The VLDB Journal (2019) 28:961–985
https://doi.org/10.1007/s00778-019-00577-6

REGULAR PAPER

Skyline queries over incomplete data streams

Weilong Ren[1] · Xiang Lian[1] · Kambiz Ghazinour[1,2]

Received: 4 February 2019 / Revised: 16 August 2019 / Accepted: 27 September 2019 / Published online: 17 October 2019

Abstract
Nowadays, efficient and effective processing over massive stream data has attracted much attention from the database community, which are useful in many real applications such as sensor data monitoring, network intrusion detection, and so on. In practice, due to the malfunction of sensing devices or imperfect data collection techniques, real-world stream data may often contain missing or incomplete data attributes. In this paper, we will formalize and tackle a novel and important problem, named *skyline query over incomplete data stream* (Sky-iDS), which retrieves skyline objects (in the presence of missing attributes) with high confidences from incomplete data stream. In order to tackle the Sky-iDS problem, we will design efficient approaches to impute missing attributes of objects from incomplete data stream via *differential dependency* (DD) rules. We will propose effective pruning strategies to reduce the search space of the Sky-iDS problem, devise cost-model-based index structures to facilitate the data imputation and skyline computation at the same time, and integrate our proposed techniques into an efficient Sky-iDS query answering algorithm. Extensive experiments have been conducted to confirm the efficiency and effectiveness of our Sky-iDS processing approach over both real and synthetic data sets.

Keywords Skyline query · Incomplete data streams · Sky-iDS

1 Introduction

For decades, efficient management over massive data streams has received much attention in many real applications such as IP network traffic analysis [11], network intrusion detection [25], sensor networks [1], telephone call record management [22], Web log and clickstream mining [53], and so on. As an example, Fig. 1 shows an application of the coal mine surveillance [61], where sensors are deployed at different sites in tunnels of the coal mine and collect data attributes such as the densities of gas/oxygen/dust and temperature. These sensory samples are periodically obtained from each sensor and transmitted back to a *sink* in a streaming manner for real-time analysis, for example, detecting potentially abnormal events such as fire or gas explosion.

Table 1 depicts the sensory data stream, $iDS = (o_1, o_2, o_3, o_4, o_5, o_6, o_1, o_2, \ldots)$, collected from sensors and received by the sink (as shown in Fig. 1) in the order of their arrival times. Each record with sensor ID o_i (for $1 \le i \le 6$) has four sampled attributes such as temperature and densities of gas/oxygen/dust, which is associated with record arrival time and expiration time. For example, sensor (object) o_1 sends a sample record with attributes temperature 100 °F, and the densities of gas, oxygen, and dust all equal to 3, which arrives at the sink at time stamp 1 and will expire at time stamp 6, with a valid duration 5 ($= 6 - 1$). Similarly, objects $o_2 \sim o_6$ arrive at different times in a streaming fashion and may have distinct valid durations (due to different sensor sampling rates).

In order to timely detect dangerous events such as fire or explosion in the coal mine, one important query type in such a streaming scenario is the *skyline query* [7], which returns those sensors (and their locations in the coal mine) with high risks of incurring abnormal events (e.g., explosion event with both high temperature and density of gas). Specifically, given a database D, a skyline query retrieves those objects $o \in D$ that are not *dominated* by other objects in D, where we say an object o *dominates* another object o' (denoted as $o \prec o'$),

