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

Jia Xu[1] Zhenjie Zhang[2] Anthony K.H. Tung[2] Ge Yu[1]

[1]College of Info. Sci. & Eng., Northeastern University, China, {xujia,yuge}@ise.neu.edu.cn
[2]School of Computing, National University of Singapore, {zhenjie,atung}@comp.nus.edu.sg

ABSTRACT

Probabilistic data is coming as a new deluge along with the technical advances on geographical tracking, multimedia processing, sensor network and RFID. While similarity search is an important functionality supporting the manipulation of probabilistic data, it raises new challenges to traditional relational database. The problem stems from the limited effectiveness of the distance metric supported by the existing database system. On the other hand, some complicated distance operators have proven their values for better distinguishing ability in the probabilistic domain. In this paper, we discuss the similarity search problem with the *Earth Mover's Distance*, which is the most successful distance metric on probabilistic histograms and an expensive operator with cubic complexity. We present a new database approach to answer range queries and k-nearest neighbor queries on probabilistic data, on the basis of Earth Mover's Distance. Our solution utilizes the primal-dual theory in linear programming and deploys B^+ tree index structures for effective candidate pruning. Extensive experiments show that our proposal dramatically improves the scalability of probabilistic databases.

1. INTRODUCTION

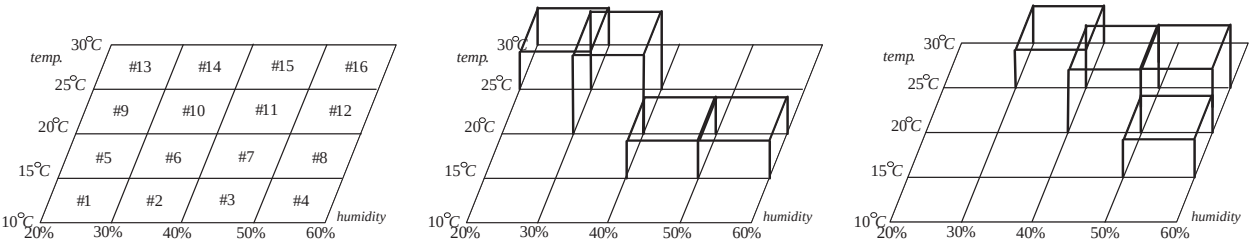
Probabilistic data is coming as a new deluge along with technical advances on geographical tracking [28], multimedia processing [13, 22], sensor network [11] and RFID [15]. This trend has led to the extensive research efforts devoted to scalable database system for probabilistic data management [3, 6, 7, 9, 10, 16, 28]. To fully utilize the information underlying the distributions, different probabilistic queries have been proposed and studied in different contexts, e.g. accumulated probability query [2, 27] and top-k query [8, 14, 17, 20, 26]. Most of the existing studies on these queries, however, simply extend the traditional database queries by handling uncertain attributes instead of exact ones. Unfortunately, these queries do not necessarily improve the usefulness of probabilistic database, because the underlying similarity measure on the probabilistic data remains unverified in their respective domains. On the other hand, research results in other areas, such as computer vision, have indicated that some complex distance operators are more meaningful for retrieval and search tasks. In this paper, we aim to bridge the gap between the database community and the real-world applications. In particular, we discuss the problem of similarity search based on the *Earth Mover's Distance* (EMD), which is one of the most popular distance operators in probabilistic domain.

Since the invention in late 1990s [21], EMD has become the de-facto distance metric used in the analysis of probabilistic histograms on multimedia data [13, 18, 22, 23, 25]. EMD is robust to outliers and tiny probability shifting, improving the quality of similarity search on probabilistic histograms. The improvement on search quality, however, pays an expensive cost on computation efficiency due to the cubic complexity of EMD with respect to the number of histogram bins. To relieve the efficiency issue, some approximation techniques are proposed in the computer vision [18, 25, 23] and algorithm community [4], to reduce the computational complexity of EMD. While these techniques accelerate the EMD computation between two probabilistic records, all of them do not scale well with huge amount of probabilistic data.

In recent years, some researchers in the database community are trying to address the similarity search problem on EMD. They attempted to design scalable solutions [5, 29], utilizing efficient and effective lower bounds on EMD. These solutions are mainly built on the scan-and-refinement framework, which incur high I/O costs and render low processing concurrency on the database system. To overcome the difficulties of these methods, we present a general approach to provide a truly scalable and highly concurrent index scheme applicable with mainstream relational databases, such as PostgreSQL and MySQL.

In our approach, all probabilistic records are mapped to a group of one-dimensional domains, using primal-dual theory [19] in linear programming. For each domain, a B^+ tree is constructed to index pointers to all probabilistic records based on the mapping values. Given a range query with a probabilistic histogram and threshold, our approach calculates a pair of lower and upper bound for each domain, guaranteeing that all of the query results are residing in the corresponding intervals. Range queries are thus issued on each B^+ tree, whose retrieved records are joined with intersection operator. Verifications and refinements are then conducted on the candidates remaining in the intersection

Proceedings of the VLDB Endowment, Vol. 3, No. 1



(a) Sensor reading domain partitions (b) Distribution of sensor s_1 (c) Distribution of sensor s_2

Figure 1: Examples of probabilistic records in form of histograms

Cells	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15	#16
p	0	0	0	0	0	0	0.2	0.2	0	0.4	0	0	0.2	0	0	0
q	0	0	0	0	0	0	0	0.2	0	0	0.3	0.3	0	0.2	0	0

Table 1: Tuple representation of example distributions in Figure 1

results. More complicated algorithms are designed to answer k-nearest neighbor query other than range query. Extensive experiments on real data sets show that our solution dramatically improves the efficiency of similarity search on EMD.

The rest of the paper is organized as follows. Appendix A reviews some related work on probabilistic database and Earth Mover's Distance. Section 2 introduces preliminary knowledge and problem definitions. Section 3 discusses our index on the probabilistic records with B^+ tree. Section 4 presents the details on the algorithms for range query and k-nearest neighbor query. Section 5 evaluates our proposal with empirical studies and Section 6 concludes this paper.

2. PRELIMINARIES

In this paper, we focus on the management of probabilistic records represented by probabilistic histograms. We use $\mathbb{D}$ to denote the object domain, covering all possible status of the objects in the real worlds. Depending on some domain knowledge, the object domain is partitioned into h cells. The distribution of each object is thus represented by a histogram, which records the probabilities of the objects appearing in the respective cells. In Figure 1, we present some examples of the probabilistic records. In this example, the distributions model the readings from sensor nodes monitoring temperature and humidity. The 2-dimensional space regarding temperature and humidity is divided into 16 cells. The distribution of sensor s_1 in Figure 1(b), for example, indicates it is more likely to observe s_1 with humidity in range of $[30\%, 40\%]$ and temperature in range of $[20°C, 25°C]$. Every distribution p is thus written in vector representation, i.e. $p = (p[1], p[2], \ldots, p[h])$, in which $p[i]$ is the probability of p in cell i. In Table 1, we list the table for the sensor reading distributions in Figure 1, in which $h = 16$ cells are numbered with increasing humidity and increasing temperature.

