to my advisors, Dr. Ken Yiu, for his patient guidance in these four years. He brings me to explore the research world and provides me with many advices during my research study. Without him, I would not have completed this research work.

Next, I would like to thanks Hui Li (HKU) and Prof. Nikos Mamoulis, who are my collaborators in the paper ”FEXIPRO: Fast and Exact Inner Product Retrieval in Recommender Systems”. Their written paper skills gives me insight on how to write a good research paper. Moreover, this work provides me insight for our research work in ”KARL: Fast Kernel Aggregation Queries”.

Later, I would like to thanks Dr. Leong Hou U (Ryan), who is my collaborator in two papers ”The Power of Bounds: Answering Approximate Earth Mover’s Distance with Parametric Bounds” and ”KARL: Fast Kernel Aggregation Queries”. Ryan’s comments can make me think deeper about my research work, especially for the topic in Earth Mover’s Distance.

After that, I would also like to thanks Prof Kien. A. Hua who is my collaborator in two research papers of template matching (Similarity Search on Matrix).

Prof Hua's comments make our research papers more readable.

During my research study. I have learnt many different knowledge and have got inspired from different research students during my study. Thanks to Ziqiang Feng, Andy He, Capital Li, Hui Li (PolyU), Qiang Li, Wengen Li, Yu Li, Petrie Wong, Chuanfei Xu, Wenjian Xu and Qiang Zhang for their comments and encouragement. I am glad to have many joyful moments with all of them.

Lastly, I need to thank my parents, especially for my mum and dad. Without them, I am not able to see the beautiful world, not to mention conducting research.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Similarity measures are ubiquitous in different types of applications in computer vision [91, 102] and machine learning [124, 107]. In these contexts, image pixel values [91], or feature histograms [124, 102, 107] are regarded as the vectors and several similarity measures (e.g. Euclidean distance [91]) can be used to measure the similarity between two objects.

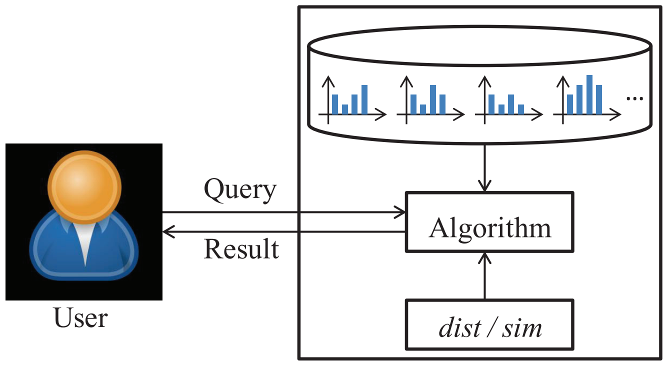


Figure 1.1. Framework

We show the framework (c.f. Figure 1.1) which summarizes how users retrieve the results from a set of vectors based on different distance or similarity functions. The vector representation depends on the application which will be summarized in Section 2.1. Our goal is to design efficient algorithms in order to support different queries issued by the users, for example: nearest neighbor search (c.f. Chapters 3 and 4) or kernel classification (c.f. Chapter 5).

Our framework (c.f. Figure 1.1) can be instantiated to support three applications in computer vision and machine learning, as shown in Table 1.1.

Table 1.1. Three applications for framework 1.1

Our users	Problem	Similarity Measure
Computer vision engineers / scientists	Template matching (SWNNS)	Euclidean distance
Layman	kNN image retrieval	Earth mover's distance
Computer servers	kernel aggregation query	Equation 2.4 + Table 2.2

Sub-Window Nearest Neighbor Search (a.k.a Template Matching, abbrev. SWNNS) [91] is the fundamental problem in computer vision, for example: object detection [18], motion estimation [88] and image editing [30]. Given any sub-image query, our goal is to find the most similar sub-image window inside the image, as shown in Figure 1.2. The image sub-window can be either rectangular-shaped [49, 55, 12, 120, 90, 109, 91, 92, 19, 20, 93] or irregular-shaped [13, 96, 39, 121, 20]. However, existing algorithms are normally inefficient [91], especially for some real-time applications, for example: motion estimation or object detection. Since we only need to return the nearest sub-window inside the image, exact algorithm with different bound functions can be used to boost up the efficiency performance.

Another application is kNN-Image retrieval/classification (c.f. Figure 1.3)

data matrix	rectangular query	irregular-shaped query
(a) weather satellite image	(b) cloud with background	(c) cloud

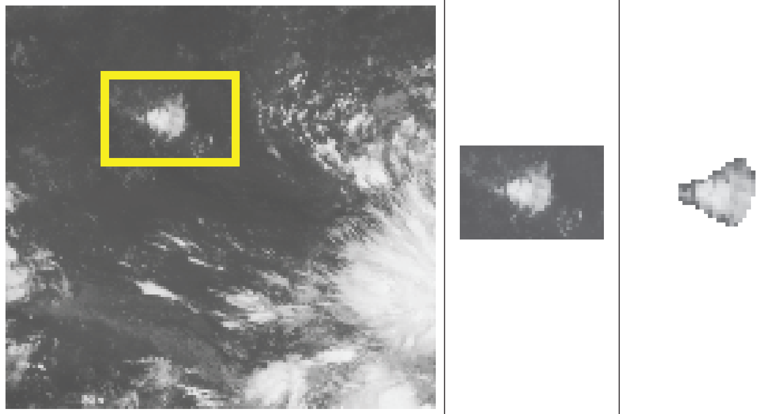


Figure 1.2. Sub-window nearest neighbor search (SWNNS)

[102], the goal is to retrieve the k most similar images for each image query from the users. Earth Mover's Distance (EMD) [102] is currently the most robust similarity measure for many feature extraction methods, compared with different other similarity measures [100], for example: Euclidean distance. However, EMD is a computational intensive operation. Even with the fastest known algorithm [89], it requires $O(d^3 \log d)$ time to compute the exact EMD value, where d is the dimensionality (i.e., number of histogram bins). With the large-scale datasets, exact EMD computation for solving kNN image retrieval/ classification problem is infeasible. The inefficiency issues are pointed out by the existing literatures, which are quoted as follows:

- *"Typically, the EMD between two histograms is modeled and solved as a linear optimization problem, the min-cost flow problem, which requires super-cubic time. The high computational cost of EMD restricts its applicability* *to datasets of low-scale."* [117]

- *"Computing the EMD entails finding a solution to the transportation problem, which is computationally intensive"* [103]

Due to the inefficiency issues, it is natural to ask: whether we can obtain the approximate value of EMD with theoretical guarantee. As such, approximate algorithms are developed to boost up the performance of computing EMD in this application.

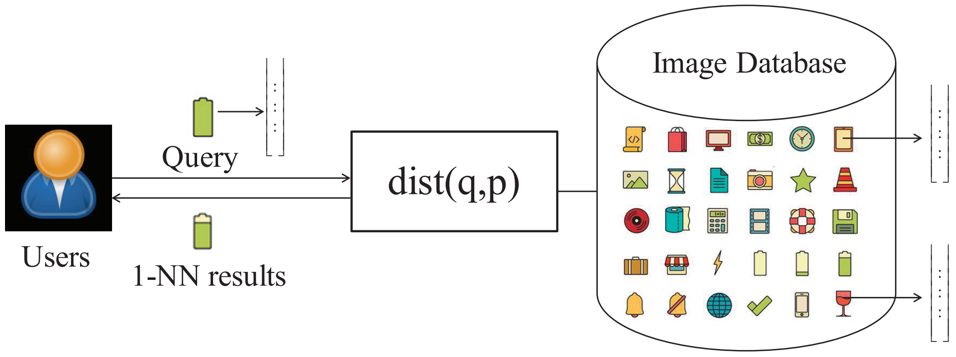


Figure 1.3. 1-NN image retrieval

In the machine learning context, Kernel methods [107] have been extensively used in many applications, such as document classification [86], network fault detection [15, 17, 134], anomaly/outlier detection [23, 80], novelty detection [108, 84, 58], image classification [29, 43], time series classification [69] and density estimation for astronomy [4]. In the above applications, a common online operation is to compute the following function:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \exp(-\gamma \cdot K(\mathbf{q}, \mathbf{p_i})) \tag{1.1}$$

where $\mathbf{q}$ is a query point, P is a dataset of points, w_i, γ are scalars, and $K(\mathbf{q}, \mathbf{p})$ denotes the kernel function between $\mathbf{q}, \mathbf{p_i}$. Kernel function will be discussed in Chapter 2. Figure 1.4 plots the function $\mathcal{F}_P(\mathbf{q})$ for each possible query point $\mathbf{q}$. In this example, $\mathcal{F}_P(\mathbf{q})$ is contributed by the sum of three terms that depend on three data points, respectively. A typical problem, which we term as the *threshold kernel aggregation query* (τKAQ), is to test whether $\mathcal{F}_P(\mathbf{q})$ is higher than a given threshold τ [107]. This creates an opportunity for achieving speedup. Instead of computing the exact $\mathcal{F}_P(\mathbf{q})$, it suffices to compute lower/upper bounds of $\mathcal{F}_P(\mathbf{q})$ and then compare them with the threshold τ. However, the above query is expensive as it takes $O(nd)$ time to compute $\mathcal{F}_P(\mathbf{q})$ online, where d is the dimensionality of data points and n is the cardinality of the dataset P. In the machine learning community, many recent works [78, 57, 68] also complain the inefficiency issue for computing kernel aggregation, which are quoted as follows:

- *"Despite their successes, what makes kernel methods difficult to use in many large scale problems is the fact that computing the decision function is typically expensive, especially at prediction time."* [78]
- *"However, computing the decision function for the new test samples is typically expensive which limits the applicability of kernel methods to real-world applications."* [57]
- *"..., it has the disadvantage of requiring relatively large computations in the testing phase"* [68]

Therefore, due to the inefficiency issues in different problems, we need to utilize our framework (c.f. Figure 4.6) to boost up the efficiency performance. Our goal is to derive fast group bounds $LB(\mathbf{q},G)$, $UB(\mathbf{q},G)$ and new indexing structures. Moreover, we also provide approximation method to boost up the performance.

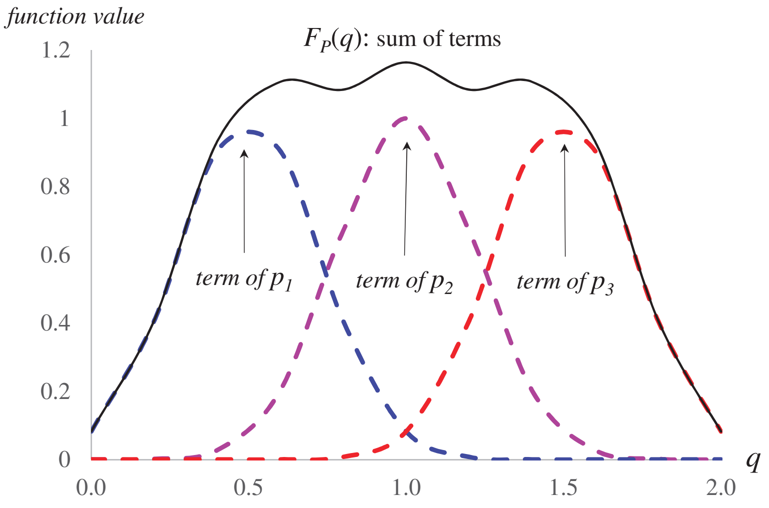


Figure 1.4. Example of $\mathcal{F}_P(\mathbf{q})$

Our main contributions are summarized as follows in different problems:

In the SWNNS problem, we first devise both the progressive individual-based and group-based lower bounding functions $LB_{level,\ell}$ and LB_{group}. Later, we propose the novel prefix-histogram indexing structure which efficiently supports our LB_{group} function. Our techniques can simultaneously support both regular-shaped and irregular-shaped image query.

In the Earth Mover's Distance (EMD) problem, we first propose an approximation framework that leverages lower and upper bound functions to compute approximate EMD with error guarantee. Later, we devise the novel progressive lower and upper bound functions for Earth Mover's Distance and different Kernel functions. We also present two approximation algorithms, under the proposed approximation framework, which are Skew Adaptive and Hybrid Adaptive to significantly speedup the computation time of approximate EMD.

In the kernel aggregation query problem, we devise the novel lower and upper bound functions for different kernel functions. Then, we also develop automatic index-tuning algorithm which automatically chooses the best index from the existing indices (kd-tree / ball-tree) with the most suitable leaf size. Next, our technique can also support the online-stages of different machine-learning/ statistical models, such as: one-class SVM, two-class SVM, kernel density estimation/ classification with different kernel functions. Lastly, our method can support the in-situ scenario in which the model (e.g., dataset P) would be updated frequently.

Chapter 2 elaborates the literature review. Chapter 3 (based on [19, 20]) summarizes the work on SWNNS problem. Chapter 4 (based on [21]) summarizes the work on Approximate EMD. Chapter 5 (based on [22]) summarizes the work on kernel computation problem. Chapter 6 concludes the thesis and discusses some future research directions.

Chapter 2

Literature Review

In this chapter, we first review the literature related to different similarity measures. Then, we point out three main types of queries which involve the similarity measures. Next, we mention the bounding functions for different types of similarity measures. Lastly, we summarize different indexing structures for multi-dimensional data to boost up the efficiency performance.

2.1 Representation of Vectors in Different Applications

The vectors $\mathbf{q}$ and $\mathbf{p}$ can have different meanings in different application scenarios. We denote the dimension of each feature vector to be d in this thesis. We also use $q[i]$ and $p[i]$ to denote the i^{th}-dimension value of vectors $\mathbf{q}$ and $\mathbf{p}$ respectively.

In image retrieval [102] or recommender system [79] or classification [107] contexts, these vectors are the feature vectors extracted by feature extraction algorithms. For example, Lab feature extraction method is used for image representation [102, 97]. Some machine learning models, such as, matrix factorization [75] can create feature vectors to represent each object (e.g. video/ movie) in recommender system.

In template matching [91] context, these vectors are represented by the raw image pixel values.

In time series retrieval [44] context, these vectors are represented by the time series value for each time stamp.

2.2 Types of Similarity Measures

Table 2.1 shows a wide range of similarity functions in the literatures which can be applied in different applications.

Table 2.1. Existing similarity measures

Similarity Measure	Application(s)	Time Complexity
Euclidean distance [106, 91]	image/ time series retrieval, template matching	$O(d)$
inner product [79]	recommender system	$O(d)$
cosine similarity [7],[127]	text retrieval template matching	$O(d)$
gaussian/ polynomial / sigmoid kernel [107]	machine learning	$O(d)$
dynamic time warping [98]	time series retrieval	$O(d^2)$
earth mover's distance [102]	image retrieval	$O(d^3 \log d)$

Euclidean distance (c.f. Equation 2.1) is the traditional similarity measure for a wide range of applications, for example: image retrieval application [76, 71, 106], template matching [91] and time series retrieval [131].

$$ED(\mathbf{q},\mathbf{p}) = \sum_{i=1}^{d} (q[i]-p[i])^2 \tag{2.1}$$

Inner product (c.f. Equation 2.2) is the important similarity measure for recommender systems, [35] to recommend the products to users.

$$IP(\mathbf{q},\mathbf{p}) = \sum_{i=1}^{d} q[i]p[i] \tag{2.2}$$

In text retrieval and template matching, cosine similarity is a popular similarity measure.

$$COS(\mathbf{q},\mathbf{p}) = \frac{\sum_{i=1}^{d} q[i]p[i]}{\sqrt{\sum_{i=1}^{d} q[i]^2}\sqrt{\sum_{i=1}^{d} p[i]^2}} \tag{2.3}$$

In machine learning community, kernel functions are applicable in different machine learning/statistical models, such as: kernel support vector machine [107] and kernel density estimation [124]. The most common kernel functions in the machine learning context are summarized in Table 2.2.

Table 2.2. Kernel functions

Kernel function	Equation
Gaussian	$K_{Gauss}(\mathbf{q},\mathbf{p}) = \exp(-\gamma ED(\mathbf{q},\mathbf{p})^2)$
Polynomial	$K_{Poly,deg}(\mathbf{q},\mathbf{p}) = (\gamma IP(\mathbf{q},\mathbf{p}) + \beta)^{deg}$
Sigmoid	$K_{Sig}(\mathbf{q},\mathbf{p}) = tanh(\gamma IP(\mathbf{q},\mathbf{p}) + \beta)$

The online operation is to compute the weighted aggregation of kernel function values with respect to a set P ($|P| = n$) of d-dimensional vectors and query vector $\mathbf{q}$ [107], which are defined as follows. Let $K(\mathbf{q}, \mathbf{p_i})$ be any kernel function in Table 2.2.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{i=1}^{n} w_i K(\mathbf{q}, \mathbf{p_i}) \tag{2.4}$$

In Equation 2.4, the weight vectors w_i depend on the nature of different models (Table 5.1).

Table 2.3. Types of weighting in $\mathcal{F}_P(\mathbf{q})$

Type of weighting	Used in model
Type I: identical, positive w_i (most specific)	Kernel density [50, 46]
Type II: positive w_i (subsuming Type I)	1-class SVM [86]
Type III: no restriction on w_i (subsuming Types I, II)	2-class SVM [107]

In time series retrieval application, Dynamic Time Warping (DTW) is another famous distance function for measuring the difference between two time series. Unlike previous similarity measures. The dimensionalities of different time series can be different. Equation 2.5 [114] defines this function, we denote the dimensionalities of $\mathbf{q}$ and $\mathbf{p}$ to be d_1 and d_2 respectively.

$$DTW(q, p) = f_{DTW}(d_1, d_2) \tag{2.5}$$

where:

$$f_{DTW}(i,j) = (q[i]-p[j])^2 + \begin{cases} min \begin{cases} f_{DTW}(i-1,j) \\ f_{DTW}(i,j-1) \\ f_{DTW}(i-1,j-1) \end{cases} & \text{if } i \neq 0, j \neq 0 \\ 0 & \text{if } i=0, j=0 \\ \infty & \text{if } i=0 \text{ xor } j=0 \end{cases} \tag{2.6}$$

In the late 90s, Rubner [101] et al. propose earth mover's distance (EMD) to measure the similarity between the feature vectors of images. They experimentally demonstrate that EMD is more robust compared with traditional Euclidean distance for image retrieval application [101, 102, 100]. EMD (c.f. Equation 2.2) is formulated as the following linear programming problem [101]. The time complexity of state-of-the-art method [89] for computing EMD is $O(d^3 \log d)$.

$$\begin{aligned} emd_c(\mathbf{q},\mathbf{p}) = \quad & \underset{f}{\text{minimize}} \sum_{i=1}^{d}\sum_{j=1}^{d} c_{i,j} f_{i,j} \\ \text{such that} \quad & \forall i,j \in [1..d] : f_{i,j} \geq 0 \\ & \forall i \in [1..d] : \sum_{j=1}^{d} f_{i,j} = q[i] \\ & \forall j \in [1..d] : \sum_{i=1}^{d} f_{i,j} = p[j] \end{aligned}$$

2.3 Queries with Similarity Measures

In this section, we discuss several queries that involve similarity measures.

Definition 2.1 *(Range Search Query) Given a query* $\mathbf{q}$*, the set* P *of vectors and the threshold* τ*. We denote dist (for example: ED) and sim (for example: IP) the dissimilarity and similarity measures respectively. Our goal is to find the vectors* $\mathbf{p}$ *that* $dist(\mathbf{q}, \mathbf{p}) \leq \tau$ *or* $sim(\mathbf{q}, \mathbf{p}) \geq \tau$*.*

Definition 2.2 *(kNN Similarity Search Query) Given a query* $\mathbf{q}$ *and the set* P *of vectors, we need to find the k-most similar vectors from the set* P *from query* $\mathbf{q}$.

For the Kernel Aggregation Query, this is only used for Equation 2.4.

Definition 2.3 *(Kernel Aggregation Query-*τ*) Given a query* $\mathbf{q}$ *and the set* P *of vectors and the classification threshold* τ*, we classify* $\mathbf{q}$ *to be either 1 or -1, according to the following equation.*

$$Class(\mathbf{q}) = \begin{cases} 1 & \text{if } \mathcal{F}_P(\mathbf{q}) \geq \tau \\ -1 & \text{otherwise} \end{cases} \tag{2.7}$$

Definition 2.4 *(Kernel Aggregation Query-*ϵ*) Given a query* $\mathbf{q}$ *and the set* P *of vectors and the relative error value* ϵ*, this problem returns an approximate value* $\widehat{F}$ *such that its relative error (from the exact value* $\mathcal{F}_P(\mathbf{q})$*) is at most* ϵ*, i.e.,*

$$(1 - \epsilon)\mathcal{F}_P(\mathbf{q}) \leq \widehat{F} \leq (1 + \epsilon)\mathcal{F}_P(\mathbf{q}) \tag{2.8}$$

2.4 Bound Functions for Similarity Measures

In different similarity measures, different properties can be exploited to derive the bounding functions. Therefore, we summarize different bound techniques for the similarity measures in this section. In this section, we use the following concepts and notations.

Definition 2.5 *(Single bounding function) Let $S(\mathbf{q}, \mathbf{p})$ be the similarity measures (either dist or sim). We denote $LB(\mathbf{q}, \mathbf{p})$ and $UB(\mathbf{q}, \mathbf{p})$ the lower and upper bound functions respectively for all pairs $(\mathbf{q}, \mathbf{p})$ if they satisfy the condition:*

$$LB(\mathbf{q}, \mathbf{p}) \leq S(\mathbf{q}, \mathbf{p}) \leq UB(\mathbf{q}, \mathbf{p}) \tag{2.9}$$

Definition 2.6 *(Group bounding function) Let G be the group of objects. We denote $LB(\mathbf{q}, G)$ and $UB(\mathbf{q}, G)$ the lower and upper bound functions between $\mathbf{q}$ the group G if:*

$$LB(\mathbf{q}, G) \leq min_{\mathbf{p} \in G} S(\mathbf{q}, \mathbf{p}) \text{ and } max_{\mathbf{p} \in G} S(\mathbf{q}, \mathbf{p}) \leq UB(\mathbf{q}, G) \tag{2.10}$$

Using range search query (Definition 2.1) as an example, the bounding functions can be used to filter some objects which are near or far away from the query $\mathbf{q}$. For example: if $\mathbf{p}$ is the vector such that $LB(\mathbf{q}, \mathbf{p}) \geq \tau$, we can immediately declare the dissimilarity function $dist(\mathbf{q}, \mathbf{p}) \geq \tau$ due to the Definition 2.5 and directly discard this vector $\mathbf{p}$ since this vector is no longer possible to be inside the solution set. Instead of filtering each vector one-by-one, we can also filter a group of objects via the group-bounding function (c.f. Definition 2.6). The filtering condition for group G is either $LB(\mathbf{q}, G) \geq \tau$ or $UB(\mathbf{q}, G) \leq \tau$. The bounding function is useful once the bound is tight and the evaluation time is much faster than the exact computation of the similarity measure.

We review some existing bound functions for the similarity measures in Table 2.1.

2.4.1 Euclidean Distance (ED)

Euclidean distance contains a wide range of properties. Faloutsos et al. [44] derive the MBR-based lower bound function for Euclidean distance. Ciaccia et al. [33, 120] utilize the metric property of Euclidean distance and derive efficient triangle inequality lower bound function. These two bound functions (LB_R and LB_{metric}) are also used as the group bounding functions [44, 33] for filtering a group of objects. On the other hand, some other individual bounding functions are also developed. Yi et al. [131] apply the convex property of the Euclidean norm and derive the fast lower bound function. Different research groups in computer vision community [55, 12, 90, 91, 92] utilize the fast orthogonal transform (called Walsh Hadamard Transform) for establishing new efficient algorithms. Some other dimension reduction bounding functions are also developed for Euclidean distance similarity measures, Gharavi-Alkhansari [49] derives the progressive bounds for boosting the efficiency performance in template matching problem. Yi et al. [131] also develop another progressive bounding functions in time series retrieval problem. Table 2.4 summarizes all bound functions. e means, on average, the number of access elements in the vectors for each query, the range of e is within [1,d]. d_r means the reduced dimension used in the bound functions.

Table 2.4. Bounding Functions for Euclidean distance

Nature	Name	References	Time Complexity
Group	LB_R	[44]	$O(d)$
Group	LB_{metric}	[33]	$O(d)$
Single (transform-based)	LB_{trans}	[55, 12, 90, 91, 92]	$O(e)$
Single (dimension reduction)	LB_{red}	[131, 49]	$O(d_r)$

2.4.2 Inner Product (IP)/ Cosine Similarity (COS)

In the group bounding function, Ram et al. [99] explore the fast group upper bounding function (UB_{ball}) for inner product and combine with the ball-tree structure to boost up the performance of this problem. Teflioudi et al. [119] utilize the Cauchy Schwarz Inequality to propose the group upper bounding functions (UB_{Cauchy}): $max_{\mathbf{p}\in G}\mathbf{q}^T\mathbf{p} \leq ||\mathbf{q}|| \times max_{\mathbf{p}\in G}||\mathbf{p}||$. In the individual bounding function, Teflioudi et al. [119] also propose another fast individual upper bounding functions. Li er al. [79] further develop SIR transformation-based algorithm to further tighten the upper bound function from [119].

Due to the similar functional-form between COS and IP, the technical bound functions are similar with each others. Existing works in the literatures [127, 7] utilize the upper bound function which is based on the derivation of the Cauchy-Schwarz Inequality. Wei et al. [127] further derive the progressive bound function to tighten the upper bound functions. These two techniques are similar with [119]. Table 2.5 summarizes all bound functions.

Table 2.5. Bounding Functions for IP and COS

Nature	Name	References	Time Complexity
Group	UB_{ball}	[99]	$O(d)$
Group	UB_{Cauchy}	[119]	$O(d)$
Single	UB_{incr}	[127, 119, 79]	$O(e)$

2.4.3 Kernels

Gray [50] and Gan et al. [46] focus on the Gaussian kernel function K_{Gauss} and apply the group-based bound function of Euclidean distance (e.g. [44]) to derive the fast lower and upper group bounding functions, denoted as $\ell(\mathbf{q}, G)$ and $u(\mathbf{q}, G)$ respectively, for $\mathcal{F}_P(\mathbf{q})$, as shown in Equations 2.11 and 2.12.

$$LB(\mathbf{q}, G) = \sum_{\mathbf{p_i} \in G} w_i \exp(-\gamma u(\mathbf{q}, G)^2) \tag{2.11}$$

$$UB(\mathbf{q}, G) = \sum_{\mathbf{p_i} \in G} w_i \exp(-\gamma \ell(\mathbf{q}, G)^2) \tag{2.12}$$

The bound functions $LB(\mathbf{q}, G)$ and $UB(\mathbf{q}, G)$ depend on the selection of group-based Euclidean bound functions $\ell(\mathbf{q}, G)$ and $u(\mathbf{q}, G)$ (c.f. Table 2.4).

2.4.4 Dynamic Time Warping (DTW)

Different bound functions for DTW have been derived in the literatures [72, 104, 45, 98]. The time complexity is summarized in Table 2.6. d_r in LB_{FTW} is the reduced dimension of the time series. Rakthanmanon et al. [98] further reduce the time complexity of LB_{Kim} and propose LB_{KimFL} which is in $O(1)$ time. Moreover, they perform the experimental evaluation of the tightness and the computation time of different bound functions. Then, they conclude the sequence of the bound evaluations (refer to Figure 9 of [98]) to be 1: LB_{KimFL}, 2: LB_{Keogh} and then 3: the progressive evaluation of the DTW.

Table 2.6. Bounding Functions for DTW

Nature	Name	References	Time Complexity
single	LB_{KimFL}	[98]	$O(1)$
single	LB_{Kim}	[72]	$O(d)$
single	LB_{Keogh}	[45]	$O(d)$
single	LB_{FTW}	[104]	$O(d_r^2)$

2.4.5 Earth Mover's Distance (EMD)

A wide range of EMD lower bound and upper bound functions are derived in the previous literatures. The most representative works for the derivation of the lower bound functions are summarized in [34, 9, 128, 103, 117]. Cohen [34] utilizes the property of ground matrix and develops the $O(d)$ time lower bound function. Assent et al. [9] relieve the constraints of the linear programming of EMD and derive the $O(d^2)$ lower bound function. Later, Wichterich et al. [128] derive another lower bound function by reducing the dimensionality of the feature vectors. Since this bound function depends on the parameter, dimensionality d_{red}, we also call this function the parametric bounding function. Ruttenberg et al. [103] demonstrate the significant tightness of [34] and further derive the group-based lower bound functions. Tang et al. [117] utilize the bipartite flow network property and derive the progressive lower bound function for $EMD(\mathbf{q}, \mathbf{p})$, which iteratively provides the tighter bound values until it either reaches the exact value or can be filtered. Consider the upper bound functions, Jang et al. [64] store a set of hilbert curves and assign the flow between the bipartite graph based on these curves. Since the flow is feasible, this can be regarded as the upper bound function of $EMD(\mathbf{q}, \mathbf{p})$. The time complexity is $O(d)$ in this upper bound function. Another upper bound function is based on the greedy assignment of flow from the bipartite graph [117] (Section 4.5). It assigns the flow from $\mathbf{q}$ to

p with the smallest cost c_{ij} first iteratively until all flow are moved to **p**. The time complexity of this upper bound function is $O(d^2)$.

We summarize the lower and upper bound functions in Table 4.1.

Table 2.7. Summary of lower and upper bound functions for EMD

Nature	Name	References	Time Complexity
group	$LB_{Proj,group}$	[103]	$O(d)$
single	LB_{Proj}	[34]	$O(d)$
single	LB_{IM}	[9]	$O(d^2)$
single	LB_{Red,d_r}	[128]	$O(d^2 + {d_r}^3 \log d_r)$
single	UB_H	[64]	$O(d)$
single	UB_G	[117]	$O(d^2)$

2.5 Approximation Methods of Similarity Measures

In reality, many applications, for example: kNN Image Retrieval, template matching and kernel computations, require the massive amount of computations of the similarity function. Exact computations of this query can be inefficient. As such, many approximate methods have been proposed for different similarity measures. We summarize different approximation methods.

2.5.1 Locality Sensitivity Hashing (LSH)

LSH [62] is originally designed for boosting up the k-nearest neighbor search with ED as the similarity measure. Its idea is to retrieve the objects which are near with the query **q** with probabilistic guarantee. This technique has been studied extensively in ED [37, 118, 47]. Gan et al. [47] provides the following guarantee (c.f. Lemma 2.1).

LEMMA 2.1 *The approximate k-nearest neighbor search algorithm in [47] can guarantee the time complexity to be* $O(d \log n + n \log n)$ *and the accuracy* $|ED(\mathbf{q}, \mathbf{p_r^{(i)}}) - ED(\mathbf{q}, \mathbf{p_{exact}^{(i)}})| \leq c^2 R$ *with constant probability.* $\mathbf{p_r^{(i)}}$ *and* $\mathbf{p_{exact}^{(i)}}$ *mean the* i^{th} *returned object from the approximate k-nearest neighbor solution set and* i^{th} *returned object from the exact k-nearest neighbor solution set respectively.* c *and* R*, which mean the constant and fixed radius respectively, are internal parameters of LSH algorithm.*

This algorithm is useful for high dimensional datasets where the dimensionality $d >> \log n$.

This approach is also generalized to other similarity measures, for example: IP [113], EMD [25].

2.5.2 Approximation Methods for EMD

As shown in Table 2.1, the time complexity of EMD is $O(d^3 \log d)$ which is inefficient. As such, using EMD as the distance function for kNN image retrieval is not efficient, which limits its application to low-scale datasets [117]. Therefore, many approximate techniques are proposed to EMD function to boost up the efficiency performance without losing too much accuracy theoretically or practically.

The literatures can be also divided into two parts. The first part guarantees the returned result is theoretically near $EMD(\mathbf{q}, \mathbf{p})$. The second part is the heuristics method which does not provide theoretical guarantee. In the first part, [61, 8, 70, 5] are the representative works in the theoretical computer science community. However, they either focus on a planar graph setting (i.e., calculating EMD on two planar pointsets) [61] or lack of flow concept (i.e., the approximate ratio is analyzed based on a uni-flow model) [8, 70, 5]. In the second part, Pele et al. [97] remove some records from the cost matrix when their values are larger than a pre-defined threshold. The EMD computation time is correlated to the sparsity of the cost matrix so that the threshold plays a role in controlling the quality and the efficiency. Jang et al. [64] store a set of hilbert curves and assign the distance between two images based on these curves. However, the approximate quality is highly relevant to the hilbert curve selection and there is no theoretical guarantee. Shirdhonkar et al. [112] utilize the wavelet theory in their approximation algorithm, which can be viewed as a heuristic solution with no theoretical guarantee.

2.6 Indexing Structures for Multi-Dimensional Data

Except for the bound functions, indexing structure is also the key to help boosting up the evaluation time for the bound functions for different similarity measures. Table 2.8 summarizes different indexing structures in the literature.

Table 2.8. Summary of different indexing structures

Name	Nature	Group Type	References
kd-tree	group	hyper-rectangle	[14]
R-tree (family)	group	hyper-rectangle	[53, 111, 11]
ball-tree	group	hyper-sphere	[87]
m-tree	group	hyper-sphere	[33]
I-distance	group	hyper-sphere	[63]
VA-file	single	n/a	[126]
Prefix Sum Array	single	n/a	[56, 123]

2.6.1 Group Filtering

In order to support group filtering for different queries (c.f. Section 2.3) Many research works focus on how to efficiently filter a group of objects at once. Tree-structures (c.f. Figure 2.1) are extensively studied for achieving this goal. We can divide the literatures into two main camps. The first camp is to build-index based on the hyper-rectangle, while another part is to build-index based on the hyper-sphere.

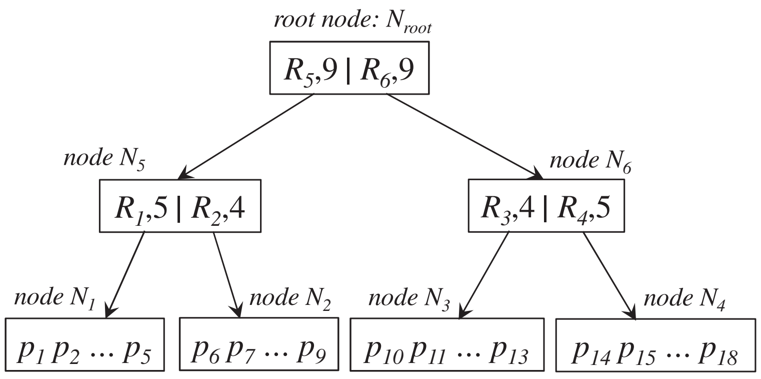


Figure 2.1. Hierarchical tree structure

In the first camp, Faloutsos et al. [44] propose using the R-tree to index the time-series in sequence database which support MBR-based bound functions. Scikit-learn [94] supports the kd-tree indexing [14] for accelerating the Approximation of Kernel Aggregate Query. Gan et al. [46] also combine the kd-tree for boosting up the computation of Classification of Kernel Aggregate Query.