✉ Xiang Lian
xlian@kent.edu

Weilong Ren
wren3@kent.edu

Kambiz Ghazinour
kghazino@kent.edu

1 Department of Computer Science, Kent State University, Kent, OH 44242, USA

2 Center for Criminal Justice, Intelligence and Cybersecurity, State University of New York, Canton, NY 13617, USA

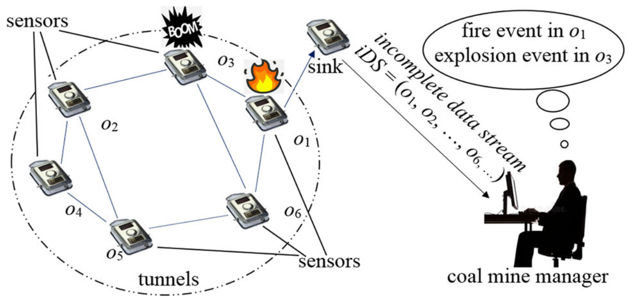


Fig. 1 An example of the coal mine surveillance

iff two conditions hold: (1) $o[A_i] \geq o'[A_i]$, for *all* attributes A_i, and; (2) $o[A_j] > o'[A_j]$, for *at least one* attribute A_j.

Note that, in this example of the coal mine surveillance, to detect sensors with high risks, one straightforward solution is to look at sensory values from each sensor using existing methods [52,63,64]. However, such a solution may encounter the problem of setting the alarming thresholds for different attributes, which are difficult to tune by the coal mine manager. In contrast, our skyline query does not require the specification of such thresholds and can directly return users with the most probable objects (i.e., sensor locations) in danger (e.g., sensors with fire/explosion events). The skyline considers multiple attributes (rather than just the value of one single attribute), which can be used for multi-criteria decision making. For skylines, we can obtain the locations of sensors that may have the most dangerous events (not dominated by other sensors). Under the dominance semantics between sensory objects, if a sensor S_1 dominates another sensor S_2, then we consider that the location of sensor S_1 is more dangerous than that of sensor S_2.

In the previous example of Table 1, object o_1 dominates object o_2, since each of the four attributes (i.e., temperature and densities of gas/oxygen/dust) in object o_1 is greater than that of object o_2. Thus, up to time stamp 2, the sink has only received two objects o_1 and o_2, and o_1 is the skyline answer (since it is not dominated by other object like o_2). Intuitively, the skyline answers, for example, sensor (object) o_1, indicate high risks of abnormal events (i.e., high temperature and/or density measures compared with other sensors), which require immediate attentions from the coal mine manager (for potential evacuation to save the lives of workers). Therefore, it is very critical, yet challenging, to study efficient and effective processing of skyline queries over such data streams.

Due to transmission errors, packet losses, low battery power, or environmental factors, some sensory data attributes may be missing and thus incomplete. For example, in Table 1, object o_3 has an incomplete attribute, the density of oxygen, whose missing value is denoted by "–". Similarly, objects $o_4 \sim o_6$ contain 1 or 2 missing attributes each. Due to the missing information, inaccurate skyline answers over incomplete streams may lead to wrong decision making about the coal mine evacuation, or even false alarms that incur losses of millions of dollars resulting from unnecessary evacuation. In such a scenario with incomplete data, it is even more challenging and important to process skyline queries efficiently and accurately over incomplete data streams.

Table 1 An incomplete data stream, iDS, collected from sensor networks in Fig. 1

Sensor ID (object)	Arr. time	Exp. time	Temperature (°F)	Density of gas	Density of oxygen	Density of dust
o_1	1	6	100	3	3	3
o_2	2	6	50	1	1	1
o_3	3	9	90	2	–	3
o_4	3	9	60	–	1	–
o_5	6	11	70	2	2	–
o_6	6	10	–	2	3	2
o_1	7	12	80	2	2	2
o_2	8	12	90	1	3	3
…	…	…	…	…	…	…

Table 2 An incomplete data stream, iDS, collected from computer networks

Router ID (object)	arr. time	exp. time	[A] No. of connections ($\times 10^3$)	[B] Connection duration (min)	[C] Transferred data size (GB)
T_1	1	6	0.5	0.5	0.2
T_2	2	6	0.5	0.2	0.5
T_3	3	9	0.5	0.5	0.5 (–)
…	…	…	…	…	…

Inspired by the example above, in this paper, we will formally propose the problem of the *skyline query over incomplete data streams* (Sky-iDS), which retrieves those skyline objects from incomplete data streams with high confidences. The Sky-iDS problem has many other real applications such as the network intrusion detection [25].