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

DEFINITION 2.1. ***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 linear program:*

$$\begin{aligned} Minimize: \quad & \frac{\sum_{i,j} f_{ij} \cdot d_{ij}}{\min\{\sum_i p[i], \sum_j q[j]\}} \\ s.t. \quad & \forall i : \sum\nolimits_j f_{ij} = p[i] \\ & \forall j : \sum\nolimits_i f_{ij} = q[j] \\ & \forall i, j : f_{ij} \geq 0 \end{aligned} \tag{1}$$

Note that the objective function in the program above can be simplified, because $\sum_i p[i] = \sum_j q[j] = 1$ in probabilistic space. There are h^2 variables used in the program above, $F = \{f_{ij}\}$, which intuitively model the flows from distribution p to distribution q. The cost of the flows is the weighted sum over all individual flows between every pair of cells. The constraints of the program guarantees that the amount of the flows from cell i is equal to the probability $p[i]$ for all i. Similarly, the probabilities flowing into cell j is exactly the same as $q[j]$ for all j. Based on the intuitions, $EMD(p, q)$ is the cost of the optimal flow set, F^*, minimizing the objective function. In Figure 2, we show an example of the optimal flow set from distribution p to distribution q in Figure 1 and Table 1, with Manhattan distance as the ground distance d_{ij} in object domain. It is thus straightforward to verify that $EMD(s_1, s_2) = 1.1$.

There are two types of similarity search queries investigated in this paper, including *range query* and *k-nearest neighbor query*. Specifically, given a probabilistic database D with n records, a query record q and a threshold θ, range query finds all records in D with EMD to q no larger than θ. A k-nearest neighbor query with respect to a record q, finds k records in D with smallest EMD to q than others.

In the rest of the section, we provide a brief review to the primal-dual theory of linear programming, paving the foundation of our indexing technique. Most of the theories and formulas introduced in this section can refer to [19].

[1]All the methods proposed do not rely on the choice of ground distance function d_{ij}.

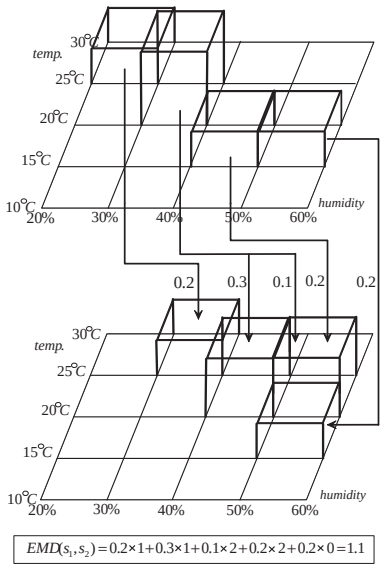


Figure 2: The optimal flowing from distribution s_1 to distribution s_2

For any linear program with minimization objective, known as the *primal* program, there always exists one and only one *dual* program, with maximization objective. Given the formulation of EMD in Definition 2.1, the 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:

$$\begin{aligned} Maximize: \quad & \sum\nolimits_i \phi_i \cdot p[i] + \sum\nolimits_j \pi_j \cdot q[j] \\ s.t. \quad & \forall i,j : \phi_i + \pi_j \le d_{ij} \\ & \forall i : \phi_i \in \mathbb{R} \\ & \forall j : \pi_j \in \mathbb{R} \end{aligned} \tag{2}$$

Given a linear program, a feasible solution to the program is a combination of variable values satisfying all the constraints in the program but not necessarily optimizing the objective function. There can be 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:

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

Equation (3) directly implies a pair of 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 primal program will be covered in Appendix B, which is used as a filter in range query and k-nearest neighbor query processing. In the following, we first present a simple example of a feasible solution to the dual program.

EXAMPLE 2.1. *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. $d_{ij} \ge 0$ for any i and j. This feasible solution leads to a trivial lower bound on $EMD(p,q)$:*

$$\sum \phi_i p[i] + \sum \pi_j q[j] = \sum p[i] - \sum q[j] = 0$$

In particular, the feasibility of a solution Φ only depends on the distance metric d_{ij} defined on the object domain. This property unveils the possibility on query-independent construction of index structure supporting a general class of lower bound computation on EMD.

3. INDEX STRUCTURE

Our index structure employs L B^+ trees, $\{T_1, \ldots, T_L\}$, to index the pointers of the probabilistic records in the database $\mathbb{D}$. Each tree T_l in the forest is associated with a feasible solution $\Phi_l = \{\phi_i^l, \pi_j^l\}$ in the dual program of EMD. Section 3.1 presents the details on the transformation from original probabilistic record to indexing value in T_l, and Section 3.2 provides some guidelines on the selection of feasible solutions. All the proofs of the lemmas and theorems in this section are available in Appendix C.

3.1 Mapping Construction

To ease the understanding difficulty on the mapping construction, the concepts of *key* and *counter-key* are first defined below.

DEFINITION 3.1. ***Key/Counter-Key***
Given a probabilistic record p and a feasible solution Φ_l in the dual program of EMD, the key of p on ϕ_l is

$$key(p, \Phi_l) = \sum\nolimits_i \phi_i^l \cdot p[i] \tag{4}$$

Similarly, counter-key is defined as follows

$$ckey(p, \Phi_l) = \sum\nolimits_j \pi_j^l \cdot p[j] \tag{5}$$

Given the selected feasible solution Φ_l, the B^+ tree T_l simply indexes all the pointers to the probabilistic records based on the value of $key(p, \Phi_l)$. The calculations on both $key(p, \Phi_l)$ and $ckey(p, \Phi_l)$ take only $O(h)$ time, linear in the number of cells in the object domain. It is important to emphasize again that Φ_l is independent to the query, facilitating the computation of $key(p, \Phi_l)$ before the insertion of p into T_l. The following two lemmas derive the lower bound and upper bound on $key(p, \Phi_l)$, in term of any query record q and $EMD(p,q)$.

LEMMA 3.1. *Given a record p indexed by T_l and a query record q, it is always valid that*

$$key(p, \Phi_l) \le EMD(p,q) - ckey(q, \Phi_l)$$

LEMMA 3.2. *Given a record p indexed by T_l and a query record q, we have*

$$key(p, \Phi_l) \ge \min_i(\phi_i + \pi_i) + key(q, \Phi_l) - EMD(p,q)$$

With the two lemmas above, all candidate records of the range query, (q, θ), must be located in the interval below in the domains constructed with T_l, i.e.,

$$key(p, \Phi_l) \in \left[\min_i(\phi_i + \pi_i) + key(q, \Phi_l) - \theta, \theta - ckey(q, \Phi_l)\right] \tag{6}$$

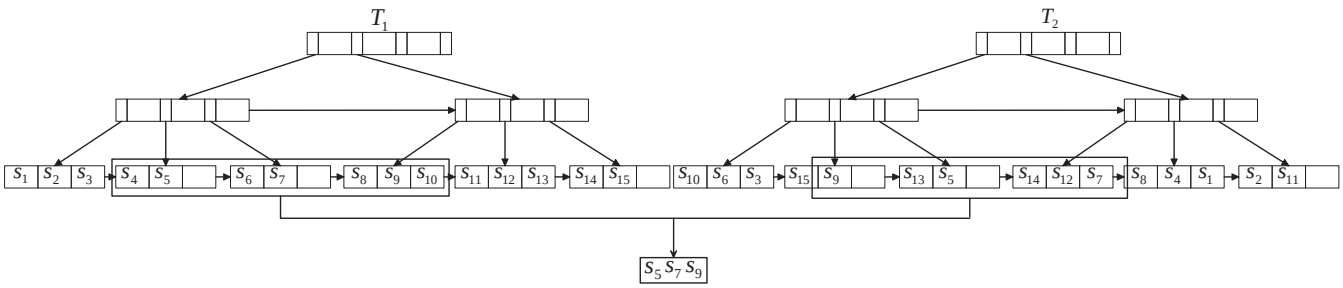


Figure 3: Example of the range query algorithm

This implies a simple scheme on range query processing. Some 1-dimensional range queries are constructed and run on all B^+ trees and the intersection of these query outputs are the candidates to the original query. Details of the algorithms will be covered later in Section 4.

