Noname manuscript No.
(will be inserted by the editor)

Efficient and Effective Similarity Search over Probabilistic Data Based on Earth Mover's Distance

Jia Xu · Zhenjie Zhang · Anthony K.H. Tung · Ge Yu

Received: date / Accepted: date

Abstract Advances in geographical tracking, multimedia processing, information extraction, and sensor networks have created a deluge of probabilistic data. While similarity search is an important tool to support the manipulation of probabilistic data, it raises new challenges to traditional relational databases. The problem stems from the limited effectiveness of the distance metrics employed by existing database systems. On the other hand, several more complicated distance operators have proven their values for better distinguishing ability in specific probabilistic domains. In this paper, we discuss the similarity search problem with respect to *Earth Mover's Distance* (EMD). EMD is the most successful distance metric for probability distribution comparison but is an expensive operator as it has cubic time complexity. We present a new database indexing approach to answer EMD-based similarity queries, including range queries and k-nearest neighbor queries on probabilistic data. Our solution utilizes *Primal-Dual Theory* from linear programming and employs a group of B^+ trees for effective candidate pruning. We also apply our filtering technique to the processing of continuous similarity queries, especially with applications to frame copy detection in real-time videos. Extensive experiments show that our proposals dramatically improve the usefulness and scalability of probabilistic data management.

Keywords Probabilistic data management · Similarity search · Earth mover's distance · Tree-based indexing

Zhenjie Zhang and Anthony K. H. Tung were supported by Singapore NRF grant R-252-000-376-279. This work is also supported by the National Natural Science Foundation of China (No. 60933001 and No. 61003058), the Fundamental Research Funds for the Central Universities (No. N090104001) and the National Basic Research Program of China (973 Program) under grant 2012CB316201.

J. Xu(✉)
College of Info. Sci. & Eng., Northeastern University, Shenyang, China
E-mail: xujia@ise.neu.edu.cn

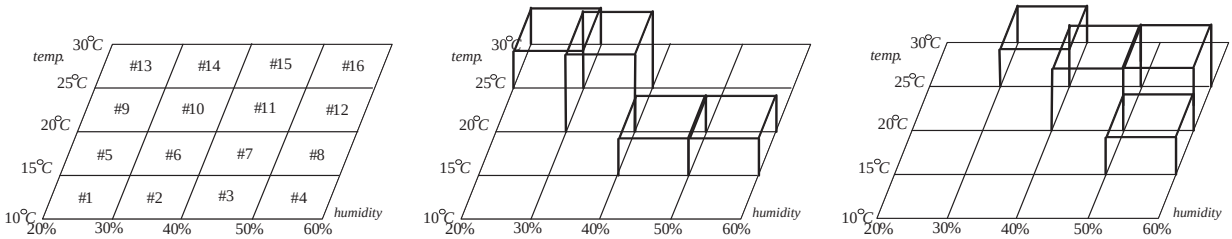
Z. J. Zhang
Advanced Digital Sciences Center, Illinois at Singapore Pte. Ltd, Singapore
E-mail: zhenjie@adsc.com.sg

A. K. H. Tung
School of Computing, National University of Singapore, Singapore
E-mail: atung@comp.nus.edu.sg

G. Yu
College of Info. Sci. & Eng., Northeastern University, Shenyang, China
E-mail: yuge@ise.neu.edu.cn

1 Introduction

Advances in geographical tracking [36], multimedia processing [20,30], information extraction [37], and sensor networks [18] have created a deluge of probabilistic data. This trend has led to extensive research efforts devoted to scalable database systems for probabilistic data management [4,8,11,14,15,22,36]. To fully utilize the information underlying these data distributions, a variety of probabilistic queries have been proposed and studied in different contexts, such as *Accumulated Probability Query* [3,35] and *Top-k Query* [13, 21,23,27,34]. Most of existing studies on these queries however, simply extend the traditional database queries on simple distance metrics, e.g., Euclidean distance, by assuming probabilistic attributes instead of exact ones. Unfortunately, such extensions do not necessarily ensure the usefulness of these probabilistic databases,



(a) Domain of temperature & humidity (b) Distribution of sensor s_1 (c) Distribution of sensor s_2

Fig. 1 Examples of probabilistic records in the form of histograms

Table 1 Example probabilistic records in Figure 1 stored in a relational table

Cells	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16
p_{s_1}	0	0	0	0	0	0	0.2	0.2	0	0.4	0	0	0.2	0	0	0
p_{s_2}	0	0	0	0	0	0	0	0.2	0	0	0.3	0.3	0	0.2	0	0

since the simple distance metrics usually fail to capture the true similarities between the distributions underlying the objects. On the other hand, research results in other areas, such as computer vision, have indicated that some complex distance operators, such as the *Quadratic Form Distance* and the *Earth Mover's Distance*, are more significant in returning meaningful results under the context of search and retrieval on probabilistic data [28]. In this paper, we offer a database solution to better serve physical-world applications that manage probabilistic data. In particular, we discuss the problem of similarity search based on the *Earth Mover's Distance* (EMD) to query probabilistic data represented as histograms[1].

Since its development in the late 1990s [29], EMD has been widely used in the analysis of probability distributions, e.g., content-based image retrieval [20, 25, 30, 31, 33], database foreign key identification [40] and database privacy protection [24]. To apply EMD on probabilistic data, a probabilistic record is represented by discrete probabilities on a group of disjoint *bins* that partitions the data domains. EMD models the dissimilarity between two probability distributions as the minimal work required of moving *earth* (i.e., probabilities) from the *source bins* to the *sink bins* until one distribution is equal to the other. The work is measured by the amount of earth that is moved (called *flow*) and the distance moved (called *ground distance*). Compared to traditional *bin-by-bin* distances, such as L_p norms, EMD not only considers the dissimilarities between each pair of aligned bins but also allows the probabilities to flow among unaligned bins. Thus, EMD is more robust to outliers and small probability shifts, improving the robustness of the similarity metric. In the following, we introduce three examples to better illustrate the sources of probabilistic data as well as the usefulness of EMD-based similarity queries in these scenarios.

Example 1 Recent years have witnessed the emergence of wireless sensor networks as an extremely helpful tool for monitoring tasks in extreme environments. A primary challenge for these networks is to minimize the energy consumption. To save energy, a common approach is to decrease the amount of data traffic within the network. To this extent, probabilistic models, such as BBQ [18] and Ken [12], are devised that try to estimate the distributions of the measurements based on previous readings, and thus reduce the need for transmission operations. The estimation is based on historical data using some standard machine learning algorithms, such as *Bayesian learning approach* [17]. In Figure 1, we illustrate a probabilistic model of possible readings from sensor nodes monitoring temperature and humidity. The 2D space regarding temperature and humidity is divided into 16 bins. The distribution of the measurements of sensor s_1 in Figure 1(b) for example, indicates that s_1 is most likely to observe humidities in the range of $[30\%, 40\%]$ with temperatures within $[20°C, 25°C]$. Every distribution is thus represented by a histogram, e.g., $p_{s_1} = (p_{s_1}[1], p_{s_1}[2], \ldots, p_{s_1}[h])$, where $p_{s_1}[i]$ is the probability of s_1's reading falling into bin i and h represents the number of bins. In Table 1, we list the sensors' reading distributions illustrated in Figure 1, in which $h = 16$ bins are numbered by increasing humidity and increasing temperature. Under this scenario, EMD-based similarity queries with respect to a query q with high probabilities on high temperature and low humidity, can be helpful for a fire monitoring and alarming system.

[1] For brevity, the probabilistic data represented in the form of a histogram will be simply termed as *histogram-representative probabilistic data* in the rest of this paper.

Table 2 Example probabilistic records for DBLP database stored in a relational table

Author	AI	Application	Bioinformatics	Database	Hardware	Software	System	Theory
John Doe	0.109	0.109	0.059	0.314	0.0987	0.091	0.123	0.093
Jane Doe	0	0.1	0.3	0	0	0.01	0.59	0

Example 2 With the emergence of the Internet, the World Wide Web has become an important source of valuable information and knowledge. Due to the unstructured organization of content, extracting information from the Internet is often rather imprecise. Probabilistic representations are thus introduced to measure the uncertainties of specific contents connected with different topics and keywords [37]. In Table 2, for example, we present a table with probabilities of computer science researchers related to eight different research topics, namely *AI, Application, Bioinformatics, Database, Hardware, Software, System,* and *Theory*, based on analysis of DBLP[2] [41]. A distribution-based similarity query with respect to the record of John Doe for example, results in a list of authors with similar publication venues, helping us to better understand research communities related to computer science. Note that these eight topics are sometimes correlated, e.g., some researchers on data mining publish papers in both the *AI* and *Database* topics. The correlations between different topics can be captured by appropriately defining the ground distances in EMD.

Example 3 Earth Mover's Distance originates from similarity search techniques in image databases. The highly effective capability of EMD for image retrieval has been demonstrated in the seminal paper by Rubner *et al.* [28]. By extracting the probabilistic distribution of a certain image feature (e.g., color, shape or texture [30]), an EMD-based similarity search will return identical images with respect to probability distributions over these features. Figure 2 shows an example of extracting the RGB color distribution from an image. The RGB color distribution, in the form of a 3D histogram, is produced by discretizing the color space into a number of bins. For example, the RGB color space in Figure 2 is partitioned into 216 bins by dividing each color channel into six domains. The probabilities are calculated by counting the number of pixels falling into each bin and dividing the counts by the total number of pixels. Setting the RGB color histogram of the famous painting *Mona Lisa* as the query, the EMD-based similarity query identifies all images in the database that have similar colors to the *Mona Lisa*.

In the above examples, probabilistic data are represented by discrete probability distributions. This representation is consistent with probabilistic data models proposed in state-of-the-art probabilistic relational databases, such as the *probabilistic tuple* proposed in the ProbView system [22], and the *maybe x-tuple* defined in the TRIO project [9]. The Earth Mover's Distance, as a very powerful distance measure, can help to enhance the quality of similarity queries on probabilistic data.

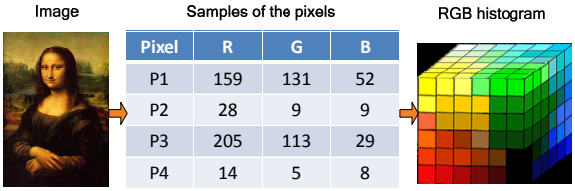


Fig. 2 Example of probabilistic record in image databases

While the storage of these probabilistic records is relatively easy, calculating the EMD is rather difficult, since it is equivalent to solving a linear programming problem with complexity of $O(h^3 \log h)$ where h is the number of bins in the probabilistic record. To relieve this efficiency bottleneck, a number of approximation techniques to reduce the computational complexity of EMD have been proposed in the computer vision [25,33, 31] and algorithm design communities [5]. While these techniques accelerate the calculation of EMD between two probabilistic records, however, they all suffer from a performance deterioration when they are applied in a database with a large number of records.

In recent years, effort has been made to address the similarity search problem using EMD. Most approaches to design scalable solutions utilize efficient and effective lower bounds estimators for EMD [6,38]. These solutions are mainly built within the *Scan-and-Refine* framework, which incurs high I/O costs and render low processing concurrency in the database systems. To overcome the difficulties of these methods, we present a general approach to provide a truly scalable and highly concurrent indexing scheme applicable to mainstream relational databases, such as *PostgreSQL*[3] and *MySQL*[4].

In our approach, all probabilistic records are mapped to a group of one-dimensional domains by using the *Primal-Dual Theory* [26] in linear programming. For

[2] `http://dblp.uni-trier.de/`

[3] `http://www.postgresql.org/`

[4] `http://www.mysql.com/`

each one-dimensional domain, a B^+ tree is constructed to index pointers to probabilistic records based on their mapping values. Given a range query search for probabilistic records from a querying histogram within some specified threshold, our approach transforms the original query to a group of one-dimensional range queries on these mapping domains, while guaranteeing that a valid query result must reside within all of the querying ranges on the B^+ trees. These one-dimensional range queries are thus executed in all B^+ trees. Candidates to the original range query are selected by means of an intersection operation on all results from the range queries. Refinements and verifications are then conducted on the remaining candidates, and the final query results are returned. To answer k-Nearest Neighbor (k-NN for short) queries, we designed a progressive processing algorithm, which automatically adjusts the search range after examining partial results from the preliminary range queries.

While traditional similarity queries are important for the analysis and management of probabilistic records stored in databases, continuous queries on probabilistic data are recently emerging as an equally important problem in many physical-world applications. The proliferation of online video web sites, such as *YouTube* and *Microsoft Soapbox*, has provided Internet users with high flexibility in regards to video sharing. The popularity of video sharing however, has brought serious problems of copyright violations. It is thus important for such systems to support online real-time video frame copy detection, to pinpoint potential problems with newly uploaded videos. Environment monitoring systems using sensor networks are another example, since there are often pressing requirements on the real-time monitoring of environmental changes, in order to be able to timely warn for potential hazardous situations. All of these problems can be consistently solved if the database system is capable of processing continuous similarity queries over a dynamic probabilistic data stream. Specifically, the system allows users to register different range queries, each of which consists of a probabilistic histogram and querying range (e.g., the color histogram of a key frame in a movie and a certain error tolerance parameter). For each uploaded frame, represented by its color histogram, the system quickly evaluates it for all of the registered queries and then reports the similar frames to the users.

Although it is straightforward to apply our principle of handling one-shot queries to prune unpromising records in the streaming environment, it remains difficult to enhance the throughput of the system when hundreds of queries are registered at the same time. In order to further improve the performance of the system, we devise new strategies to reduce the computational workload. While a group of feasible solutions derived from the dual program of EMD are also used to examine the qualification of an incoming probabilistic record to each registered query, we enhance the pruning ability of those feasible solutions by adaptively adjusting them based on the incoming record. Moreover, we carefully design a computation sharing mechanism for the condition that similar queries are registered in the system.