Specifically, in computer networks, spatially distributed routers often suffer from malicious network intrusion, where each router is connected with a number of servers. Since the network intrusion may lead to serious consequences such as virus installation, network congestion, and leakage of users' information, it is very crucial to online monitor and prevent the network intrusion, based on network statistics such as *No. of connections* (denoted as A), *connection duration* (denoted as B), and *transferred data size* (denoted as C) [17] (as depicted in Table 2). In reality, there are many routers in IP networks, and a large volume of the collected streaming network statistics arrive at fast speed, which is rather challenging for network security people to efficiently and accurately monitor. What is more, some network statistics may be missing/lost, for reasons such as the network failure, cyber attacks, or network congestion. Therefore, in this case, network security users can issue a skyline query over such incomplete network statistics from the data stream.

As an example in Table 2, for each router, T, we use $T = (A, B, C)$ to represent its collected network statistics, where A, B, and C are normalized to [0, 1]. At each time stamp, given the collected network statistics from three routers, $T_1 = (0.5, 0.5, 0.2)$, $T_2 = (0.5, 0.2, 0.5)$, and $T_3 = (0.5, 0.5, 0.5)$, network security people can obtain router T_3 as the only skyline router, based on dominance relationships among $T_1 \sim T_3$. Intuitively, T_3 is the router that may be under attack with the highest probability among the three routers and should be reported to network security people. If T_3 is safe (i.e., not under attack), then network security people may not need to monitor the rest two routers (i.e., T_1 and T_2), since T_1 and T_2 are dominated by T_3. However, in practice, these network statistics may be potentially unavailable (e.g., missing due to the network failure or network congestion). For instance, when *transferred data size* (i.e., attribute C) of T_3 is not available (i.e., $T_3 = (0.5, 0.5, -)$), it is not trivial how to retrieve skylines over such incomplete data from the stream. In this scenario, we can exactly issue a Sky-iDS query to monitor skylines over such a (incomplete) network data stream, which correspond to the routers with high risks of being under cyber attacks.

Note that, while prior works [21,27] studied the skyline query over *static* incomplete databases, their proposed approaches compute skylines by simply ignoring those missing attributes (when considering dominance relationships), which may incur biased or wrong skyline results (Please refer to Sect. 7 for a detailed example). Instead, in this paper, we will consider the imputation of missing attributes in data streams via *differential dependency* (DD) rules [47], which allows the skyline computation with all (complete or imputed) attributes and results in unbiased skylines with high confidences. Moreover, to the best of our knowledge, this is the first work to study the skyline operator over incomplete data in the streaming environment.

Specifically, in the streaming scenario, Sky-iDS query processing requires high efficiency, which is critical and important in many real applications. For example, as shown in Fig. 1, the coal mine manager needs to quickly and timely detect dangerous fire events (i.e., Sky-iDS answers) and immediately take actions. If Sky-iDS query answering is slow, then it may lead to enormous economic loss or even threaten people's lives. Similarly, in the scenario of network intrusion detection, high Sky-iDS processing cost may cause more servers and computers under attack. Therefore, it is important that we can efficiently retrieve Sky-iDS answers from incomplete data streams in these scenarios (otherwise, serious consequences like economic/life losses or network intrusion may occur). While a straightforward method can conduct the skyline query *after* the data imputation, it may still take a long time to obtain the Sky-iDS answers, which is not suitable for fast stream processing. Thus, in our work, we design an efficient Sky-iDS approach that integrates data imputation and skyline query at the same time, which can perform much better than the straightforward method.

Therefore, due to stream processing requirements such as efficient stream processing and limited memory consumption, in this paper, we will design cost-model-based and space-efficient index structures for both data imputation and query processing, devise effective pruning methods to greatly reduce the Sky-iDS search space, and propose efficient Sky-iDS answering algorithms to perform the attribute imputation and incremental skyline computation at the same time (i.e., "imputation and query processing at the same time" style).