It is worthwhile to note that an alternative solution is plausible here to deploy some multidimensional index trees, i.e. an R Tree instead of a group of B^+ trees. However, the curse of dimensionality limits the use of such multidimensional index tree with only few number of dimensions. The architecture with B^+ trees also enhances the flexibility of the system with respect to the number of adopted feasible solutions. Specifically, B^+ trees associated with specific feasible solutions can be easily inserted or removed at run time.

3.2 Selection of Feasible Solutions

The performance of the index scheme depends on the selection of the feasible solutions $\{\Phi_l\}$ for the B^+ trees $\{T_l\}$. In Example 2.1, we show that some feasible solution only provides trivial bounds on EMDs. In this part of the section, we discuss the issue on the selection of feasible solutions.

The first question is whether we can construct a feasible solution minimizing the gap between the lower bound and upper bound in Equation 6. Intuitively, a smaller gap leads to better pruning effect. Unfortunately, the following lemmas implies the gap cannot be improved.

LEMMA 3.3. *For any feasible solution Φ_l, the gap between the lower and upper bound used on the range query (q, θ) in Equation 6 is no smaller than 2θ.*

Due to the impossibility result above, we adopt some heuristic scheme to generate the feasible solutions for the dual program of EMD. Generally speaking, our selection method tries to avoid *Dominated* feasible solutions.

DEFINITION 3.2. *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 always outputs a weaker bound, i.e. $key(p, \Phi) \leq key(p, \Phi')$ and $ckey(q, \Phi) \leq ckey(q, \Phi')$ for any p and q. Basically, our scheme depends on the lemma below to eliminate dominated feasible solutions.

LEMMA 3.4. *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'.*

The lemma tells that a non-dominated feasible solution can be found if we can find the optimal solution to the dual program on any $EMD(p, q)$. Therefore, the B^+ trees are constructed by selecting appropriate probabilistic record pairs. There are two schemes designed for the selection procedure, as discussed below.

Clustering-Based Selection: In this scheme, a small sample set of the indexed records are retrieved from the database. By running clustering algorithm on the sample records, representative records are picked up from the clustering results. The pairwise EMD distance is calculated with Alpha-Beta algorithm [19], which returns optimal solutions for both primal and dual program. The solutions for the dual programs are thus adopted to construct the B^+ trees.

Random-Sampling-Based Selection: The clustering scheme pays expensive cost on the clustering procedure. To reduce the computation cost, a much cheaper random-sampling-based scheme is proposed here. Given the probabilistic database $\mathbb{D}$, it can be simply implemented by randomly picking up two records p and q from $\mathbb{D}$. The rest of the construction is similar to clustering-based selection scheme.

4. ALGORITHMS

In this section, we cover the details of the algorithms on range query (Section 4.1) and k-nearest neighbor query (Section 4.2) respectively. The pseudocodes of the algorithms are available in Appendix D.

4.1 Range Query

Based on the index scheme introduced in the previous section, it is straightforward to design processing algorithms based on Lemma 3.1 and Lemma 3.2.

In Figure 3, we present a running example to illustrate the query processing algorithm. Assume that there are 15 probabilistic records indexed in database, from s_1 to s_{15}, and two B^+ trees are used with Φ_1 and Φ_2 as the construction feasible solutions respectively. Given a range query (q, θ),the algorithm first generates two sub-range queries for T_1 and T_2, according to the ranges on the keys derived in Section 3.1. As shown in the figure, the sub-queries return two different groups of candidates with respect to their query ranges. In particular, the query result from T_1 contains 7 candidates, including $\{s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$. Similarly, T_2 returns 6 candidates, including $\{s_9, s_{13}, s_5, s_{14}, s_{12}, s_7\}$. The intersection of the two query results renders the final candidate set, $\{s_5, s_7, s_9\}$.

With the candidates to our original range query on EMD, filters are run in order to further prune the candidates. Specifically, two filters are equipped in our algorithm, $R-EMD$ (*EMD in reduced space*)[29] and LB_{IM}[5]. Details of

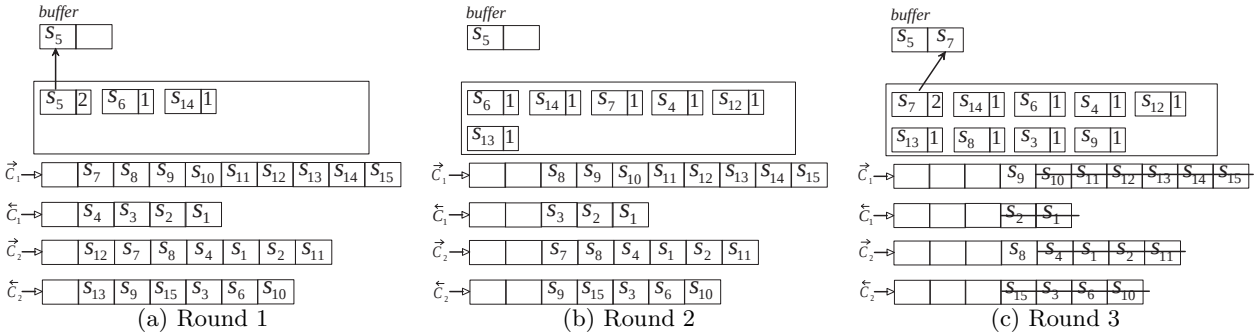


Figure 4: Running example of K-NN query processing

the two pruning filters are referred to Appendix A.3. A new upper bound, UB_P calculation is added to further prune unnecessary exact EMD computation, derived by feasible solution in primal program of EMD, which is described in Appendix B.

4.2 K-Nearest Neighbor Query

While range query is answered by selecting candidates from the results of sub-queries on the B^+ trees, the algorithm for k-nearest neighbor query is more complicated, since it is unlikely to know the range of the results on the B^+ trees. The basic idea of the algorithm is generating a sequence of candidate records based on the B^+ tree index structure. These candidates are estimated and verified to refresh the top-k nearest neighbor results. Some thresholds are accordingly updated to prune unnecessary verifications. The whole algorithm terminates when all records are pruned or verified. The complete pseudocodes are listed in Appendix D.2. In the rest of the subsection, we give a concrete example to illustrate the procedure of the algorithm.

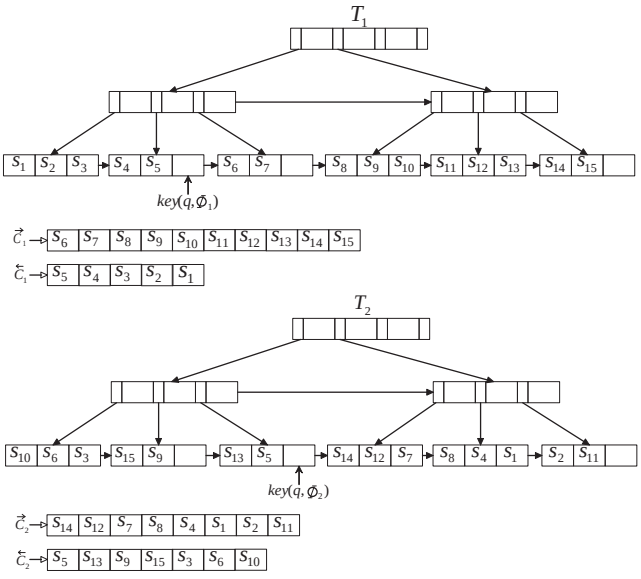


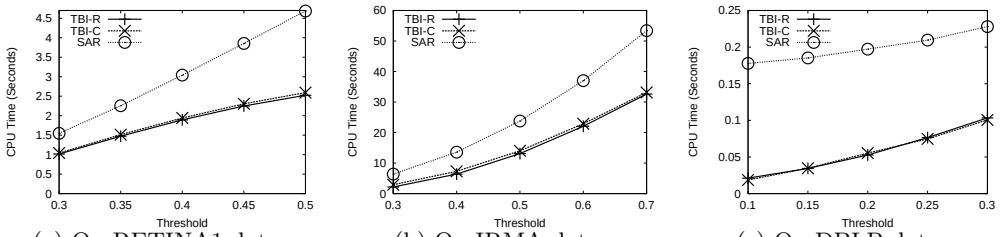
Figure 5: Construction of cursors for K-NN query

