Electronic Preprint for Journal of Information Processing Vol.24 No.1

Regular Paper

Top-*k* Similarity Search over Gaussian Distributions Based on KL-Divergence

Tingting Dong[1,a)] Yoshiharu Ishikawa[1,b)] Chuan Xiao[1,c)]

Received: June 20, 2015, Accepted: October 8, 2015

Abstract: The problem of similarity search is a crucial task in many real-world applications such as multimedia databases, data mining, and bioinformatics. In this work, we investigate the similarity search on uncertain data modeled in Gaussian distributions. By employing *Kullback-Leibler divergence* (KL-divergence) to measure the dissimilarity between two Gaussian distributions, our goal is to search a database for the top-*k* Gaussian distributions similar to a given query Gaussian distribution. Especially, we consider non-correlated Gaussian distributions, where there are no correlations between dimensions and their covariance matrices are diagonal. To support query processing, we propose two types of novel approaches utilizing the notions of *rank aggregation* and *skyline queries*. The efficiency and effectiveness of our approaches are demonstrated through a comprehensive experimental performance study.

Keywords: Gaussian distributions, Kullback-Leibler divergence, top-*k* similarity search

1. Introduction

Probabilistic modeling, which infers probability distributions from vast amount of data for real-world applications, is being practiced in a wide range of fields from statistics, machine learning and pattern recognition [4] to bioinformatics and medical informatics [16]. The research on managing probabilistic model-based data was pioneered by Deshpande et al. [11], and then received considerable attention from the database research community [1], [2], [29]. In this paper, we study the problem of processing similarity search queries over probabilistic model-based data, specifically, over objects represented by Gaussian distributions. As shown in **Fig. 1**, given a database of Gaussian distributions G and a query Gaussian distribution q, our objective is to find top-k Gaussian distributions from G that are similar to q.

Gaussian distribution, one of the most typical probability distributions, is widely used in statistics, pattern recognition and machine learning [4], [13]. Research work on Gaussian distributions has been conducted over a long period of time, including music classification [20] and search [26], and image retrieval [5], [12].

For this reason, we focus on similarity search over data modeled in Gaussian distributions, assuming that a large number of objects, represented by non-correlated Gaussian distributions are stored in the database. By *non-correlated Gaussian distribution*, we mean that all dimensions are independent with each other, i.e., the covariance matrix of each Gaussian distribution is diagonal. In this paper, we focus on non-correlated Gaussian distributions since they are frequently used in machine learning and statistics. Hereafter, we use the term Gaussian distributions for non-correlated ones. We will report query processing methods for the general correlated case in another paper because they are very different. Given a query Gaussian distribution, our task is to retrieve from the database the Gaussian distributions that are similar to the query. The top-k Gaussian distributions with the highest similarity scores are returned as the answer to the query.

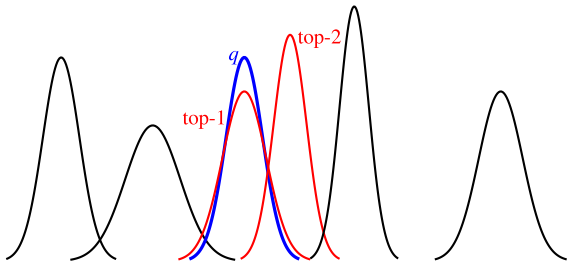


Fig. 1 A one-dimensional query example ($k = 2$).

In Ref. [5], Böhm et al. considered similarity search on feature vectors such as structural features of 2-D contours [21], time series [15] and color histograms in image databases. They represented the uncertainty of each feature vector using a non-correlated multidimensional Gaussian distribution. As discussed in Ref. [4], compared to general Gaussian distributions, the number of parameters, the storage and computational requirements can be reduced substantially by using non-correlated Gaussian distributions. In consequence, the non-correlated Gaussian distribution is often preferred in practical applications [25].

Furthermore, Gaussian mixture models (GMMs), which are linear combinations of Gaussian distributions, are known for their ability to model arbitrarily shaped distributions. For instance, GMMs are used to model driver behaviors for driver identification in Ref. [22]. We believe that our work paves the way for similarity search over GMMs, which will be beneficial for many real world applications such as finding drivers with similar driv-

[1] Nagoya University, Nagoya 464–8601, Japan
[a)] dongtt@nagoya-u.jp
[b)] ishikawa@is.nagoya-u.ac.jp
[c)] chuanx@nagoya-u.jp

ing behaviors.

To capture the similarity between a data Gaussian distribution and a query Gaussian distribution, we choose *Kullback-Leibler divergence* (KL-divergence) [10], a representative measure for quantifying the similarity between two probability distributions. KL-divergence is introduced in Ref. [19], and has then become the commonest divergence measures used in practice [9].

It is well-known that KL-divergence is a non-metric measure which violates the properties of a standard distance function in metric spaces such as the Euclidean space with the Euclidean distance. Specifically, it is asymmetric and does not satisfy the triangular inequality. Hence, existing index structures based on distance functions for metric spaces like M-tree [8] cannot be employed to solve this problem.

A naïve solution is to sequentially compute the KL-divergence with the query Gaussian distribution for each Gaussian distribution in the database, and select ones with top-k smallest KL-divergences. However, this method poses remarkable computing overhead and hence is not scalable to large datasets. In consequence, we employ the *filter-and-refine* paradigm to improve the efficiency of query processing. It first generates a set of promising candidates and filters unpromising ones without computing their similarities, and then candidate objects are refined to obtain the final results.

We propose two types of approaches utilizing the notions of *rank aggregation* [14] and *skyline queries* [6]. The first type presorts all objects in the database on their attributes and computes result objects by merging candidates from presorted lists. We modify the representative *threshold algorithm* (TA) [14] and propose two algorithms for efficient query processing. The second one transforms the problem to the computation of *dynamic skyline queries* [23]. We extend and modify the branch-and-bound skyline (BBS) algorithm [23], which is proposed to answer skyline queries, and develop a novel algorithm to solve this problem.

We note that although there have been several studies on searching in non-metric spaces [7], [27], [28], [30], they are mainly developed for the generic class of non-metric similarity measures in discrete domains. None of them paid particular attention to the case where objects are modeled in Gaussian distributions and KL-divergence is chosen as the similarity measure. To the best of our knowledge, our work is the first study in similarity search based on KL-divergence over Gaussian distributions.

Our contributions are listed as follows.

(1) We formalize the problem of top-k similarity search based on KL-divergence over Gaussian distributions, and analyze mathematically the properties of KL-divergence between two Gaussian distributions.

(2) We propose two types of approaches to improve the efficiency of query processing using the notion of rank aggregation and skyline queries.

(3) We demonstrate the efficiency and effectiveness of our approaches through a comprehensive experimental performance study.

The rest of the paper is organized as follows. We formally define the problem in Section 2. Then we analyze KL-divergence of Gaussian distributions in Section 3 and propose two types of approaches in Section 4 and Section 5. Experimental results and analyses are presented in Section 6. We review related work in Section 7. Finally, Section 8 concludes the paper.

2. Problem Definition

2.1 Gaussian Distribution

In the one-dimensional space, a Gaussian distribution is described by the average μ and the variance σ^2:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]. \quad (1)$$

In the d-dimensional space, it is represented by the average vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ [4]:

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]. \quad (2)$$

$|\boldsymbol{\Sigma}|$ (resp. $\boldsymbol{\Sigma}^{-1}$) is the determinant (resp. inverse matrix) of $\boldsymbol{\Sigma}$. $(\cdot)^T$ means the transposition of $(\cdot)$. In this work, we assume that a large number of objects represented by Gaussian distributions are stored in the database. For simplicity, objects represented by Gaussian distributions are called *Gaussian objects*, and Gaussian objects in the database are called *data Gaussian objects* afterwards.

2.2 Similarity Measure: KL-divergence

Given two continuous probability distributions $p_1(x)$ and $p_2(x)$, the *Kullback-Leibler divergence* (KL-divergence) or *relative entropy* [10] between them is

$$\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) = \int_{-\infty}^{+\infty} p_1(x)\ln\frac{p_1(x)}{p_2(x)}dx. \quad (3)$$

In information theory, KL-divergence $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2)$ is interpreted as a measure of the inefficiency of assuming that the distribution is p_2 when the true distribution is p_1 [10]. In other words, it measures the information lost when p_2 is used to approximate p_1. The smaller the KL-divergence is, the more similar the two probability distributions are. Accordingly, the problem of *KL-divergence-based top-k similarity search over Gaussian distributions (KLD-Gauss)* is actually equivalent to finding top-k Gaussian objects having the smallest KL-divergences with the query Gaussian object.

KL-divergence satisfies the following properties of standard distance functions: 1) non-negativity: $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) \geq 0$; 2) identity: $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) = 0$ if and only if $p_1(x) = p_2(x)$. However, it is not symmetric, i.e., $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) \neq \mathcal{D}_{\mathrm{KL}}(p_2\|p_1)$ in general. Moreover, it violates the notion of triangular inequality, namely, $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) + \mathcal{D}_{\mathrm{KL}}(p_2\|p_3) \geq \mathcal{D}_{\mathrm{KL}}(p_1\|p_3)$ does not necessarily hold. In other words, KL-divergence is a *non-metric* measure [28]. Hence, index structures designed for query processing in metric spaces such as M-tree [8] and iDistance [18] cannot be applied to accelerate similarity search based on KL-divergence.

As KL-divergence is asymmetric, given a data Gaussian object p and a query Gaussian object q, there are two options when

using it as the similarity measure between them: $\mathcal{D}_{\mathrm{KL}}(p\|q)$ or $\mathcal{D}_{\mathrm{KL}}(q\|p)$. It is not easy to decide which one to use, and may vary according to different applications. Both of them are common in the literature. Thus, in this paper, we study both types.

The naïve approach of solving the *KLD-Gauss* problem is to perform a sequential scan over all objects in the database and compute their KL-divergences for each query. This approach is obviously too time-consuming and will induce intolerable computation cost for many real-world applications, especially over large scale databases. To improve the efficiency of query processing, we adopt the well-known *filter-and-refine* paradigm. The rationale is to avoid unnecessary computations by developing effective filtering techniques.

In this paper, we propose two types of approaches for filtering. They utilize the notions of *rank aggregation* [14] and *skyline queries* [6], respectively. Below we first present our analysis of KL-divergence of Gaussian distributions, and then introduce our approaches.