In this paper, we make the following major contributions.

1. We formalize a novel and important problem of the *skyline query over incomplete data stream* (Sky-iDS) in Sect. 2.
2. We design effective and efficient data imputation techniques via DD rules in Sect. 3.
3. We propose effective pruning strategies to reduce the search space of the Sky-iDS problem in Sect. 4.
4. We devise effective indexes and efficient algorithms to tackle the Sky-iDS problem on incomplete data stream in Sect. 5.
5. We demonstrate through extensive experiments the effectiveness and efficiency of our Sky-iDS approach in Sect. 6.

In addition, Sect. 7 reviews related works on stream processing, differential dependency, skyline queries, stream outlier detection and repair, and incomplete data management. Section 8 concludes this paper.

2 Problem definition

In this section, we formally define the problem of a *skyline query over incomplete data streams* (Sky-iDS), which takes into account the missing attribute values during the skyline query processing.

2.1 Incomplete data streams

We first define the data model for incomplete data streams.

Definition 1 *(Incomplete Data Streams)* An *incomplete data stream*, iDS, is an ordered sequence of objects, $\{o_1, o_2, o_3, \ldots, o_r, \ldots\}$, where objects o_i arrive at time stamp $o_i.arr$ and expire at time stamp $o_i.exp$. Each object o_i contains d attributes A_j (for $1 \leq j \leq d$), some of which have missing attribute values $o_i[A_j]$, represented by "−".

In Definition 1, an incomplete data stream iDS dynamically keeps in memory all objects that are currently valid (i.e., not expired). When a new object o_t arrives, o_t will be inserted into iDS; whenever, an old object $o_i \in iDS$ expires at time stamp $o_i.exp$, it will be evicted from iDS. Each object $o_i \in iDS$ has a valid period from time stamp $o_i.arr$ to time stamp $o_i.exp$, with a duration $o_i.dur$ $(= o_i.exp - o_i.arr)$.

In the example of Fig. 1 and Table 1, the incomplete data stream is given by $iDS = (o_1, o_2, \ldots)$, in which objects like o_3 contain incomplete attributes (e.g., the missing attribute, the density of oxygen, for object o_3). At time stamp 6, new objects o_5 and o_6 are added to iDS; whereas, old expired objects o_1 and o_2 are removed from iDS, which results in valid objects $\{o_3, o_4, o_5, o_6\}$.

Without loss of generality, in this paper, we use W_t to denote a set of objects in iDS that are valid (i.e., not expired) at time stamp t. As shown in the example of Table 1, at time stamp $t = 2$, we have $W_2 = \{o_1, o_2\}$. At time stamp $t = 6$, we have $W_6 = \{o_3, o_4, o_5, o_6\}$. Similarly, at time stamp $t = 8$, we have $W_8 = \{o_3, o_4, o_5, o_6, o_1, o_2\}$. Note that, here objects o_1 and o_2 in W_8 are new updates at time stamp 8 from sensors o_1 and o_2, respectively, which are different from that in W_2 at time stamp 2.

2.2 Imputation over incomplete data stream

Imputed data stream To leverage the processing on incomplete data streams, in this paper, we will impute and model incomplete data stream iDS by *probabilistic data stream* [18], by estimating possible values of missing attributes in objects from iDS.

Definition 2 *(Imputed Data Stream)* Given an incomplete data stream $iDS = (o_1, o_2, \ldots, o_r, \ldots)$, its imputed (complete) data stream, pDS, is given by an ordered sequence of objects, $(o_1^p, o_2^p, \ldots, o_r^p, \ldots)$.