Given the k-nearest neighbor query, with probabilistic record q and integer k, the algorithm issues search queries on each index tree T_l with $key(q, \Phi_l)$, where Φ_l is the feasible solution used in the construction of T_l. In Figure 5, for example, $key(q, \Phi_1)$ is located in tree T_1 between record s_5 and s_6. After the positioning of $key(q, \Phi_1)$, the algorithm builds two cursors $\overrightarrow{C_1}$ and $\overleftarrow{C_1}$ to crawl the records from the position in right and left directions respectively. This renders two lists or record pointers, as shown in the figure. Similarly, $\overrightarrow{C_2}$ and $\overleftarrow{C_2}$ are initialized to visit the pointers sequentially on the tree T_2.

With the $2L$ cursors on L B^+ trees, our algorithm filters the candidates following the strategy of top-k query processing [12]. An empty buffer for temporary k-nearest neighbor results is initialized and all cursors start advancing on their lists in a round robin manner. A candidate is selected only when it is visited by exactly L cursors of different tree. The selected candidate is verified with the filters and finally evaluated with exact EMD computation if necessary. When the temporary k-nearest neighbor results are updated, the ranges for possible new results are calculated. The range boundaries on the trees are used to prune the cursor lists correspondingly. When all the cursor lists are finished, the algorithm stops and returns the current results in the k-nearest neighbor buffer. Figure 4 shows how the algorithm iterates over the cursors. In the first round, the cursors read the first record in their list. Since s_5 is on the top of two cursor lists, it will be added into the temporary buffer. The second round of the iterations does not select any records because no record has accumulated enough appearance in the cursor lists. The third round of the iteration selects the record s_7, leading to the update on thresholds and pruning of records on the cursor lists. The algorithm thus finishes the computation in the next round. Similar to range query, our k-nearest neighbor query algorithm applies all the filters used in range query algorithm except the UB_P.

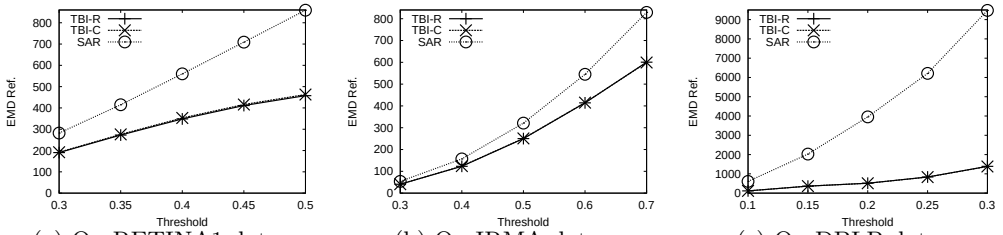
5. EXPERIMENTS

The details of the experimental settings are available in Appendix F. In this section, we present the experimental results on range query (Section 5.1), k-nearest neighbor query (Section 5.2) respectively. For the space limitation, we skip some figures and leave them to our technical report [30]. We named our approach TBI (Tree-Based Indexing) and named Scan and Refinement algorithm in [29] as SAR. TBI-R represents the feasible solution used for constructing the B^+ trees is Random-sampling-based while TBI-C indicates the Clustering-based, see section 3.2. The results of TBI-R method is the average based on 3 groups of random generated feasible solutions. We use the default parameter setting values mentioned in Appendix F for those non-marked pa-



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

Figure 6: Effect of threshold on average query CPU time for range queries



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

Figure 7: Effect of threshold on average EMD refinement number for range queries

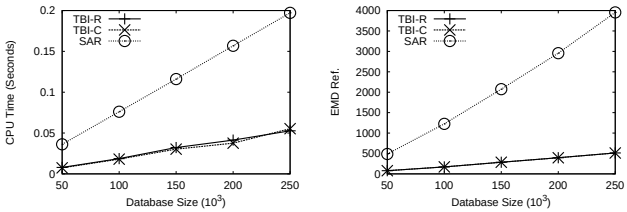
rameters in the figures.

5.1 Experiments on Range Query

Figure 6 and Figure 7 discuss the impact of similarity search threshold θ on the querying CPU time and the number of exact EMD refinements done for the queries. Figure 6 illustrates that both TBI-R and TBI-C beat SAR and the time cost of SAR can be 3-6 times larger than that of TBI on the DBLP data set. That's because the number of EMD refinement in TBI is markedly cut down especially on DBLP data set (see Figure 7). As discussed in previous sections, exact EMD calculation is an expensive operator with cubic complexity. Therefore, the number of exact EMD calculation is an important factor affecting the efficiency of EMD-based query processing. The decline of EMD refinement in TBI illustrates that our B^+ tree-based filtering technique and UB_P filtering method are better on candidate pruning for range query. Besides, although there is no remarkable difference on both the CPU time and the EMD refinement number between TBI-R and TBI-C on all data sets, TBI-R wins TBI-C a little which is quite delightful for TBI-R is much more easily than TBI-C to implement.

In Figure 8, we test the query CPU time by varying the number of B^+ trees in our indexing method. On RETINA1, the CPU time of both TBI-R and TBI-C gently decreases and there is not apparent difference between them. The difference between TBI-R and TBI-C is magnified on IRMA and TBI-C lags TBI-R in most of the settings. This phenomenon can be explained as the high dimensionality of IRMA leads to the worse clustering results used in the construction of B^+ trees. On DBLP data set, which has the largest cardinality ($cardinatlity = 250,000$) but the smallest histogram dimensionality ($dimensionality = 8$), the query efficiency gradually deteriorates when more than 3 or 4 trees are employed. This phenomenon also happens for TBI-R on IRMA. The reason for the performance deterioration is because that the pruning ability is strong enough with 2 or 3 trees and the addition of more trees only incur more searching time but unhelpful in reducing the number of candidates.

Figure 9 summarizes the experimental results on the effectiveness of the filters equipped in our query processing algorithm. The bars in the figure show the number of records passing the respective filters. From this figure we can see that filters after the B^+ trees index remain valuable for candidate pruning. Recall the results in Figure 7, since SAR proceeds with $R-LB_{IM}$ filter (LB_{IM} in reduced space) and then $R-EMD$ filter, our B^+ tree index and LB_P methods do provide excellent additional pruning power in an efficient way. Another observation in the figure is on the effectiveness of UB_P. It is more effective on lower dimensional data, especially on the DBLP data set.

