Top-k Dominating Queries on Incomplete Data

Xiaoye Miao, Yunjun Gao, *Member, IEEE,* Baihua Zheng, *Member, IEEE,* Gang Chen, Huiyong Cui

Abstract—The *top-k dominating* (TKD) *query* returns the k objects that dominate the *maximum* number of objects in a given dataset. It combines the advantages of skyline and top-k queries, and plays an important role in many decision support applications. Incomplete data exists in a wide spectrum of real datasets, due to device failure, privacy preservation, data loss, and so on. In this paper, for the first time, we carry out a systematic study of *TKD queries on incomplete data*, which involves the data having some missing dimensional value(s). We formalize this problem, and propose a suite of efficient algorithms for answering TKD queries over incomplete data. Our methods employ some *novel* techniques, such as *upper bound score pruning*, *bitmap pruning*, and *partial score pruning*, to boost query efficiency. Extensive experimental evaluation using both real and synthetic datasets demonstrates the effectiveness of our developed pruning heuristics and the performance of our presented algorithms.

Index Terms—Top-k dominating query, Incomplete data, Query processing, Dominance relationship, Algorithm

1 Introduction

Given a set S of d-dimensional objects, *top-k dominating* (TKD) *query* ranks the objects o in S based on the number of the objects in S dominated by o, and returns the k objects from S that dominate the *maximum* number of objects. Here, an object o dominates another object o', if o is no worse than o' in all dimensions, and is better than o' in at least one dimension. Since the TKD query identifies the most significant objects in an intuitive way, it is a powerful decision making tool to rank objects in many real life applications. Take the typical *MovieLens* dataset from a movie recommender system (http://www.imdb.com/) as an example. *MovieLens* includes a group of movies with the ratings from audiences, where every movie is represented as a multi-dimensional object with each dimension corresponding to a rating in the range of [1, 5] from an audience. Typically, a higher rating indicates a better recognition. As an example, given two movies o_1 = (5, 3, 4) and o_2 = (3, 3, 2), we understand that there are three audiences scoring o_1 and o_2, where the first audience (w.r.t. the first dimension) scores o_1 and o_2 as 5 and 3 respectively, the second audience (w.r.t. the second dimension) scores both o_1 and o_2 as 3, and the third audience (w.r.t. the third dimension) scores o_1 and o_2 as 4 and 2 respectively. Hence, among three audiences, both the first and the third audiences think o_1 is better than o_2, and the second audience thinks they are equally good. According to the dominance definition, it can derive that, o_1 dominates o_2, meaning that no audience rates o_2 higher than o_1. Thus, if a movie dominates many other movies, it is very likely that the movie is rather popular. Note that, *MovieLens* dataset is widely utilized in many previous works, including [1], [2]. Intuitively, a TKD query could identify the k most popular movies for moviegoers. Because of its large application base, the TKD query has received lots of attention from the database community [3], [4], [5], [6], [7], [8], [9]. Nonetheless, we would like to highlight that existing work related to this query only focuses on *complete* data or *uncertain* data.

In a real movie recommender system, it is very common that the ratings from some users are *missing*, because a user tends to only rate those movies he/she knows. As a result, each movie is denoted as a multi-dimensional object with some *blank* (i.e., *incomplete*) dimensions. Therefore, the set of movie ratings is incomplete. As shown in Fig. 1, since the audience a_2 watches the last three movies m_2, m_3, and m_4 but not the first one m_1, i.e., *Schindler's List (1993)*, a_2 only rates movies m_2, m_3, and m_4. Data incompleteness is universal, and querying incomplete data has become more and more important recently. It has also triggered lots of efforts in the database community, including incomplete data model [10], [11], [12], query evaluation [13], indexing [14], [15], skyline computation [1], [2], [16], similarity search [17], top-k retrieval [18], [19], etc.

Here, we would like to point out the difference between incomplete data and uncertain data. In this paper, as illustrated in Fig. 1, we treat a data object with missing value(s) as an incomplete data object, which follows the model introduced in [1] that requires *zero prior knowledge* of missing dimensional value(s). On the other hand, for uncertain data, the uncertainty of missing data value(s) is usually expressed in terms of *probabilities* or is derived by some probability distributions. Normally, these probabilities have been specified in an original dataset. As pointed out explicitly in [20], "*missingness is the state of being missing*" for missing data, and the missingness implies "*a static state, not fluid or probabilistic one*". Thus, the incomplete data model (the one our work is based on) and the probabilistic concept (the one used by uncertain data) are two approaches for handling missing data. It is worth mentioning that, compared with uncertain data model, incomplete data model has one significant advantage, i.e., it does not require any assumption on data correlation or prior knowledge.

- *X. Miao, Y. Gao, G. Chen, and H. Cui are with the College of Computer Science, Zhejiang University, 38 Zheda Road, Hangzhou 310027, China. E-mail: {miaoxy, gaoyj, cg, cuihy}@zju.edu.cn.*
- *Y. Gao and G. Chen are with the Key Laboratory of Big Data Intelligent Computing of Zhejiang Province, Zhejiang University, Hangzhou, China.*
- *B. Zheng is with the School of Information Systems, Singapore Management University, Singapore 178902, Singapore. E-mail: bhzheng@smu.edu.sg.*

ID	Film Name	Film Ratings from Audiences				
		a_1	a_2	a_3	a_4	a_5
m_1	Schindler's List (1993)	–	–	3	4	2
m_2	The Godfather (1972)	5	3	4	–	–
m_3	The Silence of the Lambs (1991)	–	2	1	5	3
m_4	Star Wars (1977)	3	1	5	3	4

Fig. 1: Example of a movie recommender system

In this paper, we consider an incomplete dataset where some objects face the missing of attribute values in some dimensions, and study the problem of *TKD query processing over incomplete data*. In particular, a TKD query on incomplete data returns the k objects that dominate the maximum number of objects from a given incomplete data set. Note that, the definition of the *dominance relationship* over incomplete data follows the definition presented in [1]. Specifically, an object o dominates another object o', denoted as $o \prec o'$, if o is no worse than o' in all common observed dimensions, and o is better than o' in at least one common observed dimension. To facilitate the presentation, we define a function score(o) that counts the number of the objects dominated by object o. Consider the four movies listed in Fig. 1 as an example dataset S, i.e., $S = \{m_1, m_2, m_3, m_4\}$. Movie m_2 dominates movie m_3. This is because, on the two common observed dimensions 2 and 3, $m_2.[2] > m_3.[2]$ and $m_2.[3] > m_3.[3]$. Thus, we can get the score of m_2, i.e., score(m_2) = $|\{m_i \in S | m_2 \prec m_i\}|$ = $|\{m_1, m_3\}|$ = 2. Similarly, we have score(m_1) = 0, score(m_3)= 0, and score(m_4) = $|\{m_1\}|$= 1. Given a T1D (k = 1) query on the dataset S, movie m_2 is returned for its largest score value.

At first glance, TKD queries on incomplete data share some similarities with the skyline operator over incomplete data [1], since they both are based on the same dominance definition. However, we would like to highlight that TKD queries on incomplete data have a desirable advantage, i.e., its output is *controllable* via a parameter k, and hence, it is invariable to the scale of the incomplete dataset in different dimensions. In addition, we want to emphasize the dominance relationship definition on incomplete data is actually *meaningful*. Take movies m_1 and m_2 in the recommender system depicted in Fig. 1 as an example. The audiences a_1 and a_2 only rate m_2 but not m_1, whereas the audiences a_4 and a_5 only rate m_1 but not m_2. Thus, we cannot determine the dominance relationship between m_1 and m_2 according to the rates from audiences a_1, a_2, a_4, and a_5. On the other hand, based on audience a_3, m_2 is better than m_1 as he/she gives a higher score to m_2 compared with m_1. To sum up, for the two movies m_1 and m_2, one audience ranks m_2 higher than m_1 while none of audiences ranks m_2 lower than m_1. Therefore, we argue that m_2 is liked by more audiences, and hence, it deserves a stronger recommendation compared with m_1.

To the best of our knowledge, this is the first attempt to explore the TKD query on incomplete data. Although the TKD query over complete data or uncertain data has been well studied, TKD query processing on incomplete data still remains a big challenge. This is because existing techniques [3], [4], [5], [6], [7], [8] cannot be applied to handle the TKD query over incomplete data efficiently. Specifically, the R-tree/aR-tree and the transitivity of dominance relationship used in traditional and uncertain databases are not directly applicable to incomplete data. It is partially because R-tree/aR-tree could not be built on incomplete data directly, since the MBRs of tree nodes do not exist due to the missing dimensional values of data objects. Also, the transitivity of dominance relationship does not hold for incomplete data. In addition, the probability model of uncertain TKD queries is different from our model as mentioned earlier. Consequently, new efficient algorithms catered for incomplete data are desired.

An intuitive method for supporting the TKD query on incomplete data is to conduct exhaustive pairwise comparisons among the whole dataset to get the score of every object o, i.e., the number of the objects dominated by o, and to return the k objects with the highest scores. Clearly, this approach is *inefficient*, due to the extremely large size of the candidate set and the expensive cost of brute-force based score computation. Hence, in this paper, we first propose two algorithms, namely, *extended skyband based* (ESB) *algorithm* using local skyband technique and *upper bound based* (UBB) *algorithm* using upper bound score pruning, to effectively reduce the candidate set. Also, we present *bitmap index guided* (BIG) *algorithm*, which calculates the score values via fast bit operations under bitmap index, to cut down significantly the score computation cost. Furthermore, we develop the *improved* BIG (IBIG) *algorithm* by employing the bitmap compression techniques and the binning strategies to trade the efficiency for space in the TKD query over incomplete data. In brief, the key contributions of this paper are summarized as follows.

- We formalize the problem of TKD query in the context of incomplete data. To our knowledge, there is *no prior work* on this problem.
- We propose efficient algorithms for processing TKD queries on incomplete data, using several novel heuristics.
- We present an *adaptive binning strategy* with an efficient method for choosing the appropriate number of bins to minimize the space of bitmap index for IBIG.
- We conduct extensive experiments using both real and synthetic datasets to demonstrate the effectiveness of our developed pruning heuristics and the performance of our proposed algorithms.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 formulates the problem. Section 4 elaborates our efficient algorithms for TKD query processing on incomplete data. Experimental results and our findings are reported in Section 5. Finally, Section 6 concludes the paper with some directions for future work.

2 Related Work

In this section, we first overview previous work on TKD queries in traditional and uncertain databases, and then survey the existing work related to querying incomplete data.

2.1 Top-k Dominating Queries

Papadias et al. [5] first introduce the top-k dominating (TKD) query as a variation of skyline queries, and they present a skyline based algorithm for processing TKD queries on the traditional complete dataset indexed by an R-tree. To boost efficiency, Yiu and Mamoulis [6], [7] propose two approaches

based on the aR-tree to tackle the TKD query. More recently, some new variants of TKD queries are studied, including subspace dominating query [21], continuous top-k dominating query [22], [23], metric-based top-k dominating query [9], top-k dominating query on massive data [24], etc.

In addition, the probabilistic top-k dominating (PTKD) query has also been explored [3], [4], [8], [25]. Specifically, Lian and Chen [3], [4] investigate PTKD query on uncertain data, which returns the k uncertain objects that are expected to dynamically dominate the largest number of uncertain objects in both the full space and subspace. Zhang et al. [8] consider the threshold-based PTKD query in full spaces. Zhan et al. [25] adopt the parameterized ranking semantics to formally define TKD query on multi-dimensional uncertain objects.

Note that, as mentioned in Section 1, the traditional and probabilistic TKD query algorithms using the R-tree/aR-tree and/or the transitivity of dominance relationship are not applicable to the TKD query on incomplete data.

2.2 Querying Incomplete Data

Data missing is a ubiquitous issue, and the study of incomplete data has attracted much attention. There are many efforts on modeling incomplete data, such as c-table [12], the classical logic and modal logic tools for modeling and processing incomplete data [13], model comparisons for incomplete data [11], I-SQL and world-set algebra language for incomplete data [10], etc. In addition, there are four common index structures to index incomplete data, namely, bitstring-augmented R-tree (BR-tree), MOSAIC [15], bitmap index, and quantization index [14].