Each object $o_i^p \in pDS$, obtained from object $o_i \in iDS$ with missing attribute(s) $o_i[A_j]$ (="−"), is a probabilistic object, which consists of instances o_{il} (with the imputed attribute values). Each instance o_{il} is associated with an existence probability $o_{il}.p$, where $\sum_{\forall o_{il} \in o_i^p} o_{il}.p = 1$.

Definition 2 defines a probabilistic data stream pDS, imputed from incomplete data stream iDS. Specifically, we can estimate and impute possible values of each missing attribute $o_i[A_j]$ in objects $o_i \in iDS$, and represent the resulting probabilistic object o_i^p by several instances o_{il}. Each instance o_{il} contains complete/imputed attribute values, associated with an existence probability $o_{il}.p \in (0, 1]$, which indicates the confidence that instance o_{il} actually exists in reality (i.e., truly representing object o_i).

Table 3 shows an example of the imputed data stream pDS at time stamp $t = 6$ (i.e., $W_6 = (o_3^p, o_4^p, o_5^p, o_6^p)$), obtained from incomplete data stream iDS in Table 1. As an example, probabilistic object o_3^p has two instances o_{31} and o_{32}, with the imputed possible values 2 and 3 for attribute "density of oxygen", which are associated with existence probabilities 0.4 and 0.6, respectively. Similarly, probabilistic object o_4^p contains 4 instances $o_{41} \sim o_{44}$, where each missing attribute, "density of gas" or "density of dust", has two possible (imputed) values (i.e., 1 or 2). In particular, instance o_{41} has "density of gas" equal to 1 with probability 0.8, and "density of dust" equal to 1 with probability 0.7. Thus, the instance o_{41} has the existence probability 0.56 $(= 0.8 \times 0.7)$.

The cases of probabilistic objects o_5^p and o_6^p are similar and thus omitted here.

Possible worlds over imputed data stream Following the literature of probabilistic databases [12], we consider the *possible worlds* semantics over (imputed) probabilistic data stream pDS at time stamp t, that is, a set, W_t, of valid (not expired) objects, where each possible world is a materialized instance of $W_t \in pDS$ that can appear in the real world.

Definition 3 *(Possible Worlds of the Imputed Data Stream, $pw(W_t)$)* Given an imputed data stream pDS at time stamp t (i.e., W_t), a possible world, $pw(W_t)$, of W_t is a set of object instances o_{il}, where o_{il} is an instance of probabilistic object $o_i^p \in W_t$ (i.e., satisfying $o_i.exp > t$).

Each possible world, $pw(W_t)$, has an appearance probability, $Pr\{pw(W_t)\}$, given as follows:

$$Pr\{pw(W_t)\} = \prod_{\forall o_{il} \in pw(W_t)} o_{il}.p. \quad (1)$$

Table 3 The imputed data stream, pDS, at time stamp 6 (i.e., W_6) in the example of Table 1

Object	Instance	Temperature (°F)	Density of gas	Density of oxygen	Density of dust	prob.
o_3^p	o_{31}	90	2	**2**	3	0.4
	o_{32}	90	2	**3**	3	0.6
o_4^p	o_{41}	60	**1**	1	**1**	0.56
	o_{42}	60	**1**	1	**2**	0.24
	o_{43}	60	**2**	1	**1**	0.14
	o_{44}	60	**2**	1	**2**	0.06
o_5^p	o_{51}	70	2	2	**2**	1.0
o_6^p	o_{61}	**90**	2	3	2	0.6
	o_{62}	**80**	2	3	2	0.4

Table 4 Possible worlds, $pw(W_6)$, of W_6 from the imputed data stream, pDS, at time stamp 6 in Table 3

Possible world of W_6	Content of $pw(W_6)$	Appearance probability
$pw_1(W_6)$	$(o_{31}, o_{41}, o_{51}, o_{61})$	0.1344
$pw_2(W_6)$	$(o_{31}, o_{41}, o_{51}, o_{62})$	0.0896
$pw_3(W_6)$	$(o_{32}, o_{41}, o_{51}, o_{61})$	0.2016
$pw_4(W_6)$	$(o_{32}, o_{41}, o_{51}, o_{62})$	0.1344
…	…	…
$pw_{16}(W_6)$	$(o_{32}, o_{44}, o_{51}, o_{62})$	0.0144