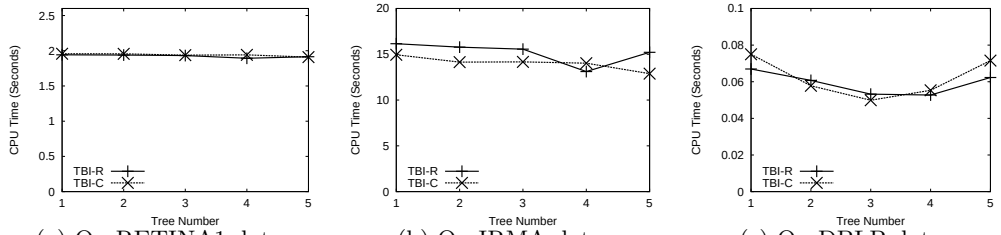


(a) On CPU Time (b) On EMD Refinement

Figure 10: Effect of data size on range queries

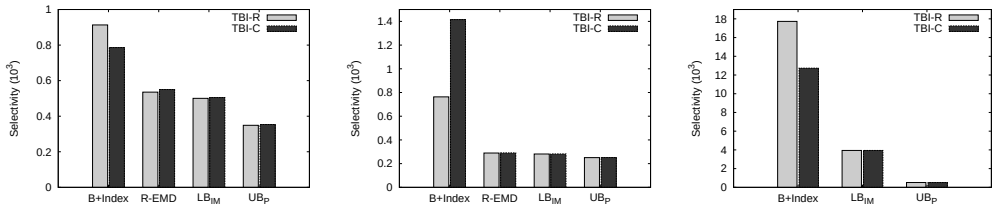
In Figure 10, we show the impact of database cardinality on CPU time and EMD refinement number. We can observe from the figure that without the assistance of B^+ tree index and UB_P filters, the SAR suffers a quick linear increase on both CPU time and EMD refinement number while our TBI methods exhibit a much slower growth. This explicates that the pruning effects of B^+ tree filter and UB_P filter remain significant even when the data cardinality is as large as 250,000.

The results of I/O cost are depicted in Figure 11. In Figure 11(a), we vary the database cardinality and observe that the I/O cost increases linearly with respect to the size of database. That's for with the increase of data cardinality, we need to visit much more records under a certain search range. When we alter the tree number, the I/O cost



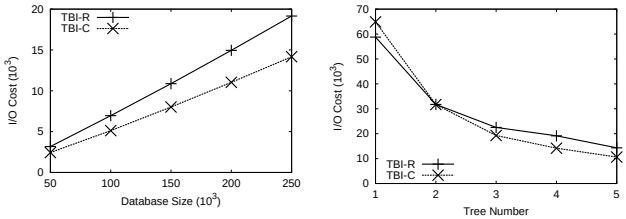
(a) On RETINA1 data (b) On IRMA data (c) On DBLP data

Figure 8: Effect of tree number on average query CPU time for range queries



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

Figure 9: Effect of filter functions on query selectivity for range queries



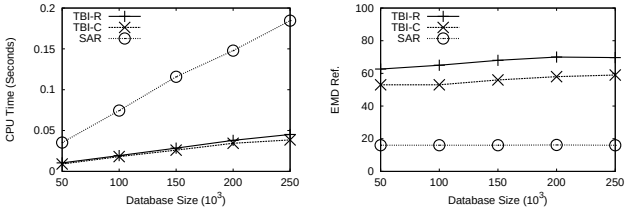
(a) Varying Data Size (b) Varying Tree Number

Figure 11: Test of I/O cost for range queries

drops evidently between tree number 1 to 2 and then declines slowly from 2 to 5. The reason is that installing more than 2 B^+ trees can not apparently decrease the number of candidates need to be accessed and thus the I/O decline becomes less remarkable.

5.2 Experiments on k-Nearest Neighbor Query

In this subsection, we empirically evaluate the performance of our algorithms on k-nearest neighbor query. Figure 12 summarizes the CPU time of k-NN query over different data sets. Our TBI approaches are generally better than SAR on all data sets and have an obvious advantage on large data sets. To explain this, we have claimed in Appendix A.3 that SAR can obtain an excellent query-based data ranking based on the KNOP framework [24]. Query-based data ranking is quite helpful to prune the unpromising records in the k-NN query and thus the EMD refinement number in SAR is lower than that of ours on IRMA and DBLP data sets (see Figure 13). However, the time cost in ranking becomes a bottleneck when the data cardinality is quite large (e.g., DBLP data set) or the time complexity of ranking distance function is very high (i.e., on IRMA data set, the ranking distance function can be the EMD over any two 60-dimensional reduced histograms and the ground distance used by EMD is the Euclidean distance between 40-dimensional feature vectors). That's why TBI can still win SAR although SAR has less EMD refinement number on IRMA and DBLP data sets. As to RETINA1, the ranking based on the reduced 18-dimensional histograms can not reflect the accurate order in the original 96-dimensional data space and thus increase the EMD refinement number which naturally leads to the degradation of query efficiency.



(a) On CPU Time (b) On EMD Refinement

Figure 14: Effect of data size for k-NN queries

The results on CPU time and EMD refinement number with varying the data size of DBLP in Figure 14 also match our expectation. For the ranking order of records in SAR is so perfect, its EMD refinement number approaches the optimal value 16 in a 16-NN query. However, the ranking cost causes the SAR to exhibit a poor CPU time.

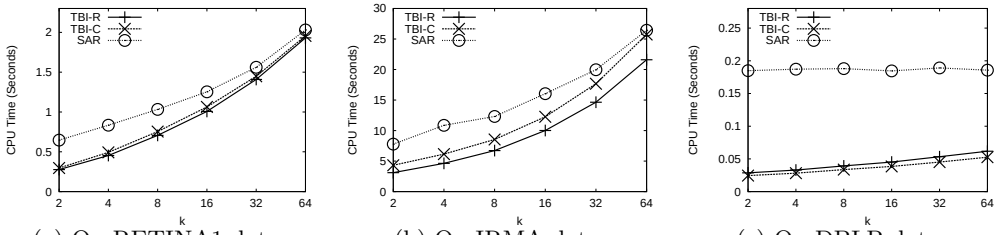
6. CONCLUSION

In this paper, we present a new indexing scheme for the general purposes of similarity search on Earth Mover's Distance. Our index method relies on the primal-dual theory to construct mapping functions from the original probabilistic space to one-dimensional domain. Each mapping domain is thus effectively indexed with B^+ tree structure. This proposal shows great advantage on the efficiency of query processing on different data sets.

7. ACKNOWLEDGEMENTS

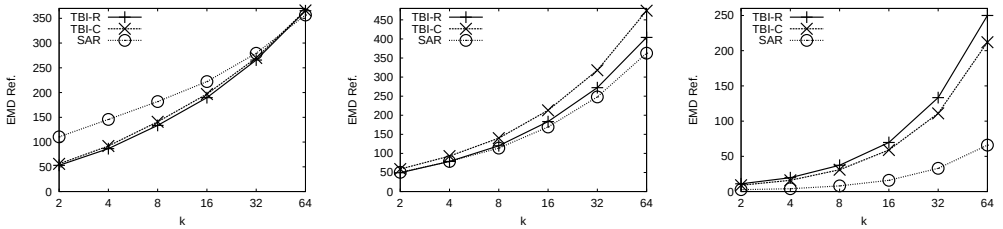
Zhenjie Zhang and Anthony K. H. Tung were supported by Singapore NRF grant R-252-000-376-279. This work is also supported by the National Natural Science Foundation of China (No. 60933001 and No. 60773220) and the National Basic Research Program of China (973 Program) under Grant No. 2006CB303103. The authors would like to thank Marc Wichterich for providing the codes of [29].

8. REFERENCES



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

Figure 12: Effect of k on average query CPU time for k-NN queries



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

Figure 13: Effect of k on average EMD refinement number for k-NN queries

[1] T. lehmann et al. irma project site. http://ganymed.imib.rwth-aachen.de/irma/.

[2] P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *PODS*, pages 137–146, 2009.

[3] 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.

[4] A. Andoni, P. Indyk, and R. Krauthgamer. Earth mover distance over high-dimensional spaces. In *SODA*, pages 343–352, 2008.

[5] I. Assent, A. Wenning, and T. Seidl. Approximation techniques for indexing the earth mover's distance in multimedia databases. In *ICDE*, page 11, 2006.