In the second camp, Ciaccia et al. [33] propose M-tree to index the general feature vectors. Moore [87] propose another ball-tree structure for boosting up different other machine learning problems, such as K-mean. This structure has been also used for Approximation of Kernel Aggregation Query [50] which is also currently supported by Scikit-learn [94]. Later, I-Distance [63] has been also proposed to support group-based filtering in different metric-based similarity measures. In recommender system context, cone-tree [99] has been proposed for efficient computation for the inner product retrieval problem.

Many other variants of these index-structures are summarized in the monographs [105, 132] and the most recent experimental paper [28].

2.6.2 Single Vector Filtering

Many problems in computer vision and machine learning normally involve high-dimensional data. For example: In the kNN image retrieval context, Lab feature extraction normally produces 256-dimensional feature vector [102]. However, once the dimension is increased to more than 10-100, existing group-based filtering indexing structure [126], for example: R-tree [53], can degenerate into the basic sequential scan method without any improvement. This effect is called curse of dimensionality [126]. As such, Weber et al. [126] propose the compression-based method, in which they store smaller number of bits for each dimension value and then create the compressed dataset in the offline stage. Their algorithm further scans this compressed dataset in the online stage and obtain the lower bound function.

In some problems, the dimension of each multi-dimensional query vector is not known in advance. For example: In template matching problem for object detection application, the size of query (image object) is not necessary pre-known in the pre-processing stage for any object detection system. In time-series retrieval problem, the length of query time series is also not known in advance [98].

Therefore, it is very space-inefficient if we use tree-based structures (c.f. Section 2.6.1) for this type of applications [98] since we need to prebuild the tree for each dimension. Prefix-Sum Array [56] (also known as Integral Image [123] in computer vision context) is applied for existing problem (e.g. template matching [91]) to boost up the evaluation time of bounding functions which involves the summation terms, for example: one bound function [131], which is based on dimension reduction technique and $e = 1$ (c.f. Table 2.4), of Euclidean distance is: $LB_{\oplus}$ (c.f. Table 2.4, [131]), which is:

$$LB_{\oplus}(\mathbf{q}, \mathbf{p}) = \frac{1}{\sqrt{d}} \left| \sum_{i=1}^{d} q[d] - \sum_{i=1}^{d} p[d] \right| \tag{2.13}$$

The computation time of the summation terms $\sum_{i=1}^{d} q[d]$ and $\sum_{i=1}^{d} p[d]$ can be achieved in $O(1)$ time via the prefix-sum array and thus, $LB_{\oplus}(\mathbf{q}, \mathbf{p})$ can be also computed in $O(1)$ time.

2.6.3 Optimal Order of Filtering

Generic nearest neighbor search algorithms [110, 77] are applicable to any types of objects and similarity measures, while Ref. [110, 77] focus on the dissimilarity function $dist(\mathbf{q}, \mathbf{p})$. Ref. [110] requires using a lower bound function $LB(\mathbf{q}, \mathbf{p})$. Its search strategy [110] is to examine objects in ascending order of $LB(\mathbf{q}, \mathbf{p})$ and then compute their exact distances to q, until the current $LB(\mathbf{q}, \mathbf{p})$ exceeds the best NN distance found so far. Ref. [77] takes an additional upper bound function $UB(\mathbf{q}, \mathbf{p})$ as input and utilizes it to further reduce the searching time.

Chapter 3

Sub-Window Nearest Neighbor Search (SWNNS) on Matrix

SWNNS (a.k.a Template Matching, c.f. Figure 3.1) is the fundamental problem in computer vision. Many existing solutions have been proposed to solve SWNNS problem for rectangular-shape query [49, 55, 12, 120, 90, 109, 91, 92, 19, 20, 93] and irregular-shape query [96, 39, 121, 13].

Dual-Bound [109] 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.

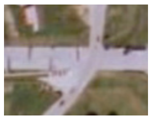
data matrix	rectangular query	irregular-shaped query
		
(a) satellite map image	(b) junction with background	(c) junction

Figure 3.1. Sub-window nearest neighbor search (SWNNS)

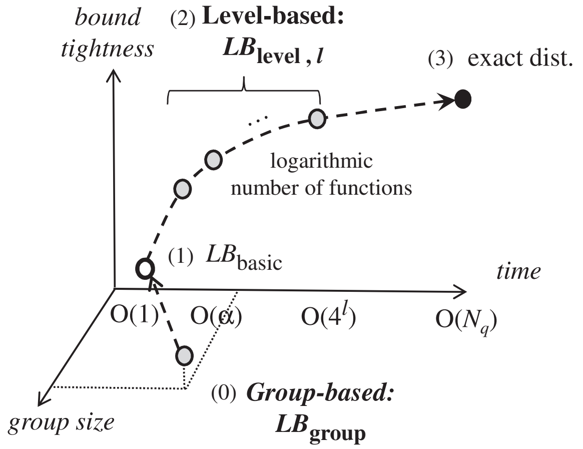

bound tightness
(2) Level-based: $LB_{\text{level},l}$
(3) exact dist.
logarithmic number of functions
(1) LB_{basic}
time
O(1)
O(α)
O(4^l)
O(N_q)
(0) Group-based: LB_{group}
group size

Figure 3.2. Illustration of our progressive approach

Our work [19] first 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 3.2. 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 1.2b) 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 1.2c.

The state-of-the-art exact method for this problem [39] is an extended version of Dual-Bound [109].

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 [19], our new work [20] is to develop an efficient solution for answering SWNNS on irregular-shaped queries (Section 3.3). 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 this chapter is organized as follows. Section 5.1 defines our problem and introduces background information. Section 3.2 presents our proposed solution for rectangular queries. Section 3.3 studies the SWNNS problem for queries with irregular shapes. Section 3.4 discusses our experimental results. Section 3.5 elaborates on the related work on SWNNS problem. Section 3.6 concludes the paper with future research directions.

3.1 Preliminaries

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

3.1.1 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.

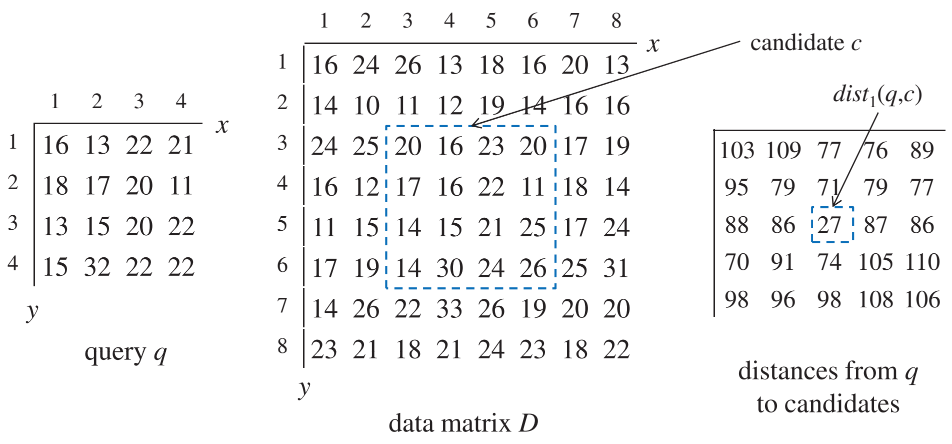
Problem 3.1 (Sub-window NN Search) *Given a query matrix q and a data matrix D, this problem finds the candidate c_{best} such that it has the minimum*

$dist(q, c_{best})$, 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 3.3 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 3.3 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.



query q

	1	2	3	4
1	16	13	22	21
2	18	17	20	11
3	13	15	20	22
4	15	32	22	22

data matrix D

	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

distances from q to candidates

103	109	77	76	89
95	79	71	79	77
88	86	27	87	86
70	91	74	105	110
98	96	98	108	106

Figure 3.3. Example for the problem

3.1.2 Prefix-Sum Matrix & Basic Lower Bounds

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

Definition 3.1 (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* $A[x_1..x_2, y_1..y_2]$.

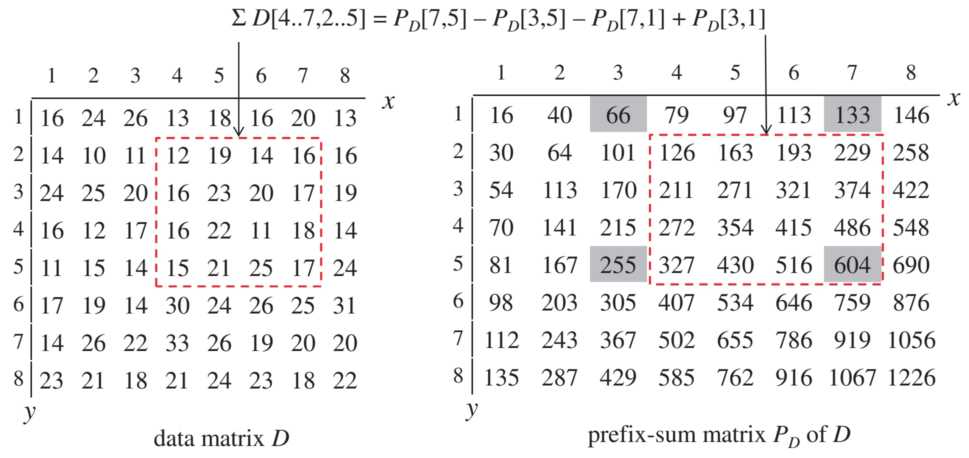
As we will introduce shortly, lower bound functions require summing the values in a rectangular region in a matrix. We can speed up their computation by using a prefix-sum matrix [56], also known as an integral image [123] in the computer vision community.

Definition 3.2 (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 [56]. 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 3.1.

$$\sum A[R] = \begin{cases} P_A[x_2, y_2] & \text{if } x_1 = 1, y_1 = 1 \\ P_A[x_2, y_2] - P_A[x_1 - 1, y_2] & \text{if } x_1 > 1, y_1 = 1 \\ P_A[x_2, y_2] - P_A[x_2, y_1 - 1] & \text{if } x_1 = 1, y_1 > 1 \\ P_A[x_2, y_2] + P_A[x_1 - 1, y_1 - 1] & \\ -P_A[x_1 - 1, y_2] - P_A[x_2, y_1 - 1] & \text{otherwise} \end{cases} \tag{3.1}$$

Figure 3.4 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.



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

data matrix D

y \ 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 \ 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

Figure 3.4. Example of a prefix-sum matrix

We introduce the basic lower bound function LB_{basic}, which is used as a building block in Figure 3.2. 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 [131]. The second one ($LB_{\Delta}(q, c)$) is derived from the triangle inequality of the L_p distance [33, 63]. 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: $D'[i, j] = |D[i, j]|^p$.

$$LB_{\oplus}(q,c) = \frac{\sqrt[p]{N_q}}{N_q} \cdot \left| \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} q[i,j] - \sum_{i=1}^{L_q} \sum_{j=1}^{W_q} c[i,j] \right| \tag{3.2}$$

$$LB_{\triangle}(q,c) = \left| \sqrt[p]{\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |q[i,j]|^p} - \sqrt[p]{\sum_{i=1}^{L_q} \sum_{j=1}^{W_q} |c[i,j]|^p} \right| \tag{3.3}$$

As a remark, we are aware of lower bound functions used in the pattern matching literature [90, 120, 12, 55, 91]. 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.

3.2 Progressive Search Approach

We first present our idea and algorithm in Section 3.2.1. Then, we elaborate the lower bound functions used in the algorithm in Sections 3.2.2, 3.2.3, 3.2.4.

3.2.1 The Flow of Proposed Algorithm

We illustrate the flow of our proposed NN search method in Figure 3.5. Like [110, 77], 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}.

Table 3.1 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 3.1). We have introduced LB_{basic} (e.g., $LB_{\Delta}, LB_{\oplus}$) in Section 3.1.2. We will develop a level-based bound $LB_{level,\ell}$ and a group-based bound LB_{group} in Sections 3.2.2 and 3.2.3, respectively. Section 3.2.4 explores an efficient technique for computing LB_{group}, which involves a tunable parameter α.

We summarize our method in Algorithm 1. Like [110, 77], we employ a min-heap H in order to process entries in ascending order of their lower bound distance. We also maintain the best distance found thus far τ_{best} during the search. The algorithm terminates when the deheaped entry's lower bound dis-

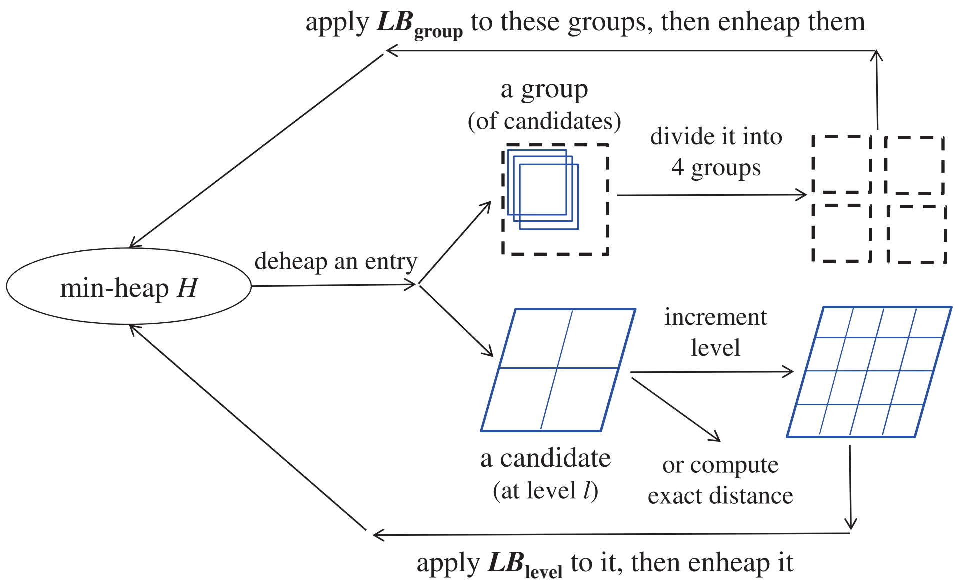

apply LB_{group} to these groups, then enheap them
a group (of candidates)
divide it into 4 groups
min-heap H
deheap an entry
increment level
a candidate (at level l)
or compute exact distance
apply LB_{level} to it, then enheap it

Figure 3.5. The flow of our progressive search method

Table 3.1. 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

tance 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 [110, 77] 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:

$$\ell_{\max} = \lceil \log_2(\max\{L_q, W_q\}) \rceil \tag{3.4}$$

Algorithm 1 Progressive Search Algorithm for NN search

```
1: procedure PROGRESSIVE SEARCH(query $q$, data matrix $D$)
2:     $\tau_{best} \leftarrow \infty$; $e_{best} \leftarrow \emptyset$          ▷ the best entry found so far
3:     create a min-heap $H$
4:     create a heap entry $e_{root}$
5:     $e_{root}.G \leftarrow [0..L_D - 1, 0..W_D - 1]$          ▷ the entire region
6:     $e_{root}.bound \leftarrow LB_{group}(q, e.G)$
7:     enheap $e_{root}$ to $H$
8:     while $H \neq \emptyset$ do
9:         $e \leftarrow$ deheap an entry in $H$
10:        if $e.bound \geq \tau_{best}$ then          ▷ termination condition
11:            break
12:        if $|e.G| \neq 1$ then          ▷ group entry
13:            divide $e$ into 4 entries $e_1, e_2, e_3, e_4$
14:            for each $e_i$, $i \leftarrow 1$ to 4 do
15:                $e_i.bound \leftarrow LB_{group}(q, e_i.G)$
16:                $e_i.\ell \leftarrow 0$
17:                if $e_i.bound < \tau_{best}$ then    enheap $e_i$ to $H$
18:        else          ▷ candidate entry
19:            if $e.\ell < \ell_{\max}$ then
20:                $e.bound \leftarrow LB_{level,\ell}(q, e)$
21:                increment $e.\ell$
22:                if $e.bound < \tau_{best}$ then    enheap $e$ to $H$
23:            else          ▷ the deepest level
24:                $temp \leftarrow dist(q, e)$
25:                if $temp < \tau_{best}$ then    $\tau_{best} \leftarrow temp; e_{best} \leftarrow e$
```

3.2.2 Progressive Filtering for Candidates

As discussed before, the lower bound LB_{basic} and the exact distance $dist$ have a significant gap in terms of computation time and bound tightness (cf. Figure 3.2). 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 combine these 4^ℓ lower bound distances into $LB_{level,\ell}$ in Equation 3.5. $LB_{level,\ell}$ takes at most 4^ℓ region-sum operations, as each LB_{basic} takes one region-sum operation.

$$LB_{level,\ell}(q,c) = \Big(\sum_{v=1}^{4^\ell} LB_{basic}(q[R_v], c[R_v])^p\Big)^{1/p} \tag{3.5}$$

For example, in Figure 3.6, 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 3.1 *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. [19]]*

Note that [49, 131] 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 3.6. If we cannot filter c at level ℓ, then we attempt to

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

filter it with minimal extra effort, i.e., at level $\ell + 1$.

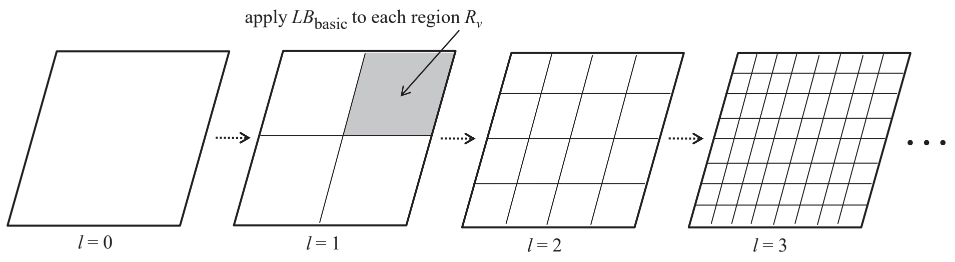


Figure 3.6. $LB_{level,\ell}$ **at different levels**

3.2.3 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 3.7. 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 3.3 (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**.

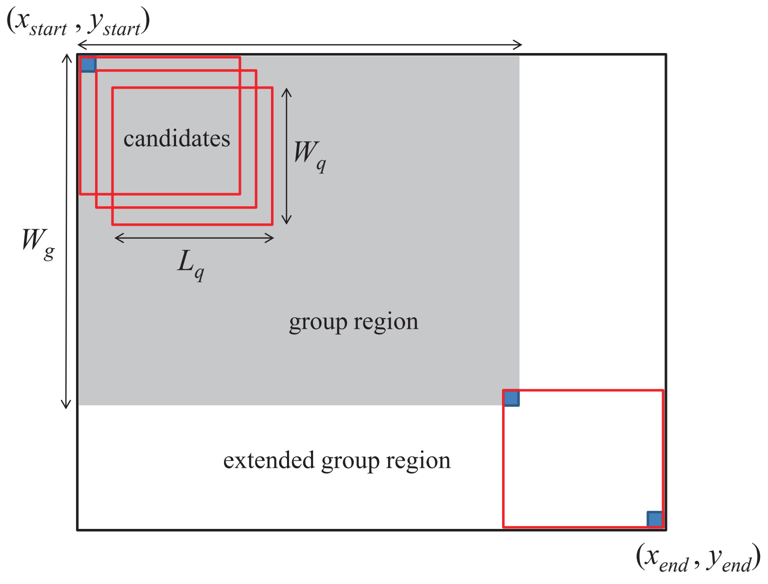


Figure 3.7. A group with $L_g \times W_g$ consecutive candidates

We illustrate these concepts in Figure 3.8. 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, $\mathcal{SL}_{N_q}(D[ext_q(G)]) = 9 + 9 + 10 + 10 = 38$.

Definition 3.4 *(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 $\mathcal{SH}_k(D[ext_q(G)])$ as* **the sum of highest k elements** *in $D[ext_q(G)]$.*

We then extend basic lower bound functions (e.g., $LB_{\oplus}, LB_{\Delta}$) for a group G. We propose the lower bound functions $LB^{\oplus}_{group}$ and LB^{Δ}_{group} for G in Equations 3.6, 3.7. In Equation 3.7, 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 func-

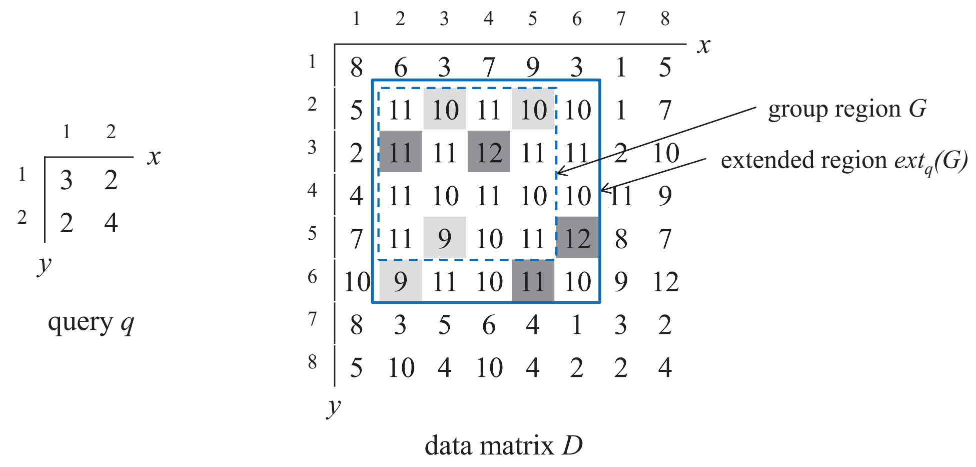


Figure 3.8. Illustration of $\mathcal{L}_{N_q}(D[ext_q(G)])$ (in light color) and $\mathcal{H}_{N_q}(D[ext_q(G)])$ (in dark color)

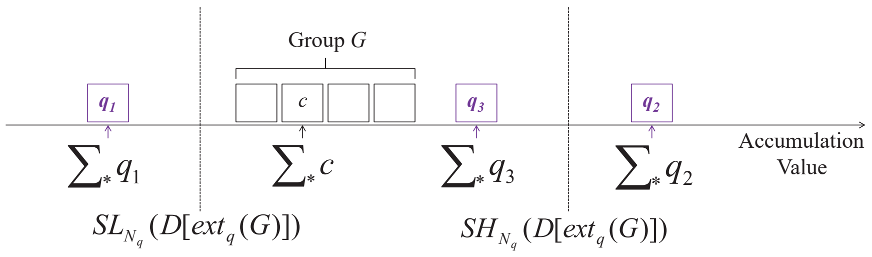


Figure 3.9. Illustration of the idea in $LB^{\oplus}_{group}(q, G)$

tions serve as lower bounds of $LB_{\oplus}(q, c), LB_{\Delta}(q, c)$ for any candidate c in G (cf. Lemmas 3.2,3.3).

$$LB^{\oplus}_{group}(q,G) = \begin{cases} \frac{\sqrt[p]{N_q}}{N_q}(\mathcal{SL}_{N_q}(D[ext_q(G)]) - \sum_* q) & \text{if } \mathcal{SL}_{N_q}(D[ext_q(G)]) > \sum_* q \\ \frac{\sqrt[p]{N_q}}{N_q}(\sum_* q - \mathcal{SH}_{N_q}(D[ext_q(G)])) & \text{if } \mathcal{SH}_{N_q}(D[ext_q(G)]) < \sum_* q \\ 0 & \text{otherwise} \end{cases} \quad (3.6)$$

$$LB^{\Delta}_{group}(q,G) = \begin{cases} \sqrt[p]{\mathcal{SL}_{N_q}(D^{\circ p}[ext_q(G)])} - \sqrt[p]{\sum_* |q[i,j]|^p} & \text{if } \mathcal{SL}_{N_q}(D^{\circ p}[ext_q(G)]) > \sum_* |q[i,j]|^p \\ \sqrt[p]{\sum_* |q[i,j]|^p} - \sqrt[p]{\mathcal{SH}_{N_q}(D^{\circ p}[ext_q(G)])} & \text{if } \mathcal{SH}_{N_q}(D^{\circ p}[ext_q(G)]) \\ & \quad < \sum_* |q[i,j]|^p \\ 0 & \text{otherwise} \end{cases} \quad (3.7)$$

$$LB'^{\oplus}_{group}(q,G) = \begin{cases} \frac{\sqrt[p]{N_q}}{N_q}(\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) - \sum_* q) & \text{if } \mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]}) > \sum_* q \\ \frac{\sqrt[p]{N_q}}{N_q}(\sum_* q - \mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]})) & \text{if } \mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]}) < \sum_* q \\ 0 & \text{otherwise} \end{cases} \quad (3.8)$$

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

LEMMA 3.2 *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 3.3 *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. □

Figure 3.9 explains why $LB^{\oplus}_{group}(q, G)$ 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 $LB^{\oplus}_{group}(q, G)$, respectively. For convenience, we drop the subscript N_q in the notations $\mathcal{SL}$ and $\mathcal{SH}$. By Equation 3.2, 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 $LB^{\Delta}_{group}(q,G)$.

During our search procedure (cf. Figure 3.5 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.

3.2.4 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 3.6) or $\mathcal{SL}_{N_q}(D^{\circ p}[ext_q(G)])$ and $\mathcal{SH}_{N_q}(D^{\circ p}[ext_q(G)])$ (Equation 3.7), 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 3.8, $D[2,2] = 11$, $D_{min} = 1$ 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:

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

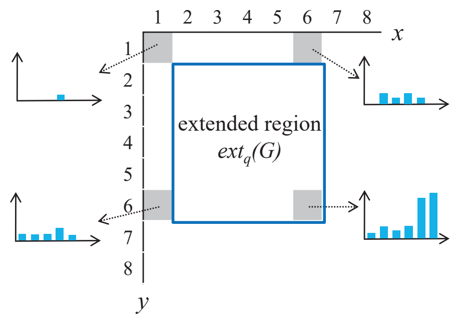
where

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

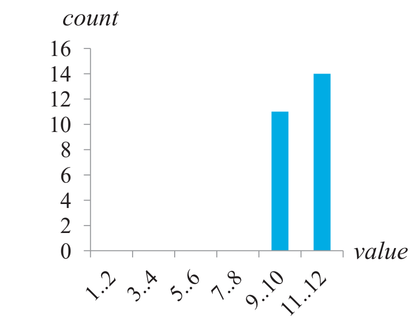
As a remark, the prefix histogram matrix occupies $O(\alpha N_D)$ space.

Figure 3.10a 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.

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 3.10b. 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 3.5.



(a) prefix histogram matrix PH_β



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

Figure 3.10. Prefix histogram matrix, $\alpha = 6, D_{min} = 1, D_{max} = 12$

Definition 3.5 (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 $CH_{D[ext_q(G)]}$.*

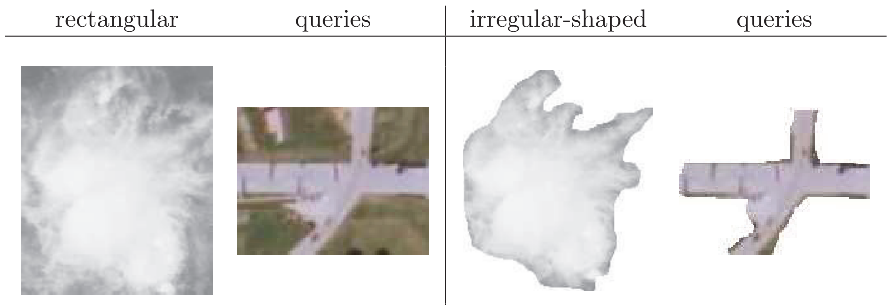
While scanning the bins of $CH_{D[ext_q(G)]}$ from left to right, we examine the count and the minimum bound of each bin to derive $\mathcal{SL}'_{N_q}(CH_{D[ext_q(G)]})$. A similar method can be used to derive $\mathcal{SH}'_{N_q}(CH_{D[ext_q(G)]})$. The cost of computing a group-based lower bound equals 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 3.10b. 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 3.8). Then we propose $LB'^{\oplus}_{group}(q, G)$ in Equation 3.8 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.

3.3 Extension for Irregular-Shaped Queries

As discussed before, some applications may need to deal with irregular-shaped queries. For example, in geospatial data integration [27, 26, 31], the query can be a road junction which may have a T-shape. In cloud motion detection [16], the query can be an irregular cloud. Figure 3.11 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 [39]. The binary mask matrix can be extracted by image segmentation methods or by application requirements [16, 27, 121].



(a) cloud with background (b) junction with background (c) cloud (d) junction

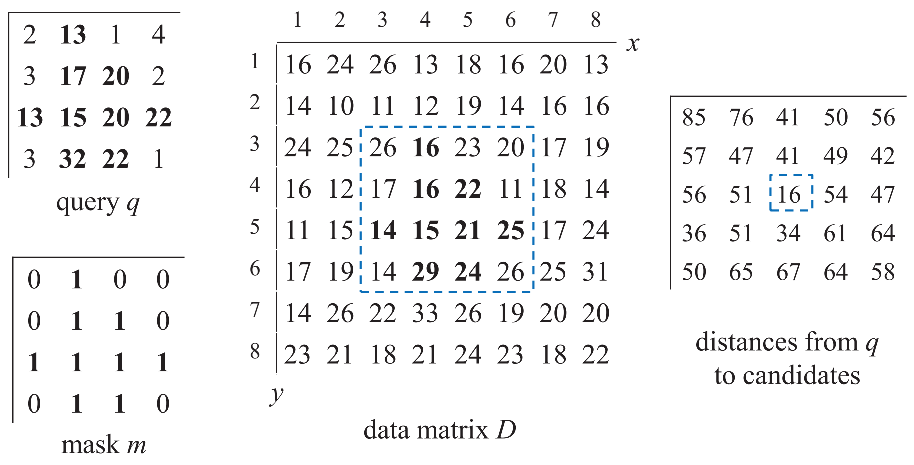
Figure 3.11. Examples of irregular-shaped queries

Problem 3.2 (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 $dist^{\diamondsuit}(q, c_{best})$*

where the distance is defined as:

$$dist^{\diamondsuit}(q,c) = \Big(\sum_{i=1}^{L_q}\sum_{j=1}^{W_q} m[i,j]\cdot|q[i,j]-c[i,j]|^p\Big)^{1/p} \tag{3.9}$$

We illustrate this problem in Figure 3.12. 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.



2	**13**	1	4
3	**17**	**20**	2
13	**15**	**20**	**22**
3	**32**	**22**	1

query *q*

0	**1**	0	0
0	**1**	**1**	0
1	**1**	**1**	**1**
0	**1**	**1**	0

mask *m*

y \ *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	26	**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	**29**	**24**	26	25	31
7	14	26	22	33	26	19	20	20
8	23	21	18	21	24	23	18	22

data matrix *D*

85	76	41	50	56
57	47	41	49	42
56	51	16	54	47
36	51	34	61	64
50	65	67	64	58

distances from *q* to candidates

Figure 3.12. 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 3.3.1. Second, we develop a more efficient extension by partitioning the mask in Section 3.3.2.

3.3.1 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 3.6,3.7) 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

$$M_q = \{(i, j) : m[i, j] = 1\} \tag{3.10}$$

By using M_q, we revise the equations for LB_{group} into $LB^{\oplus,\diamondsuit}_{group}(q, G)$ and $LB^{\Delta,\diamondsuit}_{group}(q, G)$, in Equations 3.11 and 3.12, respectively. We omit the proofs of their lower bound property as they are similar to the proofs of Lemmas 3.2 and 3.3. Figure 3.13 illustrates how to compute $LB^{\oplus,\diamondsuit}_{group}(q, G)$. 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: $LB^{\oplus,\diamondsuit}_{group}(q, G) = (|12 + 14 + 16 + 16 + 16| - |7 + 5 + 5 + 5 + 5|) = 47$.

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 3.12 and assume $p = 1$. By Equation 3.9, the exact distance is: $dist^{\diamondsuit}(q, c_{3,3}) = 16$. 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 $LB_{level,0}(q, c_{3,3}) > dist^{\diamondsuit}(q, c_{3,3})$. This happens because LB_{basic} considers all entries (including irrelevant entries) in a candidate. To prevent such violation, a simple solution is to disable $LB_{level,\ell}$.

$$LB_{group}^{\oplus,\diamondsuit}(q,G) = \begin{cases} \frac{\sqrt[p]{|M_q|}}{|M_q|}(\mathcal{SL}_{|M_q|}(D^{\circ 1}[ext_q(G)]) - \sum_{M_q} q) & \text{if } \mathcal{SL}_{|M_q|}(D^{\circ 1}[ext_q(G)]) > \sum_{M_q} q \\ \frac{\sqrt[p]{|M_q|}}{|M_q|}(\sum_{M_q} q - \mathcal{SH}_{|M_q|}(D^{\circ 1}[ext_q(G)])) & \text{if } \mathcal{SH}_{|M_q|}(D^{\circ 1}[ext_q(G)]) < \sum_{M_q} q \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

$$LB_{group}^{\Delta,\diamondsuit}(q,G) = \begin{cases} \sqrt[p]{\mathcal{SL}_{|M_q|}(D^{\circ p}[ext_q(G)])} - \sqrt[p]{\sum_{M_q} |q[i,j]|^p} & \text{if } \mathcal{SL}_{|M_q|}(D^{\circ p}[ext_q(G)]) > \sum_{M_q} |q[i,j]|^p \\ \sqrt[p]{\sum_{M_q} |q[i,j]|^p} - \sqrt[p]{\mathcal{SH}_{|M_q|}(D^{\circ p}[ext_q(G)])} & \text{if } \mathcal{SH}_{|M_q|}(D^{\circ p}[ext_q(G)]) \\ & \quad < \sum_{M_q} |q[i,j]|^p \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

$$\text{where } \mathcal{L}_{|M_q|}(D[G.R^{ext}]) \text{ is the lowest } |M_q| \text{ values in the submatrix } D[G.R^{ext}] \tag{3.13}$$

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

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 $LB_{group}^{\diamondsuit}$ at Lines 6 and 15. Third, we replace $dist(q, c)$ by $dist^{\diamondsuit}(q, c)$ at Line 23, and compute it efficiently by Equation 3.14.

$$dist^{\diamondsuit}(q,c) = \Big(\sum_{(i,j)\in M_q} |q[i,j] - c[i,j]|^p\Big)^{1/p} \tag{3.14}$$

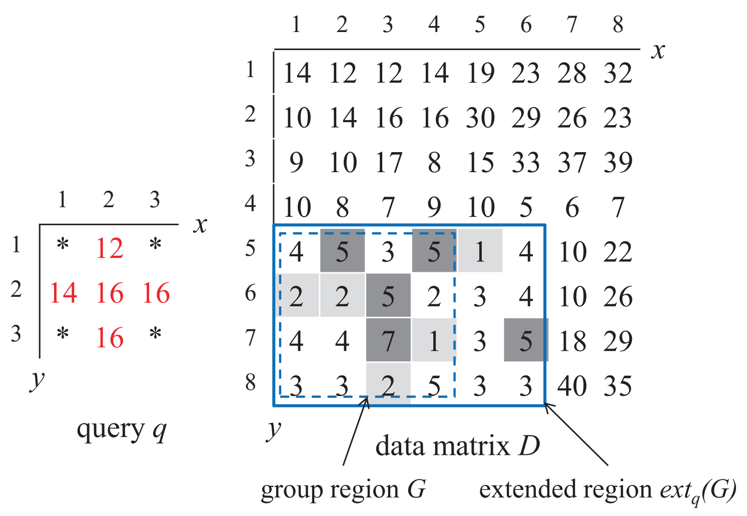


Figure 3.13. Group-based lower bound for irregular-shaped query

3.3.2 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 3.6 (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 3.14 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.