Recently, many queries over incomplete data have been investigated, including ranking queries [18], [19], skyline queries [1], [2], [16], and similarity queries [17]. Haghani et al. [18] solve continuous monitoring top-k queries over incomplete data streams. Soliman et al. [19] explore a novel probabilistic model, and formulate several types of ranking queries on such model. Khalefa et al. [1] develop ISkyline algorithm to obtain skyline objects from incomplete data. Gao et al. [2] propose efficient kISB algorithm for processing k-skyband queries over incomplete data. Lofi et al. [16] present an approach to compute the skyline using crowd-enabled databases with the challenge of dealing with missing information in datasets. Cheng et al. [17] study the similarity search on dimension incomplete data.

It is worth pointing out that, our work differs from all the aforementioned works in that we aim at the problem of processing top-k dominating queries on incomplete data, which is, to our knowledge, the first attempt on this problem.

3 Problem Formulation

In this section, we formalize the dominance relationship and the TKD query on incomplete data. Table 1 summarizes the symbols used frequently in the rest of this paper.

To simplify the representation and computation, we use a dash "-" to represent a missing dimensional value for an object o, and a bit vector with d bits, denoted as β_o, to denote whether dimensional values of the object o are missing, e.g., the i-th bit of β_o is *on* (i.e., 1) if its i-th dimensional value of o is *observed*; otherwise, the i-th bit is *off* (i.e., 0). As an example, in Fig. 2, the incomplete object c = (5, -) has its $\beta_c = 10$, since it has only one observed dimensional value 5 on x-axis. In addition, two objects o and o' are *comparable* only if they both have observed values in at least one common dimension, i.e., the result of bitwise-and (&) of their bit vectors is not zero (β_o & $\beta_{o'} \neq 0$).

TABLE 1: Symbols and description

Notation	Description
o	a d-dimensional data object with missing values on some dimensions
S	a set of d-dimensional data objects o
$o.[i]$	the i-th dimensional value of an object o
β_o	a bit vector corresponding to an object o with d bits denoting whether the d-th dimensional value of o is observed
$\prec$	dominance
S_C (S_G)	the candidate (final result) set of a TKD query
τ	the k-th highest score of objects in S_C
F	a priority queue containing the objects in S sorted in descending order of their MaxScore
$Iset(o)$	a set of dimensions i of an object o where $o.[i]$ is observed
$\Phi(o)$	a set of objects in S that are incomparable to o
$\Gamma(o)$	a set of objects o' that are strictly worse than o in all dimensions for which the values of o and o' are both observed
$\Lambda(o)$	a set of objects o' that share the same value as o in at least one dimension and meanwhile are dominated by o

To analyze missing data, there are three different missing models [26], i.e., *missing completely at random* (MCAR), *missing at random* (MAR), and *not missing at random* (NMAR). Without loss of generality, in this paper, we assume that the values are *at least approximately missing at random*, and we only consider the objects with at least one observed dimensional value. Based on these assumptions, the dominance relationship on incomplete data [1] is introduced below.

Definition 1: **(dominance relationship on incomplete data [1])**. Given two objects o and o', o dominates o', denoted as $o \prec o'$, if the following two conditions hold: (i) for every dimension i, either $o.[i]$ is no larger than $o'.[i]$ or at least one of them is missing, i.e., $\forall i$ with $\beta_o[i]$ & $\beta_{o'}[i] = 1$, $o.[i] \leq o'.[i]$; and (ii) there is at least one dimension j, in which both $o.[j]$ and $o'.[j]$ are observed and $o.[j]$ is less than $o'.[j]$, i.e., $\exists j$ with $\beta_o[j]$ & $\beta_{o'}[j] = 1$, $o.[j] < o'.[j]$.

In the above definition, the smaller the value, the better. Take Fig. 2 as an example. Object f = (4, 2) is said to dominate object c = (5, -) as $f.[1](=4) < c.[1](=5)$ satisfies. For objects c and e = (-, 4), they both only have one dimensional observed value and they do not dominate each other, since they are not comparable. Note that, the dominance relationship over incomplete data loses the transitivity. As shown in Fig. 2, $f \prec e$ and $e \prec b$ according to Definition 1. Nonetheless, f does not dominate b. Moreover, as mentioned in [1], there may be a cyclic dominance relationship on incomplete data.

The dominance relationship defined in Definition 1 is meaningful. Given two objects o and o', if we have no information about their missing value(s), there is no clear judgement on which object is better for the dimensions with missing value(s). Therefore, we can only utilize the common observed dimensional values to decide the dominance relationship of those two objects, as discussed in Section 1.

It is worth noting that, there are other similar dominance

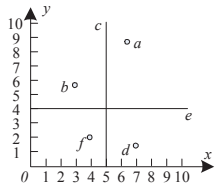


Fig. 2: Illustration of a TKD query

B_3(-, -, 4, 9)	C_1(2, -, -, 3)
D_2(2, 1, -, 4)	A_4(-, 7, 4, 5)
C_2(2, -, -, 1)	C_3(3, -, -, 2)
B_2(-, -, 3, 1)	B_5(-, -, 7, 4)
C_4(3, -, -, 3)	C_5(3, -, -, 4)
A_2(-, 1, 2, 1)	A_3(-, 1, 3, 4)
D_5(5, 5, -, 4)	A_5(-, 4, 8, 3)
D_1(3, 5, -, 2)	B_4(-, -, 3, 7)
D_3(2, 4, -, 1)	A_1(-, 3, 1, 3)
B_1(-, -, 1, 2)	D_4(4, 4, -, 5)

Fig. 3: A sample dataset

definitions over incomplete data, such as *missing flexible dominance* (MFD) operator which is flexible, reasonable, and fair in many real-life applications. MFD differentiates three cases of the corresponding dimensional values for two incomplete objects o and o', case (i) both $o.[i]$ and $o'.[i]$ are observed; case (ii) only one of $o.[i]$ and $o'.[i]$ is observed; and case (iii) both are missing, in order to flexibly emphasize on the existence values. Assume that there is a weight vector $W = \{w_1, w_2, \cdots, w_d\}$ corresponding to the data space D with cardinality $|D| = d$, and a real parameter λ $(0 < \lambda < 1)$. Based on Definition 1, MFD defines an additional weight for two objects $o \prec o'$, termed as $\Omega(o, o')$, as the accumulated weight on which at least one of the corresponding dimensional values is observed. Formally, $\Omega(o, o') = \sum_{i \in D_1} w_i + \lambda \sum_{j \in D_2} w_j$, where D_1 contains all the dimensions i such that both $o.[i]$ and $o'.[i]$ are observed, and D_2 contains all the dimensions j such that one and only one of $o.[j]$ and $o'.[j]$ is observed. It is important to note that, the dimensions where both objects miss their values are ignored in Ω, and a larger $\Omega(o, o')$ value indicates a higher recognition for the dominance $o \prec o'$.

Take two objects o_1 = (-, 3, 2) and o_2 = (-, 2, -) as an example. Since $o_1 \prec o_2$, MFD operator defines a weight $\Omega(o_1, o_2) = w_2 + \lambda w_3$. Thus, by setting the score of an object o as the accumulated Ω values between o and objects O dominated by o (i.e., $\mathsf{score}(o) = \sum_{o' \in O} \Omega(o, o')$), the TKD query could identify the most meaningful and influential objects over incomplete dataset. Note that, it is fair for the object that has very different number of attributes, and the weight vector W can be tunable flexibly in different real-life applications.

Without loss of generality, in this paper, we formalize the score of an object o in Definition 2 and the TKD query on incomplete data in Definition 3 based on Definition 1. However, as mentioned earlier, our proposed algorithms can be easily generalized to solve the TKD query under MFD operator, which is also one of our future work.

Definition 2: **(score)**. Given an incomplete dataset S and an object $o \in S$, the score of o, denoted as $\mathsf{score}(o)$, is the number of the objects $o' \in S - \{o\}$ that are dominated by o, i.e., $\mathsf{score}(o) = |\{o' \in S - \{o\} | o \prec o'\}|$.

Definition 3: **(TKD query on incomplete data)**. Given an incomplete dataset S, a top-k dominating (TKD) query over S retrieves the set $S_G \subseteq S$ of k objects with highest score values, i.e., $S_G \subseteq S$ and $|S_G| = k$ and $\forall o \in S_G$, $\forall o' \in (S - S_G)$, $\mathsf{score}(o) \geq \mathsf{score}(o')$.

Consider the dataset shown in Fig. 2, object f dominates 3 objects $\{a, c, e\}$, and hence, $\mathsf{score}(f) = 3$. Similarly, based on Definition 2, $\mathsf{score}(b) = \mathsf{score}(c) = \mathsf{score}(e) = 2$, $\mathsf{score}(d) = 1$, and $\mathsf{score}(a) = 0$. Thus, a T1D $(k = 1)$ query on the dataset depicted in Fig. 2 returns the result set $\{f\}$, as it has the maximal score. Note that, when there is a tie, we adopt random selection as a tie breaker in this paper.

It is important to note that, following the previous works [1], [2], [14], [15], we focus on this incomplete data model to solve TKD query. There are another two popular methods to tackle incomplete data including the one based on probability distribution [3], [4], [8] and the one based on missing value inference [20], [26] via Expectation-Maximization (EM) principle, multiple imputation, or human intelligence. The missing data inference method to process the TKD query will be exploited in our future work. In addition, unless mentioned otherwise, the incomplete dataset illustrated in Fig. 3 serves as a running example in the rest of the paper.

4 TKD Query on Incomplete Data

In this section, we first present three efficient algorithms, i.e., *extended skyband based* (ESB) *algorithm*, *upper bound based* (UBB) *algorithm*, and *bitmap index guided* (BIG) *algorithm*, to support TKD query processing on incomplete data. Then, we propose an *improved BIG* (IBIG) *algorithm* based on BIG, to further minimize the space cost of bitmap index.

4.1 Extended Skyband Based Algorithm

The intuitive method (denoted as Naive) for the TKD query on incomplete data is to compute the score of every object o by conducting exhaustive pairwise comparisons among the whole dataset, and to return the k objects with the highest scores. However, this Naive approach is *inefficient* due to the extremely large size of the candidate set and the expensive cost of score computation.

Fortunately, the objects with observed attributes falling inside the same set of dimensions actually satisfy the transitive dominance relationship. To this end, we re-organize the objects into buckets. Here, each bucket corresponds to a given subset of d' $(\leq d)$ dimensions, and it accommodates all the objects whose observed attributes fall in those d' dimensions exactly. Accordingly, we present our first algorithm, namely, *extended skyband based* (ESB) *algorithm*, which uses *local skyband technique* to answer the TKD query over incomplete data. Here, we borrow the concept of k-skyband (kSB) query on incomplete data [2]. The kSB query is a variant of skyline queries, and it retrieves the objects dominated by less than k objects. Since the objects within the same bucket share the same bit vector, they can be regarded as a complete dataset in d'-dimensional space with $d' \leq d$. If we perform a kSB query for each bucket, the kSB query results collectively form a candidate set for a TKD query, as stated in Lemma 1.

Lemma 1: **(local skyband technique)**. Given an incomplete dataset S, let bucket O_b represent the set of objects $o \in S$ sharing the same bit vector b, i.e., $O_b = \{o \in S \mid \beta_o = b\}$, and S_S be the local skyband object set returned by a k-skyband query over O_b. For an object $o \in O_b$, if o is not included in S_S, o can not be returned by the TKD query on S, i.e., $o \notin S_S \Rightarrow o \notin S_G$.