[6] O. Benjelloun, A. D. Sarma, A. Y. Halevy, and J. Widom. Uldbs: Databases with uncertainty and lineage. In *VLDB*, pages 953–964, 2006.

[7] R. Cheng, S. Singh, and S. Prabhakar. U-dbms: A database system for managing constantly-evolving data. In *VLDB*, pages 1271–1274, 2005.

[8] G. Cormode, F. Li, and K. Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *ICDE*, pages 305–316, 2009.

[9] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[10] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

[11] 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.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[13] K. Grauman and T. Darrell. Fast contour matching using approximate earth mover's distance. In *CVPR*, pages 220–227, 2004.

[14] 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.

[15] N. Khoussainova, M. Balazinska, and D. Suciu. Probabilistic event extraction from rfid data. In *ICDE*, pages 1480–1482, 2008.

[16] 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.

[17] J. Li, B. Saha, and A. Deshpande. A unified approach to ranking in probabilistic databases. *PVLDB*, 2(1):502–513, 2009.

[18] 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.

[19] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[20] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[21] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *ICCV*, pages 59–66, 1998.

[22] 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.

[23] R. Sandler and M. Lindenbaum. Nonnegative matrix factorization with earth mover's distance metric. In *CVPR*, pages 1873–1880, 2009.

[24] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD Conference*, 1998.

[25] S. Shirdhonkar and D. W. Jacobs. Approximate earth mover's distance in linear time. In *CVPR*, 2008.

[26] 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.

[27] Y. Tao, X. Xiao, and R. Cheng. Range search on multidimensional uncertain data. *ACM Trans. Database Syst.*, 32(3):15, 2007.

[28] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.

[29] 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.

[30] J. Xu, Z. Zhang, A. K. Tung, and G. Yu. Tra5-10: Efficient and effective similarity search over probabilistic data based on earth mover's distance, national university of singapore, 2010.

[31] 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.

APPENDIX

A. RELATED WORK

In this section, literature reviews are conducted to provide a brief introduction. In particular, Section A.1 focuses on the definitions on existing probabilistic queries in database systems and techniques of answering these queries. Section A.2 introduces the recent studies on approximation techniques of Earth Mover's Distance for fast evaluation. Section A.3 discusses in details on the existing solutions to similarity search on EMD, from database's perspective of view.

A.1 Probabilistic Queries in Database

Recent years have witnessed the fast advances of probabilistic data management, especially in techniques for efficient and effective query processing. In all of the studies on probabilistic query processing, two types of queries have been intensively investigated, *top-k query* and *accumulated probability query*[2].

DEFINITION A.1. ***Top-K Query***
Given the multidimensional database $D = \{s_1, s_2, \ldots, s_n\}$ with exact values in d-dimensional space and weighting vector $(w_1, w_2, \ldots, w_d)$, top-k query returns k objects with the maximal weighted sum on all dimensions.

While the definition on top-k query is clearly stated, it is challenging to extend it to probabilistic database. If the object s_i is uncertain on some dimensions, the weighted aggregation also becomes uncertain. To overcome the difficulty, different solutions are proposed to complete the semantic of top-k query in probabilistic database, including Uncertain Top-k [26], Uncertain Rank-k [26], Probabilistic Threshold Top-k [14], Expected Rank-k [8], PRF^{ω} and PRF^{e} [17].

DEFINITION A.2. ***Accumulated Probability Query***
Given the distributions from database $D = \{p_1, p_2, \ldots, p_n\}$, accumulated probability query with range R and threshold θ, return all distributions appearing in R with probability larger than θ, i.e. $\{p_i \in D \mid \Pr(p_i \in R) \geq \theta\}$.

Different indexing techniques have been proposed to support queries following the definition above. When the underlying domain contains a single dimension, for example, Agarwal et al. [2] proposed an index structure approximating the distributions with line segments. If the data is represented with *Possible World* model, some efficient Monte-Carlo simulation methods are proposed to evaluate the accumulated probability query efficiently [9, 20]. There is also some research studies on R-tree based index structure with multi-dimensional probability distributions [27].

The problem of range query and k-nearest neighbor query on EMD is totally different from the query types mentioned above. First, the objective is to discover similar distributions, but not to concern the probability of a specified region in the space. Second, the k-nearest neighbor is based on the distance to the querying distribution, which cannot be formulated with a simple ranking scheme as top-k query does.

[2]While it is called *range query* in some studies, we use the name here to distinguish from our range query definition w.r.t. EMD.

A.2 Earth Mover's Distance

Due to the formulation based on linear programming, the computation cost of Earth Mover's Distance is expensive. When first proposed in [21], Rubner et al. showed that exact EMD can be evaluated with the existing algorithm designed for the *Transportation Problem*. The complexity of the algorithm is cubic to the number of bins in the histograms. This has become the 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 [18], Ling and Okada investigated a special case of EMD, which used Manhattan distance as the ground distance d_{ij}. They modified the original *Simplex Algorithm* [19] to exploit the property of Manhattan distance. Although there is no theoretical proof on the acceleration effect, their empirical studies implies that their algorithm takes quadratic time in term of the cell number.

Shirdhonkar and Jacobs [25], as an another attempt, proposed a new distance function to approximate EMD. They conduct wavelet decomposition on the dual program of EMD and eliminates the parameters on small waves. The new distance can be efficiently calculated with linear time to the number of cells in the histograms. However, their method does not scale well with large data set.

A.3 Similarity Search on EMD

Existing solutions to the indexing problem on EMD mainly rely on the framework of scan and refinement [5, 29]. In this framework, a linear scan on the complete data records renders some candidates to the query results based on some efficient lower bound filters on EMD. A detailed framework of [29], for example, is shown in Figure 15. In pre-processing step, dimensionality reduction is conducted. In the scan phase, all reduced records are verified with two filters, EMD computation in reduced space and LB_{IM}. Given a range query, a final verification phase returns the final result by verifying the distances of all candidates passing the previous filters. For k-nearest neighbor query, the algorithm follows the optimal multi-step retrieval framework, known as KNOP [24], to guarantee not more candidates are produced in the each filter step. Firstly, all records are sorted on a lower bound. Another sequence of random accesses are conducted based on that ranking, until the top-k threshold is smaller than the lower bound of next record in the order. The major drawback of KNOP ,w.r.t. EMD, is the high I/O cost incurred by the sorting operation. Our solution based on B^+ tree successfully overcomes this difficulty and achieves much better scalability and high concurrency.

In the following, we discuss in details on two lower bounding techniques on EMD, LB_{IM} and *Dimensionality Reduction*, both of which are equipped in our index scheme.

Independent Minimization Lower Bound **(LB_{IM}):** Given two probabilistic records p and q of dimensionality h which satisfy $\sum_i p[i] = \sum_j q[j] = 1$, the independent minimization lower bound is the optimal result of the following linear program:

$$\begin{aligned} \underset{s.t.}{Minimize}: \quad & \sum\nolimits_{i,j} f_{ij} \cdot d_{ij} \\ & \forall i : \sum\nolimits_{j} f_{ij} = p[i] \\ & \forall i, j : f_{ij} \leq q[j] \\ & \forall i, j : f_{ij} \geq 0 \end{aligned} \tag{7}$$

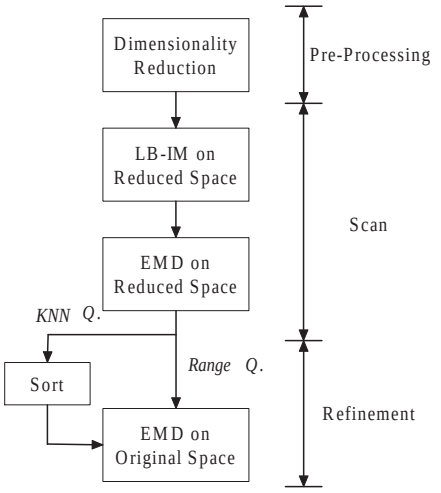


Figure 15: The Framework of scan and refinement algorithm

LB_{IM} simplifies the original linear programming problem of EMD by replacing the constraint $\sum_i f_{ij} = q[j]$ with $f_{ij} \leq q[j]$. Intuitively, this LB_{IM} relaxes the original constraints on EMD by only requiring the incoming flow not to exceed the bin's capacity. This lower bound can be efficiently evaluated with quadratic complexity to the bin number.