3. KL-divergence of Gaussian Distributions

Given two one-dimensional Gaussian distributions $p_1(x) = \mathcal{N}(\mu_1, \sigma_1)$ and $p_2(x) = \mathcal{N}(\mu_2, \sigma_2)$, their KL-divergence, denoted as $D^1_{\mathrm{KL}}(p_1\|p_2)$, is as follows [24]:

$$\mathcal{D}^1_{\mathrm{KL}}(p_1\|p_2) = \frac{1}{2}\left[\frac{(\mu_1-\mu_2)^2}{\sigma_2^2} + \frac{\sigma_1^2}{\sigma_2^2} - \ln\frac{\sigma_1^2}{\sigma_2^2} - 1\right]. \tag{4}$$

The two types of KL-divergence, $\mathcal{D}^1_{\mathrm{KL}}(p\|q)$ and $\mathcal{D}^1_{\mathrm{KL}}(q\|p)$, are referred to as $\mathcal{D}^1_{\mathrm{KL1}}$ and $\mathcal{D}^1_{\mathrm{KL2}}$, respectively.

In the d-dimensional space, given $p_1(x) = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $p_2(x) = \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, $D^d_{\mathrm{KL}}(p_1\|p_2)$ is defined by

$$\mathcal{D}^d_{\mathrm{KL}}(p_1\|p_2) = \frac{1}{2}\left[\sum_{i=1}^{d}\left(\frac{(\mu_{1,i}-\mu_{2,i})^2}{\sigma_{2,i}^2} + \frac{\sigma_{1,i}^2}{\sigma_{2,i}^2} - \ln\frac{\sigma_{1,i}^2}{\sigma_{2,i}^2}\right) - d\right], \tag{5}$$

where $\mu_{1,i}$ (resp. $\mu_{2,i}$) is the i-th ($1 \le i \le d$) element of p_1 (resp. p_2)'s average vector $\boldsymbol{\mu}_1$ (resp. $\boldsymbol{\mu}_2$). According to the independence assumption, $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ are diagonal matrices and their diagonal elements are denoted by $\sigma^2_{1,i}$ and $\sigma^2_{2,i}$, respectively. Obviously, $\mathcal{D}^d_{\mathrm{KL}} = \sum_{i=1}^{d}\mathcal{D}^i_{\mathrm{KL}}$, where $\mathcal{D}^i_{\mathrm{KL}}$ is the KL-divergence in the i-th dimension.

Similarly, the two types of $\mathcal{D}^d_{\mathrm{KL}}$, $\mathcal{D}^d_{\mathrm{KL}}(p\|q)$ and $\mathcal{D}^d_{\mathrm{KL}}(q\|p)$, are denoted by $\mathcal{D}^d_{\mathrm{KL1}}$ and $\mathcal{D}^d_{\mathrm{KL2}}$, respectively. Since they are sums of the one-dimensional case, their properties of monotonicity in each dimension are the same to that of $\mathcal{D}^1_{\mathrm{KL1}}$ and $\mathcal{D}^1_{\mathrm{KL2}}$, which will be discussed subsequently.

3.1 $\mathcal{D}^1_{\mathrm{KL}}(p\|q)$: $\mathcal{D}^1_{\mathrm{KL1}}$

As the smaller $\mathcal{D}^1_{\mathrm{KL1}}$ is, the more similar p is to q, we differentiate $\mathcal{D}^1_{\mathrm{KL1}}$ on p, specifically on μ_p and σ_p^2, and obtain the following equations:

$$\frac{\partial\mathcal{D}^1_{\mathrm{KL1}}}{\partial\mu_p} = \frac{\mu_p-\mu_q}{\sigma_q^2} \tag{6}$$

$$\frac{\partial\mathcal{D}^1_{\mathrm{KL1}}}{\partial\sigma_p^2} = \frac{\sigma_p^2-\sigma_q^2}{\sigma_p^2\sigma_q^2}. \tag{7}$$

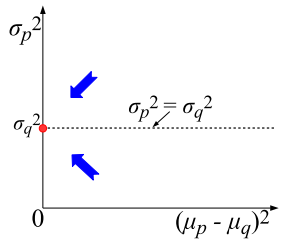


Fig. 2 Property of $\mathcal{D}^1_{\mathrm{KL1}}$.

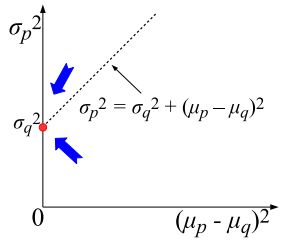


Fig. 3 Property of $\mathcal{D}^1_{\mathrm{KL2}}$.

By letting both Eq. (6) and Eq. (7) equal to 0, we derive the following property illustrated in **Fig. 2**. The arrows indicate decreasing directions of KL-divergence. We use $(\mu_p - \mu_q)^2$ as the horizontal axis to make the figure easy to understand.

Property 1 $\mathcal{D}^1_{\mathrm{KL1}}$ is a monotonically increasing function centered at (μ_q, σ_q^2) and divided by $\mu_p = \mu_q$, $\sigma_p^2 = \sigma_q^2$.

(1) As μ_p increases, $\mathcal{D}^1_{\mathrm{KL1}}$ increases monotonically when $\mu_p > \mu_q$, and decreases monotonically when $\mu_p < \mu_q$.

(2) As σ_p^2 increases, $\mathcal{D}^1_{\mathrm{KL1}}$ increases monotonically when $\sigma_p^2 > \sigma_q^2$, and decreases monotonically when $\sigma_p^2 < \sigma_q^2$.

(3) $\mathcal{D}^1_{\mathrm{KL1}}$ is minimized at $\mu_p = \mu_q$, $\sigma_p^2 = \sigma_q^2$, and its minimum is 0.

Obviously, $\mathcal{D}^1_{\mathrm{KL1}}$ is divisionally monotonous. The closer a point is to the center (μ_q, σ_q^2) and the dividing lines $\mu_p = \mu_q$ and $\sigma_p^2 = \sigma_q^2$, the smaller $\mathcal{D}^1_{\mathrm{KL1}}$ is. In other words, smaller $|\mu_p - \mu_q|$ and $|\sigma_p^2 - \sigma_q^2|$ lead to smaller $\mathcal{D}^1_{\mathrm{KL1}}$.

3.2 $\mathcal{D}^1_{\mathrm{KL}}(q\|p)$: $\mathcal{D}^1_{\mathrm{KL2}}$

Similarly, we differentiate $\mathcal{D}^1_{\mathrm{KL2}}$ on μ_p and σ_p^2, and get the following equations:

$$\frac{\partial\mathcal{D}^1_{\mathrm{KL2}}}{\partial\mu_p} = \frac{\mu_p-\mu_q}{\sigma_p^2} \tag{8}$$

$$\frac{\partial\mathcal{D}^1_{\mathrm{KL2}}}{\partial\sigma_p^2} = \frac{\sigma_p^2-\sigma_q^2-(\mu_p-\mu_q)^2}{\sigma_p^4}. \tag{9}$$

In the same way, by letting both Eq. (8) and Eq. (9) equal to 0, we can obtain the following property illustrated in **Fig. 3**. It differs from $\mathcal{D}^1_{\mathrm{KL1}}$ in that its plane is divided by $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$ instead of $\sigma_p^2 = \sigma_q^2$.

Property 2 $\mathcal{D}^1_{\mathrm{KL2}}$ is a monotonically increasing function centered at (μ_q, σ_q^2) and divided by $\mu_p = \mu_q$, $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$.

(1) As μ_p increases, $\mathcal{D}^1_{\mathrm{KL2}}$ increases monotonically when $\mu_p > \mu_q$, and decreases monotonically when $\mu_p < \mu_q$.

(2) As σ_p^2 increases, $\mathcal{D}^1_{\mathrm{KL2}}$ increases monotonically when $\sigma_p^2 > \sigma_q^2 + (\mu_p - \mu_q)^2$, and decreases monotonically when $\sigma_p^2 < \sigma_q^2 + (\mu_p - \mu_q)^2$.

(3) $\mathcal{D}^1_{\mathrm{KL2}}$ is minimized at $\mu_p = \mu_q$, $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$, and its minimum is 0.

Similarly, the closer a point is to the center (μ_q, σ_q^2) and the dividing lines $\mu_p = \mu_q$ and $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$, the smaller $\mathcal{D}^1_{\mathrm{KL2}}$ is. Hence, smaller $|\mu_p - \mu_q|$ and $|\sigma_p^2 - \sigma_q^2 - (\mu_p - \mu_q)^2|$ indicate smaller $\mathcal{D}^1_{\mathrm{KL2}}$.

4. TA-based Query Processing

The problem of sequential scan lies in that, it has to compute KL-divergences of all objects in the database, even though only k of them will be the answer. Based on this observation and monotonic properties of KL-divergence discussed in Section 3, we consider selecting a small number of promising candidate objects by presorting to avoid computing KL-divergences for unpromising objects. For example, in the one-dimensional case, we can sort the database on μ and σ^2 in advance, and only consider ones whose μ and σ^2 are close enough to that of q.

This is the basic idea of our first type of approaches, which utilize the notion of *rank aggregation* [14]. Generally speaking, we rank objects on each attribute and aggregate ones with high ranks to obtain the final top-k objects. Below we use the representative *threshold algorithm* (TA) [14] for description. To better solve the *KLD-Gauss* problem, we propose novel algorithms by modifying the TA algorithm. Below we first describe the TA algorithm and then introduce our proposed algorithms.

4.1 The TA Algorithm

TA assumes that a middleware system S aggregates answers to queries from various subsystems. Each subsystem S_i supports two modes of data access: *sorted access* and *random access*. In the *sorted access* mode, S obtains the grade of an object in the sorted list of S_i by proceeding through the list sequentially from the top. On the other hand, *random access* to S_i returns the corresponding grade of a given id.

For example, consider a database of images in **Table 1** with two subsystems S_{color} and S_{shape}. S_{color} returns images based on their colors, and S_{shape} is based on shapes. Given a query of Q : ($color$ = "*red*") $\wedge$ ($shape$ = "*round*"), S merges images with high redness grades in S_{color} and high roundness grades in S_{shape} to obtain images satisfying the query.