The major contributions of the paper are summarized below. Note that this paper is an extended version of [39], with new technical contributions on the concurrency protocol implementation and continuous query processing.

1. We present what is, to our knowledge, the first tree-based indexing structure to support similarity search on histogram-representative probabilistic data based on the Earth Mover's Distance.
2. We propose a new query processing technique by transforming similarity search queries to a group of range queries on one-dimensional mapping domains.
3. We discuss a progressive searching method to support k-NN queries with dynamic search range updates.
4. We design and analyze the concurrency protocol in our system architecture and empirically evaluate the transaction concurrency.
5. We extend our techniques to handle continuous monitoring queries on probabilistic data streams and devise optimization methods to enhance efficiency.

The rest of the paper is organized as follows. Section 2 introduces the preliminaries and problem definitions. Section 3 discusses our index structure on the probabilistic records using B^+ trees. Section 4 presents the details on the algorithms for one-shot similarity queries, i.e., range queries and k-nearest neighbor queries. Section 5 extends our techniques to handle continuous similarity queries. Section 6 evaluates our proposals with experiments on physical-world data sets. Section 7 reviews related works on probabilistic databases and Earth Mover's Distance and Section 8 concludes the paper.

2 Preliminaries

In this paper, we discuss the management of probabilistic records represented by *histograms*. A histogram can be defined as a *probabilistic tuple* in the ProbView system [22] or a *maybe x-tuple* in the Trio project [9]. We use $\mathbb{D}$ to denote the original object domain, covering all possible states of the objects in the physical world. Depending on the domain specific knowledge, the object

domain is partitioned into a suitable number of h bins. The probabilistic record of an object, represented by a histogram, thus records the probabilities of the object appearing in the respective bins/states.

To define *Earth Mover's Distance*, a metric ground distance on $\mathbb{D}$, d_{ij}, is provided to measure the distance between any pair of bins i and j. If Manhattan distance is employed as d_{ij}, in the example of Figure 1(a), we have $d_{ij} = 2$ when $i = 10, j = 13$ and $d_{ij} = 1$ when $i = 7, j = 8$. Given the ground distance d_{ij}, the formal definition of *Earth Mover's Distance* is given below[5].

Definition 1 Earth Mover's Distance (EMD) Given two probabilistic records p and q, the *Earth Mover's Distance* between p and q, $EMD(p, q)$, is the optimum achieved by the following optimization program:

$$\begin{aligned} Minimize: & \sum_{i=1}^{h}\sum_{j=1}^{h} f_{ij}d_{ij} \\ s.t. \ & \forall i : \sum_{j} f_{ij} = p[i] \\ & \forall j : \sum_{i} f_{ij} = q[j] \\ & \forall i, j : f_{ij} \geq 0 \end{aligned} \quad (1)$$

A total of h^2 variables, denoted by $F = \{f_{ij}\}$, are used in the program above. Intuitively, each $f_{ij} \in F$ is the probability flow from bin i of p to bin j of q. Therefore, F forms a complete flow from probabilistic record p to probabilistic q, if and only if 1) the sum of flows from bin i of p is exactly $p[i]$; 2) the sum of flows to bin j of q is exactly $q[j]$; and 3) all flows are non-negative, which are describe by the three constraints listed in the program above. The cost of flow set F is formed by the weighted sum over all individual flows between every pair of bins. The Earth Mover's Distance is the cost of the optimal flow set (denoted as F^*), that minimizes the cost of transforming p to q. In Figure 3, we present the optimal flow set from probabilistic record p_{s_1} to probabilistic record p_{s_2} in Figure 1 and Table 1, with Manhattan distance as the ground distance d_{ij} for the domain of sensor readings. It is thus straightforward to verify that $EMD(p_{s_1}, p_{s_2}) = 1.1$.

In this paper, we study two types of similarity search queries, namely *Range Query* and *k-Nearest Neighbor Query*. In particular, given a snapshot of the database $D = \{p_1, p_2, \ldots, p_n\}$ with n histogram records p_i, a Range Query $\mathrm{RQ}(q, \theta)$ consists of a querying probabilistic record q and a threshold θ. The result of $\mathrm{RQ}(q, \theta)$ contains all records in D with an EMD to q no larger than θ, or in other words $\mathrm{RQ}(q, \theta) = \{p_i \in D \mid EMD(p_i, q) \leq \theta\}$. Similarly, a k-Nearest Neighbor query $k\mathrm{NN}(q, k)$ with respect to a record q and a positive integer k, finds exactly k records in D with the lowest EMD values with respect to q. For examples of range queries and k-NN queries in physical-world applications, the reader is referred to Example 1-3.

[5] None of the methods proposed in this paper depend on the selection of the ground distance function d_{ij}.

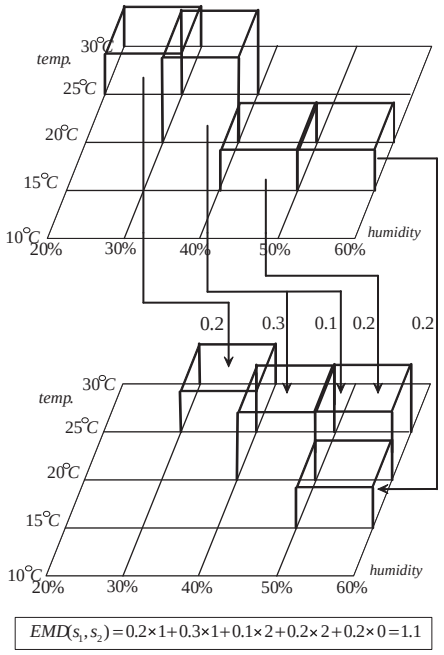


Fig. 3 The optimal flow set from distribution p_{s_1} to distribution p_{s_2}

In most commercial database systems, concurrency control is an important issue. Database updates and search queries are usually executed simultaneously in such systems, and it is also desirable for a probabilistic database to maximize concurrency. We therefore ensure that our index scheme supports a high level of concurrency. In summary, the basic problem of similarity search we want to solve in this paper is formalized in Problem 1.

Problem 1 Design a database indexing scheme, which supports efficient range queries and k-NN queries based on EMD, as well as concurrent updates at the same time.

For Problem 1, we want to emphasize that every user should be able to issue queries to the database at any time and the system returns similarity search results to the user based on the current records in the database. In some applications, this method may not be suitable. Besides finding similar records on the current snapshot of the database, users may also be interested in monitoring every new record coming into the database. In the following, we will give a concrete example on such an environment.

Example 4 With the proliferation of online video sharing in recent years, video copyright infringement is appearing as a major concern for video sharing systems, such as YouTube and Facebook. There are pressing needs for these systems to identify videos with potential problems whenever they are uploaded. A possible solution is to keep a pool of commercial movies in the system and checks if any new uploaded video clip includes content from these movies. In particular, it employs EMD as the underlying distance metric to compare the key frames from the known movies and the uploaded video clips from its users. If the EMD between frames are less than some error tolerance parameter, the system blocks the display of the video clip.

To meet the requirements of such applications, multiple *Continuous Similarity Queries* are to be registered in the database system. The system then continuously updates the query results for each registered query upon the insertion of new probabilistic records [7]. To summarize, the formal problem formulation of continuous similarity search query is provided below.

Problem 2 Given a group of *Continuous Similarity Queries* $\text{CSQ}(q_i, \theta_i)$, for each incoming probabilistic record p, find out every $\text{CSQ}(q_i, \theta_i)$, such that $EMD(q_i, p) \leq \theta_i$.

To solve both problems listed above, we present several new methods to handle EMD, utilizing the primal-dual theory from linear programming. In the rest of this section, we provide a brief review of the primal-dual theory. For a more detailed explanation of this theory, the reader is referred to [26].

The primal-dual theory states that, for any linear program with a minimization objective, there always exists one and only one dual program with maximization objective. Thus, given the formulation of EMD in Definition 1, its dual program can be constructed as follows. In the dual program, there are $2h$ variables, $\{\phi_1, \phi_2, \ldots, \phi_h\}$ and $\{\pi_1, \pi_2, \ldots, \pi_h\}$, each of which corresponds to one constraint in the primal program. The dual program can thus be written as:

$$\begin{aligned} Maximize: \ & \sum_{i=1}^{h} \phi_i \cdot p[i] + \sum_{j=1}^{h} \pi_j \cdot q[j] \\ s.t. \ & \forall i, j : \phi_i + \pi_j \leq d_{ij} \\ & \forall i : \phi_i \in \mathbb{R} \\ & \forall j : \pi_j \in \mathbb{R} \end{aligned} \tag{2}$$

Given a linear program, a feasible solution to the program is a set of variable values satisfying all constraints in the program but not necessarily optimizing the objective function. There can be an arbitrarily large number of feasible solutions to a linear program. Assume that $F = \{f_{ij}\}$ and $\Phi = \{\phi_i, \pi_j\}$ are two feasible solutions to the primal program (Equation (1)) and the dual program (Equation (2)) of EMD respectively. We have:

$$\sum \phi_i p[i] + \sum \pi_j q[j] \leq EMD(p, q) \leq \sum f_{ij} d_{ij} \tag{3}$$

Equation (3) directly implies the existence of a lower bound and upper bound on the EMD between p and q. Our index scheme mainly relies on the feasible solutions to the dual program. The upper bound, derived with the feasible solution to the primal program will be covered in Appendix A [1], which is used as a filter in range query and k-NN query processing. In the following, we first present a simple example of a feasible solution to the dual program.

Example 5 It is easy to verify that there is a trivial feasible solution with $\phi_i = 1$ for all i and $\pi_j = -1$ for all j, if d_{ij} is a metric distance, i.e., $d_{ij} \geq 0$ for any i and j. This feasible solution leads to a trivial lower bound on $EMD(p, q)$:

$$EMD(p, q) \geq \sum \phi_i p[i] + \sum \pi_j q[j] = \sum p[i] - \sum q[j] = 0$$

We want to emphasize that all the constraints in Equation (2) involve only $\Phi = \{\phi_i, \pi_j\}$ and d_{ij}. This implies that the feasibility of a solution Φ only depends on the distance metric d_{ij}, rather than p and q. Thus, it is possible to derive a feasible solution for index construction, regardless of the data distribution and the query.

3 Index Structure and Pruning Principles

Generally speaking, our index structure employs a forest of B^+ trees, $\{T_1, \ldots, T_L\}$, to index pointers to the probabilistic records in database D. Each tree T_l in the forest is associated with a feasible solution $\Phi_l = \{\phi_i^l, \pi_j^l\}$ to the dual program of EMD. This feasible solution Φ_l uniquely formulates a transformation from the probabilistic domain to a one-dimensional space and facilitates the indexing by the B^+ tree according to the mapping values. Section 3.1 discusses the details of the transformation and an index on the mapping values. Furthermore, Section 3.2 provides some guidelines for the selection of feasible solutions for the B^+ trees.

3.1 Mapping to One-Dimensional Space

To facilitate the introduction of the mapping construction, we first define the concepts of *key* and *counter-key* below.

Definition 2 Key/Counter-Key
Given a probabilistic record p and a feasible solution $\Phi_l = \{\phi_i^l, \pi_j^l\}$ to the dual program of EMD, the *key* of p w.r.t. Φ_l given by:

$$key(p, \Phi_l) = \sum\nolimits_i \phi_i^l \cdot p[i]$$

The *counter-key* of p w.r.t. Φ_l is defined as:

$$ckey(p, \Phi_l) = \sum\nolimits_j \pi_j^l \cdot p[j]$$

Given a selected feasible solution Φ_l, the associated B^+ tree T_l simply indexes all pointers to probabilistic records based on the value of $key(p, \Phi_l)$. Note that the calculations on both $key(p, \Phi_l)$ and $ckey(p, \Phi_l)$ take only $O(h)$ time, linear to the number of bins in the object domain. It is also important to emphasize again that Φ_l is independent with respect to the query, facilitating the computation of $key(p, \Phi_l)$ before the insertion of p into T_l.

To efficiently support similarity search queries, we build up the connection between the key/counter-key and the Earth Mover's Distance. Specifically, the following two equations derive the lower bound and upper bound on $key(p, \Phi_l)$, in terms of any query record q and the distance between p and q.