Rule for Histogram Reduction:

A general linear dimensionality reduction of histogram from dimensionality d to d' is pictured by a reduction matrix $R = [r_{ij} \in \mathbb{R}^{d \times d'}]$. And the reduction procedure of a d-dimensional histogram H to a d'-dimensional histogram H' is given by:

$$H' = H \cdot R \tag{8}$$

where the reduction matrix $R \in \Re_{d \times d'}$ is defined by complying with the following constraints:

$$\forall 1 \leq i \leq d \quad \forall 1 \leq j \leq d' \quad : r_{ij} \in \{0, 1\} \tag{9}$$

$$\forall 1 \leq i \leq d \quad : \sum_j^{d'} r_{ij} = 1 \tag{10}$$

$$\forall 1 \leq j \leq d' \quad : \sum_i^{d} r_{ij} \geq 1 \tag{11}$$

Any dimensionality reduction of the Earth Mover's Distance also requires the specification of the corresponding reduced cost matrix which provides the ground distance information in the reduced space. The optimal reduced cost matrix with respect to a certain reduction matrix R can be obtained by following the rule below:

Rule for Cost Matrix Reduction.

The optimal reduced cost matrix $C' = [c'_{i'j'}]$ is defined by:

$$c'_{i'j'} = min\{c_{ij} | r_{ii'} = 1 \wedge r_{jj'} = 1\} \tag{12}$$

This rule can ensure the lower bound property to the original cost matrix and thus the EMD over the reduced histograms and the LB_{IM} over the reduced histograms can be the lower bound to the original distance function.

B. UTILIZING FEASIBLE SOLUTION IN PRIMAL PROGRAM

Algorithm 1 Select Primal Feasible Solution (record p, record q)

```
Sort the probabilities {p[i]} in non-ascending order
Initialize an array {r[j]} with r[j] = q[j]
Initialize flow set {f_ij} with f_ij = 0
for each p[i] in the order do
   for each r[j] in ascending order of d_ij do
      if r[j] > p[i] then
         f_ij = p[i]
         r[j] = r[j] − p[i]
      else
         f_ij = r[j]
         r[j] = 0
         p[i] = p[i] − r[j]
Return {f_ij}
```

Equation 3 implies that any feasible solution to the primal program serves as an upper bound on EMD. In this section, we discuss the details on the fast construction of such feasible solution. This technique is utilized to prune the computation of exact $EMD(p, q)$ in range query when the upper bound of $EMD(p, q)$ is already smaller than the the threshold θ.

Intuitively speaking, our feasible solution construction algorithm sorts the probabilities of p in non-ascending order. For each $p[i]$, the algorithm tries to construct flows to assign the probability to cells with closer distance. The assignment automatically removes the capacity of the target cell. This procedure continues until all the probabilities are assigned. The correctness of the algorithm relies on the fact that $\sum p[i] = \sum q[j]$. Thus, the assignment always ends with a valid flow set satisfying all the constraints in the primal program of EMD. In Algorithm 1, the details of the method are presented. The array $\{r[i]\}$ is used to maintain the current capacity of the cells in the histogram. If $r[j]$ can be fully absorbed by the nearest cell, the algorithm finishes the computation on $p[i]$. Otherwise, the computation keeps assigning $p[i]$ to other cells until enough capacity is met.

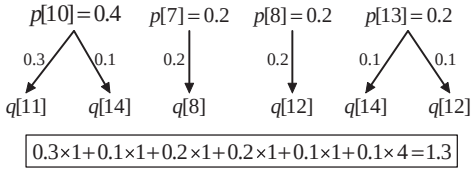


Figure 16: Example on feasible solution construction

Recall the example shown in Table 1. If running the algorithm above s_1 and s_2 in the table, the algorithm finishes with the feasible solution in Figure 16. The cost of the feasible solution is the upper bound UB_P of the original EMD (Here, UB_P=1.3, if d_{ij} is Manhattan distance on the cell positions). Compared against the optimal flows in Figure 2, the upper bound is slightly larger than the exact $EMD(p, q) = 1.1$.

C. THEOREM AND LEMMA PROOFS

In this section, we provide all the proofs to the theorems

and lemmas in this paper.

Proof to Lemma 3.1

PROOF. Based on the primal-dual theory shown in Equation (3), we know $\sum_i \phi_i p[i] + \sum_j \pi_j q[j] \leq EMD(p,q)$ for any feasible solution Φ. By replacing $\sum_i \phi_i p[i]$ with $key(p,\Phi)$ and $\sum_j \pi_j q[j]$ with $ckey(q,\Phi)$, we reach the conclusion of the lemma. □

Proof to Lemma 3.2

PROOF. Due to the symmetry property on the metric distance, we have $EMD(p,q) = EMD(q,p)$. Thus, the lower bound on $EMD(q,p)$ also works for $EMD(p,q)$. By applying Lemma 3.1, we have

$$key(q,\Phi) + ckey(p,\Phi) \leq EMD(p,q) \tag{13}$$

On the other hand, if add $key(p,\Phi)$ into $ckey(p,\Phi)$, the following inequalities can be derived.

$$\begin{aligned} key(p,\Phi) + ckey(p,\Phi) &= \sum\nolimits_i \phi_i p[i] + \sum\nolimits_j \pi_j p[j] \\ &= \sum\nolimits_j (\phi_j + \pi_j) p[j] \\ &\geq \sum\nolimits_j \min_i(\phi_i + \pi_i) p[j] \\ &= \min_i(\phi_i + \pi_i) \end{aligned} \tag{14}$$

Combing Equation (13) and Equation (14), some simple algebra brings us to the conclusion of the lemma. □

Proof to Lemma 3.3

PROOF. The gap between the lower bound and upper bound on the range query (q,θ) in Equation 6 is minimized with the following inequalities.

$$\begin{aligned} & (\theta - ckey(q,\Phi_l)) - \Big(\min_i(\phi_i + \pi_i) + key(q,\Phi_l) - \theta\Big) \\ = \;& 2\theta - \min_i(\phi_i + \pi_i) - (ckey(q,\Phi_l) + key(q,\Phi_l)) \\ \geq \;& 2\theta - (ckey(q,\Phi_l) + key(q,\Phi_l)) \\ = \;& 2\theta \end{aligned} \tag{15}$$

The first inequality is due to the 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 with the lower bound on $EMD(q,q)$. □

Proof to Lemma 3.4

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 based on Definition 3.2. This leads to the inequality below.

$$\sum\nolimits_i \phi'_i p[i] + \sum\nolimits_j \pi'_j q[j] \geq \sum\nolimits_i \phi_i p[i] + \sum\nolimits_j \pi_j q[j] \tag{16}$$

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

D. ALGORITHM PSEUDOCODES

In this section, detailed algorithm pseudocodes are provided to supplement Section 4.