In the example of Table 3, probabilistic objects o_3^p, o_4^p, o_5^p, and o_6^p in W_6 have 2, 4, 1, and 2 possible instances, respectively. Therefore, there are totally 16 ($= 2 \times 4 \times 1 \times 2$) possible worlds of W_6 over imputed data stream pDS at time stamp 6, as depicted in Table 4. The appearance probability of each possible world can be computed by Eq. (1), for example, $Pr\{pw_1(W_6)\} = o_{31}.p \times o_{41}.p \times o_{51}.p \times o_{61} = 0.4 \times 0.56 \times 1 \times 0.6 = 0.1344$.

2.3 Skyline queries on incomplete data stream

In this subsection, we will define the skyline query over incomplete data streams (Sky-iDS). Before we introduce the Sky-iDS query, we first provide the definition of the dominance between two certain (or imputed probabilistic) objects.

Definition 4 *(Dominance Between Certain Objects o and o'* [7]*)* Given two objects o and o', we say that object o dominates object o', denoted by $o \prec o'$, if two conditions are satisfied:

- for any dimension $1 \leq i \leq d$, $o[A_i] \geq o'[A_i]$ holds, and;
- for some dimension $1 \leq j \leq d$, $o[A_j] > o'[A_j]$ holds.

Without loss of generality, in this paper, we use "the larger, the better" semantics (i.e., larger attribute values are better) for the dominance definition (and skyline as discussed later). Intuitively, as given in Definition 4, object o dominates object o', if and only if two conditions hold: (1) o is not worse than o' for all attributes A_i, and (2) o is strictly better than o' on at least one attribute A_j. If only the first condition is satisfied, we denote it as $o \preccurlyeq o'$.

In the example of Table 1, object o_1 dominates o_2, since all the four attribute values of o_1 are larger than that of o_2, respectively.

Next, we define the dominance probability between two imputed probabilistic objects o^p and o'^p.

Definition 5 *(The Dominance Probability Between the Imputed Probabilistic Objects o^p and o'^p)* Given two imputed probabilistic objects o^p and o'^p, the *dominance probability*, $Pr\{o^p \prec o'^p\}$, between o^p and o'^p is given by:

$$Pr\{o^p \prec o'^p\} = \sum_{\forall o \in o^p} \sum_{\forall o' \in o'^p} o.p \cdot o'.p \cdot \chi(o \prec o'), \qquad (2)$$

where o and o' are instances of probabilistic objects o^p and o'^p, respectively, and $\chi(z)$ is either 1 (if z is $true$) or 0 (if z is $false$).

As an example in Table 3, we compute the dominance probability, $Pr\{o_3^p \prec o_6^p\}$, between two probabilistic objects o_3^p and o_6^p. In particular, we first consider the dominance relationships between instances from o_3^p and o_6^p (based on Definition 4) and thus have: $\chi(o_{31} \prec o_{61}) = 0$, $\chi(o_{31} \prec o_{62}) = 0$, $\chi(o_{32} \prec o_{61}) = 1$, and $\chi(o_{32} \prec o_{62}) = 1$. Then, by Eq. (2), we can obtain the dominance probability: $Pr\{o_3^p \prec o_6^p\} = o_{31}.p \times o_{61}.p \times 0 + o_{31}.p \times o_{62}.p \times 0 + o_{32}.p \times o_{61}.p \times 1 + o_{32}.p \times o_{62}.p \times 1 = 0.6$.

