Information Sciences 544 (2021) 343–371

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Effective and efficient top-*k* query processing over incomplete data streams

Weilong Ren [a], Xiang Lian [a,*], Kambiz Ghazinour [a,b]

[a] Department of Computer Science, Kent State University, Kent, OH 44242, USA
[b] Center for Criminal Justice, Intelligence and Cybersecurity, State University of New York, Canton, NY 13617, USA

ARTICLE INFO

Article history:
Received 25 February 2020
Received in revised form 27 July 2020
Accepted 5 August 2020
Available online 5 September 2020

Keywords:
Top-*k* query
Incomplete data streams
Top*k*-iDS

ABSTRACT

Nowadays, efficient and effective stream processing has become increasingly important in many real-world applications such as sensor data monitoring, network intrusion detection, IP network traffic analysis, and so on. In practice, stream data often encounter the problem of having some data attributes missing, due to reasons such as packet losses, network congestion/failure, and so on. In such a scenario, it is rather important, yet challenging, to accurately and efficiently monitor top-*k* objects over incomplete data stream, which may potentially indicate some dangerous and critical security events (e.g., fire, network intrusion, or denial-of-service attack). In this paper, we formally define the problem of top-*k* query over incomplete data stream (Top*k*-iDS), which continuously detects top-*k* objects with the highest ranking scores over an incomplete data stream. Due to unique characteristics such as incompleteness and stream processing, we propose a cost-model-based data imputation approach, design effective pruning strategies to reduce the Topk-iDS search space, and carefully devise dynamically updated data synopses to facilitate Top*k*-iDS query processing. We also propose an efficient algorithm to perform the data imputation and incremental Topk-iDS computation at the same time. Finally, through extensive experiments, we evaluate the efficiency and effectiveness of our proposed Topk-iDS query answering approach over both real and synthetic data sets..

1. Introduction

Stream data processing has received much attention from the database community, due to a wide spectrum of real-world applications such as sensor data monitoring [1], network intrusion detection [2], IP network traffic analysis [3], and so on. In reality, due to various reasons such as the network congestion/delay, data lost during the transmission, and/or device failures, sometimes, data objects from streams may not be fully available (i.e., some attributes might be missing). It is thus rather challenging to effectively and efficiently manage and query such streams with incomplete data objects.

In this paper, we consider a classic and important problem, the *top-k query* over data streams, with missing attributes in streaming objects. We have the following motivation example of monitoring security events (e.g., fire) in a forest.

Example 1. (Forest Security Monitoring) Consider an example of the forest security monitoring in Fig. 1, where 5 sensors, $o_1 \sim o_5$, are deployed in a forest. Each sensor o_i $(1 \leqslant i \leqslant 5)$ is in charge of a forest area, and collects sensory data (i.e., periodically received attributes) such as the temperature, the density of oxygen, and sunlight intensity (as depicted in

* Corresponding author.
E-mail addresses: wren3@kent.edu (W. Ren), xlian@kent.edu (X. Lian), kghazino@kent.edu (K. Ghazinour).

https://doi.org/10.1016/j.ins.2020.08.011
0020-0255/

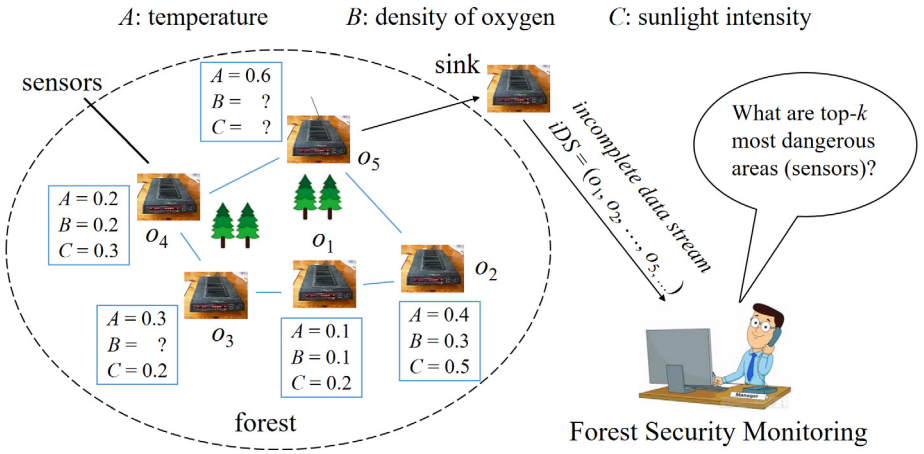

Fig. 1. An example of the top-k query for forest security monitoring (attributes A, B and C are normalized).

Table 1
An incomplete data stream, *iDS*, collected from sensor networks in Fig. 1 (with the ranking function $\mathcal{F}(o_i) = o_i[A] + o_i[B] + o_i[C]$).

sensor ID	arrival time	[A] temperature	[B] density of oxygen	[C] sunlight intensity	ranking score
o_1	1	0.1	0.1	0.2	0.4
o_2	2	0.4	0.3	0.5	1.2
o_3	3	0.3	–	0.2	**?**
o_4	4	0.2	0.2	0.3	0.7
o_5	5	0.6	–	–	**?**
...	...	...	...	...	...

Table 1) in real time. Since forest security events (e.g., fire) may lead to serious consequences such as personal safety, biodiversity losses, and/or financial losses, it is very important to monitor and prevent such dangerous events.

In the example of Fig. 1, a forest security monitoring system will monitor real-time streaming information, $iDS = (o_1, o_2, \ldots, o_5, \ldots)$, from sensors, and obtain top-k sensors with the highest risks that forest security events (e.g., fire) occur. Assume that we use a simple ranking function $\mathcal{F}(\cdot)$ that sums up all the 3 attributes of each sensor. Intuitively, high ranking scores $\mathcal{F}(\cdot)$ indicate high risk levels of the sensors. Then, we can retrieve top-k sensors $o_i \in iDS$ with the highest ranking scores (e.g., sensor o_4 has high score 0.7), as shown in Table 1.

In practice, however, due to hardware failure or packet losses, attributes from some sensors (objects) may not be available (e.g., the *density of oxygen* of sensor o_3 and the *sunlight intensity* for sensor o_5), as shown in Table 1. Thus, in order to monitor critical events such as fire, we should conduct the top-k query over such an incomplete network data stream accurately and efficiently, which reports k sensors with the highest risks to forest security people. ■

In the example above, there are two major challenges for monitoring top-k query answers (i.e., k most risky sensors with fire events) over incomplete data stream. First, the ranking function $\mathcal{F}(\cdot)$ may involve missing attributes in objects from incomplete data stream *iDS*. It is not trivial how to obtain the ranking scores of incomplete objects with missing attributes. While existing works [4] simply ignored the missing attributes, by setting their values (e.g., *density of oxygen and sunlight intensity*) to 0, the resulting top-k query may incur inaccurate (low or high) ranking scores, and in turn erroneous query results. For example, incomplete object o_5 in Table 1 tends to have low (biased) ranking score, if ignoring missing attributes $o_5[B]$ and $o_5[C]$. Thus, alternatively, we need to design an accurate data imputation approach to impute the missing attributes.

Second, it is also challenging to efficiently and accurately retrieve the top-k query answers over incomplete data streams. Due to unique stream processing requirements (e.g., high-speed processing and limited memory consumption), we need to design effective pruning strategies to quickly filter out false alarms of top-k query answers, and devise effective and space-efficient data synopses to enable both data imputation and incremental top-k answer computations at the same time. Note that, a straightforward method to answer top-k queries over incomplete data stream is to first impute incomplete objects, followed by top-k query answering, which may not suit for fast processing requirement in the stream environment. Thus, in this paper, we explore the style of performing the "data imputation and top-k query processing at the same time", which, to our best knowledge, has not been studied before.

1.1. The Top*k*-iDS problem.

Inspired by the example above, in this paper, we formally define the *top-k query over incomplete data stream* (Top*k*-iDS), which continuously monitors and retrieves objects o_i from incomplete data streams *iDS*, that have the highest top-*k* ranking scores with high confidences. To tackle the challenges of Top*k*-iDS query answering, we propose an effective and efficient approach to enable the data imputation and Top*k*-iDS query answering over incomplete data stream at the same time.

In addition to the forest fire monitoring (as mentioned in Example 1), the Top*k*-iDS problem can also be applied to other application domains such as IP network traffic analysis or network intrusion detection. For example, in the application of IP network traffic analysis, when the network traffic is busy, the routers may drop packets and affect user experience. Thus, it is important to conduct the Top*k*-iDS query to effectively and quickly detect top-*k* busiest routers (ranked by network traffic factors, some of which are missing due to packet losses), and adaptively control online network traffic.

Similarly, for network intrusion detection, routers (connected with a number of servers) are usually distributed at different locations of a computer network and receive statistical attributes such as *No. of connections*, *connection duration*, and *transferred data size*, which can be used to detect and prevent network attacks such as network congestion, malicious virus installation, denial-of-service, and/or leakage of users' information. However, due to the network delay or packages losses, statistical attributes from some routers may not be available. In this case, we can issue a Top*k*-iDS query over (incomplete) network streaming data to retrieve top-*k* risky routers that are under attack.

1.2. ***Differences from prior works***

While prior works [4,5] studied the top-*k* query over incomplete data stream, they assume that the data incompleteness is caused by different arrival timestamps of object attributes, and this incompleteness will be fixed when the missing attributes arrive at the data stream. Thus, data attributes in prior works are only temporarily missing. In contrast, in this paper, we assume that some attributes of objects in the data stream are *permanently* missing, due to packet losses or environmental factors. Furthermore, previous techniques [4,5] set the missing attribute values to 0 (while attribute values are temporarily missing), which underestimates the ranking scores and may cause wrong decision making. To solve this problem, in this paper, we adopt *differential dependency* (DD) rules [6] to turn incomplete data into complete (imputed) ones. To our best knowledge, no previous works explored the top-*k* query over incomplete data stream in the presence of permanently missing attributes. For the comparison of our work with top-*k* queries on uncertain data [7–9], please refer to Section 2.

Most importantly, in order to efficiently and effectively process Top*k*-iDS queries over incomplete data stream, we specifically design the cost-model-based and space-efficient index structures for both data imputation and query processing, explore efficient pruning strategies to reduce the imputation space and query time, and propose efficient Top*k*-iDS query answering algorithms to enable the data imputation and query processing at the same time.

In this paper, we make the following major contributions:

1. We formalize an important problem of the *top-k query over incomplete data streams* (Top*k*-iDS) in Section 3.
2. We propose effective and efficient cost-model-based data imputation techniques via DD rules in Section 4.
3. We devise effective pruning strategies to reduce the search space of the Top*k*-iDS problem in Section 5.
4. We design effective indexes and efficient algorithms to tackle the Top*k*-iDS problem in Section 6.
5. Extensive experiments have demonstrated the efficiency and effectiveness of our proposed Top*k*-iDS processing approach on real/synthetic data sets in Section 7.

In addition, Section 2 reviews related works on stream processing, differential dependency, top-*k* queries, and incomplete databases. Section 8 concludes this paper.

2. Related work

2.1. Stream processing

Stream processing usually requires fast processing speeds and limited memory consumption. Many previous works studied various queries over data streams, for example, the top-*k* query [10–12], join [13], aggregate queries [14], nearest neighbor queries [15], skyline queries [16], keyword search [17], event detection [18], and so on. These works often assumed that the underlying stream data are complete. Therefore, their proposed techniques cannot be directly applied to our Top*k*-iDS problem in the scenario of incomplete data streams.

2.2. Differential dependency

Differential dependency (DD) [6] can be used for data cleaning [19], data repairing [20], data imputation [21], and so on. Prokoshyna et al. [19] detected records that violate DDs and cleaned those inconsistent records. Moreover, DD can be also used for constraint-based data repairs over various data types like graphs [20]. Song et al. [20] utilized DDs to repair vertex

labels in network graphs. Song et al. [21] applied DD rules to impute the missing attributes via extensive similarity neighbors with the same determinant attributes on static databases. In contrast, our work considers not only the data imputation over streams (instead of static databases), but also top-k query processing at the same time. Thus, it is more challenging to efficiently tackle the Top$k$-iDS problem.

2.3. Top-k queries over certain/uncertain databases

Prior works on top-k query processing over certain data explored many techniques such as PREFER [22], the *threshold algorithm* (TA) [23], branch-and-bound ranked search (BRS) [24], and so on. Regarding probabilistic top-k queries, different top-k semantics were proposed over static probabilistic databases, such as U-Top$k$ [25], U-kRanks [25], probabilistic threshold top-k (PT-k) [7], probabilistic k top-k (Pk-top$k$) [9], a unified top-k approach [26], and so on. Specifically, U-Top$k$ [25] returns a vector of top-k highest ranked objects with the highest probability for all possible worlds; U-kRanks [25] obtains k objects, where the i-th ($1 \leqslant i \leqslant k$) returned object has the highest probability to have rank i in possible worlds; PT-k [7] retrieves a set of objects with high probabilities (greater a threshold) to be ranked top-k among all possible worlds; and Pk-top$k$ [9], a variant of the PT-k semantics, returns k objects with the highest probabilities to be ranked top-k among all possible worlds.

In this paper, we adopt the PT-k semantics for monitoring top-k answers over the sliding window in incomplete data streams. Previous works on top-k queries over certain/uncertain data usually considered queries over complete data (rather than incomplete data). Thus, we cannot directly apply previous top-k techniques on (static) complete data to solving our Top$k$-iDS problem over incomplete data streams.

Some previous works [4,5] considered top-k query processing over data streams, where some attribute values have transmission delays (however, finally these attributes will arrive). When some attributes are not available, these works simply set these attribute values to 0 [4]. This might return inaccurate or even wrong top-k answer set, and may not instantly report dangerous events (e.g., computer network intrusion). In contrast, our Top$k$-iDS problem assumes that some attributes are missing permanently (e.g., due to packet losses), and imputes the missing attributes via DDs. With a different assumption about the data model, we cannot use previous approaches to tackle our Top$k$-iDS problem.

Furthermore, for our Top$k$-iDS problem, it is not efficient to conduct the data imputation before the top-k query answering. Therefore, we follow the style of "data imputation and query processing at the same time", which is quite different from previous works on top-k query processing on certain [11] or static uncertain database [7,9].

2.4. ***Incomplete databases***

There are three widely adopted models for missing data [27]: *missing completely at random* (MCAR), *missing at random* (MAR), and *not missing at random* (NMAR). Specifically, MCAR assumes that an attribute A_j is missing randomly (i.e., neither related to other attributes X ($A_j \notin X$) nor A_j itself); MAR assumes that the missing attribute A_j is related to attributes X; and NMAR assumes the missing value on attribute A_j is only related to A_j itself, instead of other attributes X. In this paper, we consider the MAR missing data model.

In the literature of incomplete databases under the MAR model, the imputation techniques of incomplete data can be classified into categories such as rule-based [28], statistical-based [29], pattern-based [30], and constraint-based [31,32]. Due to the sparseness of data sets, these works may encounter the problem of having zero or small size of samples for imputing the missing attributes. This may lead to serious problems such as biased imputed attributes or even the failure of imputing data [6]. In contrast, our work uses a complete data repository R to impute missing attributes via DDs. Moreover, [33,34] studied join and skyline operators over incomplete data streams, which have different query semantics from our Top$k$-iDS problem. Thus, we cannot directly borrow previous techniques, and have to design novel pruning methods, data synopses, and stream processing algorithms specifically for our Top$k$-iDS problem.

3. Problem definition

In this section, we formally define the problem of a *top-k query over incomplete data stream* (Top$k$-iDS), which takes into account missing attribute values during the top-k query processing.

3.1. Incomplete data stream

3.1.1. Incomplete data stream

We first define the data models for incomplete data stream and the *sliding window* over a data stream.

Definition 1. (Incomplete Data Stream) An incomplete data stream, iDS, is an ordered sequence of objects, $\{o_1, o_2, \ldots, o_r, \ldots\}$. Each object o_i arrives at timestamp i, and contains d attributes A_j (for $1 \leqslant j \leqslant d$), some of which have missing attribute values, denoted as $o_i[A_j]$ = "–".

In Definition 1, at each timestamp i, a new object o_i arrives at data stream *iDS*. As an example in Table 1, at timestamp 1, object o_1 arrives at *iDS*; at timestamp 2, a new object o_2 arrives at *iDS*; and so on.

3.1.2. The sliding window model

To process queries over data stream, a *sliding window* model [35] is often used, which always considers the most recent objects from the data stream.

Definition 2. (Sliding Window, W_t) Given an incomplete data stream *iDS*, a positive integer w, and a timestamp t, a sliding window, W_t, contains w most recent objects, $\{o_{t-w+1}, o_{t-w+2}, \ldots, o_t\}$, from *iDS*, where each object $o_i \in iDS$ $(t - w + 1 \leqslant i \leqslant t)$ arrives at timetamp i.