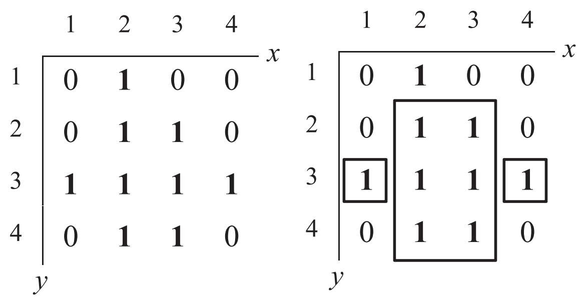


Figure 3.14. A valid partition Γ of a mask m

Given a valid partition Γ of a mask m, we define the lower bound function $LB_\Gamma(q, c)$ in Equation 3.15.

$$LB_\Gamma(q, c) = \Big(\sum_{R\in\Gamma} LB_{basic}(q[R], c[R])^p\Big)^{1/p} \tag{3.15}$$

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 3.4).

LEMMA 3.4 $LB_\Gamma(q, c) \leq dist^\diamondsuit(q, c)$.

Proof.

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

□

In general, we may employ a sequence of valid partitions $\langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell^{\diamondsuit}_{max}-1} \rangle$ with an increasing number of rectangles, where $\ell^{\diamondsuit}_{max}$ 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 3.15. If we cannot filter c at level ℓ, then we attempt to filter it with minimal extra effort, i.e., at level $\ell + 1$.

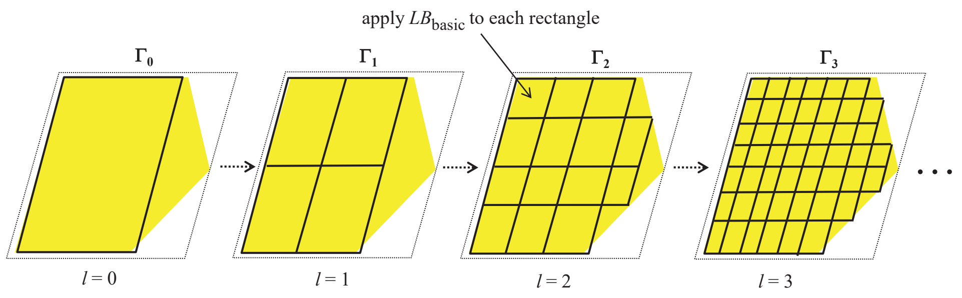


Figure 3.15. Level ℓ in irregular partition plan

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 $\Gamma_{seq} = \langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell^{\diamondsuit}_{max}-1} \rangle$ from the mask m, by using heuristics to be discussed in Section 3.3.2.2. Second, at Line 19, we replace $LB_{level,\ell}(q, e)$ by $LB_{\Gamma_\ell}(q, e)$.

3.3.2.1 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 $\Gamma_{seq} = \langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell^{\diamondsuit}_{max}-1} \rangle$. To simplify our analysis, we disable group-based pruning and measure the computation cost as the total number of rectangles used in calling $LB_{\Gamma_\ell}(q,c), dist^{\diamondsuit}(q,c)$ only.

Given a candidate c of D, we have:

$$cost(LB_{\Gamma_\ell}(q,c)) = |\Gamma_\ell|$$

$$cost(dist^{\diamondsuit}(q,c)) = |M_q|$$

where M_q was defined in Equation 3.10.

We denote the NN distance by $\tau_{opt} = \min_c dist^{\diamondsuit}(q,c)$. Since the algorithm employs a min-heap, it examines candidates in ascending 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 $\ell^{\diamondsuit}_{max}-1$, 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 $LB_{\Gamma_\ell}(q,c)$:

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

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

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

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 $\langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell^{\diamondsuit}_{max}-1} \rangle$ 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 3.1, we will show that the decision version of our problem (in Definition 3.7) is NP-hard via reduction from a known NP-complete problem called the *Rectilinear Picture Compression* (RPC) decision problem [48] (p.232) (in Definition 3.8).

Definition 3.7 (Γ-decision problem)

Instance: *$\langle q, D, m, \ell^{\diamondsuit}_{max}, LB_{basic}, K \rangle$, where q, D are matrices, m is a binary matrix, $\ell^{\diamondsuit}_{max}, K$ are integers, and LB_{basic} is a basic lower bound function.*

Problem: *Is there any sequence $\Gamma_{seq} = \langle \Gamma_0, \Gamma_1, \Gamma_2, \cdots, \Gamma_{\ell^{\diamondsuit}_{max}-1} \rangle$ such that it satisfies Definition 3.6 and $cost(\Gamma_{seq}, q, m, D) \leq K$?*

Definition 3.8 (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 3.1 *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 $\ell^{\diamondsuit}_{max}$ 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: $\tau_{opt} = dist^{\diamondsuit}(q,c) = \left(\sum_{(i,j)\in M_q} |0-1|^p\right)^{\frac{1}{p}} = |M_q|^{\frac{1}{p}}$.

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 3.6. We then set $\Gamma_0 = S$ and plan to show that Γ-decision instance returns true. By Equation 3.15, we derive:

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

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 $\tau_{opt} = dist^{\diamondsuit}(q,c) \geq LB_{\Gamma_0}(q,c)$. 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 3.6, 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'. $\square$

3.3.2.2 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 3.6).

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 3.16a and KD-tree split at level $\ell = 2$ in Figure 3.16b, respectively.

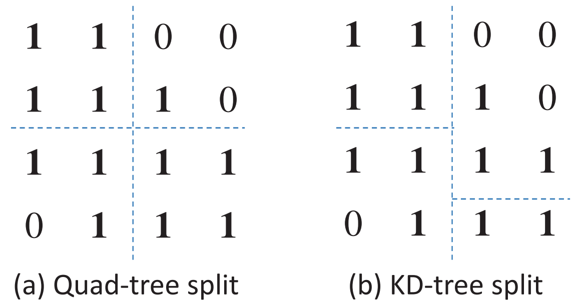


Figure 3.16. Examples on split

Suppose that, after splitting, we obtain the rectangles in Figure 3.17a. 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 3.6).

- **Drop**: We simply drop invalid rectangles, as shown in Figure 3.17b.

- **Grow**: Consider the bottom-left invalid rectangle in Figure 3.17c. 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.

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.

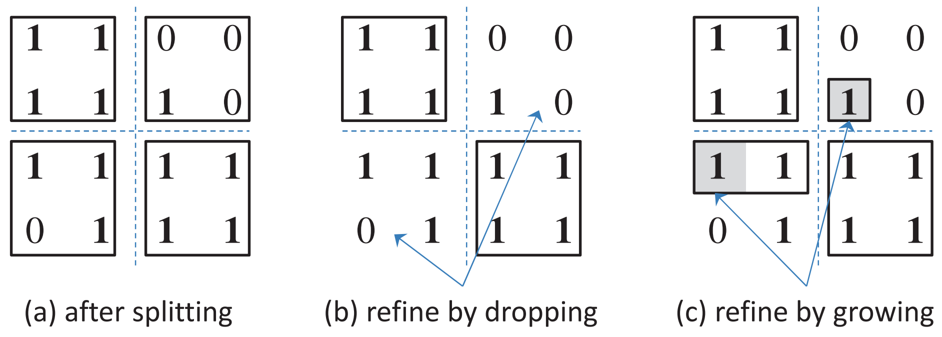


Figure 3.17. Examples on mend

3.4 Experimental Evaluation

We present our experiments for rectangular queries and irregular-shaped queries in Sections 3.4.1 and 3.4.2, respectively. We implemented all algorithms in C++ and conducted experiments on an Intel i7 3.4GHz PC running Ubuntu.

3.4.1 Experiments for Rectangular Queries

3.4.1.1 Experimental Setting

We summarize our methods and the state-of-the-art [109] (denoted as Dual) in Table 3.2a. 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 3.2. The list of our methods and the competitors

Method	Techniques used
Dual [109]	[109]
$\text{PSL}_{\oplus}$	Section 3.2.2
PSL_{Δ}	Section 3.2.2
$\text{PSLG}_{\oplus}$	Sections 3.2.2 and 3.2.3

(a) methods for rectangular queries

Method	Techniques used
iDual [39]	[39]
iPSG	Section 3.3.1
$\text{iPSL}_{quad,drop}$, $\text{iPSL}_{quad,grow}$ $\text{iPSL}_{kd,drop}$, $\text{iPSL}_{kd,grow}$	Section 3.3.2
$\text{iPSLG}_{quad,drop}$, $\text{iPSLG}_{quad,grow}$ $\text{iPSLG}_{kd,drop}$, $\text{iPSLG}_{kd,grow}$	Sections 3.3.1, 3.3.2

(b) methods for irregular-shaped queries

Note that each method (in Table 3.2) 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 3.3.

Table 3.3a lists the details of our datasets and queries. We collect these datasets from [3, 91]. *Photo640, Photo1280* and *Photo2560* [91] contain 30 images of the size 640×480, 1280×960 and 2560×1920 respectively. *Weather* [3] 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 3.3. Our datasets and queries

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

(a) the setting for rectangular queries

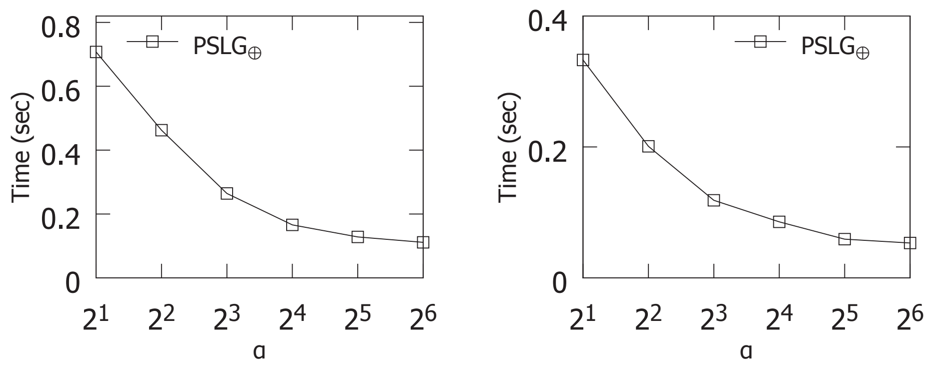
Dataset	Number of images	Number of queries per image	Query extraction method
Photo2560	30	10	Matlab segmentation
Weather	30	1	Manual extraction

(b) the setting for irregular-shaped queries

3.4.1.2 Results

First, we study the effect of the number of bins α on the response time of our method PSLG$_\oplus$. Figure 3.18 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.

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.



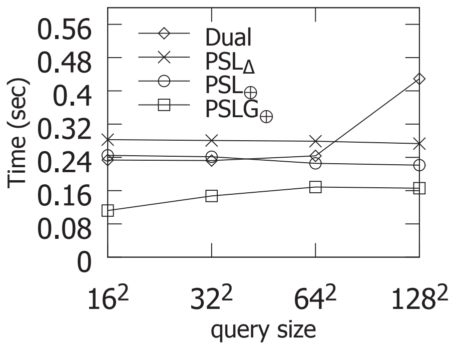
(a) Photo2560 (b) Weather

Figure 3.18. Effect of the number of bins α

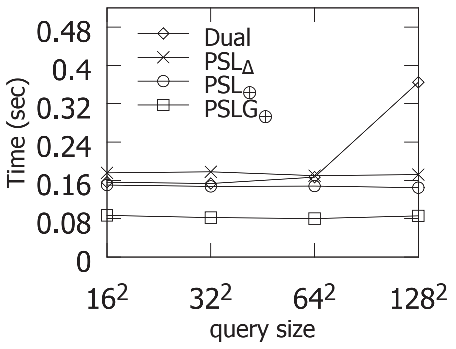
Next, we evaluate the scalability of methods with respect to the query size N_q. Figure 3.19 shows the response time of methods versus the query size N_q. Since Dual [109] can only support query size of the form $2^r \times 2^r$, we use query sizes like $32^2, 64^2, \cdots$ in this experiment. Thanks to the group lower bound function, PSLG$_\oplus$ outperforms all other methods and scales better with respect to N_q. On the other hand, Dual, 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.

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 3.20 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 PSLG$_\oplus$ and the other methods widens.

To test the robustness of methods, we follow [109] and add Gaussian noise



(a) Photo2560



(b) Weather

Figure 3.19. Effect of the query size N_q

into each query image. The query size is fixed to 128 × 128 in this experiment. Figure 3.21 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.

3.4.2 Experiments for Irregular-Shaped Queries

3.4.2.1 Experimental Setting

We summarize our methods and the state-of-the-art [39] (denoted as iDual) in Table 3.2b. 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

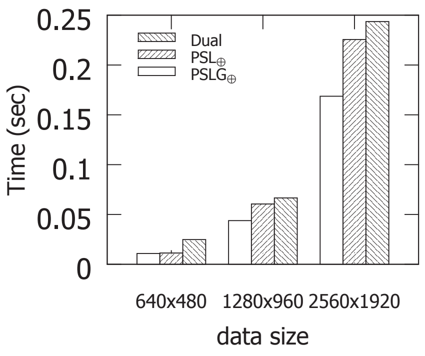


Figure 3.20. Effect of the data size N_D

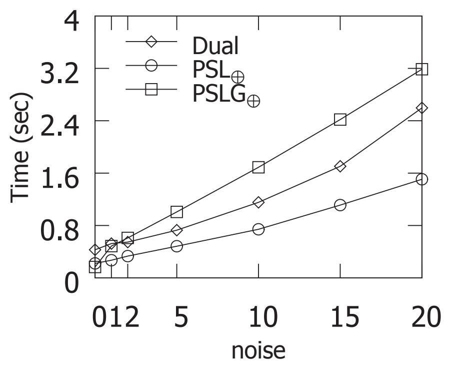
represent which partitioning techniques are used.

Table 3.3b lists the details of our datasets and queries. The experiments in [39] 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 [91], we apply the Matlab segmentation function on each rectangular query in order to obtain an irregular-shaped query. For the Weather dataset [3], we follow the same approach as in [121] and manually extract a cloud pattern from each data image.

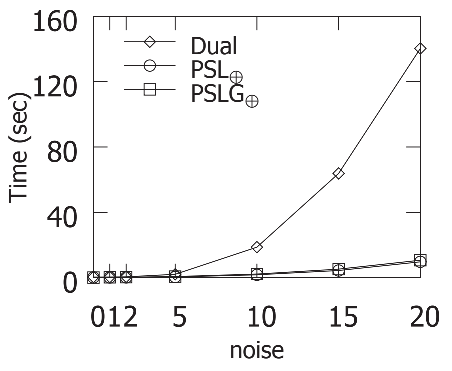
The response time of our iPSLG methods includes the partitioning time. In all of our experiments, the partitioning time is at most 1.2% of the response time only, implying that our partitioning heuristics incur very low overhead.

3.4.2.2 Results

We first compare the effectiveness of our partitioning heuristics and name these methods as iPSL$_{kd,drop}$, iPSL$_{quad,drop}$ iPSL$_{kd,grow}$ and iPSL$_{quad,grow}$. Fig-



(a) Photo2560



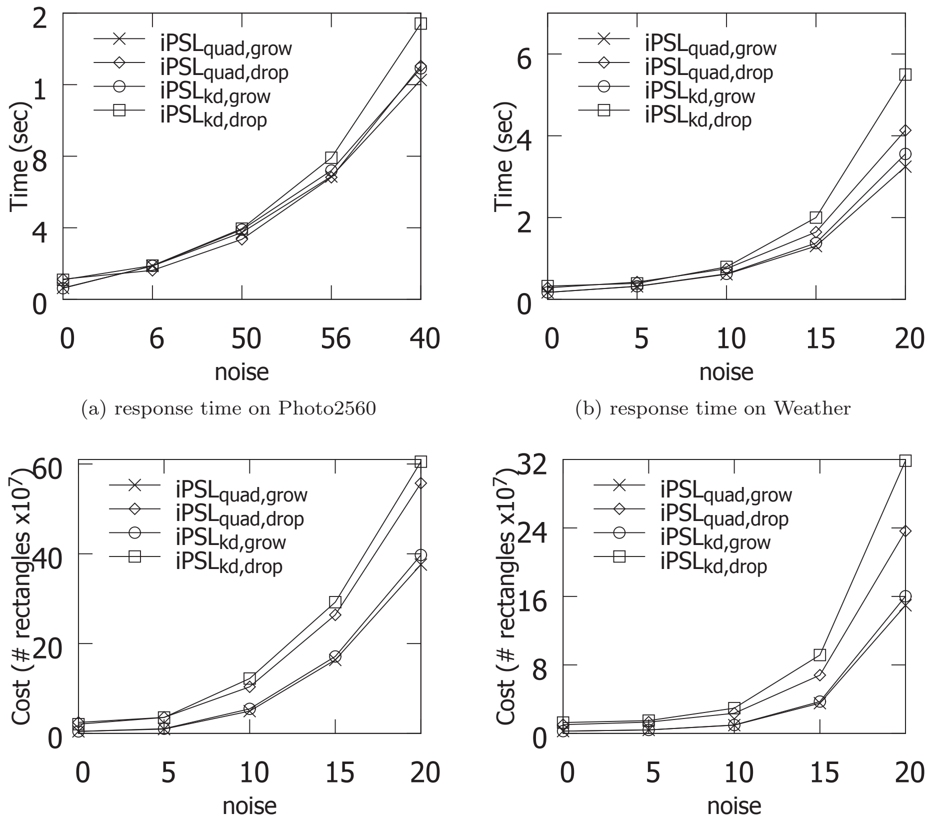
(b) Weather

Figure 3.21. Effect of the noise

ures 3.22a and b plot the response time of these methods with respect to the Gaussian noise as described in Section 3.4.1.2. 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 iPSL$_{quad,grow}$ is faster than others up to 20% and 40%, on the Photo2560 and the Weather datasets, respectively.

In Section 3.3.2, 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 3.22c and d. We observe that the trends are similar to those in Figures 3.22a and b. Again, 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 (iPSL$_{quad,grow}$), and one using both types of lower

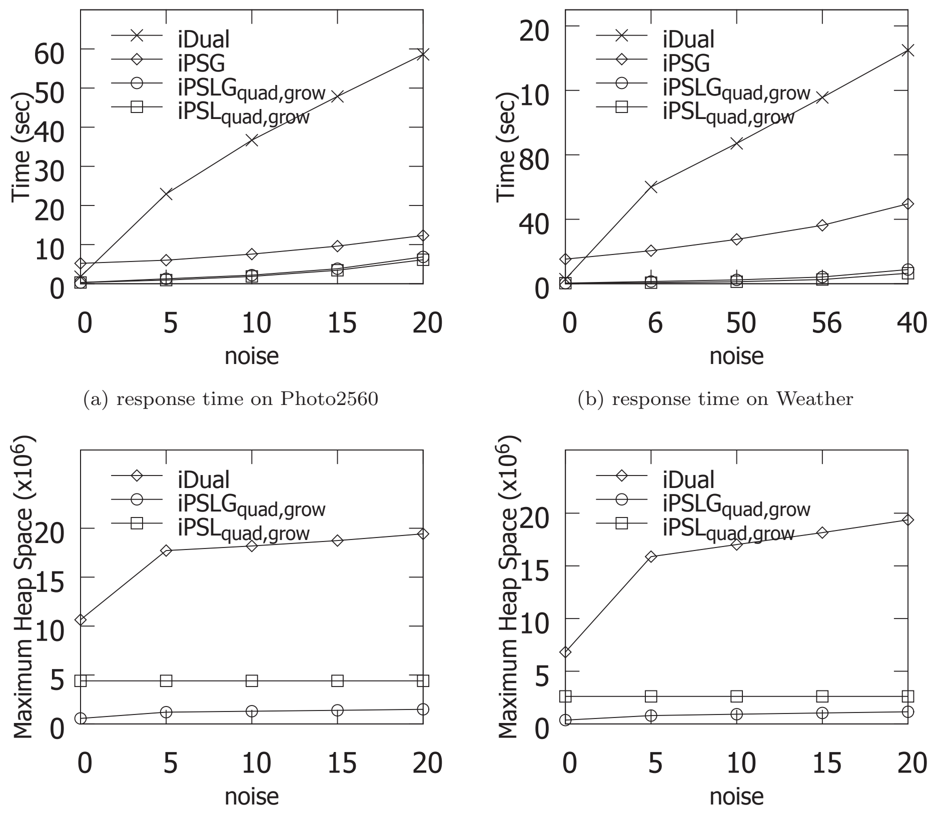


(a) response time on Photo2560 (b) response time on Weather

(c) cost on Photo2560 (d) cost on Weather

Figure 3.22. Comparisons of partitioning heuristics, varying the noise

bounds (iPSLG$_{quad,grow}$). Figure 3.23a 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 iPSL$_{quad,grow}$ and iPSLG$_{quad,grow}$ outperform iPSG, implying that our lower bounds in Section 3.3.2 are more powerful than the simple bound in Section 3.3.1. We then plot the maximum heap space of these methods, in terms of the number of rectangles, in Figure 3.23c 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.

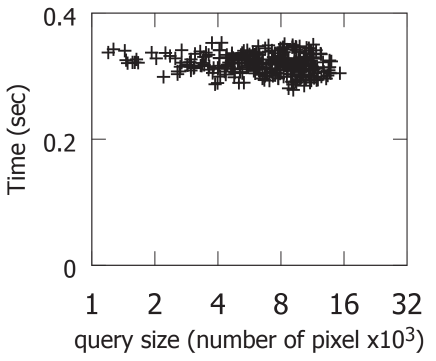


(a) response time on Photo2560 (b) response time on Weather

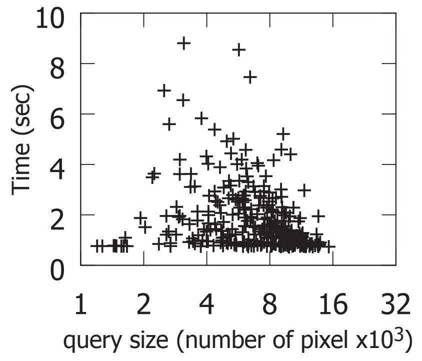
(c) max. heap space on Photo2560 (d) max. heap space on Weather

Figure 3.23. Comparisons of methods, varying the noise

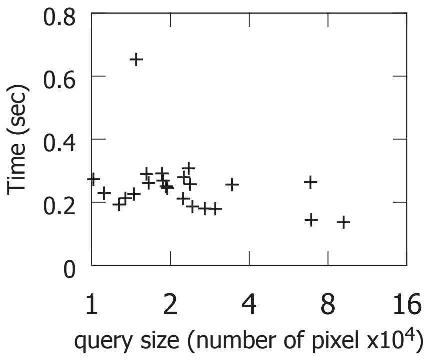
Finally, we test the effect of the query size on the response time of our method iPSL$_{quad,grow}$ and the competitor iDual, without noise. We measure the query size as the number of '1'-entries in the mask. The sizes of queries range from 960 to 16384 in the Photo2560 dataset and from 5400 to 84710 in the Weather dataset. We plot the results in Figure 3.24. Both the average response time and the worst-case response time of iPSL$_{quad,grow}$ outperform iDual significantly.



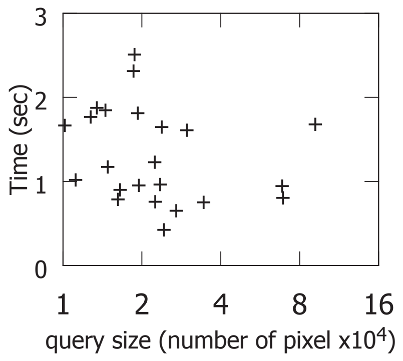
(a) iPSL$_{quad,grow}$ on Photo2560



(b) iDual on Photo2560



(c) iPSL$_{quad,grow}$ on Weather



(d) iDual on Weather

Figure 3.24. Effect of query size, fixing $\sigma = 0$

3.5 Related Work

The literature review of the recent techniques of Euclidean distance have been summarized in Section 2.4.1. In this section, we cover the works directly related to our problem. Then, we mainly discuss the differences between our work [19, 20] and previous works.

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 [90, 120, 12, 55, 91, 49, 109] 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. [91] propose a unified framework that covers range search solutions [90, 120, 12, 55, 49]. The state-of-the-art NN search method is [109]. 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, [109] 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 [49], our method (in Section 3.2) differs from [49] in two aspects. First, our method is applicable to any rectangular query, but [49] 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 3.2, 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. [85] 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 [13, 95, 96, 39, 121]. Like [13], our method partitions an irregular-shaped query into regions. While Ref.[13] 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 [13] and thus reduces the computation overhead during pruning. Several heuristics [95, 96, 121] 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 [39] is an extension of [109]. 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 [18, 74, 30]. A representative method [30] requires solving template matching (SWNNS) as a sub-problem. As such, our proposed method can be applied to speed up the method in [30].

The similarity search on a time series [131, 45, 98] 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.

3.6 Chapter Summary

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.

Chapter 4

The Power of Bounds: Answering Approximate Earth Mover's Distance with Parametric Bounds

In this chapter, we focus on computing approximate EMD yet allowing user to control the error. Specifically, given an error parameter ϵ, our problem is to find an approximate EMD value R such that R is within $1 \pm \epsilon$ times the exact EMD value. We are not aware of efficient algorithms that satisfy the above error requirement. The database community has derived several lower and upper bound functions of EMD [9, 10, 64, 102, 103, 117, 128, 130]. However, these bound functions provide no guarantee on the error of the bound.

Motivated by this, we wonder *whether existing lower and upper bound functions can be exploited to solve our problem efficiently.* Intuitively, if we can obtain a lower bound ℓ and an upper bound u of the exact EMD value such that they are sufficiently close (e.g., $u/\ell \leq 1+\epsilon$), then we get an approximate EMD value with error guarantee. The next question is how to select appropriate lower and upper bound functions with respect to ϵ. Consider all possible pairs of $\langle LB_i, UB_j \rangle$ where LB_i is a lower bound function, and UB_j is an upper bound function. Ideally, if we can accurately estimate the response time and the error for each pair $\langle LB_i, UB_j \rangle$, as shown in Figure 4.1, then the optimal solution is to choose the cheapest pair (i.e., $\langle LB_1, UB_4 \rangle$) whose error is below ϵ. The challenge is how to estimate quickly the response time and the error, while the estimates are reasonably accurate. This issue is complicated by the fact that, even within the same dataset, the same pair $\langle LB_i, UB_j \rangle$ of bound functions may yield different response time and error for different pairs of histograms.

Another issue is that, the limited number of bound functions prevents us to conduct fine-grained optimization. We need a wide spectrum of bound functions for EMD to provide sufficient trade-off points for optimization. To address this issue, we propose the concept of *parametric dual bound function*, which produces both lower and upper bounds simultaneously via shared computation, while its running time and tightness can be controlled via a parameter. In Figure 4.1, we indicate a parametric dual bound function by a dotted line in blue. By choosing its parameter value carefully, it is possible to obtain a better choice than the pair $\langle LB_1, UB_4 \rangle$. Since it is common to have skewed data in real applications, we will exploit the characteristics of skewed data to design a parametric dual bound function. As a remark, Wichterich et al. [128] have devised a parametric lower bound function, but not any parametric upper bound function. In contrast, we utilize shared computation to compute both lower and upper bounds simultaneously.

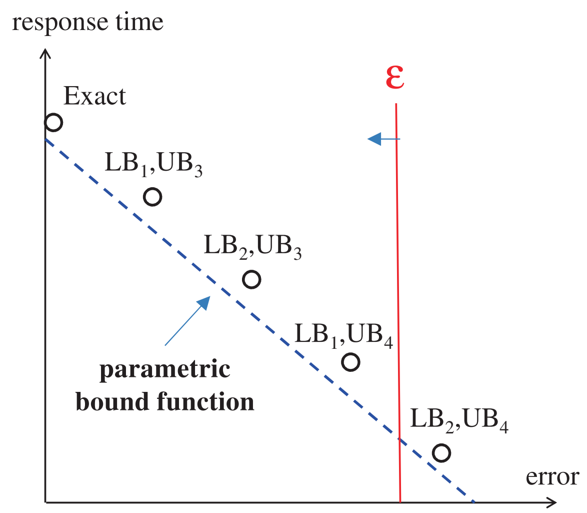


Figure 4.1. Illustration of bound functions

We attempt to tackle our problem in two directions. First, we propose an *adaptive* approach, which does not rely on any training. It gradually invokes tighter bound functions until satisfying the error requirement. Then, we develop an enhancement, called *lightweight adaptive* approach, by reducing the number of calls to bound functions. Finally, we apply training to collect statistics, and exploit them to boost the performance of our solution.

In our experimental study, we will evaluate both the efficiency and the effectiveness of our proposed methods. We will conduct case study on the representative application (i.e., kNN image retrieval) and demonstrate that approximate EMD values yield reasonably accurate results. Our methods achieve an order of magnitude speedup over the fastest exact computation method.

In Section 4.1, we define our problem formally and briefly review existing bound functions for EMD in the literature. We then present our parametric dual bound functions in Section 4.2. After that, we propose our approximation framework and also our solutions in Section 4.3. In Section 4.4, we present experimental results on real datasets. We discuss the related work in Section 4.5 and then conclude in Section 4.6.

4.1 Preliminaries

4.1.1 Problem Definition

The *Earth Mover Distance* (EMD) [102] can be used to measure the dissimilarity between two histograms (e.g., probability distributions). We represent a histogram by $\mathbf{p} = [p_1, p_2, ..., p_d]$, where d is the dimensionality (i.e., number of bins). Following the previous works [102, 117, 103], we assume that each histogram $\mathbf{p}$ is normalized, i.e., $\sum_{j=1}^{d} p_j = 1$. Given two histograms $\mathbf{q}$ and $\mathbf{p}$, the EMD between them is defined as the *minimum-cost flow* on a bipartite flow network between $\mathbf{q}$ and $\mathbf{p}$. We denote a cost matrix by c and a flow matrix by f, where $c_{i,j}$ models the cost of moving flow from q_i to p_j, and $f_{i,j}$ represents the amount of flow to move from q_i to p_j. Formally, we define $emd_c(\mathbf{q}, \mathbf{p})$ as the following linear programming problem.

$$\begin{aligned} emd_c(\mathbf{q},\mathbf{p}) = \quad & \underset{f}{\text{minimize}} \sum_{i=1}^{d}\sum_{j=1}^{d} c_{i,j} f_{i,j} \\ \text{such that} \quad & \forall i,j \in [1..d] : f_{i,j} \geq 0 \\ & \forall i \in [1..d] : \sum_{j=1}^{d} f_{i,j} = q_i \\ & \forall j \in [1..d] : \sum_{i=1}^{d} f_{i,j} = p_j \end{aligned}$$

According to Ref. [102], EMD satisfies the triangle inequality provided that the cost matrix c is a metric (i.e., non-negativity, symmetry and triangle inequality for all $c_{i,j}$).

Lemma 4.1 (Proved in Ref. [102]) *For any histograms* $\mathbf{q}, \mathbf{p}, \mathbf{r}$ *with the same dimensionality, we have:*

$$emd_c(\mathbf{q}, \mathbf{p}) \leq emd_c(\mathbf{q}, \mathbf{r}) + emd_c(\mathbf{r}, \mathbf{p})$$

EMD is computationally expensive. Even with the fastest known algorithm [89], it is still expensive to compute the exact EMD value, which takes $O(d^3 \log d)$ time. Instead, we propose to compute an approximate EMD value R with bounded error. We formulate our problem below; it guarantees that R is within $1 \pm \epsilon$ times the exact EMD value. Our objective is to develop efficient algorithms for this problem.

Problem 4.1 (Error-Bounded EMD) *Given an error threshold* ϵ*, this problem returns a value* R *such that* $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$*, where the relative error of* R *is defined as:*

$$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) = \frac{|R - emd_c(\mathbf{q}, \mathbf{p})|}{emd_c(\mathbf{q}, \mathbf{p})} \tag{4.1}$$

4.1.2 Existing Bound Functions

We introduce existing lower bound and upper bound functions for EMD in the literature, which will be used in subsequent sections. These bound functions must satisfy the following properties (c.f. Definition 4.1).

DEFINITION 4.1 $LB(\mathbf{q},\mathbf{p})$ *is called a lower bound function if* $emd_c(\mathbf{q},\mathbf{p}) \geq LB(\mathbf{q},\mathbf{p})$ *for any* $\mathbf{q},\mathbf{p}$. $UB(\mathbf{q},\mathbf{p})$ *is called an upper bound function if* $emd_c(\mathbf{q},\mathbf{p}) \leq UB(\mathbf{q},\mathbf{p})$ *for any* $\mathbf{q},\mathbf{p}$.

We summarize several representative lower and upper bound functions in the literature in Table 4.1. For each bound function, we show its name (in subscript), its time complexity, and its reference(s). We refer the interested readers to the references.

Most of the bound functions yield time complexities in terms of the histogram dimensionality d. Wichterich et al. [128] propose a *parametric* lower bound function LB_{Red,d_r}, which accepts an additional parameter d_r (i.e., reduced dimensionality) to control its running time and tightness. However, we are not aware of any parametric upper bound function in the literature.