Assume that Q requests top-3 images based on the score function $s_Q(x) = min\{s_{red}(x), s_{round}(x)\}$, where $s_{red}(x)$ (resp. $s_{round}(x)$) denotes the redness (resp. roundness) grade of image x. The left table lists top-5 images with their grades in each subsystem. Assume we retrieve one image by each sorted access. First, we obtain $\{s_{red}(x_1) = 0.96, s_{round}(x_2) = 0.95\}$. Other grades of x_1 and x_2, $s_{round}(x_1) = 0.76$ and $s_{red}(x_2) = 0.91$, can be obtained via random access. Thus, $s_Q(x_1) = 0.76$ and $s_Q(x_2) = 0.91$. The two images are both added into a result set $R = \{(x_2, 0.91), (x_1, 0.76)\}$. At the same time, the threshold τ is kept $\tau = min\{0.96, 0.95\} = 0.95$.

Table 1 An example of the TA algorithm.

Top images of subsystems

Rank	$s_{red}(x_i)$	$s_{round}(x_i)$
1	(0.96, x_1)	(0.95, x_2)
2	(0.91, x_2)	(0.92, x_3)
3	(0.85, x_4)	(0.85, x_5)
4	(0.81, x_3)	(0.83, x_4)
5	(0.72, x_5)	(0.76, x_1)

Top images in result

Rank	$s_Q(x_i)$, *attribute*
1	(x_2, 0.91), *red*
2	(x_4, 0.83), *round*
3	(x_3, 0.81), *red*

This is the possible best score of all images unprocessed. Once scores of images in R are all no smaller than τ, the algorithm stops and returns R.

Next, $s_{red}(x_2) = 0.91$ and $s_{round}(x_3) = 0.92$ are retrieved. Since x_2's score has already been computed, we do random access only for x_3 and compute its score $s_Q(x_3) = 0.81$. Then x_3 is added into R and τ is updated to 0.91. In the next step, since x_4 has a better score than x_1, we update $R = \{(x_2, 0.91), (x_4, 0.83), (x_3, 0.81)\}$ and $\tau = 0.85$. Finally, as x_3 and x_4 have already been processed, we only need to update $\tau = 0.81$. At this point, τ is no better than any of images in R, i.e., no image unprocessed can have a better score than that in R. Therefore, the algorithm terminates and returns R as shown in the right table of Table 1. For ease of understanding, we associate each score with its corresponding attribute, i.e., red or round.

4.2 The Proposed Algorithms

To utilize TA, we redefine *sorted access* and *random access* for the *KLD-Gauss* problem. Given an object id, *random access* returns the corresponding average vector and covariance matrix. We design two types of *sorted access*. The first one retrieves $\mu_{p,j}$ or $\sigma^2_{p,j}$ $(1 \le j \le d)$ and object id in the ascending order of $|\mu_{p,j} - \mu_{q,j}|$ or $|\sigma^2_{p,j} - \sigma^2_{q,j}|$ (or $|\sigma^2_{p,j} - \sigma^2_{q,j} - (\mu_{p,j} - \mu_{q,j})^2|$, omit afterwards). The second one gives access to $\mathcal{D}^j_{\mathrm{KL}}$ $(1 \le j \le d)$ and object id in the ascending order of $\mathcal{D}^j_{\mathrm{KL}}$. They are called *CompleteTA* (CTA) and *PartialTA* (PTA), respectively, and will be detailed subsequently.

4.2.1 The CTA Algorithm

For CTA, we presort the database on $\mu_{p,j}$ and $\sigma^2_{p,j}$ $(1 \le j \le d)$, and get $2d$ sorted lists. For example, in **Table 2** the left table shows a list of 12 one-dimensional objects, and the right table shows their sorted orders on μ_i and σ_i^2 (called S_μ and S_{σ^2}, respectively). In the multidimensional case, for each dimension j, we sort all objects on both $\mu_{i,j}$ and $\sigma^2_{i,j}$ and get $2d$ sorted lists: $(S_{\mu,1}, S_{\sigma^2,1}), \ldots, (S_{\mu,d}, S_{\sigma^2,d})$. By default, in each dimension j, the sorted access to each list of average returns an object p with its average $\mu_{p,j}$ in the ascending order of $|\mu_{p,j} - \mu_{q,j}|$, and the sorted access to each list of variance returns another object g with its variance $\sigma^2_{g,j}$ in the ascending order of $|\sigma^2_{g,j} - \sigma^2_{q,j}|$.

Algorithm 1 shows the straightforward application of the TA algorithm using the redefined random access and sorted access. The algorithm runs by dimensions. In each dimension j, we retrieve an object g_a with its average $\mu_{a,j}$ by sorted access to the list of average. The average value of the retrieved object, $\mu_{a,j}$, is

Table 2 An example dataset and its sorted orders.

g_i	μ_i	σ_i^2
g_1	2	2.5
g_2	3.5	2.1
g_3	5	2.7
g_4	7	2.4
g_5	9	2.5
g_6	8	1.8
g_7	4	0.7
g_8	6	1.2
g_9	6.4	1.1
g_{10}	9	0.8
g_{11}	8.5	0.2
g_{12}	10.6	0.5

Order	S_μ	S_{σ^2}
1	$(2, g_1)$	$(0.2, g_{11})$
2	$(3.5, g_2)$	$(0.5, g_{12})$
3	$(4, g_7)$	$(0.7, g_7)$
4	$\mathbf{(5, g_3)}$	$\mathbf{(0.8, g_{10})}$
5	$(6, g_8)$	$\mathbf{(1.1, g_9)}$
6	$(6.4, g_9)$	$(1.2, g_8)$
7	$(7, g_4)$	$(1.8, g_6)$
8	$(8, g_6)$	$(2.1, g_2)$
9	$(8.5, g_{11})$	$(2.4, g_4)$
10	$(9, g_{10})$	$(2.5, g_1)$
11	$(9, g_5)$	$(2.5, g_5)$
12	$(10.6, g_{12})$	$(2.7, g_3)$

Algorithm 1 The straightforward CTA algorithm

```
1: Initialize the top-k list R and the query q (μ_q, σ_q^2);
2: repeat
3:     foreach dimension j do
4:         SortedAccess_avg() → (μ_{a,j}, g_a);
5:         SortedAccess_var() → {(σ̄^2_{v,j}, ḡ_v), (σ̲^2_{v,j}, g̲_v)};
6:         if Any of g_a, ḡ_v, g̲_v has not been accessed then
7:             Do RandomAccess and calculate its KL-divergence;
8:             Update R;
9:         end if
10:    end for
11:    Let μ_j, σ̄^2_j, σ̲^2_j be the last values accessed by SortedAccess;
12:    μ ← {μ_j|1 ≤ j ≤ d}; σ̄^2 ← {σ̄^2_j|1 ≤ j ≤ d}; σ̲^2 ← {σ̲^2_j|1 ≤ j ≤ d};
13:    τ ← min{CalcKLD(μ, σ̄^2), CalcKLD(μ, σ̲^2)};
14: until |R| = k and KL-divergences in R are all no greater than τ;
15: return R;
```

closest to that of the query, $\mu_{q,j}$, i.e., $|\mu_{a,j} - \mu_{q,j}|$ is the smallest. If there is a tie, i.e., there are two objects g_a and $g_{a'}$ satisfying $|\mu_{a,j} - \mu_{q,j}| = |\mu_{a',j} - \mu_{q,j}| = \Delta_\mu$, we can break it randomly since the algorithm relies on τ and the computation of τ depends on Δ_μ not $\mu_{a,j}$ or $\mu_{a',j}$.

On the other hand, because of the asymmetry of KL-divergence discussed in Section 2, each sorted access to the list of variance should return two objects in the two directions of $\overline{\sigma}^2_{v,j} : \sigma^2_{v,j} \geq \sigma^2_{q,j}$ and $\underline{\sigma}^2_{v,j} : \sigma^2_{v,j} < \sigma^2_{q,j}$ (one in each direction) instead of one object to ensure correctness. We explain it using the following example. Consider a query q with $\mu_q = 5$, $\sigma^2_q = 1$ and $k = 3$. In the first step, while we retrieve $(5, g_3)$ from S_μ, from S_{σ^2} both $(1.1, g_9)$ and $(0.8, g_{10})$ need to be retrieved (in bold) to ensure correctness because we do not know which of them will have a smaller KL-divergence with respect to q. If we retrieve only $(1.1, g_9)$ and find $(5, 0.8)$ has a smaller KL-divergence than that of $(5, 1.1)$ with respect to q when computing τ, τ will be larger than it is supposed to be and this may lead to a wrong result. In other words, we start processing from the entries in bold and continue searching in both directions.

If an object is accessed for the first time by sorted access, we obtain its average vector and covariance matrix by random access and calculate its KL-divergence (Line 6–7). For example, we do random access for g_3, g_9 and g_{10}, and compute their KL-divergences ($\mathcal{D}^1_{\mathrm{KL1}}$ is used for computing KL-divergences in this example).

Then we update the top-k list R as follows. When $|R| < k$, the object with its KL-divergence is added to R directly. When $|R|$ achieves k, if its KL-divergence is better than any entry in R, we add it into R and delete the entry with the largest KL-divergence so that R only maintains the best k objects. Meanwhile, we compute the threshold τ using the last accessed average and variance values (Line 11–13). As τ is the best KL-divergence of all objects unseen, the algorithm terminates when KL-divergences in R are all no greater than τ.

We show the processing steps of the example query in **Table 3**. Continuing with the example, we update $R = \{(g_3, 0.35), (g_9, 0.98), (g_{10}, 8.01)\}$. At the same time, we compute KL-divergences of $(5, 1.1)$ and $(5, 0.8)$ with respect to the query $(5, 1)$, and update τ as the smaller one, which is 0.002.

Table 3 Query processing using the straightforward CTA.

Step	Retrieved objects	R	τ
1	g_3, (g_9, g_{10})	(g_3, 0.35), (g_9, 0.98), (g_{10}, 8.01)	0.002
2	g_7, (g_7, g_8)	(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)	0.508
3	g_8, (g_6, g_{12})	(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)	0.597

Table 4 Query processing using the improved CTA.

Step	Retrieved objects	R	τ
1	g_3, g_9	(g_3, 0.35), (g_9, 0.98)	0.002
2	g_7, g_8	(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)	0.508
3	g_8, g_{10}	(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)	0.512
4	g_9, g_7	(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)	1.01

In the second step, $(4, g_7)$ from S_μ, $(1.2, g_8)$ and $(0.7, g_7)$ from S_{σ^2}, are retrieved. Since KL-divergences of g_7 and g_8 are smaller than that of g_9 and g_{10}, we update R as $\{(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)\}$ and $\tau = 0.508$. In the last step, we retrieve $(6, g_8)$ from S_μ, $(1.8, g_6)$ and $(0.5, g_{12})$ from S_{σ^2}. R stays the same, but τ is updated to 0.597. Finally, as all the objects in R have no greater KL-divergences than τ, we stop searching and return R.