In Definition 2, at timestamp t, the sliding window W_t contains a fixed number (i.e., w) of the latest objects from *iDS*. Following the sliding window model, a sliding window W_t can be incrementally maintained, by adding newly arriving objects and evicting the expired (old) objects.

Example 2. As an example in Fig. 1 and Table 1, assuming window size $w = 3$, the sliding window W_3 contains three objects, o_1, o_2 and o_3, at timestamp 3. At timestamp 4, a new object o_4 arrives at *iDS*, and the oldest one o_1 expires. Thus, object o_4 is added to W_3, and o_1 is removed from W_3, which results in an updated sliding window $W_4 = \{o_2, o_3, o_4\}$. Similarly, we can obtain the sliding window W_5 of size 3 at timestamp 5, that is, $W_5 = \{o_3, o_4, o_5\}$. ■

3.2. Imputation on incomplete data stream

In this paper, we impute the missing attributes by *differential dependency* (DD) rules [6], which can be used to impute a missing attribute value of an incomplete object based on other (complete) attributes of the incomplete object. Assume that we have a *static* data repository R, which contains (historical) complete objects to facilitate the data imputation. Then, we can utilize the data repository R and DD rules (detected from R) to estimate the missing attributes of incomplete objects from data stream *iDS*.

3.2.1. Differential dependency (DD)

We give the definition of a DD rule as follows.

Definition 3. (Differential Dependency [6], *DD*) A *differential dependency* (DD) rule is in the form of $(X \rightarrow A_j, \phi[XA_j])$, where X is a set of *determinant attributes*, A_j is a *dependent attribute* $(A_j \notin X)$, and $\phi[Y]$ is a differential function that specifies distance range restrictions on an attribute set Y (i.e., a set of distance restriction intervals, $A_y.I = [0, \epsilon_{A_y}]$, of attributes A_y in Y).

Intuitively, given a DD rule $(X \rightarrow A_j, \phi[XA_j])$ in Definition 3, if any two objects s_i and s_k satisfy the distance restrictions, $\phi[X]$, on determinant attributes X, that is, $|s_i[A_x] - s_k[A_x]| \in [0, \epsilon_{A_x}]$ $(= A_x.I)$ holds for each attribute $A_x \in X$, then their attribute A_j also follows the distance restriction $\phi[A_j]$, that is, $|s_i[A_j] - s_k[A_j]| \in [0, \epsilon_{A_j}]$ (i.e., $A_j.I$).

As an example, Table 2 has a DD rule, $DD_1 : (A \rightarrow B, \{[0, 0.1], [0, 0.1]\})$. That is, for any two objects such as s_1 and s_2, if their attributes A are within 0.1 distance (i.e., $|s_1[A] - s_2[A]| = 0.1 \leqslant \epsilon_A$ holds), then their B attribute values must be within 0.1 distance (i.e., $|s_1[B] - s_2[B]| = 0.1 \leqslant \epsilon_B$).

The Advantages of Employing DDs as the Imputation Tool. The advantages of DDs as our imputation method are threefold. First, compared with the state-of-the-art constraint-based approach [31], DDs do not require any labelled data, and they can be automatically learned from a static data repository R. Moreover, compared with the methods requiring exact matching (e.g., edit rule [28]), DDs can tolerate differential differences between attribute values, which can lead to a good imputation accuracy, even in sparse data sets. Furthermore, compared with the imputation methods (e.g., SCREEN [32]) based on incomplete data themselves only, DDs have less chance to fail the imputation by leveraging a static repository R. For example, [32] cannot deal with the case that two consecutive records are incomplete at the same time. In contrast, our DD-based imputation does not have the limitations above.

3.2.2. Missing data imputation

With a DD rule in the form of $X \rightarrow A_j$, given an incomplete object o_i with missing attribute A_j and a complete object s_k, if objects o_i and s_k satisfy the distance constraints on attributes X, attribute value $s_k[A_j]$ of complete object s_k can be regarded as a candidate value to fill missing attribute $o_i[A_j]$ of incomplete object o_i. Note that, in this paper, we do not consider the distance constraint, $A_j.I$, on dependent attribute A_j in the process of data imputation. That is, in the process of imputing $o_i[A_j]$, instead of a value interval $[s_k[A_j] - \epsilon_{A_j}, s_k[A_j] + \epsilon_{A_j}]$, we only consider $s_k[A_j]$ as a candidate value.

After the imputation, an incomplete object o_i can have several possible instances $o_{i,l}$ with different imputed attribute values $o_{i,l}[A_j]$. Without loss of generality, we assume each instance $o_{i,l}$ is supported (imputed) by $o_{i,l}.freq$ objects (or a combination of objects for multiple missing attributes) in R. Then, each instance is associated with a probability $o_{i,l}.p = \frac{o_{i,l}.freq}{\sum_{\forall o_{i,x} \in o_i} o_{i,x}.freq}$.

Table 2
An example of a complete data repository R with 2 DD rules, $DD_1 : (A \rightarrow B, \{[0, 0.1], [0, 0.1]\})$ and $DD_2 : (A \rightarrow C, \{[0, 0.2], [0, 0.2]\})$.

obj.	[A] **temperature**	[B] **density of oxygen**	[C] **sunlight intensity**
s_1	0.2	0.2	0.3
s_2	0.3	0.3	0.2
s_3	0.6	0.5	0.5
s_4	0.6	0.5	0.8
s_5	0.7	0.6	0.8
s_6	0.8	0.7	0.8

Example 3. Table 2 shows a complete data repository, $R = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, with two DD rules, $DD_1 : (A \rightarrow B, \{[0, 0.1], [0, 0.1]\})$ and $DD_2 : (A \rightarrow C, \{[0, 0.2], [0, 0.2]\})$, where A, B, and C represent attributes *temperature*, *density of oxygen*, and *sunlight intensity*, respectively.

Incomplete object o_3 in Table 1 has a missing attribute B (i.e., the *density of oxygen*). We can apply the DD rule, $DD_1 : (A \rightarrow B, \{[0, 0.1], [0, 0.1]\})$, to impute this missing attribute $o_3[B]$ via R in Table 2. Since object o_3 satisfies the distance constraints on attribute A with objects s_1 and s_2 in R (e.g., $|o_3[A] - s_1[A]| = 0.1 \in [0, 0.1]$), we can thus treat attribute values $s_1[B] = 0.2$ and $s_2[B] = 0.3$ as 2 possible values of missing attribute $o_3[B]$, that is, attribute $o_3[B]$ may be either 0.2 or 0.3, each associated with a probability 0.5. ■

Note that, we select DD rules as our imputation tool, since DD rules can tolerate differential differences between attribute values. By using a complete data repository R, the imputation via DDs can provide good imputation results even on sparse data, which cannot be achieved by traditional exact matching techniques such as editing rule [28] and relational dependency network [29].

3.2.3. Imputed data stream

Next, we define the imputed data stream, by inferring the missing attributes in incomplete data stream *iDS*, based on DDs and R.

Definition 4. (Imputed Data Stream, *pDS*) Given an incomplete data stream $iDS = (o_1, o_2, \ldots, o_t, \ldots)$ with some missing attribute(s) $o_i[A_j]$ (="–"), its imputed (complete) data stream, $pDS = (o_1^p, o_2^p, \ldots, o_t^p, \ldots)$, is composed of imputed (complete) objects, o_i^p, obtained from objects o_i, by filling all the missing attribute values. Each imputed object $o_i^p \in pDS$ has a set of mutually exclusive (complete) instances, $o_{i,l}$, with existence probabilities $o_{i,l}.p$, where $\sum_{o_{i,l} \in o_i^p} o_{i,l}.p = 1$ holds.

In Definition 4, the imputed data stream *pDS* is obtained by replacing those incomplete objects o_i in *iDS* with their possible instances/samples $o_{i,l}$ (forming uncertain objects [36] or X-relation objects [37]), each associated with an existence probability $o_{i,l}.p$.

Example 4. Table 3 shows an example of the imputed data stream *pDS*, based on incomplete data stream *iDS* in Table 1. By filling the missing attributes of incomplete objects, o_3 and o_5, in Table 1, the imputed data stream *pDS* is given by $\{o_1^p, o_2^p, o_3^p, o_4^p, o_5^p, \ldots\}$, as depicted in Table 3.

As mentioned in Example 3, object $o_3 \in iDS$ has two possible B (i.e., *the density of oxygen*) values, 0.2 and 0.3, both with confidences 0.5. As a result, the imputed (probabilistic) object $o_3^p \in pDS$ has two instances, $o_{3,1}$ and $o_{3,2}$, with equal existence probabilities 0.5. The case of the imputed object $o_5^p \in pDS$ is similar. ■.

3.2.4. Possible worlds of the imputed data stream

In the literature of probabilistic databases, we usually use the *possible worlds* semantics [38]. Here, for the imputed data stream *pDS*, we also consider possible worlds, $pw(W_t)$, of the sliding window W_t in *pDS*, each of which is a materialized instance of the sliding window W_t that can appear in reality. We have the following formal definition:

Definition 5. (Possible Worlds of the Imputed Data Stream, $pw(W_t)$) Given a sliding window W_t of an imputed data stream *pDS*, a possible world, $pw(W_t)$, is a materialization of the sliding window W_t that can appear in the real-world, where each (incomplete) object $o_i \in W_t$ contributes one (imputed) instance $o_{i,l}$ to $pw(W_t)$.

Each possible world, $pw(W_t)$, has an appearance probability, given by:

$$Pr\{pw(W_t)\} = \prod_{o_{i,l} \in pw(W_t)} o_{i,l}.p. \quad (1)$$

Intuitively, in Definition 5, each possible world $pw(W_t)$ of the sliding window W_t corresponds to a combination of instances from probabilistic objects (i.e., objects with imputed attributes) $o_i \in W_t$, which may appear in the real-world with some probability $Pr\{pw(W_t)\}$ (as given in Eq. (1)).

Table 3
The imputed data stream, *pDS*, in the example of Table 1 ($DD_1 : (A \rightarrow B, \{[0, 0.1], [0, 0.1]\})$ and $DD_2 : (A \rightarrow C, \{[0, 0.2], [0, 0.2]\})$.

obj.	inst.	[A] temperature	[B] density of oxygen	[C] sunlight intensity	ranking score	prob.
o_1^p	$o_{1,1}$	0.1	0.1	0.2	0.4	1
o_2^p	$o_{2,1}$	0.4	0.3	0.5	1.2	1
o_3^p	$o_{3,1}$	0.3	**0.2**	0.2	0.7	0.5
	$o_{3,2}$	0.3	**0.3**	0.2	0.8	0.5
o_4^p	$o_{4,1}$	0.2	0.2	0.3	0.7	1
	$o_{5,1}$	0.6	**0.5**	**0.5**	1.6	0.167
o_5^p	$o_{5,2}$	0.6	**0.5**	**0.8**	1.9	0.5
	$o_{5,3}$	0.6	**0.6**	**0.5**	1.7	0.083
	$o_{5,4}$	0.6	**0.6**	**0.8**	2.0	0.25

Example 5. In Table 3, let $W_5 = \{o_3^p, o_4^p, o_5^p\}$ be a sliding window at timestamp 5 of size $w = 3$ from the imputed data stream *pDS*. Table 4 shows all the 8 possible worlds, $pw_1(W_5) \sim pw_8(W_5)$, of sliding window W_5. For example, possible world $pw_1(W_5)$ contains object instances $o_{3,1}, o_{4,1}$, and $o_{5,1}$, and its appearance probability, $Pr\{pw_1(W_5)\}$, is given by multiplying their existence probabilities, that is, $Pr\{pw_1(W_5)\} = o_{3,1}.p \cdot o_{4,1}.p \cdot o_{5,1}.p = 0.5 \times 1 \times 0.167 = 0.0835$. ■

3.3. Top-k queries over incomplete data stream

In this subsection, we provide the definition of the *top-k query over incomplete data stream* (Top*k*-iDS) below.

3.3.1. Ranking function

In this paper, we consider the ranking function, $\mathcal{F}(\cdot)$, below to rank (instances of) objects over the data stream.

Definition 6. (Ranking Function, $\mathcal{F}(\cdot)$) Given positive weights, $A_j.v$, for attributes A_j and an object s_k, a ranking function, $\mathcal{F}(s_k)$, returns the ranking score of object s_k as follows:

$$\mathcal{F}(s_k) = \sum_{j=1}^{d} \left(s_k[A_j] \cdot A_j.v\right). \tag{2}$$

Intuitively, in Definition 6, the weight $A_j.v$ of attribute A_j (as given in Eq. (2)) indicates the importance of attributes A_j. As a result, a ranking function $\mathcal{F}(\cdot)$ will calculate the ranking score of a complete object s_k, by summing up the attribute values $s_k[A_j]$ weighted by $A_j.v$.

Example 6. In the example of Table 1, the ranking function is given by: $\mathcal{F}(s_k) = s_k[A] + s_k[B] + s_k[C]$. Thus, in Table 3, an object instance $o_{1,1} = (0.1, 0.1, 0.2)$ has the ranking score $\mathcal{F}(o_{1,1}) = 0.1 + 0.1 + 0.2 = 0.4$.

3.3.2. The Topk-iDS problem

We next formalize the Top*k*-iDS query, which follows the *probabilistic threshold top-k (PT-k)* semantics [7].

Definition 7. (Top-*k* Queries Over Incomplete Data Stream, Top*k*-iDS) Given an incomplete data stream *iDS*, a ranking function $\mathcal{F}(\cdot)$, and a probabilistic threshold α, a top-*k* query over incomplete data stream (Top*k*-iDS) continuously monitors top-*k* (imputed) objects $o_i \in W_t$ from *iDS*, such that they have the highest ranks with probabilities, $Pr_{Topk-iDS}(o_i^p)$, greater than threshold α, that is,

$$Pr_{Topk-iDS}(o_i^p) = \sum_{pw(W_t), o_{i,l} \in pw(W_t)} Pr\{pw(W_t)\} \cdot \chi(o_{i,l}, k, pw(W_t)) > \alpha \tag{3}$$

where $o_{i,l}$ is an instance of the imputed object o_i^p that appears in possible world $pw(W_t)$, and function $\chi(o_{i,l}, k, pw(W_t)) = 1$ if $o_{i,l}$ is among k instances with the highest ranking scores in possible world $pw(W_t)$ (otherwise, $\chi(o_{i,l}, k, pw(W_t)) = 0$).

In Definition 7, at timestamp t, Top*k*-iDS will retrieve all top-*k* ranked imputed objects $o_i^p \in W_t$ with probabilities, $Pr_{Topk-iDS}(o_i^p)$, greater than α. Intuitively, to compute the Top*k*-iDS probability $Pr_{Topk-iDS}(o_i^p)$ (as given in Eq. (3)), we can sum up all appearance probabilities, $Pr\{pw(W_t)\}$, of possible worlds $pw(W_t)$, which contain an instance $o_{i,l}$ of the imputed object o_i^p, where $o_{i,l}$ is in the top-*k* result of $pw(W_t)$ (i.e., when $\chi(o_{i,l}, k, pw(W_t)) = 1$ holds).

Table 4
Possible worlds, $pw(W_5)$, of W_5 from the imputed data stream, *pDS*, at timestamp 5 in Table 3 ($w = 3$).

Possible world of W_5	**Content of** $pw(W_5)$	**Probability** $Pr\{pw(W_5)\}$
$pw_1(W_5)$	$\{o_{3,1}, o_{4,1}, o_{5,1}\}$	0.0835
$pw_2(W_5)$	$\{o_{3,1}, o_{4,1}, o_{5,2}\}$	0.25
$pw_3(W_5)$	$\{o_{3,1}, o_{4,1}, o_{5,3}\}$	0.0415
$pw_4(W_5)$	$\{o_{3,1}, o_{4,1}, o_{5,4}\}$	0.125
$pw_5(W_5)$	$\{o_{3,2}, o_{4,1}, o_{5,1}\}$	0.0835
$pw_6(W_5)$	$\{o_{3,2}, o_{4,1}, o_{5,2}\}$	0.25
$pw_7(W_5)$	$\{o_{3,2}, o_{4,1}, o_{5,3}\}$	0.0415
$pw_8(W_5)$	$\{o_{3,2}, o_{4,1}, o_{5,4}\}$	0.125

Example 7. We consider object $o_5^p \in pDS$ in Table 3, where $w = 3, t = 5, k = 2$, and $\alpha = 0.8$. In particular, object o_5 has rank 1 in all 8 possible worlds, as shown in Table 4. Thus, we can sum up appearance probabilities of these possible worlds, and obtain the probability $Pr_{Top2-iDS}(o_5^p) = 1$ (as given in Eq. (3)), which is greater than α (i.e., 0.8). Object o_5 is thus one of our Top2-iDS query answers. ■

3.3.3. Challenges

The major challenges to tackle the Top*k*-iDS problem are threefold. First, many existing works [4,5,7] on stream processing usually assume that the data are complete or data with the missing attributes are simply ignored. However, this assumption may not hold in practice (e.g., sensory data may be missing during transmission) or cause inaccurate/wrong query results. Thus, existing top-*k* query processing techniques cannot be directly applied to our Top*k*-iDS problem over incomplete data stream, and we should design effective and efficient approach to impute the missing attributes of incomplete objects.