Table 4.1. Summary of lower and upper bound functions for EMD

Name	Type	Time Complexity	Reference	Parametric
LB_{IM}	lower	$O(d^2)$	[9]	no
LB_{Proj}	lower	$O(d)$	[103, 34]	no
LB_{Red,d_r}	lower	$O(d^2 + {d_r}^3 \log d_r)$	[128]	yes
UB_G	upper	$O(d^2)$	[117]	no
UB_H	upper	$O(d)$	[64, 65]	no

4.2 Parametric Dual Bounding

Although there exists a parametric lower bound function [128], we are not aware of any parametric upper bound function in the literature. Instead of providing separate functions for lower bound and upper bound, we propose another *parametric dual bound* function, which utilizes shared computation to compute both lower and upper bounds simultaneously, and offers control on running time and tightness via a parameter. Moreover, our parametric bound functions take advantage of skewed property of data, we provide the case study in Section 4.2.4 to demonstrate our parametric lower bound is generally superior than [128] for small error in this type of datasets.

4.2.1 Exact EMD on Sparse Histograms

Recall from Section 5.1 that the computation of $emd_c(\mathbf{q}, \mathbf{p})$ is equivalent to the minimum-cost flow problem on a bipartite flow network. This flow network contains d^2 edges because there is an edge between each q_i and each p_j.

It turns out that, when both $\mathbf{q}$ and $\mathbf{p}$ are sparse histograms (i.e., having 0 in many bins), it is possible to shrink the flow network without affecting the exact EMD value. Consider the example in Figure 4.2 where the dimensionality is $d = 4$. Since each flow $f_{i,j}$ (from q_i to p_j) must be non-negative, any bin with zero value must have zero flow (to and from other bins). Therefore, we can safely remove the edge for $f_{i,j}$ if either $q_i = 0$ or $p_j = 0$. For example, in Figure 4.2, it suffices to keep only $2 \times 2 = 4$ edges in the flow network.

Formally, we introduce the notation $\Phi(\mathbf{p})$ to measure the denseness of the histogram.

Definition 4.2 *Let* $\Phi(\mathbf{p})$ *be the number of non-zero bins in histogram* $\mathbf{p}$*, i.e.,* $\Phi(\mathbf{p}) = \texttt{COUNT}\{j : p_j \neq 0\}$.

With this idea, we can compute the exact value of $emd_c(\mathbf{q}, \mathbf{p})$ in $O(d_s^3 \log d_s)$

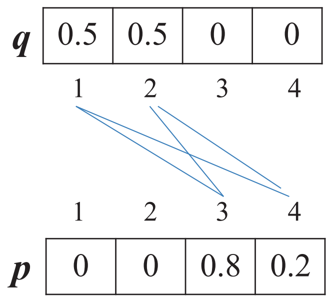


Figure 4.2. Bipartite graph for sparse EMD computation

time, where $d_s = \max(\Phi(\mathbf{q}), \Phi(\mathbf{p}))$.

4.2.2 Skew-Transform Operation

Based on the idea of efficient EMD evaluation on sparse histograms, we propose the *Skew-Transform* operation which will be used for our skew-based bound functions. This operation takes a histogram $\mathbf{p}$ and an integer λ as input, and returns a histogram $\mathbf{p}'$ that contains exactly λ non-zero bins (i.e., $\Phi(\mathbf{p}') = \lambda$). We illustrate an example of this operation in Figure 4.3, with the input $\mathbf{p} = [0.1, 0.1, 0.6, 0.2]$ and $\lambda = 2$. After moving values in bin 1 and bin 2 to bin 3, we obtain the histogram $\mathbf{p}' = [0, 0, 0.8, 0.2]$, which contains exactly two non-zero bins. As we will explain later, an upper bound $ub_{move}(\mathbf{p}, \mathbf{p}')$ can be derived efficiently from the sequence of movements. In this case, we have: $ub_{move}(\mathbf{p}, \mathbf{p}') = 0.1 \cdot c_{2,3} + 0.1 \cdot c_{1,3}$.

We adopt a greedy method to implement the skew-transform operation. First, we select a source bin (say, s) with the smallest non-zero value. Then, we select a target bin (say, t) such that it has non-zero value and the smallest movement cost $c_{s,t}$. We repeat the above procedure until the result histogram $\mathbf{p}'$

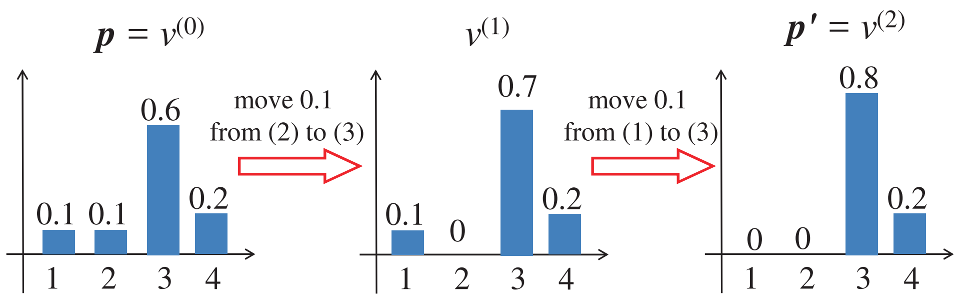


Figure 4.3. Example for skew transform

contains exactly λ non-zero bins. The pseudo-code of this method is described in Algorithm 2.

Algorithm 2 Skew Transform Operation

```
1: procedure SKEW-TRANSFORM( histogram $\mathbf{p}$, cost matrix $c$, integer $\lambda$)
2:     $\mathbf{p}' \leftarrow \mathbf{p}$
3:     $ub_{move} \leftarrow 0$
4:     while $\Phi(\mathbf{p}') > \lambda$ do
5:         $s \leftarrow \arg\min\{i : p'_i \neq 0\}$
6:         $t \leftarrow \arg\min\{j : c_{s,j}, p'_j \neq 0, j \neq s\}$
7:         $\delta \leftarrow p'_s$
8:         $ub_{move} \leftarrow ub_{move} + c_{s,t}\delta$
9:         $p'_t \leftarrow p'_t + \delta$; $p'_s \leftarrow 0$
10:    return $(\mathbf{p}', ub_{move})$
```

The value ub_{move} computed by Algorithm 2 is indeed an upper bound of $emd_c(\mathbf{p}, \mathbf{p}')$. We prove this in the following lemma.

LEMMA 4.2 *When Algorithm 2 terminates, it holds that:* $ub_{move} \geq emd_c(\mathbf{p}, \mathbf{p}')$.

Proof. In each iteration of the Algorithm 2, the movement of value δ from bin s to bin t is feasible; it preserves the summation terms $\sum_{j=1..d} f_{ij} = p_i$, $\forall i$, $\sum_{i=1..d} f_{ij} = p'_j$, $\forall j$ and ensures that each movement incurs non-negative flow from bin s to t. Therefore, $f_{ij} \geq 0$, $\forall i, j$. Therefore, the total movement cost is at least the minimum possible cost $emd_c(\mathbf{p}, \mathbf{p}')$. □

The time complexity of Algorithm 2 is $O((d - \lambda)d)$.

4.2.3 Skew-Based Bound Functions

We illustrate the idea behind our bound functions in Figure 4.4.

- First, we transform histograms $\mathbf{q}$ and $\mathbf{p}$ into sparser histograms $\mathbf{q}'$ and $\mathbf{p}'$. To control their sparsity, we introduce a parameter λ in the transform operation and require that $\Phi(\mathbf{q}') = \Phi(\mathbf{p}') = \lambda$.
- Then, we derive lower and upper bound functions for $emd(\mathbf{q}, \mathbf{p})$ by using the transformed histograms (i.e., $\mathbf{q}', \mathbf{p}'$) and their relationships with the original histograms (i.e., $\mathbf{q}, \mathbf{p}$).

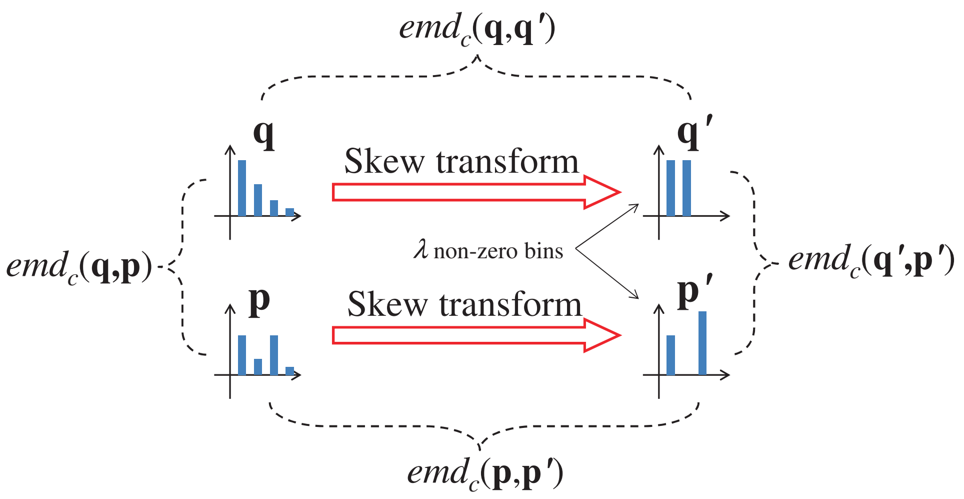

$emd_c(\mathbf{q},\mathbf{q}')$
$\mathbf{q}$
Skew transform
$\mathbf{q}'$
$emd_c(\mathbf{q},\mathbf{p})$
λ non-zero bins
$emd_c(\mathbf{q}',\mathbf{p}')$
$\mathbf{p}$
Skew transform
$\mathbf{p}'$
$emd_c(\mathbf{p},\mathbf{p}')$

Figure 4.4. Skew-based lower and upper bounds

As shown in Figure 4.4, our bounds for $emd_c(\mathbf{q}, \mathbf{p})$ depend on three terms $emd_c(\mathbf{q}, \mathbf{q}')$, $emd_c(\mathbf{p}, \mathbf{p}')$, and $emd_c(\mathbf{q}', \mathbf{p}')$. Since $\mathbf{q}'$ and $\mathbf{p}'$ are sparse, we can compute $emd_c(\mathbf{q}', \mathbf{p}')$ efficiently by the idea in Section 4.2.1. However, this idea cannot be used to accelerate the computation of $emd_c(\mathbf{q}, \mathbf{q'})$, and $emd_c(\mathbf{p}, \mathbf{p'})$. To reduce the computation time, we replace $emd_c(\mathbf{q}, \mathbf{q'}), emd_c(\mathbf{p}, \mathbf{p'})$ by fast-to-compute upper bounds $UB(\mathbf{q}, \mathbf{q'}), UB(\mathbf{p}, \mathbf{p'})$ (cf. Section 4.2.2). Specifically, we propose the following parametric functions in terms of λ and call them as *skew-based bound functions*:

$$\begin{aligned} LB_{skew,\lambda}(\mathbf{q}, \mathbf{p}) &= emd_c(\mathbf{q'}, \mathbf{p'}) - UB(\mathbf{q}, \mathbf{q'}) - UB(\mathbf{p}, \mathbf{p'}) \\ UB_{skew,\lambda}(\mathbf{q}, \mathbf{p}) &= emd_c(\mathbf{q'}, \mathbf{p'}) + UB(\mathbf{q}, \mathbf{q'}) + UB(\mathbf{p}, \mathbf{p'}) \end{aligned} \tag{4.2}$$

such that (i) $\Phi(\mathbf{q'}) = \Phi(\mathbf{p'}) = \lambda$ and (ii) $UB(\cdot, \cdot)$ is an upper bound function of $emd_c(\cdot, \cdot)$.

Observe that both $LB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$ and $UB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$ share all the terms, suggesting an opportunity for shared computation for both bounds.

Lemma 4.3 shows that $LB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$ and $UB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$ are lower and upper bound functions for $emd_c(\mathbf{q}, \mathbf{p})$, respectively.

LEMMA 4.3 *For any histograms* $\mathbf{q}, \mathbf{p}$*, we have:* $LB_{skew,\lambda}(\mathbf{q}, \mathbf{p}) \leq emd_c(\mathbf{q}, \mathbf{p}) \leq UB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$.

Proof. By the triangle inequality (Lemma 4.1), we obtain:

$$emd_c(\mathbf{q}, \mathbf{p'}) \leq emd_c(\mathbf{q}, \mathbf{q'}) + emd_c(\mathbf{q'}, \mathbf{p'})$$
$$emd_c(\mathbf{q}, \mathbf{p}) \leq emd_c(\mathbf{q}, \mathbf{p'}) + emd_c(\mathbf{p'}, \mathbf{p})$$

Adding these two inequalities, we get:

$$\begin{aligned} emd_c(\mathbf{q}, \mathbf{p}) &\leq emd_c(\mathbf{q'}, \mathbf{p'}) + emd_c(\mathbf{q}, \mathbf{q'}) + emd_c(\mathbf{p'}, \mathbf{p}) \\ &\leq emd_c(\mathbf{q'}, \mathbf{p'}) + UB(\mathbf{q}, \mathbf{q'}) + UB(\mathbf{p}, \mathbf{p'}) \end{aligned}$$

This implies that $emd_c(\mathbf{q}, \mathbf{p}) \leq UB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$.

By using Lemma 4.1 in another way, we obtain:

$$emd_c(\mathbf{q}', \mathbf{p}) \geq emd_c(\mathbf{q}', \mathbf{p}') - emd_c(\mathbf{p}', \mathbf{p})$$
$$emd_c(\mathbf{q}, \mathbf{p}) \geq emd_c(\mathbf{q}', \mathbf{p}) - emd_c(\mathbf{q}', \mathbf{q})$$

Adding these two inequalities, we get:

$$\begin{aligned} emd_c(\mathbf{q}, \mathbf{p}) &\geq emd_c(\mathbf{q}', \mathbf{p}') - emd_c(\mathbf{q}', \mathbf{q}) - emd_c(\mathbf{p}', \mathbf{p}) \\ &\geq emd_c(\mathbf{q}', \mathbf{p}') - UB(\mathbf{q}, \mathbf{q}') - UB(\mathbf{p}, \mathbf{p}') \end{aligned}$$

This implies that $emd_c(\mathbf{q}, \mathbf{p}) \geq LB_{skew,\lambda}(\mathbf{q}, \mathbf{p})$. □

Regarding the time complexity, the transformation operation (in Section 4.2.2) takes $O((d-\lambda)d)$ time, and the exact EMD computation on transformed histograms takes $O(\lambda^3 \log \lambda)$ time. Thus, the total time complexity is: $O((d-\lambda)d + \lambda^3 \log \lambda)$.

4.2.4 Case study on Parametric Lower Bound Functions

Recall from Table 4.1, LB_{Red,d_r} [128] is the only parametric lower bound function in the literature. In order to compare the effectiveness of our parametric bound functions, we first sample 1000 $(\mathbf{q}, \mathbf{p})$ pairs from CAL-RGB and CAL-Lab datasets (see Section 4.4.1.1 for details). Then we test the running time (per pair) with respective to the average relative error of each bound, i.e. $\frac{1}{1000}\sum_{(\mathbf{q},\mathbf{p})} E_{\mathbf{q},\mathbf{p}}(LB_{Red,d_r}(\mathbf{q}, \mathbf{p}))$ and $\frac{1}{1000}\sum_{(\mathbf{q},\mathbf{p})} E_{\mathbf{q},\mathbf{p}}(LB_{skew,\lambda}(\mathbf{q}, \mathbf{p}))$. We select the most suitable parameters of λ and d_r such that the relative errors to be approximately 0.05, 0.1, 0.15, 0.2, 0.25 and 0.3. As shown in Figure 4.5, our bound function $LB_{skew,\lambda}$ generally outperforms the existing bound function LB_{Red,d_r} under the small average relative error (e.g. 0 to 0.2). The main reason is our bound function $LB_{skew,\lambda}$ applies the smallest bin value shift first greedy strategy (cf. Figure 4.3) which is more suitable for skewed data. However, LB_{Red,d_r} does not consider this property.

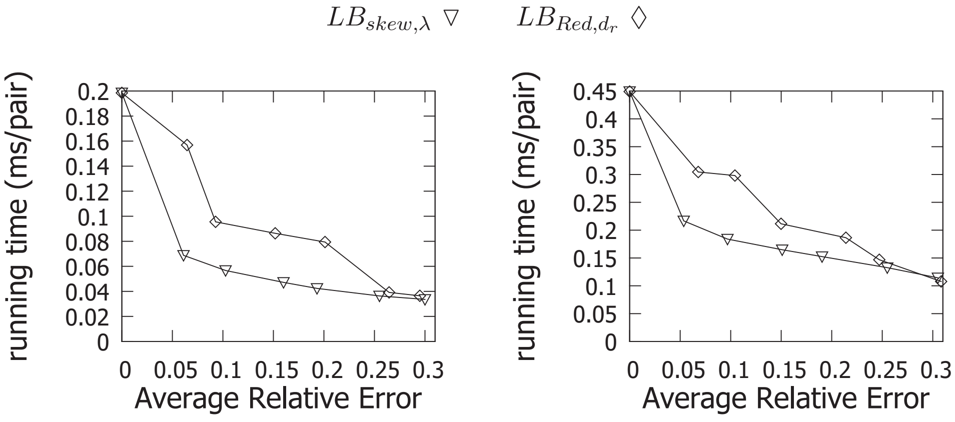


Figure 4.5. Case study for our $LB_{skew,\lambda}$ and LB_{Red,d_r} [128]

4.3 Approximation Framework

Although the lower/upper bound functions (cf. Table 4.1 and Section 4.2) may be used to compute approximate EMD value R, they provide no guarantee on the relative error (i.e., $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$).

In contrast, we propose a framework to compute an approximate EMD value with bounded error. Our framework leverages on lower/upper bound functions for EMD. As shown in Figure 4.6, our framework consists of the following two components.

- The *controller* selects a lower bound function and an upper bound function. Then it computes a lower bound ℓ, an upper bound u, and an approximate result R which is the value between ℓ and u.
- The *validator* receives information (e.g., ℓ, u, R) from the controller, and then checks whether the relative error definitely satisfies $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$.

If the validator returns true, then the controller reports R to the user. Otherwise, the controller needs to obtain tighter bounds for ℓ and u, and repeats the above procedure.

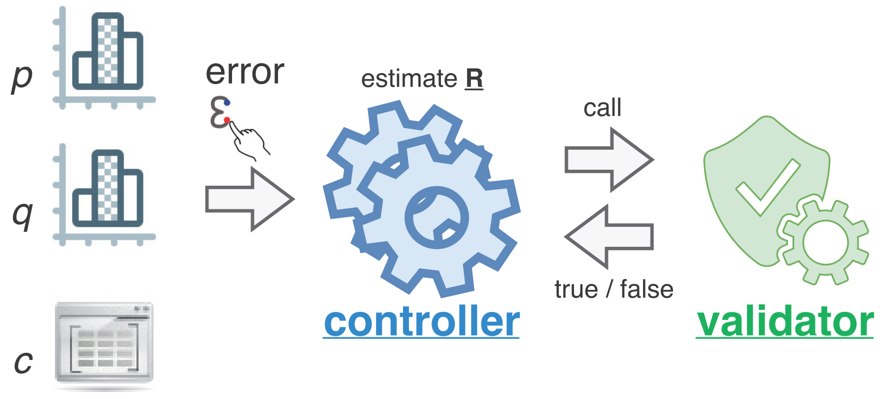


Figure 4.6. Framework

4.3.1 Validator

In order to secure the correctness of our framework, we specify the following requirements for the validator:

- If it returns true, then it guarantees that the approximate result R must satisfy $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$.

- Otherwise, it does not provide any guarantee for R.

For brevity in the remaining subsection, we use ℓ, u and e to represent $LB(\mathbf{q},\mathbf{p})$, $UB(\mathbf{q},\mathbf{p})$ and $emd_c(\mathbf{q},\mathbf{p})$.

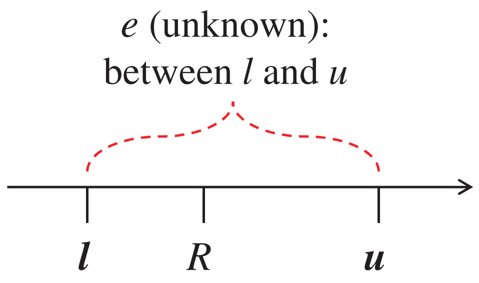


Figure 4.7. Validation

As shown in Figure 4.7, since the validator does not know the exact value e, it cannot directly compute the relative error of R, i.e., $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$. Nevertheless, the validator can compute the *maximum possible* relative error of R, according to Lemma 4.4.

LEMMA 4.4 $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \max\left(\frac{R}{\ell} - 1, 1 - \frac{R}{u}\right)$.

Proof. By the definition of $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$, we have:

$$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) = \frac{|R-e|}{e} = \left|\frac{R}{e} - 1\right|$$

<u>Case 1: $R \geq e$</u>

$$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) = \frac{R}{e} - 1 \leq \frac{R}{\ell} - 1 \quad \text{(By } e \geq \ell\text{)}$$

Case 2: $R < e$

$$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) = 1 - \frac{R}{e} \leq 1 - \frac{R}{u} \quad \text{(By } e \leq u\text{)}$$

Combining both cases, we obtain the following inequality:

$$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \max\left(\frac{R}{\ell} - 1, 1 - \frac{R}{u}\right)$$

□

Our next step is to find the optimal R in order to minimize the maximum possible relative error $\max\left(\frac{R}{\ell} - 1, 1 - \frac{R}{u}\right)$. According to Theorem 4.1, it is minimized when $R = \frac{2\ell u}{\ell+u}$, and the maximum possible relative error becomes $\frac{u-\ell}{u+\ell}$.

In subsequent sections, we use the notation $\mathbb{E}_{max}(\ell, u)$ to represent $\frac{u-\ell}{u+\ell}$, and use the notation $\mathbb{R}(\ell, u)$ to represent $\frac{2\ell u}{\ell+u}$.

Theorem 4.1 *If $R = \frac{2\ell u}{\ell+u}$, then:*

(1) $\max\left(\frac{R}{\ell} - 1, 1 - \frac{R}{u}\right)$ *achieves minimum*

(2) $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \frac{u-\ell}{u+\ell}$.

Proof. For (1), we observe that the first term $\frac{R}{\ell} - 1$ is monotonic increasing with R and the second term $1 - \frac{R}{u}$ is monotonic decreasing with R. In order to minimize it, we set:

$$\frac{R}{\ell} - 1 = 1 - \frac{R}{u} \iff R = \frac{2\ell u}{\ell + u}$$

For (2),

$$\begin{aligned}
\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) &= \frac{|R-e|}{e} = \left|\frac{R}{e} - 1\right| = \left|\frac{2\ell u}{e(\ell+u)} - 1\right| \\
&\leq \max\left(\left|\frac{2\ell u}{\ell(\ell+u)} - 1\right|, \left|\frac{2\ell u}{u(\ell+u)} - 1\right|\right) \\
&= \frac{u-\ell}{u+\ell}
\end{aligned}$$

□

Therefore, once the condition $\frac{u-\ell}{u+\ell} \leq \epsilon$ is fulfilled, $R = \frac{2\ell u}{\ell+u}$ can achieve the bounded error $E_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$.

4.3.2 Training-free Controllers

In this section, we propose two control algorithms: Adaptive (ADA) and Lightweight Adaptive (ADA-L) which are based on our developed parametric lower and upper bound functions (cf. Section 4.2). These two control methods do not have the preprocessing stage.

4.3.2.1 Adaptive (ADA)

Recall from Section 4.2.3, our parametric dual bound functions $LB_{skew,\lambda}$ and $UB_{skew,\lambda}$ depend on the parameter λ, which can affect both the running time and the error. This calls for an automatic method for selecting a suitable value for λ, upon the arrival of the $(\mathbf{q}, \mathbf{p})$-pair.

We propose the adaptive approach (ADA) as illustrated in Figure 4.8. It gradually applies tighter bounds until passing the validation test. We present this method in Algorithm 3. It consists of an adaptive phase which performs validation by our parametric bound functions $LB_{skew,\lambda}, UB_{skew,\lambda}$ in ascending order of λ. We denote the increasing sequence of λ values by Λ (cf. Line 2). The algorithm executes our parametric bound functions in ascending order of $\lambda \in \Lambda$ until passing. It terminates as soon as it passes the validation test.

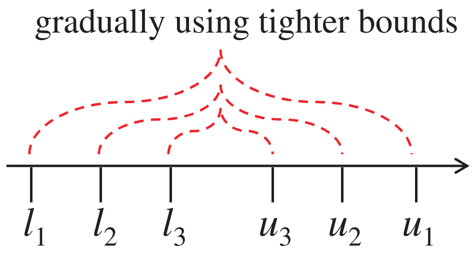


Figure 4.8. Adaptive approach

Algorithm 3 Adaptive Algorithm (ADA)

```
1: procedure ADA( histogram $\mathbf{q}$, histogram $\mathbf{p}$, cost matrix $c$, error threshold $\epsilon$ )
2:     initialize the sequence $\Lambda$ of increasing integers
3:     for each $\lambda \in \Lambda$ do                    ▷ compute $LB_{skew,\lambda}, UB_{skew,\lambda}$
4:         $(\mathbf{q}', ub_1) \leftarrow$Skew-Transform$(\mathbf{q}, \lambda)$
5:         $(\mathbf{p}', ub_2) \leftarrow$Skew-Transform$(\mathbf{p}, \lambda)$
6:         $temp \leftarrow emd_c(\mathbf{q}', \mathbf{p}')$          ▷ expensive call
7:         $\ell \leftarrow temp - ub_1 - ub_2$; $u \leftarrow temp + ub_1 + ub_2$
8:         if $\mathbb{E}_{max}(\ell, u) \leq \epsilon$ then          ▷ Theorem 4.1
9:             return $R = \frac{2\ell u}{\ell + u}$
10:    return $emd_c(\mathbf{q}, \mathbf{p})$                ▷ expensive call
```

We propose one instantiation for Λ below:

- **Exponential sequence:** We introduce a parameter $\alpha > 1$ and construct $\Lambda = \langle \lfloor \alpha^i \rfloor : i \geq 0, \lfloor \alpha^i \rfloor < d \rangle$. For example, when $\alpha = 1.4$ and $d = 25$, the sequence is: $\langle 1, 1, 1, 2, 3, 5, 7, 10, 14, 20 \rangle$. In implementation, we omit duplicate integers in the sequence.

Theoretically, we show that ADA can be worse than the ADA-Opt (which knows the optimal λ value in advance for each $(\mathbf{q}, \mathbf{p})$ pair) by only a constant factor 5.18 if $\alpha = 1.2$.

LEMMA 4.5 *For every* $(\mathbf{q}, \mathbf{p})$ *pair, let* $\mathbb{T}(\mathsf{ADA}(\mathbf{q}, \mathbf{p}))$ *and* $\mathbb{T}(\mathsf{ADA\text{-}Opt}(\mathbf{q}, \mathbf{p}))$ *be the running time of* $\mathsf{ADA}(\mathbf{q}, \mathbf{p})$ *and* $\mathsf{ADA\text{-}Opt}(\mathbf{q}, \mathbf{p})$ *respectively. If* $\alpha = 1.2$*, we have:*

$$\frac{\mathbb{T}(\mathsf{ADA}(\mathbf{q}, \mathbf{p}))}{\mathbb{T}(\mathsf{ADA\text{-}Opt}(\mathbf{q}, \mathbf{p}))} \leq 5.18$$

The detail proof is shown in Appendix (Section 4.7).

4.3.2.2 Lightweight Adaptive (ADA-L)

ADA may examine several λ and compute $emd_c(\mathbf{q}', \mathbf{p}')$ multiple times (in the adaptive phase). Thus, ADA can be expensive when ϵ is small. To avoid such overhead, we propose a lightweight version of the adaptive method such that it computes $emd_c(\mathbf{q}', \mathbf{p}')$ exactly once.

We show this lightweight method (ADA-L) in Figure 4.9. This algorithm (cf. Algorithm 4) applies the skew-transform operation on histograms $\mathbf{q}'$ and $\mathbf{p}'$ such that they have one more zero bin. If the validation condition is satisfied, then we continue the loop. Otherwise, we terminate the loop and return $emd_c(\mathbf{q}', \mathbf{p}')$ as the approximate result.

The correctness of the validation condition is established by the following theorem.

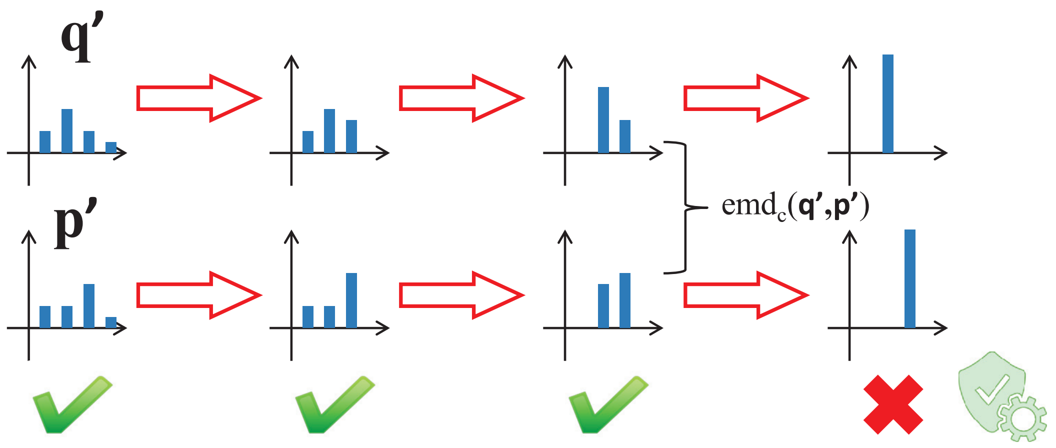


Figure 4.9. Lightweight adaptive approach

THEOREM 4.2 *If* $R = emd_c(\mathbf{q}', \mathbf{p}')$, *then:*

$$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \frac{UB(\mathbf{q}, \mathbf{q}') + UB(\mathbf{p}, \mathbf{p}')}{LB(\mathbf{q}, \mathbf{p})} \tag{4.3}$$

Proof. In the proof of Lemma 4.3, we have: $emd_c(\mathbf{q}, \mathbf{p}) \leq emd_c(\mathbf{q}', \mathbf{p}') + UB(\mathbf{q}, \mathbf{q}') + UB(\mathbf{p}, \mathbf{p}')$ and $emd_c(\mathbf{q}, \mathbf{p}) \geq emd_c(\mathbf{q}', \mathbf{p}') - UB(\mathbf{q}, \mathbf{q}') - UB(\mathbf{p}, \mathbf{p}')$. Thus, we obtain: $|emd_c(\mathbf{q}', \mathbf{p}') - emd_c(\mathbf{q}, \mathbf{p})| \leq UB(\mathbf{q}, \mathbf{q}') + UB(\mathbf{p}, \mathbf{p}')$.

$$\begin{aligned}
\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) &= \frac{|emd_c(\mathbf{q}', \mathbf{p}') - emd_c(\mathbf{q}, \mathbf{p})|}{emd_c(\mathbf{q}, \mathbf{p})} \quad \text{(Given value of } R\text{)} \\
&\leq \frac{UB(\mathbf{q}, \mathbf{q}') + UB(\mathbf{p}, \mathbf{p}')}{emd_c(\mathbf{q}, \mathbf{p})} \\
&\leq \frac{UB(\mathbf{q}, \mathbf{q}') + UB(\mathbf{p}, \mathbf{p}')}{LB(\mathbf{q}, \mathbf{p})} \quad \text{(By Definition 4.1)}
\end{aligned}$$

□

Therefore, once the condition $UB(\mathbf{q}, \mathbf{q}') + UB(\mathbf{p}, \mathbf{p}') \leq \epsilon LB(\mathbf{q}, \mathbf{q}')$ is fulfilled, $R = emd_c(\mathbf{q}', \mathbf{p}')$ can achieve the bounded error $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$.

Before computing R, we can bound the error $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$ by using three terms, namely $UB(\mathbf{q},\mathbf{q}')$, $UB(\mathbf{p},\mathbf{p}')$ and $LB(\mathbf{q},\mathbf{p})$. In the algorithm, the value ℓ (cf. Line 2) corresponds to $LB(\mathbf{q},\mathbf{p})$, and the value $ub_{sum} + ub_1 + ub_2$ (cf. Line 8) corresponds to $UB(\mathbf{q},\mathbf{q}') + UB(\mathbf{p},\mathbf{p}')$.

Algorithm 4 Lightweight Adaptive Method (ADA-L)

```
1: procedure ADA-L( histogram $\mathbf{q}$, histogram $\mathbf{p}$, cost matrix $c$, error threshold $\epsilon$ )
2:     $\ell \leftarrow LB_{Proj}(\mathbf{q},\mathbf{p})$                    ▷ Use the fastest lower bound function
3:     $ub_{sum} \leftarrow 0$
4:     $\mathbf{q}' \leftarrow \mathbf{q}$, $\mathbf{p}' \leftarrow \mathbf{p}$
5:     while $\Phi(\mathbf{q}') > 1$ and $\Phi(\mathbf{p}') > 1$ do
6:         $(\mathbf{q}'', ub_1) \leftarrow$Skew-Transform$(\mathbf{q}', c, \Phi(\mathbf{q}') - 1)$
7:         $(\mathbf{p}'', ub_2) \leftarrow$Skew-Transform$(\mathbf{p}', c, \Phi(\mathbf{p}') - 1)$
8:         if $ub_{sum} + ub_1 + ub_2 \leq \epsilon \cdot \ell$ then                ▷ Theorem 4.2
9:             $ub_{sum} \leftarrow ub_{sum} + ub_1 + ub_2$
10:            $\mathbf{q}' \leftarrow \mathbf{q}''$, $\mathbf{p}' \leftarrow \mathbf{p}''$
11:        else
12:            break
13:    return $emd_c(\mathbf{q}', \mathbf{p}')$                          ▷ expensive call
```

The most appealing property of this ADA-L is that it avoids the expensive call of EMD operation in each iteration. The fast incremental upper bound function (cf. Lemma 4.2) incrementally updates $UB(\mathbf{q},\mathbf{q}')$ and $UB(\mathbf{p},\mathbf{p}')$ which leads to efficient computation in each iteration.

4.3.3 Training-based Controller (ADA-H)

Some applications, e.g., image retrieval [102, 67, 66] and image classification [133], might have huge historical workload data, i.e., $\Gamma =$ set of $(\mathbf{q},\mathbf{p})$. Such rich information can help to pick the bounds such that the framework can find a good approximate result R at lower cost.

As discussed in Table 4.1, there exist several lower bound functions $LB \in Set_{LB}$ and upper bound functions $UB \in Set_{UB}$ at lower cost as compared to our ADA-L. If we can select the fast combination of LB and UB for $(\mathbf{q}, \mathbf{p})$ with the validation condition $E_{max}(\ell, u) \leq \epsilon$ is fulfilled, then the response time can be further reduced. The question is which bound functions in Set_{LB} and Set_{UB} should be picked.