Proof. By contradiction. Assume that there is an object $o' \in O_b$ with $o' \notin S_S$ but $o' \in S_G$. As $o' \notin S_S$, o' is dominated by at least k objects from the bucket O_b, denoted as $D_{o'}$ with $|D_{o'}| \geq k$. Since the dominance relationship over the objects inside the same bucket satisfies transitivity, the objects dominated by o'

Algorithm 1 Extended Skyband Based Algorithm (ESB)

Input: an incomplete data set S, a parameter k
Output: the result set S_G of a TKD query on S
/* $kSB(O)$: the result set of a k-skyband query on a bucket O. */
1: initialize sets $S_C \leftarrow S_G \leftarrow \varnothing$
2: **for** each object $o \in S$ **do**
3: insert o into a bucket O based on β_o (create O if necessary)
4: **for** each bucket O **do**
5: $S_C \leftarrow S_C \bigcup kSB(O)$
6: **for** each object $o \in S_C$ **do**
7: update score(o) by comparing o with all the objects in S
8: add the k objects in S_C having the highest scores to S_G
9: **return** S_G

are also dominated by all the objects in $D_{o'}$. In other words, all the objects in $D_{o'}$ dominate more objects than o', and thus, they all have higher scores than o'. As $|D_{o'}| \geq k$, it is confirmed that $o' \notin S_G$, which contradicts with our assumption. Thus, our assumption is invalid. The proof completes. ■

On top of the candidate set formed based on Lemma 1, we propose ESB algorithm with its pseudo-code depicted in Algorithm 1. ESB adopts the pruning-and-filtering method to tackle the TKD query on incomplete data. It first partitions objects $o \in S$ into its corresponding bucket based on its β_o (lines 2-3), and then performs a local kSB query for objects within the same bucket (lines 4-5). The collection of the returned results from kSB queries form a candidate set S_C to complete the pruning step. Next, the filtering step starts. ESB ranks the candidates in S_C based on their score values, and returns the top-k candidates with the highest scores as the final query result (lines 6-9). Example 1 illustrates how ESB algorithm works.

Example 1: Assume ESB algorithm is invoked for a T2D ($k = 2$) query over our sample dataset shown in Fig. 3, with the processing steps illustrated in Fig. 4. It first clusters the objects into buckets based on their bit vectors. For instance, when the first object B_3 in Fig. 3 is evaluated, a bucket B corresponding to the bit vector, i.e., $\beta_{B_3} = 0011$, is created, and B_3 is the first object enrolled. In total, four buckets are created with each having five objects, as depicted in Fig. 4. Note that, it is a coincidence that there are five objects in each of the four buckets. ESB then performs local 2-skyband queries on the objects within every bucket, with the local 2-skyband objects returned by each bucket, denoted as S_s, highlighted in Fig. 4. Four sets of local 2-skyband objects form the candidate set S_C that contains 11 objects, with $S_C = \{A_1, A_2, A_3, B_1, B_2, C_1, C_2, C_3, D_1, D_2, D_3\}$. For these 11 objects in S_C, we then derive their scores based on the numbers of objects in S they dominate. As shown in Fig. 4, objects A_2 and C_2 share the same highest score (i.e., 16), and thus, they are returned as the final result (i.e., S_G) of the T2D query over S. □

4.2 Upper Bound Based Algorithm

Although ESB algorithm can answer TKD queries on incomplete data, its performance is highly dependent on the size of S_C, which is determined by the nature of data. In an extreme case, $|S_C| = |S|$, meaning that the filtering step of ESB fails since it cannot filter out any object. As a result, the scores

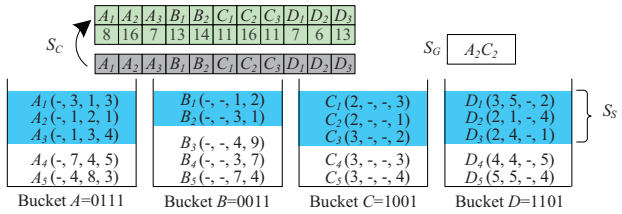


Fig. 4: Example of ESB

of all objects have to be derived. Motivated by the instability of ESB algorithm, we propose the *upper bound based* (UBB) *algorithm* for supporting the TKD query over incomplete data, which utilizes the upper bound scores of objects to determine the access order of objects to reduce the candidate set size.

In particular, UBB integrates the ranking process into the object evaluation, and enables an early termination of TKD query processing before evaluating all the candidates. Based on this intention, we present the concept of *upper bound score* that returns the maximum number of the objects that a specified object o dominates, i.e., the upper bound of score(o), as stated in Lemma 2 below.

Lemma 2: **(upper bound score)**. Given an object $o \in S$, let $T_i(o)$ be the maximum set of the objects p dominated by o on the i-th dimension. Formally, $T_i(o)$ can be defined as

$$T_i(o) = \begin{cases} \{p \in S - \{o\} | o.[i] \leq p.[i]\} \cup S_i & \text{if } i \in Iset(o) \\ S & \text{otherwise} \end{cases} \quad (1)$$

where S_i represents the set of the objects whose i-th dimensional values are missing, and $Iset(o)$ denotes the set of dimensions where o has observed values. Based on $T_i(o)$, the upper bound of score(o), denoted as MaxScore(o), could be derived as MaxScore$(o) = \min\{|T_1(o)|, |T_2(o)|, ..., |T_d(o)|\}$.

Proof. First, we prove that $|T_i(o)|$ is an upper bound score for an object $o \in S$. On the one hand, for the case that $i \notin Iset(o)$, it is obvious that $|T_i(o)|$ (= $|S|$) is an upper bound for score(o). On the other hand, if $i \in Iset(o)$, according to Eq. (1), the object set $T_i(o)$ contains all objects that are not better than o in the i-th dimension and all the objects with their i-th dimensional values missing. Based on Definition 1, $T_i(o)$ contains all the possible objects that have the potential to be dominated by o. Thus, $|T_i(o)|$ is an upper bound of score(o) for $i \in Iset(o)$. In summary, we can conclude that $|T_i(o)|$ is an upper bound for score(o). Then, it is easy to find that the minimum cardinality of $T_i(o)$ for $1 \leq i \leq d$ (i.e., MaxScore(o)) is also an upper bound score for o. The proof completes. ■

For ease of understanding, we illustrate how to derive MaxScore(B_3) for object B_3 in our sample dataset (shown in Fig. 3). We first get $T_1(B_3) = T_2(B_3) = S$ because $B_3.[1]$ and $B_3.[2]$ are missing, $T_3(B_3) = \{A_4, A_5, B_5, C_1, C_2, C_3, C_4, C_5, D_1, D_2, D_3, D_4, D_5\}$, and $T_4(B_3) = \varnothing$. Then, MaxScore(B_3) = $\min\{|T_1(B_3)|, |T_2(B_3)|, |T_3(B_3)|, |T_4(B_3)|\} = 0$. It is worth noting that, MaxScore can be calculated at $O(N \cdot \lg N)$ cost based on the B^+-tree structure, where N is the dataset cardinality. Based on the concept of MaxScore(o), a pruning strategy is developed, as stated in Heuristic 1, which serves as an *early termination condition* in UBB algorithm.

Heuristic 1: **(upper bound score pruning)**. Given a TKD query on an incomplete dataset S, let S_C be a candidate set containing k objects for the query and τ be the smallest score for all objects in S_C, i.e., $|S_C| = k$ and $\forall c \in S_C$, score$(c) \geq \tau$

Algorithm 2 Upper Bound Based Algorithm (UBB)

Input: an incomplete data set S, a parameter k, a pre-computed priority queue F sorting all objects from S in descending order of their MaxScore

Output: the result set S_G of a TKD query on S

1: initialize sets $S_C \leftarrow S_G \leftarrow \varnothing$ and $\tau \leftarrow -1$
2: **while** F is not empty **do**
3: $o \leftarrow$de-queue(F)
4: **if** MaxScore(o) $\leq \tau$ **then break** // Heuristic 1
5: **else**
6: score(o) $\leftarrow$ Get-Score(o)
7: **if** score(o) $> \tau$ or $\tau < 0$ **then**
8: $S_C \leftarrow S_C \bigcup \{o\}$
9: **if** $|S_C| > k$ **then**
10: $S_C \leftarrow S_C - \{p\}$ with $p \in S_C$ and score(p)=τ
11: update $\tau \leftarrow \min\{\text{score}(c) \mid c \in S_C\}$ if $|S_C| = k$
12: **return** $S_G \leftarrow S_C$

ID	C_2	A_2	B_2	B_1	C_3	D_3	A_1	C_1	C_4	D_1	A_5	A_3	B_5	C_5	D_2	D_5	A_4	D_4	B_4	B_3
MaxScore	19	17	16	15	15	15	12	12	12	12	10	8	8	8	8	8	3	3	1	0

Fig. 5: The priority queue F for the dataset in Fig. 3

and $\exists c' \in S_C$, score(c') $= \tau$. For a specified object $o \in S$ with MaxScore(o) $\leq \tau$, it can be safely pruned away as it cannot be an actual answer object for the TKD query over S.

Proof. First, $\forall c \in S_C$, score(c) $\geq \tau$. Second, score(o) $\leq$ MaxScore(o) $\leq \tau$. Thus, $\forall c \in S_C$, score(c) $\geq$ score(o). As $|S_C| = k$, there are k objects having higher scores than o, and object o cannot be a real answer object for the TKD query. ■

Based on Heuristic 1, we develop UBB algorithm with its pseudo-code presented in Algorithm 2. It takes as inputs an incomplete dataset S, a parameter k, and a priority queue F with all the objects $o \in S$ sorted in *descending* order of their MaxScore(o). First of all, UBB initializes two sets S_C and S_G to empty, and sets τ as -1 (line 1). Here, τ is to record the minimum score of the objects in S_C, and it is set to -1 if the set S_C contains less than k objects. It then visits the objects in F one by one until F is empty or early termination condition is satisfied (lines 2-11). Specifically, UBB dequeues the top object o of F. If MaxScore(o) $\leq \tau$, the early termination condition of Heuristic 1 is satisfied, and the while-loop can be finished. Note that the condition MaxScore(o) $\leq \tau$, and objects in F are sorted based on descending order of MaxScore guarantee that object o and all the remaining objects in F have their scores bounded by τ. In addition, it also ensures that the candidate set S_C is full, and it contains k objects with their scores $\geq \tau \geq 0$. On the other hand, if the early termination condition is not satisfied, the evaluation continues (lines 5-11). UBB computes score(o) via a function Get-Score that derives score(o) based on pairwise comparisons (line 6). If τ is -1, the candidate set S_C is not full, and object o is enrolled into S_C (line 8). Otherwise, if score(o) is larger than τ, object o is also enrolled into S_C to replace the object $p \in S_C$ having smallest score value (i.e., τ). τ is updated as well if S_C is updated (lines 9-11).

Example 2: We illustrate UBB algorithm for a T2D query on a sample dataset depicted in Fig. 3, with the priority queue F over the sample dataset shown in Fig. 5. First, UBB evaluates the head object of F (object C_2 first and object A_2 second), after which $S_C = \{C_2, A_2\}$ and $\tau = 16$. Then, UBB evaluates the next dequeued object B_2. As MaxScore(B_2) $= 16$, the early termination condition is satisfied. Thus, UBB returns $\{C_2, A_2\}$ as the final result set of the T2D query and terminates. □

ID	v_1	-	2	3	4	5	v_2	-	1	3	4	5	7	v_3	-	1	2	3	4	7	8	v_4	-	1	2	3	4	5	7	9
A_1	-	1	1	1	1	*3*	1	1	0	0	0	0	*1*	1	0	0	0	0	0	0	*3*	1	1	1	0	0	0	0	0
A_2	-	1	1	1	1	*1*	1	0	0	0	0	0	*2*	1	1	0	0	0	0	0	*1*	1	0	0	0	0	0	0	0
A_3	-	1	1	1	1	*1*	1	0	0	0	0	0	*3*	1	1	1	0	0	0	0	*4*	1	1	1	1	0	0	0	0
A_4	-	1	1	1	1	*7*	1	1	1	1	1	0	*4*	1	1	1	1	0	0	0	*5*	1	1	1	1	1	0	0	0
A_5	-	1	1	1	1	*4*	1	1	1	0	0	0	*8*	1	1	1	1	1	1	0	*3*	1	1	1	0	0	0	0	0
B_1	-	1	1	1	1	-	1	1	1	1	1	1	*1*	1	0	0	0	0	0	0	*2*	1	1	0	0	0	0	0	0
B_2	-	1	1	1	1	-	1	1	1	1	1	1	*3*	1	1	1	0	0	0	0	*1*	1	0	0	0	0	0	0	0
B_3	-	1	1	1	1	-	1	1	1	1	1	1	*4*	1	1	1	1	0	0	0	*9*	1	1	1	1	1	1	1	0
B_4	-	1	1	1	1	-	1	1	1	1	1	1	*3*	1	1	1	0	0	0	0	*7*	1	1	1	1	1	1	0	0
B_5	-	1	1	1	1	-	1	1	1	1	1	1	*7*	1	1	1	1	1	0	0	*4*	1	1	1	1	0	0	0	0
C_1	*2*	1	0	0	0	0	-	1	1	1	1	1	-	1	1	1	1	1	1	1	*3*	1	1	1	0	0	0	0	0
C_2	*2*	1	0	0	0	0	-	1	1	1	1	1	-	1	1	1	1	1	1	1	*1*	1	0	0	0	0	0	0	0
C_3	*3*	1	1	0	0	0	-	1	1	1	1	1	-	1	1	1	1	1	1	1	*2*	1	1	0	0	0	0	0	0
C_4	*3*	1	1	0	0	0	-	1	1	1	1	1	-	1	1	1	1	1	1	1	*3*	1	1	1	0	0	0	0	0
C_5	*3*	1	1	0	0	0	-	1	1	1	1	1	-	1	1	1	1	1	1	1	*4*	1	1	1	1	0	0	0	0
D_1	*3*	1	1	0	0	0	*5*	1	1	1	1	0	0	-	1	1	1	1	1	1	*2*	1	1	0	0	0	0	0	0
D_2	*2*	1	0	0	0	0	*1*	1	0	0	0	0	0	-	1	1	1	1	1	1	*4*	1	1	1	1	0	0	0	0
D_3	*2*	1	0	0	0	0	*4*	1	1	1	0	0	0	-	1	1	1	1	1	1	*1*	1	0	0	0	0	0	0	0
D_4	*4*	1	1	1	0	0	*4*	1	1	1	0	0	0	-	1	1	1	1	1	1	*5*	1	1	1	1	1	0	0	0
D_5	*5*	1	1	1	1	0	*5*	1	1	1	1	0	0	-	1	1	1	1	1	1	*4*	1	1	1	1	0	0	0	0