Second, it is quite challenging to effectively tackle our Top*k*-iDS problem under *possible worlds* semantics [38] over the imputed data stream. As shown in Eq. (3), due to the exponential number of possible worlds, the time complexity is high with a large window size *w* and a large number of missing attributes of incomplete objects. Thus, it is time consuming to process our Top*k*-iDS problem in the stream environment, and we should design effective approaches to reduce the search space of our Top*k*-iDS problem.

Third, it is non-trivial how to effectively and efficiently perform the Top*k*-iDS query over incomplete data stream. This is because, we need to perform the data imputation, and meanwhile dynamically update and maintain the probabilistic top-*k* objects over sliding windows as time changes. Therefore, in this paper, we need to propose effective indexing methods to facilitate efficient Top*k*-iDS query processing algorithms.

Algorithm 1. Top*k*-iDS Processing Framework

Input: an incomplete data stream iDS, a static (complete) data repository R, a timestamp t, an integer k, and a probabilistic threshold α
Output: a Topk-iDS query answer set over W_t
// Pre-Computation Phase
offline construct indexes, I_j, over data repository R
// Imputation and Topk-iDS Query Processing Phase
for *each expired object* o_i' *at timestamp* t **do**
  update *top-k dual layers*, TDL, over W_t with o_i' and evict o_i' from W_t
for *each new object* o_i *arriving at* W_t **do**
  traverse index, I_j, over R and *top-k dual layers*, TDL, over W_t at the same time to enable DD attribute imputation and top-k computation, resp.
  if *object* o_i^p *cannot be pruned* **then**
    incrementally update the *top-k dual layers*, TDL, with object o_i^p
return all objects on the first layer of *top-k dual layers*, TDL, as actual Top$k$-iDS answers

3.4. Topk-iDS processing framework

Algorithm 1 illustrates a framework for our Top*k*-iDS query answering, which consists of two phases. In the first *pre-computation phase* (Section 6.1), we offline build indexes $\mathcal{I}_j$ (for $1 \leqslant j \leqslant d$) over a static (complete) data repository R for imputing attributes A_j (line 1). Next, in the *imputation and Topk-iDS query processing phase* (Sections 6.2 and 6.3), we dynamically maintain a data synopsis, called *top-k dual layers* (*TDL*), which consists of two layers of potential Top*k*-iDS candidates over incomplete data stream *iDS*. For expired objects $o_{i'}$, we delete $o_{i'}$ from sliding window W_t and update *TDL* (lines 2–3). Similarly, regarding the insertion of a new object o_i, we will use indexes $\mathcal{I}_j$ over R to help with the data imputation (as will be discussed in Section 4) via *differential dependencies* (DDs), and apply pruning mechanisms (discussed later in Section 5) to reduce the search space of the Top*k*-iDS problem (lines 5–6). If object o_i cannot be pruned, then we will insert it into *TDL*. Finally, we directly return actual Top*k*-iDS answers from the top-*k* dual layer data structure *TDL* (line 8).

Table 5 depicts the commonly-used symbols and their descriptions in this paper.

4. Cost-model-based imputation of incomplete objects

In this section, we will explore how to impute the missing attributes of incomplete objects in data streams via *differential dependencies* (DDs) [6] discovered from a historical complete data repository R. Instead of imputing the missing attribute via a single DD (Section 3.2), in the sequel, we will consider the attribute imputation via multiple DDs with the same dependent attribute A_j (for $1 \leqslant j \leqslant d$). In particular, in order to select good CDDs for attribute imputation, we introduce a *conceptual imputation lattice* (as a guideline) and an effective cost model via the fractal dimension [39].

4.1. Missing attribute imputation via DDs

In some scenario, there may be more than one DD rule, $X_1 \rightarrow A_j, X_2 \rightarrow A_j$, ..., and $X_l \rightarrow A_j$, with the same dependent attribute A_j. In this case, without prejudice, we may need to consider combining all these DDs, that is, $(X_1X_2 \ldots X_l \rightarrow A_j, \phi[X_1X_2 \ldots X_lA_j])$, for imputing missing attributes A_j.

The advantages of combining these DDs are as follows. Intuitively, the intersection of determinant attribute sets from different DD rules makes the query range $X_1.I \wedge X_2.I \wedge \ldots \wedge X_l.I$ more selective. For example, given a schema with three attributes $\{A, B, C\}$, and two DD rules, $DD_1 : A \rightarrow C, \{[0, 0.1], [0, 0.1]\}$ and $DD_2 : B \rightarrow C, \{[0, 0.2], [0, 0.1]\}$, in order to impute attribute C, we can use a combined DD rule $AB \rightarrow C$ with tighter distance constraints, that is, $A.I \wedge B.I = [0, 0.1; 0, 0.2]$. This will lead to a smaller set of object candidates for filling the missing attribute A_j of an incomplete object, higher confidences for fewer imputed instances of incomplete object, and lower time cost for the data imputation.

4.2. DD selection strategy

As mentioned above, when multiple (l) DDs with the same dependent attribute A_j are discovered from the historical repository R, we can combine them to obtain one DD rule $X_1X_2 \ldots X_l \rightarrow A_j$, and use the combined rule to impute the missing attribute A_j of an incomplete object o_i. However, there is a prerequisite for the usage of the combined rule, that is, there must exist at least one complete object in R satisfying the distance constraints (w.r.t. object o_i) on attributes $X_1X_2 \ldots X_l$.

If no complete objects can be obtained from R for imputing $o_i[A_j]$, we can alternatively select a subset of attributes in $X_1X_2 \ldots X_l$ for imputing $o_i[A_j]$. However, there are an exponential number of attribute subsets, and we need to design an effective strategy to choose an appropriate DD rule.

In the sequel, weintroduce a *conceptual imputation lattice* to assist the DD selection.

Conceptual imputation lattice (Lat_j). Given l different DD rules, $X_1, X_2, \ldots, X_l$, with the same dependent attribute A_j, we consider a *conceptual imputation lattice*, Lat_j, where $1 \leqslant j \leqslant d$. Each lattice Lat_j is composed of $(l + 1)$ levels. To be specific, on level 0 of the lattice, we have an empty set, indicating that none of the DD rules can be used to impute missing attribute A_j; on level 1, we have l different DD rules $X_s \rightarrow A_j$ (for $1 \leqslant s \leqslant l$); on level 2, we have DD rules in the form $X_aX_b \rightarrow A_j$ (for $1 \leqslant a, b \leqslant l$ and $a \neq b$); and so on. Finally, on level l, we only have one DD rule, that is, $X_1X_2 \ldots X_l \rightarrow A_j$.

DD selection via Lat_j. Our DD selection strategy is to traverse lattice Lat_j to find the best DD rule from level l to level 0. When we reach a DD rule DD_s on a level lv of Lat_j, we will first estimate whether or not there are any objects s_r in data repository R that satisfy the distance constraints w.r.t. o_i. If the estimated number of objects s_r is nonzero, we will use this DD rule DD_s for imputing $o_i[A_j]$; otherwise, we continue to check the next DD rule on level lv. When the traversal of the lattice reaches level 0 (i.e., no DD rule can be used to impute $o_i[A_j]$), we will apply a statistics-based method [29] to impute $o_i[A_j]$ with the probabilistic distribution of attribute A_j in R.

Since it is not efficient to directly count the number of objects in R satisfying distance constraints, in this paper, we also provide a cost model (please refer to Appendix B.1 for details) to estimate this COUNT aggregate within a query range (i.e., following distance constraints) via *fractal dimension* [39]. If the expected COUNT of a DD rule is greater than or equal to 1, we will use this DD rule for imputing $o_i[A_j]$.

Table 5
Symbols and descriptions.

Symbol	Description
iDS	An incomplete data stream
pDS	An imputed (probabilistic) data stream
o_i	An object arriving at timestamp i from stream iDS
o_i^p	An imputed probabilistic object in the imputed stream pDS
w	The size of the sliding window
W_t	The most w recent objects from stream iDS or pDS at timestamp t
$pw(W_t)$	A possible world of imputed probabilistic objects in W_t
$\mathcal{F}(\cdot)$	A ranking function given by Eq. (2)

5. Pruning strategies

5.1. Problem reduction

In Definition 7, for an incomplete object o_i, we calculate its top-k probability by taking into account all possible worlds $pw(W_t)$ of the sliding window $W_t \in pDS$ at timestamp t, which is inefficient, or even impossible, to enumerate (as given in Eq. (3)). Intuitively, for o_i, its top-k probability only depends on whether there are more than k other objects o_z in W_t that have higher ranking scores than o_i (i.e., $\mathcal{F}(o_z) > \mathcal{F}(o_i)$). Therefore, we can reduce the problem of calculating the top-k probability, $Pr_{Topk-iDS}(o_i^p)$, of the imputed object o_i^p from the possible-world level to the object level. Specifically, in the stream environment (instead of static uncertain databases [7,40–43]), we rewrite the top-k probability $Pr_{Topk-iDS}(o_i^p)$ of o_i^p by summing up the probabilities that each instance, $o_{i,l} \in o_i^p$, is a top-k object instance among all objects in the sliding window W_t, that is,

$$Pr_{Topk-iDS}(o_i^p) = \sum_{\forall o_{i,l} \in o_i^p} \left(o_{i,l}.p \cdot \sum_{j=1}^{k} H(W_t, j-1) \right), \tag{4}$$

where $o_{i,l}.p$ is the existence probability that instance $o_{i,l}$ of o_i^p appears in the real-world, and function $H(W_t, j-1)$ is the probability that $(j-1)$ objects in W_t have higher ranking scores than that of instance $o_{i,l}$.

5.2. An equivalent form of $Pr_{Topk-iDS}(o_i^p)$

From Eq. (4), we can see that the top-k probability of instance $o_{i,l}$ (i.e., $\sum_{j=1}^{k} H(W_t, j-1)$) is only related to those instances with higher ranking scores than $o_{i,l}$ (rather than the entire sliding window W_t).

For objects in W_t, we can obtain a *sorted instance list*, denoted as L_{ins}, by sorting their instances in a non-increasing order of their ranking scores.Similar to [7], for each object o_z with some instances ranked before instance $o_{i,l}$ in the list L_{ins}, we combine these instances of o_z into a condensed instance o_z^c $(= \{o_{z,s} | o_{z,s} \in o_z^p, \mathcal{F}(o_{z,s}) > \mathcal{F}(o_{i,l})\})$, associated with probability $o_z^c.p$ $(= \sum_{\forall o_{z,s} \in o_z^c} o_{z,s}.p)$ and an interval of ranking scores (which is enclosed by the corresponding instance scores). This way, these condensed instance o_z^c (w.r.t. instance $o_{i,l}$) can form a so-called *dominant condensed instance set*, $S_{dom}(o_{i,l})$, where $S_{dom}(o_{i,l}) = \{o_z^c | o_z \in W_t, \forall o_{z,s} \in o_z^c, \mathcal{F}(o_{z,s}) > \mathcal{F}(o_{i,l})\}$.

With the notion of $S_{dom}(o_{i,l})$, we can rewrite Eq. (4) as:

$$Pr_{Topk-iDS}(o_i^p) = \sum_{\forall o_{i,l} \in o_i^p} \left(o_{i,l}.p \cdot \sum_{j=1}^{k} H(S_{dom}(o_{i,l}), j-1) \right), \tag{5}$$

where $H(S_{dom}(o_{i,l}), j-1)$ is the probability that exactly $(j-1)$ compressed objects from $S_{dom}(o_{i,l})$ appear.

Note that, although we reduce the top-k probability calculation from the possible-world level (i.e., Eq. (3)) to the object level (i.e., Eq. (5)), it is still time-consuming to compute this probability involving many probabilistic objects. To further reduce the time complexity, we will introduce several pruning strategies, which discard false alarms with low top-k confidences (i.e., $\leqslant \alpha$) and reduce the problem search space.

5.3. Pruning strategies

In this subsection, we propose three pruning strategies, *top-instance pruning*, *mid-instance pruning*, and *multi-instance pruning*, respectively, which can help discard false alarms with low top-k confidences (i.e., $\leqslant \alpha$).

5.3.1. Terminologies

Before we introduce the pruning strategies, we first present several terms and their meanings below.

- **The top instance** o_i^+: an instance with the highest ranking score among all instances of object $o_i^p \in W_t$.

- **The bottom instance** o_i^-: an instance with the lowest ranking score among all instances of object $o_i^p \in W_t$.
- **The middle instance** $o_{i,m}$: given all instances, $o_{i,l}$, of object o_i^p in a non-increasing order of their ranking scores, the middle instance $o_{i,m}$ satisfies the conditions that (1) $\sum_{l=1}^{m} o_{i,l}.p \leqslant \alpha$, and (2) $\sum_{l=1}^{m+1} o_{i,l}.p > \alpha$.

5.3.2. Top-instance pruning

The *top-instance pruning* method filters out false alarms via *top instances* of objects. Specifically, if the top instance o_i^+ of an imputed object o_i^p cannot have top-k highest ranking scores (in possible worlds) during its lifetime, then object o_i^p can never be in the Top$k$-iDS query answer set over the stream.

Below, we formally give this top-instance pruning lemma.

Lemma 5.1. (Top-Instance Pruning) *Given a top instance o_i^+ of imputed object $o_i^p \in W_t$ and its dominant condensed instance set $S_{dom}(o_i^+)$, object o_i^p can be safely pruned, if there exist k condensed instances $o_z^c \in S_{dom}(o_i^+)$ such that: (1) $\prod_{\forall o_z^c} o_z^c.p > 1-\alpha$, and (2) $\forall z, t_z > t_i$, where t_z and t_i are arrival times of objects o_z and o_i, respectively.*

Proof. Please refer to Appendix A.1.

As illustrated in Fig. 2, for imputed object o_i^p, k condensed instances o_z^c ($1 \leqslant z \leqslant k$) arrive after object o_i^p and have ranking scores higher than its top instance o_i^+. Moreover, the multiplication of existence probabilities from k condensed instances is greater than $1-\alpha$ (i.e., $\prod_{z=1}^{k} o_z^c.p > 1-\alpha$). Due to the existence of these k condensed instances (objects), top instance o_i^+ (or object o_i^p) cannot be in the top-k result with high confidence.

5.3.3. Mid-instance pruning

Similarly, our second *mid-instance pruning* method will check whether or not a middle instance $o_{i,m}$ of object o_i^p can become a top-k object in its lifetime. If the answer is no, then object o_i^p can be pruned. Below, we provide this pruning method.

Lemma 5.2. (Mid-Instance Pruning) *Given a middle instance $o_{i,m}$ of imputed object o_i^p, and its dominant condensed instance set $S_{dom}(o_{i,m})$, object o_i^p can be safely pruned, if there are at least k condensed instances $o_z^c \in S_{dom}(o_{i,m})$, such that for these objects o_z, we have: (1) $\mathcal{F}(o_z^-) \geqslant \mathcal{F}(o_{i,m})$, and (2) $t_z > t_i$.*

Proof. Please refer to Appendix A.2.

5.3.4. Multi-instance pruning

We will try the third *multi-instance pruning* strategy, in the case that an object o_i^p cannot be pruned by neither the top-instance nor mid-instance pruning methods. That is, for an imputed object o_i^p, we will derive an upper bound, $UB\_Pr_{Topk-iDS}(o_i^p)$, of its top-k probability $Pr_{Topk-iDS}(o_i^p)$ (given in Eq. (5)). If this upper bound is smaller than α, then we can safely prune object o_i^p.

In order to compute the upper bound $UB\_Pr_{Topk-iDS}(o_i^p)$, we alternatively consider summing up probability upper bounds of instances $o_{i,l}$ in object o_i^p. We have the following pruning lemma via multiple instances.

Lemma 5.3. (Multi-Instance Pruning) *Given L instances, $o_{i,l}$ ($1 \leqslant l \leqslant L$), of an imputed object o_i^p in a non-ascending order of their ranking scores, and dominant condensed instance sets $S_{dom}(o_{i,l})$, object o_i^p can be safely pruned, if for some $1 \leqslant v \leqslant L$, it holds that: (1) for all $o_z^c \in S_{dom}(o_{i,l}), t_z > t_i$, and (2) $UB\_Pr_{Topk-iDS}(o_i^p) < \alpha$. Here, we have the probability upper bound:*

$$UB\_Pr_{Topk-iDS}(o_i^p) = \sum_{a=1}^{v} o_{i,a}.p \cdot UB_1(o_{i,a}) + \sum_{b=v+1}^{L} o_{i,b}.p \cdot UB_2(o_{i,b}) \tag{6}$$

where $UB_1(o_{i,a}) = \sum_{j=1}^{k} \left(\binom{N}{j-1} \cdot \prod_{x=1}^{j-1} o_x^c.p \cdot \prod_{y=j}^{N} (1 - o_y^c.p) \right)$ (for $\forall o_x^c, o_y^c \in S_{dom}(o_{i,a}), o_x^c.p > o_y^c.p$), N is the number of condensed instances in $S_{dom}(o_{i,a})$, $\binom{n}{m}$ is the number of combinations to select m out of n objects, and $UB_2(o_{i,b})$ is the smallest $UB_1(o_{i,a})$ among all instances $o_{i,a}$ ($1 \leqslant a \leqslant v$).