In this work, we propose another control method ADA-H which picks the sequence of bound functions (from Set_{LB} and Set_{UB}) based on ϵ and the statistics of the workload Γ in the offline stage. After obtaining this sequence, we utilize this sequence to handle the online computation.

4.3.3.1 Offline stage

We first build the following tables for Γ.

Definition 4.3 (V_ϵ-Table) *$V_\epsilon(LB, UB)$ denotes the set of $(\mathbf{q}, \mathbf{p})$ pairs where their estimated results (using LB and UB) pass the validation stage subject to the error threshold ϵ.*

$$V_\epsilon(LB, UB) = \{(\mathbf{q}, \mathbf{p}) \in \Gamma | E_{max}(LB(\mathbf{q}, \mathbf{p}), UB(\mathbf{q}, \mathbf{p})) \leq \epsilon\}$$

Definition 4.4 ($\mathbb{T}$-Table) *$\mathbb{T}$-Table records the response time $\mathbb{T}(LB(\mathbf{q}, \mathbf{p}), UB(\mathbf{q}, \mathbf{p}))$ of different bound functions $LB \in Set_{LB}$ and $UB \in Set_{UB}$ for every $(\mathbf{q}, \mathbf{p})$ pair.*

Given the V_ϵ-Table and the $\mathbb{T}$-Table of a workload set Γ, we want to find a sequence of bounds (from Set_{LB} and Set_{UB}) such that the response time of evaluating these bounds is minimized subject to a constraint that all estimated result R of $(\mathbf{q}, \mathbf{p}) \in \Gamma$ satisfies the validation condition $E_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$.

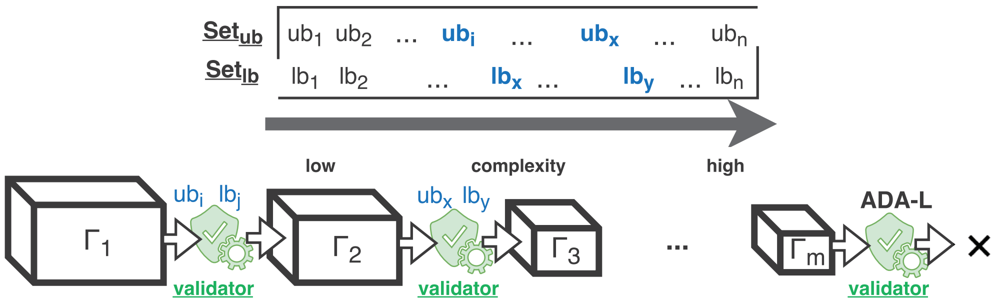


Figure 4.10. Picking a sequence of bounds in the offline stage

Figure 4.10 shows our basic idea. A *feasible* sequence of bounds in this example is (lb_j, ub_i), (lb_y, ub_x), ... , ADA-L. These bounds secure that every pair in Γ fulfills the error threshold ϵ, which is verified by the validator (based on the information in V_ϵ-Table). Note that we need ADA-L at the last step to secure the feasibility (since ADA-L is an adaptive method so it always finds a result fulfilling $E_{\mathbf{q},\mathbf{p}}(R) \leq \epsilon$).

Among all feasible sequences of bounds, we want to find the best sequence of bounds that minimizes the response time (based on the information in $\mathbb{T}$-Table). However, finding the best sequence of bounds that optimizes the objective and fulfills the constraint is a combinatorial problem. For the sake of processing, we simplify the problem and use a greedy method to find the sequence of bounds.

1. We sort the bounds of Set_{LB} and Set_{UB} based on their running time.

2. Pick the fastest pair of bounds into the suggested sequence S and call the validator to partition Γ into Γ_1 and R_1, where R_1 indicates those pairs passing the validator and Γ_1 is the remaining pairs.

3. Get the next fastest pair (LB, UB) of bounds and call the validator to

partition Γ_i into Γ_{i+1} and R_{i+1}. We estimate the response time by the following equation.

$$\sum_{(\mathbf{q},\mathbf{p})\in\Gamma_i} \mathbb{T}(LB(\mathbf{q},\mathbf{p}), UB(\mathbf{q},\mathbf{p})) + \sum_{(\mathbf{q},\mathbf{p})\in\Gamma_{i+1}} \mathbb{T}(\textsf{ADA-L}(\mathbf{q},\mathbf{p})) \tag{4.4}$$

4. If Eq. 4.4 is smaller than $\sum_{(\mathbf{q},\mathbf{p})\in\Gamma_i} \mathbb{T}(\textsf{ADA-L}(\mathbf{q},\mathbf{p}))$, then we pick this pair into the suggested sequence S. Repeat Step (3) until the Γ_i becomes $\emptyset$.

4.3.3.2 Online stage

When processing a new pair $(\mathbf{q},\mathbf{p})$ (of the same application domain), our controller only evaluates those picked sequence of bounds in S (cf. Algorithm 5). This chosen sequence S offers very good performance in practice since the validator skips to check many ineffective bound pairs. We will show the detail performance in the experimental section.

Algorithm 5 ADA-H (Online)

```
1: procedure ADA-H ONLINE( histogram $\mathbf{q}$, histogram $\mathbf{p}$, sequence of bounds $S$, cost matrix
   $c$, error threshold $\epsilon$ )
2:     for each $(LB, UB) \in S$ do
3:         $\ell \leftarrow LB(\mathbf{q},\mathbf{p})$, $u \leftarrow UB(\mathbf{q},\mathbf{p})$
4:         if $\mathbb{E}_{max}(\ell, u) \le \epsilon$ then                ▷ Theorem 4.1
5:             return $R = \frac{2\ell u}{\ell+u}$
6:     Return ADA-L$(\mathbf{q},\mathbf{p},c,\epsilon)$
```

4.4 Experimental Evaluation

We introduce the experimental setting in Section 4.4.1. Then, we evaluate the effectiveness of different bound functions in Section 4.4.2. Next, we present the experiments for approximate EMD computation in Section 4.4.3. Then, we demonstrate the effectiveness and the efficiency of our methods on k-NN content-based image retrieval in Section 4.4.4. We implemented all algorithms in C++ and conducted experiments on an Intel i7 3.4GHz PC running Ubuntu.

4.4.1 Experimental Setting

4.4.1.1 Datasets

We have collected five raw datasets of images as listed in Table 5.6. These datasets have been used extensively in the computer vision and the information retrieval areas. For each raw image, we apply a *histogram extraction method* to obtain a color histogram $\mathbf{p}$.

We consider two representative methods for extracting color histograms, as shown in Table 4.3. RGB color histogram is the traditional representation method since 1990s [116, 54, 115]. It is still effective for content-based image retrieval [40] and EMD-based applications [135]. Lab color histogram is extensively used in computer vision and image retrieval applications [102, 112, 97, 129]. We follow the setting of [40, 102] to extract these two types of color histograms. Specifically, we divide the color space uniformly into $4 \times 4 \times 4$ partitions and $4 \times 8 \times 8$ partitions, for RGB and Lab respectively. According to [102, 117], we compute the cost matrix c by setting $c_{i,j}$ to the Euclidean distance between the centers of partitions i and j in the color space.

By using each histogram extraction method on each raw dataset, we obtain ten datasets: UW-RGB, VOC-RGB, COR-RGB, CAL-RGB, FL-RGB, UW-Lab, VOC-Lab, COR-Lab, CAL-Lab, FL-Lab. We name each dataset by the format [raw dataset]-[histogram name].

Table 4.2. Raw datasets of images

Raw dataset	# of images	Used in
UW	1,109	[40]
VOC	5,011	[42]
COR (Corel)	10,800	[125]
CAL (Caltech)	30,609	[51]
FL (Flickr)	1,000,000	[60]

Table 4.3. Methods for extracting color histograms

Histogram name	Dimensionality	Used in
RGB	64	[40]
Lab	256	[102]

4.4.1.2 Exact EMD computation

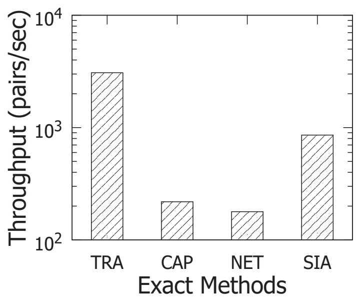
For the sake of fairness, we consider representative methods for computing exact EMD and attempt to identify the fastest one on our datasets. These methods include: (i) two algorithms CAP and NET from the Lemon Graph Library[1], (ii) SIA [117] and (iii) TRA [102].

In this experiment, we randomly sample 1000 pairs of histograms from a dataset, and measure the throughput (number of processed pairs/sec) of each method. Figure 4.11 shows the throughput on two datasets: CAL-RGB and CAL-Lab. Observe that TRA performs the best on both datasets. We obtain similar trends on other datasets. Therefore, we use TRA for exact EMD computation in the remaining experimental study.

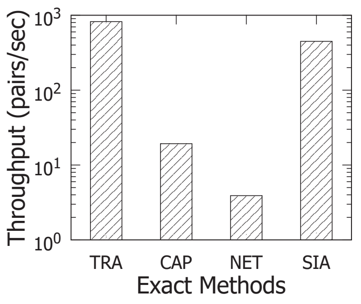
4.4.1.3 Oracle

In order to demonstrate the usefulness of different control algorithms in our approximation framework, we define the following omniscient method Oracle.

[1]http://lemon.cs.elte.hu/trac/lemon



(a) CAL-RGB



(b) CAL-Lab

Figure 4.11. Throughput of exact EMD computation methods

DEFINITION 4.5 *Given histograms* $\mathbf{q}, \mathbf{p}$*, we define the* optimal *pair of lower and upper bound functions as follows:*

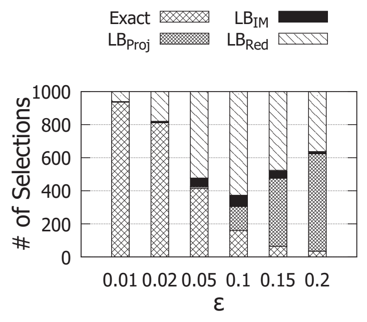
$$\begin{aligned} \mathit{Oracle}(\mathbf{q}, \mathbf{p}) \quad = \quad & \arg\min_{(LB,UB)}\{\mathbb{T}(LB(\mathbf{q}, \mathbf{p}), UB(\mathbf{q}, \mathbf{p})) \\ & : LB \in Set_{LB}, UB \in Set_{UB}, \\ & \mathbb{E}_{max}(LB(\mathbf{q}, \mathbf{p}), UB(\mathbf{q}, \mathbf{p})) \leq \epsilon\} \end{aligned} \tag{4.5}$$

For each $(\mathbf{q}, \mathbf{p})$ pair, Oracle pre-knows the fastest pair of (ℓ, u) which fulfills the validation condition $E_{max}(\ell, u) \leq \epsilon$. As such, it acts as the most efficient solution for all control methods in our approximation framework. In later sections, we will demonstrate how efficient of our solutions compare with Oracle.

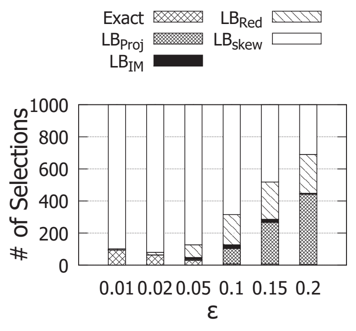
4.4.2 Are Parametric Dual Bound Functions Useful?

In this section, we compare the effectiveness of our derived parametric dual bound functions with existing bounds. Recall from Section 4.4.1.3, Oracle$(\mathbf{q}, \mathbf{p})$ is denoted by the best lower and upper bound pair for $(\mathbf{q}, \mathbf{p})$ pair. Therefore, if the bound is frequently selected by Oracle, that bound is more useful.

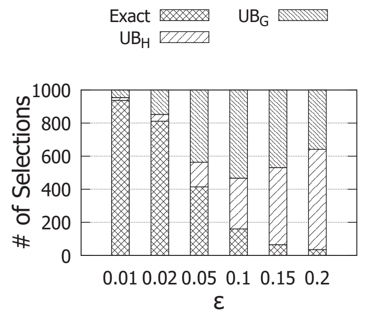
We first randomly sample 1000 $(\mathbf{q}, \mathbf{p})$ pairs of histogram from the dataset CAL-Lab. For a given error threshold ϵ, we count the number of lower and upper bound functions selected by Oracle in these histogram pairs. Observe from Figures 4.12a and c, Exact are frequently chosen at small ϵ (e.g. 0.01 and 0.02). Therefore, existing bound functions are not useful for small ϵ case in which LB_{skew} and UB_{skew} are widely applicable for these cases (c.f. Figures 4.12b and d). Moreover, LB_{skew} and UB_{skew} are frequently selected compared with other bound functions in small to moderate ϵ values (0.01-0.2) by Oracle.



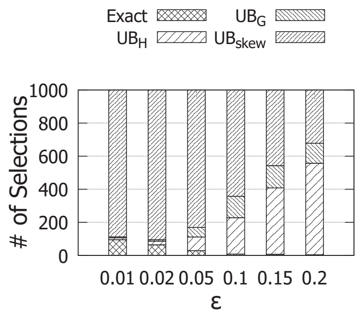
(a) LB Selections



(b) LB Selections (with LB_{skew})



(c) UB Selections



(d) UB Selections (with UB_{skew})

Figure 4.12. Number of lower and upper bound functions selected by Oracle **in CAL-Lab dataset**

4.4.3 Approximate EMD Computation

In this section, we test the throughput and the error of various approximate EMD computation methods. Our competitors are lower/upper bound functions LB_{IM}, LB_{Proj}, LB_{Red}, UB_G, UB_H [9, 103, 128, 117, 64] and an approximate method in the computer vision area FEMD[2] [97]. Our proposed methods are ADA, ADA-L and ADA-H. Note that our methods offer guarantee on error threshold ϵ, but our competitors do not provide such guarantee. By default, we set $\epsilon = 0.2$.

In each dataset, we randomly sample 1000 testing pairs of histograms, and measure the throughput (pairs/sec) of all methods.

4.4.3.1 Effect of pre-processing in ADA-H

The performance of our ADA-H method depends on the number of pairs in the pre-processing steps. For fairness, we make sure that pre-processing pairs are different from testing pairs. In this experiment, we vary the number of pre-processing pairs and plot the throughput in Figure 4.13. Observe that the throughput becomes stable when the numbers of preprocessing pairs are 100, 1000 and 10000. By default, we use 100 pre-processing pairs for ADA-H in subsequent experiments.

Figure 4.14 shows the preprocessing time in COR-RGB and COR-Lab datasets. The preprocessing time is proportional to the number of pairs used for training. However, using 100 training-pairs leads to stable performance in

[2]Implementation at http://www.ariel.ac.il/sites/ofirpele/FastEMD/

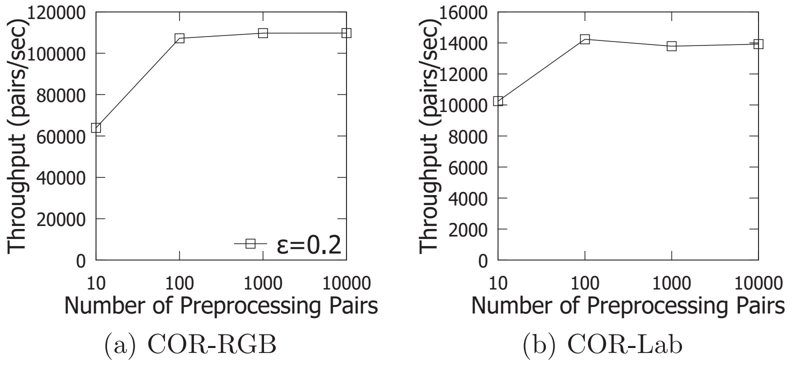


Figure 4.13. Throughput vs. number of pre-processing pairs in ADA-H**, fixing** $\epsilon = 0.2$

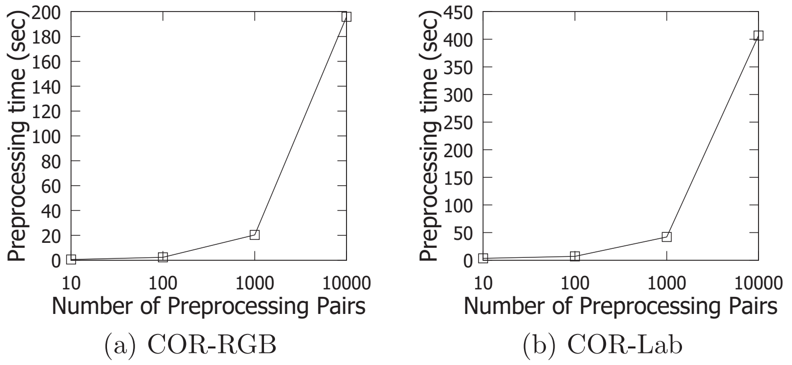


Figure 4.14. Preprocessing time in ADA-H

the online stage, as shown in Figure 4.13. Therefore, the training time is not the bottleneck in general.

4.4.3.2 Comparisons among our methods

In order to conduct meaningful comparisons, we compare our methods with three benchmarks.

- Exact: the fastest exact EMD computation method (TRA), according to Figure 4.11.
- ADA-Opt: an optimal skew method that knows the optimal lambda value in advance. Its throughput serves as the upper bound of our skew-based methods ADA and ADA-L.
- Oracle: the theoretically optimal method, which knows the optimal pair of bound functions in advance (Section 4.4.1.3).

Figure 4.15 plots the throughput of ADA-Opt and our adaptive methods (ADA and ADA-L). Observe that ADA-L can achieve a similar throughput compared to ADA-Opt. Since ADA-L performs better than ADA in practice, we exclude ADA for subsequent experiments. For more details in ADA-L, we provide additional experiments in Appendix (cf. Sections 4.7.2 and 4.7.3).

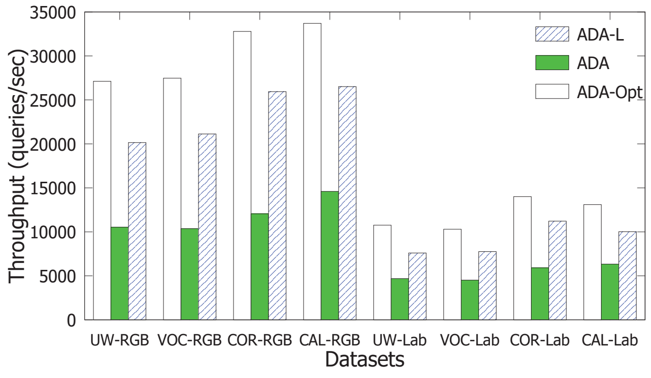


Figure 4.15. Throughput between our methods and ADA-Opt **method, fixing** $\epsilon = 0.2$

In Figure 4.16, we study the effect of the error threshold ϵ on the throughput of our methods ADA-H and ADA-L. We also report the throughput of Oracle and

Exact in this experiment. In general, our methods achieve higher throughput than Exact. Our best method ADA-H can achieve significant speed-up (e.g., by an order of magnitude) on various datasets. Even though Exact can also achieve 1600-7000 pairs/sec in all datasets, which are not slow in general, some applications, for example: kNN-image retrieval and classification [102] or EMD similarity join [59] involve many EMD computations, especially for large-scale datasets, which make Exact inefficient for these applications. We will discuss in detail in our case study (c.f. Section 4.4.4).

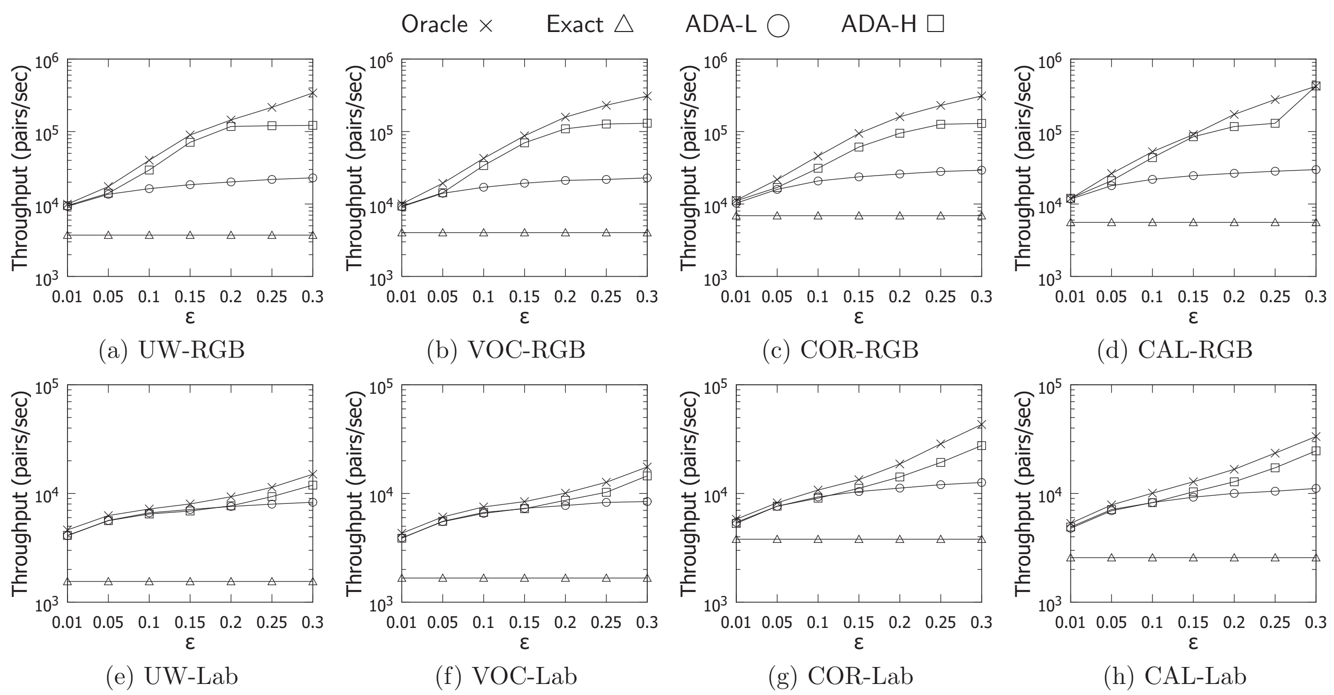
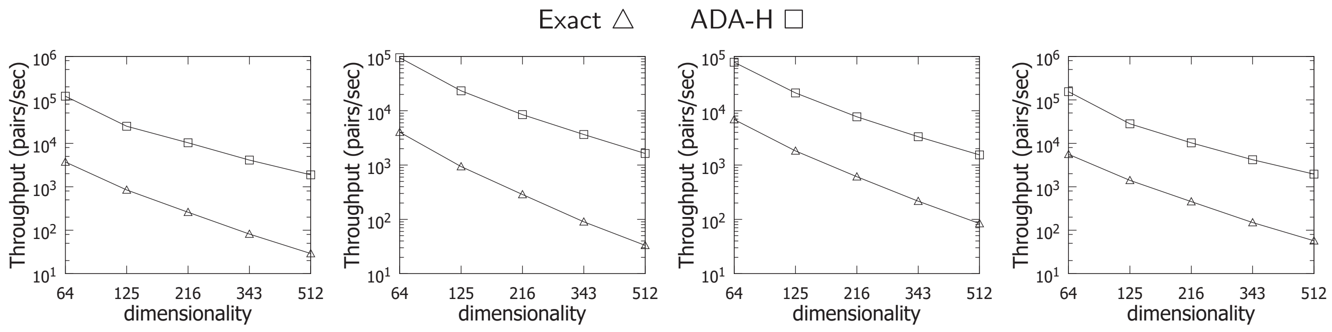


Figure 4.16. Effect of the error threshold ϵ on different datasets

In the next experiment, we vary the dimensionality d of the dataset by using RGB color histogram with $d = m^3$ bins. Figure 4.17 shows the throughput of Exact and ADA-H as a function of the dimensionality. As expected, the throughput decreases when the dimensionality d increases. ADA-H consistently outperforms Exact by an order of magnitude.



(a) UW-RGB (b) VOC-RGB (c) COR-RGB (d) CAL-RGB

Figure 4.17. Effect of the dimensionality d on different datasets

4.4.3.3 Comparisons with competitors

We proceed to compare our best method ADA-H with other approximation methods. We classify our competitors into two types:

- *non-parametric approximation methods* whose throughput cannot be tuned (i.e., LB_{IM}, LB_{Proj}, UB_G, UB_H [9, 103, 117, 64]),
- *parametric approximation methods* whose throughput can be tuned via a parameter (i.e., LB_{Red} [128], FEMD [97], SIA_B [117]).

Table 4.4 shows how we choose the parameter for each parametric approximation method. LB_{Red} [128] is the dimension reduction technique for EMD, we choose different reduced dimensions, d_{red} for conducting this experiment. FEMD [97] utilizes the threshold to truncate the edges (i, j) which costs c_{ij} exceed the threshold in the bipartite flow network of $EMD(\mathbf{q}, \mathbf{p})$. Tang et al. [117] develop the progressive lower bound function and apply UB_G for upper bound function, SIA_B combines these bounding functions with our approximation framework (c.f. Section 4.3). Since SIA_B utilizes our approximation framework, this is the only existing parametric approximate method which can provide the theoretical guarantee of the returned result.

Table 4.4. Parameter tuning

Method	para.	RGB	Lab
LB_{Red}	d_{red} [128]	{12,18,...,60}	{24,56,...,248}
FEMD	Threshold [97]	{50,100,...,350}	{12,24,...,84}
SIA_B, ADA-H	ϵ	{0.01,0.05,0.1,...,0.3}	

In order to obtain a holistic view, we plot the throughput and the error of a method as a point. The error of a method is taken as the average relative error (per tested pair). The performance of all methods are shown in Figure 4.18.

First, we compare the performance of ADA-H with nonparametric approximation methods. Since LB_{Proj}, UB_H, LB_{IM} and UB_G take at most $O(d^2)$ time, they are normally faster than ADA-H (especially for the points with small error) but incur high error. Next, we consider the parametric approximation methods, ADA-H obtains better performance in terms of both throughput and error in most of the tested cases.

4.4.4 Case Study on kNN Content-based Image Retrieval

We conduct case study to demonstrate the effectiveness and the efficiency of methods on kNN content-based image retrieval. Our competitor, denoted by Exact-kNN, is the fastest known method for exact kNN search with EMD [117]. Our kNN search method is the same as Exact-kNN, except that we replace the

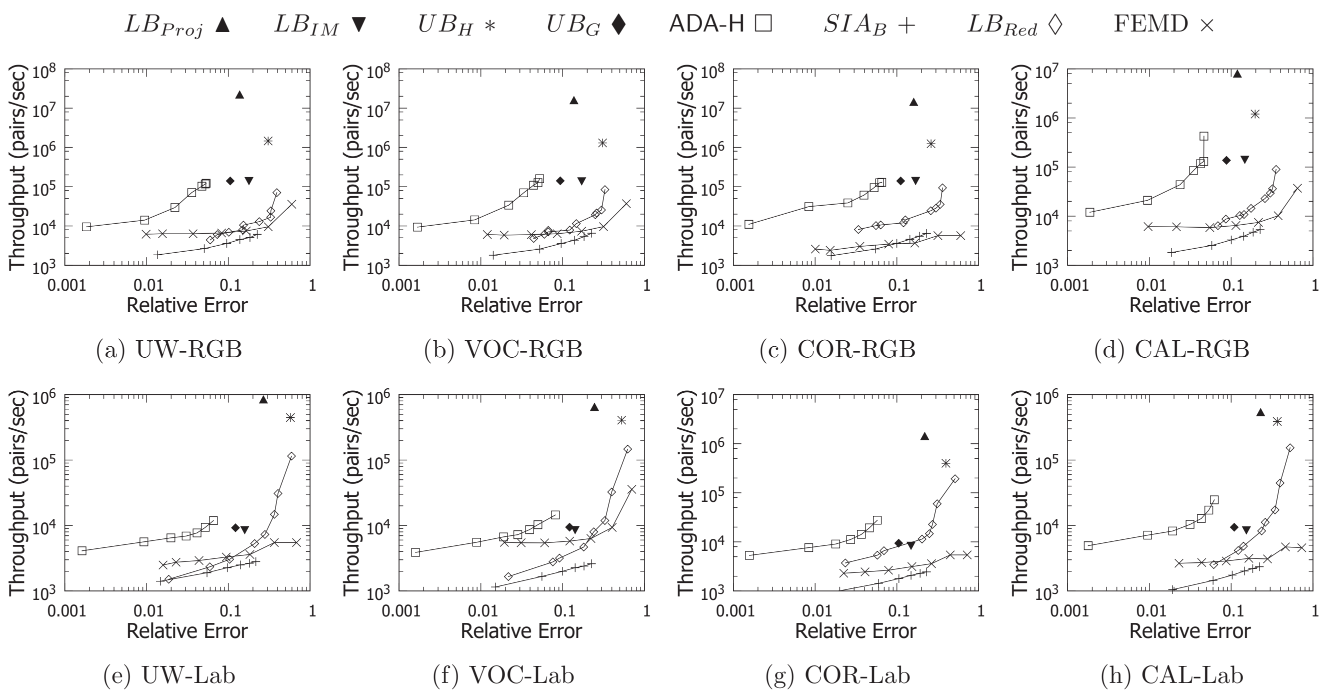
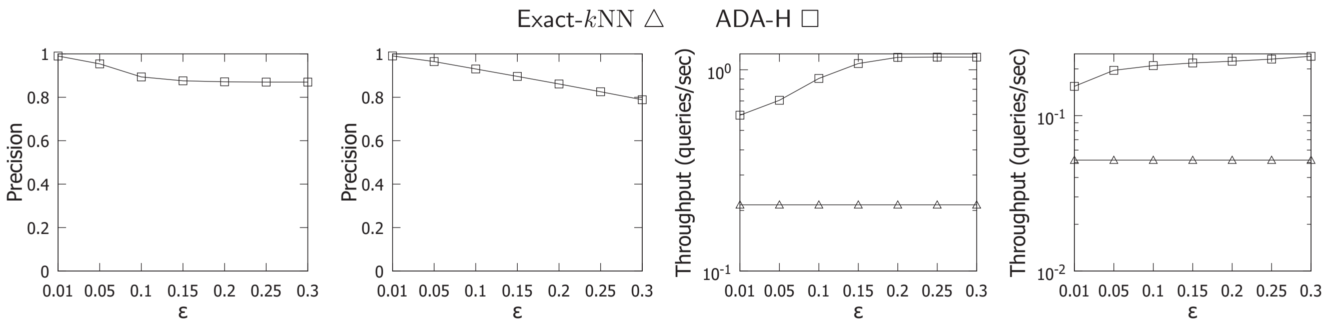


Figure 4.18. Comparisons with all approximation methods on different datasets

refinement stage by our approximate method ADA-H.

We use the largest datasets (FL-RGB, FL-Lab) for testing. In each dataset, we randomly sample 100 query histograms. For each method, we measure its efficiency as the query throughput (queries/sec), and measure its effectiveness as the average precision per query, where the precision is defined as the fraction of the retrieved results in the exact kNN results.

We investigate the effect of ϵ on the kNN retrieval performance in terms of both precision and efficiency. In this experiment, we set $k = 100$ by default and vary ϵ from 0.05 to 0.3. Figure 4.19 shows the precision and the throughput of ADA-H compared with Exact-kNN. Observe that the precision remains above 0.8 (in Figures 4.19(a) and (b)) when ϵ is relatively large (e.g., $\epsilon = 0.3$). Fig-



(a) precision on FL-RGB (b) precision on FL-Lab (c) throughput on FL-RGB (d) throughput on FL-Lab

Figure 4.19. Effect of the error threshold ϵ on the kNN content-based image retrieval, fixing $k = 100$

ures 4.19(c) and (d) demonstrate that ADA-H outperforms Exact-kNN by 3-5x and 3.5-7x on FL-RGB and FL-Lab, respectively.

Next, we test the effect of k on the kNN retrieval performance in terms of both precision and efficiency. Figures 4.20(a) and (b) show the precision of ADA-H as a function of k. The precision of ADA-H is high and it is independent of k. In Figures 4.20(c) and (d), we observe that the throughput of ADA-H is not sensitive to k. On the other hand, the throughput of Exact-kNN is linearly proportional to k. Overall, ADA-H outperforms Exact-kNN by 2.38-5x and 3.38-7.26x on FL-RGB and FL-Lab, respectively.

4.5 Related work

The literature review of the recent bounding functions of Earth Mover's distance have been summarized in Section 2.4.5. In this section, we cover the works directly related to our problem. Then, we mainly discuss the differences

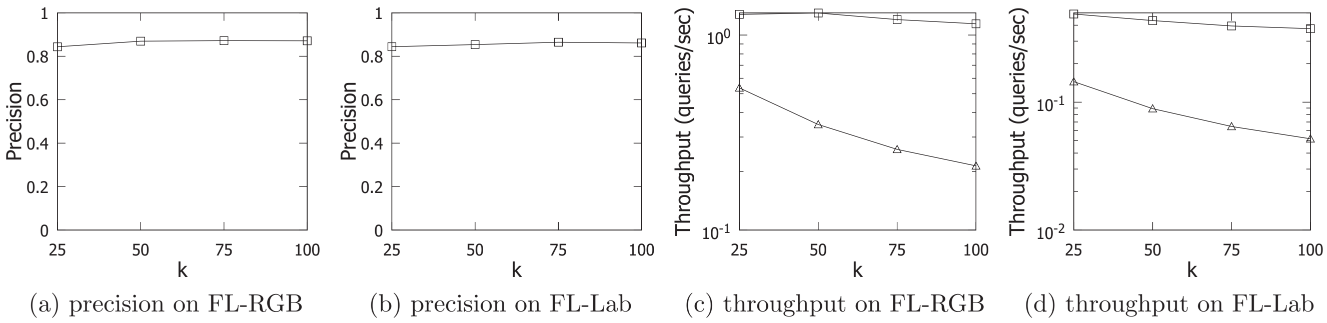


Figure 4.20. Effect of the result size k on the performance of kNN content-based image retrieval, fixing $\epsilon = 0.2$

between our work [21] and previous works.

The *Earth Mover's Distance* (EMD) was first introduced in [101] as a similarity metric in image databases. Comparing to other bin-by-bin distances (e.g., Euclidean distance), the cross-bin calculation makes EMD better match the human perception of differences. EMD can be regarded as a special case of the minimum cost flow problem and many algorithms have been proposed in literature [6], e.g., capacity scaling algorithm, cost scaling algorithm, transportation simplex, and network simplex. However, their worst case time complexity remains super cubic to the number of bins, which limits the applicability of EMD.