Fig. 6: The bitmap index for the dataset in Fig. 3

4.3 Bitmap Index Guided Algorithm

Our proposed UBB algorithm limits the size of candidate set by utilizing upper bound score pruning technique for the TKD query on incomplete data. However, the upper bound score may be rather loose, thereby we have to derive the real scores for many objects (even the whole dataset) via exhaustive pair comparisons, which degrades search performance significantly. Thus, an efficient score computation method is in demand. As a solution, we introduce a newly proposed bitmap index on incomplete data and propose the *bitmap index guided* (BIG) *algorithm* to solve the TKD query on incomplete data. Combining MaxScore technique, BIG enables a novel bitmap pruning using a bitmap index, and employs fast bit-wise operations for more efficient score computation. Furthermore, we also develop an improved version of BIG (denoted as IBIG) to minimize the bitmap storage cost via the bitmap compression techniques and an adaptive binning strategy.

As we know, the traditional bitmap index (e.g., [27], [28], [29], [30]) is based on *complete* data, and it supports dominance relationship checking via bit-wise operations. Nonetheless, it is not applicable to our problem which is based on incomplete data. Hence, a new bitmap index has to be designed to deal with missing data. Moreover, the dominance relationship of TKD query with incomplete data cannot be derived based only on the bit operations. Thus, an efficient algorithm based on the bitmap index supporting missing data is also desired.

Specifically, our new bitmap index is built as follows. First, an object o is represented by a bit string with $\sum_{i=1}^{d}(C_i+1)$ bits in the bitmap index, where each dimension of o is represented by a sub-string with (C_i+1) bits. Here, C_i is the total number of different observed values (i.e., domain) on the i-th dimension, and the extra one bit denotes the missing value. Take the sample dataset (shown in Fig. 3) as an example. For the 1st dimension, there are in total four different observed values, i.e., $\{2, 3, 4, 5\}$, contributed by 20 objects in the dataset with $C_1 = 4$. Thus, we use a (4 + 1)-bit string to represent the values of the 20 objects in the 1st dimension in the bitmap index. Note that, for a group of values on any dimension, our bitmap index only cares about how many different values are there on this dimension in order to decide the length of the sub-bit string for representing the dimension. Hence, the bitmap

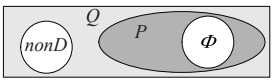


Fig. 7: The Venn diagram of object sets P, Q, Φ, and nonD

index does support floating-point numbers. If every object has distinct i-th dimensional values for a given dataset, C_i could be as large as the dataset cardinality. It is worth noting that, the values of C_is do not influence query efficiency but only the bitmap storage cost.

Next, we explain how to use a sub-string with (C_i+1) bits to index the values observed in the i-th dimension. In short, the (C_i+1) bits refer to a series of ranked dimensional values in the i-th dimension. Take the five-bit string representing the 1st dimension (with four observed values 2, 3, 4, and 5) introduced above as an example. The first bit is w.r.t. the missing case, the second bit corresponds to the dimensional value 2, the third bit refers to the dimensional value 3, and so on. We utilize the range encoding method to form the bitmap index. If a value is observed, its corresponding bit, together with all the bits following it, is set to 0. As an example, $C_1.[1]=2$, and hence, the bit w.r.t. value 2 (i.e., the second bit) and all the subsequent bits are set to 0 (i.e., 10000); and $D_4.[1]=4$, and thus, the bit w.r.t. value 4 (i.e., the fourth bit) and all the following bits are set to 0 (i.e., 11100). It is important to note that, the missing value is always encoded as a sub-string with all '1', in order to simplify dominance checking. The bit-strings of all the objects form the bitmap index. We plot the bitmap index for our sample dataset in Fig. 6, where we also list $o.[i]$ values under v_i columns for ease of reference.

We are now ready to introduce the score computation (refer to Definition 2). To begin with, we introduce four object sets w.r.t. an object o, i.e., P, Q, $\Phi(o)$, and $\mathsf{nonD}(o)$, with their containment relationship plotted in Fig. 7. Specifically, Q denotes the set of objects, excluding object o, that are not better than o or missing in the dimensions in $Iset(o)$; P is a subset of Q which refers to the set of objects that are strictly worse than o or missing on each dimension in $Iset(o)$; $\Phi(o)$ represents the set of objects that are incomparable to o; and $\mathsf{nonD}(o)$ refers to the set of objects in $(Q-P)$ that are not dominated by o.

In the following, we first explain how the bitmap index can facilitate the computation of sets Q and P, as presented in Definition 4. It is important to note that, both Q^i and P^i consist of the objects that might be dominated by o purely based on the values in the i-th dimension, and P^i is a subset of Q^i. In this paper, for clarity, we represent the corresponding (*vertical*) bit-vectors encoding the object sets Q^i and P^i as $[Q^i]$ and $[P^i]$ respectively, which are abstracted from the bitmap index. Specifically, both $[Q^i]$ and $[P^i]$ have the length of $|S|$ bits, with one bit corresponding to an object in S. If an object is included in Q^i or P^i, the corresponding bit in $[Q^i]$ or $[P^i]$ is set to 1 in our bitmap index. Otherwise, the bit is set to 0.

Definition 4: Let $Q^i=P^i=S$ if $o.[i]$ is missing; otherwise $Q^i=\{p\in S \mid o.[i]\le p.[i]\vee p.[i]$ is missing$\}$ and $P^i=\{p\in S \mid o.[i]<p.[i]\vee p.[i]$ is missing$\}$, then sets Q and P are calculated as

$$Q=\bigcap_{i=1}^{d} Q^i-\{o\} \qquad P=\bigcap_{i=1}^{d} P^i \tag{2}$$

For instance, our sample dataset has 20 objects, and hence,

ID	C_2	A_2	B_2	B_1	C_3	D_3	A_1	C_1	C_4	D_1	A_5	A_3	B_5	C_5	D_2	D_5	A_4	D_4	B_4	B_3
MaxScore	19	17	16	15	15	15	12	12	12	12	10	8	8	8	8	8	3	3	1	0
MaxBitScore	19	17	16	15	13	15	10	12	10	9	5	8	4	7	8	4	1	3	1	0

Fig. 8: Comparison between MaxScore and MaxBitScore

the bit-vectors $[Q^i]$ and $[P^i]$ for a specified object o have 20 bits, with the first bit w.r.t. A_1, the second bit w.r.t. A_2, and so on. Take B_3 as an example. Its corresponding set $Q^3=\{A_4,\ A_5,\ B_3,\ B_5,\ C_1,\ C_2,\ C_3,\ C_4,\ C_5,\ D_1,\ D_2,\ D_3,\ D_4,\ D_5\}$, and the corresponding bit-vector $[Q^3]$= 00011001011111111111. Thus, based on bit-vectors $[Q^i]$ and $[P^i]$ in the bitmap index, we can easily get sets Q and P by fast bit-wise operations, without comparing the real dimensional values (utilized in ESB and UBB algorithms).

In addition, we observe that set Q, which is formed at a small cost by bit-wise operations with the help of our bitmap index, provides another upper bound score MaxBitScore(o), as stated in Heuristic 2.

Heuristic 2: **(bitmap pruning)**. Given an incomplete dataset S and a candidate set S_C for a TKD query containing k objects in S, let τ be the minimum score of the objects in S_C. For a specified object $o\in(S-S_C)$ with $\mathsf{MaxBitScore}(o)=|Q|\le\tau$, it can be pruned away safely since it cannot be a real answer object for the TKD query on S.

Proof. Given an object o with $\mathsf{score}(o)=\lambda$, there are λ objects dominated by o, denoted as $R=\{p^n \mid o\prec p^n, p^n\in S, n=1,...,\lambda\}$. We can easily observe that $\forall p^n\in R$, $p^n\in Q$. Hence, R is a subset of Q, i.e., $R\subseteq Q$, and thereby, $\lambda=\mathsf{score}(o)=|R|\le|Q|=\mathsf{MaxBitScore}(o)\le\tau$. Consequently, we are certain that $\mathsf{score}(o)\le\tau$ and the proof completes. ■

Back to our example object B_3. We have $\bigcap_{i=1}^{4}Q^i=\{B_3\}$, and thus, $\mathsf{MaxBitScore}(B_3)=|Q|=|\bigcap_{i=1}^{4}Q^i-\{B_3\}|=0$. If S_C has k objects with $\tau=1$, object B_3 can be discarded safely based on Heuristic 2. It is worth noting that, compared with MaxScore(o), MaxBitScore(o) actually offers a tighter upper bound for score(o), i.e., $\mathsf{MaxBitScore}(o)\le\mathsf{MaxScore}(o)$ as stated in Lemma 3. We also list these two upper bounds for our sample objects in Fig. 8 for illustration purpose.

Lemma 3: Given a TKD query on an incomplete dataset S and an object $o\in S$, $\mathsf{MaxBitScore}(o)\le\mathsf{MaxScore}(o)$.

Proof. On the one hand, for each object $q\in Q$, the condition (i.e., $o.[i]\le q.[i]\vee q.[i]$ is missing) holds for all the dimensions $i\in Iset(o)$ where $o.[i]$ is observed. Hence, we can get that $q\in T_i(o)$ for all the dimensions $i\in Iset(o)$. On the other hand, for the dimensions i' where $o.[i']$ is missing, $T_{i'}(o)=S$ according to Eq. (1), and thus, we have $q\in T_{i'}(o)$. In a word, assume that $\min\{|T_1(o)|, |T_2(o)|, ..., |T_d(o)|\}=|T_t(o)|$, we have $q\in T_t(o)$. As a summary, for any $q\in Q$, we have $q\in T_t(o)$ and hence $Q\subseteq T_t(o)$. Thus, $\mathsf{MaxBitScore}(o)=|Q|\le|T_t(o)|=\mathsf{MaxScore}(o)$. The proof completes. ■

As we know, $\mathsf{score}(o)=|R|$ if set R contains the set of objects dominated by object o. Assume that we partition R into two disjoint sub-sets $\Gamma(o)$ and $\Lambda(o)$ such that $\Gamma(o)=P-\Phi(o)$, which includes all the objects o' that are *strictly worse* than o in all dimensions where both o and o' have observed values (i.e., $Iset(o)\cap Iset(o')$) and meanwhile are dominated by o, and set $\Lambda(o)=Q-P-\mathsf{nonD}(o)$, which consists of the objects o'' that share the same value as o in at least one dimension and meanwhile are dominated by o. Then, we have $\mathsf{score}(o)=$

Algorithm 3 Get-Score Algorithm for BIG (BIG-Score)

Input: a bitmap index, an object o, $\Phi(o)$, τ, S_C, k
Output: the score of o, i.e., score(o)

1: get $[P^i]$ and $[Q^i]$ of o for each $1 \le i \le d$ from bitmap index
2: $Q \leftarrow \bigcap_{i=1}^{d} Q^i - \{o\}$, MaxBitScore($o$) $\leftarrow |Q|$
3: **if** $|S_C| = k$ and MaxBitScore(o) $\le \tau$ **then** // Heuristic 2
4: **return** 0 // o is pruned away
5: **else**
6: $P \leftarrow \bigcap_{i=1}^{d} P^i$, $\Gamma(o) \leftarrow P - \Phi(o)$
7: **for** each pair of $\langle p, i\rangle \in (Q-P) \times Iset(o)$ **do**
8: **if** $p.[i] = o.[i]$ **then** $p.tagT \leftarrow p.tagT + 1$
9: nonD(o) $\leftarrow$ nonD(o) $\cup \{p \in (Q-P) \mid p.tagT = |\beta_p \ \& \ \beta_o|\}$
10: $\Lambda(o) \leftarrow Q - P -$ nonD(o)
11: **return** score(o) $\leftarrow |\Gamma(o)| + |\Lambda(o)|$

$|\Gamma(o)| + |\Lambda(o)| = |Q - \Phi(o) - \text{nonD}(o)|$.