Proof. Please refer to Appendix A.3.

Intuitively, in Lemma 5.3, we only need to access partial instances $o_{i,a}$ ($1 \leqslant a \leqslant v$), and compute the probability upper bound $UB\_Pr_{Topk-iDS}(o_i^p)$ based on Eq. (6), which can be used to enable the multi-instance pruning.

We apply the three pruning rules above in the order of top-instance pruning, middle-instance pruning, and multi-instance pruning, where the latter ones (e.g., multi-instance pruning) have higher pruning power than the former one(s),

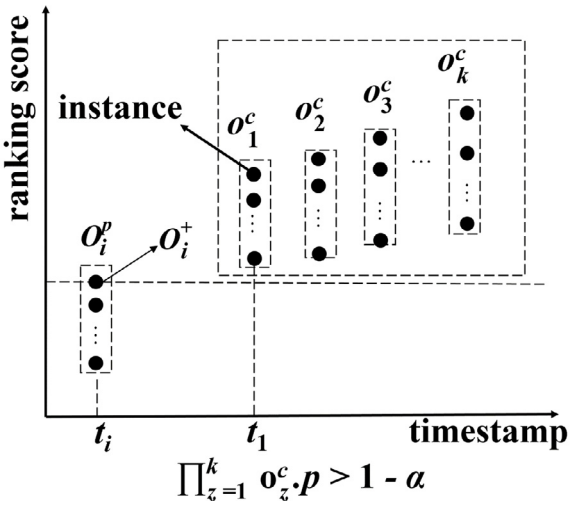

Fig. 2. Illustration of the top-instance pruning.

but need more computation cost. In other words, we first use the top-instance pruning rule with less computation cost, then use the middle-instance pruning (if top-instance pruning rule fails to prune), and finally apply the multi-instance pruning with the highest pruning power (in case the previous two pruning methods fail to prune). Therefore, these three pruning methods are applied together in order in our approach to provide high pruning power and meanwhile efficient computation cost. The three pruning strategies above are applied in Section 6.3 (Algorithm 2), and evaluated in Section 7.3.

6. Top-*k* processing over incomplete data streams

In the sequel, Section 6.1 discusses how to build the index structure, I_j over the data repository R, which is to facilitate and accelerate missing attribute imputation. Then, Section 6.2 proposes a novel data synopsis, namely Top-k dual layers (*TDL*), built over sliding window W_t of incomplete data stream, which is to dynamically maintain candidate top-k objects over W_t. Finally, Section 6.3 presents how to leverage the index structure I_j and data synopsis *TDL* to perform data imputation and top-k query answering at the same time over incomplete data stream.

6.1. Cost-model-based indexes on data repository R for imputation

In this subsection, we will present our cost-model-based indexes, I_j ($1 \leqslant j \leqslant d$), over a complete repository R, each of which can facilitate quick imputation of a missing attribute A_j in incomplete objects from *iDS*.

6.1.1. Index structure

To enable fast imputation of missing attributes in objects $o_i \in W_t$, we will derive d indexes, I_j (for $1 \leqslant j \leqslant d$), each of which can quickly and accurately access complete objects s_r in repository R, and impute the values of missing attribute A_j.

Given l DD rules, $\{X_1 \rightarrow A_j, X_2 \rightarrow A_j, \ldots, X_l \rightarrow A_j\}$, with dependent attribute A_j, we will build an index I_j over R for attributes in $U_j = X_1 \cup X_2 \cup \ldots \cup X_l$. In particular, we divide (complete) objects $s \in R$ into n clusters, $cls_1 \sim cls_n$, of sizes within $[m, M]$. Then, we invoke the normal "insertion" function, and insert clusters c_k (for $1 \leqslant k \leqslant n$) into an R$*$-tree [44].

Fig. 3 shows an example of constructing such an index w.r.t. a DD rule (i.e., $DD_1 : A \rightarrow B$), where $U_j = \{A\}$ and $A_j = B$. In this toy example, we have 3 clusters, $c_1 \sim c_3$, which are stored in the leaf nodes of an R$*$-tree.

Moreover, each node e in the index I_j is associated with a histogram, H_{U_j}, over attributes U_j, which stores a summary of complete objects s in e. To construct this histogram H_{U_j}, we first divide the data space of node e into λ^d buckets, *buc*, of equal size, where λ is the number of intervals for each attribute in U_j and d is the number of attributes (i.e., $|U_j|$). Next, for each bucket *buc*, we store the following information for the pruning (discussed in Section 5.3):

1. the number, $buc.cnt$, of complete objects s_r in bucket *buc*, and
2. the bound, $buc.I = [buc.A_j^-, buc.A_j^+]$, of attribute A_j for all objects $s \in buc$.

In the previous example of Fig. 3, histogram H_A of MBR node e contains 2 buckets, buc_1 and buc_2, over attribute A. For instance, bucket buc_1 contains a count aggregate $buc_1.cnt$ and an interval $buc_1.I$ for attribute B.

Please refer to Appendix B for the details of our cost model for selecting good clusters in *cls*, as well as the data imputation and object pruning via the index I_j.

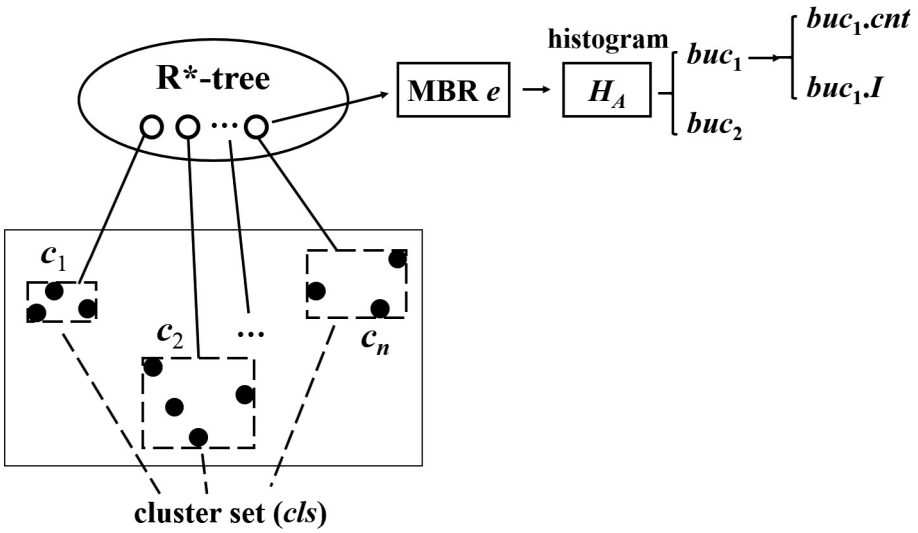

Fig. 3. Index over repository R with histogram w.r.t $DD_1 : A \rightarrow B$ in Table 3.

6.2. Top-k dual-layer synopsis for the sliding window

Next, we propose a data synopsis, namely *top-k dual layers* (*TDL*), for the sliding window W_t of incomplete data stream *iDS*, which stores potential Top*k*-iDS query answers.

6.2.1. Top-k dual layers (TDL)

The *TDL* data synopsis is built over a sliding window W_t of the imputed data stream *pDS*, which contains two layers. In particular, the first layer contains the current top-*k* query answer set (i.e., satisfying $Pr_{Topk-iDS}(o_i^p) > \alpha$ for all objects o_i^p on this layer); the second layer keeps a set of objects, o_i^p, that are not currently in the top-*k* answer set, but might become top-*k* answers in future before they expire (i.e., $0 < Pr_{Topk-iDS}(o_i^p) \leqslant \alpha$ holds at the current timestamp).

As shown in Fig. 4, each valid object o_i^p in synopsis *TDL* is associated with two data structures: a *dominant condensed instance set* $S_{dom}(o_i^+)$ of its top instance o_i^+ (as defined in Section 5.1; which can be used by Lemma 5.1 *vector* containing 8 items listed below:

1. the layer number (1 or 2), $o_i^p.layer$, on which object o_i^p stays on *TDL*;
2. a pointer, $o_i^-.pos$, pointing to the bottom instance o_i^- of object o_i^p in the *sorted instance list* L_{ins};
3. a pointer, $o_{i,m}.pos$, pointing to the middle instance $o_{i,m}$ of o_i^p in the *sorted instance list* L_{ins};
4. a pointer, $o_i^+.pos$, pointing to the top instance o_i^+ of o_i^p in the *sorted instance list* L_{ins};
5. *(layer 2 only)* a pointer, $o_{z,s}.pos$, pointing to an instance $o_{z,s}$ in *sorted instance list* L_{ins}, where $o_{z,s}$ is the first detected instance in L_{ins} (starting from the beginning) such that the instance set prior to $o_{z,s}$ makes $Pr_{Topk-iDS}(o_i^p) \leqslant \alpha$ hold (via Eq. (5));
6. *(layer 2 only)* a timestamp, t_s, indicating the timestamp to re-check the top-*k* probability of o_i^p;
7. the earliest expiration timestamp, t_k, among all condensed objects stored in the *dominant condensed instance set* $S_{dom}(o_i^+)$; and
8. the number, N_m, of condensed instances o_z^c in the dominant condensed instance set $S_{dom}(o_{i,m})$, where the bottom instances o_z^- of objects o_z^p have higher ranking score than that of middle instance $o_{i,m}$ (i.e., $\mathcal{F}(o_z^-) > \mathcal{F}(o_{i,m})$) and expire after object o_i (i.e., $t_z > t_i$).

Fig. 4 shows an example of an object o_i^p in *TDL*, which has two extra structures, $S_{dom}(o_i^+)$ and the *vector*. The *vector* stores information such as the layer number for Item (1) (i.e., $o_i^p.layer = 2$ means object o_i^p is on layer 2 of *TDL*), the pointers (i.e., four pointers in Items (2)–(5), pointing to instances in L_{ins}), timestamps (Items (6) and (7)), and a counter (Item (8)).

All the stored information above will help accelerate the Top*k*-iDS query processing, which will later be discussed in Section 6.3.

Properties of *TDL*. We list the properties of the *TDL* synopsis below.

Property 1. (Completeness) The *TDL* synopsis contains all the objects o_i^p from *iDS* that have chance to be top-*k* objects before they expire.

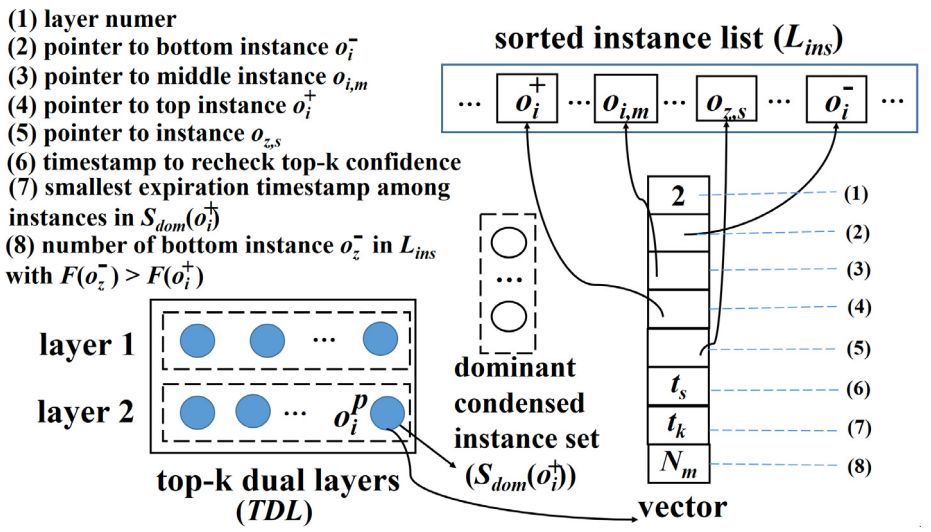

Fig. 4. Illustration of the top-k dual layers.

Proof. Please refer to Appendix C.1.

Property 2. (No False Dismissals) If an imputed object o_i^p is not on the first layer of the top-k dual layers *TDL* over W_t, then o_i^p cannot be a top-k object at current timestamp t.

Proof. Please refer to Appendix C.2.

6.3. Dynamic maintenance of the Top-k dual layers

As discussed in Section 6.2, *TDL* maintains the current top-k objects (on layer 1) and potential top-k objects in the future (on layer 2). In the sequel, we will explore how to dynamically maintain the *TDL*, upon the arrival of new objects and expiration of old objects.

Insertions. When a new object o_i arrives at timestamp t, we need to decide how to update the *TDL* synopsis with object o_i. In particular, Algorithm 2 shows the pseudo code of the insertion, which first inserts object o_i into a layer in *TDL*, and then updates other objects o_g^p in *TDL*.*Finding the layer to insert new object* o_i^p. At timestamp t, when a new (incomplete) object o_i arrives, we will utilize index I_j and DDs to obtain the imputed probabilistic object o_i^p (with attribute intervals on the node level; line 1), and update the *sorted instance list* L_{ins} by inserting instances (or that with attribute intervals) in o_i^p (line 2). Then, we will check if o_i^p can be inserted on first layer of the *TDL* via the *dominant condensed instance set* $S_{dom}(o_i^-)$ of its bottom instance o_i^- (lines 3–6). If there are less than k condensed instances o_z^c having higher ranking scores than that of bottom instance o_i^- (lines 3–4), or o_i^- is a top-k object via Eq. (5) (lines 5–6), object o_i is definitely a current top-k answer, and we will put it on layer 1 of *TDL*. If we cannot determine which layer o_i should stay via its bottom instance o_i^-, we will calculate the top-k probability of multiple instances of object o_i^p via Eq. (5) (lines 7–10). If o_i^p satisfies the constraint of the top-k probability (i.e., $\geqslant \alpha$), we insert o_i^p into layer 1 of *TDL* (lines 7–8); otherwise, o_i^p is not currently a top-k answer, and we put it on layer 2 of *TDL* (lines 9–10).

After we add o_i^p to *TDL*, we can compute $S_{dom}(o_i^+)$ and its vector (as mentioned in Fig. 4) for imputed object o_i^p, which can assist top-k probability calculation later (line 11).*Updating objects* o_g^p *in TDL.* For each object o_g^p in *TDL*, we first check whether or not there are any instances of o_i^p having higher ranking scores than that of its top instance o_g^+ (line 13). If the answer is yes, we may need to update $S_{dom}(o_g^+)$ and the earliest expiration timestamp t_k, for condensed objects in $S_{dom}(o_g^+)$ (lines 14–15). After that, we will prune o_g^p via Lemmas 5.1 $\sim$ 5.3 (lines 16–23). Specifically, we first check if object o_g^p can be pruned by k condensed instances o_z^c (top-instance pruning as given in Lemma 5.1; lines 16–17). If o_g^p cannot be pruned by Lemma 5.1, we will see if it can be pruned by mid-instance pruning method via Lemma 5.2 (lines 18–21). If o_g^p still cannot be pruned, we will use the multi-instance pruning in Lemma 5.3 (lines 22–23).

If o_g^p cannot be pruned, we will update data structures associated with o_g^p in *TDL* (lines 24–33). Specifically, if o_g^p is on the first layer of *TDL* and some of its instances have lower ranking scores than that of object o_i^p, we will re-check its top-k probability (lines 25–26). If $Pr_{Topk-iDS}(o_g^p) < \alpha$, we will move o_g^p from layer 1 of *TDL* to layer 2, and update its pointer $o_g^p.pos$ and timestamp t_s (lines 27–29). If o_g^p is not on layer 1, we will check whether or not the current timestamp t reaches the recheck-

ing timestamp t_s (line 30). If $t \geqslant t_s$, we will update the two data structures of o_g^p (line 31). After the update, if current top-k probability of o_g^p is greater than α, we move o_g^p from layer 2 to layer 1 on *TDL* (lines 32–33).