In order to employ EMD as a similarity metric in large datasets, the database community attempted to use a filter-and-refinement framework to reduce the number of exact EMD computations. The key factor of the filter-and-refinement framework is to provide a tight lower / upper bound estimation such that more EMD computations can be pruned at the filtering stage. Thereby, there are plenty of EMD bounding techniques [82, 9, 128, 10, 130, 64, 103, 117] being proposed in the database community. In this work, we design a new approximate framework by reutilizing these bounds, which not only provides high quality approximate result (due to the tightness of these bounds) but also reduce the implementation difficulty.

In the theoretical computer science community, there are quite a few of studies [61, 8, 70, 5] in calculating approximate EMD. However, they either focus on a planar graph setting (i.e., calculating EMD on two planar point-sets) [61] or lack of flow concept (i.e., the approximate ratio is analyzed based on a uni-flow model) [8, 70, 5].

Approximate EMD has also been studied in the computer vision community [112, 97, 64]. Pele et al. [97] remove some records from the cost matrix when their values are larger than a pre-defined threshold. The EMD computation time is correlated to the sparsity of the cost matrix so that the threshold plays a role in controlling the quality and the efficiency. Jang et al. [64] store a set of hilbert curves and assign the distance between two images based on these curves. However, the approximate quality is highly relevant to the hilbert curve selection and there is no theoretical guarantee. Shirdhonkar et al. [112] utilize the wavelet theory in their approximation algorithm, which can be viewed as a heuristic solution with no theoretical guarantee.

4.6 Chapter Summary

This chapter studies the computation of approximate EMD value with bounded error. Specifically, we guarantee to return an approximate EMD value that is within $1 \pm \epsilon$ times the exact EMD value. We have presented an adaptive approach for our problem. In our experimental evaluation, we have used five raw image datasets with two histogram extraction methods. Our best method, ADA-H, yields up to an order of magnitude speedup over the fastest exact computation algorithm. We have also evaluated the effectiveness of ADA-H on the kNN content-based image retrieval application. In the future, we plan to investigate how to extend our approximation framework to other applications, e.g. EMD similarity join, and other similarity functions, e.g. edit distance.

4.7 Appendix

4.7.1 The Exponential Sequence Λ in ADA

In the following, we compare the running time of ADA with an ADA-Opt algorithm, which knows additional information in advance. Specifically, ADA-Opt knows the best λ to be chosen for a given $(\mathbf{q}, \mathbf{p})$-pair.

According to the analysis below, by using the exponential sequence with $\alpha = 1.2$, the running time of ADA is bounded by a constant multiple (i.e., 5.18) of the running time of ADA-Opt.

4.7.1.1 Analysis

For the sake of analysis, we model the running time as follows.

$$\mathbb{T}(emd_c(\mathbf{q}, \mathbf{p})) = d^3 \log d \tag{4.6}$$

$$\mathbb{T}(LB_{skew,\lambda}(\mathbf{q}, \mathbf{p}), UB_{skew,\lambda}(\mathbf{q}, \mathbf{p})) = \lambda^3 \log \lambda \tag{4.7}$$

because the state-of-the-art EMD computation algorithm requires $O(d^3 \log d)$ time. We fix the hidden constant factor to 1 and the log base to 2.

Our competitor is the ADA-Opt method, which knows in advance the value

d_* as defined below:

$$d_* = \min\{\lambda : \mathbb{E}_{max}(LB_{skew,\lambda}(\mathbf{q},\mathbf{p}), UB_{skew,\lambda}(\mathbf{q},\mathbf{p})) \leq \epsilon\} \tag{4.8}$$

Therefore, ADA-Opt suffices to call the fastest $LB_{skew,\lambda}$ and $UB_{skew,\lambda}$ once, then passes the validation test. Thus, we have: $\mathbb{T}(\mathsf{ADA\text{-}Opt}(\mathbf{q},\mathbf{p})) = d_*^3 \log d_*$.

We assume that $\mathbb{E}_{max}(LB_{skew,\lambda}(\mathbf{q},\mathbf{p}), UB_{skew,\lambda}(\mathbf{q},\mathbf{p}))$ decreases when λ increases. Therefore, ADA terminates when $\lambda \in \Lambda$ is the smallest integer that satisfies $\lambda \geq d_*$.

We define the ratio of the running time of ADA to ADA-Opt:

$$Ratio = \frac{\mathbb{T}(\mathsf{ADA}(\mathbf{q},\mathbf{p}))}{\mathbb{T}(\mathsf{ADA\text{-}Opt}(\mathbf{q},\mathbf{p}))} \tag{4.9}$$

Theorem 4.3 *Given the exponential sequence* $\Lambda = \langle \lfloor \alpha^i \rfloor : i \geq 0, \lfloor \alpha^i \rfloor < d \rangle$, *we have:*

$$\frac{\mathbb{T}(\mathit{ADA}(\mathbf{q},\mathbf{p}))}{\mathbb{T}(\mathit{ADA\text{-}Opt}(\mathbf{q},\mathbf{p}))} \leq \frac{\alpha^6(1+\log_2 \alpha)}{\alpha^3 - 1}$$

Proof. When $d_* = 1$, the iteration $i = 0$ can directly handle it. We have $Ratio = 1$ in this case. In the remaining discussion, we assume that $d_* > 1$.

Let n be the positive number such that

$$\alpha^{n-1} < d_* \leq \alpha^n \tag{4.10}$$

ADA terminates when it reaches the iteration $n = \lceil \log_\alpha d_* \rceil$. Thus, we have:

$$\mathbb{T}(\mathsf{ADA}(\mathbf{q},\mathbf{p})) = \sum_{i=0}^{n} \lfloor \alpha^i \rfloor^3 \log \lfloor \alpha^i \rfloor \leq \sum_{i=1}^{n} \alpha^{3i} \log \alpha^i$$

$$\begin{aligned}
Ratio &\leq \frac{\sum_{i=1}^{n} \alpha^{3i} \log \alpha^i}{{d_*}^3 \log d_*} \\
&\leq \frac{n \log \alpha}{{d_*}^3 \log d_*} \cdot \sum_{i=1}^{n} \alpha^{3i} \\
&= \frac{n \log \alpha}{{d_*}^3 \log d_*} \cdot \frac{\alpha^3(\alpha^{3n} - 1)}{\alpha^3 - 1}
\end{aligned}$$

Since $n = \lceil \log_\alpha d_* \rceil$, we have $n \leq \log_\alpha d_* + 1$ and $\alpha^{3n} \leq \alpha^{3(\log_\alpha d_* + 1)} = {d_*}^3 \alpha^3$.

Therefore:

$$\begin{aligned}
Ratio &\leq \frac{(\log_\alpha d_* + 1) \log \alpha}{{d_*}^3 \log d_*} \cdot \frac{\alpha^3({d_*}^3 \alpha^3 - 1)}{\alpha^3 - 1} \\
&= \frac{\alpha^3({d_*}^3 \alpha^3 - 1)}{{d_*}^3(\alpha^3 - 1)} \cdot \left(\frac{\log d_*}{\log \alpha} + 1 \right) \cdot \frac{\log \alpha}{\log d_*} \\
&= \frac{\alpha^3({d_*}^3 \alpha^3 - 1)}{{d_*}^3(\alpha^3 - 1)} \cdot (1 + \log_{d_*} \alpha) \\
&= \frac{\alpha^3}{\alpha^3 - 1} \cdot \left(\alpha^3 - \frac{1}{{d_*}^3} \right)(1 + \log_{d_*} \alpha)
\end{aligned}$$

Since $\log_{d_*} \alpha \leq \log_2 \alpha$ and $-\frac{1}{d_*} \leq 0$, we have:

$$Ratio \leq \frac{\alpha^6(1 + \log_2 \alpha)}{\alpha^3 - 1}$$

□

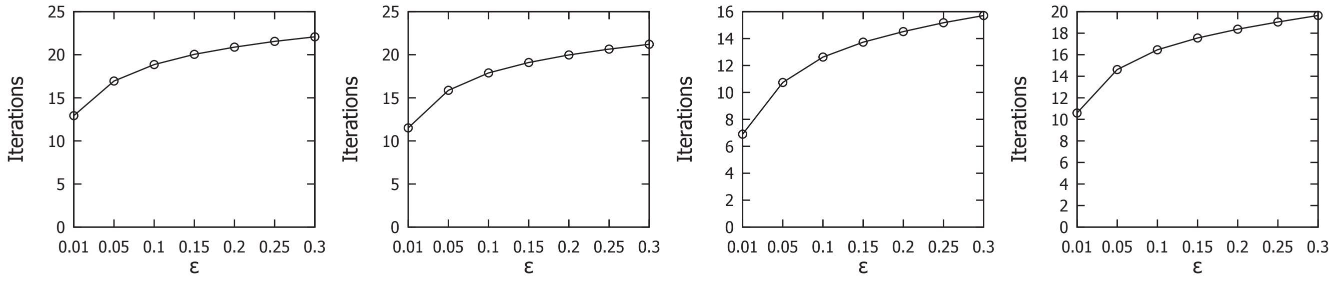
Corollary 4.1 *Given that $\alpha = 1.2$, we have:*

$$\frac{\mathbb{T}(\mathit{ADA}(\mathbf{q}, \mathbf{p}))}{\mathbb{T}(\mathit{ADA\text{-}Opt}(\mathbf{q}, \mathbf{p}))} \leq 5.18$$

Proof. By finding the minimum value of $\frac{\alpha^6(1+\log_2 \alpha)}{\alpha^3 - 1}$ with a numerical solver. □

4.7.2 Number of iterations for ADA-L

We provide the additional experimental study for the number of iterations for ADA-L (i.e. skew operations in Figure 4.9) with the variations of different ϵ, as shown in Figure 4.21. When ϵ is larger, more skew operations can be performed. However, once the number of iterations is larger, it is harder for the feature vectors to be skewed more, as each non-zero value dimension contains larger value (cf. Figure 4.9) and thus, the cost to skew one more dimension is larger. This explains the slow trend increasing when ϵ is larger.



(a) UW-RGB (b) VOC-RGB (c) COR-RGB (d) CAL-RGB

Figure 4.21. Number of iterations for ADA-L**, varying ϵ from 0.01 to 0.3**

4.7.3 Tightness of the inequality in Theorem 4.2

We first define the term $\mathbb{E}^{skew}_{\mathbf{q},\mathbf{p}}(R)$ and fix $\epsilon = 0.2$.

$$\mathbb{E}^{skew}_{\mathbf{q},\mathbf{p}}(R) = \frac{UB(\mathbf{q},\mathbf{q}') + UB(\mathbf{p},\mathbf{p}')}{LB(\mathbf{q},\mathbf{p})} \tag{4.11}$$

Table 4.5 shows the average $\mathbb{E}^{skew}_{\mathbf{q},\mathbf{p}}(R)$ and $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$. Even though the upper bound $\mathbb{E}^{skew}_{\mathbf{q},\mathbf{p}}(R)$ can achieve nearly 0.2, the average relative error $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$ still retain smaller than 0.05 in practice. In our future work, we exploit whether we

Table 4.5. Average $\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$ and $\mathbb{E}^{skew}_{\mathbf{q},\mathbf{p}}(R)$ in different datasets, $\epsilon = 0.2$

Error	UW-RGB	VOC-RGB	COR-RGB	CAL-RGB
$\mathbb{E}_{\mathbf{q},\mathbf{p}}(R)$	0.035	0.038	0.035	0.042
$\mathbb{E}^{skew}_{\mathbf{q},\mathbf{p}}(R)$	0.164	0.165	0.161	0.156

can further tighten this inequality.

Chapter 5

KARL: Fast Kernel Aggregation Queries

In this era of digitalization, a vast amount of data are being continuously collected and analyzed. Kernel functions are typically used in two tasks: (i) kernel density estimation (for statistical analysis) and (ii) support vector machine classification (for data mining). These tasks are actively used in the following applications. Network security systems [17, 15] utilize kernel SVM to detect suspicious packets. In medical science, medical scientists [32] utilize kernel SVM to identify tumor samples. Astronomical scientists [4] utilize kernel density estimation for quantifying the galaxy density. In particle physics, physicists utilize kernel density estimation to search for particles [36]. For example, Figure 5.1 illustrates the usage of kernel density estimation on a real dataset (miniboone [2]) for searching particles. Physicists are interested in the dense region (in yellow).

Implementation-wise, both commercial database systems (e.g., Oracle, Ver-

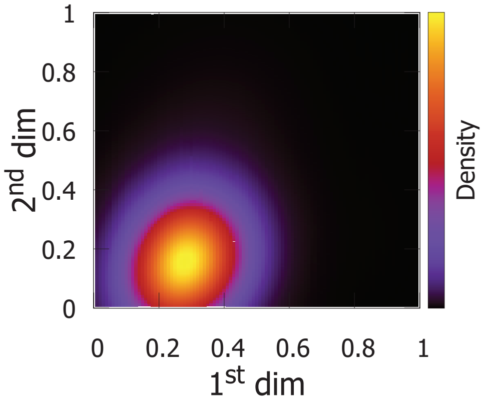


Figure 5.1. Kernel density estimation on the miniboone dataset, using 1^{st} and 2^{nd} dimensions

tica) and open-source libraries (e.g., LibSVM [24]) provide functions for support vector machines (SVM), which can combine with different kernel functions.

In the above applications, a common online operation is to compute the following function:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) \tag{5.1}$$

where $\mathbf{q}$ is a query point, P is a dataset of points, w_i, γ are scalars, and $dist(\mathbf{q}, \mathbf{p_i})$ denotes the Euclidean distance between $\mathbf{q}$ and $\mathbf{p_i}$. A typical problem, which we term as the *threshold kernel aggregation query* (τKAQ), is to test whether $\mathcal{F}_P(\mathbf{q})$ is higher than a given threshold τ [107]. This creates an opportunity for achieving speedup. Instead of computing the exact $\mathcal{F}_P(\mathbf{q})$, it suffices to compute lower/upper bounds of $\mathcal{F}_P(\mathbf{q})$ and then compare them with the threshold τ.

In addition, different types of weighting (for w_i) have been used in different statistical/learning models, as summarized in Table 5.1. Although there exist several techniques to speedup the computation of $\mathcal{F}_P(\mathbf{q})$, each work focuses on one type of weighting only [50, 46, 68, 57, 78]. In contrast, this paper intends to handle the kernel aggregation query under all types of weightings.

Table 5.1. Types of weighting in $\mathcal{F}_P(\mathbf{q})$

Type of weighting	Used in model	Techniques
Type I: identical, positive w_i (most specific)	Kernel density [50, 46]	Quality-preserving solutions [50, 46]
Type II: positive w_i (subsuming Type I)	1-class SVM [86]	Heuristics [81]
Type III: no restriction on w_i (subsuming Types I, II)	2-class SVM [107]	Heuristics [68, 57, 78]

The above query is expensive as it takes $O(nd)$ time to compute $\mathcal{F}_P(\mathbf{q})$ online, where d is the dimensionality of data points and n is the cardinality of the dataset P. In the machine learning community, many recent works [78, 57, 68] also complain the inefficiency issue for computing kernel aggregation, which are quoted as follows:

- *"Despite their successes, what makes kernel methods difficult to use in many large scale problems is the fact that computing the decision function is typically expensive, especially at prediction time."* [78]
- *"However, computing the decision function for the new test samples is typically expensive which limits the applicability of kernel methods to real-world applications."* [57]
- *"..., it has the disadvantage of requiring relatively large computations in the testing phase"* [68]

Existing solutions are divided into two camps. The machine learning community tends to improve the response time by using heuristics [68, 57, 78, 81] (e.g., sampling points in P), which may affect the quality of the model (e.g., classification/prediction accuracy). The other camp, which we are interested in, aims to enhance the efficiency while preserving the quality of the model. The pioneering solutions in this category are [50, 46], albeit they are only applicable to queries with Type I weighting (see Table 5.1). Their idea [50, 46] is to build an index structure on the point set P offline, and then exploit index nodes to derive lower/upper bounds and attempt pruning for online queries.

In this paper, we identify several important research issues that have not yet been addressed in [50, 46], as listed below:

1. **Tighter bound functions:** How to design lower/upper bound functions that are always tighter than existing ones? How to compute them quickly?

2. **Type of weighting:** The techniques in [50, 46] are applicable to Type I weighting only (see Table 5.1). Can we develop a general solution for all types of weighting?

3. **Automatic index tuning:** The performance of a solution may vary greatly across different types of index structures. How to develop an automatic index tuning technique for achieving the best possible efficiency?

4. **In-situ scenario:** In this scenario, the entire dataset is not known in advance. An example scenario is online kernel learning [38, 83, 73], in which the model (e.g., dataset P) would be updated frequently. The end-to-end response time includes the index construction time and the tuning time as well. How to develop a quick tuning technique while enjoying the benefit of a reasonably-good index structure?

Our proposal is **K**ernel **A**ggregation **R**apid **L**ibrary (KARL)[1], a comprehensive solution for addressing all the issues mentioned above. It utilizes a novel bounding technique and index tuning in order to achieve excellent efficiency. Experimental studies on many real datasets reveal that our proposed method achieves speedups of 2.5-738 over the state-of-the-art.

Two widely-used libraries, namely LibSVM [24] and Scikit-learn [94], provide convenient programming support for practitioners to handle kernel aggregation queries. Implementation-wise, LibSVM is based on the sequential scan method, and Scikit-learn is based on the algorithm in [50] for query type I. We compare them with our proposal (KARL) in Table 5.2. As a remark, since Scikit-learn supports query types II and III via the wrapper of LibSVM [94], we remove those two query types from the row of Scikit-learn in Table 5.2. The features of KARL are: (i) it supports all three types of weightings as well as both ϵKAQ and τKAQ queries, (ii) it supports index structures, (iii) it yields much lower response time than existing libraries.

Table 5.2. Comparisons of libraries

Library	Supported queries	Supported weightings	Support indexing	Response time
LibSVM [24]	τKAQ	Types I, II, III	no	high
Scikit-learn [94]	ϵKAQ	Type I	yes	high
KARL (this paper)	ϵKAQ, τKAQ	Types I, II, III	yes	low

We first introduce the preliminaries in Section 5.1, and then present our solution in Section 5.2. We later extend our techniques for different types of weighting and kernel functions in Section 5.3. After that, we present our experiments in Section 5.4. Then, we present our related work in Section 5.5. Lastly,

[1]https://github.com/edisonchan2013928/KARL-Fast-Kernel-Aggregation-Queries

we conclude the paper with future research directions in Section 5.6.

5.1 Preliminaries

We consider two popular types of kernel aggregation queries in the literature [107, 50]. The first variant is to test whether $\mathcal{F}_P(\mathbf{q})$ is higher than a threshold [107]. We term this as the *threshold kernel aggregation query* (τKAQ), which simply tests whether $\mathcal{F}_P(\mathbf{q}) \geq \tau$, where τ is a given threshold. The second variant is to compute an approximate value of $\mathcal{F}_P(\mathbf{q})$ with accuracy bound [50]. We call this as *approximate kernel aggregation query* (ϵKAQ), which returns an approximate value within $(1 \pm \epsilon)$ times the exact value of $\mathcal{F}_P(\mathbf{q})$.

5.1.1 Problem Statement

First, we reiterate the kernel aggregation query (KAQ) as discussed in the introduction.

Definition 5.1 (KAQ) *Given a query point* $\mathbf{q}$ *and a set of points* P*, this query computes:*

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \, \mathcal{K}(\mathbf{q}, \mathbf{p_i}) \tag{5.2}$$

where w_i *is a scalar indicating the weight of the* i*-th term, and* $\mathcal{K}(\mathbf{q}, \mathbf{p_i})$ *denotes the kernel function.*

In the machine learning and statistics communities [24, 107, 124], the typical kernel functions are the Gaussian kernel function, the polynomial kernel function, and the sigmoid kernel function. For example, the Gaussian kernel function is expressed as $\mathcal{K}(\mathbf{q}, \mathbf{p_i}) = \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2)$, where γ is a positive scalar denoting smoothing parameter, and $dist(\mathbf{q}, \mathbf{p_i})$ denotes the Euclidean distance between $\mathbf{q}, \mathbf{p_i}$.

Then we formally define two variants of KAQ: *threshold kernel aggregation query* (τKAQ) [107] and *approximate kernel aggregation query* (ϵKAQ) [50].

Problem 5.1 (τKAQ) *Given a threshold value τ, a query point $\mathbf{q}$, and a set of points P, this problem returns a Boolean value denoting whether $\mathcal{F}_P(\mathbf{q}) \geq \tau$.*

Problem 5.2 (ϵKAQ) *Given a relative error value ϵ, a query point $\mathbf{q}$, and a set of points P, this problem returns an approximate value $\widehat{F}$ such that its relative error (from the exact value $\mathcal{F}_P(\mathbf{q})$) is at most ϵ, i.e.,*

$$(1-\epsilon)\mathcal{F}_P(\mathbf{q}) \leq \widehat{F} \leq (1+\epsilon)\mathcal{F}_P(\mathbf{q}) \tag{5.3}$$

Table 5.3 summarizes the types of queries that can be used for each application model. Table 5.4 summarizes the frequently-used symbols in this paper.

Table 5.3. Example applications for the above queries

Application model	Relevant queries	Obtained from training/learning	Specified by user
Kernel density [50, 46]	ϵKAQ, τKAQ	N.A.	query point $\mathbf{q}$, point set P, parameters ϵ, τ, γ
1-class SVM [86]	τKAQ	point set P, weights w_i, parameters τ, γ	query point $\mathbf{q}$
2-class SVM [107]	τKAQ	point set P, weights w_i, parameters τ, γ	query point $\mathbf{q}$

Table 5.4. Symbols

Symbol	Description
P	Point set
$\mathcal{F}_P(\mathbf{q})$	Kernel aggregation function (Equation 5.2)
$\mathcal{K}(\mathbf{q},\mathbf{p})$	Kernel (e.g., Gaussian, polynomial)
$Lin_{m,c}(x)$	Linear function $mx+c$
$\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$	Linear lower bound of $\mathcal{F}_P(\mathbf{q})$
$\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u})$	Linear upper bound of $\mathcal{F}_P(\mathbf{q})$
$dist(\mathbf{q},\mathbf{p})$	Euclidean distance between $\mathbf{q}$ and $\mathbf{p}$

5.1.2 State-of-the-Art (SOTA)

We proceed to introduce the state-of-the-art [50, 46] (SOTA), albeit it is only applicable to queries with Type I weighting (see Table 5.1). In this case, we denote the common weight by w.

Bounding functions.

We introduce the concept of *bounding rectangle* [105] below.

Definition 5.2 *Let R be the bounding rectangle for a point set P. We denote its interval in the j-th dimension as $[R[j].l, R[j].u]$, where $R[j].l = \min_{\mathbf{p}\in P} \mathbf{p}[j]$ and $R[j].u = \max_{\mathbf{p}\in P} \mathbf{p}[j]$.*

Given a query point $\mathbf{q}$, we can compute the minimum distance $mindist(\mathbf{q}, R)$ from $\mathbf{q}$ to R, and the maximum distance $maxdist(\mathbf{q}, R)$ from $\mathbf{q}$ to R.

It holds that $mindist(\mathbf{q}, R) \leq dist(\mathbf{q},\mathbf{p}) \leq maxdist(\mathbf{q}, R)$ for every point $\mathbf{p}$ inside R.

With the above notations, the lower bound $LB_R(\mathbf{q})$ and the upper bound

$UB_R(\mathbf{q})$ for $\mathcal{F}_P(\mathbf{q})$ (Equation 5.1) are defined as:

$$\begin{aligned} LB_R(\mathbf{q}) &= w \cdot R.\texttt{count} \cdot \exp(-\gamma \cdot maxdist(\mathbf{q}, R)^2) \\ UB_R(\mathbf{q}) &= w \cdot R.\texttt{count} \cdot \exp(-\gamma \cdot mindist(\mathbf{q}, R)^2) \end{aligned}$$

where $R.\texttt{count}$ denotes the number of points (from P) in R, and w denotes the common weight (for Type I weighting). It takes $O(d)$ time to compute the above bounds online.

Refinement of bounds.

The state-of-the-art [50, 46] employs a hierarchical index structure (e.g., k-d tree) to index the point set P. Consider the example index in Figure 5.2. Each non-leaf entry (e.g., $R_5, 9$) stores the bounding rectangle of its subtree (e.g., R_5) and the number of points in its subtree (e.g., 9).

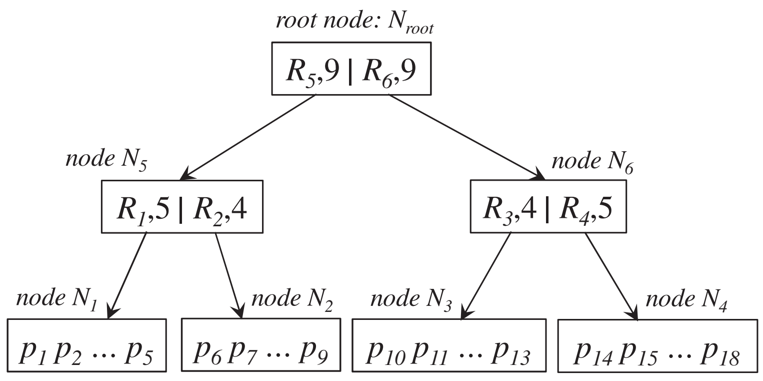


Figure 5.2. Hierarchical index structure

We illustrate the running steps of the state-of-the-art on the above example index in Table 5.5. For conciseness, the notations $LB_R(\mathbf{q}), UB_R(\mathbf{q}), \mathcal{F}_P(\mathbf{q})$ are abbreviated as $lb_R, ub_R, \mathcal{F}_P$ respectively. The state-of-the-art maintains a lower bound $\widehat{lb}$ and upper bound $\widehat{ub}$ for $\mathcal{F}_P(\mathbf{q})$. Initially, the bounding rectangle of the root node (say, R_{root}) is used to compute $\widehat{lb}$ and $\widehat{ub}$. It uses a priority queue to manage the index entries that contribute to the computation of those bounds; the priority of an index entry R_i is defined as the difference $ub_{R_i} - lb_{R_i}$. In each iteration, the algorithm pops an entry R_i from the priority queue, processes the child entries of R_i, then refines the bounds incrementally and updates the priority queue. For example, in step 2, the algorithm pops the entry R_5 from the priority queue, inserts its child entries R_1, R_2 into the priority queue, and refines the bounds incrementally. Similar technique can be also found in similarity search community (e.g., [19, 20]).

Table 5.5. Running steps for state-of-the-art

Step	Priority queue	Maintenance of lower bound $\widehat{lb}$ and upper bound $\widehat{ub}$
1	R_{root}	$\widehat{lb} = lb_{R_{root}}$, $\widehat{ub} = ub_{R_{root}}$
2	R_5, R_6	$\widehat{lb} = lb_{R_5} + lb_{R_6}$, $\widehat{ub} = ub_{R_5} + ub_{R_6}$
3	R_6, R_1, R_2	$\widehat{lb} = lb_{R_6} + lb_{R_1} + lb_{R_2}$, $\widehat{ub} = ub_{R_6} + ub_{R_1} + ub_{R_2}$
4	R_1, R_2, R_3, R_4	$\widehat{lb} = lb_{R_1} + lb_{R_2} + lb_{R_3} + lb_{R_4}$, $\widehat{ub} = ub_{R_1} + ub_{R_2} + ub_{R_3} + ub_{R_4}$
5	R_2, R_3, R_4	$\widehat{lb} = \mathcal{F}_{p_1 \cdots p_5} + lb_{R_2} + lb_{R_3} + lb_{R_4}$, $\widehat{ub} = \mathcal{F}_{p_1 \cdots p_5} + ub_{R_2} + ub_{R_3} + ub_{R_4}$

The state-of-the-art terminates upon reaching a termination condition. For τKAQ, the termination condition is: $\widehat{lb} \geq \tau$ or $\widehat{ub} < \tau$. For ϵKAQ, the termination condition is: $\widehat{ub} < (1+\epsilon)\widehat{lb}$.

5.2 Our Solution: KARL

Our proposed solution, KARL, adopts the state-of-the-art (SOTA) for query processing, except that existing bound functions (e.g., $LB_R(\mathbf{q})$ and $UB_R(\mathbf{q})$) are replaced by our bound functions.

Our key contribution is to develop tighter bound functions for $\mathcal{F}_P(\mathbf{q})$. In Section 5.2.1, we propose a novel idea to bound the function $\exp(-x)$ and discuss how to compute such bound functions quickly. In Section 5.2.2, we devise tighter bound functions and show that they are always tighter than existing bound functions. Then, we discuss automatic tuning in Section 5.2.3.

In this section, we assume using Type I weighting and the Gaussian kernel in the function $\mathcal{F}_P(\mathbf{q})$. We leave the extensions to other types of weighting and kernel functions in Section 5.3.

5.2.1 Fast Linear Bound Functions

We wish to design bound functions such that (i) they are tighter than existing bound functions (cf. Section 5.1.2), and (ii) they are efficient to compute, i.e., taking only $O(d)$ computation time.

In this section, we assume Type I weighting and denote the common weight by w. Consider an example on the dataset $P = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}\}$. Let x_i denotes the value $\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2$. With this notation, the value $\mathcal{F}_P(\mathbf{q})$ can be simplified to:

$$w\Big(\exp(-x_1) + \exp(-x_2) + \exp(-x_3)\Big).$$

In Figure 5.3, we plot the function value $\exp(-x)$ for x_1, x_2, x_3 as points.

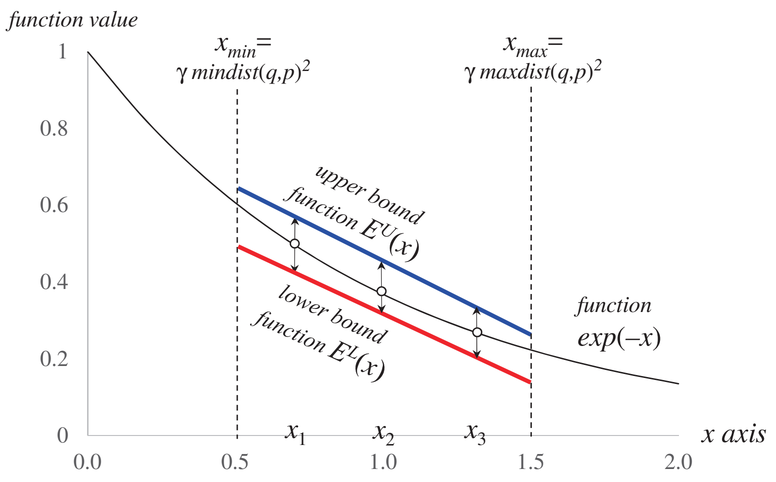


Figure 5.3. Linear bounds

We first sketch our idea for bounding $\mathcal{F}_P(\mathbf{q})$. First, we compute the bounding interval of x_i, i.e., the interval $[x_{min}, x_{max}]$, where $x_{min} = \gamma \cdot mindist(\mathbf{q}, R)^2$, $x_{max} = \gamma \cdot maxdist(\mathbf{q}, R)^2$, and R is the bounding rectangle of P. Within that interval, we employ two functions $E^L(x)$ and $E^U(x)$ as lower and upper bound functions for $\exp(-x)$, respectively (see Definition 5.3). We illustrate these two functions by a red line and a blue line in Figure 5.3.

Definition 5.3 (Constrained bound functions) *Given a query point* $\mathbf{q}$ *and a point set* P*, we call two functions* $E^L(x)$ *and* $E^U(x)$ *to be lower and upper bound functions for* $\exp(-x)$*, respectively, if*

$$E^L(x) \leq \exp(-x) \leq E^U(x)$$

holds for any $x \in [\gamma \cdot mindist(\mathbf{q}, R)^2, \gamma \cdot maxdist(\mathbf{q}, R)^2]$*, where* R *is the bounding* *rectangle of P.*

In this paper, we model bound functions $E^L(x)$ and $E^U(x)$ by using two *linear functions* $Lin_{m_l,c_l}(x) = m_l x + c_l$ and $Lin_{m_u,c_u}(x) = m_u x + c_u$, respectively. Then, we define the aggregation of a linear function $Lin_{m,c}$ as:

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = \sum_{\mathbf{p_i} \in P} w\Big(m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c\Big) \tag{5.4}$$

With this concept, the functions $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ and $\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u})$ serve as a lower and an upper bound function for $\mathcal{F}_P(\mathbf{q})$, subject to the condition stated in the following lemma:

LEMMA 5.1 *Suppose that Lin_{m_l,c_l} and Lin_{m_u,c_u} are lower and upper bound functions for* $\exp(-x)$*, respectively, for the query point* $\mathbf{q}$ *and point set P. It holds that:*

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l}) \leq \mathcal{F}_P(\mathbf{q}) \leq \mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u}) \tag{5.5}$$

Observe that the bound functions in Figure 5.3 are not tight. We will devise tighter bound functions in the next subsection.

Fast computation of bounds.

The following lemma allows $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ to be efficiently computed, i.e., in $O(d)$ time.

LEMMA 5.2 *Given two values m and c, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ (Equation 5.4) can be* *computed in $O(d)$ time and it holds that:*

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = wm\gamma\Big(|P| \cdot ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{a_P} + b_P\Big) + wc|P|$$

where $\mathbf{a_P} = \sum_{\mathbf{p_i} \in P} \mathbf{p_i}$ *and* $b_P = \sum_{\mathbf{p_i} \in P} ||\mathbf{p_i}||^2$.

Proof.

$$\begin{aligned}
\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) &= \sum_{\mathbf{p_i} \in P} w\Big(m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c\Big) \\
&= wm\gamma \sum_{\mathbf{p_i} \in P} \Big(||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{p_i} + ||\mathbf{p_i}||^2\Big) + wc|P| \\
&= wm\gamma\Big(|P| \cdot ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{a_P} + b_P\Big) + wc|P|
\end{aligned}$$

Observe that both terms $\mathbf{a_P}$ and b_P are independent of the query point $\mathbf{q}$. Therefore, with the pre-computed values of $\mathbf{a_P}$ and b_P, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ can be computed in $O(d)$ time. □

5.2.2 Tighter Bound Functions

We proceed to devise tighter bound functions by using Lin_{m_l,c_l} and Lin_{m_u,c_u}.

Linear function Lin_{m_u,c_u} for modeling $E^U(x)$.