BIG-Score implements the score calculation based on bitmap index using score(o) $= |\Gamma(o)| + |\Lambda(o)|$, with its pseudo-code listed in Algorithm 3. First, before deriving the real score, it derives the upper bound MaxBitScore(o), and compares it with τ, the minimum score value of a candidate object (lines 1-2). If the filtering condition depicted in Heuristic 2 is satisfied, object o can be filtered out immediately without calculating its score (lines 3-4). Otherwise, the object passes the filtering, and we need to derive its real score based on $|\Gamma(o)| + |\Lambda(o)|$ (lines 5-11). To be more specific, it derives $\Gamma(o)$ (line 6) and $\Lambda(o)$ (lines 7-10). As mentioned earlier, $(Q-P)$ forms a candidate set for nonD(o), and hence, for objects $p \in (Q-P)$, BIG-Score checks each observed dimension i of o to find the objects with $p.[i] = o.[i]$. A counter $tagT$ is associated with every object $p \in (Q-P)$ to count the number of dimensions i such that $p.[i] = o.[i]$. In other words, objects $p \in (Q-P)$ with corresponding $tagT$ being equivalent to $|Iset(o) \cap Iset(p)|$ form nonD(o). Once nonD(o) is formed, $\Lambda(o)$ value is derived, and BIG-Score returns $|\Gamma(o)| + |\Lambda(o)|$ and terminates.

With the support of bitmap index, we propose BIG algorithm by integrating bit-wise operation on score computation and bitmap pruning to process efficiently the TKD query on incomplete data. Its main framwork is given in Algorithm 4, which is similar as UBB algorithm. Note that, BIG utilizes MaxScore(o) and MaxBitScore(o) for filtering. Given an object o, it first calculates MaxScore(o) and then compares it with τ. If object o cannot be filtered by MaxScore(o), it computes MaxBitScore(o) and again compares it against τ. As MaxBitScore(o) stands for a tighter upper bound of score(o), more unqualified objects are expected to be filtered out. For all the qualified objects, we need to derive their actual scores via BIG-Score.

Example 3: We illustrate how BIG algorithm answers a T2D query issued on our sample dataset. Suppose the corresponding priority queue F and the bitmap index are ready, as shown in Fig. 5 and Fig. 6 respectively. In addition, Φ is available with $\forall o \in S$, $\Phi(o) = \varnothing$. BIG starts its evaluation by continuously en-queueing the head entry from F. First, object C_2 is evaluated. BIG invokes BIG-Score to compute C_2's score. Specifically, BIG-score first gets the corresponding bit vectors of C_2 from the bitmap index (depicted in Fig. 6) as follows:

$[P^1] = 11111111110011110011$, $[P^2] = 11111111111111111111$,
$[P^3] = 11111111111111111111$, $[P^4] = 10111101111011111011$,

Algorithm 4 BIG Algorithm

Input: a bitmap index of an incomplete dataset S, F, Φ, k
Output: the result set S_G of the TKD query on S

Lines 1-5 are the same as lines 1-5 in Algorithm 2 including the upper bound score pruning
6: get score(o) by calling BIG-Score function // Algorihm 3
Lines 7-12 are identical to lines 7-12 in Algorithm 2

$[Q^1] = 11111111111111111111$, $[Q^2] = 11111111111111111111$,
$[Q^3] = 11111111111111111111$, $[Q^4] = 11111111111111111111$.

Since $|S_C| = 0$, it then computes $\bigcap_{i=1}^{4}[Q^i]$ = 1111111111 1111111111, $[P] = \bigcap_{i=1}^{4}[P^i]$ = 10111101110011110 011, and sets $|\Gamma(C_2)| = |P - \Phi(C_2)| = |P| = 14$. Next, BIG-Score examines the objects in $Q - P = \{A_2, B_2, C_1, D_2, D_3\}$. Thereafter, as $Iset(C_2) = \{d_1, d_4\}$, BIG-Score checks values of the objects from $(Q-P)$ w.r.t. the first and the fourth dimensions. On the first dimension, it finds objects $\{C_1, D_2, D_3\}$ with values being $C_2.[1]$. It gets objects $\{A_2, B_2, D_3\}$ having the same fourth dimensional value as $C_2.[4]$. Among those objects, A_2, B_2, and D_3 form nonD(C_2) because each of them has $tagT$ value equivalent to the number of dimensions that it is comparable with C_2. Thereafter, $|\Lambda(C_2)| = |Q - P - \text{nonD}(C_2)| = 2$, and score($C_2$) = $|\Gamma(C_2)| + |\Lambda(C_2)| = 14 + 2 = 16$. Object C_2 is then enrolled into a candidate set S_C. Next, object A_2 is evaluated. Similarly, BIG invokes BIG-Score to get score(A_2) = 16, and enrolls A_2 to S_C. Now, $|S_C| = 2$ and $\tau = 16$. Then, object B_2 is evaluated. As MaxScore(B_2) = 16 which is the same as τ, BIG terminates early according to Heuristic 1. Finally, the result set $\{C_2, A_2\}$ of the T2D query is returned. □

4.4 Improvement on BIG

Although BIG algorithm can significantly improve the search performance of TKD queries over incomplete data compared with ESB and UBB algorithms, we are also aware that the bitmap index size (denote as $cost_s$) is rather large, i.e., $cost_s = \sum_{i=1}^{d}(C_i + 1) \times |S|$, especially when the dimensionality of the search space (i.e., d) is high and/or the domain cardinality (i.e., C_i) is high. To this end, we propose the *improved* BIG (termed as IBIG) *algorithm* to efficiently address the storage issue by using the bitmap compression technique and the binning strategy. Specifically, the compression techniques are applied on the "vertical" bitsets, such as $[Q^i]$ and $[P^i]$, while the binning strategy compresses the bitmap index on the "horizontal" bitsets, i.e., for the bit-string of every object in the dataset. In the following, we detail these two techniques utilized in IBIG algorithm respectively.

First, we introduce two most efficient and popular *compression techniques*, i.e., *Word Aligned Hybrid* (WAH) [29] and *Compressed 'n' Composable Integer Set* (CONCISE) [31], to compress the bitmap index vertically. In this paper, we choose CONCISE instead of WAH. This is because, as shown in [31], CONCISE has better compression ratio than WAH, and its computational complexity is comparable to that of WAH. We also demonstrate that CONCISE does perform better than WAH via an empirical evaluation, to be reported in Section 5.1. Please refer to [29], [31] for the details of WAH and CONCISE. However, we also notice that even after we incorporate CONCISE into IBIG, the bitmap space could still be

large. The reason is that, the bitmap encoded method we used, i.e., range encoding, is not amenable to compression [30]. As to be confirmed in our experiments (to be presented in Section 5.1), the existing compression techniques are not very effective when they are applied to our setting.

Therefore, we propose the *binning strategy* for IBIG in order to further cut down the bitmap storage consumption horizontally. Instead of using one bit for one distinct value, it utilizes one bit to encode a range of dimensional values to reduce the index size. Before we formally introduce the concept of the binning strategy, we first present an intuitive example to introduce its main idea. As depicted in Fig. 6, there are four distinct observed values (i.e., 2, 3, 4, 5) in the first dimension, and the original bitmap index uses a five-bit string to represent the values accordingly. Now assume that we use two value bins to capture the observed values in the same dimension, with one bin covering value 2 and the other covering values 3, 4, and 5. Consequently, we only need a three-bit string to represent the observed values in the first dimension, one bit for missing value and two bits for two value bins. Under this binning strategy, our sample object D_4 with $D_4.[1] = 4$ is represented as 110 instead of 11100 in the original bitmap index.

For ease of presentation on the binning strategy, we introduce some notations. For each dimension i, we order objects based on their corresponding values (i.e., $o.[i]$), with min_i and max_i referring to the *minimum* and *maximum* observed values in the i-th dimension. Let N be the cardinality of the dataset, N_{ik} represent the number of objects that have the k-th smallest value in the i-th dimension, and S_i denote the set of objects with the missing values in the i-th dimension. The basic idea of our binning strategy is to employ one bit to encode a range of values. In other words, it partitions the observed values in one dimension into multiple bins with each bin capturing a range. The ranges w.r.t. two different bins are disjoint, and the ranges w.r.t. all the bins in the i-th dimension cover the domain of the values in the i-th dimension. Note that, we assume that the number of bins in the i-th dimension, denoted as $(\xi_i + 1)$, is specified, with ξ_i bins for all the observed values and one bin for the missing value.

Next, we explain how to partition the observed values in the i-th dimension into ξ_i bins. We first sort all the observed values in the i-th dimension based on ascending order, and then utilize Eq. (3) to determine the capacity of the first bin for the i-th dimension, denoted as b_{i1}. In particular, the first bin of the i-th dimension will cover first b_{i1} distinct values of that dimension, i.e., all the objects with their observed values in the i-th dimension falling with the range of $[min_i, v(b_{i1})]$ are accommodated by the first bin. Note that, $v(b_{i1})$ denotes the b_{i1}-th minimal observed value in the i-th dimension.

$$\sum_{k=1}^{b_{i1}} N_{ik} \approx (N - |S_i|)/\xi_i \tag{3}$$

Take our sample dataset as an example. For the first dimension ($i = 1$), we suppose $\xi_1 = 2$ and have $|S_1| = 10$ as there are 10 objects with missing observed values in this dimension. Among objects with observed values in the first dimension, there are four objects with the smallest value 2, four objects

ID	v_1	-	2	3-5	v_2	-	1-3	4-7	v_3	-	1-2	3	4-8	v_4	-	1-2	3	4-9
A_1	-	1	1	1	*3*	1	0	0	*1*	1	0	0	0	*3*	1	1	0	0
A_2	-	1	1	1	*1*	1	0	0	*2*	1	0	0	0	*1*	1	0	0	0
A_3	-	1	1	1	*1*	1	0	0	*3*	1	1	0	0	*4*	1	1	1	0
A_4	-	1	1	1	*7*	1	1	0	*4*	1	1	1	0	*5*	1	1	1	0
A_5	-	1	1	1	*4*	1	1	0	*8*	1	1	1	0	*3*	1	1	0	0
B_1	-	1	1	1	-	1	1	1	*1*	1	0	0	0	*2*	1	0	0	0
B_2	-	1	1	1	-	1	1	1	*3*	1	1	0	0	*1*	1	0	0	0
B_3	-	1	1	1	-	1	1	1	*4*	1	1	1	0	*9*	1	1	1	0
B_4	-	1	1	1	-	1	1	1	*3*	1	1	0	0	*7*	1	1	1	0
B_5	-	1	1	1	-	1	1	1	*7*	1	1	1	0	*4*	1	1	1	0
C_1	*2*	1	0	0	-	1	1	1	-	1	1	1	1	*3*	1	1	0	0
C_2	*2*	1	0	0	-	1	1	1	-	1	1	1	1	*1*	1	0	0	0
C_3	*3*	1	1	0	-	1	1	1	-	1	1	1	1	*2*	1	0	0	0
C_4	*3*	1	1	0	-	1	1	1	-	1	1	1	1	*3*	1	1	0	0
C_5	*3*	1	1	0	-	1	1	1	-	1	1	1	1	*4*	1	1	1	0
D_1	*3*	1	1	0	*5*	1	1	0	-	1	1	1	1	*2*	1	0	0	0
D_2	*2*	1	0	0	*1*	1	0	0	-	1	1	1	1	*4*	1	1	1	0
D_3	*2*	1	0	0	*4*	1	1	0	-	1	1	1	1	*1*	1	0	0	0
D_4	*4*	1	1	0	*4*	1	1	0	-	1	1	1	1	*5*	1	1	1	0
D_5	*5*	1	1	0	*5*	1	1	0	-	1	1	1	1	*4*	1	1	1	0