If object o_g^p is pruned, we will update data structures of each object o_h^p in *TDL* (when some instances of o_g^p affect the probability of instances in o_h^p (lines 34–36)

Algorithm 2. Insertion

Input: the top-k dual layers TDL and a new object o_i at timestamp t
Output: the updated TDL

1 impute and obtain o_i^p via DDs and index I_j
2 update sorted instance list L_{ins}
3 **if** *the number of condensed objects* $o_z^c \in S_{dom}(o_i^-)$ *is less than* k **then**
4 put o_i on the first layer of TDL
5 **else if** $Pr_{Topk-iDS}(o_i^-) > \alpha$ *via Eq. (5)* **then**
6 put o_i on the first layer of TDL
7 **else if** $Pr_{Topk-iDS}(o_i^p) > \alpha$ *via Eq. (5)* **then**
8 put o_i on the first layer of TDL
9 **else**
10 put o_i on the second layer of TDL // a potential answer in the future
11 compute $S_{dom}(o_i^+)$ and a vector of 8 items for object o_i^p
12 **for** *each object* o_g^p *in* TDL **do**
13 **if** $\mathcal{F}(o_i^+) > \mathcal{F}(o_g^+)$ **then**
14 update the dominant condensed instance set $S_{dom}(o_g^+)$ of top instance o_g^+ of o_g^p
15 update the timestamp t_k of o_g^p
16 **if** $\exists$ k *condensed instances* $o_z^c \in S_{dom}(o_g^+)$, *such that* $\prod o_z^c.p > 1-\alpha$ *and* $t_k > t_g$ **then**
17 delete object o_g^p from TDL *// Lemma 5.1*
18 **else if** $\mathcal{F}(o_i^-) > \mathcal{F}(o_{g,m})$ *and* $t_i > t_g$ **then**
19 $N_m \leftarrow N_m + 1$
20 **if** $N_m = k$ **then**
21 delete object o_g^p from TDL *// Lemma 5.2*
22 **if** o_g^p *is not pruned and* $Pr_{Topk-iDS}(o_g^p) \leq \alpha$ *via the dominant condensed instant set* $S_{dom}(o_{g,l})$ *of instances* $o_{g,l} \in o_g^p$ **then**
23 delete object o_g^p from TDL *// Lemma 5.3*
24 **else if** o_g^p *is not pruned* **then**
25 **if** o_g^p *is on the first layer* **then**
26 **if** $\mathcal{F}(o_i^+) > \mathcal{F}(o_g^-)$ *and* $Pr_{Topk-iDS}(o_g^p) \leq \alpha$ *via Eq. (5)* **then**
27 move o_g^p from layer 1 to layer 2 on TDL
28 update pointer $o_{z,s}.pos$ for object o_g^p
29 update timestamp t_s of o_g^p
30 **else if** $t \geq t_s$ **then**
31 update data structure stored in o_g^p
32 **if** $Pr_{Topk-iDS}(o_g^p) > \alpha$ **then**
33 move o_g^p from layer 2 to layer 1 on TDL
34 **else**
35 **for** *each object* o_h^g *satisfying* $\mathcal{F}(o_g^+) > \mathcal{F}(o_h^-)$ **do**
36 update data structures of object o_h^p

Deletions. When an old object o_i^p expires, Algorithm 3 dynamically updates the *TDL* synopsis. Specifically, Algorithm 3 first removes the expired object o_i^p from *TDL* (line 2). Then, we will consider those objects o_z^p whose top-k probabilities are affected by the expiration of o_i^p (i.e., objects o_z^p with instances having lower scores than that of object o_i^p; line 3). For these affected objects o_z^p, we update their structure information (line 4). Moreover, if they are layer 1 of *TDL*, their top-k probabilities will not decrease, and we can keep them on the first layer. On the other hand, those objects o_z^p on layer 2 of *TDL* will have increasing top-k probabilities $Pr_{Topk-iDS}(o_i^p)$, which thus should be re-computed (lines 5–6). If their top-k probabilities are greater than threshold α, we move them from layer 2 to layer 1 in *TDL*, and update the pointer $o_{zs}.pos$ and timetamp t_s with *null* values (note: only objects on layer 2 of *TDL* are associated with these two variables; lines 7–10).

Top$k$-iDS Query Answer Retrieval. Since layer 1 of the *TDL* synopsis contains all objects o_i^p with top-k probabilities $Pr_{Topk-iDS}(o_i^p) > \alpha$, we can directly return all the objects on layer 1 of *TDL* as our actual Top$k$-iDS query answers.

Complexity Analysis. The insertion function in Algorithm 2 requires $O(L \cdot |S_{dom}(o_{i,l})|)$ time complexity, where L is the average number of instances per imputed object o_i^p, and $|S_{dom}(o_{i,l})|$ is the average number of condensed instances in $S_{dom}(o_{i,l})$ for some instances $o_{i,l}$. Moreover, the deletion function in Algorithm 3 has $O(N_{exp} \cdot N_{obj} \cdot N_{ins})$ time complexity, where N_{exp} is the number of expired objects o_e, N_{obj} is the number of affected objects o_i^p on layer 2 of *TDL* due to the expired objects o_e (i.e., satisfying $\mathcal{F}(o_e^+) > \mathcal{F}(o_i^+)$ and $o_i^p.layer = 2$), and N_{ins} is the number of the affected instances, $o_{i,l}$, in the affected objects, o_i^p (i.e., satisfying $\mathcal{F}(o_{i,l}) < \mathcal{F}(o_{e,s})$).

Algorithm 3. Deletion

Input: the top-k dual layers TDL and current timestamp t
Output: the updated TDL

```
1  for each expired object o_i^p ∈ W_t do
2      remove o_i^p from TDL
3      for each object o_z^p ∈ W_{t-1} satisfying F(o_i^+) > F(o_z^-) do
4          update the data structure of o_z^p
5          if o_z^p is on layer 2 of TDL then
6              re-compute Pr_{Topk-iDS}(o_i^p) via Eq. (5)
7              if Pr_{Topk-iDS}(o_z^p) > α then
8                  add o_z to layer 1 of TDL
9                  update pointer o_{z,s}.pos with null
10                 update the re-checking timestamp t_s with null
```

7. Experimental evaluation

7.1. Experimental settings

7.1.1. Real/synthetic data sets

In this paper, we evaluate the performance of our Top$k$-iDS approach on both real and synthetic data sets.

7.1.2. Real data sets

We use Intel lab data,[1] Pump sensor data for predictive maintenance,[2] and UCI gas sensor data for home activity monitoring,[3] denoted as *Intel*, *Pump*, and *Gas*, respectively. *Intel* data contain 2.3 million records from 54 sensors deployed in the Intel Berkeley Research lab; *Pump* data hold 220 K sensory data from 52 sensors on Apr. 1–Aug. 31, 2018; *Gas* data consist of 919,438 samples from 8 MOX gas sensors, and a temperature and humidity sensor. We extract 4 attributes (i.e., temperature, humidity, light, and voltage) from *Intel* data, 10 attributes (i.e., sensor_01-sensor_10) from *Pump* data, and 10 attributes (i.e., temperature, humidity, and resistance of sensors 1–8) from *Gas* data. Then, as shown in Table 6, we detect DD rules among attributes [6], by checking complete objects s_r in data repository R and all possible combinations between any two attributes (i.e., determinant and dependent) in the data schema. Note that, all DD rules $X \rightarrow A_j$ in Table 6 are the ones with minimum intervals on dependent attribute A_j. For details of the DD detection, please refer to [6].

[1] http://db.csail.mit.edu/labdata/labdata.html.
[2] https://www.kaggle.com/nphantawee/pump-sensor-data/version/1.
[3] http://archive.ics.uci.edu/ml/datasets/gas+sensors+for+home+activity+monitoring.

Table 6
The tested data sets and their DD rules.

Data Sets	DD Rules
Intel	$voltage \rightarrow temperature, \{[0, 0.0001], [0, 0]\}$
	$voltage \rightarrow humidity, \{[0, 0.0001], [0, 0]\}$
	$Voltage \rightarrow light, \{[0, 0.0001], [0, 0]\}$
	$Light \rightarrow voltage, \{[0, 0.0001], [0, 0.989]\}$
Pump	$Sensor\_06 \rightarrow sensor\_01, \{[0, 0.0001], [0, 0]\}$
	$Sensor\_06 \rightarrow sensor\_02, \{[0, 0.0001], [0, 0]\}$
	$sensor\_06 \rightarrow sensor\_03, \{[0, 0.0001], [0, 0]\}$
	$Sensor\_06 \rightarrow sensor\_04, \{[0, 0.0001], [0, 0]\}$
	$Sensor\_08 \rightarrow sensor\_05, \{[0, 0.0001], [0, 0]\}$
	$Sensor\_07 \rightarrow sensor\_06, \{[0, 0.0001], [0, 0.0206]\}$
	$Sensor\_01 \rightarrow sensor\_07, \{[0, 0.0001], [0, 0.073]\}$
	$sensor\_07 \rightarrow sensor\_08, \{[0, 0.0001], [0, 0.06]\}$
	$sensor\_01 \rightarrow sensor\_09, \{[0, 0.0001], [0, 0.065]\}$
	$sensor\_08 \rightarrow sensor\_10, \{[0, 0.0001], [0, 0]\}$
Gas	$Resistance4 \rightarrow resistance1, \{[0, 0.0001], [0, 0.177]\}$
	$Resistance3 \rightarrow resistance2, \{[0, 0.0001], [0, 0.2615]\}$
	$Resistance2 \rightarrow resistance3, \{[0, 0.0001], [0, 0.279]\}$
	$Resistance5 \rightarrow resistance4, \{[0, 0.0001], [0, 0.239]\}$
	$Resistance4 \rightarrow resistance5, \{[0, 0.0001], [0, 0.2]\}$
	$Resistance1 \rightarrow resistance6, \{[0, 0.0001], [0, 0.038]\}$
	$resistance3 \rightarrow resistance7, \{[0, 0.0001], [0, 0.1]\}$
	$Temperature \rightarrow resistance8, \{[0, 0.0001], [0, 0.006]\}$
	$Resistance8 \rightarrow temperature, \{[0, 0.0001], [0, 0.007]\}$
	$Resistance8 \rightarrow humidity, \{[0, 0.0001], [0, 0.043]\}$
Uniform *Correlated* *Anti-correlated*	$B \rightarrow A, \{[0, 0.0001], [0, 0.001]\}$
	$C \rightarrow B, \{[0, 0.0001], [0, 0.001]\}$
	$D \rightarrow C, \{[0, 0.0001], [0, 0.001]\}$
	$E \rightarrow D, \{[0, 0.0001], [0, 0.001]\}$
	$F \rightarrow E, \{[0, 0.0001], [0, 0.001]\}$
	$G \rightarrow F, \{[0, 0.0001], [0, 0.001]\}$
	$H \rightarrow G, \{[0, 0.0001], [0, 0.001]\}$
	$I \rightarrow H, \{[0, 0.0001], [0, 0.001]\}$
	$J \rightarrow I, \{[0, 0.0001], [0, 0.001]\}$
	$A \rightarrow J, \{[0, 0.0001], [0, 0.001]\}$

7.1.3. Synthetic data sets

We produce three types of d-dimensional data sets: *Uniform*, *Correlated*, and *Anti-correlated*, which follow uniform, correlated, and anti-correlated distributions [45], respectively. In particular, we first generate 5000 seeds following the corresponding data distribution, and then randomly obtain the remaining data objects, based on seeds and DD rules (as depicted in Table 6).

7.1.4. Incomplete data generation

Given the number, m, of missing attributes, for real and synthetic data above, we randomly set m out of d attributes as missing (i.e., "-"), and turn complete objects in the stream into incomplete ones. Note that, instead of directly using incomplete data in the real world, we manually turn complete data into incomplete ones. This way, we can obtain the ground truth of Top$k$-iDS query answers, which can be used to evaluate the effectiveness of our proposed solutions to the Top$k$-iDS problem.

7.1.5. Competitor

We compare our Top$k$-iDS approach with five competitors, namely $DD + Topk$, $DD + PTk$, $Mean + Topk$, $Zero + Topk$, and $Con + Topk$. The details of above five baseline methods as follows.

- $DD + Topk$: this baseline method first imputes missing attribute values via *differential dependencies* (DDs) [6], and then performs the top-k query over imputed data streams via the proposed techniques in this paper;
- $DD + PTk$: this baseline method first imputes missing attribute values via DDs [6], and then conducts the top-k query operator over imputed data streams via the algorithm in [7];
- $Mean + Topk$: this baseline first imputes missing attribute values by taking the average of all the detected values via DDs [6], and then uses the proposed techniques in this paper.
- $Zero + Topk$: this method first imputes missing attribute values with 0 (the same as [5,4]), and then performs the top-k query operator via the proposed techniques in this paper.
- $Con + Topk$: this baseline method first imputes missing attribute values via a *constraint-based* imputation method [46], and then performs the top-k query processing over imputed data streams via the proposed methods in this paper.

Table 7
The parameter settings.

Parameters	Values
Probabilistic threshold α	0.1, 0.2, **0.5**, 0.8, 0.9
Dimensionality d	2, 3, **4**, 5, 6, 10
parameter k	5, 8, **10**, 20, 50, 100
The number, W_t, of valid objects in *iDS*	1.5 K, 3 K, **6 K**, 12 K, 15 K, 30 K
The size, R|, of data repository R	10 K, 20 K, **30 K**, 40 K, 50 K
The number, m, of missing attributes	**1**, 2, 3

7.1.6. Measures

We will report the *wall clock time* for both real and synthetic data sets, which is the overall CPU time for the data imputation via $\mathcal{I}_j$ (as described in Section 6.1) and and top-k query answering via top-k dual layers *TDL* (as discussed in Section 6.3). Please note that, we do not separately report the imputation and query times, since we perform the data imputation and query processing at the same time.

7.1.7. Parameter settings

We run our experiments on a PC with Intel(R) Core(TM) i7-6600U CPU 2.70 GHz and 32 GB memory. Table 7 depicts parameter settings of our experiments, and we bold all parameter defaults. When we test one parameter, we will keep all other parameters as their default values. All algorithms were implemented by C++, where the code is available at http://www.cs.kent.edu/ wren/topk.

7.2. Verification of the cost model

We first verify our cost model for estimating the number, cnt_Q, of objects o_i that fall into the query range Q $(= \bigwedge_{A_x \in X}[o_i[A_x] - \epsilon_{A_x}, o_i[A_x] + \epsilon_{A_x}])$ based on DD rules with determinant attributes X. Note that, this cost model is important

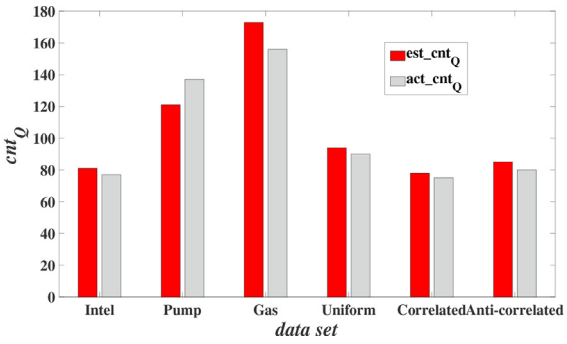

Fig. 5. Cost model verification for the DD selection.

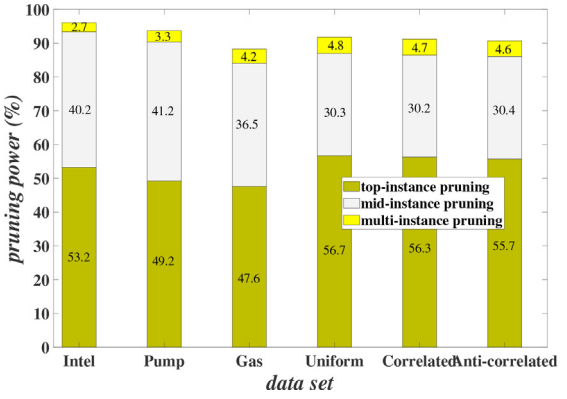

Fig. 6. Pruning power evaluation over real/synthetic data sets.