Recall that, by using the query point $\mathbf{q}$ and the bounding rectangle R (of point set P), we obtain the bounding interval $[x_{min}, x_{max}]$, where $x_{min} = \gamma \cdot mindist(\mathbf{q}, R)^2$ and $x_{max} = \gamma \cdot maxdist(\mathbf{q}, R)^2$. Since $\exp(-x)$ is a convex function, the chord between two points (say, $(x_{min}, \exp(-x_{min}))$ and $(x_{max}, \exp(-x_{max}))$ must always reside above the curve $\exp(-x)$. We illustrate this in Figure 5.4.

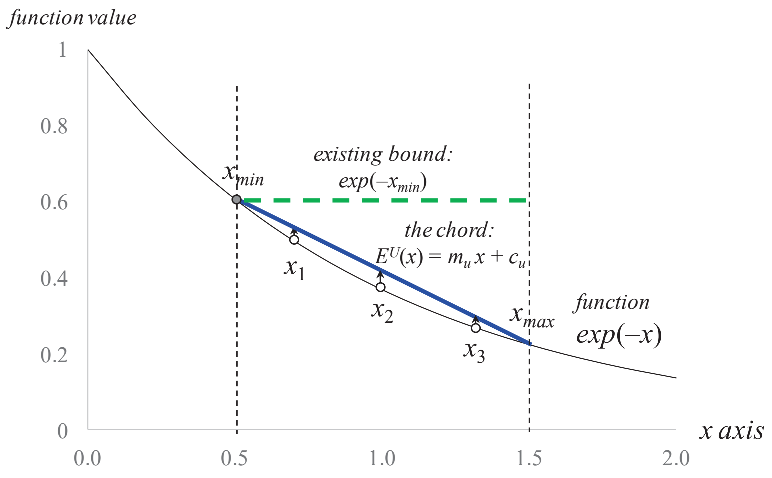


Figure 5.4. Chord-based upper bound function

Regarding the linear function Lin_{m_u,c_u}, its slope m_u and the intercept c_u are computed as:

$$m_u = \frac{\exp(-x_{max}) - \exp(-x_{min})}{x_{max} - x_{min}} \tag{5.6}$$

$$c_u = \frac{x_{max}\exp(-x_{min}) - x_{min}\exp(-x_{max})}{x_{max} - x_{min}} \tag{5.7}$$

It turns out that the above chord-based linear function Lin_{m_u,c_u} leads to a tighter upper bound than the existing bound $\exp(-x_{min})$ (see Section 5.1.2). Clearly, as shown in Figure 5.4, the projected values on the blue line (Lin_{m_u,c_u}) are smaller than the existing bound $\exp(-x_{min})$ (green dashed line in Figure 5.4).

Lemma 5.3 *There exists a linear function Lin_{m_u,c_u} such that $\mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u}) \leq UB_R(\mathbf{q})$, where $UB_R(\mathbf{q})$ is the upper bound function used in the state-of-the-art (see Section 5.1.2).*

Linear function Lin_{m_l,c_l} for modeling $E^L(x)$.

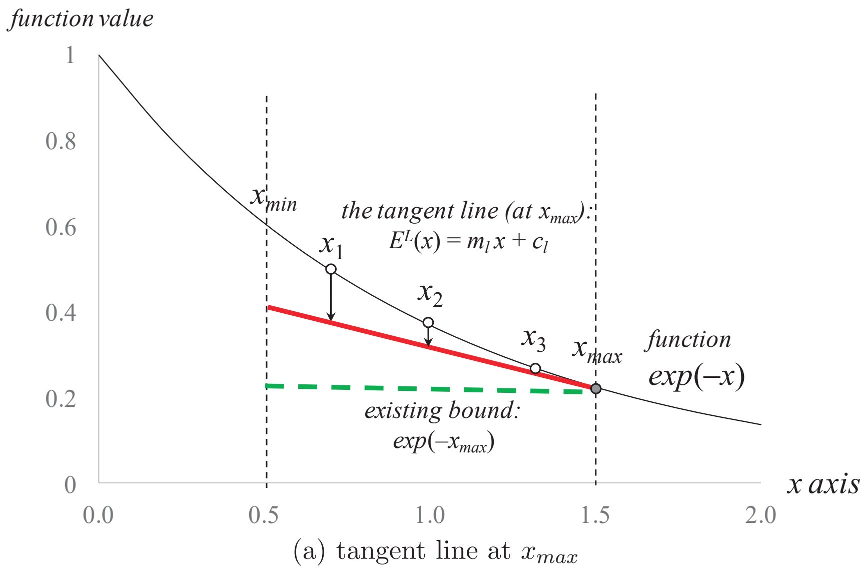
We exploit a property of convex function [52], namely that, any tangent line of a convex function must be a lower bound of the function. This property is applicable to $\exp(-x)$ because it is also a convex function.

We illustrate the above property in Figure 5.5a. For example, the tangent line of function $\exp(-x)$ at point $(x_{max}, \exp(-x_{max}))$ serves as a lower bound function for $\exp(-x)$. Furthermore, this lower bound is already tighter than the existing bound $\exp(-x_{max})$ (see Section 5.1.2). Note that in Figure 5.5a, the projected values on the red line (Lin_{m_l,c_l}) are higher than the existing bound $\exp(-x_{max})$ (green dashed line in Figure 5.5a).

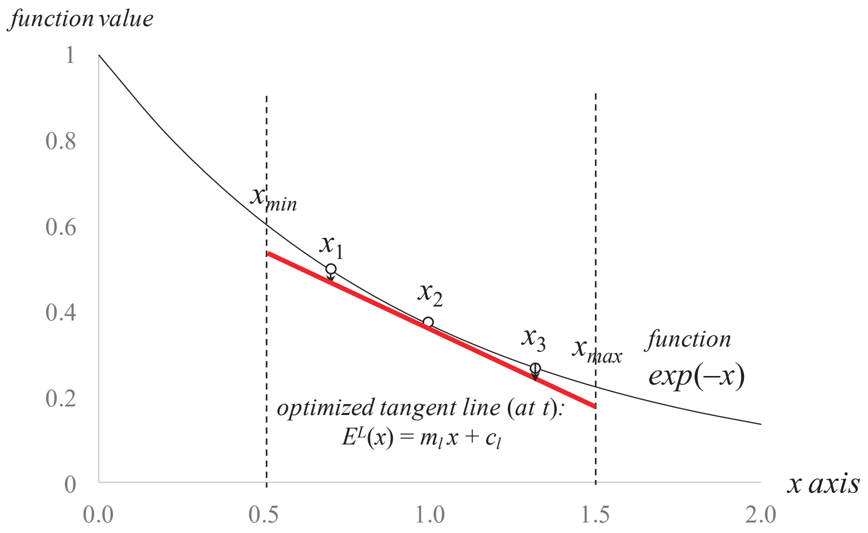
LEMMA 5.4 *There exists a linear function Lin_{m_l,c_l} such that $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l}) \geq LB_R(\mathbf{q})$, where $LB_R(\mathbf{q})$ is the lower bound function used in the state-of-the-art (see Section 5.1.2).*

Interestingly, it is possible to devise a tighter bound than the above. Figure 5.5b depicts the tangent line at point $(t, \exp(-t))$. This tangent line offers a much tighter bound than the one in Figure 5.5a.

In the following, we demonstrate how to find the optimal tangent line (i.e., leading to the tightest bound). Suppose that the lower bound linear function Lin_{m_l,c_l} is the tangent line at point $(t, \exp(-t))$. Then, we derive the slope m_l



(a) tangent line at x_{max}



(b) optimized tangent line at t

Figure 5.5. Tangent-based lower bound function

and the intercept c_l as:

$$\begin{aligned} m_l &= \left.\frac{d\exp(-x)}{dx}\right|_{x=t} = -\exp(-t) \\ c_l &= \exp(-t) - m_l t = (1+t)\exp(-t) \end{aligned}$$

The following theorem establishes the optimal value t_{opt} that leads to the tightest bound.

THEOREM 5.1 *Consider the function $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ as a function of t, where $m_l = -\exp(-t)$ and $c_l = (1+t)\exp(-t)$. This function yields the maximum value at:*

$$t_{opt} = \frac{\gamma}{|P|} \cdot \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \tag{5.8}$$

Proof. Let $H(t) = \mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ be a function of t. For the sake of clarity, we define the following two constants that are independent of t:

$$z_1 = w\gamma \cdot \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \text{ and } z_2 = w|P|$$

Together with the given m_l and c_l, we can rewrite $H(t)$ as:

$$H(t) = -z_1 \exp(-t) + z_2(1+t)\exp(-t)$$

The remaining proof is to find the maximum value of $H(t)$. We first compute the first derivative of $H(t)$ (in terms of t):

$$\begin{aligned} H'(t) &= z_1 \exp(-t) + z_2 \exp(-t) - z_2(1+t)\exp(-t) \\ &= (z_1 + z_2 - z_2 - z_2 t)\exp(-t) \\ &= (z_1 - z_2 t)\exp(-t) \end{aligned}$$

Next, we compute the value t_{opt} such that $H'(t_{opt}) = 0$. Since $exp(-t_{opt}) \neq$

0, we get:

$$\begin{aligned} z_1 - z_2 t_{opt} &= 0 \\ t_{opt} &= \frac{z_1}{z_2} = \frac{\gamma}{|P|} \cdot \sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2 \end{aligned}$$

Then we further test whether t_{opt} indeed yields the maximum value. We consider two cases for $H'(t)$. Note that both z_1 and z_2 are positive constants.

1. For the case $t > t_{opt}$, we get $H'(t) < 0$, implying that $H(t)$ is a decreasing function
2. For the case $t < t_{opt}$, we get $H'(t) > 0$, implying that $H(t)$ is an increasing function.

Thus, we conclude that the function $H(t)$ yields the maximum value at $t = t_{opt}$.

□

The optimal value t_{opt} involves the term $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$. This term can be computed efficiently in $O(d)$ time by the trick in Lemma 5.2. By applying Lemma 5.2 and substituting $w = m = \gamma = 1$ and $c = 0$, we can express $\sum_{\mathbf{p_i} \in P} dist(\mathbf{q}, \mathbf{p_i})^2$ in the form of $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$, which can be computed in $O(d)$ time.

Case study.

We conduct the following case study on the augmented k-d tree, in order to demonstrate the performance of KARL and the tightness of our bound functions compared to existing bound functions. First, we pick a random query point from the home dataset [2] (see Section 5.4.1 for details). Then, we plot the lower/upper bound values of SOTA and KARL versus the number of iterations. Observe that our bounds are much tighter than existing bounds, and thus KARL terminates sooner than SOTA.

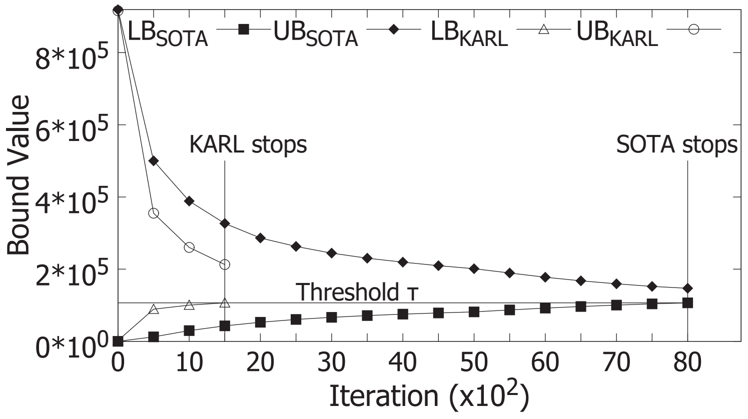


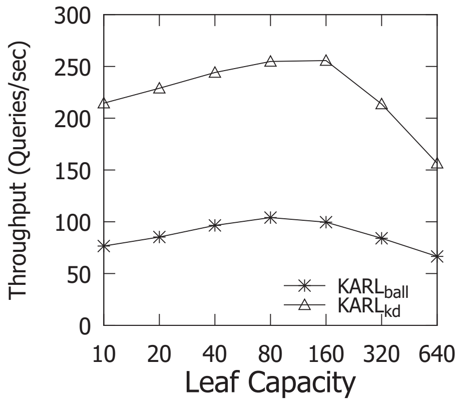
Figure 5.6. Bound values of SOTA and KARL vs. the number of iterations; for type I-τ query on the home dataset

5.2.3 Automatic Tuning

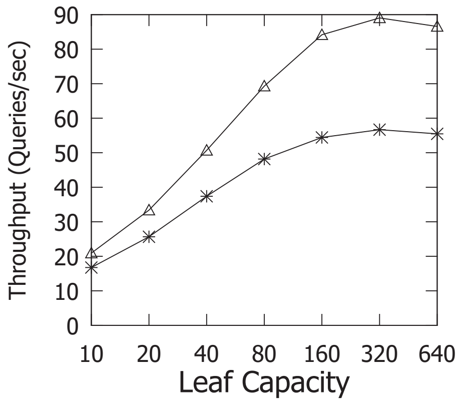
The performance of KARL depends on the choices of the index structure and the index height. Popular index structures include the k-d tree and the ball tree, which are also supported in an existing machine learning library (e.g., Scikit-learn [94]). In addition, the height of such index structure can be controlled via the parameter ‘leaf node capacity’ (i.e., the maximum number of points per leaf node).

To demonstrate the above effect, we conduct the following test by using different index structures with different values of leaf node capacity. Then we measure the throughput (i.e., the number of processed queries per second) of KARL in each index structure. Figures 5.7a and b show the throughput of KARL on two datasets (home and susy respectively). In each figure, the speedup of the best choice to the worst choice can be up to 4 times. Furthermore, the optimal choice can be different on different datasets.

To tackle the above issue, we propose some automatic tuning techniques.



(a) dataset home



(b) dataset susy

Figure 5.7. The throughput of query type I-τ, varying the leaf node capacity

Offline tuning scenario.

In this scenario, we are given ample time for tuning and the dataset is provided in advance.

Observe that two index structures with similar leaf node capacity (e.g., 100 and 101) tend to offer similar performance. It is sufficient to vary the leaf node capacity in an exponential manner (e.g., 10,20,40,80,160,320,640). Next, we build an index structure for each parameter value and for each index type. Finally, we sample a small subset Q of query points, then recommend the index structure having the highest throughput on Q. According to our experimental results, it is enough to set the sample size to $|Q| = 1000$.

In-situ scenario: online tuning.
This scenario is more challenging because the dataset is not known in advance. The end-to-end response time includes index construction time, tuning time, and query execution time. To achieve high throughput, we should reduce the index construction time and the tuning time, while figuring out a reasonably good index.

First, we recommend to build the k-d tree due to its low construction time. It suffices to build a single k-d tree with all levels (i.e., $\log_2(n)$ levels). Denote the entire tree by $\mathcal{T}$, and the tree with the top i levels by $\mathcal{T}_i$. Observe that the tree $\mathcal{T}_i$ can be simulated by using the entire tree $\mathcal{T}$, by skipping lower/upper bound computations in the lowest $\log_2(n) - i$ levels of $\mathcal{T}$.

Suppose that we are given the number of queries to be executed. We sample s% (say, 1%) of those queries and then partition them into $\log_2(n)$ groups. For the i-group, we run its sample queries on the tree $\mathcal{T}_i$. Then, we obtain the value i^* that yields the best performance. Finally, we execute the remaining $(100-s)$% of queries on the k-d tree $\mathcal{T}_{i^*}$.

5.3 Extensions

The state-of-the-art solution has not considered other types of weighting nor other types of kernel functions. In this section, we adapt our bounding techniques (in Sections 5.2.1 and 5.2.2) to address these issues.

5.3.1 Other Types of Weighting

We extend our bounding techniques for the following function:

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i} \in P} w_i \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2)$$

under other types of weighting.

5.3.1.1 Type II Weighting

For Type II weighting, each w_i takes a positive value. Note that different w_i may take different values.

First, we redefine the aggregation of a linear function $Lin_{m,c}$ as:

$$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) = \sum_{\mathbf{p_i} \in P} w_i \Big(m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c \Big) \tag{5.9}$$

This function can also be computed efficiently (i.e., in $O(d)$ time), as shown in the following lemma.

Lemma 5.5 *Under Type II weighting, $\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ (Equation 5.9) can be computed in $O(d)$ time, given two values of m and c.*

Proof.

$$\begin{aligned}
& \mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) \\
= \ & \sum_{\mathbf{p_i} \in P} w_i \Big(m(\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) + c \Big) \\
= \ & \sum_{\mathbf{p_i} \in P} w_i \Big(m\gamma \Big(||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{p_i} + ||\mathbf{p_i}||^2 \Big) \Big) + c \sum_{\mathbf{p_i} \in P} w_i \\
= \ & m\gamma \Big(w_P \cdot ||\mathbf{q}||^2 - 2\mathbf{q} \cdot \mathbf{a_P} + b_P \Big) + c w_P
\end{aligned}$$

where $\mathbf{a_P} = \sum_{\mathbf{p_i} \in P} w_i \mathbf{p_i}$, $b_P = \sum_{\mathbf{p_i} \in P} w_i ||\mathbf{p_i}||^2$ and $w_P = \sum_{\mathbf{p_i} \in P} w_i$.

The terms $\mathbf{a_P}, b_P, w_P$ are independent of $\mathbf{q}$. With their pre-computed values,

$\mathcal{FL}_P(\mathbf{q}, Lin_{m,c})$ can be computed in $O(d)$ time. □

It remains to discuss how to find tight bound functions. For the upper bound function, we adopt the same technique in Figure 5.4. For the lower bound function, we use the idea in Figure 5.5b, except that the optimal value t_{opt} should also depend on the weighting.

THEOREM 5.2 *Consider the function $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ as a function of t, where $m_l = -\exp(-t)$ and $c_l = (1+t)\exp(-t)$. This function yields the maximum value at:*

$$t_{opt} = \frac{\gamma}{w_P} \cdot \sum_{\mathbf{p_i} \in P} w_i dist(\mathbf{q}, \mathbf{p_i})^2 \tag{5.10}$$

where $w_P = \sum_{\mathbf{p_i} \in P} w_i$.

Proof. Following the proof of Theorem 5.1, we let $H(t) = \mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l})$ be a function of t and we define the following two constants.

$$z_1 = \gamma \cdot \sum_{\mathbf{p_i} \in P} w_i dist(\mathbf{q}, \mathbf{p_i})^2 \text{ and } z_2 = w_P$$

Then, we follow exactly the same steps of Theorem 5.1 to derive the maximum value (Equation 5.10). □

Again, the value t_{opt} can also be computed efficiently (i.e., in $O(d)$ time).

5.3.1.2 Type III Weighting

For Type III weighting, there is no restriction on w_i. Each w_i takes either a positive value or a negative value.

Our idea is to convert the problem into two subproblems that use Type II weighting. First, we partition the point set P into two sets P^+ and P^- such that: (i) all points in P^+ are associated with positive weights, and (ii) all points in P^- are associated with negative weights. Then we introduce the following notation:

$$\underline{\mathcal{F}_{P^-}}(\mathbf{q}) = \sum_{\mathbf{p_i} \in P^-} |w_i| \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) = -\mathcal{F}_{P^-}(\mathbf{q})$$

This enables us to rewrite the function $\mathcal{F}_P(\mathbf{q})$ as:

$$\begin{aligned}
\mathcal{F}_P(\mathbf{q}) &= \sum_{\mathbf{p_i} \in P^+ \cup P^-} w_i \exp(-\gamma \cdot dist(\mathbf{q}, \mathbf{p_i})^2) \\
&= \mathcal{F}_{P^+}(\mathbf{q}) + \mathcal{F}_{P^-}(\mathbf{q}) \\
&= \mathcal{F}_{P^+}(\mathbf{q}) - \underline{\mathcal{F}_{P^-}}(\mathbf{q})
\end{aligned}$$

Since the weights in both $\mathcal{F}_{P^+}(\mathbf{q})$ and $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$ are positive, the terms $\mathcal{F}_{P^+}(\mathbf{q})$ and $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$ can be bounded by using the techniques for Type II weighting.

The upper bound of $\mathcal{F}_P(\mathbf{q})$ can be computed as the upper bound of $\mathcal{F}_{P^+}(\mathbf{q})$ minus the lower bound of $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$.

The lower bound of $\mathcal{F}_P(\mathbf{q})$ can be computed as the lower bound of $\mathcal{F}_{P^+}(\mathbf{q})$ minus the upper bound of $\underline{\mathcal{F}_{P^-}}(\mathbf{q})$.

5.3.2 Other Kernel Functions

In this section, we develop our bounding techniques for other kernel functions, such as the polynomial kernel function, and the sigmoid kernel function.

First, we consider the polynomial kernel function $\mathcal{K}(\mathbf{q}, \mathbf{p_i}) = (\gamma\ \mathbf{q}\cdot\mathbf{p_i} + \beta)^{deg}$, where γ, β are scalar values, and deg denotes the polynomial degree. In this context, we express the function $\mathcal{F}_P(\mathbf{q})$ as follows.

$$\mathcal{F}_P(\mathbf{q}) = \sum_{\mathbf{p_i}\in P} w_i(\gamma\ \mathbf{q}\cdot\mathbf{p_i} + \beta)^{deg} \tag{5.11}$$

For the sake of discussion, we assume Type II weighting (i.e., positive weight coefficients w_i).

We introduce the notation x_i to represent the term $\gamma\ \mathbf{q}\cdot\mathbf{p_i} + \beta$. The bounding interval of x_i, i.e., the interval $[x_{min}, x_{max}]$, is computed as:

$$\begin{aligned} x_{min} &= \gamma\ \mathbb{IP}_{min}(\mathbf{q}, R) + \beta \\ x_{max} &= \gamma\ \mathbb{IP}_{max}(\mathbf{q}, R) + \beta \end{aligned}$$

where R is the bounding rectangle of P, and $\mathbb{IP}_{min}(\mathbf{q}, R), \mathbb{IP}_{max}(\mathbf{q}, R)$ represent the minimum and the maximum inner product between $\mathbf{q}$ and R, respectively.

We then extend the efficient computation techniques in Section 5.2.1. Suppose that we are given two linear functions $Lin_{m_l,c_l}(x) = m_l x + c_l$ and $Lin_{m_u,c_u}(x) = m_u x + c_u$ such that $Lin_{m_l,c_l}(x) \leq x^{deg} \leq Lin_{m_u,c_u}(x)$ for all x. Similar to Lemma 5.1, we can obtain the following property: $\mathcal{FL}_P(\mathbf{q}, Lin_{m_l,c_l}) \leq \mathcal{F}_P(\mathbf{q}) \leq \mathcal{FL}_P(\mathbf{q}, Lin_{m_u,c_u})$, where:

$$\begin{aligned} \mathcal{FL}_P(\mathbf{q}, Lin_{m,c}) &= \sum_{\mathbf{p_i}\in P} w_i(m(\gamma\mathbf{q}\cdot\mathbf{p_i} + \beta) + c) \\ &= m\gamma\mathbf{q}\cdot\hat{\mathbf{a_P}} + (m\beta + c)\hat{b_P} \end{aligned}$$

where $\hat{\mathbf{a_P}} = \sum_{\mathbf{p_i} \in P} w_i \mathbf{p_i}$ and $\hat{b_P} = \sum_{\mathbf{p_i} \in P} w_i$. Such function can be computed in $O(d)$ time, provided that the terms $\hat{\mathbf{a_P}}, \hat{b_P}$ have been precomputed.

We proceed to discuss how to devise the bounding linear functions for x^{deg}. When deg is even, the function x^{deg} satisfies the convex property, and thus the techniques in Section 5.2.2 remain applicable. However, when deg is odd, the techniques in Section 5.2.2 are not applicable. For example, we plot the function x^3 in Figure 5.8 and notice that the chord between x_{min} and x_{max} is no longer an upper bound function.

Our idea is to exploit the monotonic increasing property of the function x^{deg} (given that deg is odd). We illustrate how to construct the bounding linear functions in Figure 5.8. For the upper bound function, we start with the horizontal line $y = (x_{max})^{deg}$ and then rotate-down the line until its left-hand-side hits the function x^{deg}. For the lower bound function, we start with the horizontal line $y = (x_{min})^{deg}$ and then rotate-up the line until its right-hand-side hits the function x^{deg}. The parameters of these lines can be derived by mathematical techniques.

The above idea is also applicable to the sigmoid kernel because it is also a monotonic increasing function. The tightness of all these bound functions depends on x_{min} and x_{max}.

5.4 Experimental Evaluation

We introduce the experimental setting in Section 5.4.1. Next, we demonstrate the performance of different methods in Section 5.4.2. Then, we compare

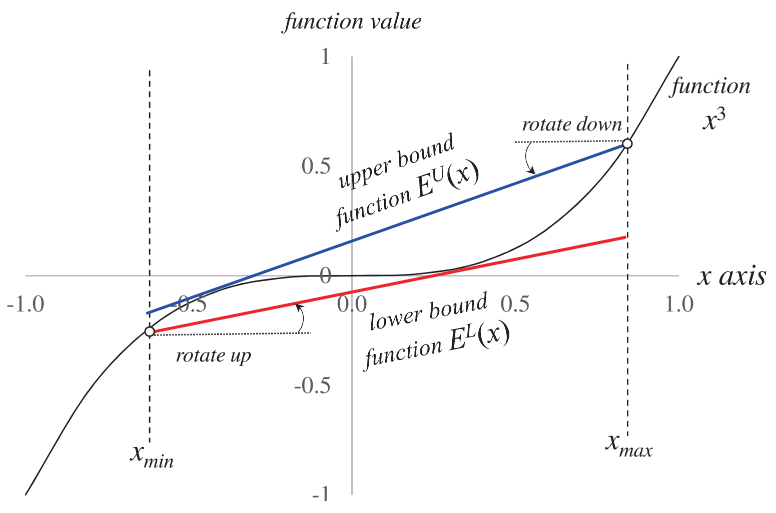


Figure 5.8. Lower and upper bound functions for x^3

the tightness of KARL and SOTA bound functions in Section 5.4.3. After that, we perform experimental analysis of our index-tuning method (KARL$_{\text{auto}}$) in Section 5.4.4. Next, we compare different methods for in-situ applications in Section 5.4.5. Lastly, we extend our techniques to combine with polynomial kernel function in Section 5.4.6.

5.4.1 Experimental Setting

5.4.1.1 Datasets

For Type-I, Type-II, Type-III weighting, we take the application model as kernel density estimation, 1-class SVM, and 2-class SVM, respectively. We use a wide variety of real datasets for these models, as shown in Table 5.6. The value n_{raw} denotes the number of points in the raw dataset, and the value d denotes the data dimensionality. The source/reference for each dataset is also provided.

These datasets either come from data repository websites [2, 24] or have been used in recent papers [46, 41].

For Type-I weighting, we follow [46] and use the Scott's rule to obtain the parameter γ. Type-II and Type III datasets require a training phase. We consider two kernel functions: the Gaussian kernel and the polynomial kernel. We denote the number of remaining points in the dataset after training as n_{model}^{gauss} and n_{model}^{poly}, for the Gaussian kernel and the polynomial kernel respectively.

The LIBSVM software [24] is used in the training phase. The training parameters are configured as follows. For each Type-II dataset, we apply 1-class SVM training, with the default kernel parameter $\gamma = \frac{1}{d}$ [24]. Then we vary the training model parameter ν from 0.01 to 0.3 and choose the model which provides the highest accuracy. For each Type-III dataset, we apply 2-class SVM training with the automatic script in [24] to determine the suitable values for training parameters.

Table 5.6. Details of datasets

Model	Raw dataset	n_{raw}	n_{model}^{gauss}	n_{model}^{poly}	d
Type I: kernel density	mnist [24]	60000	n/a	n/a	784
	miniboone [2]	119596	n/a	n/a	50
	home [2]	918991	n/a	n/a	10
	susy [2]	4990000	n/a	n/a	18
Type II: 1-class SVM	nsl-kdd [1]	67343	17510	6738	41
	kdd99 [2]	972780	19461	19462	41
	covtype [24]	581012	25486	14165	54
Type III: 2-class SVM	ijcnn1 [24]	49990	9592	9706	22
	a9a [24]	32561	11772	15682	123
	covtype-b [24]	581012	310184	323523	54

5.4.1.2 Methods for Comparisons

SCAN is the sequential scan method which computes $\mathcal{F}_P(\mathbf{q})$ without any pruning. Scikit-learn (abbrev. Scikit) is the machine learning library which supports the approximate KDE problem [50] (i.e., query type I-ϵ) and handles SVM-based classification (by LIBSVM [24]), i.e., query type I-τ, types II-τ and III-τ. SOTA is the state-of-the-art method which was developed by [46] for handling the Kernel Density Classification problem, i.e., I-τ query. We modify and extend their framework to handle other types of queries. Our KARL follows the framework of [46], combining with our linear bound functions, LB_{KARL} and UB_{KARL}. Both SOTA and KARL can work seamlessly with various index-structures. The space complexity of all these methods are $O(|P|d)$. Even for the largest tested dataset (susy), the memory consumption is only at most 1.34GBytes.

For the indexing options, kd-tree [105] and ball-tree [122, 87] are currently supported by Scikit. We only report the best result of Scikit (denoted as $\text{Scikit}_{\text{best}}$). For consistency, we also evaluate SOTA and KARL with these two indices. KARL can automatically choose suitable index and leaf capacity among these two indices, which we called $\text{KARL}_{\text{auto}}$. To demonstrate our effectiveness compared with SOTA, we select the best index with the best leaf capacity during the comparison in later sections, which we denote it by $\text{SOTA}_{\text{best}}$. For in-situ application, we combine the online-tuning method with KARL which we called $\text{KARL}_{\text{auto}}^{\text{online}}$. We also compare this method with the best performance of SOTA for this scenario, which we term it as $\text{SOTA}_{\text{best}}^{\text{online}}$.

We implemented all algorithms in C++ and conducted experiments on an Intel i7 3.4GHz PC running Ubuntu. For each dataset, we randomly sample

10,000 points from the dataset as the query set Q. Following [46], we measure the efficiency of a method as the throughput (i.e., the number of queries processed per second).

5.4.2 Efficiency Evaluation for Different Query Types

We test the performance of different methods for four types of queries which are I-ϵ, I-τ, II-τ and III-τ. The parameters of these queries are set as follows.

Type I-ϵ. We set the relative error $\epsilon = 0.2$ for each dataset.

Type I-τ. We fix the mean value μ of $\mathcal{F}_P(\mathbf{q})$ from the set Q, i.e., $\mu = \sum_{\mathbf{q}\in Q} \mathcal{F}_P(\mathbf{q})/|Q|$ as the threshold τ for each dataset in Table 5.7.

Types II-τ and III-τ. The threshold τ can be obtained during the training phase.

Table 5.7. All methods for different types of queries

Type	Datasets	SCAN	LIBSVM	$\text{Scikit}_{\text{best}}$	$\text{SOTA}_{\text{best}}$	$\text{KARL}_{\text{auto}}$
I-ϵ	miniboone	36.1	n/a	36	16.5	**301**
	home	15.2	n/a	11.9	36.2	**187**
	susy	2.02	n/a	1.17	0.77	**13.2**
I-τ	miniboone	36.1	34	n/a	102	**510**
	home	15.2	14.1	n/a	93.2	**258**
	susy	2.02	1.86	n/a	3.58	**83.4**
II-τ	nsl-kdd	283	481	n/a	748	**20668**
	kdd99	260	520	n/a	1269	**11324**
	covtype	158	462	n/a	448	**6022**
III-τ	ijcnn1	903	1170	n/a	1119	**826928**
	a9a	162	610	n/a	546	**6885**
	covtype-b	13	38.4	n/a	33.9	**274**

Table 5.7 shows the throughput of different methods for all types of queries.

In the result of query type I-ϵ, SCAN is comparable to Scikit$_{\text{best}}$ and SOTA$_{\text{best}}$ since the bounds computed by the basic bound functions are not tight enough. The performance of Scikit$_{\text{best}}$ and SOTA$_{\text{best}}$ is affected by the overhead of the loose bound computations. KARL$_{\text{auto}}$ uses our new bound functions which are shown to provide tighter bounds. These bounds lead to significant speedup in all evaluated datasets, e.g., KARL$_{\text{auto}}$ is at least 5.16 times faster than other methods.

For the type I-τ threshold-based queries, our method KARL$_{\text{auto}}$ improves the throughput by 2.76x to 21.2x when comparing to the runner-up method SOTA. The improvement becomes more obvious for type II-τ and type III-τ queries. The improvement of KARL$_{\text{auto}}$ can be up to 738x as compared to SOTA. KARL$_{\text{auto}}$ achieves significant performance gain for all these queries due to its tighter bound value compared with SOTA.

Sensitivity of τ. In order to test the sensitivity of threshold τ in different methods, we select seven thresholds from the range $\mu - 2\sigma$ to $\mu + 4\sigma$, where $\sigma = \sqrt{\sum_{\mathbf{q}\in Q}(\mathcal{F}_P(\mathbf{q}) - \mu)^2/|Q|}$ is the standard deviation. Figure 5.9 shows the results on three datasets. As a remark, for the miniboone dataset, we skip the thresholds $\mu-\sigma$ and $\mu-2\sigma$ as they are negative. Due to the superior performance of our bound functions, KARL$_{\text{auto}}$ outperforms SOTA$_{\text{best}}$ by nearly one order of magnitude in most of the datasets regardless of the chosen threshold.