In each step, as the straightforward algorithm retrieves two objects with respect to the variance in a brute-force way, it tends to do many unnecessary accesses. We avoid them by considering the priority of the object in each direction and access only one in each step. We derive the following lemma to guide the algorithm (see the proof in A.1).

Lemma 1 (1) Assume $|\mu_{p,j} - \mu_{q,j}| = |\mu_{p',j} - \mu_{q,j}|$ $(1 \leq j \leq d)$ and $\sigma^2_{p,m} = \sigma^2_{p',m}$ $(1 \leq m \leq d, m \neq j)$. If $\sigma^2_{p,j} > \sigma^2_{q,j}$, $\sigma^2_{p',j} < \sigma^2_{q,j}$, and $(\sigma^2_{p,j} - \sigma^2_{q,j}) \leq (\sigma^2_{q,j} - \sigma^2_{p',j})$, then $\mathcal{D}_{\mathrm{KL}}(p\|q) < \mathcal{D}_{\mathrm{KL}}(p'\|q)$. (2) Assume $|\mu_{p,j} - \mu_{q,j}| = |\mu_{p',j} - \mu_{q,j}|$ $(1 \leq j \leq d)$ and $\sigma^2_{p,m} = \sigma^2_{p',m}$ $(1 \leq m \leq d, m \neq j)$. If $\sigma^2_{p,j} > \sigma^2_{q,j} + (\mu_{p,j} - \mu_{q,j})^2$, $\sigma^2_{p',j} < \sigma^2_{q,j} + (\mu_{p',j} - \mu_{q,j})^2$, and $(\sigma^2_{p,j} - \sigma^2_{q,j} - (\mu_{p,j} - \mu_{q,j})^2) \leq (\sigma^2_{q,j} + (\mu_{p',j} - \mu_{q,j})^2 - \sigma^2_{p',j})$, then $\mathcal{D}_{\mathrm{KL}}(q\|p) < \mathcal{D}_{\mathrm{KL}}(q\|p')$.

Based on Lemma 1, during the bidirectional search over the variance, when an object p with $\sigma^2_{p,j} > \sigma^2_{q,j}$ is obtained, we only need to consider another object p' with $\sigma^2_{p',j} < \sigma^2_{q,j}$, if $\sigma^2_{p',j}$ is nearer to $\sigma^2_{q,j}$ (or $\sigma^2_{q,j} + (\mu_{p',j} - \mu_{q,j})^2$) than that of p. For example, in the first step, in S_{σ^2} when comparing $(1.1, g_9)$ with $(0.8, g_{10})$, since $(1.1 - 1) \leq (1 - 0.8)$, we only retrieve g_9 and delay the retrieval of g_{10}. In other cases, we compare their KL-divergences using the current average value μ_j obtained, i.e., KL-divergences of $(\mu_j, \sigma^2_{p,j})$ and $(\mu_j, \sigma^2_{p',j})$, and select the one with a smaller KL-divergence with respect to q.

We show the running steps of the improved CTA algorithm in **Table 4**. At first, we retrieve $(5, g_3)$ from S_μ. Meanwhile, we retrieve $(1.1, g_9)$ from S_{σ^2} based on Lemma 1 as explained above. Accordingly, we update R and τ. Then we continue to retrieve $(4, g_7)$ from S_μ. Since $(1.2 - 1) \leq (1 - 0.8)$, we retrieve $(1.2, g_8)$ from S_{σ^2} based on Lemma 1 and update R and τ accordingly. In the third step, when comparing $(1.8, g_6)$ with $(0.8, g_{10})$ in S_{σ^2}, because they do not satisfy Lemma 1, we compute their KL-divergences using the current average value $\mu_8 = 6$. In other words, we compare KL-divergences of $(6, 1.8)$ and $(6, 0.8)$ with respect to $(5, 1)$. Then g_{10} is selected. As we can see, while the straightforward CTA algorithm accessed 7 objects, the improved one only retrieved 5 objects with one additional step.

4.2.2 The PTA Algorithm

For PTA, in each dimension we construct a two-dimensional

R-tree of all data Gaussian objects. When a query q comes, by using the skyline-based approach described in Section 5, we can obtain the top objects with the smallest $\mathcal{D}^i_{\mathrm{KL}}$ with respect to q in each dimension i (detailed in Section 5.6). The PTA algorithm is similar to the CTA algorithm, except that we retrieve objects based on $\mathcal{D}^i_{\mathrm{KL}}$ instead of μ_i and σ_i^2 with respect to q, and τ is calculated by $\tau = \sum_{i=1}^{d} \mathcal{D}^i_{\mathrm{KL}}$.

Next, we introduce another approach, which transforms the *KLD-Gauss* problem to *dynamic skyline queries* [23] by using the notion of *skyline queries* [6].

5. Skyline-based Query Processing

In this section, by utilizing the properties of KL-divergence and Gaussian distributions, we propose another approach for solving the *KLD-Gauss* problem. We extend and modify the BBS (Branch-and-Bound Skyline) algorithm [23], which is proposed to answer skyline queries. Below we first introduce the concept of skyline queries and the BBS algorithm, and then describe our extensions and the proposed algorithm.

5.1 Skyline Queries and Dynamic Skyline Queries

Given a dataset D, a skyline query [6] returns all the objects not *dominated* by others within D. p_i is said to be *dominated* by p_j, if p_j is better than p_i on at least one attribute, and better than or equals to p_i on other attributes. A common example is, given a list of hotels with prices and distances from a beach, to find ones having the lowest prices and the shortest distances. A hotel with \$200 and 1 *km* from the beach is preferable to (*dominate*) the one of \$250 and 1.5 *km*.

A dynamic skyline query, a variation of a skyline query, returns all the objects that are not *dynamically dominated* with respect to a set of *dimension functions* $f_1, f_2, \ldots, f_m$ specified by the query [23]. Each function f_i takes as parameters a subset of the n attributes in the original n-dimensional space. The objective is to return the skyline in the new m-dimensional space defined by $f_1, f_2, \ldots, f_m$.

Continuing with the hotel example, assume for each hotel we store its x and y coordinates and price c (3-dimensional) information in the database. Dynamic skyline queries can be used to retrieve hotels that are closest to a specified location (x_u, y_u) and price c_u. In other words, closeness functions are defined by $f_1 = \sqrt{(x - x_u)^2 + (y - y_u)^2}$, and $f_2 = |c - c_u|$. Note that (x_u, y_u) and c_u normally vary with queries.

In Ref. [23], Papadias et al. proposed the BBS algorithm for processing skyline queries. They proved that BBS is I/O optimal; that is, it accesses the least number of disk pages among all algorithms based on R-trees. Hence, the following discussion concentrates on this technique.

5.2 The BBS Algorithm

Consider the task of computing the skyline of the example dataset in Table 2, i.e., finding objects with the smallest averages and variances. According to BBS, we construct an R-tree to index all objects. Each Gaussian object is inserted into the R-tree as a two-dimensional point (μ_i, σ_i^2). Its image and hierarchical structure are shown in **Fig. 4** and **Fig. 5**. Each group of objects, i.e., an R-tree node, is represented by an MBR (Minimum Bounding Rectangle).

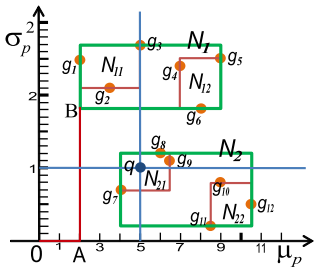


Fig. 4 R-tree image.

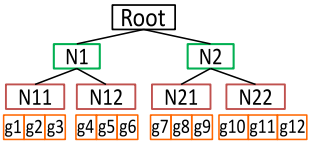


Fig. 5 R-tree structure.

A priority queue Q is employed to maintain entries (R-tree nodes or Gaussian objects) to be accessed in the ascending order of *mindist*. The *mindist* of an entry, is the smallest cityblock (L_1) distance of its MBR to a reference point. For example, the *mindist* of N_1 to the origin O is calculated by summing up the length of OA and AB, where B is the lower-left point of N_1, and A is the projection of B on the μ_p-axis. The *mindist* of a Gaussian object to O, e.g., $g_1 = (2, 2.5)$, is simply its L_1 distance to O, i.e., $2 + 2.5 = 4.5$.

We use the example in Fig. 4 to illustrate the algorithm. After expanding the root node, entries in Q are $\{(N_1, 3.8), (N_2, 4.2)\}$. Then we expand N_1 and insert N_{11}, N_{12} into Q. Q becomes $\{(N_{11}, 4.1), (N_2, 4.2), (N_{12}, 8.8)\}$. When expanding N_{11}, since g_3 is dominated by g_1 (and g_2), it is rejected. After expanding N_2, $Q = \{(g_1, 4.5), (N_{21}, 4.7), (g_2, 5.6), (N_{22}, 8.7), (N_{12}, 8.8)\}$.

Next g_1 is added into a candidate set S as the first skyline object. Since N_{21} is not dominated by g_1, it is expanded and g_7 is inserted into Q. Then $Q = \{(g_7, 4.7), (g_2, 5.6), (N_{22}, 8.7), (N_{12}, 8.8)\}$. g_7 and then g_2 are both added into S because g_7 is not dominated by g_1, and g_2 is not dominated by g_7 and g_1. Subsequently, expanding N_{22} leads to another candidate g_{11}. The last entry N_{12} will not be expanded as it is dominated by g_7. Finally, $S = \{g_1, g_7, g_2, g_{11}\}$ is returned as the skyline result.

BBS can be applied to compute dynamic skylines by expanding entries in Q according to *mindist* in the dynamic space [23]. In others words, we compute *mindist* with respect to the query object q instead of the origin O. Dynamic skylines can be computed in the same way except that the *mindist* of each entry in Q will be changed (each *mindist* is computed on-the-fly when the entry is considered for the first time). Assuming $f_1 = |\mu_p - \mu_q|$ and $f_2 = |\sigma_p^2 - \sigma_q^2|$, the dynamic skyline result of the query $q = (5, 1)$ in Fig. 4 is $\{g_3, g_8, g_9\}$.