Definition 6 *(Skyline Queries Over Incomplete Data Stream, Sky-iDS)* Given an incomplete data stream iDS and a probabilistic threshold α, a *skyline query over incomplete data stream* (Sky-iDS) continuously monitors those objects $o_i \in W_t$ from iDS at any time stamp t, such that their imputed probabilistic objects o_i^p are not dominated by other imputed objects $o_j^p \in W_t$ with skyline probabilities, $P_{Sky\text{-}iDS}(o_i^p)$, greater than threshold α, that is,

$$P_{Sky\text{-}iDS}(o_i^p) = \sum_{\forall pw(W_t)} Pr\{pw(W_t)\} \cdot \chi\left(\bigwedge_{\forall o_j^p \neq o_i^p \text{ and } o_{il}, o_{js} \in pw(W_t)} o_{js} \not\prec o_{il}\right) > \alpha, \qquad (3)$$

where $pw(W_t)$ is a possible world of W_t containing instances o_{il} or o_{js} of objects $o_i^p, o_j^p \in W_t$, respectively, $o_{js} \not\prec o_{il}$ indicates that o_{js} is not dominated by o_{il}, and $\chi(z)$ is given in Definition 5.

Intuitively, users can register a Sky-iDS query in Definition 6 by specifying a parameter α, which will continuously monitor those skyline objects over incomplete data stream iDS with high confidences (i.e., satisfying Inequality (3)).

As an example in Table 3, at time stamp $t = 6$, the Sky-iDS query will compute skyline answers over $W_6 = \{o_3^p, o_4^p, o_5^p, o_6^p\}$. Specifically, as given in Definition 6, we need to enumerate all possible worlds $pw_1(W_6) \sim pw_{16}(W_6)$ (as shown in Table 4) and compute the skyline probability, for example, $P_{Sky\text{-}iDS}(o_3^p)$, of each object over all possible worlds in Inequality (3). In W_6, we obtain $P_{Sky\text{-}iDS}(o_3^p) = 1$. If the user-specified probabilistic threshold α is 0.45, then we have $P_{Sky\text{-}iDS}(o_3^p) > \alpha$, which indicates that object o_3^p is one of our Sky-iDS query answers at time stamp $t = 6$.

Challenges To tackle the Sky-iDS problem, there are three major challenges. First, many existing works [15,33] on stream processing usually assume that the underlying data are complete. However, this assumption does not always hold in practice (e.g., sensory data attributes may be missing or not available). Directly discarding incomplete data objects may lead to the bias of skyline query results over the purged data stream. Thus, we cannot directly apply skyline query processing techniques over complete data to solve our Sky-iDS problem over incomplete data stream, and we should design an effective and efficient approach to impute possible missing attribute values of incomplete data objects.

Second, in the stream environment, it is rather challenging to efficiently process the imputed probabilistic data stream under *possible worlds* semantics [12]. In particular, as shown in Inequality (3), there are an exponential number of possible worlds, which are inefficient, or even infeasible, to enumerate. Thus, we need to design an effective approach to reduce the problem to the one over imputed objects in probabilistic data stream.

Third, it is not trivial either how to efficiently process the Sky-iDS query in incomplete data stream. In other words, we need to dynamically and incrementally maintain the Sky-iDS query answer set, upon insertions and deletions in incomplete data stream. Therefore, in this paper, we should design effective pruning or indexing mechanisms to reduce the problem search space and enable efficient Sky-iDS query answering.

2.4 Sky-iDS processing framework

Algorithm 1 illustrates a framework for our Sky-iDS query processing, which consists of three phases. In the first *offline pre-computation phase*, we offline build indexes $\mathcal{I}_j$ over a static (complete) data repository R for imputing attributes A_j, respectively (line 1). In the second *imputation and incremental Sky-iDS computation phase*, upon deletions (lines 2–3) and insertions (lines 4–7), we dynamically maintain a data synopsis, called *skyline tree* ST, over incomplete data stream iDS, which stores potential Sky-iDS candidates. For insertions in particular, we use indexes $\mathcal{I}_j$ over R to facilitate data imputation via DDs and apply our pruning strategies to rule out false alarms of Sky-iDS candidates (lines 5–6). Note that, in this paper, we focus on DDs, and leave other imputation methods as our future work. Finally, in the *refinement phase*, we refine Sky-iDS candidates in the skyline tree ST and return actual Sky-iDS answers (line 8).