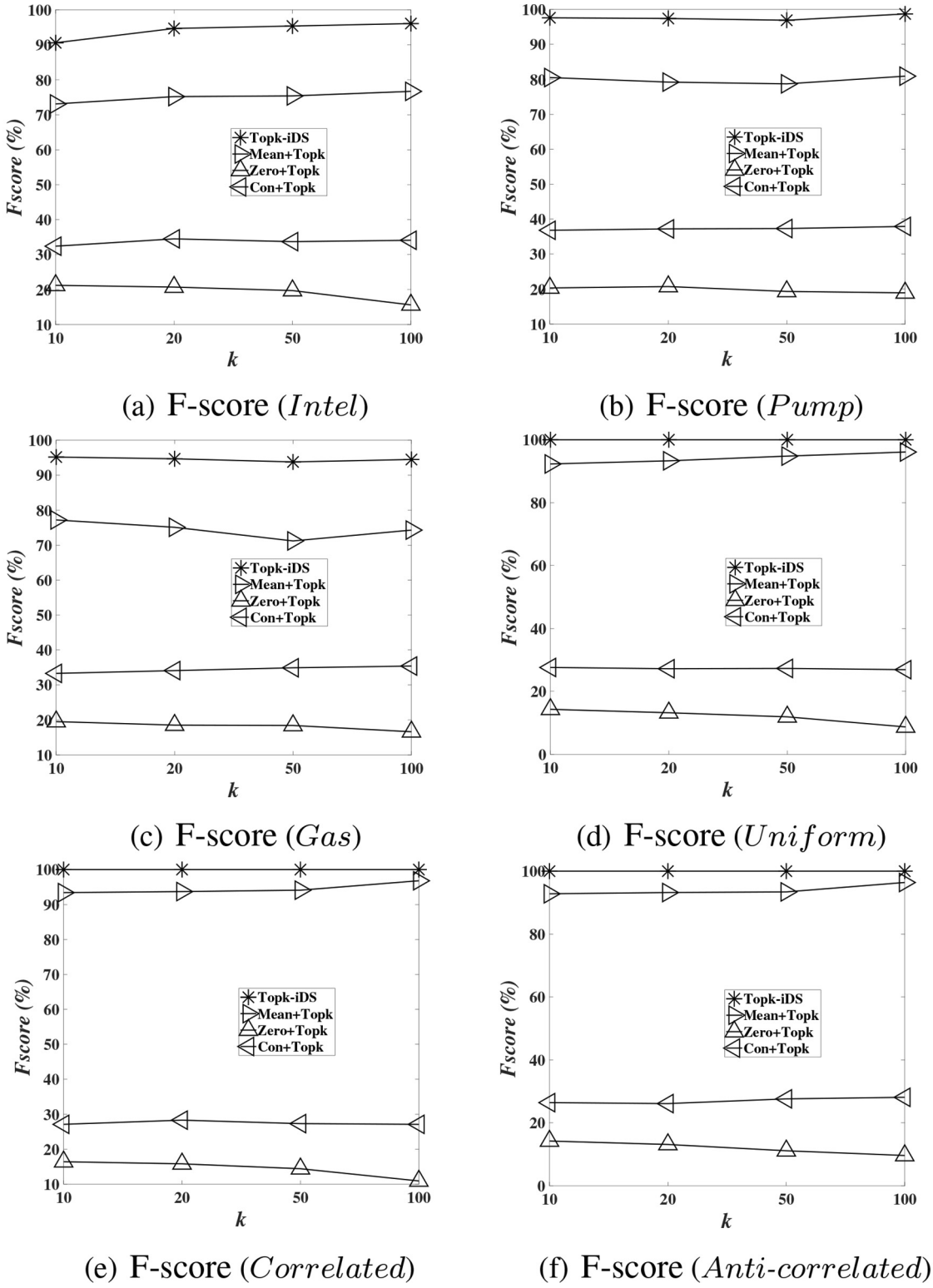

Fig. 7. The Top*k*-iDS effectiveness vs. parameter *k*.

for selecting a good (combined) DD rule for the data imputation, when we access the *conceptual imputation lattice*, Lat_j. As shown in Fig. 5, we compare the estimated cnt_Q (via Eq. (9) in Appendix B.1), denoted as est_cnt_Q, with the actual one,

act_cnt_Q, over both real and synthetic data sets. Our experimental results show that the estimated count cnt_Q can closely mimic the actual one, which confirms the correctness and effectiveness of our proposed cost model. This also indicates that we can use the cost model to guide the selection of a good DD rule (balancing between accuracy and efficiency) in the lattice Lat_j.

7.3. Evaluation of Topk-iDS pruning strategies

Fig. 6 shows the pruning power of our proposed pruning rules (in Section 5.3) over real/synthetic data sets, where all parameters are set to their default values (Table 7). As stated in Section 6.3 (Algorithm2), we apply the pruning rules in the order of *top-instance pruning*, *mid-instance pruning*, and *multi-instance pruning*. From Fig. 6, we can see that the *top-instance pruning* strategy can rule out almost half of the objects (i.e., 47.6–53.2% for real data and 55.7–56.7% for synthetic data). Then, the *mid-instance pruning* can further prune the remaining unpruned objects (i.e., 36.5–41.2% and 30.2–30.4% for real and synthetic data, respectively), followed by the *multi-instance pruning* (i.e., 2.7–4.2% for real data and 4.6–4.8% for synthetic data). Overall, the three pruning strategies can together prune 88.3–96.1% for real data and 90.7–91.8% for synthetic data, which confirms the effectiveness of our three pruning rules.

7.4. The effectiveness of Topk-iDS queries

In this subsection, we report the effectiveness of our proposed Top*k*-iDS approach, in terms of *F-score*. Specifically, in our experiments, we first obtain complete real/synthetic data sets, and then randomly mark some attribute(s) as missing attributes. Thus, we know the groundtruth of actual top-*k* query answers over complete data sets, and the *F-score* is given by:

$$F-score = 2 \times \frac{recall \times precision}{recall + precision} \tag{7}$$

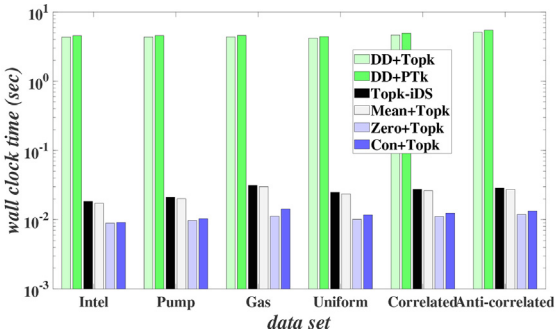

Fig. 8. The performance vs. real/synthetic data sets.

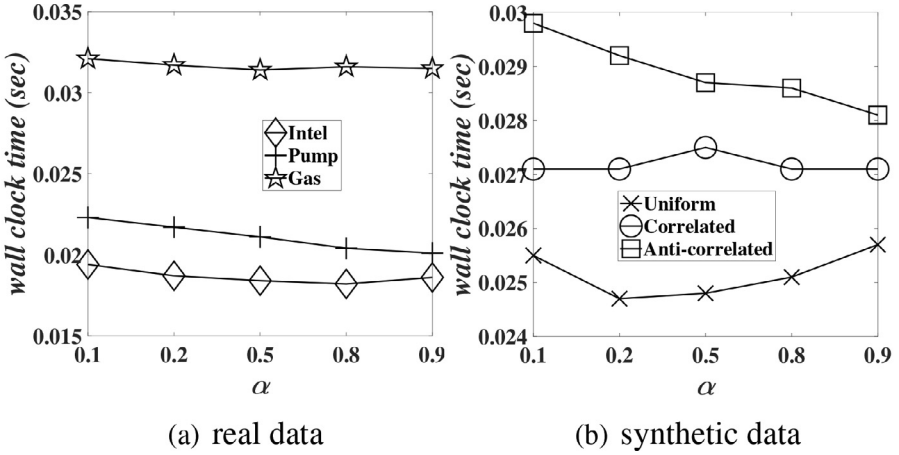

Fig. 9. The performance vs. probabilistic threshold α.

where the *recall* is given by the number of actual top-k answers in our Top$k$-iDS query results divided by the number of actual top-k answers (i.e., k); and the *precision* is defined as the number of actual top-k answers in our Top$k$-iDS query results divided by the total number of objects returned by our Top$k$-iDS approach.

7.4.1. ***The Top$k$-iDS effectiveness vs. parameter*** k

Fig. 7 compares the query accuracy (i.e., *F-score*) of our Top$k$-iDS approach w.r.t. that of three baseline competitors (i.e., $Mean + Topk, Zero + Topk$ and $Con + Topk$) over real/synthetic data sets, where $k = 10, 20, 50$, and 100, and other parameters are set to their default values. Note that, since $DD + Topk$ and $DD + PTk$ use the same imputation method as the Top$k$-iDS method (i.e., using DDs as the imputation tool and considering multiple instances for incomplete tuples), they have the same *F-score* as Top$k$-iDS, and we do not plot them in figures. For real data sets (i.e., *Intel*, *Pump* and *Gas*), Fig. 7 shows that our Top$k$-iDS approach outperforms the three baseline methods and achieves high *F-score* with different k values (i.e., above 90% for *Intel*, 96% for *Pump*, and 93% for *Gas*), which confirms the effectiveness of considering multiple object instances for incomplete objects. Moreover, $Zero + Topk$ achieves the lowest *F-score* over all real data sets, which shows that it is inappropriate to ignore the missing attribute values when we deal with the Top$k$-iDS problem.

For synthetic data sets (i.e., *Uniform*, *Correlated*, and *Anti-correlated*), from Fig. 7, we observe the same trend as that of real data. Specifically, the Top$k$-iDS approach achieves higher *F-score* (very close to 100%) over synthetic data than that over real data sets. This is because synthetic data is densely generated based on 5,000 seeds (i.e., pivots) and DD rules, thus, the recall ratio for the imputation (in turn, the *F-score*) on synthetic data is higher than that of real data.

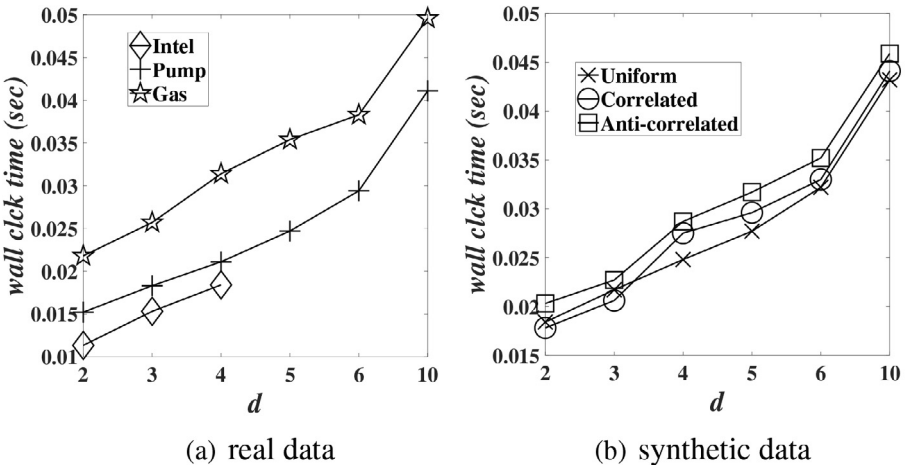

Fig. 10. The performance vs. dimensionality d.

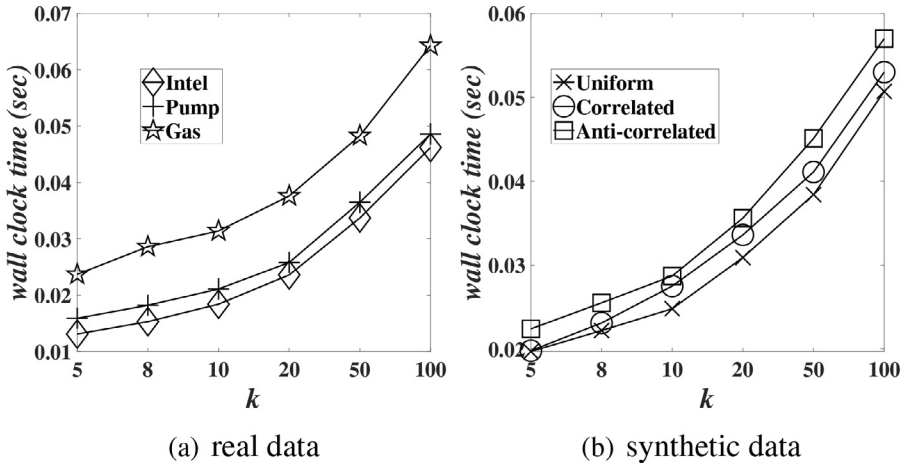

Fig. 11. The performance vs. the parameter k.

7.5. The efficiency of Topk-iDS queries

7.5.1. The Top$k$-iDS performance vs. real/synthetic data sets

Fig. 8 compares the performance of our Top$k$-iDS algorithm with that of the five baseline methods (i.e., $DD + Topk, DD + PTk, Mean + Topk, Zero + Topk$, and $Con + Topk$) over real/synthetic data, where default parameter values are used. From experimental results, Top$k$-iDS outperforms $DD + Topk$ and $DD + PTk$ algorithms by about 2 orders of magnitude, in terms of the wall clock time, which confirms the efficiency of our Top$k$-iDS approach (i.e., the style of "data imputation and top-k processing at the same time"). Compared with the baselines (i.e., $Zero + Topk, Con + Topk$, and $Mean + Topk$) with only one imputed value for each missing attribute, our Top$k$-iDS approach has multiple possible imputed values, and thus incurs higher wall clock time. Nevertheless, our Top$k$-iDS approach has higher query accuracy (as confirmed in Section 7.4).

In particular, the performance of our Top$k$-iDS approach on *Intel* data is always better than that of *Pump* and *Gas* data. The reason is that there are fewer possible imputed values in *Intel* w.r.t. that of *Pump* and *Gas* data. Similar trends can be found in subsequent experimental results.

Below, we will test the robustness of our Top$k$-iDS approach by varying different parameters over real/synthetic data sets. In order to clearly plot the trend of our Top$k$-iDS approach, we will not report the results of the five baseline methods here.

7.5.2. The Top$k$-iDS performance vs. probabilistic threshold α

Fig. 9 evaluates the effect of probabilistic threshold α on the performance of our Top$k$-iDS approach, where α varies from 0.1 to 0.9, and other parameters are set to default values. Note that, smaller α may cause more potential top-k objects, and weaken the pruning power (as mentioned in Section 5.3), whereas larger α may lead to higher calculation cost of the top-k probability (as given in Eq. 5) for more instances. From figures, we can see that the wall clock time is not very sensitive to the α value, and remains low in both real and synthetic data (less than 0.0325 and 0.03 s, respectively), which indicates the efficiency of our Top$k$-iDS approach for different α values.

7.5.3. The Topk-iDS performance vs. dimensionality d

Fig. 10 demonstrates the efficiency of our Top$k$-iDS approach over real and synthetic data, by varying the number, d, of attributes in *iDS* (and R) from 2 to 10, where other parameters are by default. Note that, in Fig. 10(a), *Intel* data only have 4 attributes. With the increase of the d value, the wall clock time also increases. This is due to the "the curse of dimensionality" problem [47]. Nonetheless, the overall cost for all real/synthetic data is still quite low (i.e., less than 0.05 s and 0.046 s, resp.), which shows good performance of our Top$k$-iDS approach.

7.5.4. The Topk-iDS performance vs. parameter k

Fig. 11 shows the effect of parameter k for the Top$k$-iDS query processing, where $k = 5, 8, 10, 20, 50$, and 100, and other parameters are set to their default values. In figures, the wall clock time increases when k becomes larger. This is reasonable, since we need to retrieve more objects from data stream when k is larger. However, the overall time cost remains low (i.e., less than 0.065 s for $k = 100$), which shows good scalability of our Top$k$-iDS approach for large k.

We also conducted experiments by varying other data distributions or parameters (e.g., the number, $|W_t|$, of valid objects in W_t, the size, $|R|$, of the data repository R, and the number, m, of missing attributes), please refer to Appendix D for details. In summary, extensive experiments have verified the effectiveness and efficiency of our Top$k$-iDS approach.

8. Conclusions

In this paper, we formulate and tackle the Top$k$-iDS query over incomplete data stream, which monitors top-k objects over the sliding window, with missing attributes. In order to efficiently and effectively process the Top$k$-iDS query, we design effective imputation, pruning, and data synopsis mechanisms to facilitate an efficient Top$k$-iDS query answering algorithm. Our Top$k$-iDS framework follows the style of "imputation and top-k query processing at the same time". Extensive experiments have demonstrated the efficiency and effectiveness of our proposed Top$k$-iDS processing approach over real/synthetic data under various parameter settings.

There are several research directions as our future works. First, we use the count-based sliding window [48] (as given in Definition 2) for our problem, and we can consider the problem study with the time-based sliding window [16]. Second, we do not consider the distance constraint $A_j.I$ on dependent attribute A_j in the process of missing data imputation (Section 3.2), and thus we would like to study the imputation strategy by considering the distance interval (i.e., $A_j.I$) on dependent attribute A_j in the future. Third, we adopt DD [6] as our imputation method, and leave the interesting topic of using other imputation methods (e.g., relational dependency network [29]) as our future works. Fourth, we calculate the ranking function (Definition 6) by summing up the attribute values weighted by the corresponding weights, and we will explore other ranking functions as our future work. Fifth, we follow the *probabilistic threshold top-k (PT-k)* semantics [7] in our framework, and it is interesting to study the topics that extend our framework to support other top-k semantics (e.g., Pk-top$k$ [9], U-Top$k$ [25], and U-kRanks [25]).