Fig. 9: The binned bitmap index for the dataset in Fig. 3

with the second smallest value 3, one object with the third smallest value 4, and one object with the largest value 5, i.e., $N_{11} = 4$, $N_{12} = 4$, $N_{13} = 1$, and $N_{14} = 1$. Based on Eq. (3), we have $(N - |S_1|)/\xi_1 = 10/2 = 5$. Consequently, b_{11} is set to 1, and the first bin covers $N_{11} = 4$ objects with the smallest value 2 (i.e., objects C_1, C_2, D_2, and D_3). Note that, although $N_{11} = 4$ is smaller than the capacity of the first bin, we cannot increase b_{11} to 2 as $N_{11} + N_{12} = 8 > 5$.

$$\sum_{k=b_{i1}+1}^{b_{i2}} N_{ik} \approx (N - |S_i| - \sum_{k=1}^{b_{i1}} N_{ik})/(\xi_i - 1) \tag{4}$$

Then, we can determine the value of b_{i2} according to Eq. (4). Note that, Eq. (4) is general, and it can help to approximate b_{ik} values for $1 < k < \xi_i$. Once the b_{ik} values for the first $(\xi_i - 1)$ bins are derived, the $v(b_{ik})$ value for the last bin (i.e., $k = \xi_i$) is set to max_i in order to cover the remaining objects. Back to our sample dataset, for the first dimension (i.e., $i = 1$), since $\xi_1 = 2$ and $v(b_{11}) = 2$, $v(b_{12})$ is set to $max_1(= 5)$, and the second bin will hold all the objects o with $o.[1] \in (2,5]$. If we set $\xi_1 = \xi_2 = 2$ and $\xi_3 = \xi_4 = 3$, the *binned* bitmap index for our sample dataset is shown in Fig. 9. Compared with the original bitmap index depicted in Fig. 6, the binning strategy can reduce the bitmap storage overhead efficiently.

Our binning strategy is *flexible* and *adaptive*. It can better accommodate the situation where there are more objects in one value than that in others. In particular, for uniformly distributed data, every bin generated by the strategy contains the same number of dimensional values. When the data distribution is not uniform, our binning strategy automatically adapts to the data distribution and minimizes the fluctuation in query processing. Note that, we will address the issue of how to choose a proper ξ in Section 4.5, based on the trade-off between index storage cost and query processing cost.

After we present the structural difference between the bitmap index and the binned bitmap index, we then explain the impact of the new binned bitmap index on the search algorithm. First, the content of set Q^i is different. In the binned bitmap index, given an object o, if $o.[i]$ is observed, set Q^i contains all the objects that are located in the same bin as $o.[i]$. Second, the bitmap pruning presented in Heuristic 2 is still applicable to binned bitmap index, whereas the statement $\mathsf{MaxBitScore}(o) \leq \mathsf{MaxScore}(o)$ presented in Lemma 3 is no longer valid. Consequently, the filtering based on $\mathsf{MaxBitScore}(o)$ under binned bitmap index might not be able to achieve a good pruning power. As an alternative, we

develop a new partial score pruning heuristic, as presented in Heuristic 3, to help prune away certain unqualified objects. Third, the detailed algorithm for score calculation under binned bitmap index, termed as BIG-Score, is slightly different from that under the original bitmap index.

Heuristic 3: **(partial score pruning)**. Given a TKD query over an incomplete dataset S and a candidate set S_C containing k objects, let τ be the smallest score for all objects in S_C. For a specified object $o \in (S - S_C)$, if $|\mathsf{nonD}(o)| > |Q| - |\Phi(o)| - \tau$, the object o can be discarded safely.

Proof. The proof is intuitive, and skipped for space saving. ■

As shown in Algorithm 5, IBIG-Score shares the same flow as BIG-Score and Get-Score. It first implements the filtering step based on Heuristic 2 (lines 2-4). If object o cannot be filtered, its score score(o) has to be derived based on $|\Gamma(o)| + |\Lambda(o)|$ (lines 5-14). Like BIG-Score, it derives $\Gamma(o)$ and forms nonD(o) (lines 6-13). Nevertheless, unlike BIG-Score, it implements Heuristic 3 (lines 11-12). Once nonD(o) is formed, $\Lambda(o)$ can be derived, and score(o) is obtained to complete IBIG-Score (lines 14-15).

As a summary, our IBIG algorithm is an improved version of BIG, and it shares the same framework as BIG. There are two major differences. First, IBIG is built on binned bitmap index, which employs an existing bitmap compression method (e.g., CONCISE in our implementation) and the binning strategy. Second, IBIG algorithm invokes IBIG-Score for score calculation while BIG algorithm relies on BIG-Score. As IBIG is similar as BIG in terms of query processing, we omit an illustrative example for space saving. It is worth pointing out that, in order to get the score of an object o, we utilize B$^+$-trees in our implementation to get the set nonD(o) quickly and to avoid unqualified checks. However, the usage of B$^+$-trees is optional, which depends on the trade-off between extra space cost and enhanced efficiency.

Algorithm 5 Get-Score Algorithm for IBIG (IBIG-Score)

Input: a *binned* bitmap index, an object o, $\Phi(o)$, τ, S_C, k
Output: the score of o, i.e., score(o)

1: get $[P^i]$ and $[Q^i]$ of o for each $1 \le i \le d$ from bitmap index
2: $Q \leftarrow \bigcap_{i=1}^{d} Q^i - \{o\}$, MaxBitScore($o$) $\leftarrow |Q|$
3: **if** $|S_C| = k$ and MaxBitScore(o) $\le \tau$ **then** // Heuristic 2
4: **return** 0 // o is pruned away
5: **else**
6: $P \leftarrow \bigcap_{i=1}^{d} P^i$, $\Gamma(o) \leftarrow P - \Phi(o)$
7: **for** each pair of $\langle p, i \rangle \in (Q - P) \times Iset(o)$ **do**
8: **if** $p.[i] = o.[i]$ **then** $p.tagT \leftarrow p.tagT + 1$
9: **else if** $p.[i] < o.[i]$ **then**
10: nonD(o) $\leftarrow$ nonD(o) $\cup \{p\}$
11: **if** $|S_C| = k$ and $|\mathsf{nonD}(o)| > |Q| - |\Phi(o)| - \tau$ **then**
12: **return** 0 // o is discarded by Heuristic 3
13: nonD(o) $\leftarrow$ nonD(o) $\cup \{p \in (Q - P) | p.tagT = |\beta_p \ \& \ \beta_o|\}$
14: $\Lambda(o) \leftarrow Q - P - \mathsf{nonD}(o)$
15: **return** score(o) $\leftarrow |\Gamma(o)| + |\Lambda(o)|$

4.5 Discussion

As mentioned earlier, the number of bins in the i-th dimension (i.e., ξ value) has a direct impact on the performance of IBIG algorithm. In the following, we present an analytical model to analyse the space cost and query processing cost affected by ξ, and discuss how to select a proper ξ value to optimize the *space-time* trade-off for IBIG. For ease of analysis, let N, d, and σ be the cardinality, the dimensionality, and the missing rate of the dataset, respectively. Then, the space cost, denoted as $cost_s$, is the size of the binned bitmap index, formally,

$$cost_s = N \times (\xi + 1) \times d \tag{5}$$

On the other hand, the query cost, denoted as $cost_t$, can be approximated by the cost incurred to form set nonD(o), as stated in Eq. (6). The reason is that the score calculation based on $|Q - \Phi(o) - \mathsf{nonD}(o)|$ relies on Q, $\Phi(o)$, and nonD(o). Given the fact that Q can be formed by fast bit-operations and $\Phi(o)$ is an input, the cost of score calculation is mainly contributed by the formation of set nonD(o). In addition, score calculation is the most expensive operation in TKD query processing, and its cost dominates the main query cost.

$$cost_t = d \times (\log(\sigma N) + \left\lceil \frac{\sigma N}{\xi} \right\rceil - 1) \tag{6}$$

In particular, for an object o that cannot be pruned away, IBIG has to form the set nonD(o) in order to obtain its real score. However, the cost of getting nonD(o) is related to the value of ξ. In our implementation, for each observed dimension of o, we need to traverse the B$^+$-tree to locate the minimum boundary of the bin where o is located, which takes $\log(\sigma N)$ cost. To further validate whether all the objects located in this bin are worse than o in this dimension, we need to access $(\lceil \sigma N/\xi \rceil - 1)$ key values via the sequential scanning in B$^+$-tree in the worst case. Consider that, in the worst case, we need to traverse all the B$^+$-trees, and thus, the total cost is $d \times (\log(\sigma N) + \lceil \sigma N/\xi \rceil - 1)$.

Based on Eq. (5) and Eq. (6), we can find that as the value of ξ grows, the space saving is reduced and the query processing cost drops, which will be confirmed in our experiments (to be presented in Section 5.1). This is because when ξ ascends, the range of dimensional values captured by every bin becomes small, and the average key value size (i.e., $\lceil \sigma N/\xi \rceil - 1$) visited by IBIG in every B^+ tree becomes small. In an extreme case when ξ is set to the number of distinct dimensional values (i.e., $\xi = C + 1$), the binned bitmap index is the same as the bitmap index built by BIG.

In other words, a small ξ value can help to cut down the index size efficiently but the query cost is increased. Hence, we cannot minimize both the index size and the query processing cost simultaneously. As both the space cost and the query cost are important performance metrics, and they both are affected by ξ, we consider the product of those two costs as the main cost, as shown in Eq. (7).

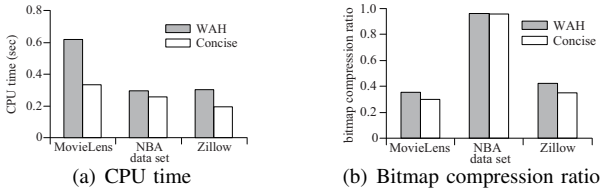
$$\begin{aligned} cost &= cost_s \times cost_t \\ &= N \cdot (\xi + 1) \cdot d^2 \cdot (\log(\sigma N) + \left\lceil \frac{\sigma N}{\xi} \right\rceil - 1) \end{aligned} \tag{7}$$

$$\xi = \sqrt{\frac{\sigma N}{\log(\sigma N) - 1}} \tag{8}$$

Note that, some existing works have also adopted the same equation to analyze the cost [30]. Therefore, in order to optimize the trade-off between space cost and query cost, we try to minimize *cost*, i.e., we set the derivative of *cost* to be zero and obtain the optimal value of ξ as depicted in Eq. (8).

TABLE 2: Parameter ranges and default values

Parameter	Range
k (of top-k dominating query)	4, **8**, 16, 32, 64
Number of objects N	50K, **100K**, 150K, 200K, 250K
Dimensionality dim	5, **10**, 15, 20, 25
Missing rate σ(%)	0, 5, **10**, 20, 30, 40
Dimensional cardinality c	50, **100**, 200, 400, 800



(a) CPU time (b) Bitmap compression ratio

Fig. 10: WAH vs. CONCISE

As an example, for N =100K and $\sigma = 0.1$, we can get the optimal bin size $\xi = 29$. When N =16K and $\sigma = 0.2$, the optimal bin size ξ is 17. We will further verify the accuracy of our analysis in our empirical study.

Before we end our discussion, we analyze the time complexities of our proposed algorithms. Let N be the dataset cardinality. First, ESB and UBB algorithms take $O(N^2)$ time for TKD query processing. This is because both algorithms might need to derive the score for each object in the worst case, and score calculation via the basic Get-Score(o) function has to conduct pairwise comparisons between o and the other $(N-1)$ objects. Second, both BIG and IBIG have their complexities as $O(N^2)$, as the score computation of one object using bitwise operations can be finished in $O(N)$ and they both need to check all the objects in the worst case. However, we would like to highlight that BIG and IBIG algorithms are much more efficient than ESB and UBB algorithms, with the help of fast bit-wise operations based on the (binned) bitmap index, as demonstrated in our experimental results.