Given a record p indexed by T_l and a query record q, based on the primal-dual theory shown in Equation (3), it always holds that:

$$key(p, \Phi_l) \leq EMD(p, q) - ckey(q, \Phi_l) \quad (4)$$

Moreover, it can be shown that:

$$key(p, \Phi_l) \geq \min_i(\phi_i + \pi_i) + key(q, \Phi_l) - EMD(p, q) \quad (5)$$

Proof Due to the symmetry property on the metric distance, we have $EMD(p, q) = EMD(q, p)$. Thus, a lower bound for $EMD(q, p)$ is also a lower bound for $EMD(p, q)$. By applying Equation (4), we have:

$$key(q, \Phi_l) + ckey(p, \Phi_l) \leq EMD(q, p) = EMD(p, q) \quad (6)$$

Additionally, when summing up $key(p, \Phi_l)$ and $ckey(p, \Phi_l)$, the following inequalities can be derived.

$$\begin{aligned} key(p, \Phi_l) + ckey(p, \Phi_l) &= \sum\nolimits_i \phi_i p[i] + \sum\nolimits_j \pi_j p[j] \\ &= \sum\nolimits_j (\phi_j + \pi_j) p[j] \\ &\geq \sum\nolimits_j \min_i(\phi_i + \pi_i) p[j] \\ &= \min_i(\phi_i + \pi_i) \end{aligned} \quad (7)$$

The last equality holds due to the fact that $\sum_j p[j] = 1$. After combining Equation (6) and Equation (7), some simple algebraic operations bring us to the proof of the Equation (5). □

Based on the Equation (4) and Equation (5) above, for all result records that satisfy a specific range query $\mathrm{RQ}(q, \theta)$, their key values must be located in the interval of the domain constructed by Φ_l which is associated with T_l. That is:

$$\begin{aligned} &key(p, \Phi_l) \\ &\in \left[\min_i(\phi_i + \pi_i) + key(q, \Phi_l) - \theta, \theta - ckey(q, \Phi_l)\right] \end{aligned} \quad (8)$$

This implies a simple scheme to handle range queries w.r.t. EMD. Given a range query $\mathrm{RQ}(q, \theta)$, a group of one-dimensional sub-queries are constructed by means of Equation (8), according to each Φ_l associated with T_l. These sub-queries are then run on the corresponding B^+ trees. A probabilistic record p is a valid candidate for $\mathrm{RQ}(q, \theta)$, only when p appears in all results of the sub-queries. Therefore, the intersection of all sub-query results generates a candidate set for $\mathrm{RQ}(q, \theta)$. Details of the algorithms will be covered later in Section 4.1.

There are a couple of important advantages to our indexing scheme from a system's perspective. First, the B^+ tree is an I/O efficient structure for query processing. The existing solutions to similarity searches using EMD based on the *Scan-and-Refine* framework, incur high I/O costs during candidate selection. Our scheme dramatically alleviates this problem by using the B^+ tree to conduct the first candidate pruning step. Secondly, the B^+ tree structure is a well studied and optimized data structure, available in most commercial relational database package, making it easy to implement our scheme in any system. This helps to reduce the development difficulty of our index structure. Moreover, our indexing scheme is able to achieve high throughput in physical-world applications and directly supports transactions, since the B^+ tree structure is friendly to transaction management and supportive to high concurrency. These properties widen the applicability of our indexing scheme, especially in web-based applications, when different participants are simultaneously updating and querying the database system.

It is also worthwhile to note that alternative solutions are possible for indexing the probabilistic records. One could choose to employ other multidimensional index trees, such as an R Tree instead of a group of B^+ trees. However, the curse of dimensionality will lead to inefficient pruning and low concurrency performance. The architecture employing B^+ trees also enhances the flexibility of the system with respect to the number of adopted feasible solutions. A B^+ tree associated with a specific feasible solution can be easily inserted or removed at run time, without affecting the other trees. This facilitates simple tuning of the system when the data distribution changes over time.

3.2 Selection of Feasible Solutions

The performance of the indexing scheme depends on the selection of the feasible solutions $\{\Phi_l\}$ for the B^+ trees $\{T_l\}$. In Example 5, we showed that some feasible solutions only provide trivial bounds on EMDs. In this section, we will discuss the issue of finding better feasible solutions to improve the efficiency of similarity query processing.

The first question we will study is whether we can construct a feasible solution minimizing the gap between the lower bound and upper bound in Equation (8). Intuitively speaking, a smaller gap will lead to better pruning effects. Unfortunately, the following lemma shows that there is a lower bound on the gap which indicates that no matter how we perform the optimization the gap cannot be zero.

Lemma 1 *For any feasible solution Φ_l, the gap between the lower and upper bound used on the range query $RQ(q,\theta)$ in Equation (8) can not be smaller than 2θ.*

Proof The gap between the lower bound and upper bound on the range query $\mathrm{RQ}(q,\theta)$ in Equation (8) is minimized with the following inequalities.

$$\begin{aligned}
&(\theta - ckey(q,\Phi_l)) - \Big(\min_i(\phi_i+\pi_i) + key(q,\Phi_l) - \theta\Big)\\
&= 2\theta - \min_i(\phi_i+\pi_i) - (ckey(q,\Phi_l) + key(q,\Phi_l))\\
&\geq 2\theta - (ckey(q,\Phi_l) + key(q,\Phi_l))\\
&\geq 2\theta \qquad (9)
\end{aligned}$$

The first inequality is due to the metric property of d_{ij} and the constraint on ϕ_i and π_j, i.e., $\phi_i+\pi_i \leq d(i,i) = 0$. The second inequality is derived by employing the fact that the lower bound of $EMD(q,q)$ equals to 0. □

To find a feasible solution with the minimum gap for all data records is non-trivial. Due to the complexity of the high-dimensional probabilistic data space, a perfect feasible solution for probabilistic record p_1 and query q may not be a good choice for probabilistic record p_2 and q. Because we do not have an algorithm that provably finds a gap-minimizing feasible solution, we adopt two heuristic schemes to generate near-optimal feasible solutions for the dual program of EMD. Generally speaking, our selection method tries to avoid *dominated feasible solutions.*

Definition 3 A feasible solution Φ is dominated by another feasible solution Φ', if $\phi'_i \geq \phi_i$ for all i and $\pi'_j \geq \pi_j$ for all j.

A dominated feasible solution is undesirable, since it will always lead to weaker bounds, i.e., $key(p,\Phi) \leq key(p,\Phi')$ and $ckey(q,\Phi) \leq ckey(q,\Phi')$ for any p and q, when Φ is dominated by Φ'. Basically, our scheme depends on the next lemma to eliminate dominated feasible solutions.

Lemma 2 *If Φ is the optimal solution to the dual program on $EMD(p,q)$ for any p, q, it is not dominated by any other feasible solution Φ'.*

Proof If there exists some feasible solution Φ' dominating Φ, it is true that $\phi'_i \geq \phi_i$ for all i and $\pi'_j \geq \pi_j$ for all j. This leads to the following inequality.

$$\sum\nolimits_i \phi'_i p[i] + \sum\nolimits_j \pi'_j q[j] \geq \sum\nolimits_i \phi_i p[i] + \sum\nolimits_j \pi_j q[j] \quad (10)$$

Since Φ' is also a feasible solution to the constraints, Φ' is then a better solution than Φ to the dual program on $EMD(p,q)$. This contradicts to the optimality condition of Φ for the dual program. Therefore, such Φ' does not exist. □

Lemma 2 shows that a non-dominated feasible solution can be identified by calculating the optimal solution to the dual program of $EMD(p,q)$ for any pair of probabilistic records p and q. Therefore, the selection of a feasible solution for tree T_l is equivalent to the selection of an appropriate probabilistic record pair (p,q). In the following, we present two heuristic schemes for this selection procedure.

Clustering-Based Selection: In this scheme, the system applies a uniform sampling algorithm to retrieve a small sample set S from the probabilistic record database D. A clustering algorithm is then run on the sample set S to discover a group of representative records, i.e., $R = \{p_1, p_2, \ldots, p_k\}$. For every pair of p_i and p_j in R, the system calculates the optimal solution to the dual program of $EMD(p_i,p_j)$ using the *Alpha-Beta Algorithm* [26]. This generates $\frac{k(k-1)}{2}$ feasible solutions for our B^+ trees to use. To ensure every tree T_l is assigned with one feasible solution Φ_l, we select a sufficiently large k so that $\frac{k(k-1)}{2} \geq L$, where L is the number of B^+ trees.

Random-Sampling-Based Selection: The clustering-Based Selection is rather expensive due to the clustering phase. To reduce the computational costs, a much cheaper random-sampling-based scheme is proposed. Given a probabilistic database D, for each tree T_l, the new scheme randomly picks two records p_i and p_j from D. The optimal solution to the dual program of $EMD(p_i,p_j)$ is used for mapping probabilistic records to key values in tree T_l. In Section 6, we will provide detailed empirical evaluations on the performance of the two schemes defined above.

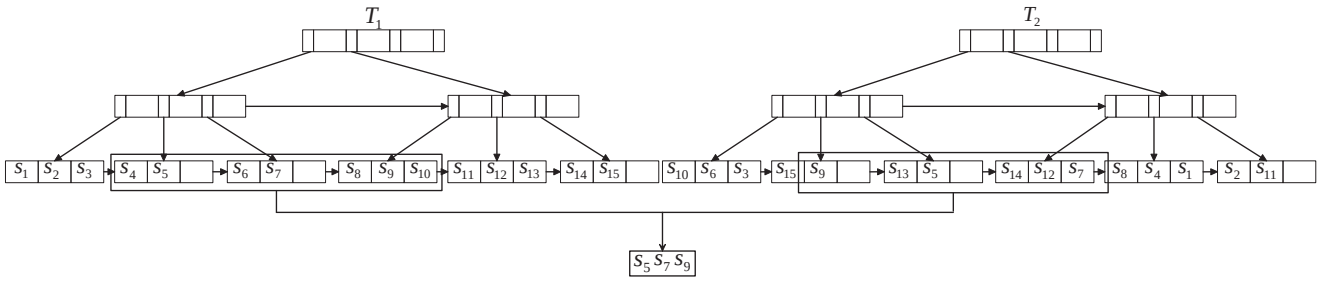


Fig. 4 Example of the candidate selection with the B^+ trees

4 Algorithms on Range Query and k-NN Query

In this section, we present detailed algorithms for *Range Query* (Section 4.1) and *k-Nearest Neighbor Query* (Section 4.2). In Section 4.3 we will discuss the implementation of a concurrency protocol based on similar protocols as defined in traditional relational database. The algorithms presented in this section provide a complete solution to Problem 1 introduced in Section 2.

4.1 Range Query

Based on the indexing scheme and Equations (4) and (5) introduced in the previous section, it is straightforward to design processing algorithms for range queries.

In Figure 4, we present a running example to illustrate the candidate selection phase with the help of the B^+ trees. Assume that there are 15 probabilistic records $\{s_1, s_2, ..., s_{15}\}$ indexed in the database, and two B^+ trees are constructed with Φ_1 and Φ_2 as feasible solutions respectively. Given a range query $\mathrm{RQ}(q, \theta)$, the algorithm first generates two sub-range queries for T_1 and T_2, according to Equation (4) and Equation (5) derived in Section 3.1. As shown in the figure, the two sub-queries return two different sets of candidates. In particular, the query result from T_1 contains seven candidates, namely $\{s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$. Similarly, T_2 returns six candidates based on the sub-range query, namely $\{s_9, s_{13}, s_5, s_{14}, s_{12}, s_7\}$. The intersection of the two sub-query results leads to the final candidate set, $\{s_5, s_7, s_9\}$.

Given the resulting candidates of the intersection operation, a number of filters are run in order to further prune the candidates. Specifically, two existing filters are employed in our algorithm, *R-EMD* (*EMD in the Reduced space*) [38] and LB_{IM} (Lower Bound filter based on the Independent Minimization) [6]. For details on the two pruning filters, we refer interested readers to Appendix B [1]. A new Upper Bound filter, named UB_P, is employed to identify candidate records that certainly belong to the final result. In this way, the exact EMD is only calculated for a small subset of records for which it is not obvious whether they are valid answers to the range query. The UB_P filter is based on the upper bound on $EMD(p, q)$ derived by the feasible solution to the primal program of EMD. Details on the filter are available in Appendix B.

4.2 k-Nearest Neighbor Query

While a range query is answered by intersecting candidates from the results of sub-range queries on the B^+ trees, the algorithm for k-NN query is more complicated. The complication is due to the difficulty of knowing the optimal search range to return exactly k nearest neighbors. Intuitively, our algorithm for a k-nearest neighbor queries implicitly generates a sequence of candidate records based on the B^+ tree index structure. These candidates are then verified with both filters and with exact EMD computations. The pruning thresholds are accordingly refreshed when new records more similar to query q are inserted into the temporary result buffer. The whole algorithm terminates when all records have been pruned or verified. In the remainder of this section, we will give a concrete example to illustrate the procedure of the algorithm.

Given a query $k\mathrm{NN}(q, k)$, with queried probabilistic histogram record q and a positive integer k, the algorithm first constructs two iteration cursors for each B^+ tree T_l. For T_l, the algorithm calculates the $key(q, \Phi_l)$, based on the querying record q and the feasible solution Φ_l. A search query with $key(q, \Phi_l)$ is executed on T_l to locate the pointer in the index with a mapping value closest to $key(q, \Phi_l)$. In Figure 6, for example, $key(q, \Phi_1)$ is located in tree T_1 between record s_5 and s_6. After the positioning with the value $key(q, \Phi_1)$, the algorithm builds two cursors $\overrightarrow{C_1}$ and $\overleftarrow{C_1}$, which are used to crawl the pointers from the current location in the right and left directions respectively. This implicitly generates two iteration lists on the record pointers. Since there are two trees in our example, the algorithm also initializes $\overrightarrow{C_2}$ and $\overleftarrow{C_2}$ to visit the pointers iteratively on the other tree T_2.