CRediT authorship contribution statement

Weilong Ren: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing - original draft, Writing - review & editing. **Xiang Lian:** Conceptualization, Methodology, Validation, Formal analysis, Writing - original draft, Writing - review & editing, Funding acquisition. **Kambiz Ghazinour:** Conceptualization, Methodology, Validation, Formal analysis, Writing - review & editing, Funding acquisition.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Proofs of Lemmas for Pruning Strategies

A.1. Proof of Lemma 5.1

Proof. In all the possible worlds, $pw_z(W_t)$, where all k dominant condensed instances $o_z^c \in S_{dom}(o_i^+)$ appear, we can get $Pr_{Topk-iDS}(o_i^+) = 0$, since all these k instances o_z^c have higher ranking scores than that of top instance o_i^+ in possible worlds $pw_z(W_t)$. The top instance o_i^+ has the highest ranking score among all instances of o_i^p, we can get $Pr_{Topk-iDS}(o_i^p) \leqslant Pr_{Topk-iDS}(o_i^+)$. Thus, $Pr_{Topk-iDS}(o_i^p) = 0$ in all possible worlds $pw_z(W_t)$ where the k dominant condensed instances o_z^c appear, and object o_i^p may be top-k objects only in other possible worlds where not all the k dominant condensed instances o_z^c appear at the same time. We know that the existence probability sum of all possible worlds is equal to 1 (i.e., $\sum pw(W_t) = 1$). If $\prod_{\forall o_z^c} o_z^c.p > 1 - \alpha$, the probability sum of all other possible worlds is smaller than α (i.e., $\sum pw(W_t) - pw_z(W_t) \leqslant \alpha$). Even object o_i^p is always a top-k object in all these possible world (except possible worlds $pw_z(W_t)$), its top-k probability is smaller than α. Besides, if all these k condensed instances o_z^c expire later than o_i^p (i.e., $t_z > t_i$), object o_i^p can never be a top-k object in its lifetime due to the existence of the k condensed instances o_z^c in $S_{dom}(o_i^+)$, and can be safely pruned.

A.2. Proof of Lemma 5.2

Proof. For the middle instance $o_{i,m}$ of imputed object o_i^p, if there are k condensed instances, o_z^c, in its *dominant condensed instance set* $S_{dom}(o_{i,m})$, such that (1) their bottom instances o_z^- all have higher ranking scores than that of instance $o_{i,m}$ (i.e., $\mathcal{F}(o_z^-) > \mathcal{F}(o_{i,m})$), and (2) objects o_z expire after o_i ($t_z > t_i$), the middle instance $o_{i,m}$ cannot become a top-k object in its lifetime. That is, $Pr_{Topk-iDS}(o_{i,m}) = 0$. Thus all instances of o_i^p with lower ranking scores than that of $o_{i,m}$ cannot become top-k instances (i.e., $\forall o_{i,l}$ ($l \geqslant m$), $Pr_{Topk-iDS}(o_{i,l}) = 0$). We can obtain a top-$k$ probability upper bound of object o_i^p by assuming all its instances prior to $o_{i,m}$ are always top-k object in their possible worlds, i.e., $Pr_{Topk-iDS}(o_i^p) \leqslant \sum_{l=1}^{m-1} o_{i,l}.p \cdot Pr_{Topk-iDS}(o_{i,l}) \leqslant \sum_{l=1}^{m-1} o_{i,l}.p$. Since $\sum_{l=1}^{m} \leqslant \alpha$, we can get $Pr_{Topk-iDS}(o_i^p) \leqslant \sum_{l=1}^{m-1} o_{i,l}.p \leqslant \sum_{l=1}^{m} o_{i,l}.p \leqslant \alpha$. So imputed object o_i^p cannot become a top-k object in its lifetime and can be safely pruned.

A.3. Proof of Lemma 5.3

Proof. Based on Eq. (5), for an instance $o_{i,l}$ of imputed object o_i^p, when it is ranked as rank-j (for $1 \leqslant j \leqslant k$), its corresponding top-k probability equals to $H(S_{dom}(o_{i,l}), j-1)$. That is, only $(j-1)$ out of N $(= |S_{dom}(o_{i,l})|)$ condensed instances o_z^c (for $1 \leqslant z \leqslant N$) in its dominant condensed instance set $S_{dom}(o_{i,l})$ appear, which has $\binom{N}{j-1}$ possible combinations. We sort the condensed instances o_z^c in $S_{dom}(o_{i,l})$ based on their existence probabilities (i.e., $o_z^c.p$) in a decreasing order, and use o_x^c (for $1 \leqslant x \leqslant j-1$) and o_y^c (for $j \leqslant y \leqslant N$) to represent the first $(j-1)$ and the remaining condensed instances in the sorted dominant condensed instance set $S_{dom}(o_{i,l})$. Thus, in the $\binom{N}{j-1}$ possible worlds that only $(j-1)$ objects in $S_{dom}(o_{i,l})$ appear, the most probable one (i.e., with the biggest existence probability) is that only the first $j-1$ condensed instances o_x^c in $S_{dom}(o_{i,l})$ appear, with existence probability $\prod_{x=1}^{j-1} o_x^c.p \cdot \prod_{y=j}^{N}(1 - o_y^c.p)$. By using this most probable probability as the existence probabilities of all $\binom{N}{j-1}$ possible worlds, we can get an upper bound of instance $o_{i,l}$ to be ranked as rank-j, that is,

$H(S_{dom}(o_{i,l}),j-1) \leqslant \binom{N}{j-1} \cdot \prod_{x=1}^{j-1} o_x^c.p \cdot \prod_{y=j}^{N}(1-o_y^c.p)$. Thus, we can derive an upper bound of instance $o_{i,l}$ to be a top-k object by combing the upper bounds of $o_{i,l}$ to be ranked as rank-1 to rank-k, i.e., $Pr_{Topk-iDS}(o_{i,l}) \leqslant \sum_{j=1}^{k} H(S_{dom}(o_{i,l}),j-1) \leqslant UB_1(o_{i,l})$, where $UB_1(o_{i,l}) = \sum_{j=1}^{k}(N \quad j-1) \cdot \prod_{x=1}^{j-1} o_x^c.p \cdot \prod_{y=j}^{N}(1-o_y^c.p)$. Furthermore, we can derive an upper bound of imputed object o_i^p to be a top-k object by combing all its instance upper bounds, i.e., $Pr_{Topk-iDS}(o_i^p) = \sum_{\forall o_{i,l}} o_{i,l}.p \cdot Pr_{Topk-iDS}(o_{i,l}) \leqslant \sum_{\forall o_{i,l}} o_{i,l}.p \cdot UB_1(o_{i,l})$. The above upper bound of imputed object o_i^p needs to check and obtain the upper bound of all its instances $o_{i,l}$ (for $1 \leqslant l \leqslant L$). Actually, we may not need to check all its instances. Instead, we only need to check partial of its instances. The idea is as follow. We sort all instances $o_{i,l}$ of o_i^p based on their ranking scores in a decreasing order, and thus the upper bound of instance $o_{i,l}$ is also an upper bound of instance $o_{i,l+1}$. Given the sorted instances $o_{i,l}$, we will check the first v instances, and obtain each of their probability upper bound $UB_1(o_{i,a})$ (for $1 \leqslant a \leqslant v$). From these v upper bounds $UB_1(o_{i,a})$, we choose the smallest one, denoted as $UB_2(o_{i,b})$, as the upper bounds of all unchecked instances $o_{i,b}$ (for $v+1 \leqslant b \leqslant L$). By combing all these upper bounds, we then can get the probability upper bound, $UB\_Pr_{Topk-iDS}(o_i^p)$, of o_i^p, as shown in Eq. (6). If $UB\_Pr_{Topk-iDS}(o_i^p) \leqslant \alpha$, and all condensed instances o_z^c in $S_{dom}(o_{i,l})$ expire later than object o_i^p (i.e., $t_z > t_i$), then object o_i^p can never become a top-k object, and can be safely pruned.

Appendix B. Cost models and apply of index I_j

B.1. Cost-model-based estimation via fractal dimension

Given a DD rule, $Y \rightarrow A_j$, the data space of repository R can be reduced (projected) from d dimensions to $|Y|+1$ dimensions, where $|Y|$ is the number of attributes in Y. Within the $(|Y|+1)$-dimensional subspace, we can calculate its corresponding *fractal dimension* [39], denoted as D_2^Y, given as follows.

$$D_2^Y = \frac{\partial log(\sum p_i^2)}{\partial log(r)}, \qquad r \in (r_1, r_2) \tag{8}$$

where the $(|Y|+1)$-dimensional space of data repository R is divided into multiple regular cells, c_i, with equal side length $r \in (r_1, r_2)$, and p_i is the percentage of data points in R falling into a cell c_i.

Given the DD, $Y \rightarrow A_j$, and an incomplete object o_i with missing attribute A_j, we can obtain a query range, Q, enclosed by the intervals, $A_x.I$, on attributes $A_x \in Y$. With the query range Q, we can estimate the number of points, cnt_Q, falling into Q via Eq. (9) [39].

$$cnt_Q = \left(\frac{Vol(\epsilon, Q)}{Vol(\epsilon, \square)}\right)^{\frac{D_2^Y}{|Y|+1}} \times (N-1) \times 2^{D_2^Y} \times \epsilon^{D_2^Y} \tag{9}$$

where ϵ is the distance of the center of gravity of Q to the most remote point in x-axis within query range Q, $Vol(\epsilon, Q)$ is the volume of Q, $Vol(\epsilon, \square)$ is the volume of regular cube (with dimension $|Y|+1$) with distance ϵ between the most remote point in x-axis and its center of gravity of $\square$, and N is the number of objects falling into the regular cube.

During the lattice traversal, when we encounter a DD rule, $Y \rightarrow A_j$, we can apply Eq. (9) to estimate the expected COUNT aggregate, cnt_Q, in Q. If it holds that $cnt_Q \geqslant 1$, we will use this DD for the imputation of attribute A_j. If no object is actually found by this DD, we will consider the next DD rule with $cnt_Q \geqslant 1$ in lattice Lat_j.

B.2. Cost model for selecting good clusters in cls

Note that, with different clustering methods over R, we may obtain distinct cluster sets *cls*, with various imputation costs. Therefore, it is important to obtain a "good" cluster set *cls* that minimizes the imputation cost. Alternatively, given a query range, Q, over data repository R (for the imputation), we aim to maximize the probability that clusters in *cls* do not intersect with Q (i.e., with low imputation cost).

Specifically, given a DD: $(X \rightarrow A_j, \phi[XA_j])$ and an incomplete object o_i (with missing attribute A_j), we have the query range $Q(o_i, DD)$ defined as follows:

$$Q(o_i, DD) = \bigwedge_{A_x \in X} [o_i[A_x] - \epsilon_{A_x}, o_i[A_x] - \epsilon_{A_x}] \tag{10}$$

For any cluster $c \in cls$, we use $c.MBR$ to represent its space range over attributes X. Our goal is to find the best cluster set cls^* satisfying the following condition:

$$cls^* = \arg\max_{cls} \sum_{i=1}^{s} \sum_{\forall DD} \sum_{\forall c_k \in cls} \psi(c_k, Q(o_i, DD)) \tag{11}$$

where incomplete object o_i are samples (following the distribution of historical data), s is the number of samples, and function $\psi(c_k, Q(o_i, DD))$ returns the probability that cluster $c \in cls$ does not intersect with query range $Q(o_i, DD)$.

Intuitively, the desired cluster set cls^* should maximize the summed probability that clusters $c_k \in cls^*$ do not intersect with the query range $Q(o_i, DD)$, which in turn minimizes the imputation cost.

In light of the cost model in Eq. (11), we will design an effective cost-model-based clustering method, which aims to find optimal clusters cls^* guided by Eq. (11) iteratively.

Based on the cost model in Eq. (11), given a data repository R, DD rules, object dimensionality d, page size *page_size* (e.g., typically 4*KB*), object (or MBR) size *obj_size* (e.g., *obj_size* $= 4$ *bytes* $\times 2 \times d$ for floating numbers), and a user-specified number of rounds *round*, Algorithm 4 obtains the optimal cluster set cls^*. In lines 1–3, we initialize three variables *max*, *min*, and *S*, which define the maximum and minimum numbers of objects that a cluster can store, and the number of generated clusters, respectively. In lines 4–6, we define another three variables, cls^*, *Pivots* and *Prob*, which represent the optimal cluster set cls^*, the set of *S* pivots to generate *S* clusters, and the summed probability that query ranges of all sampling objects o_i in Eq. (11) do not intersect with the data space c_k.*MBR* of clusters $c_k \in cls^*$, respectively. Also, in line 7, variable, *iter*, stores the current iteration for searching the cluster set cls^*. Here, *iter* is limited by the user-defined upper bound *round*. In the first iteration, we will generate the cluster set based on the *S* pivots in *Pivots*, by assigning objects s_r to its nearest pivot under the limitation that the size of each cluster is between *min* and *max* (lines 9–10). In the following iterations (i.e., *iter* > 1), we will first generate new pivots *pvt* (different from the *S* pivots in *Pivots*) in line 12, and obtain a new pivot set, *newPivots*, by replacing a random pivot in *Pivots* with *pvt* (line 13), and then generate a cluster set, *newcls*, based on *S* pivots in *newPivots* (line 14). In lines 16–24, we calculate the summed probability that the query range of all sampling objects o_i does not intersect with the cluster $c_k \in cls$, as formally defined in Eq. (11). In particular, for iterations from 2 to *round*, we use a new variable, *newProb*, to store the summed probability. After each iteration, in lines 25–28, we will compare the probability, *newProb*, in current iteration with the best probability *Prob*. If *newProb* $>$ *Prob*, we update the best pivot set *Pivots* (line 27) and the best cluster set cls^* (line 28). After all iterations, we return the best cluster set cls^* as our output (line 29).

Algorithm 4. Selection of the best cluster set cls^* based on the cost model in Eq. (11).

Input: a data repository R, DD rules, dimensionality d, page size $page\_size$, object size obj_size, and the number, $round$, of rounds
Output: an optimal cluster set cls^*

```
1  max ← ⌊page_size / obj_size⌋ // maximum number of objects one cluster can store
2  min ← M/2 // minimum number of objects one cluster can store
3  S ← 2×|R| / (max+min) // number of clusters, and |R| is the size of data repository R
4  Pivots ← S random points in the repository space
5  Prob ← 0
6  cls* ← null
7  iter ← 1
8  for iter = 1 to round do
9      if iter ≤ 1 then
10         generate a cluster set cls* based on S pivots in Pivots
11     else
12         generate a new pivot pvt.
13         obtain a new pivot sets, newPivots, via randomly replacing one pivot in Pivots by pvt.
14         generate a cluster set newcls based on S pivots in newPivots
15     newProb ← 0
16     for i = 1 to s do
17         for j = 1 to d do
18             for each DD rule, DD, with dependent attribute A_j do
19                 for c_k ∈ cls* (or newcls) do
20                     if cluster c_k does not intersect with query range Q(o_i, DD) then
21                         if iter = 1 then
22                             Prob ← Prob + ψ(c_k, Q(o_i, DD))
23                         else
24                             newProb ← newProb + ψ(c_k, Q(o_i, DD))
25     if iter > 1 then
26         if newProb > Prob then
27             Pivots ← newPivots
28             cls* ← newcls
29 return cluster set cls*
```

B.3. Data imputation and pruning via I_j

Data Imputation via I_j. Next, we consider how to use DDs and index I_j to help impute the missing attribute A_j of an incomplete object o_i. First, as shown in Section 4, we need to select an available DD via fractal dimension. With the selected DD rule and incomplete object o_i, we can obtain a query range Q, as shown in Eq. (10). Then, starting from the root node, $root(I_j)$, of index I_j, when we encounter a non-leaf node e, we will check if the space of the node is intersected with the query range Q. If the answer is yes, we can obtain an interval that the possible values of missing attribute $o_i[A_j]$ fall into, by integrating the intervals of intersected buckets $buc_q.I$ (with query range Q) in the intersected node e. Besides, we need to further check the intersection between the query range Q with the space of children nodes of e. When we reach the leaf node level, if query range Q is intersected with some leaf nodes, we will check the clusters cls under these leaf nodes, and use the attribute values $s_r[A_j]$ of objects s_r in the intersected clusters cls to impute the missing attribute A_j of incomplete object o_i. Compared with the possible intervals that missing attribute $o_i[A_j]$ may fall into, the imputation in cluster level can obtain the exact possible values of $o_i[A_j]$, with corresponding confidences (i.e., probabilities).

Object Pruning via I_j. As shown in Section 5.3, we propose three pruning strategies in instance level. Actually, the apply of pruning strategies in instance level in Section 5.3 need to access the complete objects s_r in clusters under the leaf nodes of index I_j, as discussed in data imputation via I_j. But it is possible that we can prune some objects with low top-k probabilities before we reach the exact complete objects in clusters under index I_j. So we will briefly discuss how to apply these pruning strategies via the non-leaf or leaf nodes in index I_j.

As mentioned in data imputation via I_j, to impute an incomplete object o_i with missing attribute A_j, when we reach a node e in I_j intersected with query range Q, we can further obtain its intersected buckets buc_q with Q. Via these intersected buckets, we can obtain some intervals that values $o_i[A_j]$ may fall into, with corresponding probabilities, which can be used for the pruning strategies in Section 5.3. Thus, instead of getting some exact instance ranking scores of imputed object o_i^p, we can get some intervals of ranking scores of o_i^p, as shown in Fig. 12. In Fig. 12, when we apply Lemma 5.1, the upper bound of ranking scores of imputed object o_i^p is higher than that in Fig. 2, while the lower bounds of ranking scores of compressed object o_z^c are lower than that in Fig. 2. So the top-k probability of object o_i^p in Fig. 12 is higher than that in Fig. 2. In other words, if o_i^p can be pruned via nodes in I_j (Fig. 12), it can be pruned via instance level (Fig. 2). For the apply of the other two pruning strategies in Section 5.3 via I_j, we omit their details here.