5 Experimental Evaluation

In this section, we verify the effectiveness of the bitmap compression and the binning strategy of bitmap index, and evaluate the performance of our proposed algorithms for TKD queries over incomplete data. All algorithms are implemented in Java SE7, and all experiments are conducted on an Intel Core i5 Duo 3.10GHz PC with 4GB RAM, running Microsoft Windows 7 Professional Edition.

In our experiments, we use both real and synthetic datasets. For real datasets, we employ *MovieLens*, *NBA*, and *Zillow* that are widely utilized in many previous works on dominance problem [1], [2], [5], [7], [8], [9], [21], [22], [23]. (i) ***MovieLens*** contains 3,700 movie records, with each having 60 dimensions representing the ratings from 60 audiences. The rating values vary from 1 to 5, and the missing rate of *MovieLens* is 95%, i.e., only 5% of the ratings are available. (ii) ***NBA*** includes the complete records of 16,000 NBA players in multiple dimensions. We extract 4 attributes, i.e., *game played*, *minutes played*, *total points*, and *offensive rebounds*, and remove some statistics randomly to achieve 20% missing rate. (iii) ***Zillow*** contains 200,000 entries about real estate in the United States with five attributes, namely, number of bedrooms, number of bathrooms, living area, lot area, and estimated price, with missing rate of 14.2%. In addition,

TABLE 3: Preprocessing time (in seconds) of proposed algorithms

Dataset	MaxScore	Φ	Bitmap index	Binned bitmap index
MovieLens	0.02	0.1	0.3	0.1
NBA	1.3	0.5	44.8	1.1
Zillow	3.8	24.5	5749.3	1049.7
IND	0.7	22.3	225.1	70.6
AC	0.7	21.9	201.3	61.9

synthetic datasets are generated based on *independent* (*IND*) and *anti-correlated* (*AC*) distributions respectively, following the common methodology used in [32]. We remove some attribute values randomly to simulate the incomplete datasets.

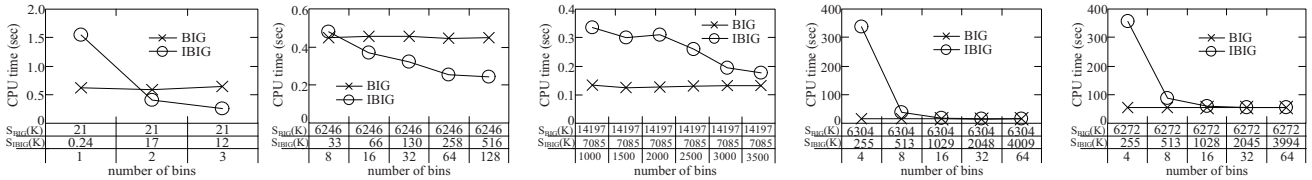
We explore several factors in our experiments, including k, cardinality N, dimensionality dim, missing rate σ (i.e., the probability that $o.[i]$ is missing for any object o in S), and dimensional cardinality c (i.e., the number of distinct values on one dimension). The settings of all these parameters are summarized in Table 2, where the default values are shown in bold. In every set of experiments, we only change one parameter, with the rest set to their defaults. In addition, in our experiments, the intuitive Naive method (mentioned in Section 4.1) is also implemented as the baseline for TKD queries on incomplete data, in addition to our newly proposed ESB, UBB, and BIG (IBIG) algorithms.

5.1 Bitmap Compression and Binning Strategy

The first set of experiments evaluates the efficiency of two compression approaches, i.e., WAH and CONCISE, on real datasets for IBIG algorithm. We report the CPU time and the bitmap compression ratio in Fig. 10. Here, the bitmap compression ratio is the ratio of the size of a compressed bitmap to that of its original bitmap. It is observed that CONCISE has slightly better compression ratio and less CPU time than WAH. Hence, in the rest of experimental evaluation, we use CONCISE compression technique for IBIG algorithm. Note that, our results are consistent with the discussion presented in [31], i.e., CONCISE is better than WAH. In addition, we also observe that *NBA*, because of its distribution, is not suitable for compression, as its compression ratio is near to 1. This also shows that existing compression techniques might not be able to reduce the size of bitmap index efficiently.

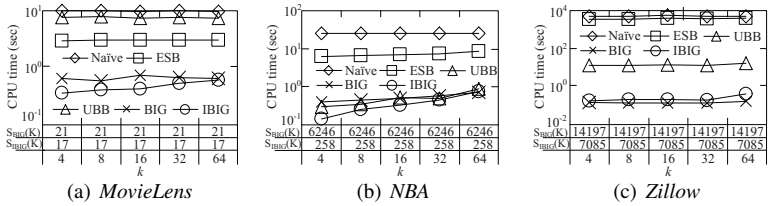
The second set of experiments verifies the effect of the binning strategy on the performance of IBIG algorithm under different bin size ξs, compared against that of BIG algorithm. The CPU time and the bitmap index sizes are reported in Fig. 11 by varying ξ values. Here, we assume that ξ_is in different dimensions share the same value ξ on *MovieLens*, *NBA*, *IND*, and *AC* datasets, since they share similar domain in different dimensions. For *Zillow*, there are 6, 10, 35, ξ, and 1000 bins w.r.t. the five dimensions (i.e., number of bedrooms, number of bathrooms, living area, lot area, and price), because these five domains are very different.

Consistent with our expectation, as ξ value increases, the efficiency of IBIG is improved while the index size (with S_{BIG} and S_{IBIG} being the sizes of original bitmap index and binned bitmap index respectively) also grows. As the number of bins ascends, the binned bitmap index requires more space to index bins, and hence, the space saving from the binning strategy is less significant. Note that, BIG incurs shorter query time than IBIG in some cases (with small bin size). This is because, the smaller the bin size, the more the objects in every bin,



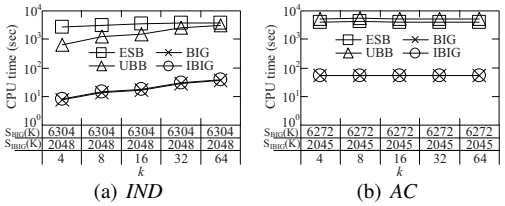
(a) *MovieLens* (b) *NBA* (c) *Zillow* (d) *IND* (e) *AC*

Fig. 11: TKD cost on incomplete data vs. ξ



(a) *MovieLens* (b) *NBA* (c) *Zillow*

Fig. 12: TKD cost on incomplete real data vs. k



(a) *IND* (b) *AC*

Fig. 13: TKD cost on incomplete synthetic data vs. k

TABLE 4: Dissimilarity (denoted as D_J) of the TKD query result between ours and the one based on missing value inference method

k	4	8	16	32	64
D_J	0.400	0.400	0.476	0.476	0.562

resulting in more validation cost for IBIG algorithm. However, the storage cost of BIG is much more expensive than that of IBIG in all cases. In addition, the index size of IBIG under *Zillow* remains almost unchanged under different ξ values, since the bin size only on one dimension increases with other setting to their defaults. In general, IBIG with the binning strategy does effectively cut down the storage cost, and its CPU cost in most cases (e.g., when ξ is not too small) is comparable with that under original BIG.

Recall that, based on the analysis presented in Section 4.5, the optimal bin size ξ is near 29 when N = 100K and $\sigma = 0.1$ for *IND* and *AC*. For *NBA* with N = 16K and $\sigma = 0.2$, the optimal bin size ξ is derived near 17. They are consistent with our experimental results. It is worth mentioning that, we will further improve the space-time trade-off discussion in our future work for the datasets with different/special dimensional domains, such as *MovieLens* with domain range [1, 5] and *Zillow* with very different dimension domains. Without loss of generality, in the rest of experiments, we employ IBIG algorithm with 2, 64, 3000, 32, and 32 bins for *MovieLens*, *NBA*, *Zillow*, *IND*, and *AC* respectively.

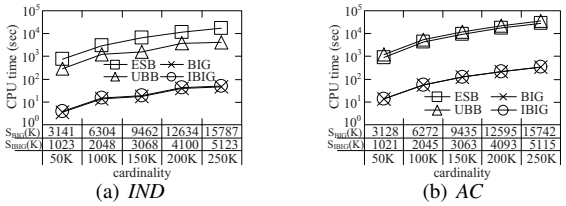
5.2 Results on TKD Queries over Incomplete Data

In this subsection, we first report the preprocessing time incurred by MaxScore and Φ computation, (original) bitmap index, and *binned* bitmap construction in Table 3 under default settings. Note that, constructing original bitmap index is more costly than constructing the binned one. This is because, the original bitmap index needs to create and maintain more bit-vectors than the binned one, resulting in more overhead. In the following, we study the impact of various parameters on the performance of TKD query processing algorithms.

For **real datasets**, we explore the influence of k on the efficiency of algorithms. Fig. 12 shows the experimental results when k varies from 4 to 64. Obviously, in all cases, BIG and IBIG outperform ESB, UBB, and Naive algorithms, and all the algorithms incur more CPU time as k grows. This is because BIG and IBIG utilize three effective heurisitcs to reduce the candidate set size, and they both derive the scores of objects via faster bitwise operations. When k ascends, more candidate objects have to be evaluated, resulting in longer CPU time. Furthermore, under the comparable query time, IBIG incurs a much smaller space cost, as compared with BIG (i.e., $S_{IBIG} < S_{BIG}$). In addition, we also observe that the advantage of BIG (IBIG) over UBB becomes less significant in *NBA* dataset. The reason is that the upper bound score (i.e., MaxScore) used in UBB is rather tight for *NBA*, and it prunes a large number of objects, which leaves very limited space of improvement for the bitwise operations and heuristics employed by BIG (IBIG) to demonstrate their power. Note that, due to space limitation and the big performance gap among algorithms, we have to plot the performance in log scale. Under this plot setting, UBB seems to be very close to IBIG on *NBA* in terms of performance. Nonetheless, we would like to highlight that IBIG constantly outperforms UBB under *NBA* dataset with different k values. On the other hand, the advantage of BIG (IBIG) over ESB and UBB is significant under other datasets. Also notice that, as discussed in Section 4.1, Naive is clearly inferior to other algorithms and hence is skipped in the rest of experiments.

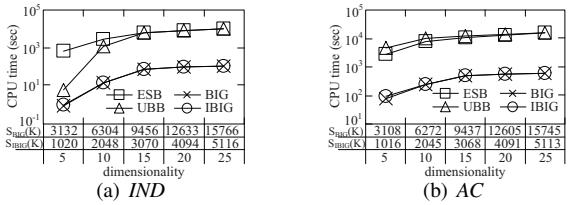
In addition, in order to show how the TKD query result on incomplete data (denoted as A) is close to the one based on missing value inference method (denoted as B), we report the Jaccard distance D_J $(= 1 - \frac{|A \cap B|}{|A \cup B|})$ between A and B on *NBA* dataset under various k values in Table 4. Note that, in order to get the TKD query result based on missing value inference method, we use GraphLab Create implementation (https://dato.com/products/create/) to predict the missing values, where factorization model is chosen and all parameters are set to defaults with the number of factors set to 8 and L2 regularizations used on the factors, and the optimization process is iterated at a maximum of 50 times. Consider that, for the TKD query, $D_J = 1 - \frac{|A \cap B|}{2k - |A \cap B|}$, and D_J is a decreasing function for $|A \cap B|$. If A and B share $\frac{k}{2}$ answer objects, the Jaccard distance D_J is $1 - \frac{\frac{k}{2}}{2k - \frac{k}{2}}$ $(= \frac{2}{3})$. Since all the D_J in Table 4 are smaller than $\frac{2}{3}$, it means that the number of shared answer objects between A and B is larger than $\frac{k}{2}$ (i.e., they share more than half of answer objects).