Algorithm 2 Range Query (record q, threshold θ, B^+ trees $\{T_l\}$)

1: **for** each T_l **do**
2: Calculate $minSum_l = \min_i(\phi_i^l + \pi_i^l)$
3: $lb = minSum_l + key(q,\Phi_l) - \theta$
4: $ub = \theta - ckey(q,\Phi_l)$
5: $C_l = RangeQuery(T_l, lb, ub)$
6: Clear buffer B
7: $B = C_1 \cap \ldots \cap C_L$
8: Filter the B based on R-EMD
9: Filter the B based on LB_{IM}
10: Filter the B based on UB_P
11: Refine the B using the original EMD
12: Return B

D.1 Range Query Algorithm

Given a range query, (q,θ), from line 1-6 of Algorithm 2, we firstly calculate the lower bound and upper bound based on the theory proposed in Section 3.1. Range queries on the B^+ trees with the corresponding query range return candidate sets to $\{C_l\}$. Intersection on these sets renders a new candidate set in buffer B. Candidates in B are then filtered with the bound derived with $R-EMD$, LB_{IM} and UB_P in order. Finally, the exact query result is verified with exact EMD computation, as shown from line 7 to line 11.

D.2 kNN Query Algorithm

In Algorithm 3, from line 1-2, we firstly locate the address of the leaf node whose key value is closest to $key(q,\Phi_l)$. At second, two pointers, $\overrightarrow{C_l}$ and $\overleftarrow{C_l}$, are initialized with pointers to that leaf node. In line 4, we set kNN threshold ε to MAX which means that we need to consider all data records at the first round. Traversal on each B^+ tree continue, using pointers $\overrightarrow{C_l}$ and $\overleftarrow{C_l}$, on line 7-16. Emphatically, $\overrightarrow{C_l}$ and $\overleftarrow{C_l}$ crawl the data records from the position in right and left directions respectively. If the crawled record is observed by L B^+ trees, it is added into the *checkList*. Later, each record in the *checkList* is verified by use of B^+ Tree filter(line 20-21), $R-EMD$ filter(line 22-23) and LB_{IM} filter(line 24-25). Final refinement using exact EMD computation is performed and the update on both ε and kNN list when a record in *checkList* has an EMD value smaller than the current ε, see line 27-31. The updated ε is then used to update the range boundaries of L trees. If all elements within its tree's range boundary is visited and the *checkList* is empty, the algorithm terminates and returns the final kNN results.

E. ALGORITHM COMPLEXITY ANALYSIS

For the algorithm complexity analysis between TBI and SAR, we provide the worst case analysis on kNN query below in brief. The worst case for both TBI and SAR happens when none of the filtering techniques is able to prune any candidate record from the database. Assume that the database cardinality is N, the number of histogram bins is d and there are L B^+ trees equipped. SAR takes $O(N)$ time to scan the complete data set, $O(N \log 2N)$ time to sort all the candidates and $O(Nd^3)$ time to verify the EMDs for all records. The total computation time of SAR is thus $O(N \log N + Nd^3)$. In the same situation, TBI spends $O(L(\log N + N))$ time on candidate selection over all L B^+

Algorithm 3 kNN Query (record q, k, B^+ trees $\{T_l\}$)

```
1: for each T_l do
2:    Initialize →C_l and ←C_l using the info of key(q, Φ_l)
3: Initialize each element in array status as 0
4: ε =MAX
5: while TRUE do
6:   for each T_L do
7:     if →C_r.next(ε)! = NULL then
8:       rId = →C_r.getNext()
9:       status[rId] + +;
10:      if status[rId] == L then
11:        checkList.add(rId)
12:    if ←C_l.next(ε)! = NULL then
13:      lId = ←C_l.getNext()
14:      status[lId] + +
15:      if status[lId] == L then
16:        checkList.add(lId)
17:    if (Cannot getNext in all trees)
       &&(checkList.empty==TRUE) then
18:      Return {kNNList}
19:    for each element el_i in checkList do
20:      if max_l(key(el_i, Φ_l) + ckey(el_i, Φ_l)) > ε then
21:        continue;
22:      else if Can be filtered by R-EMD then
23:        continue;
24:      else if Can be filtered by LB_IM then
25:        continue;
26:      else
27:        if EMD(el_i, q) < ε then
28:          kNNList.add(el_i)
29:          if kNNList.size == k + 1 then
30:            Delete the one in kNNList with the
               largest EMD to q
31:            ε = max_i(EMD(kNNList[i], q))
```

trees and $O(Nd^3)$ time on distance refinements. This leads to the total complexity of $O(LN + Nd^3)$. Restate that L is normally a small constant and N can converge the infinity, which means the time complexity of TBI is better than that of SAR.

F. EXPERIMENT SETUP

In this section, we introduce the setup of the experiments, including the data preparation, experimental environment and parameter settings. We begin with describing the three real data sets we used.

RETINA1 Data Set: This is an image data set consists of 3,932 feline retina scans labeled with various antibodies. For each image, twelve 96-dimensional histograms are abstracted. Each bin of the histogram has a 2-dimensional *feature* vector. A *feature* represents the location of its corresponding bin and is used for the ground distance calculation.

IRMA Data Set: This data set contains 10,000 radiography images from the Image Retrieval in Medical Application (IRMA) project [1]. The dimensionality of each histogram in IRMA is 199 and the *feature* of each bin is a 40-dimensional vector. Thus, IRMA becomes the most time-consuming data set for each individual EMD calculation amongst our three real data sets.

DBLP Data Set: This is a 8-dimensional histogram data set with 250,100 records, and it is generated from the DBLP database retrieved in Oct. 2007. The 8 dimensions of each histogram represent 8 different domains in computer science, including artificial intelligence, application, database, hardware, software, system, theory and bio-information. We define the feature of each bin/domain considering its correlation to the following three aspects, i.e., computer, mathematics and architecture. As thus, each histogram dimension will have an 3-dimensional *feature* vector. For the other specific content of DBLP data set, please refer to [31].

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
k of kNN query	2,4,8,**16**,32,64
ground distance	**Euclidean**, Manhattan
DBLP data size	50,100,150,200,**250** ($\times 10^3$)

Table 2: Varying parameters

RETINA1 and IRMA data sets are also used by literature [29]. We calculate the ground distance between arbitrary two bins based on their *feature* vectors. For example, on IRMA data set, the ground distance between bin i and bin j can be the Euclidean Distance between their corresponding 40-dimension *feature* vectors.

The reported results in our experiments are the averages over a query workload of 100. 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. Therefore, the cardinalities of the RETINA1, IRMA and DBLP databases are 3,832, 9,900 and 250,000 respectively. We compare our similarity query algorithm with the scan-and-refine algorithm (we named it as SAR) proposed in literature [29]. The SAR algorithm, to the best of our knowledge, is the most efficient exact EMD-based k-NN algorithm over high-dimensional histograms. The dimension reduction matrixes used in SAR are the most excellent ones according to the experimental results in [29]. Specifically speaking, we use the 18-dimensional reduction matrix generated by FB-ALL-KMed method on RETINA1 data set and use the 60-dimensional reduction matrix yielded by also the FB-ALL-KMed method on IRMA data set. The default filter function chain we used in our TBI methods follows 'B^+ *Tree Index* → *R-EMD*(EMD on reduced space)→ LB_{IM} → UB_P → *EMD*(EMD on original space)'. Particularly, we leave out the UB_P filter in the k-NN query. And we skip the *R-EMD* filter on the DBLP data set, for the histogram dimensionality in DBLP is 8 and it is already very low. To verify the efficiency of our algorithm, we measure the *Query Response Time*, *the Number of EMD Refinement*, *Querying I/O cost* in our experiments.

In Table 2, we summarize the parameters and their varying ranges in our experiments. The default value of each parameter is highlighted in bold.

All the programs are compiled by Microsoft VS 2005 in Windows XP and run on a PC with Intel Core2 2.4GHz CPU, 2G RAM and 150G hard disk.