5.3 Transformation and Extension

According to Section 3, the closer μ_p is to μ_q, σ_p^2 is to σ_q^2, the smaller KL-divergence is. This is analogous to a dynamic skyline query by assuming $f_1 = |\mu_p - \mu_q|, f_2 = |\sigma_p^2 - \sigma_q^2|$ (or $f_2 = |\sigma_p^2 - \sigma_q^2 - (\mu_p - \mu_q)^2|$). Hence, we transform the *KLD-Gauss*

problem to computing the dynamic skyline with respect to q.

In Ref. [23], BBS is applied directly to compute dynamic skylines. However, we note that since KL-divergence is *asymmetric* over σ_i^2, the two subspaces divided by $\sigma_p^2 = \sigma_q^2$ (or $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$) should be treated separately when we use BBS. In each dimension, we need to maintain two priority queues, and merge two result sets to obtain the final top-k objects. This is obviously inefficient and would incur extra computation cost.

Note that a straightforward method is to use iterative skyline computation according to the property that the top-1 object, based on any monotone ranking function, must be one of the skyline objects [17]. After deleting the top-1 object from our consideration, we recompute dynamic skyline objects. Then we select the top-2 object among them. Top-$3, 4, \ldots, k$ objects are computed in a similar way. This method would also lead to heavy cost.

The two concerns motivate us to propose a more efficient approach. We develop our ideas by extending the BBS algorithm. For ease of presentation, we describe our approach using $\mathcal{D}^1_{\mathrm{KL1}}$ and present necessary modifications in Section 5.5.

Because of the asymmetry of KL-divergence, we can only determine the dominance relationship between an object and each entry in the same subspace where the object locates, and cannot conclude whether it dominates any entry in the other subspace directly. For example, in Fig. 4 we can directly determine that g_8 dominates N_{12}. However, we do not know whether g_8 dominates g_7 or N_{22}. As a result, the dynamic skyline in each subspace has to be computed separately.

In order to reduce the cost caused by separate subspace computation, we consider enhancing the filtering power of each object from the subspace where it locates to the other subspace. If the dominance relationship between the two subspaces can be determined, we only need to maintain one priority queue and compute the two dynamic skylines together. To this end, for each point in one subspace, we find its counterpoint in the other subspace so that it can be used for filtering in that subspace.

This idea is confirmed by the properties of KL-divergence discussed in Section 3.1. Since KL-divergence shows monotonicity in both sides of $\sigma_p^2 = \sigma_q^2$ and is minimized at $\sigma_p^2 = \sigma_q^2$, for each point g in one subspace there must be one and only one corresponding point g' that satisfies $\mu_{g'} = \mu_g$ and $\mathcal{D}^1_{\mathrm{KL}}(g'\|q) = \mathcal{D}^1_{\mathrm{KL}}(g\|q)$. We call g' the *equal-KLD point* of g. The two equations produce $\sigma_{g'}^2 - \sigma_q^2 \ln \sigma_{g'}^2 = \sigma_g^2 - \sigma_q^2 \ln \sigma_g^2$. Because of its complexity, there is no analytical solution for $\sigma_{g'}^2$. We compute $\sigma_{g'}^2$ numerically by the *Newton method* and use its approximation (a slightly larger value to ensure correctness) instead. For example, the *equal-KLD point* of $g_8 = (6, 1.2)$ is $g'_8 = (6, 0.82)$, which can be used to filter entries such as N_{22} additionally (i.e., g'_8 dominates N_{22}).

We formally define the *equal-KLD point* as follows.

Definition 1 (*equal-KLD point*) Given a Gaussian point g, g' is the *equal-KLD point* of g, if g' satisfies $\mathcal{D}^1_{\mathrm{KL}}(g'\|q) = \mathcal{D}^1_{\mathrm{KL}}(g\|q)$.

Accordingly, we extend the definition of *dynamically dominate* to *dynamically KLD-dominate* as follows.

Definition 2 (*dynamically KLD-dominate*) Given a Gaussian point g and an entry e, g *dynamically KLD-dominates* e, if g's *equal-KLD point* g' *dynamically dominates* e.

For example, by computing g_8's *equal-KLD point* $g'_8 = (6.01, 1.1)$, we can conclude that g_8 *dynamically KLD-dominates* $g_9 = (6.4, 1.1)$ since g'_8 *dynamically dominates* g_9. As g is an *equal-KLD point* of itself, we can equally say g *dynamically KLD-dominates* e, if g *dynamically dominates* e. The relationship of g *dynamically KLD-dominates* e guarantees that the KL-divergence of g is smaller than that of any object within e.

To reduce the overhead of iterative skyline computation, we modify the way of dominance checking that the BBS algorithm does. We sort all candidates in the ascending order of KL-divergence. For each entry in the priority queue, we check the dominance relationship from the k-th candidate to the last one instead of checking all candidates as BBS does. If all entries are *dynamically KLD-dominated* by these "inferior" (from k to the last) candidates, the "superior" top-(k-1) and the first "inferior" candidates will be the final top-k result.

5.4 The Proposed Algorithm: SKY

Based on the discussion above, we propose the SKY algorithm shown in Algorithm 2. An R-tree is employed to index averages and variances of all Gaussian objects in the database. Given the constructed R-tree, a query Gaussian object q, and a constant k, the algorithm returns the top-k Gaussian objects having the smallest KL-divergences with q.

The algorithm begins from the root node and continues until all entries in the priority queue Q are processed. When checking the top entry e of Q against the candidate set S (Line 6), if $|S| < k$, we add it to S if it is an Gaussian object, and expand it in the case of an R-tree node. Otherwise (i.e., $|S| \geq k$), we sort S in the ascending order of KL-divergence, and compare e against "inferior" candidates in S, i.e., from the k-th candidate until the last one. In this way, we can avoid the expensive iterative dynamic skyline computation. We reject e if it is *dynamically KLD-dominated* by S. If not, it is either added into S with its divergence (when e is a Gaussian object) or expanded (when e is

Algorithm 2 Skyline-based query processing algorithm

```
1:  procedure KLD_Query_SKY(R-tree, q, k)
2:      Q ← (0, Root);                          ▷ Initialize Q with the Root of R-tree
3:      S ← ∅;                                  ▷ Initialize the candidate set
4:      while Q is not empty do
5:          e ← Q.top(); Q.pop();
6:          Check e against S;
7:          if e is not dynamically KLD-dominated by S then
8:              if e is a Gaussian object g then
9:                  Add g and its KL-divergence into S;
10:             else                                ▷ e is an R-tree node N
11:                 foreach child N_i do
12:                     Check N_i against S;
13:                     if N_i is not dynamically KLD-dominated by S then
14:                         Q.push(N_i, mindist(N_i));
15:                     end if
16:                 end for
17:             end if
18:         end if
19:     end while
20:     return top-k of S;
21: end procedure
```

an R-tree node). In the expansion of an R-tree node N, we check each of its child nodes N_i against S and insert N_i to Q if N_i is not *dynamically KLD-dominated* by S. Finally, the top-k candidates of S will become the result.

Another problem is that, there still will be quite a long list of entries in Q waiting for dominance checking. In order to prune non-promising entries, for each "inferior" candidate p, we derive its *maximum mindist* (*mmdist*) using the *Lagrange multiplier*.

$$\underset{\mu_p,\sigma_p^2}{\text{maximize}} \quad mmdist = |\mu_p - \mu_q| + |\sigma_p^2 - \sigma_q^2|$$

$$\text{subject to} \quad \frac{1}{2}\left[\frac{(\mu_p-\mu_q)^2}{\sigma_q^2} + \frac{\sigma_p^2}{\sigma_q^2} - \ln\frac{\sigma_p^2}{\sigma_q^2} - 1\right] = C.$$

To guide the algorithm based on *mmdist*, we further derive the following lemma (see the proof in A.2).

Lemma 2 Any entry (Gaussian object or R-tree node) whose *mindist* is larger than *mmdist* of an object g will be *dynamically KLD-dominated* by g.

According to Lemma 2, we only need to consider entries with *mindist* < *mmdist* so that the searching process can be finished early. We note that this filtering technique only works for $\mathcal{D}^1_{\mathrm{KL1}}$. In other cases, all entries in Q have to be processed.

We use the example in Fig. 4 to illustrate our algorithm in the case of $\mathcal{D}^1_{\mathrm{KL1}}$. Assume $k = 3$. After expanding the root node, $Q = \{(N_2, 0), (N_1, 0.8)\}$. Then we expand N_2 and $Q = \{(N_{21}, 0), (N_1, 0.8), (N_{22}, 3.7)\}$. Next, N_{21} is expanded and g_7, g_8, g_9 are inserted into Q. Entries in Q are $\{(N_1, 0.8), (g_8, 1.2), (g_7, 1.3), (g_9, 1.5), (N_{22}, 3.7)\}$.

After expanding N_1, Q is $\{(N_{11}, 1.1), (g_8, 1.2), (g_7, 1.3), (g_9, 1.5), (N_{12}, 2.8), (N_{22}, 3.7)\}$. Then N_{11}'s child objects g_1, g_2, g_3 are inserted into Q. Thus, $Q = \{(g_8, 1.2), (g_7, 1.3), (g_9, 1.5), (g_3, 1.7), (g_2, 2.6), (N_{12}, 2.8), (N_{22}, 3.7), (g_1, 4.5)\}$.

The top three entries g_8, g_7, g_9 with KL-divergences are added into S successively and $S = \{(g_8, 0.51), (g_7, 0.53), (g_9, 0.98)\}$. At the same time, we compute the *mmdist* of g_9, which is 4.03, and keep $\delta = 4.03$. Next, since g_3 is not *dynamically KLD-dominated* by g_9, it is inserted into S and $S = \{(g_3, 0.35), (g_8, 0.51), (g_7, 0.53), (g_9, 0.98)\}$. The *mmdist* of g_7 is $2.40 < 4.03$. Thus, δ is updated to 2.40. Since the *mindist* of g_2 is larger than δ, the algorithm stops and returns $\{(g_3, 0.35), (g_8, 0.51), (g_7, 0.53)\}$.