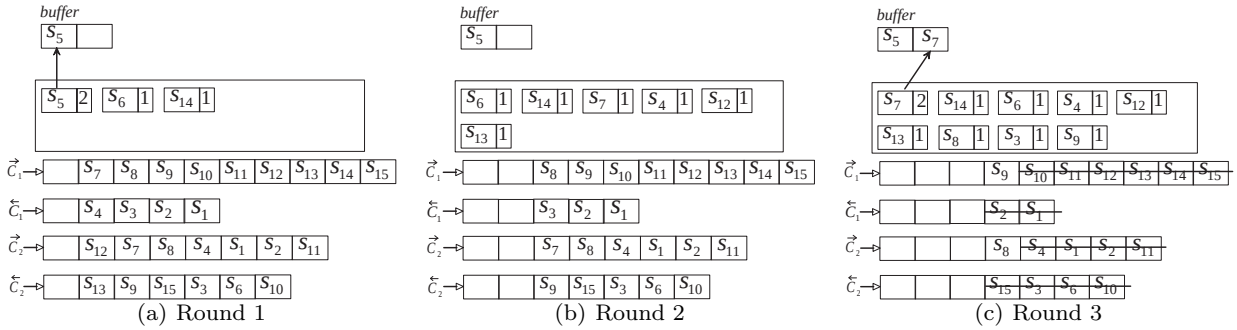


Fig. 5 Running example of k-NN query processing

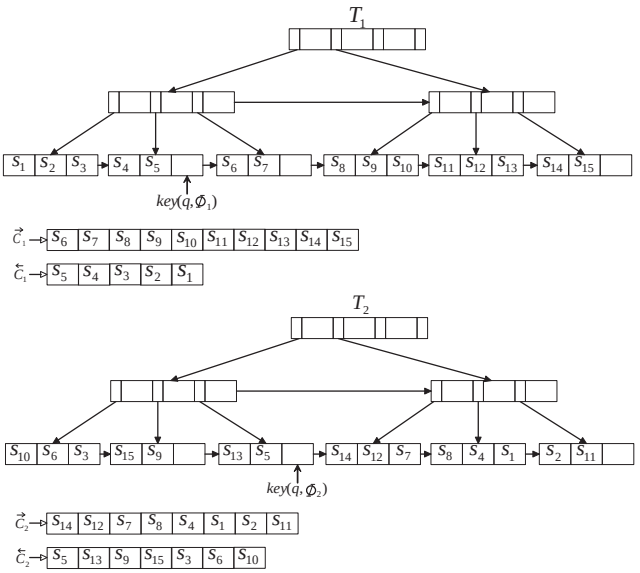


Fig. 6 Construction of cursors for k-NN query

If the system is equipped with L different B^+ trees, there are totally $2L$ cursors initialized on these B^+ trees. In the following, our algorithm applies the query processing strategy similar to *Top-k Query* [19]. Before starting the iterations, a temporary buffer for k-nearest neighbor results is created. In each iteration, all cursors fetch the next record pointer on their lists in a round robin manner. A candidate is confirmed only when it is visited by exactly L cursors of different trees. Once a candidate is confirmed, it is verified with the filters and finally evaluated by means of exact EMD computation if necessary. The new candidate p is directly inserted into the temporary buffer if it contains less than k records. If there are k records in the buffer and $EMD(p, q)$ is smaller than at least one existing record in the buffer, p replaces the record in the buffer which has the maximal distance to querying record q. When such a replacement happens on the buffer, the algorithm also updates the maximal distance threshold for other unvisited records in the iteration lists. In particular, the distance threshold is derived according to the maximal distance from the records in the buffer to the query record q, similar to the strategy used for the range query. Such distance thresholds help the algorithm to prune candidates unlikely to be closer to q than the existing records in the buffer. In Figure 5, we give an example to show how the algorithm iterates on the cursor lists, with the parameter $k = 2$. In the first iteration, all cursors read the first record in their lists. Since s_5 is on the top of two cursor lists, it is directly added into the temporary buffer. The second iteration does not select any record because no new record has accumulated enough appearances in the cursor lists. The third iteration selects the record s_7. The buffer now consists of $k = 2$ candidates, letting the algorithm know that records with distance larger than $EMD(s_7, q)$ cannot be part of the query result. This leads to the elimination of the records on the tails of the cursor lists. Similar to range query algorithm, our k-nearest neighbor query algorithm applies all the filters used in range query algorithm, except for the filter UB_P.

The complete pseudocode of the query processing is listed in Algorithm 1.

4.3 Concurrency Control

The algorithms presented in previous sections all assume that there is a single thread visiting the database to answer similarity queries. In practice, to maximize the usefulness of the system, most commercial relational database systems allow multiple threads to access the database at the same time. These threads can generally execute any type of operations on data, including record insertion, deletion and querying. As is pointed out in Problem 1, it is important for the indexing scheme to support concurrency protocols. If these

Algorithm 1 k**-NN Query** (query record q, parameter k, B^+ trees $\{T_l\}$)

```
1: for each T_l do
2:    find the record pointer p_l indexed in T_l with the closest
      mapping value to key(q, Φ_l)
3:    initialize C_l^→ and C_l^← from the pointer p_l
4: initialize each element in array status as 0
5: τ = ∞
6: while TRUE do
7:    for each T_l do
8:       if C_r^→.next(τ) ≠ NULL then
9:          rId = C_l^→.getNext()
10:         status[rId] + +
11:         if status[rId] == L then
12:            checkList.add(rId)
13:      if C_l^←.next(τ)! = NULL then
14:         lId = C_l^←.getNext()
15:         status[lId] + +
16:         if status[lId] == L then
17:            checkList.add(lId)
18:      if (cannot getNext in all trees)
         &&(checkList.empty==TRUE) then
19:         break the while loop
20:      for each element el_i in checkList do
21:         if max_l(key(el_i, Φ_l) + ckey(el_i, Φ_l)) > τ then
22:            break
23:         else if can be filtered by R-EMD then
24:            break
25:         else if can be filtered by LB_IM then
26:            break
27:         else
28:            if EMD(el_i, q) < τ then
29:               kNNList.add(el_i)
30:               if kNNList.size == k + 1 then
31:                  delete the one in kNNList with the
                     largest EMD to q
32:                  τ = max_i(EMD(kNNList[i], q))
33: Output all records in kNNList
```

protocols are not supported, the index structure will become the bottleneck of the probabilistic database, since all operations need to access it. Insertion and deletion operations, for example, must modify the index structure to refresh the existence of a record. Similarity queries, on the other hand, also rely on the consistency of the index to locate if records truly exist in the database. In this section, we design and analyze the concurrency control protocol on our tree-based indexing structure.

Our concurrency control protocol employs the *Read-Committed* isolation [10] from standard databases. Compared with the most stringent isolation, namely the *Serializable* isolation, Read-Committed isolation can preserve consistency and provide high concurrency performance, because it releases read locks as soon as the read operation is finished. Recall the example showing a range query in Figure 4. Based on the Read-Committed protocol, during the range searching on leaf nodes of tree T_1, the system locks[6] the record pointer s_4 at first. Once the s_4 is successfully read, the system releases its read lock and then continues to block s_5. Thus, the bottom level of tree T_1 from s_4 to s_{10} are locked one by one during the searching procedure. Moreover, the algorithm also locks the intermediate nodes in the tree which lead the algorithms from root to s_4. Similarly, record pointers s_9 to s_7 in tree T_2 are also sequentially blocked whenever their contents are accessed. Note that all of the locks are executed from left to right in the index structure, which is an important property for avoiding deadlock.

For a k-nearest neighbor query, the algorithm locks all intermediate structures routing to its reference pointer p_l on tree T_l. During the iterations of the algorithm, it orderly locks every pointer it has to access from both cursor lists on tree T_l, in effect executing the locks in both directions. In the following, we show that our index structure always outputs consistent results, no matter how many threads are allowed in the system.

Lemma 3 *Our index structure always outputs consistent query results when the concurrency protocol is applied.*

The proof of the lemma simply relies on the properties of the Read-Committed isolation [10]. Although Lemma 3 guarantees the consistency of the query results, it does not ensure the system is deadlock-free. For one thing, insertion or deletion in a B^+ tree might cause deadlocks, in that adding or removing pages sometimes requires the writing thread locks its parent page. Recall that when we locate a record, we lock pages from top to bottom. Thus, locking a parent page reverses the locking order and brings the potential risk for lock contention. Additionally, k-NN queries may conflict with other operations. The major reason behind the deadlocks caused by k-NN queries is that all other operations are visiting the index structure from left to right, while k-NN queries move the cursors in two reverse directions. It is thus impossible to avoid deadlocks in our system. Whenever the system encounters a deadlock, in our implementation, it rolls back the transaction with the least number of locks. In Section 6.2.3, we provide empirical evaluations of our concurrency protocol with respect to the system efficiency and the impact of the k-NN queries to the occurrence of deadlocks.

5 Algorithms for Continuous Similarity Queries

In the previous section we presented complete algorithms to solve Problem 1. In this section we address

[6] The locks mentioned in the processing of our range queries or k-NN queries refer to read locks.

Problem 2. In the setting of continuous similarity query processing, every new probabilistic record p coming into the system has to be compared against each registered query in the system. A first glance at the problem may suggest the use of a method similar to the one defined in the previous section. However, in Problem 2, every range query $\mathrm{RQ}(q_i, \theta_i)$ may have different values for the distance threshold parameter θ_i. This leads to difficulties in pruning the registered queries for the incoming probabilistic record p. To overcome these difficulties, we propose a different framework for processing continuous queries.

Given a feasible solution Φ_l to the dual program of EMD, every registered Continuous Similarity Query $\mathrm{CSQ}(q_i, \theta_i)$ is transformed into an interval on a one-dimensional domain by the transformation in terms of Φ_l. According to the theory derived in Section 2, we know a probabilistic record p is a candidate for the query result, only if $key(p, \Phi_l)$ is in the range of Equation (8). Therefore, we regard every query $\mathrm{CSQ}(q_i, \theta_i)$ as an interval in a one-dimensional space, i.e.,

$$\left[\min_j(\phi_j + \pi_j) + key(q_i, \Phi_l) - \theta_i, \theta_i - ckey(q_i, \Phi_l)\right]$$

For every feasible solution Φ_l, the system stores these intervals for all queries in an array structure A_l, sorted based on the lower boundaries of each intervals. For each incoming probabilistic record p, the system checks every array structure A_l and identifies all queries whose interval covers the corresponding key value of p. A query with a potential hit from each feasible solution is called a candidate query. After retrieving all candidate queries from all feasible solutions, query q_i can be the final candidate for p if it is accepted as the candidate query for all feasible solutions. After that, other filters and exact EMD computations are further run to verify if p is within the distance θ_i from the candidate query q_i.

The basic framework can be further improved in two ways. First, feasible solutions are usually precomputed by the system to construct the intervals for each registered query. When the distributions of the data and queries evolve with the incoming data stream, some of the feasible solutions may not be effective enough for candidate query pruning. It is thus desirable for the system to adaptively update feasible solutions at run-time. Secondly, when there are multiple queries registered in the system, it is possible to reduce the computational workload by reusing the results of other queries for evaluating the qualification of the current query. Such a computation sharing scheme is able to greatly cut the amount of calculations needed. In the rest of this section, we explore these two optimization techniques.

5.1 Adaptive Feasible Solution Update

When a probabilistic record p arrives in the stream, the system needs to check the qualification of p for each registered query q_i. Given a feasible solution set $\{\Phi_l\}$, the system will accept p as the candidate result of query q_i only if $key(p, \phi_l)$ falls into the interval of $[\min_j(\phi_j + \pi_j) + key(q_i, \Phi_l) - \theta_i, \theta_i - ckey(q_i, \Phi_l)]$, for every Φ_l. Therefore, a query q_i is rejected as the candidate query for record p if p fails to find its mapping position in any filtering interval of q_i constructed with Φ_l. As a result, the efficiency of the continuous query processing is highly proportional to the pruning ability of each feasible solution. In a few applications of continuous query processing, such as online video monitoring (see Example 4), arriving records are *temporally correlated*, i.e., the probability that two consecutive frames are similar is very high. This fact indicates that optimizations can be made by adaptively updating the feasible solutions based on the recent probabilistic record coming to the system. There are two technical problems in implementing this idea: 1) *When* to renew a feasible solution, so that the update will not become a burden to the system; and 2) *How* to evaluate the effectiveness of a feasible solution so that the least effective solution can be replaced. Next, we will define the trigger event for feasible solution update, after which the calculation of the effectiveness of a feasible solution will be explained in detail.

The most direct trigger event is to renew the feasible solution set based on each incoming record. However, gaining a new feasible solution requires running an exact EMD calculation which itself is a bottleneck for the query processing. In order to avoid incurring additional EMD refinements, we define the trigger event as:

Definition 4 Trigger Event
A trigger event for updating the feasible solution set is the occurrence of an inevitable exact EMD verification.

Whenever a trigger event occurs, a new feasible solution, denoted as Φ_{new}, is derived which will replace the currently least effective feasible solution. To select the least efficient feasible solution, the effectiveness of a feasible solution must be measured. A feasible solution can be evaluated based on the statistical data of its pruning ability. Assume that the system has a registered query set $Q = \{\mathrm{CSQ}(q_1, \theta_1), \ldots, \mathrm{CSQ}(q_N, \theta_N)\}$ and a feasible solution set $\Phi = \{\Phi_1, \ldots, \Phi_L\}$. Given a new record p, $I_l^n(p)$ is a binary indicator on the qualification of p for query q_n with respect to the feasible solution Φ_l. Specifically, $I_l^n(p) = 1$ if Φ_l is able to prune p for query q_n, otherwise $I_l^n(p) = 0$. After processing p with all the feasible solutions for all queries, we have a

binary matrix $M^p_{L\times N}$, named *Indicator Matrix*, which records every indicator $I^n_l(p)$. Based on the indicator matrix, we define two effectiveness scores to evaluate feasible solutions.

Definition 5 Independent Pruning Score (IPS) Given an indicator matrix $M^p_{L\times N}$, the *Independent Pruning Score* of Φ_l, denoted as IPS(Φ_l), equals to the number of 1s in l^{th} row of the $M^p_{L\times N}$.

Therefore, the independent pruning score measures the independent filtering capability of each feasible solution.