Sensitivity of ϵ. In Scikit-learn library [94], we can select different relative error ϵ, which is called as tolerance in the approximate KDE. To test the sensitivity, we vary the relative error ϵ for different datasets with query type I-ϵ. If the value of ϵ is very small, the room for the bound estimations is very limited so that neither

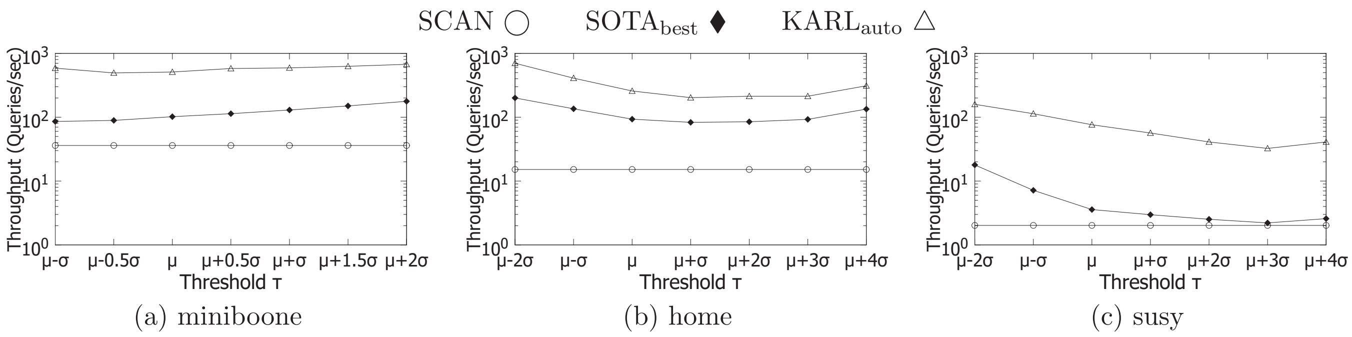


Figure 5.9. Query throughput with query type I-τ, varying the threshold τ

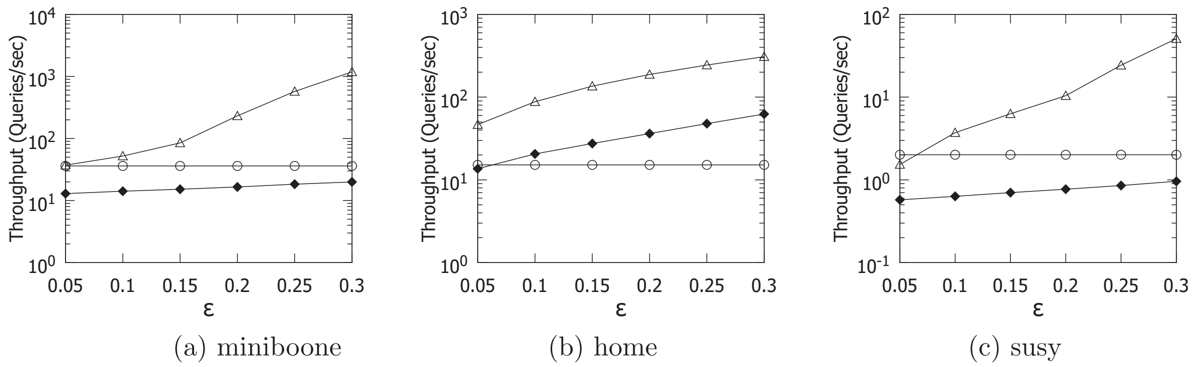
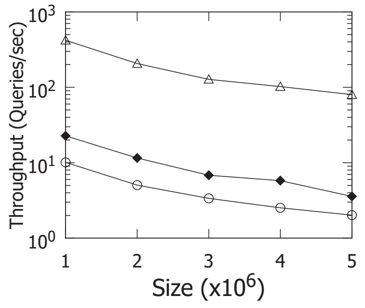


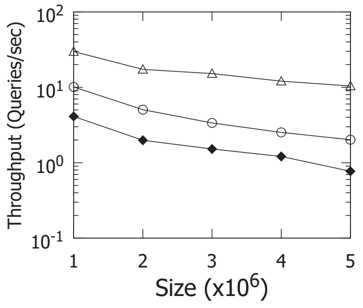
Figure 5.10. Query throughput with query type I-ϵ, varying the relative error ϵ

KARL$_{\text{auto}}$ nor SOTA$_{\text{best}}$ perform well in very small ϵ setting (e.g., 0.05). For other general ϵ settings, our method KARL$_{\text{auto}}$ consistently outperforms other methods by a visible margin (c.f. Figure 5.10).

Sensitivity of dataset size. In the following experiment, we test how the size of the dataset affects the evaluation performance of different methods for both query types I-ϵ and I-τ. We choose the largest dataset (susy) and vary the size via sampling. The trend in Figure 5.11 meets our expectation; a smaller size implies a higher throughput. Our KARL$_{\text{auto}}$ in general outperforms other methods by one order of magnitude in a wide range of data sizes.



(a) type I-τ, fixing $\tau = \mu$



(b) type I-ϵ, fixing $\epsilon = 0.2$

Figure 5.11. Query throughput on the susy dataset, varying the dataset size

Sensitivity of dimensionality. In this experiment, we choose the dataset (mnist) with the largest dimensionality (784) and then vary the dimensionality via PCA dimensionality reduction as in [46]. The default threshold $\tau = \mu$ is used. As shown in Figure 5.12, our method KARL$_{\text{auto}}$ consistently outperforms existing methods under different dimensionalities.

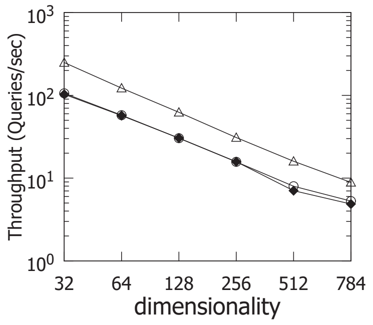


Figure 5.12. Query throughput with query type I-τ ($\tau = \mu$) on the mnist dataset, varying the dimensionality

5.4.3 Tightness of Bound Functions

Recall from Section 5.2, we have theoretically shown that our developed linear bound functions LB$_{\text{KARL}}$ and UB$_{\text{KARL}}$ are tighter than SOTA bound functions. In this section, we explore how tight our bound functions can be better than SOTA in practice.

For the sake of fairness, we fix the tree-structure to be the kd-tree with leaf capacity 80. We use the following equation to measure the average tightness of bound values.

$$Error_{bound} = \frac{1}{|Q| \cdot L} \sum_{l=1}^{L} \sum_{\mathbf{q} \in Q} \left| \frac{\sum_{R_j \in \mathbf{R}_l} bound(\mathbf{q}, R_j) - \mathcal{F}_P(\mathbf{q})}{\mathcal{F}_P(\mathbf{q})} \right|$$

where $\mathbf{R}_l$ denotes the set of entries (cf. Figure 5.2) in the l^{th} level of kd-tree (with L levels in total).

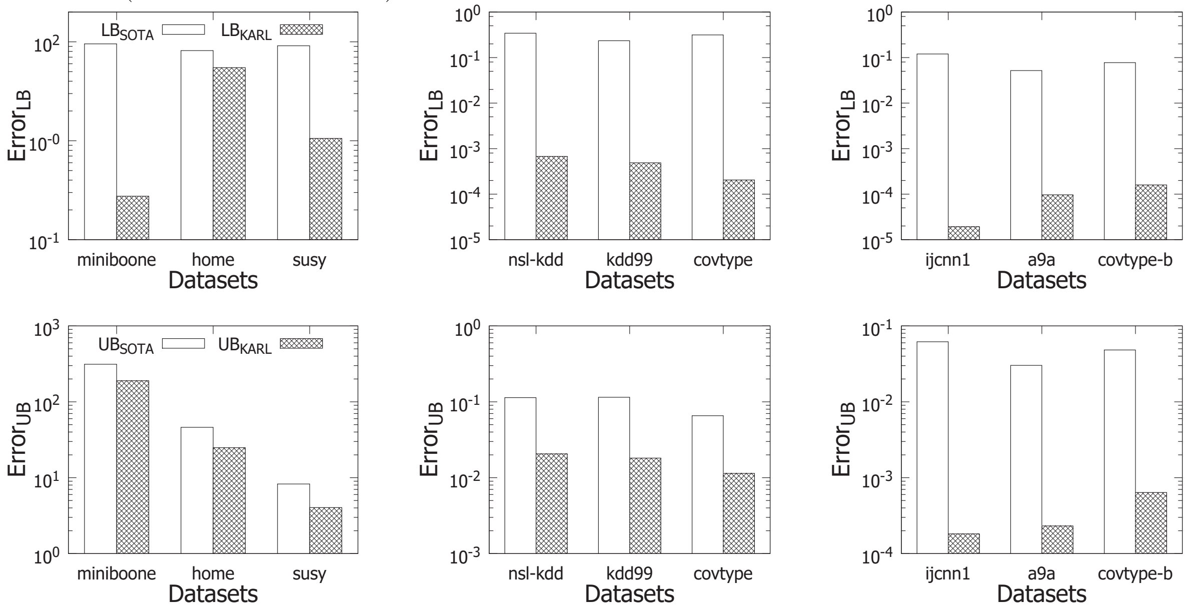


Figure 5.13. $Error_{LB}$ **and** $Error_{UB}$ **for Type-I, II and III queries (left,middle and right respectively)**

Our bound functions are in practice much tighter than LB_{SOTA} and UB_{SOTA} in all evaluated datasets, especially for LB_{KARL}, as shown in Figure 5.13. In addition, the bound functions (of SOTA and KARL) provide very tight bounds for query types II and III. There are two reasons for this phenomenon, including the data distribution and data normalization. First, the data points of types II and III are the support vectors (being trained by SVM), which are the nearest points to the decision boundary [107] and are very close to each other. Second, the set of support vectors are normalized to the domain $[0, 1]^d$ [24]. This limits the range of possible values of the exponential function so that the lower and upper bounds are close to the exact value.

5.4.4 Offline Index Tuning

Recall from Figure 5.2, different indexing structures can be applied to our problems. One natural question is how can we predict the suitable index-structure. In this section, we demonstrate our offline tuning method $\text{KARL}_{\text{auto}}$ (cf. Section 5.2.3), which automatically selects the best tree-structure with the suitable leaf capacity from kd-tree [105] and ball-tree [122, 87]. These two tree-structures are currently supported by Scikit-learn library [94].

Our solution $\text{KARL}_{\text{auto}}$ randomly samples 1000 vectors, denoted by the sample set S, from each dataset and predicts the performance based on the throughput, using different leaf capacities of tree. Table 5.8 shows that in the offline stage, our method $\text{KARL}_{\text{auto}}$ can provide good prediction which yields an online throughput near the best solution $\text{KARL}_{\text{best}}$.

5.4.5 Online Index Tuning for In-situ Applications

In some online learning scenarios [38, 83, 73], we may not be able to pre-build the index. Thereby, we cannot simply omit the index construction time. In this section, we consider our online-tuning solution $\text{KARL}_{\text{auto}}^{\text{online}}$ (cf. Section

Table 5.8. Query throughput for variants of KARL, using sample set with $|S| = 1000$

Type	Datasets	$\text{KARL}_{\text{worst}}$	$\text{KARL}_{\text{auto}}$	$\text{KARL}_{\text{best}}$
I-ϵ	miniboone	88.1	302	304
	home	35.9	185	188
	susy	5.5	12.9	13.3
I-τ	miniboone	64.8	514	566
	home	76.6	258	258
	susy	16.7	84	89
II-τ	nsl-kdd	4357	20668	20677
	kdd99	5911	11324	11325
	covtype	915	6022	6038
III-τ	ijcnn1	388109	826928	843601
	a9a	408	6885	6891
	covtype-b	52	274	277

5.2.3). All results are shown in Table 5.9.

For query types I-ϵ and I-τ, $\text{SOTA}_{\text{best}}^{\text{online}}$ is not efficient because its bounding functions are not tight (recall from Figure 5.6). Our method $\text{KARL}_{\text{auto}}^{\text{online}}$ outperforms existing methods significantly on all tested datasets.

For query types II-τ and III-τ, our $\text{KARL}_{\text{auto}}^{\text{online}}$ can significantly increase the throughput in several datasets since each support vector in a dataset is near to each other.

5.4.6 Efficiency for Polynomial Kernel

Recall from Section 5.3.2, our linear bound functions can be also used for polynomial kernel. In this section, we experimentally test the online query throughput with query types II-τ and III-τ. We use the polynomial kernel with degree 3 which is the default setting in LIBSVM [24]. We also normalize the

Table 5.9. In-situ solutions for different types of queries

Type	Datasets	baseline	$\text{SOTA}_{\text{best}}^{\text{online}}$	$\text{KARL}_{\text{auto}}^{\text{online}}$
I-ϵ	miniboone	36.1	16.4	**217**
	home	35.8	32.1	**184**
	susy	2.02	0.75	**7.26**
I-τ	miniboone	36.1	101	**419**
	home	15.2	92.1	**243**
	susy	2.02	3.57	**51**
II-τ	nsl-kdd	481	733	**9869**
	kdd99	260	1264	**7920**
	covtype	462	439	**2389**
III-τ	ijcnn1	1170	1112	**426132**
	a9a	610	543	**1966**
	covtype-b	38.4	33.5	**101**

datasets to the domain $[-1, 1]^d$ [24]. Then, we apply the same setting in the training phases in 1-class and 2-class SVM which are stated in Section 5.4.2. Table 5.10 shows that our method $\text{KARL}_{\text{auto}}$ outperforms $\text{SOTA}_{\text{best}}$ by 3x to 165x.

Table 5.10. Query throughput with query type II/III-τ using polynomial kernel

Type	Datasets	baseline	$\text{SOTA}_{\text{best}}$	$\text{KARL}_{\text{auto}}$
II-τ	nsl-kdd	909	1200	**4522**
	kdd99	314	639	**2741**
	covtype	537	6423	**88396**
III-τ	ijcnn1	1154	1122	**185372**
	a9a	463	422	**2813**
	covtype-b	36.4	30.5	**187**

5.5 Related Work

The literature review of the recent techniques of kernel functions have been summarized in Section 2.4.3. In this section, we cover the works directly related to our problem. Then, we mainly discuss the differences between our work [22] and previous works.

The term “kernel aggregation query” abstracts a common operation in several statistical and learning problems such as kernel density estimation [50, 46], 1-class SVM [86], and 2-class SVM [107].

Kernel density estimation is a non-parametric statistical method for density estimation. To speedup kernel density estimation, existing works would either compute approximate density values with accuracy guarantee [87] or test whether density values are above a given threshold [46]. Zheng et al. [136] focus on fast kernel density estimation on low-dimensional data (e.g., 1d, 2d) and propose sampling-based solutions with theoretical guarantees on both efficiency and quality. On the other hand, [87, 46] assume that the point set P is indexed by a k-d tree, and apply filter-and-refinement techniques for kernel density estimation. The library Scikit-learn [94] adopts the implementation in [87]. Our proposal, KARL, adapts the algorithm in [87, 46] to evaluate kernel aggregation queries. The key difference between KARL and [87, 46] lies in the bound functions. As explained in Section 5.2.2, our proposed linear bound functions are tighter than existing bound functions used in [87, 46]. Furthermore, we extend our linear bound functions to deal with different types of weighting and kernel functions, which have not been considered in [87, 46].

SVM is proposed by the machine learning community to classify data objects or detect outliers. SVM has been applied in different application domains, such as document classification [86], network fault detection [134, 17, 15], anomaly/outlier detection [23, 80], novelty detection [58, 84, 108], tumor samples classification [32], image classification [29], time series classification [69]. The typical process is divided into two phases. In the offline phase, training/learning algorithms are used to obtain the point set P, the weighting, and parameters.

Then, in the online phase, threshold kernel aggregation queries can be used to support classification or outlier detection. Two approaches have been studied to accelerate the online phase. The library LibSVM [24] assumes sparse data format and applies inverted index to speedup exact computation. The machine learning community has proposed heuristics [68, 57, 78, 81] to reduce the size of the point set P in the offline phase, in order to speedup the online phase. However, these heuristics may affect the prediction quality of SVM. Our proposed bound functions have not been studied in the above work.

5.6 Chapter Summary

In this chapter, we study kernel aggregation queries, which can be used to support a common operation in kernel density estimation [50, 46], 1-class SVM [86], and 2-class SVM [107].

Our key contribution is the development of fast linear bound functions, which are proven to be tighter than existing bound functions, yet allowing fast computation. In addition, we propose a comprehensive solution that can support different types of kernel functions and weighting schemes. Our automatic tuning methods support identification of efficient index structure, which depends on the underlying point set P.

Experimental studies on a wide variety of datasets show that our solution yields higher throughput than the state-of-the-art by 2.5–738 times.

A promising future research direction is to consider more statistical/learning tasks based on kernel functions, e.g., kernel regression and multi-class kernel SVM.

Chapter 6

Conclusions and Suggestions for Future Research

Similarity measures are important for many computer vision and machine learning tasks. In this paper, we propose both exact or approximation algorithms combining with the developed lower and upper bound functions to boost up the efficiency performance in three fundamental problems.

6.1 Contributions

The first one is for the template matching problem (SWNNS) in Chapter 3 which are deemed to be inefficient in previous work. Our developed solution boosts up the performance for nearly 9-20x faster which can enable different applications in computer vision, for example: object detection or motion estimation.

The second one is for the image retrieval problem in Chapter 4. Earth mover's distance is a robust similarity measure for this application, but it is slow. Our developed approximation methods can not only retain the accurate result compared with the exact one, but also they can provide an order of magnitude speedup over the fastest exact computation algorithm.

The third one is for the kernel aggregation query in Chapter 5. This type of query is computed in kernel based machine learning/ statistics models such as: one-class SVM, two-class SVM or kernel density estimation/ classification. Directly computing this query is not efficient. Our developed solution can boost up the efficiency performance by 2.5-738x faster in various datasets, models and kernel types.

6.2 Future Directions

In Chapter 3, our problem formulation is only limited to the rectangular/ irregular shaped query. However, existing object detection algorithms also need to handle some objects which contain a wide range of deformation, such as: rotation, affline transformation. Our developed techniques may not be straightforward to apply in these scenarios. As such, our next goal is to develop algorithms to boost up the efficiency performance for these tasks.

In Chapter 4, we develop an approximate framework to efficiently boost up the efficiency performance of evaluating EMD function. However, this framework is general which can also support other similarity measures. One research direction is to apply this framework to other similarity measures.

In Chapter 5, our solution only covers some popular machine learning models, for example: svm or kernel density estimation, in the online phase. However, it is interesting to know whether our techniques can be applied to other models, such as: kernel regression model. Another interesting direction is how to extend our techniques to boost up the training phase of these models. Recently, many works also focus on deep learning based models for different machine learning tasks with successful results. Our next goal is to exploit the property in this categories of model and develop efficient algorithms.

References

[1] Nsl-kdd dataset. `https://github.com/defcom17/NSL_KDD`.

[2] UCI machine learning repository. `http://archive.ics.uci.edu/ml/index.php`.

[3] Weather datasets. `http://weather.is.kochi-u.ac.jp/sat/GAME/`.

[4] Comparison of density estimation methods for astronomical datasets. *Astronomy and Astrophysics*, 531, 7 2011.

[5] Pankaj K. Agarwal and R. Sharathkumar. Approximation algorithms for bipartite matching with metric and geometric costs. In *STOC*, pages 555–564, 2014.

[6] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows - theory, algorithms and applications*. Prentice Hall, 1993.

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

[8] Alexandr Andoni, Piotr Indyk, and Robert Krauthgamer. Earth mover distance over high-dimensional spaces. In *SODA*, pages 343–352, 2008.

[9] Ira Assent, Andrea Wenning, and Thomas Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *ICDE*, page 11, 2006.

[10] Ira Assent, Marc Wichterich, Tobias Meisen, and Thomas Seidl. Efficient similarity search using the earth mover's distance for large multimedia databases. In *ICDE*, pages 307–316, 2008.

[11] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[12] Gil Ben-Artzi, Hagit Hel-Or, and Yacov Hel-Or. The gray-code filter kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):382–393, 2007.

[13] Michal Ben-Yehuda, Lihi Cadany, and Hagit Hel-Or. Irregular pattern matching using projections. In *ICIP*, pages 834–837, 2005.

[14] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[15] Monowar H. Bhuyan, D. K. Bhattacharyya, and Jugal K. Kalita. Network anomaly detection: Methods, systems and tools. *IEEE Communications Surveys and Tutorials*, 16(1):303–336, 2014.

[16] Remus Brad and Ioan Alfred Letia. Extracting cloud motion from satellite image sequences. In *ICARCV*, pages 1303–1307, 2002.

[17] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys and Tutorials*, 18(2):1153–1176, 2016.

[18] Yunliang Cai and George Baciu. Detecting, grouping, and structure inference for invariant repetitive patterns in images. *IEEE Trans. Image Processing*, 22(6):2343–2355, 2013.

[19] Tsz Nam Chan, Man Lung Yiu, and Kien A. Hua. A progressive approach for similarity search on matrix. In *SSTD*, pages 373–390, 2015.

[20] Tsz Nam Chan, Man Lung Yiu, and Kien A. Hua. Efficient sub-window nearest neighbor search on matrix. *IEEE Trans. Knowl. Data Eng.*, 29(4):784–797, 2017.

[21] Tsz Nam Chan, Man Lung Yiu, and Leong Hou U. *The Power of Bounds: Answering Approximate Earth Mover's Distance with Parametric Bounds.* submitted to TKDE.

[22] Tsz Nam Chan, Man Lung Yiu, and Leong Hou U. *KARL: Fast Kernel Aggregation Queries.* ICDE, 2019.

[23] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.

[24] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[25] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.

[26] Ching-Chien Chen, Craig A. Knoblock, and Cyrus Shahabi. Automatically conflating road vector data with orthoimagery. *GeoInformatica*, 10(4):495–530, 2006.

[27] Ching-Chien Chen, Cyrus Shahabi, and Craig A. Knoblock. Utilizing road network data for automatic identification of road intersections from high resolution color orthoimagery. In *STDBM 04*, pages 17–24, 2004.

[28] Lu Chen, Yunjun Gao, Baihua Zheng, Christian S. Jensen, Hanyu Yang, and Keyu Yang. Pivot-based metric indexing. *PVLDB*, 10(10):1058–1069, 2017.

[29] Qiang Chen, Zheng Song, Jian Dong, ZhongYang Huang, Yang Hua, and Shuicheng Yan. Contextualizing object detection and classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(1):13–27, 2015.

[30] Ming-Ming Cheng, Fang-Lue Zhang, Niloy J. Mitra, Xiaolei Huang, and Shi-Min Hu. Repfinder: finding approximately repeated scene elements for image editing. *ACM Trans. Graph.*, 29(4):83:1–83:8, 2010.

[31] Yao-Yi Chiang, Craig A. Knoblock, Cyrus Shahabi, and Ching-Chien Chen. Automatic and accurate extraction of road intersections from raster maps. *GeoInformatica*, 13(2):121–157, 2009.

[32] Hua-Sheng Chiu and et al. Pan-Cancer Analysis of lncRNA Regulation Supports Their Targeting of Cancer Genes in Each Tumor Context. *Cell Reports*, 23(1):297–312, April 2018.

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

[34] Scott D. Cohen and Leonidas J. Guibas. The earth mover's distance: Lower bounds and invariance under translation technical report. 1997.

[35] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*, pages 191–198, 2016.

[36] Kyle Cranmer. Kernel estimation in high-energy physics. 136:198–207, 2001.

[37] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.

[38] Manuel Davy, Frédéric Desobry, Arthur Gretton, and Christian Doncarli. An online support vector machine for abnormal events detection. *Signal Processing*, 86(8):2009–2025, 2006.

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

[40] Thomas Deselaers, Daniel Keysers, and Hermann Ney. Features for image retrieval: an experimental comparison. *Inf. Retr.*, 11(2):77–107, 2008.

[41] Remi Domingues, Maurizio Filippone, Pietro Michiardi, and Jihane Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, 2018.

[42] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. `http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html`.

[43] Mark Everingham, S. M. Ali Eslami, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[44] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, 1994.

[45] Ada Wai-Chee Fu, Eamonn J. Keogh, Leo Yung Hang Lau, Chotirat (Ann) Ratanamahatana, and Raymond Chi-Wing Wong. Scaling and time warping in time series querying. *VLDB J.*, 17(4):899–921, 2008.

[46] Edward Gan and Peter Bailis. Scalable kernel density classification via threshold-based pruning. In *SIGMOD*, pages 945–959, 2017.

[47] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.

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

[49] Mohammad Gharavi-Alkhansari. A fast globally optimal algorithm for template matching using low-resolution pruning. *IEEE Trans. Image Processing*, 10(4):526–533, 2001.

[50] Alexander G. Gray and Andrew W. Moore. Nonparametric density estimation: Toward computational tractability. In *SDM*, pages 203–211, 2003.

[51] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. technical report 7694, caltech. 2007.

[52] O. Güler. *Foundations of Optimization*. Graduate Texts in Mathematics. Springer New York, 2010.

[53] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[54] James L. Hafner, Harpreet S. Sawhney, William Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(7):729–736, 1995.

[55] Yacov Hel-Or and Hagit Hel-Or. Real-time pattern matching using projection kernels. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(9):1430–1445, 2005.

[56] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in OLAP data cubes. In *SIGMOD*, pages 73–88, 1997.

[57] Cho-Jui Hsieh, Si Si, and Inderjit S. Dhillon. Fast prediction for large-scale kernel machines. In *NIPS*, pages 3689–3697, 2014.

[58] Chengqiang Huang, Geyong Min, Yulei Wu, Yiming Ying, Ke Pei, and Zuochang Xiang. Time series anomaly detection for trustworthy services in cloud computing systems. *IEEE Trans. Big Data*, to appear.

[59] Jin Huang, Rui Zhang, Rajkumar Buyya, Jian Chen, and Yongwei Wu. Heads-join: Efficient earth mover's distance similarity joins on hadoop. *IEEE Trans. Parallel Distrib. Syst.*, 27(6):1660–1673, 2016.

[60] Mark J. Huiskes and Michael S. Lew. The MIR flickr retrieval evaluation. In *SIGMM*, pages 39–43, 2008.

[61] Piotr Indyk. A near linear time constant factor approximation for euclidean bichromatic matching (cost). In *SODA*, pages 39–42, 2007.

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

[63] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive b$^+$-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005.

[64] Min-Hee Jang, Sang-Wook Kim, Christos Faloutsos, and Sunju Park. A linear-time approximation of the earth mover's distance. In *CIKM*, pages 505–514, 2011.

[65] Min-Hee Jang, Sang-Wook Kim, Christos Faloutsos, and Sunju Park. Accurate approximation of the earth mover's distance in linear time. *J. Comput. Sci. Technol.*, 29(1):142–154, 2014.

[66] Feng Jing, Mingjing Li, HongJiang Zhang, and Bo Zhang. An efficient and effective region-based image retrieval framework. *IEEE Trans. Image Processing*, 13(5):699–709, 2004.

[67] Feng Jing, Mingjing Li, HongJiang Zhang, and Bo Zhang. Relevance feedback in region-based image retrieval. *IEEE Trans. Circuits Syst. Video Techn.*, 14(5):672–681, 2004.

[68] Ho Gi Jung and Gahyun Kim. Support vector number reduction: Survey and experimental evaluations. *IEEE Trans. Intelligent Transportation Systems*, 15(2):463–476, 2014.

[69] Argyro Kampouraki, George Manis, and Christophoros Nikou. Heartbeat time series classification with support vector machines. *IEEE Trans. Information Technology in Biomedicine*, 13(4):512–518, 2009.

[70] Michael Kapralov and Rina Panigrahy. NNS lower bounds via metric expansion for l ∞ and EMD. In *ICALP*, pages 545–556, 2012.

[71] Daniel A. Keim and Benjamin Bustos. Similarity search in multimedia databases. In *ICDE*, page 873, 2004.

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

[73] Jyrki Kivinen, Alexander J. Smola, and Robert C. Williamson. Online learning with kernels. In *NIPS*, pages 785–792, 2001.

[74] Yan Kong, Weiming Dong, Xing Mei, Xiaopeng Zhang, and Jean-Claude Paul. Simlocator: robust locator of similar objects in images. *The Visual Computer*, 29(9):861–870, 2013.

[75] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[76] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot L. Siegel, and Zenon Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB*, pages 215–226, 1996.

[77] Hans-Peter Kriegel, Peer Kröger, Peter Kunath, and Matthias Renz. Generalizing the optimality of multi-step k -nearest neighbor query processing. In *SSTD*, pages 75–92, 2007.

[78] Quoc V. Le, Tamás Sarlós, and Alexander J. Smola. Fastfood - computing hilbert space expansions in loglinear time. In *ICML*, pages 244–252, 2013.

[79] Hui Li, Tsz Nam Chan, Man Lung Yiu, and Nikos Mamoulis. FEXIPRO: fast and exact inner product retrieval in recommender systems. In *SIGMOD*, pages 835–850, 2017.

[80] Bo Liu, Yanshan Xiao, Philip S. Yu, Longbing Cao, Yun Zhang, and Zhifeng Hao. Uncertain one-class learning and concept summarization learning on uncertain data streams. *IEEE Trans. Knowl. Data Eng.*, 26(2):468–484, 2014.

[81] Yi-Hung Liu, Yan-Chen Liu, and Yen-Jen Chen. Fast support vector data descriptions for novelty detection. *IEEE Trans. Neural Networks*, 21(8):1296–1313, 2010.

[82] Vebjorn Ljosa, Arnab Bhattacharya, and Ambuj K. Singh. Indexing spatially sensitive distance measures using multi-resolution lower bounds. In *EDBT*, pages 865–883, 2006.

[83] Jing Lu, Steven C. H. Hoi, Jialei Wang, Peilin Zhao, and Zhiyong Liu. Large scale online kernel learning. *Journal of Machine Learning Research*, 17:47:1–47:43, 2016.

[84] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *IJCNN*, pages 1741–1745, 2003.

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

[86] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *Journal of Machine Learning Research*, 2:139–154, 2001.

[87] Andrew W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, pages 397–405, 2000.

[88] Yair Moshe and Hagit Hel-Or. Video block motion estimation based on gray-code kernels. *IEEE Trans. Image Processing*, 18(10):2243–2254, 2009.

[89] James B. Orlin. A faster strongly polynominal minimum cost flow algorithm. In *STOC*, pages 377–387, 1988.

[90] Wanli Ouyang and Wai-kuen Cham. Fast algorithm for walsh hadamard transform on sliding windows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):165–171, 2010.

[91] Wanli Ouyang, Federico Tombari, Stefano Mattoccia, Luigi di Stefano, and Wai-kuen Cham. Performance evaluation of full search equivalent pattern matching algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(1):127–143, 2012.

[92] Wanli Ouyang, Renqi Zhang, and Wai-kuen Cham. Segmented gray-code kernels for fast pattern matching. *IEEE Trans. Image Processing*, 22(4):1512–1525, 2013.

[93] Wanli Ouyang, Tianle Zhao, Wai-kuen Cham, and Liying Wei. Fast full-search-equivalent pattern matching using asymmetric haar wavelet packets. *IEEE Trans. Circuits Syst. Video Techn.*, 28(4):819–833, 2018.

[94] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

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

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

[97] Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *ICCV*, pages 460–467, 2009.

[98] Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, pages 262–270, 2012.

[99] Parikshit Ram and Alexander G. Gray. Maximum inner-product search using cone trees. In *SIGKDD*, pages 931–939, 2012.

[100] Yossi Rubner, Jan Puzicha, Carlo Tomasi, and Joachim M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84(1):25–43, 2001.

[101] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[102] Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.

[103] Brian E. Ruttenberg and Ambuj K. Singh. Indexing the earth mover's distance using normal distributions. *PVLDB*, 5(3):205–216, 2011.

[104] Yasushi Sakurai, Masatoshi Yoshikawa, and Christos Faloutsos. FTW: fast similarity search under the time warping distance. In *PODS*, pages 326–337, 2005.

[105] H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann. 2006.

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

[107] Bernhard Schölkopf and Alexander Johannes Smola. *Learning with Kernels: support vector machines, regularization, optimization, and beyond.* Adaptive computation and machine learning series. MIT Press, 2002.

[108] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, and John C. Platt. Support vector method for novelty detection. In *NIPS*, pages 582–588, 1999.

[109] Haim Schweitzer, Rui A. Deng, and Robert Finis Anderson. A dual-bound algorithm for very fast and exact template matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(3):459–470, 2011.

[110] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD*, pages 154–165, 1998.

[111] Timos K. Sellis, Nick Roussopoulos, and Christos Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. In *VLDB*, pages 507–518, 1987.

[112] Sameer Shirdhonkar and David W. Jacobs. Approximate earth mover's distance in linear time. In *CVPR*, 2008.

[113] Anshumali Shrivastava and Ping Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *NIPS*, pages 2321–2329, 2014.

[114] Diego Furtado Silva and Gustavo E. A. P. A. Batista. Speeding up all-pairwise dynamic time warping matrix calculation. In *SDM*, pages 837–845, 2016.

[115] Arnold W. M. Smeulders, Marcel Worring, Simone Santini, Amarnath Gupta, and Ramesh C. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(12):1349–1380, 2000.

[116] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[117] Yu Tang, Leong Hou U, Yilun Cai, Nikos Mamoulis, and Reynold Cheng. Earth mover's distance based similarity search at scale. *PVLDB*, 7(4):313–324, 2013.

[118] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[119] Christina Teflioudi, Rainer Gemulla, and Olga Mykytiuk. LEMP: fast retrieval of large entries in a matrix product. In *SIGMOD*, pages 107–122, 2015.

[120] Federico Tombari, Stefano Mattoccia, and Luigi di Stefano. Full-search-equivalent pattern matching with incremental dissimilarity approximations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(1):129–141, 2009.

[121] Dai-Duong Truong, Vinh-Tiep Nguyen, Anh Duc Duong, Chau-Sang Nguyen Ngoc, and Minh-Triet Tran. Realtime arbitrary-shaped template matching process. In *ICARCV*, pages 1407–1412, 2012.

[122] Jeffrey K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

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

[124] M.P. Wand and M.C. Jones. *Kernel Smoothing*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[125] James Ze Wang, Jia Li, and Gio Wiederhold. Simplicity: Semantics-sensitive integrated matching for picture libraries. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(9):947–963, 2001.

[126] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[127] Shou-Der Wei and Shang-Hong Lai. Fast template matching based on normalized cross correlation with adaptive multilevel winner update. *IEEE Trans. Image Processing*, 17(11):2227–2235, 2008.

[128] Marc Wichterich, Ira Assent, Philipp Kranen, and Thomas Seidl. Efficient emd-based similarity search in multimedia databases via flexible dimensionality reduction. In *SIGMOD*, pages 199–212, 2008.

[129] Jia Xu, Bin Lei, Yu Gu, Marianne Winslett, Ge Yu, and Zhenjie Zhang. Efficient similarity join based on earth mover's distance using mapreduce. *IEEE Trans. Knowl. Data Eng.*, 27(8):2148–2162, 2015.

[130] Jia Xu, Zhenjie Zhang, Anthony K. H. Tung, and Ge Yu. Efficient and effective similarity search over probabilistic data based on earth mover's distance. *PVLDB*, 3(1):758–769, 2010.

[131] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *VLDB*, pages 385–394, 2000.

[132] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Advances in Database Systems. Springer US, 2006.

[133] Jianguo Zhang, Marcin Marszalek, Svetlana Lazebnik, and Cordelia Schmid. Local features and kernels for classification of texture and object categories: A comprehensive study. *International Journal of Computer Vision*, 73(2):213–238, 2007.

[134] Liangwei Zhang, Jing Lin, and Ramin Karim. Adaptive kernel density-based anomaly detection for nonlinear systems. *Knowledge-Based Systems*, 139(Supplement C):50 – 63, 2018.

[135] Qi Zhao, Zhi Yang, and Hai Tao. Differential earth mover's distance with its applications to visual tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(2):274–287, 2010.

[136] Yan Zheng, Jeffrey Jestes, Jeff M. Phillips, and Feifei Li. Quality and efficiency for kernel density estimates in large data. In *SIGMOD*, pages 433–444, 2013.