5.5 Extending the Skyline-based Approach

The case of $\mathcal{D}^1_{\mathrm{KL2}}$ can be processed similarly except the filtering technique based on *mmdist* cannot be applied. In the multi-dimensional case, the algorithm is the same to the one shown in Algorithm 2. A $2d$-dimensional R-tree is constructed to index d-dimensional Gaussian objects in the database. The following definitions and computations will replace their counterparts in the one-dimensional case: (1) $mindist = \sum_{i=1}^{d}(|\mu_{p,i}-\mu_{q,i}|+|\sigma^2_{p,i}-\sigma^2_{p,i}|)$ (2) Compute *equal-KLD point* based on $\mathcal{D}^d_{\mathrm{KL1}}$ or $\mathcal{D}^d_{\mathrm{KL2}}$. (3) An object g *dynamically KLD-dominates* an entry e, if g's *equal-KLD point* g' *dynamically dominates* e in all dimensions.

5.6 Application to the PTA Algorithm

As discussed in Section 4.2.2, in each dimension PTA accesses the top objects with the smallest $\mathcal{D}^i_{\mathrm{KL}}$ using the skyline-based method SKY. Since PTA performs multiple sorted accesses in each dimension, we need to maintain all dominated entries instead of rejecting them as SKY does. We associate each "inferior" candidate with all entries dominated by it, and release them for further processing when this candidate becomes "superior" in the next sorted access.

Table 5 A two-dimensional example dataset.

g_i	$\mu_{i,1}$	$\sigma^2_{i,1}$	$\mu_{i,2}$	$\sigma^2_{i,2}$
g_1	6.0	0.8	5.8	2.0
g_2	6.0	3.0	6.4	1.6
g_3	8.2	1.2	3.1	1.6
g_4	2.5	3.4	4.7	0.9

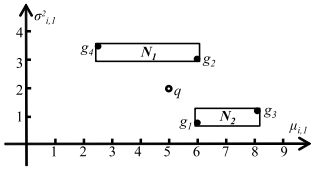


Fig. 6 R-tree image ($d = 1$).

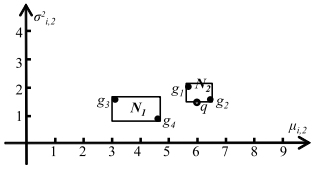


Fig. 7 R-tree image ($d = 2$).

We use the following example two-dimensional dataset shown in **Table 5** to illustrate the PTA algorithm. We construct an R-tree in each dimension. Their R-tree images in the first dimension and the second dimension are shown in **Fig. 6** and **Fig. 7**, respectively.

Consider the same query $q = (5.0, 2.0; 6.0, 1.5)$ with $k = 2$. Assume that we retrieve two objects in each dimension. At first, in the first dimension after expanding the root node in Fig. 6, we obtain $Q_1 = \{(N_1, 1.0), (N_2, 1.8)\}$ where Q_1 is a priority queue maintaining entries (R-tree nodes and Gaussian objects) to be processed in the ascending order of their *mindist*s from q. Then we expand N_1 and insert its two child Gaussian objects, g_2 and g_4, into $Q_1 = \{(N_2, 1.8), (g_2, 2.0), (g_4, 3.9)\}$. Next, we expand N_2 in the same way and obtain $Q_1 = \{(g_2, 2.0), (g_1, 2.2), (g_4, 3.9), (g_3, 4.0)\}$. The next entries, g_2 and g_1, are added into the candidate set S_1 successively with their KL-divergences in this dimension, $S_1 = \{(g_2, 0.297), (g_1, 0.408)\}$. At the same time, we compute the *mmdist* of g_1 and assign it to $\delta_1 = 3.904$. Since the *mindist* of the next entry g_4 is smaller than δ_1, i.e., $3.9 < 3.904$, we continue to check g_4. When checking it against S_1, we find that g_4 is *dynamically KLD-dominated* by g_1. Hence, we add it to the dominating list of g_1, and obtain $S_1 = \{(g_2, 0.297), (g_1, 0.408, \{(g_4, 3.9)\})\}$. At this time, we have $Q_1 = \{(g_3, 4.0)\}$. Since the *mindist* of the next entry g_3 is larger than δ_1, based on Lemma 2 we stop searching and return g_2 and g_1 with the second smallest $D^1_{KL} = 0.408$.

In the second dimension, we calculate the smallest D^2_{KL} in the same way as D^1_{KL} in the first dimension. After expanding the root node in Fig. 7, we obtain $Q_2 = \{(N_2, 0.1), (N_1, 1.3)\}$. Then we expand N_2 and obtain $Q_2 = \{(g_2, 0.5), (g_1, 0.7), (N_1, 1.3)\}$. Next, we add g_2 and g_1 into S_2 successively with their KL-divergences in

this dimension, $S_2 = \{(g_1, 0.036), (g_2, 0.054)\}$. At the same time, we compute the *mmdist* of g_2 and assign it to $\delta_2 = 0.897$. Since the *mindist* of the next entry in $Q_2 = \{(N_1, 1.3)\}$ is larger than δ_2, again based on Lemma 2 we stop searching and return g_1 and g_2 with the second smallest $D^2_{KL} = 0.054$.

What follows is that the algorithm will compute the overall KL-divergences of g_2 and g_1 in the two dimensions (0.35 and 0.44), respectively, and update $\tau = D^1_{KL} + D^2_{KL} = 0.462$. Since the second smallest KL-divergence 0.44 is larger than τ, we first release $(g_4, 3.9)$ from S_1 to Q_1 and delete the entries of g_1 and g_2 from S_1 and S_2, and then repeat the same process in the next round until the kth smallest KL-divergence is no larger than τ.

6. Experiments

6.1 Experimental Setup

We generate both data Gaussian distributions and query Gaussian distributions randomly under the same setting. Each average value is generated from $(0, 1000)$, and each variance value is generated from $(0, 100)$. The parameters tested in our experiments and their default values (in bold) are summarized in **Table 6**. To test the effect of data distribution, we also generate three datasets with independent, correlated, and anti-correlated distributions, respectively, using the standard skyline data generator in Ref. [6].

We compare our TA-based algorithms CTA, PTA and skyline-based algorithm SKY, with the sequential scan methods SS and the extended BBS algorithm BBS. In each dimension i ($1 \le i \le d$), BBS processes objects in the two subspaces divided by $\sigma^2_{p,i} = \sigma^2_{q,i}$ (or $\sigma^2_{p,i} = \sigma^2_{q,i} + (\mu_{p,i} - \mu_{q,i})^2$) separately. In other word, it maintains 2^d priority queues and compute each top object by merging candidate objects from these priority queues.

During the preprocessing, we build a $2d$-dimensional R-tree for SKY and BBS. Each Gaussian distribution is inserted as a $2d$-dimensional point, consisting of the d-dimensional average vector and d-dimensional variance vector. For PTA, we construct d two-dimensional R-trees. Each two-dimensional R-tree maintains the average and variance value in the i-th dimension ($1 \le i \le d$), and provides *sorted access* to the ranked $\mathcal{D}^i_{\mathrm{KL}}$ and object id (see Section 4). *Random access* is supported by an Gaussian object list in the order of object id. All R-trees and lists are stored in the secondary memory.

In each experiment, we run 100 queries and use the average runtime for performance evaluation. We do not consider I/O access because the system tends to load all indices into main memory upon reading and the runtime is mainly spent on CPU. All experiments are conducted using a workstation with Intel Xeon(R) CPU E3-1241 v3 (3.50 GHz), RAM 16 GB, running Ubuntu 12.10 LTS.

6.2 Performance Evaluation

We first compare SKY with BBS. While SKY takes about 0.02 seconds using both the first type and the second type of KL-divergence (called KLD1 and KLD2 afterwards), BBS takes 2.45 seconds for KLD1, and 0.23 seconds for KLD2. This demonstrates the effectiveness of our extension and modification to the basic BBS algorithm. Because of the uneven property of KLD2 shown in Fig. 3, most of objects are assigned to the same priority queue. Consequently, BBS spends less time using KLD2 than that using KLD1. We leave BBS out of our consideration in the following experiments because of its clear inefficiency.

Table 6 Parameters for testing.

Parameter	Testing Range
k	1, **10**, 20, 30, 40, 50, 60, 70, 80, 90, 100
data size	1 k, 10 k, 100 k, **1000 k**, 10000 k
dimension	1, **2**, 3, 4, 5
distribution	**independent**, correlated, anti-correlated

6.2.1 Effect of k

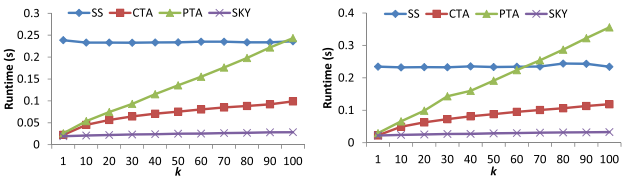
Intuitively, a larger k incurs more cost, since retrieving more objects potentially leads to more computations. The constantly increasing runtime of PTA, CTA and SKY shown in **Fig. 8** follows this intuition. SS is an exception of this intuition since it computes KL-divergence for all data Gaussian objects regardless of k. CTA and SKY outperform SS over all k, and PTA performs better than SS when $k < 100$ for KLD1 ($k < 60$ for KLD2). The difference is that PTA displays the most significant increase while the runtime of CTA rises more slowly, and SKY still keeps the low runtime as k increases. Even when k is 100, the runtime of SKY is only 12% of that of SS.

That's because the TA-based approaches, CTA and PTA, retrieve many objects iteratively based on their partial information in each dimension (average, variance or KL-divergence), and they need to retrieve much more objects with a larger k. On the other hand, SKY retrieves less objects based on their full information in all dimensions and thus costs less runtime. Especially, PTA needs to maintain all intermediate entries and compute the dynamic skyline to retrieve candidate objects in each dimension. Thus, it takes even more runtime than CTA, which retrieves candidates more easily by sorted access to presorted lists.

6.2.2 Effect of Data Size

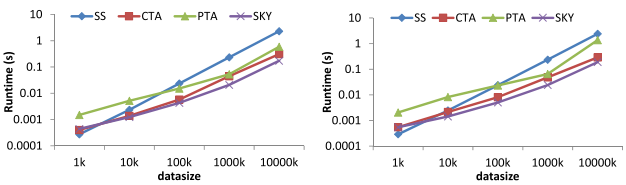
Figure 9 shows the scalability of the four approaches. All of them consume more runtime with a larger data size. SS exhibits the worst scalability to large scale datasets, while SKY demonstrates the best. The performance of PTA and CTA is between them with that CTA performs better.