Definition 6 Dependent Pruning Score (DPS) Given an indicator matrix $M^p_{L\times N}$, the *Dependent Pruning Score* of Φ_l, denoted as DPS(Φ_l), is the number of unique 1s in the l^{th} row of the $M^p_{L\times N}$, while a 1 is unique when all other indicators amongst its column are equal to 0.

Thus, a feasible solution with a relatively high DPS value is better able to prune more unpromising queries which could not be eliminated by other solutions.

We give a small example to illustrate the calculation procedure on IPS and DPS. Given four feasible solutions ($\Phi_1, \Phi_2, \Phi_3, \Phi_4$) and ten registered queries ($q_1, \cdots q_{10}$), the indicator matrix based on the processing statistics with record p is shown in Figure 7, in which cells with unique 1s are marked in grey. Based on their respective definitions, IPS and DPS values for Φ_1 to Φ_4 are listed in Table 3.

With IPS and DPS values in the table, the system performs a two phases ranking on the feasible solutions. First, all feasible solutions are ranked in descending order according to their DPS values. We use DPS as our first ranking criterion because DPS measures the distinct pruning ability of one feasible solution. For those feasible solutions with the same DPS value, the system further ranks them by their IPS values. Consequently, the ranking of feasible solutions for our example in Table 3 is $\Phi_1 > \Phi_2 > \Phi_4 > \Phi_3$. It is intuitive to understand the meaning of the final ranking: Φ_1 is the most effective feasible solution, because it can prune two unique queries (i.e., q_4 and q_8) accepted by all other feasible solutions. Although Φ_2 and Φ_4 have the same DPS value, Φ_2 is more effective than Φ_4, since it can eliminate more records than Φ_4, as is shown on IPS values. Finally, Φ_3 is the least effective feasible solution for its DPS value equals to zero. Having the ranking of all feasible solutions, we automatically select the least effective for replacement. In our example, Φ_3 will be removed from the feasible solution set. After that a new feasible solution which is derived from the latest unavoidable EMD refinement will be inserted into the solution set to cooperatively prune following records with those remaining feasible solutions.

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
Φ_1	1	0	1	1	1	1	0	1	0	1
Φ_2	0	0	1	0	1	0	1	0	1	0
Φ_3	1	0	1	0	1	1	1	0	0	1
Φ_4	1	1	0	0	0	1	0	0	0	0

Fig. 7 Example of an indicator matrix based on the statistics of processing record p. The gray squares indicate unique 1s.

Table 3 Values for IPS and DPS

Feasible Solution	IPS	DPS
Φ_1	7	2
Φ_2	5	1
Φ_3	6	0
Φ_4	3	1

5.2 Multi-query Optimization

Because continuous queries are long-running and many applications may register a large number of continuous queries simultaneously over a shared data stream, multi-query optimization is of importance. Our main idea for multi-query optimization is based on the definition of an *Inferrable Query Set* and we give its formal definition in Definition 7.

Definition 7 Inferable Query Set (IQS) Given a continuous query set $Q = \{CSQ(q_1, \theta_1), \cdots, CSQ(q_N, \theta_N)\}$, and a certain data record p, the inferable query set of query q_i based on p, denoted as $IQS_{q_i,p} = \{q_j\}$, is a subset of Q which satisfies the following inequality.

$$\forall q_j \in IQS_{q_i,p} \quad EMD(q_i, q_j) - EMD(q_i, p) > \theta_j \qquad (11)$$

Theorem 1 ***Pruning Law for Multi-query Optimization***
Given a data record p and a query q_i, p will not be in the answer set for any query in $IQS_{q_i,p}$.

Proof Given any query q_j, based on the *Triangle Inequality*, we have $\text{EMD}(q_j, p) \geq \text{EMD}(q_i, q_j) - \text{EMD}(q_i, p)$. Thus, $\text{EMD}(q_i, q_j) - \text{EMD}(q_i, p)$ derives a lower bound for $\text{EMD}(q_j, p)$. If $q_j \in IQS_{q_i,p}$, based on Definition 7, Inequality 11 holds. This leads to the conclusion that $\text{EMD}(q_j, p) > \theta_j$ which indicates the disqualification of record p for query q_j and proves the correctness of Theorem 1. □

With the theorem given above, if q_j is in $IQS_{q_i,p}$, we can conclude in advance that p is not a query result for q_i. This will undoubtedly avoid the potential processing for any query in $IQS_{q_i,p}$. Moreover, for the purpose of guaranteeing the filtering performance of Inequality (11), we need to make sure the value of $\text{EMD}(q_i, q_j) - \text{EMD}(q_i, p)$ is at least greater than 0, or else the inequality defined above has no pruning effect. Our solution is that we maintain a set $S_i = \{q_j | j > i\}$ for each query q_i, where each query in S_i is sorted based on their EMD with respect to q_i. Based on this order, after an EMD refinement between q_i and p, we can quickly obtain a subset of S_i with an EMD value larger than $\text{EMD}(q_i, q_j) - \text{EMD}(q_i, p)$ using a binary search. After that, the inferable query set can be found from that subset on the basis of Inequality (11).

These two mechanisms are described in Algorithm 2. In Algorithm 2, we maintain a set S_i for each query q_i. All elements in S_i are the following queries of q_i and we order them by their EMD values to q_i (lines 3-4). Given an arriving probabilistic record p, we first check whether there are any new feasible solutions generated in the previous iteration (line 6). If there are, we calculate the IPS and DPS values for all feasible solutions based on statistical information captured in the indicator matrix, which was updated during the processing of the previous record (line 7). Then, we perform a two phases ranking for all feasible solutions based on their corresponding IPS and DPS values as described before (lines 8-9). After that, we replace the least effective solution with a new solution randomly chosen from the new solution list (line 10). Within lines 14-17, we verify p's qualification to each registered query using L feasible solutions and update the indicator matrix at the same time. If all feasible solutions accept p as a candidate for query q_i, we further check the qualification of p using *R-EMD*, LB_{IM} and UB_p (lines 19-24). If we still cannot eliminate p, we need to do the exact EMD computation (line 26). Based on the calculated EMD value, we take the multi-query optimization into operation (lines 29-33). Moreover, we add the newly generated feasible solution obtained from the exact EMD calculation into the new solution list (line 34).

Algorithm 2 Continuous Similarity Query with Adaptive Feasible Solution Update and Multi-query Optimization (query set $Q = \{q_1, \cdots, q_N\}$, threshold set $\Theta = \{\theta_1, \cdots, \theta_N\}$, and feasible solution set $\Phi = \{\Phi_1, \cdots, \Phi_L\}$

```
1: initialize the empty result set array RS[N]
2: initialize the indicator matrix M_{L×N}
3: for each query q_i do
4:    maintain a set S_i={q_j|N ≥ j > i
      && EMD(q_i,q_j)≥ EMD(q_i,q_{j+1})}
5: for each arriving records p do
6:    if newSoluList.empty()==FALSE then
7:       calculate the IPS and DPS values for each Φ_l based
         on M_{L×N}
8:       rank all feasible solutions by their DPS values
9:       rank all feasible solutions that have the same DPS
         value by their IPS values
10:      update the Φ_l with the lowest rank to a new solution
         randomly chosen from newSoluList
11:      newSoluList.clear()
12:   reset M_{L×N} /* reset every element in M_{L×N} to 0 */
13:   candidateQuery.clear()
14:   for each query q_i do
15:      if p cannot be filtered by all Φ_l then
16:         candidateQuery.add(q_i)
17:      M_{L×N}.update()
18:   for each query q_i ∈ candidateQuery do
19:      if R-EMD.filter(θ) then
20:         candidateQuery.delete(q_i)
21:      else if LB_IM.filter(θ) then
22:         candidateQuery.delete(q_i)
23:      else if UB_p.filter(θ_i) then
24:         RS[i].add(p)
25:      else
26:         calculate EMD(q_i,p)
27:         if EMD(q_i,p)≤ θ_i then
28:            RS[i].add(p)
29:         for each q_j ∈ S_i with
            EMD(q_i,q_j)>EMD(q_i,p) do
30:            if candidateQuery.exist(q_j) then
31:               lb =EMD(q_i,q_j)-EMD(q_i,p)
32:               if lb > θ_j then
33:                  candidateQuery.delete(q_j)
34:         newSoluList.add(Φ_new)
35: return {RS[N]}
```

6 Empirical Studies

In this section, we will first introduce the experimental setup (Section 6.1). Secondly, we present the experimental results for one-shot queries, including the range query (Section 6.2.1) and the k-nearest neighbor query (Section 6.2.2). After that, a transaction throughput test based on these two types of one-shot queries is discussed in Section 6.2.3. Finally, in Section 6.3, we evaluate the experimental results for continuous similarity queries. For more information about our system implementation, please access our homepage [7] to download the *Dll File* and *User Manual*.

6.1 Experimental Setup

In this section, we introduce the experimental setup, including the data preparation, verification methods, parameter settings and experimental environment. We

[7] `http://faculty.neu.edu.cn/ise/xujia/home/TBI-Introduction.html`

Table 4 Experimental parameters. Default value is typeset in Bold

Parameters	Varying Range
Search range RETINA1-θ	0.3,0.35,**0.4**,0.45,0.5
Search range IRMA-θ	0.3,0.4,**0.5**,0.6,0.7
Search range DBLP-θ	0.1,0.15,**0.2**,0.25,0.3
Search range MTV/Movie-θ	3,6,**9**,12,15
k for k-NN query	2,4,8,**16**,32,64
B^+ tree number	1,2,3,**4**,5
Method for choosing feasible solutions	**Random-sampling-based**,Clustering-based
Query number in continuous query	5,10,**20**,40,80
Reduced dimension in continuous query	32,64,**96**,128,160
Ground distance	**Euclidean**, Manhattan
Cardinality of DBLP data set	50,100,150,200,**250** ($\times 10^3$)

begin with describing the five physical-world data sets used in our experiments.

We evaluate the one-shot query algorithms using the following three real data sets, where RETINA1 and IRMA data sets are also used by [38].

RETINA1: This is an image data set consisting of 3,932 feline retina scans labeled with various antibodies. For each image, twelve 96-dimensional histograms are extracted. Each bin of a histogram has a 2-dimensional *Feature Vector*. Each feature vector represents the bin's location based on which we calculate the ground distance.

IRMA: This data set contains 10,000 radiography images from the Image Retrieval in Medical Application (IRMA) project [2]. For each image, a 199-dimensional histogram is extracted based on the information of 199 patches collected at both salient points and on a uniform grid. There is a 40-dimensional feature vector defined for each bin, which represents the 40 principal component coefficients for each patch. For details on the feature extraction process, please refer to [16]. With the largest histogram dimensionality and feature vector dimensionality, IRMA is the most time-consuming data set for individual EMD calculations.

DBLP: This is an 8-dimensional histogram data set with 250,100 records which was generated from the DBLP database in October 2007. As in Example 2 of Section 1, the 8 dimensions represent 8 different research domains in computer science, namely *AI, Application, Bioinformatics, Database, Hardware, Software, System and Theory*. We define the feature vector for each bin/research domain, considering its correlation to the following three aspects: *Computer*, *Mathematics* and *Architecture*. Thus, there is a 3-dimensional feature vector corresponding to each histogram bin. For further details on the DBLP data set, please refer to [41].

During the process of evaluating one-shot queries, each complete data set is divided into a query data set containing 100 query histograms and the remaining data form the database to be queried. As a result,

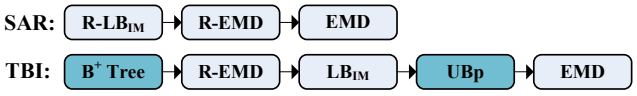


Fig. 8 Default filter chain settings (We mark the new filters proposed in this paper in blue)

the cardinalities of the RETINA1, IRMA and DBLP databases are 3,832, 9,900 and 250,000 respectively. We refer to our approach for one-shot queries as TBI (Tree-Based Indexing). TBI-R is used to represent the TBI employed with the feasible solutions produced by the Random-sampling approach and TBI-C connotes the TBI with the feasible solution generated by Clustering-based method, as described in Section 3.2. We use the average experimental results based on three randomly generated feasible solutions to evaluate the performance of TBI-R. The filter train employed in TBI-R and TBI-C follows the default setting as shown in Figure 8, where *R-EMD*[38] denotes a lower bound filter of EMD utilizing the properties of EMD in the *Reduced* space and LB_{IM} is a *Independent Minimization* lower bound filter described in [6]. The same sequence of filters, but now skipping the UB_P is used for the one-shot k-NN query experiments. We compare the TBI-R and TBI-C, with the *Scan-And-Refine* (SAR) algorithm as proposed in [38]. The setup of the filter chain in SAR follows the description in Figure 8. To the best of our knowledge, SAR is the most efficient exact EMD-based similarity search algorithm over high-dimensional histograms. The dimension reduction matrices used in both TBI and SAR are chosen to be the most efficient ones according to the experimental results in [38]. Specifically speaking, for the RETINA1 and IRMA data sets, we use an 18- and 60-dimensional reduction matrices respectively, generated using the *FB-ALL-KMed* method [38]. For the DBLP data set, the filters that start with a letter R, namely *R-EMD* and $R\text{-}LB_{IM}$, are removed from all filter chains, since the histogram dimensionality in DBLP is 8, which is already quite low.

The following two data sets are derived to evaluate the continuous similarity query algorithms. In order to obtain histograms satisfying the data continuity, we use two videos, i.e., a Music Video Clip and a Movie, as the sources of query data sets. For each video stream, we continuously sample frames at a frequency of 4 frames per second. After that, a 256-dimensional grey level histogram is extracted from each sampled frame. The feature vector of each bin is set as a value which is increased with the index number of each bin. More information for Music Video Clip and Movie data sets is given below.