Appendix C. Proofs of properties for double dual layer TDL

C.1. Proof of property 1 of TDL

Proof. By definition of top-k dual layers, layer 1 keeps the current top-k answer set (i.e., $Pr_{Topk-iDS}(o_i^p) > \alpha$) and layer 2 holds all objects o_i^p that are not within the current top-k answer set but may become top-k objects in the future (i.e., $0 < Pr_{Topk-iDS}(o_i^p) \leqslant \alpha$). Thus, this property is proved.

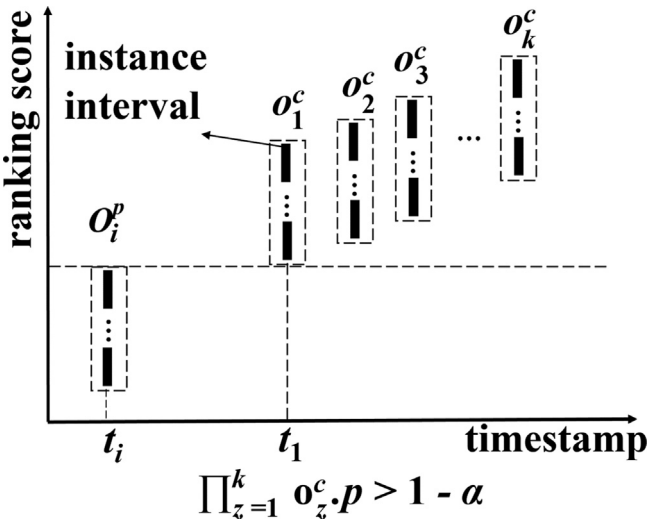

Fig. 12. The top-instance pruning via index I_j.

C.2. Proof of property 2 of TDL

Proof. For a valid object $o_i^p \in W_t$, the only requirement for o_i^p to be a current top-k object is that o_i^p has the probability greater than threshold α (i.e., $Pr_{Topk-iDS}(o_i^p) > \alpha$) to be a top-$k$ object, which is the requirement of objects on layer 1 of *TDL*.

Appendix D. More experimental results

The Top$k$-iDS Performance vs. the Number, $|W_t|$, of Valid Objects in W_t. Fig. 13 illustrates the performance of our Top$k$-iDS approach for different numbers, $|W_t|$, of objects in the sliding window W_t, where $|W_t| = 1.5K, 3K, 6K, 12K, 15K$, and $30K$, and default values are used for other parameters. When $|W_t|$ becomes larger, the wall clock time of our Top$k$-iDS approach also increases for all real/synthetic data sets. This is reasonable, since more Top$k$-iDS query candidates need to be dynamically maintained in the *TDL* synopsis. With different $|W_t|$, the wall clock time remains low (e.g., less than 0.46 s even when $|W_t| = 30$ K).

The Top$k$-iDS Performance vs. the Size, $|R|$, of the Data Repository R. Fig. 14 examines the performance of our Top$k$-iDS approach for different sizes, $|R|$, of (static) data repository R, where $|R|$ varies from 10 K to 50 K. From figures, when the data size $|R|$ increase, the wall clock time increases smoothly. With more complete data s_r in data repository R, the missing attributes will have more potential candidates, which will generate more possible instances for imputed objects. Nevertheless, the wall clock time still remains low (i.e., less than 0.038 s and 0.035 s for real and synthetic data sets, respectively), which verifies the effectiveness and efficiency of our index I_j over the data repository R.

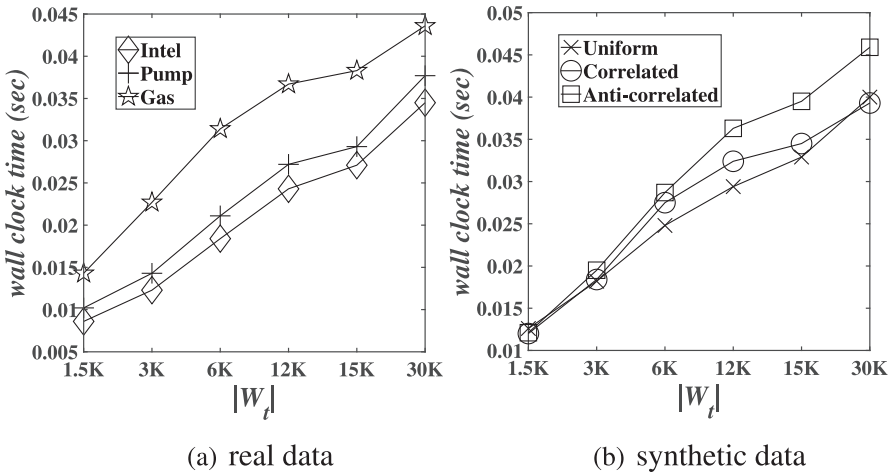

Fig. 13. The performance vs. No., $|W_t|$, of objects in W_t.

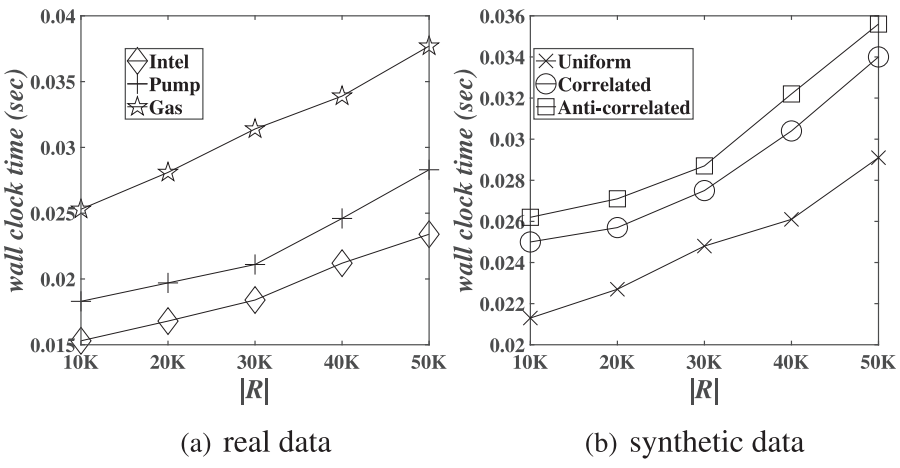

Fig. 14. The performance vs. the size, $|R|$, of data repository.

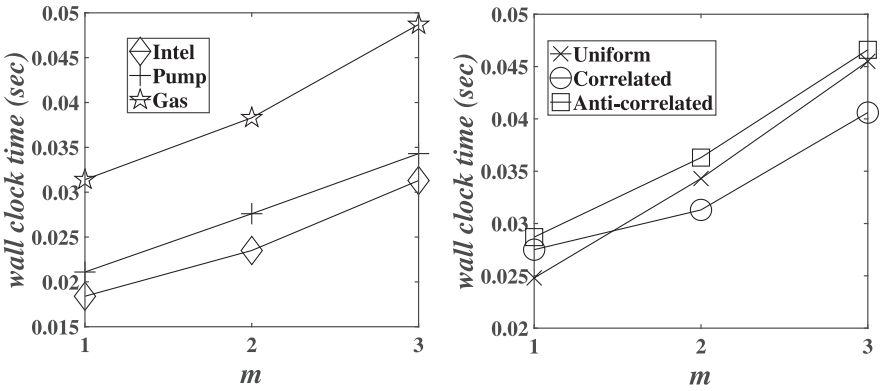

Fig. 15. The performance vs. No., m, of missing attributes.

The Top$k$-iDS Performance vs. the Number, m, of Missing attributes. Fig. 15 reports the effect of the number, m, of missing attributes on the performance of our Top$K$-iDS approach over synthetic data sets, where m varies from 1 to 3, and other parameters are set to default values. With more missing attributes, the number of possible instances in an imputed object will also increase. As shown in the figure, for large m, the wall clock time of our Top$k$-iDS approach increases smoothly, and remains low for $m = 3$ (i.e., less than 0.049 s and 0.047 s for real and synthetic data sets, respectively), which indicates the efficiency and effectiveness of our Top$k$-iDS approach.

References

[1] D.J. Abadi, W. Lindner, S. Madden, J. Schuler, An integration framework for sensor networks and data stream management systems, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases-Volume 30, 2004, pp. 1361–1364.
[2] L. Dhanabal, S. Shantharajah, A study on nsl-kdd dataset for intrusion detection system based on classification algorithms, Int. J. Adv. Res. Comput. Commun. Eng. 4 (6) (2015) 446–452.
[3] C. Cranor, T. Johnson, O. Spataschek, V. Shkapenyuk, Gigascope: a stream database for network applications, in, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 647–651.
[4] P. Haghani, S. Michel, K. Aberer, Evaluating top-k queries over incomplete data streams, in, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, 2009, pp. 877–886.
[5] K. Kolomvatsos, C. Anagnostopoulos, S. Hadjiefthymiades, A time optimized scheme for top-k list maintenance over incomplete data streams, Inf. Sci. 311 (2015) 59–73.
[6] S. Song, L. Chen, Differential dependencies: reasoning and discovery, ACM Trans. Database Syst. (TODS) 36 (3) (2011) 1–41.
[7] M. Hua, J. Pei, W. Zhang, X. Lin, Ranking queries on uncertain data: a probabilistic threshold approach, in, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, 2008, pp. 673–686.
[8] M.A. Soliman, I.F. Ilyas, K.C.-C. Chang, Probabilistic top-k and ranking-aggregate queries, ACM Trans. Database Syst. (TODS) 33 (3) (2008) 1–54.
[9] C. Jin, K. Yi, L. Chen, J.X. Yu, X. Lin, Sliding-window top-k queries on uncertain streams, Proc. VLDB Endowment 1 (1) (2008) 301–312.
[10] F.M. Choudhury, Z. Bao, J.S. Culpepper, T. Sellis, Monitoring the top-m rank aggregation of spatial objects in streaming queries, in: 2017 IEEE 33rd International Conference on Data Engineering (ICDE), IEEE, 2017, pp. 585–596.
[11] K. Mouratidis, S. Bakiras, D. Papadias, Continuous monitoring of top-k queries over sliding windows, in, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, 2006, pp. 635–646.
[12] G. Das, D. Gunopulos, N. Koudas, N. Sarkas, Ad-hoc top-k query answering for data streams, in: Proceedings of the 33rd International Conference on Very Large Data Bases, Citeseer, 2007, pp. 183–194.
[13] A. Das, J. Gehrke, M. Riedewald, Approximate join processing over data streams, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 40–51.
[14] N. Tatbul, S. Zdonik, Window-aware load shedding for aggregation queries over data streams, in: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB Endowment, 2006, pp. 799–810.
[15] N. Koudas, B.C. Ooi, K.-L. Tan, R. Zhang, Approximate nn queries on streams with guaranteed error/performance bounds, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases-Volume 30, 2004, pp. 804–815.
[16] Y. Tao, D. Papadias, Maintaining sliding window skylines on data streams, IEEE Trans. Knowl. Data Eng. 18 (3) (2006) 377–391.
[17] L. Qin, J.X. Yu, L. Chang, Scalable keyword search on large data streams, VLDB J. 20 (1) (2011) 35–57.
[18] X. Zhou, L. Chen, Event detection over twitter social media streams, VLDB J. 23 (3) (2014) 381–400.
[19] N. Prokoshyna, J. Szlichta, F. Chiang, R.J. Miller, D. Srivastava, Combining quantitative and logical data cleaning, Proc. VLDB Endowment 9 (4) (2015) 300–311.
[20] S. Song, H. Cheng, J.X. Yu, L. Chen, Repairing vertex labels under neighborhood constraints, Proc. VLDB Endowment 7 (11) (2014) 987–998.
[21] S. Song, A. Zhang, L. Chen, J. Wang, Enriching data imputation with extensive similarity neighbors, Proc. VLDB Endowment 8 (11) (2015) 1286–1297.
[22] V. Hristidis, N. Koudas, Y. Papakonstantinou, Prefer: a system for the efficient execution of multi-parametric ranked queries, ACM Sigmod Record 30 (2) (2001) 259–270.
[23] R. Fagin, A. Lotem, M. Naor, Optimal aggregation algorithms for middleware, J. Comput. Syst. Sci. 66 (4) (2003) 614–656.
[24] Y. Tao, V. Hristidis, D. Papadias, Y. Papakonstantinou, Branch-and-bound processing of ranked queries, Inf. Syst. 32 (3) (2007) 424–445.
[25] M.A. Soliman, I.F. Ilyas, K.C.-C. Chang, Top-k query processing in uncertain databases, in: 2007 IEEE 23rd International Conference on Data Engineering, IEEE, 2007, pp. 896–905.
[26] J. Li, B. Saha, A. Deshpande, A unified approach to ranking in probabilistic databases, VLDB J. 20 (2) (2011) 249–275.
[27] J.W. Graham, Missing Data: Analysis and Design, Springer Science & Business Media, 2012.

[28] W. Fan, J. Li, S. Ma, N. Tang, W. Yu, Towards certain fixes with editing rules and master data, Proc. VLDB Endowment 3 (1–2) (2010) 173–184.
[29] C. Mayfield, J. Neville, S. Prabhakar, Eracer: a database approach for statistical inference and data cleaning, in, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, 2010, pp. 75–86.
[30] Z.-G. Liu, Q. Pan, J. Dezert, A. Martin, Adaptive imputation of missing values for incomplete pattern classification, Pattern Recogn. 52 (2016) 85–95.
[31] A. Zhang, S. Song, J. Wang, P.S. Yu, Time series data cleaning: from anomaly detection to anomaly repairing, Proc. VLDB Endowment 10 (10) (2017) 1046–1057.
[32] S. Song, A. Zhang, J. Wang, P.S. Yu, Screen Stream data cleaning under speed constraints, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 827–841.
[33] W. Ren, X. Lian, K. Ghazinour, Efficient join processing over incomplete data streams, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 209–218.
[34] W. Ren, X. Lian, K. Ghazinour, Skyline queries over incomplete data streams, VLDB J. 28 (6) (2019) 961–985.
[35] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2002, pp. 1–16..
[36] R. Cheng, D.V. Kalashnikov, S. Prabhakar, Querying imprecise data in moving object environments, IEEE Trans. Knowl. Data Eng. 16 (9) (2004) 1112–1127.
[37] J. Widom, Trio: A system for data, uncertainty, and lineage, Manag. Min. Uncert. Data 35 (2008) 1–35.
[38] N. Dalvi, D. Suciu, Efficient query evaluation on probabilistic databases, VLDB J. 16 (4) (2007) 523–544.
[39] A. Belussi, C. Faloutsos, Self-spacial join selectivity estimation using fractal concepts, ACM Trans. Inf. Syst. (TOIS) 16 (2) (1998) 161–201.
[40] G. Aquino, J.D.J. Rubio, J. Pacheco, G.J. Gutierrez, G. Ochoa, R. Balcazar, D.R. Cruz, E. Garcia, J.F. Novoa, A. Zacarias, Novel nonlinear hypothesis for the delta parallel robot modeling, IEEE Access 8 (2020) 46324–46334.
[41] J. de Jesús Rubio, Sofmls: online self-organizing fuzzy modified least-squares network, IEEE Trans. Fuzzy Syst. 17 (6) (2009) 1296–1309.
[42] J.A. Meda-Campaña, On the estimation and control of nonlinear systems with parametric uncertainties and noisy outputs, IEEE Access 6 (2018) 31968–31973.
[43] I. Elias, J. d. J. Rubio, D.R. Cruz, G. Ochoa, J.F. Novoa, D.I. Martinez, S. Muñiz, R. Balcazar, E. Garcia, C.F. Juarez, Hessian with mini-batches for electrical demand prediction, Appl. Sci. 10 (6) (2020) 2036.
[44] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The r*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, 1990, pp. 322–331.
[45] S. Borzsony, D. Kossmann, K. Stocker, The skyline operator, in: Proceedings 17th International Conference on Data Engineering, IEEE, 2001, pp. 421–430..
[46] A. Zhang, S. Song, J. Wang, Sequential data cleaning: a statistical approach, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 909–924.
[47] S. Berchtold, D.A. Keim, H.-P. Kriegel, The x-tree: an index structure for high-dimensional data, Very Large Data-Bases (1996) 28–39.
[48] R. Ananthakrishna, A. Das, J. Gehrke, F. Korn, S. Muthukrishnan, D. Srivastava, Efficient approximation of correlated sums on data streams, IEEE Trans. Knowl. Data Eng. 15 (3) (2003) 569–572.