When the data size increases, the density of objects in the space becomes larger. This means that there will be more objects with



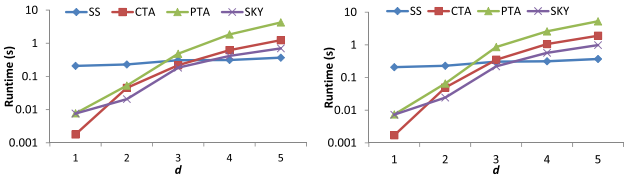
(a) KLD1 (b) KLD2

Fig. 8 Effect of k.



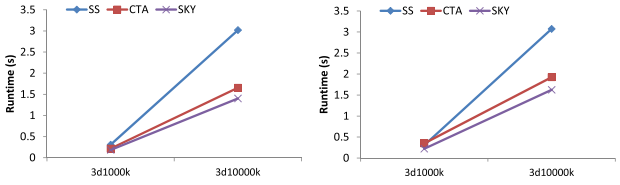
(a) KLD1 (b) KLD2

Fig. 9 Effect of data size.



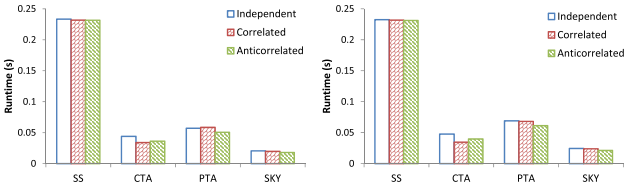
(a) KLD1 (b) KLD2

Fig. 10 Effect of dimension.



(a) KLD1 (b) KLD2

Fig. 11 Improvement with increasing data size.



(a) KLD1 (b) KLD2

Fig. 12 Effect of data distribution.

similar averages, variances or KL-divergences in one dimension but different in other dimensions. As a result, PTA and CTA suffer more from ineffective retrievals and thus perform worse than SKY, which has better scalability due to its more effective retrieval by R-tree. Especially, SKY and CTA outperform SS when the data size is larger than 10 k and are much better with the increasing data size.

6.2.3 Effect of Dimensionality

As shown in **Fig. 10**, dimensionality affects the proposed approaches greatly, while it has little effect on SS. Since PTA and SKY both utilize R-trees for indexing and the performance of R-tree degrades with the increasing dimension d, their runtime rises very fast. The runtime of PTA rises even faster since in each dimension it retrieves objects based on partial information and this results in much more ineffective retrievals, i.e., it retrieves much more unpromising objects. CTA spends more runtime with increasing d due to the similar reason as PTA that it does much more ineffective retrievals.

When d is larger than 3, they deteriorate significantly and are defeated by SS. This indicates that our proposed approaches are more efficient in low dimensions less than 4. Moreover, their performance will be improved with a larger data size. For example, as shown in **Fig. 11**, when $d = 3$, the advantage of CTA and SKY over SS is more obvious when the data size increases from 1000 k to 10000 k.

6.2.4 Effect of Data Distribution

In **Fig. 12**, we show the effect of data distribution on the four approaches. Generally, the runtime of all approaches does not vary greatly over the three different distributions. CTA shows the most apparent decrease over correlated and anti-correlated distributions since it depends directly on the average or variance of each object in each dimension. Correlations mean that objects are concentrated in the center and thus decrease the number of ineffective retrieval on average. It is clear that SKY has the best performance. This demonstrates the capability of our proposed approaches over different data distributions.

6.2.5 Index Construction

As preprocessing, we build R-trees and lists to support efficient query processing. In each experiment, we build a $2d$-dimensional R-tree for SKY and BBS, and d two-dimensional R-trees for PTA. When using the default dataset ($d = 2$, data size = 1000 k), the index construction time is about 5 seconds for building a four-dimensional R-tree, and is about 7 seconds for building 2 two-dimensional R-trees. The lists include an object list for random access in CTA and PTA, and d sorted lists for sorted access in CTA and PTA. The time for building these two kinds of lists using the default dataset is about 1 second and 9 seconds, respectively. When the data size or d varies, the construction time varies proportionally. In view of the performance improvement shown in previous sections, the preprocessing cost of our proposed approaches is rather low.

7. Related Work

In this work, we proposed query processing approaches for top-k similarity search over Gaussian distributions based on KL-divergence, a *non-metric* similarity measure. Skopal et al. [28] surveyed the domains employing non-metric functions for similarity search, and methods for efficient and effective non-metric similarity search. To improve the searching efficiency, a class of approaches adopt the indexing strategy based on analytic properties. They analyze the properties of a specific similarity function, and develop special access methods for that function. Another advantage is their ability to support both exact and approximate search. Our work falls into this category and supports exact search.

Other approaches are based on statistical methods. For the efficiency, they perform costly preprocessing by suitably clustering or transforming the original database into another space such as the metric space based on the distance distribution statistics, so that existing metric indexing methods can be employed [7], [27], [28]. One drawback is that they cannot provide exact results. Furthermore, expensive preprocessing is often needed prior to the indexing itself. On the contrary, our proposed approaches have low preprocessing cost and support exact similarity search.

As a general class of similarity measures including KL-divergence, *Bregman divergence* has also led to a stream of research work to develop various algorithms. For example, [3] proposed clustering approaches with Bregman divergence. Zhang et al. [30] developed a prune-and-refine-based searching method for Bregman divergence, which is close to our work, but it works only for discrete probability distributions (described by d-dimensional vectors) in finite domains. Moreover, they considered only the first type of asymmetric divergences.

In Ref. [5], an index structure called *Gauss-tree* was proposed for similarity search over multivariate Gaussian distributions. They also assumed non-correlated Gaussian distributions. Al-

though their problem is very similar to ours, they defined a different similarity measure as follows. For a d-dimensional Gaussian distribution g, they used $N_{\mu_{g,j},\sigma_{g,j}}(x_j)$ to represent its probability density function in each dimension j, which is a one-dimensional Gaussian distribution with two parameters, the average $\mu_{g,j}$ and the variance $\sigma^2_{g,j}$. Given a database DB and a query Gaussian distribution q, the similarity of g in DB and q is defined as:

$$P(g|q) = \frac{p(q|g)}{\sum_{w \in DB} p(q|w)} \tag{10}$$

where

$$\begin{aligned} p(q|g) &= \prod_{j=1}^{d} \int_{-\infty}^{+\infty} N_{\mu_{q,j},\sigma_{q,j}}(x_j) \cdot N_{\mu_{g,j},\sigma_{g,j}}(x_j) dx \\ &= \prod_{j=1}^{d} N_{\mu_{q,j},\sigma_{q,j}+\sigma_{g,j}}(\mu_{g,j}) \end{aligned} \tag{11}$$

Here, $p(q|g)$ represents the probability density for observing q under the condition that we already observed g. The conditions that maximize Eq. (11) are $\mu_{g,j} = \mu_{q,j}$ and $\sigma_{g,j} \to 0$ not $\sigma^2_{g,j} = \sigma^2_{q,j}$. Hence, we think Eq. (10) is not a proper similarity measure for two Gaussian distributions.

8. Conclusion and Future Work

In this work, assuming that large scale data is modeled by Gaussian distributions, we study the problem of similarity search over non-correlated Gaussian distributions based on KL-divergence. We analyzed the mathematical properties of KL-divergence of Gaussian distributions. Based on the analysis, we proposed two types of approaches to efficiently and effectively process top-k similarity search over Gaussian distributions, which returns the k most similar ones to a given query Gaussian distribution. They utilize the notions of *rank aggregation* and *skyline queries*, respectively. We demonstrated the efficiency and effectiveness of our approaches through a comprehensive experimental performance study.

As we can see from the experimental results in Section 6, our approaches spend more preprocessing time for index construction (5–10 seconds) than the query processing time (less than 1 second) of the naïve approach SS. Thus, in the future, we plan to improve index structures to reduce preprocessing cost. Furthermore, we will study the similarity search problem in the general case of multi-dimensional Gaussian distributions and use similarity measures other than KL-divergence.

Acknowledgments This research was partly supported by the Grant-in-Aid for Scientific Research (#25280039, #26540043) from JSPS.