Music Video Clip: This data set contains 1,031 grey level histograms extracted from a music video clip of the 2008 Olympics theme song *You and Me*. Its total duration is 4 minutes and 30 seconds.

Movie: This data set contains 22,000 grey level histograms extracted from a movie named *Les aventures extraordinaires d'Adéle Blanc-Sec* with a total duration of 1 hour and 53 minutes.

We randomly sample a number of frames from both the Music Video Clip and Movie data sets to form the registered query set. For convenience, we named our basic method of handling continuous similarity queries as FSBP (Feasible Solution-Based Pruning) which denotes the feasible solution-based pruning strategy without the consideration of the adaptive feasible solution update and multi-query optimization as mentioned in Section 5. FSBP-Adap on the other hand indicates the addition of the adaptive feasible solution update module to the original FSBP. Moreover, FSBP-Adap-Opt is the same as FSBP-Adap, but now including the multi-query optimization module. The filter chain used in all FSBP-based approaches is similar to the pruning sequence of one-shot range queries, except that no B^+ tree index is built, and instead we use a group of feasible solutions to rule out unpromising queries. We compare our three FSBP-based approaches with SAR-Cont, which is an implantation of SAR in the context of continuous query processing. Instead of using the FB-All-Kmed [38] for the generation of data-reduction matrices as used in the experiments for the one-shot-queries, we will be using the KMedoid [38] method in the continuous query processing experiments. Although the matrix generated by the first method offers tighter lower bounds for the *R-EMD* filter, we cannot use it under the context of continuous query processing. For one thing, data for continuous queries are arriving in real-time, and thus it will not be possible to derive data distribution in advance. Additionally, creating a dimensionality reduction matrix for high-dimensional data is time consuming[8], which implies that we cannot produce reduction matrices under realistic time constraints.

For both of the evaluation of one-shot queries as well as continuous similarity queries, we summarize their basic parameter settings in Table 4. The default value for each parameter is highlighted in bold.

All programs are compiled using Microsoft VS 2005 under Windows XP and run on a PC with an Intel Core2 2.4GHz CPU, 2GB RAM and 150GB hard disk.

6.2 Evaluation of One-shot Queries

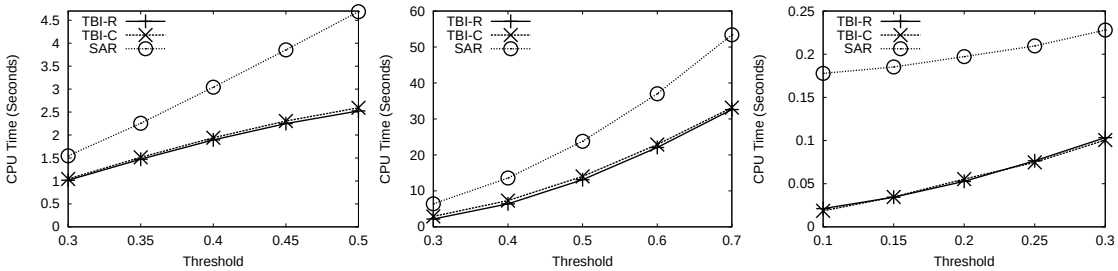
The reported results on range or k-NN query are the averages over a workload of 100 queries. To verify the efficiency of our algorithm, we measure the *Querying CPU Time*, the *Number of EMD Refinements* and the *Querying I/O cost* in Sections 6.2.1 and 6.2.2. To evaluate the concurrency efficiency of our framework, we test the *Transaction Processing Throughput* in Section 6.2.3.

6.2.1 Evaluation of Range Queries

Figures 9 and 10 display the impact of the similarity search threshold on the querying CPU time and the number of exact EMD refinements required for each query. Figure 9 illustrates that both TBI-R and TBI-C beat SAR where the time requirements of SAR can be up to 3-6 times larger than that of TBI on the DBLP data set. This is because the number of EMD refinements in TBI is greatly reduced, especially on the DBLP data set (see Figure 10). As discussed in the previous sections, calculation of the exact EMD is very expensive, and as such, the number of exact EMD calculations is an important factor affecting the efficiency of the EMD-based query processing. The reduction of the number of EMD refinements in TBI implies that our B^+ tree-based filtering technique and UB_P filtering method are better suited to candidate pruning for range queries. Moreover, TBI-R performs better than TBI-C on all data sets.

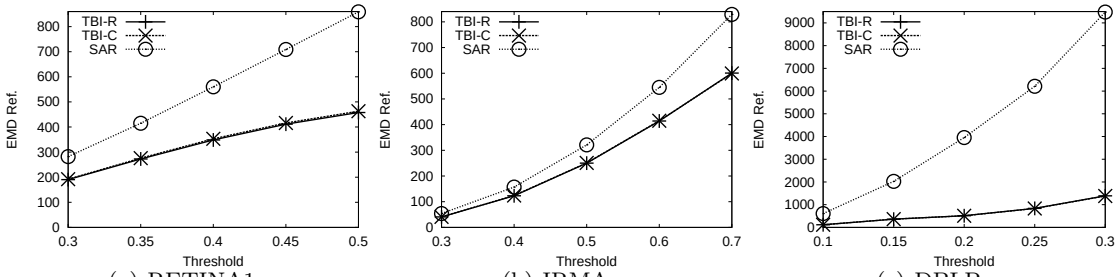
In Figure 11, we show the average CPU time per query as we vary the number of B^+ trees in our indexing structure. On RETINA1, the CPU time requirements for both TBI-R and TBI-C slowly decreases and there is no apparent difference between the two. A difference between TBI-R and TBI-C, however is obvious on the IRMA data set while TBI-C lags behind TBI-R in most

[8] Based on the authors' description of literature [38], for the IRMA data set with dimensionality 199, creating a 115 dimensional reduction matrix will take many hours. This will undoubtedly be higher for Music Video Clip and Movie data set which both have a dimensionality of 256



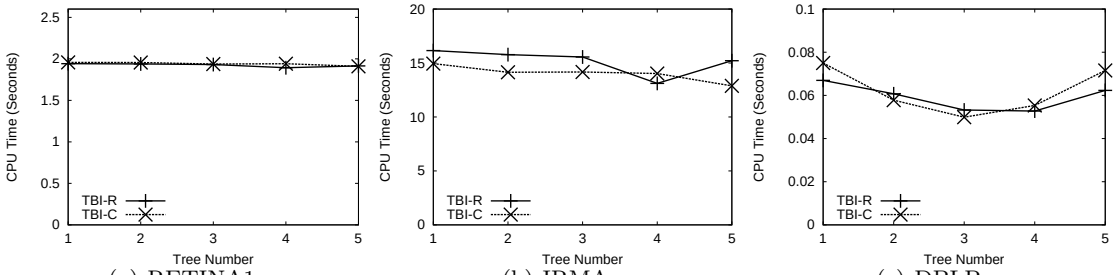
(a) RETINA1 (b) IRMA (c) DBLP

Fig. 9 Effect of range threshold on the average querying CPU time per range query



(a) RETINA1 (b) IRMA (c) DBLP

Fig. 10 Effect of range threshold on the average EMD refinement number per range query



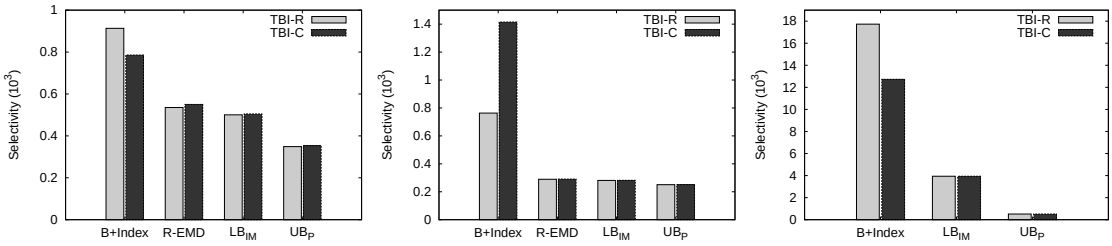
(a) RETINA1 (b) IRMA (c) DBLP

Fig. 11 Effect of the number of B^+ trees on average query CPU time per range query

of the settings. This phenomenon can be explained as the high dimensionality of IRMA leads to worse clustering results used in the construction of the B^+ indexing trees. On the DBLP data set, which has the largest cardinality ($cardinality = 250,000$) but the smallest histogram dimensionality ($dimensionality = 8$), the query efficiency gradually deteriorates when more than 3 or 4 trees are employed. This phenomenon is also visible for TBI-R on IRMA. The reason for the performance deterioration is that the pruning ability is strong enough with a certain number of B^+ trees. The addition of more trees only induces higher searching times but is unhelpful in reducing the number of candidates.

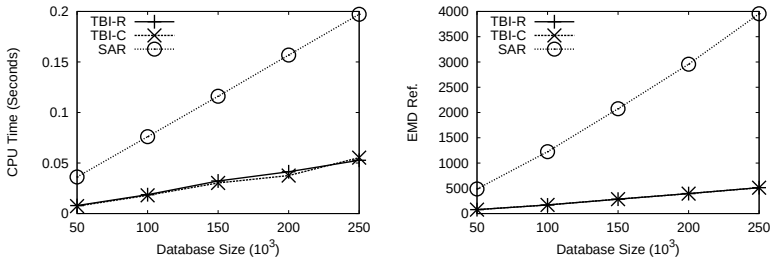
Figure 12 summarizes the experimental results on the effectiveness of different filters used in our query processing algorithm. The effectiveness of each filter is evaluated by its *Selectivity* which indicates the average number of records passing a specific filter per query. We observe that filters equipped after the B^+ tree filter remain valuable for candidate pruning. Recall that SAR also employs the *R-EMD* filter but is less efficient than TBI methods, thus it can be concluded that our B^+ tree index and UB_P filter do provide an effective and efficient additional pruning power. Another observation that can be made from the figure is on the effectiveness of UB_P, where it can be seen that UB_P is more effective for lower dimensional data, especially for the DBLP data set, as compared to higher dimensional data sets, such as RETINA1 and IRMA.

In Figure 13, we show the impact of database cardinality on CPU time and the number of EMD refinements. We can see that without the assistance of a B^+ tree index and UB_P filters, SAR suffers a quick linear increase on both CPU time as well as the number



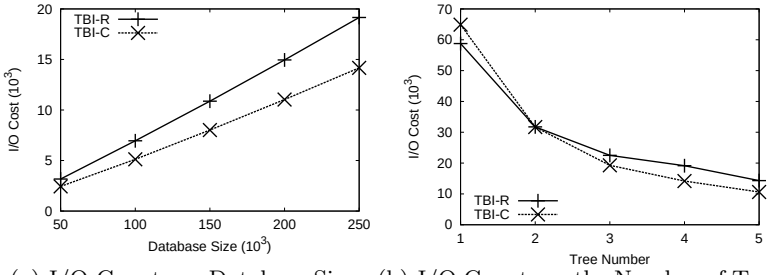
(a) RETINA1 (b) IRMA (c) DBLP

Fig. 12 Effect of filters on average selectivity per range query



(a) CPU Time vs Database Size (b) EMD Refinements vs Database Size

Fig. 13 Effect of database size on range queries



(a) I/O Counts vs Database Size (b) I/O Counts vs the Number of Trees

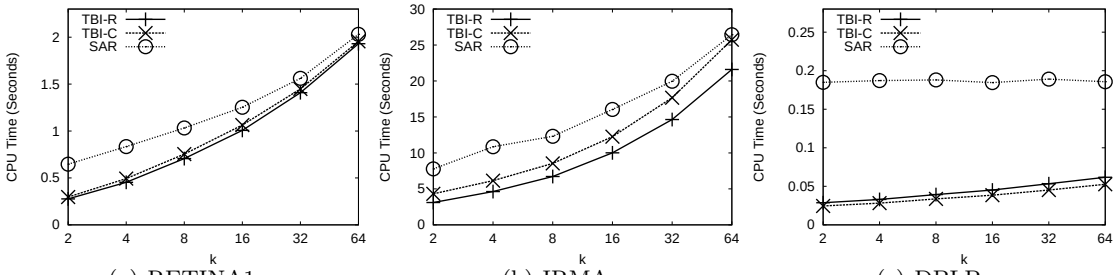
Fig. 14 Average I/O costs for range queries

of EMD refinements, while our TBI methods exhibit a much slower growth. This explicates that the pruning effects of the B^+ tree and UB_P filters remain significant even when the data cardinality is as large as 250,000.

The results of I/O cost are depicted in Figure 14. The I/O cost is evaluated by counting the average number of I/O accesses per query. In Figure 14(a), we vary the database cardinality and observe that the I/O cost increases linearly with respect to the size of the database. That is because, with the increase of the cardinality, we need to visit more records within a certain search range. When we alter the number of B^+ trees, the I/O cost drops evidently from one B^+ tree to two B^+ trees and then declines slowly from two trees to five trees. The reason is that installing more than two B^+ trees can not significantly shrink the size of the candidate set, and thus the I/O decline becomes less notable.