For **synthetic datasets**, we evaluate the influence of param-



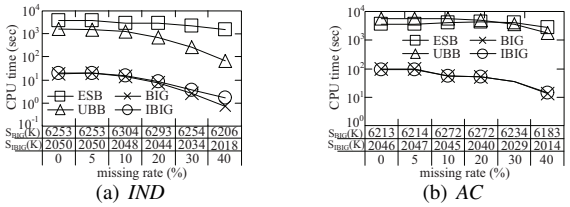
(a) *IND* (b) *AC*

Fig. 14: TKD cost on incomplete synthetic data vs. N



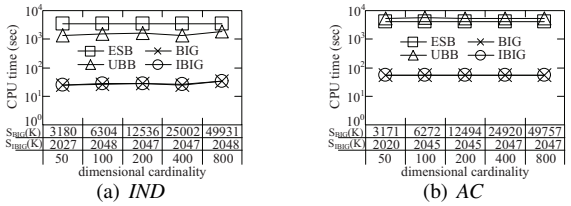
(a) *IND* (b) *AC*

Fig. 15: TKD cost on incomplete synthetic data vs. dim



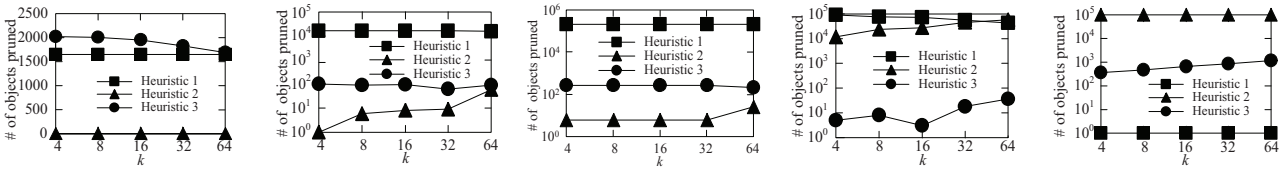
(a) *IND* (b) *AC*

Fig. 16: TKD cost on incomplete synthetic data vs. σ



(a) *IND* (b) *AC*

Fig. 17: TKD cost on incomplete synthetic data vs. c



(a) *MovieLens* (b) *NBA* (c) *Zillow* (d) *IND* (e) *AC*

Fig. 18: Pruning heuristic efficiency vs. k for TKD queries on incomplete data

eters k, N, dim, missing rate σ, and dimensional cardinality c on the performance of algorithms. First, we report the results under various k in Fig. 13. Similar to the results on real datasets, BIG and IBIG perform much better than ESB and UBB, and S_{IBIG} is much smaller than S_{BIG}.

Second, we explore the impact of cardinality N by changing N between 50K and 250K, with the results plotted in Fig. 14. Again, BIG and IBIG are better than other algorithms, and CPU time increases with the growth of N. This is because the cost of score computation for candidate TKD objects increases as N ascends, and BIG and IBIG minimize the cost via reducing the candidate set size using effective pruning heuristics and simplify the score computation via fast bitwise operations.

Then, Fig. 15 depicts the experimental results under different dimensionality dim. As expected, BIG and IBIG perform the best, and CPU time increases with dim. As dim grows, the cost for an object to be compared with other objects (i.e., score computation) ascends, and therefore, query cost grows.

Next, we vary missing rate σ in the range [0, 40%], and report the results in Fig. 16. Evidently, BIG and IBIG consistently perform much better than ESB and UBB. Note that, CPU time for the algorithms drops as σ grows. The reason is that, with the growth of missing rate σ, the number of comparable objects with respect to a given object decreases. Thus, the score computation cost drops, and CPU time decreases.

Finally, Fig. 17 shows the algorithm costs by varying dimensional cardinality c. Obviously, BIG and IBIG outperform others in all cases. It is observed that CPU time is not very sensitive to c, especially for AC dataset, since the query result does not fluctuate significantly with various c.

To sum up, based on the above experimental results, we can conclude that BIG and IBIG perform the best, followed by UBB, and ESB is the worst. It is worth noting that, under comparable query time, IBIG consumes less storage than BIG in all cases. Hence, in real-life applications with costly/limited storage space, IBIG is a more promising choice than BIG.

5.3 Effectiveness of Pruning Heuristics

The last set of experiments evaluates the effectiveness of our developed heuristics. Fig. 18 plots the number of the objects pruned under different k values on IBIG algorithm. It is observed that, the upper bound score pruning (i.e., Heuristic 1) is effective for *MovieLens*, *NBA*, *Zillow*, and *IND*, but not for *AC*. This is because the dataset following anti-correlated distribution has rather small k-th highest score, indicating that the condition of Heuristic 1 is hard to satisfy under *AC*. The bitmap pruning (i.e., Heuristic 2) is effective for *NBA*, *Zillow*, *IND*, and *AC*, but not for *MovieLens*. The reason is that, for *MovieLens*, the MaxBitScore of objects is rather loose due to its high missing ratio, i.e., 95%. The partial score pruning (i.e., Heuristic 3) is effective for all datasets. Note that, the three heuristics are not run in parallel, i.e., in Fig. 18, the number of the objects pruned by Heuristic 2 excludes those pruned by Heuristic 1, and the number of the objects pruned by Heuristic 3 excludes those pruned by Heuristic 1 and Heuristic 2. Because of different distributions, all three heuristics perform differently in different datasets. Nonetheless, they all can help to shrink the search space.

6 Conclusions

Consider the wide range of applications for top-k dominating (TKD) queries and the pervasiveness of incomplete data, we, in this paper, study the problem of the *TKD query on incomplete data* where some dimensional values are missing. To efficiently address this, we first propose ESB and UBB algorithms, which utilize novel techniques (i.e., local skyband technique and upper bound score pruning) to prune the search space. In order to further reduce the cost of score computation, we present BIG algorithm, which employs the upper bound

score pruning, the bitmap pruning and fast bitwise operations based on the bitmap index to improve the score computation and boost query performance accordingly. Moreover, in order to trade efficiency for space, we propose IBIG algorithm by using the bitmap compression technique and the binning strategy over BIG, and develop a method to choose the appropriate number of bins. Considerable experimental results on both real and synthetic datasets confirm the effectiveness and efficiency of our presented heuristics and algorithms. In the future, we will further study how to improve the quality of TKD query over incomplete data.

Acknowledgments. This work was supported in part by 973 Program 2015CB352502, NSFC Grants 61379033 and 61472348, and the Fundamental Research Funds for the Central University.

REFERENCES

[1] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data," in *ICDE*, pp. 556–565, 2008.
[2] Y. Gao, X. Miao, H. Cui, G. Chen, and Q. Li, "Processing *k*-skyband, constrained skyline, and group-by skyline queries on incomplete data," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4959–4974, 2014.
[3] X. Lian and L. Chen, "Top-*k* dominating queries in uncertain databases," in *EDBT*, pp. 660–671, 2009.
[4] X. Lian and L. Chen, "Probabilistic top-*k* dominating queries in uncertain databases," *Inf. Sci.*, vol. 226, pp. 23–46, 2013.
[5] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.
[6] M. L. Yiu and N. Mamoulis, "Efficient processing of top-*k* dominating queries on multi-dimensional data," in *VLDB*, pp. 483–494, 2007.
[7] M. L. Yiu and N. Mamoulis, "Multi-dimensional top-*k* dominating queries," *VLDB J.*, vol. 18, no. 3, pp. 695–718, 2009.
[8] W. Zhang, X. Lin, Y. Zhang, J. Pei, and W. Wang, "Threshold-based probabilistic top-*k* dominating queries," *VLDB J.*, vol. 19, no. 2, pp. 283–305, 2010.
[9] E. Tiakas, G. Valkanas, A. N. Papadopoulos, Y. Manolopoulos, and D. Gunopulos, "Metric-based top-*k* dominating queries.," in *EDBT*, pp. 415–426, 2014.
[10] L. Antova, C. Koch, and D. Olteanu, "From complete to incomplete information and back," in *SIGMOD*, pp. 713–724, 2007.
[11] T. J. Green and V. Tannen, "Models for incomplete and probabilistic information," in *EDBT*, pp. 278–296, 2006.
[12] T. Imieliński and W. Lipski Jr, "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, pp. 761–791, 1984.
[13] R. van der Meyden, "Logical approaches to incomplete information: A survey," in *Logics for databases and information systems*, pp. 307–356, 1998.
[14] G. Canahuate, M. Gibas, and H. Ferhatosmanoglu, "Indexing incomplete databases," in *EDBT*, pp. 884–901, 2006.
[15] B. C. Ooi, C. H. Goh, and K.-L. Tan, "Fast high-dimensional data search in incomplete databases," in *VLDB*, pp. 357–367, 1998.
[16] C. Lofi, K. El Maarry, and W.-T. Balke, "Skyline queries in crowd-enabled databases," in *EDBT*, pp. 465–476, 2013.
[17] W. Cheng, X. Jin, J. Sun, X. Lin, X. Zhang, and W. Wang, "Searching dimension incomplete databases," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 725–738, 2014.
[18] P. Haghani, S. Michel, and K. Aberer, "Evaluating top-*k* queries over incomplete data streams," in *CIKM*, pp. 877–886, 2009.
[19] M. A. Soliman, I. F. Ilyas, and S. Ben-David, "Supporting ranking queries on uncertain and incomplete data," *VLDB J.*, vol. 19, no. 4, pp. 477–501, 2010.
[20] J. Graham, *Missing data: Analysis and design*. Statistics for Social and Behavioral Sciences, Springer, 2012.
[21] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos, "Progressive processing of subspace dominating queries," *VLDB J.*, vol. 20, no. 6, pp. 921–948, 2011.
[22] M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos, "Continuous top-*k* dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 5, pp. 840–853, 2012.
[23] B. Santoso and G. Chiu, "Close dominance graph: An efficient framework for answering continuous top-*k* dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1853–1865, 2014.
[24] X. Han, J. Li, and H. Gao, "Tdep: efficiently processing top-*k* dominating query on massive data," *KAIS*, pp. 1–30, 2014.
[25] L. Zhan, Y. Zhang, W. Zhang, and X. Lin, "Identifying top *k* dominating objects over uncertain data," in *DASFAA*, pp. 388–405, 2014.
[26] R. J. Little and D. B. Rubin, *Statistical analysis with missing data, Second edition*. Wiley, 2002.
[27] D. Sacharidis, P. Bouros, and T. Sellis, "Caching dynamic skyline queries," in *SSDBM*, pp. 455–472, 2008.
[28] K.-L. Tan, P.-K. Eng, and B. C. Ooi, "Efficient progressive skyline computation," in *VLDB*, pp. 301–310, 2001.
[29] K. Wu, E. J. Otoo, and A. Shoshani, "Compressing bitmap indexes for faster search operations," in *SSDBM*, pp. 99–108, 2002.
[30] K. Wu, A. Shoshani, and K. Stockinger, "Analyses of multi-level and multi-component compressed bitmap indexes," *ACM Trans. Database Syst.*, vol. 35, no. 1, p. 2, 2010.
[31] A. Colantonio and R. Di Pietro, "CONCISE: Compressed 'n' composable integer set," *Inf. Process. Lett.*, vol. 110, no. 16, pp. 644–650, 2010.
[32] S. Borzsonyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, pp. 421–430, 2001.

Xiaoye Miao received the BS degree in computer science from Xi'an Jiaotong University, China, in 2012. She is currently working toward the PhD degree in the College of Computer Science, Zhejiang University, China. Her research interests include uncertain and incomplete databases.

Yunjun Gao received the PhD degree in computer science from Zhejiang University, China, in 2008. He is currently an associate professor in the College of Computer Science, Zhejiang University, China. His research interests include spatial and spatio-temporal databases, metric and incomplete/uncertain data management, and spatio-textual data processing. He is a member of the ACM and the IEEE, and a senior member of the CCF.

Baihua Zheng received the PhD degree in computer science from Hong Kong University of Science & Technology, China, in 2003. She is currently an associate professor in the School of Information Systems, Singapore Management University, Singapore. Her research interests include mobile/pervasive computing and spatial databases.

Gang Chen received the PhD degree in computer science from Zhejiang University, China. He is currently a professor in the College of Computer Science, Zhejiang University, China. His research interests include relational database systems and large-scale data management. He is a member of the ACM and a senior member of the CCF

Huiyong Cui received the BS degree in computer science from Zhejiang University of Technology, China, in 2013. He is currently working toward the MS degree in the College of Computer Science, Zhejiang University, China. His research interest includes uncertain and incomplete databases.