References

[1] Aggarwal, C.C.: *Managing and Mining Uncertain Data*, Springer (2009).
[2] Agrawal, P., Benjelloun, O., Sarma, A.D., Hayworth, C., Nabar, S.U., Sugihara, T. and Widom, J.: Trio: A System for Data, Uncertainty, and Lineage, *VLDB*, pp.1151–1154 (2006).
[3] Banerjee, A., Merugu, S., Dhillon, I.S. and Ghosh, J.: Clustering with Bregman Divergences, *Journal of Machine Learning Research*, Vol.6, pp.1705–1749 (2005).
[4] Bishop, C.M.: *Pattern Recognition and Machine Learning*, Springer (2006).
[5] Böhm, C., Pryakhin, A. and Schubert, M.: The Gauss-Tree: Efficient Object Identification in Databases of Probabilistic Feature Vectors, *ICDE* (2006).
[6] Börzsönyi, S., Kossmann, D. and Stocker, K.: The Skyline Operator, *ICDE*, pp.421–430 (2001).
[7] Chen, L. and Lian, X.: Efficient Similarity Search in Nonmetric Spaces with Local Constant Embedding, *IEEE TKDE*, Vol.20, No.3, pp.321–336 (2008).
[8] Ciaccia, P., Patella, M. and Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces, *VLDB*, pp.426–435 (1997).
[9] Contreras-Reyes, J.E. and Arellano-Valle, R.B.: Kullback-Leibler Divergence Measure for Multivariate Skew-Normal Distributions, *Entropy*, Vol.14, No.9, pp.1606–1626 (2012).
[10] Cover, T.M. and Thomas, J.A.: *Elements of Information Theory*, Wiley (2006).
[11] Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J.M. and Hong, W.: Model-based approximate querying in sensor networks, *The VLDB Journal*, Vol.14, No.4, pp.417–443 (2005).
[12] Do, M.N. and Vetterli, M.: Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance, *IEEE Trans. Image Processing*, Vol.11, No.2, pp.146–158 (2002).
[13] Duda, R.O., Hart, P.E. and Stork, D.G.: *Pattern Classification*, Wiley-Interscience (2000).
[14] Fagin, R., Lotem, A. and Naor, M.: Optimal Aggregation Algorithms for Middleware, *PODS* (2001).
[15] Faloutsos, C., Ranganathan, M. and Manolopoulos, Y.: Fast Subsequence Matching in Time-Series Databases, *ACM SIGMOD*, pp.419–429 (1994).
[16] Husmeier, D., Dybowski, R. and Roberts, S.: *Probabilistic Modelling in Bioinformatics and Medical Informatics*, Springer-Verlag (2005).
[17] Ilyas, I.F., Beskales, G. and Soliman, M.A.: A survey of top-k query processing techniques in relational database systems, *ACM Comput. Surv.*, Vol.40, No.4, pp.11:1–11:58 (2008).
[18] Jagadish, H.V., Ooi, B.C., Tan, K.-L., Yu, C. and Zhang, R.: iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search, *ACM TODS*, Vol.30, No.2, pp.364–397 (2005).
[19] Kullback, S. and Leibler, R.A.: On Information and Sufficiency, *Ann. Math. Statist.*, Vol.22, No.1, pp.79–86 (1951).
[20] Mandel, M.I. and Ellis, D.: Song-Level Features and Support Vector Machines for Music Classification, *Proc. 6th Int'l Conf. on Music Information Retrieval* (*ICMIR 2005*), pp.594–599 (2005).
[21] Mehrotra, R. and Gary, J.E.: Feature-Based Retrieval of Similar Shapes, *ICDE*, pp.108–115 (1993).
[22] Miyajima, C., Nishiwaki, Y., Ozawa, K., Wakita, T., Itou, K., Takeda, K. and Itakura, F.: Driver Modeling Based on Driving Behavior and Its Evaluation in Driver Identification, *Proc. IEEE*, pp.427–437 (2007).
[23] Papadias, D., Tao, Y., Fu, G. and Seeger, B.: Progressive skyline computation in database systems, *ACM TODS*, Vol.30, No.1, pp.41–82 (2005).
[24] Penny, W.D.: KL-Divergences of Normal, Gamma, Dirichlet and Wishart densities, Technical report, Wellcome Department of Cognitive Neurology, University College London (2001).
[25] Rosti, A.-V.I.: Linear Gaussian models for speech recognition, PhD Thesis, Cambridge University (2004).
[26] Schnitzer, D., Flexer, A. and Widmer, G.: A Filter-and-Refine Indexing Method for Fast Similarity Search in Millions of Music Tracks, *Proc. 10th Int'l Conf. on Music Information Retrieval* (*ICMIR 2009*), pp.537–542 (2009).
[27] Skopal, T.: On Fast Non-metric Similarity Search by Metric Access Methods, *EDBT*, pp.718–736 (2006).
[28] Skopal, T. and Bustos, B.: On nonmetric similarity search problems in complex domains, *ACM Comput. Surv.*, Vol.43, No.4, pp.34:1–34:50 (2011).
[29] Suciu, D., Olteanu, D., Ré, C. and Koch, C.: *Probabilistic Databases*, Morgan & Claypool (2011).
[30] Zhang, Z., Ooi, B.C., Parthasarathy, S. and Tung, A.K.H.: Similarity Search on Bregman Divergence: Towards Non-Metric Indexing, *PVLDB*, Vol.2, No.1, pp.13–24 (2009).

Appendix

A.1 Proof of Lemma 1

A.1.1 Case 1: $\mathcal{D}_{\mathrm{KL}}(p||q)$

Assume $\sigma^2_{p,j} - \sigma^2_{q,j} = C_1$ and $\sigma^2_{q,j} - \sigma^2_{p',j} = C_2$, i.e. $\sigma^2_{p,j} = \sigma^2_{q,j} + C_1$ and $\sigma^2_{p',j} = \sigma^2_{q,j} - C_2$. Then $0 < C_1 \leq C_2 < \sigma^2_{q,j}$.

Let $\Delta_1 = \mathcal{D}_{\mathrm{KL}}(p||q) - \mathcal{D}_{\mathrm{KL}}(p'||q)$. Then

$$\Delta_1 = \frac{1}{2}\ln\frac{\sigma_{q,j}^2 - C_2}{\sigma_{q,j}^2 + C_1} + \frac{C_1 + C_2}{2\sigma_{q,j}^2},$$
$$\frac{\partial\Delta_1}{\partial\sigma_{q,j}^2} = \frac{(C_1 + C_2)[(C_2 - C_1)\sigma_{q,j}^2 + C_1C_2]}{2\sigma_{q,j}^4(\sigma_{q,j}^2 - C_2)(\sigma_{q,j}^2 + C_1)} > 0.$$

Since $\Delta_1|_{\sigma_{q,j}^2\to\infty} = 0$, $\Delta_1 < 0$ holds for all $\sigma_{q,j}^2$, i.e., $\mathcal{D}_{\mathrm{KL}}(p\|q) < \mathcal{D}_{\mathrm{KL}}(p'\|q)$.

A.1.2 Case 2: $\mathcal{D}_{\mathrm{KL}}(q\|p)$

Since $|\mu_{p,j} - \mu_{q,j}| = |\mu_{p',j} - \mu_{q,j}|$, we use $|\mu_{p,j} - \mu_{q,j}|$ to represent both of them. Assume $\sigma_{p,j}^2 - \sigma_{q,j}^2 - (\mu_{p,j} - \mu_{q,j})^2 = C_1$ and $\sigma_{q,j}^2 - \sigma_{p',j}^2 - (\mu_{p,j} - \mu_{q,j})^2 = C_2$, i.e., $\sigma_{p,j}^2 = \sigma_{q,j}^2 + (\mu_{p,j} - \mu_{q,j})^2 + C_1$ and $\sigma_{p',j}^2 = \sigma_{q,j}^2 + (\mu_{p,j} - \mu_{q,j})^2 - C_2$. Then $0 < C_1 \le C_2 < \sigma_{q,j}^2 + (\mu_{p,j} - \mu_{q,j})^2$.

Let $\Delta_2 = \mathcal{D}_{\mathrm{KL}}(q\|p) - \mathcal{D}_{\mathrm{KL}}(q\|p')$ and $\alpha = \sigma_{q,j}^2 + (\mu_{p,j} - \mu_{q,j})^2$. Then

$$\Delta_2 = \frac{1}{2}\ln\frac{\alpha + C_1}{\alpha - C_2} - \frac{\alpha(C_1 + C_2)}{2(\alpha + C_1)(\alpha - C_2)},$$
$$\frac{\partial\Delta_2}{\partial\sigma_{q,j}^2} = \frac{(C_1 + C_2)[\alpha(C_2 - C_1) + 2C_1C_2]}{2(\alpha - C_2)^2(\alpha + C_1)^2} > 0.$$

Since $\Delta_2|_{\sigma_{q,j}^2\to\infty} = 0$, $\Delta_2 < 0$ holds for all $\sigma_{q,j}^2$, i.e., $\mathcal{D}_{\mathrm{KL}}(q\|p) < \mathcal{D}_{\mathrm{KL}}(q\|p')$.

□

A.2 Proof of Lemma 2

Let the *mmdist* of g be $mmdist_g$. Due to the definition of *mmdist*, for any *equal-KLD point* g' of g, $mmdist_g \ge |\mu_{g'} - \mu_q| + |\sigma_{g'}^2 - \sigma_q^2|$ holds. Since $mindist_e$, the *mindist* of an entry e, satisfies $mindist_e = |\mu_e - \mu_q| + |\sigma_e^2 - \sigma_q^2| > mmdist_g$, we have

$$|\mu_e - \mu_q| + |\sigma_e^2 - \sigma_q^2| > |\mu_{g'} - \mu_q| + |\sigma_{g'}^2 - \sigma_q^2|. \tag{A.1}$$

Given q and the KLD of g and q, and assuming

$$\mathcal{D}_{\mathrm{KL}}^1(g\|q) = \frac{1}{2}\left[\frac{(\mu_g - \mu_q)^2}{\sigma_q^2} + \frac{\sigma_g^2}{\sigma_q^2} - \ln\frac{\sigma_g^2}{\sigma_q^2} - 1\right] = C,$$

when $\sigma_g^2 = \sigma_q^2$, $(\mu_g - \mu_q)^2$ takes the maximum $2C\sigma_q^2$.

The reason is as follows. We can easily find that the function $f(x) = x - \ln x$ takes the minimum when $x = 1$. Therefore, $\frac{\sigma_g^2}{\sigma_q^2} - ln\frac{\sigma_g^2}{\sigma_q^2}$ takes the minimum when $\frac{\sigma_g^2}{\sigma_q^2} = 1$. In other words, when $\sigma_g^2 = \sigma_q^2$, $(\mu_g - \mu_q)^2$ is the maximum. We prove the lemma in the following two cases.

(1) $(\mu_e - \mu_q)^2 > 2C\sigma_q^2$:

Consider g's *equal-KLD point* g' satisfying $|\mu_{g'} - \mu_q| = \sqrt{2C}\sigma_q$, and $\sigma_{g'}^2 = \sigma_q^2$. Since $|\mu_e - \mu_q| > \sqrt{2C}\sigma_q = |\mu_{g'} - \mu_q|$, and $|\sigma_e^2 - \sigma_q^2| \ge 0 = |\sigma_{g'}^2 - \sigma_q^2|$, it is obvious that e is *dynamically dominated* by g', i.e., e is *dynamically KLD-dominated* by g.

(2) Otherwise:

Consider g's *equal-KLD point* g' satisfying $\mu_{g'} = \mu_e$. According to Eq. (A.1), we have $|\sigma_e^2 - \sigma_q^2| > |\sigma_{g'}^2 - \sigma_q^2|$. In other words, $|\mu_e - \mu_q| = |\mu_{g'} - \mu_q|$, and $|\sigma_e^2 - \sigma_q^2| > |\sigma_{g'}^2 - \sigma_q^2|$. Therefore, e is *dynamically dominated* by g', i.e., e is *dynamically KLD-dominated* by g.

□

Tingting Dong is a Ph.D. candidate in Graduate School of Information Science, Nagoya University. She received her M.S. degree from Nagoya University in 2013, and both B.S. and B.E. degrees from Dalian Jiaotong University, China in 2010. Her research interests include probabilistic databases, spatio-temporal databases, and sensor databases.

Yoshiharu Ishikawa is a professor in Graduate School of Information Science, Nagoya University. His research interests include spatio-temporal databases, mobile databases, sensor databases, data mining, information retrieval, and Web information systems. He is a member of the Database Society of Japan, IPSJ, IEICE, JSAI, ACM, and IEEE.

Chuan Xiao is an assistant professor in Institute for Advanced Research, Nagoya University. He received bachelor's degree from Northeastern University, China in 2005, and Ph.D. degree from The University of New South Wales in 2010. His research interests include similarity search, textual databases, and graph databases.