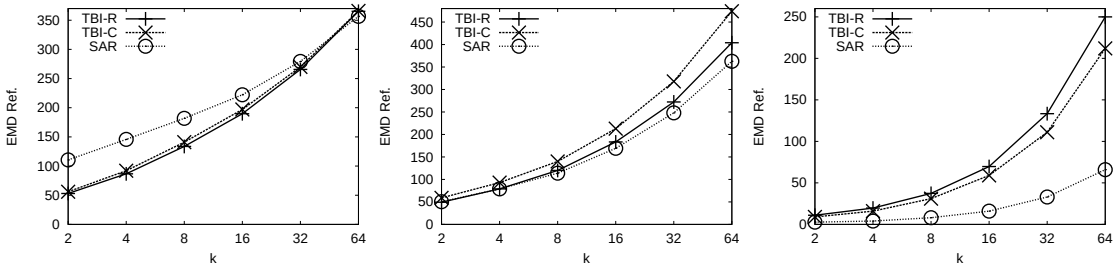
6.2.2 Evaluation of k-Nearest Neighbor Queries

In this section, we experimentally evaluate the performance of our algorithms on k-NN queries. Figure 15 summarizes the average CPU time required for k-NN queries over different data sets. Our TBI approaches are consistently faster than SAR on all data sets which is even more obvious for larger data sets. To explain why SAR incurs a smaller number of EMD refinements than the TBI approaches as shown in Figure 16, one should remember its processing steps as described in Section 7. By utilizing the optimal multi-step retrieval framework for k-NN queries, known as *KNOP* [32], SAR obtains a good query-based data ranking. Query-based data ranking is quite helpful for pruning the unpromising records in k-NN queries, thus significantly cutting down the number of required EMD refinements. However, the time cost required for the process of rank-



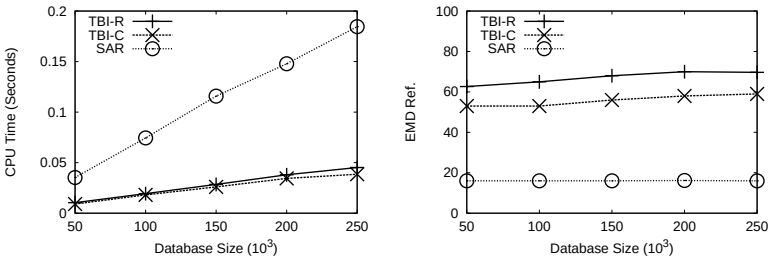
(a) RETINA1 (b) IRMA (c) DBLP

Fig. 15 Effect of k on the average query CPU time per k-NN query



(a) RETINA1 (b) IRMA (c) DBLP

Fig. 16 Effect of k on the average number of EMD refinements per k-NN query



(a) CPU Time vs Database Size (b) EMD Refinements vs Database Size

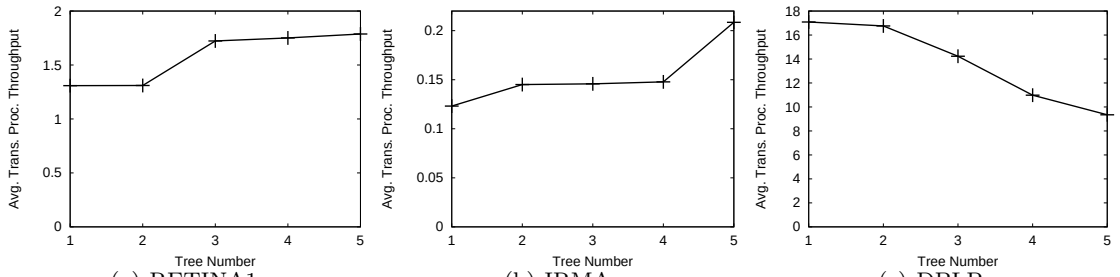
Fig. 17 Effect of database size for k-NN queries

ing becomes a bottleneck when the data cardinality is large (e.g., DBLP data set) or the computational cost of the ranking distance function is high (e.g., on IRMA data set, the ranking distance function is the EMD over 60-dimensional reduced histograms with the ground distances calculated by 40-dimensional feature vectors). That is why TBI still outperforms SAR although SAR requires a lower number of EMD refinements on IRMA and DBLP data sets. On RETINA1, on the other hand, the query-based data ranking is derived from a reduced 18-dimensional data space, which makes the obtained data ranking deviate from the true ranking in the original data space, with 96 dimensions. Therefore, SAR requires a large number of EMD refinements on RETINA1 which naturally degrades its query efficiency.

Figure 17 shows the results for the average CPU time and the number of EMD refinements by varying the data size of the DBLP data set. The results are as expected: since the ranking order of records in SAR is excellent, its number of EMD refinements approaches the optimal value 16 in a 16-NN query. However, the great ranking costs cause the SAR to exhibit poor CPU time.

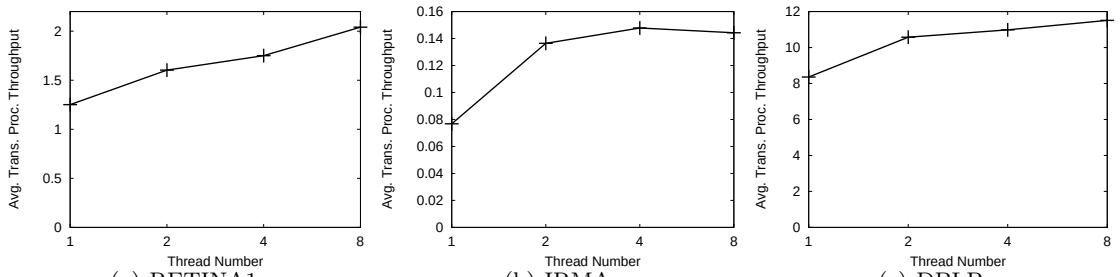
6.2.3 Evaluation of Transaction Concurrency

One of the important benefits of our scheme is the deployment of B^+ trees for concurrency control, which generally improves the performance of a system when different users access the database concurrently. To test the concurrency performance of our algorithm, we generated 150 database transactions by randomly selecting existing data records from the static data set. Several



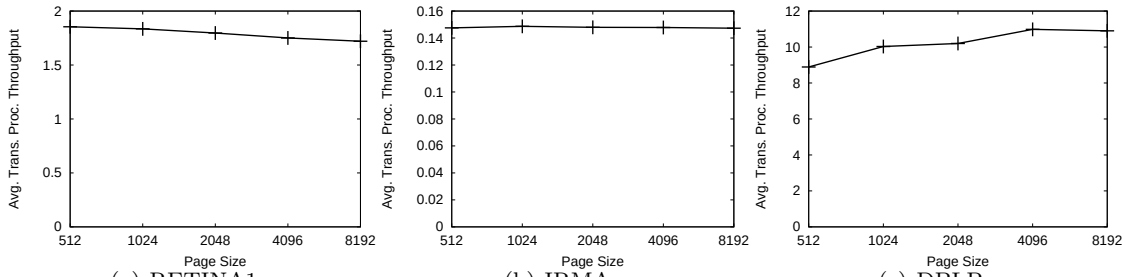
(a) RETINA1 (b) IRMA (c) DBLP

Fig. 18 Effect of number of trees on the transaction processing throughput



(a) RETINA1 (b) IRMA (c) DBLP

Fig. 19 Effect of number of threads on the transaction processing throughput



(a) RETINA1 (b) IRMA (c) DBLP

Fig. 20 Effect of page size on transaction processing throughput

worker threads are created to handle the transactions concurrently. Every worker thread runs our TBI-R algorithm and fetches one transaction from the workload pool at one time. The workload pool contains 150 transactions, including 50 insertions, 50 deletions, 25 range queries and 25 k-NN queries. Three groups of experiments are conducted to test the change of average transaction throughput, i.e., the average number of transactions processed per second, by varying the number of B^+ tree, the number of worker threads and the page size of the index structure respectively. Note that we employ the B^+ tree implementation with the guarantee of *Read-Committed* isolation from BerkeleyDB[9] as the basic index structure in this group of experiments.

The results on varying the number of trees are presented in Figure 18. It can be observed that the transaction processing throughput is improved on both the RETINA1 and IRMA data sets when more trees are created. On the other hand, the results on DBLP data set show an opposite trend. This is because DBLP has a fairly low dimensionality, leading to a small calculation time for exact EMD refinement. Under these circumstances, the time saved by adding trees is less than the time consumed for accessing multiple trees. Therefore, the CPU time increases with the addition of each B^+ tree and the overall throughput declines. Conversely, the high dimensionality of histograms on RETINA1 and IRMA data sets, requires more expensive EMD computations and thus it becomes advantageous to add more trees.

[9] `http://www.oracle.com/technology/products/berkeley-db/index.html`

Table 5 Average deadlock rate (varying the ratio of k-NN queries)

Dataset \ k-NN ratio	0.0	0.25	0.5	0.75	1.0
RETINA1	0%	0.67%	1.14%	0.76%	0%
IRMA	0%	0%	0.53%	0%	0%
DBLP	0%	0%	0%	0%	0%

Table 6 Average deadlock rate (varying the page size of database)

Dataset \ Page size	512	1024	2048	4096	8192
RETINA1	0%	0.53%	1.14%	1.73%	3.73%
IRMA	0.01%	0%	0.57%	0.48%	0.76%
DBLP	0%	0%	0%	0%	0%

From Figure 19, we can see that the throughput gradually climbs with each increase in the number of worker threads on the RETINA1 and DBLP data sets. The reason behind this increase is obvious. On the IRMA data set however, the throughput declines between 4 and 8 threads. This phenomenon can be explained as follows. Due to the high dimensionality of the IRMA data, it takes more time for each individual worker thread to finish the transaction. This leads to a higher probability of the occurrence of deadlocks between the threads, since each thread tends to lock the tree nodes with a longer locking time. Thus, several transactions are aborted and conducted from the beginning again when deadlock is detected, thus reducing the throughput of our algorithm on the IRMA data set.

Figure 20 summarizes the experimental results for system throughput by changing the page size in the B^+ trees. It is interesting to see that the three different data sets show different trends. The underlying reason for the phenomenon is the different cardinalities of the data sets. For a smaller data set, such as the RETINA1, there will be less pages in the B^+ trees with an increase in the page size. It is thus more likely for different worker threads to request the same page. Such locking conflicts are resolved by giving the authority to only one of the threads while suspending the others until the lock is released. This mechanism greatly affects the throughput especially when the database cardinality is not large. As to DBLP data set, due to its large cardinality, locking conflicts are less likely to occur. Combined with the reduction in I/O time requirements with an increasing page size, the system achieves better concurrency when larger page sizes are used. The performance of IRMA displays a trend between that of RETINA1 and DBLP, for its medium data cardinality.

The deadlock rate is an important factor influencing the concurrency of a DB application. Based on our discussion in Section 4.3, the k-NN query visits data pages in different directions and thus is impossible to avoid deadlocks when write operations exist. Next, we investigate the impact of the k-NN query on the deadlock occurrence. We mix insertion operations and k-NN queries to form a workload of 150 transactions. Ten worker threads are created to handle the workload concurrently. All tested deadlock rates are summarized in Table 5 and Table 6, with each rate representing the average over five random testings.

In Table 5, we fix the page size and alter the ratio of k-NN queries in the workload. It can be observed that on both REATINA1 and IRMA data sets, the deadlock rate reaches its maximum when k-NN queries occupy half of the workload. This shows that, although increasing the k-NN ratio leads to more read operations from reverse directions, the number of write operations, namely the insertions here, also plays a vital part in the occurrence of deadlocks. There is no deadlock found on DBLP data set. It is because DBLP has a large data cardinality and thus owns more pages, making it more difficult for two operations to meet on the same page.

In Table 6, we change the page size and show the change of deadlock rate. The total trend is that the deadlock rate increases with the page size. That is because, as the growth of page size, the number of pages in a B^+ tree diminishes. Consequently, there are fewer pages that can be locked, which potentially increases the deadlock rate. We are glad to see that DBLP escapes from lock contention even when the page size is as large as 8,192 due to its large cardinality (250,000 tuples). This shows that our concurrency protocol for k-NN queries has advantage on large data sets.

6.3 Evaluation of Continuous Similarity Queries

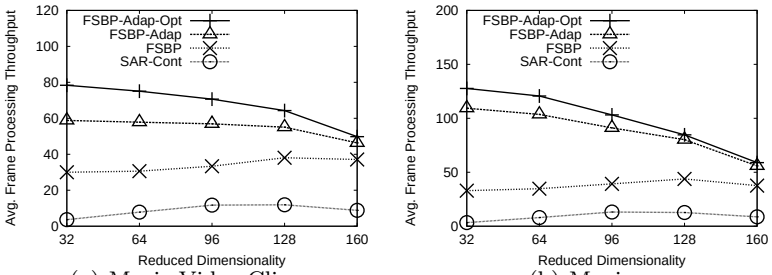
Each reported result in this section is an average over a workload of 20 continuous similarity queries. First, we evaluate the usability of our FSBP-based algorithms, namely FSBP, FSBP-Adap and FSBP-Adap-Opt, under the scenario of continuous query processing and pro-

(a) Query (b) R1 (c) R2 (d) R3 (e) R4 (f) R5

Fig. 21 Illustration of query results on Music Video Clip

(a) Query (b) R1 (c) R2 (d) R3 (e) R4 (f) R5

Fig. 22 Illustration of query results on Movie



(a) Music Video Clip (b) Movie

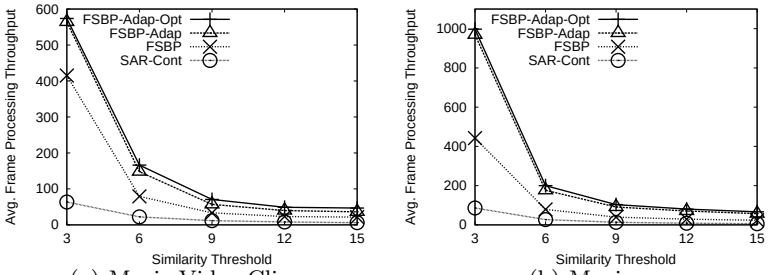
Fig. 23 Effect of reduced dimensionality on average frame processing throughput

vide screen shots of several query results. After that, we verify the efficiency of our FSBP-based algorithms by measuring the average *Frame Processing Throughput*, i.e., the average number of frames processed per second.

We use the default similarity threshold as given in Table 4, to initiate continuous queries on both the Music Video Clip and Movie data sets. By setting the a screen shot in Music Video Clip (see Figure 21(a)) as the query, Figure 21(a) to 21(f) display its partial result set. Amongst the result set, Figure 21(d) is the adjacent previous scene of the query frame while Figure 21(e) and 21(f) are its adjacent following scenes. Although Figure 21(b) and 21(c) display a different content, their grey-level color distributions are very similar to that of the query frame. This kind of *false-positive* does exist for we use the grey level histogram as the fingerprint for each frame, and can be alleviated by using a hybrid histogram combining both of the grey scale and texture information in the frame.

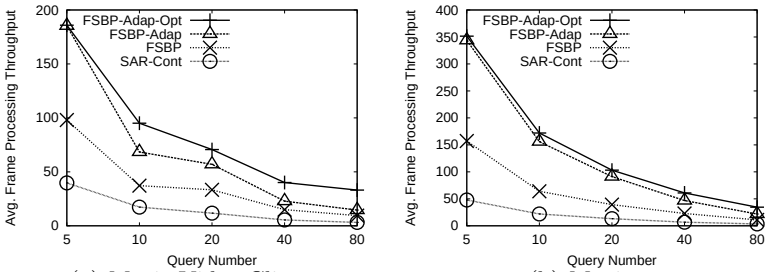
The Movie data set's query frame is shown in Figure 22(a), together with its query results, from Figure 22(a) to Figure 22(f). Besides the query frame, the system successfully returns similar frames. The result set is composed of frames with earlier as well as later time stamps, shown in 22(d)-22(f). The other two frames come from a different scene, however, their contents is very similar to the query, i.e, a broad stretch of sky and a light-colored desert.

Figure 23 illustrates the trade-off between the reduced dimensionality and the frame processing throughput. The general trend on both data sets is that the throughput increases slowly at first with decreasing data dimensionality, and then has a slight decrease after a certain reduced dimensionality. Theoretically speaking, a higher reduced dimensionality produces a tighter lower bound for the *R-EMD* filter and thus can prune more records. However, increasing the reduced dimensionality also incurs the growth of computational complexity of the *R-EMD* filter and therefore decreases the throughput. Besides the trend, it is easy to observe that our three FSBP-based methods outperform the SAR-Cont approach, with respect to the frame processing throughput. Moreover, the inclusion of the two optimization modules, namely the adaptive feasible solu-



(a) Music Video Clip (b) Movie

Fig. 24 Effect of similarity threshold on average frame processing throughput



(a) Music Video Clip (b) Movie

Fig. 25 Effect of the number of queries on the average frame processing throughput

tion update and the multi-query optimization, further increases the system performance.

We summarize the impact of different similarity thresholds on the throughput in Figure 24. As the threshold increases, more candidates need to be verified and thus the throughput will decline. However, our FSBP-based algorithms display superiority compared to the SAR-Cont technique. Furthermore, the participation of the two optimization techniques described in this paper indeed increases the system throughput.

Figure 25 shows the influence of the number of queries on the throughput. As we vary the number of the queries, our FSBP-based methods always outperform SAR-Cont. When the number of the concurrent queries reaches as large as eighty, three FSBP-based approaches can still guarantee a throughput larger than 4 frames per second which is the frame extracting rate on both data sets. When the number of queries equals to eighty, the throughput of SAR-Cont drops to 3.13 frames per second on the Music Video Clip data set and 3.42 frames per second on the Movie data set. This means that when frames arrive at a speed of 4 frames per second, SAR-Cont is unable to provide a timely service to applications. Another interesting observation is that the FSBP-Adap outperforms FSBP-Adap-Opt on the Music Video Clip data set when the number of queries equals to 5. This is because, a small query number indicates a low probability for multi-query optimization. Therefore, instead of apparently improving the performance of the system, the participation of the multi-query optimization actually downgrades system efficiency.

One more interesting conclusion from Figures 23–25 is that the throughput of the Movie data set is obviously larger than that of the Music Video Clip data set for FSBP-based methods. This can be explained by the fact that compared to the Music Video Clip data set, the correlation between two neighboring images in the Movie data set is much stronger and thus the effect of using adaptive feasible solution update is strengthened.

Recently, by employing our approach of processing EMD-based continuous queries, we have built a demonstration system[10], to demonstrate the online video frame copy detection.

7 Related Work

In this section, we provide a literature review of existing research results relating to our problem.

Recent years have witnessed the numerous advances of probabilistic data management, especially in techniques for efficient and effective query processing. Two types of queries have been intensively investigated, namely *Top-k query* and *Accumulated Probability Query*[11].

Definition 8 Top-k Query
Given a d-dimensional vector $p_i = (p_i[1], p_i[2], \ldots, p_i[d])$ and a weight vector $W = (w[1], w[2], \ldots, w[d])$, the

[10] `http://faculty.neu.edu.cn/ise/xujia/home/EUDEMON-Introduction.html`

[11] While it is called a *Range Query* in some papers, here we prefer the term *Accumulated Probability Query* to distinguish it from our range query definition w.r.t. EMD.

weighted aggregation of p_i with respect to W is defined as $\sum_j p_i[j]w[j]$. The Top-k query on the probabilistic database $D = \{p_1, p_2, \ldots, p_n\}$ where p_i is a probabilistic record, retrieves k records from the database with the maximal weighted aggregations.

While the definition on top-k query is clear when all records consist of only exact values on all attributes, it is challenging to extend it to probabilistic databases. If a record p_i is associated with different attribute values with probabilities, the weighted aggregation of p_i becomes some probability distribution. It is then unclear how to rank the probabilistic records to return the top-k records to the user. To overcome such difficulties, different solutions have been proposed to complete the semantics of top-k query in the probabilistic domain, including Uncertain Top-k [34], Uncertain Rank-k [34], Probabilistic Threshold Top-k [21], Expected Rank-k [13], PRF^{ω} and PRF^{e} [23].

Definition 9 Accumulated Probability Query Given the distribution records from the probabilistic database $D = \{p_1, p_2, \ldots, p_n\}$, an accumulated probability query with range R and threshold θ will return all distributions appearing in R with probability larger than θ, i.e., $\{p_i \in D \mid \Pr(p_i \in R) \geq \theta\}$.

The problem of range query and k-nearest neighbor query based on EMD however is different from the query types mentioned above. First, our similarity queries aim to discover similar distributions. Secondly, our similarity queries cannot be directly formulated by any simple ranking scheme as is done in top-k query.

Due to the linear programming nature of the Earth Mover's Distance, the metric is computationally rather expensive. When first proposed in [29], Rubner et al. showed that exact EMD could be evaluated with an existing algorithm designed for the *Transportation Problem*. The complexity of the algorithm is cubic with respect to the number of bins in the histograms. This has become a major efficiency bottleneck for any application employing EMD as the underlying metric.

Some attempts have been made to accelerate the computation of exact EMD. In [25], Ling and Okada investigated a special case of EMD, which used Manhattan distance as the ground distance. They modified the original *Simplex Algorithm* [26] to exploit the nature of the Manhattan distance. Although there is no theoretical proof of the acceleration effects, their empirical studies imply that their algorithm takes quadratic time with respect to the number of bins.

From an algorithmic point of view, Andoni et al. [5] devised an embedding method for EMD based on a random projection technique. After the embedding, queries on EMD can be answered in a new space using the Hamming distance. Unfortunately, their method only guarantees $O(\log n \log d)$-distortion, i.e., the ratio between original EMD and new Hamming distance is no larger than $O(\log n \log d)$ with high probability. In fact, this scheme is only of theoretical interest. For practical applications, the data cardinality can be arbitrarily large which may weaken its accuracy. Moreover, this embedding method only works on static data sets, and as such is not friendly to dynamic updates. Shirdhonkar and Jacobs [33], in another attempt, proposed a new distance operator to approximate EMD. They applied wavelet decomposition on the dual program of EMD and eliminated the parameters on small waves. The new distance can be efficiently calculated in linear time with respect to the number of bins in the histograms. However, their method does not directly support indexing structures for query processing on large data sets.

Existing solutions to the indexing problem for EMD-based queries mainly rely on the framework of *Scan-and-Refinement* [6,38]. In this framework, a linear scan on the complete data records results in a set of candidates to the query results based on efficient lower bound filters for EMD. In the pre-processing step, dimensionality reduction is conducted. In the scan phase, all reduced records are verified with two filters, namely the LB_{IM}(Appendix B [1]) and the EMD computation which are both performed in the reduced space. Given a range query, a final verification phase returns the complete query result by verifying the distances of all candidates not pruned by the previous filters. For k-nearest neighbor queries, the algorithm follows the optimal multi-step retrieval framework, known as KNOP [32], to guarantee no more candidates are produced in each filter step. First, all records are sorted based on a theoretical lower bound of the EMD. Another sequence of random accesses are conducted based on the ranking of the records, until the top-k threshold is smaller than the lower bound of next record in the sequence. The major drawback of KNOP with respect to EMD, is the high I/O cost incurred by the sorting operation.

8 Conclusion

In this paper, we present a new indexing scheme to support similarity search queries on histogram-representative probabilistic databases, based on Earth Mover's Distance (EMD). Our indexing method relies on the primal-dual theory to construct a mapping from the original probabilistic space to a one-dimensional domain. Each mapping domain is indexed using the B^{+} tree structure. We show that our efficient query processing algorithms can answer both range query and k-nearest

neighbor query. Our index structure is also shown to be friendly to concurrency protocols and is easily extensible to handle continuous queries on data streams. Our experiments prove that our proposals generally outperform the state-of-the-art methods.

References

1. J. Xu. et al. Appendix Section. `http://vldb.org/vldb_journal`. `http://faculty.neu.edu.cn/ise/xujia/home/appendix.pdf`.
2. T. Lehmann et al. IRMA project site. `http://ganymed.imib.rwth-aachen.de/irma/`.
3. P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *PODS*, pages 137–146, 2009.
4. P. Agrawal, O. Benjelloun, A. D. Sarma, C. Hayworth, S. U. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.
5. A. Andoni, P. Indyk, and R. Krauthgamer. Earth mover distance over high-dimensional spaces. In *SODA*, pages 343–352, 2008.
6. I. Assent, A. Wenning, and T. Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *ICDE*, page 11, 2006.
7. S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–120, 2001.
8. O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.
9. O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.
10. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
11. R. Cheng, S. Singh, and S. Prabhakar. U-dbms: A database system for managing constantly-evolving data. In *VLDB*, pages 1271–1274, 2005.
12. D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate data collection in sensor networks using probabilistic models. In *ICDE*, page 48, 2006.
13. G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.
14. N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.
15. N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.
16. T. Deselaers, D. Keysers, and H. Ney. Discriminative training for object recognition using image patches. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2:157–162, 2005.
17. A. Deshpande, C. Guestrin, and S. Madden. Using probabilistic models for data management in acquisitional environments. In *CIDR*, pages 317–328, 2005.
18. A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-based approximate querying in sensor networks. *VLDB J.*, 14(4):417–443, 2005.
19. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.
20. K. Grauman and T. Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR*, pages 220–227, 2004.
21. M. Hua, J. Pei, W. Zhang, and X. Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD Conference*, pages 673–686, 2008.
22. L. V. S. Lakshmanan, N. Leone, R. B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.
23. J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.
24. N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *ICDE*, pages 106–115, 2007.
25. H. Ling and K. Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(5):840–853, 2007.
26. C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*, pages 67–71. Dover Publications, 1998.
27. C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.
28. Y. Rubner, J. Puzicha, C. Tomasi, and J. M. Buhmann. Empirical evaluation of dissimilarity measures for color and texture. *Computer Vision and Image Understanding*, 84(1):25–43, 2001.
29. Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.
30. Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision*, 40(2):99–121, 2000.
31. R. Sandler and M. Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric. In *CVPR*, pages 1873–1880, 2009.
32. T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD Conference*, 1998.
33. S. Shirdhonkar and D. W. Jacobs. Approximate earth mover's distance in linear time. In *CVPR*, 2008.
34. M. A. Soliman, I. F. Ilyas, and K. C.-C. Chang. Probabilistic top-k and ranking-aggregate queries. *ACM Trans. Database Syst.*, 33(3), 2008.
35. Y. Tao, X. Xiao, and R. Cheng. Range search on multidimensional uncertain data. *ACM Trans. Database Syst.*, 32(3):15, 2007.
36. G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.
37. D. Z. Wang, M. J. Franklin, M. N. Garofalakis, and J. M. Hellerstein. Querying probabilistic information extraction. *PVLDB*, 3(1):1057–1067, 2010.
38. M. Wichterich, I. Assent, P. Kranen, and T. Seidl. Efficient emd-based similarity search in multimedia databases via flexible dimensionality reduction. In *SIGMOD Conference*, pages 199–212, 2008.
39. J. Xu, Z. Zhang, A. K. H. Tung, and G. Yu. Efficient and effective similarity search over probabilistic data based on earth mover's distance. *PVLDB*, 3(1):758–769, 2010.
40. M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava. On multi-column foreign key discovery. *PVLDB*, 3(1):805–814, 2010.
41. Z. Zhang, B. C. Ooi, S. Parthasarathy, and A. K. H. Tung. Similarity search on bregman divergence: Towards non-metric indexing. *PVLDB*, 2(1):13–24, 2009.