Algorithm 1: Sky-iDS Processing Framework

Input: an incomplete data stream iDS, a static (complete) data repository R, a timestamp t, and a probabilistic threshold α
Output: a Sky-iDS query answer set over W_t
`// Offline Pre-Computation Phase`
1 construct indexes, $\mathcal{I}_j$, over data repository R
`// Imputation and Incremental Sky-iDS Computation Phase`
2 **for** *each expired object* o'_i *at timestamp* t **do**
3   update a *skyline tree*, ST, over W_t with o'_i and evict o'_i from W_t
4 **for** *each new object* o_i *with missing attributes* A_j *arriving at* W_t **do**
5   traverse index, I_j, over R and the *skyline tree*, ST, over W_t at the same time to enable DD attribute imputation and skyline computation, resp.
6   **if** *object* o_i^p *cannot be pruned by spatial, max-corner, and min-corner pruning strategies* **then**
7   incrementally update the *skyline tree*, ST, with new object o_i^p
`// Refinement Phase`
8 refine Sky-iDS candidates in the ST index and return actual Sky-iDS answers

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

3 Incomplete object imputation

In this section, we will discuss how to impute missing attributes in incomplete data stream iDS by using rules such

Table 5 Symbols and descriptions

Symbol	Description
iDS	An incomplete data stream
pDS	An imputed (probabilistic) data stream
o_i	An object arriving at time stamp i from stream iDS
o_i^p	An imputed probabilistic object in the imputed stream pDS
W_t	A set of valid objects from stream iDS or pDS at time stamp t
$pw(W_t)$	A possible world of imputed probabilistic objects in W_t
$t \prec o_i$	Object t dominates object o_i
$t \preccurlyeq o_i$	$t \prec o_i$ or $t \equiv o_i$

Table 6 An example of a complete data repository R with 2 DD rules, $DD_1 : (A \rightarrow D, \{[0, 10], [0, 2]\})$ and $DD_2 : (BC \rightarrow D, \{[0, 1], [0, 1], [0, 1]\})$

Object	A	B	C	D
s_1	90	2	2	3
s_2	60	1	1	1
s_3	70	2	2	2
s_4	90	2	3	2

as *differential dependencies* (DDs) [47]. In the sequel, we will first briefly introduce DD rules and then present an effective approach to impute missing attributes by a historical complete data repository with the help of conceptual lattices.

3.1 Preliminary: differential dependency

Attributes of real-world objects often have inherent value correlations. The *differential dependency* (DD) technique [47] is a useful and important tool to explore such attribute correlations among objects. Specifically, given a data repository, R, with complete data objects, we can obtain a set, Ω, of DD rules [47] over R. Each DD rule, denoted as $DD_s \in \Omega$, is represented in the form of $(X \rightarrow A_j, \phi[XA_j])$, where X are *determinant attribute(s)*, A_j is a *dependent attribute* ($A_j \notin X$), and $\phi[XA_j]$ is a *differential function* on attributes X and A_j. Here, the differential function $\phi[Y]$ specifies distance range restrictions on attributes Y, which contain a number of distance intervals, $A_y.I$, for attributes $A_y \in Y$, where $A_y.I = [0, \epsilon_{A_y}]$. In this paper, we have the assumption that a data repository R containing complete data is available for data imputation via DD rules. This data repository can be obtained from historical data (e.g., from data streams or other external sources). The data repository is used as a source to impute missing attributes from other non-missing attributes, and we do not assume that we can obtain all stream data coming in the future. We will leave this interesting topic of detecting DD rules from data streams as our future work.

Table 6 shows an example of a data repository R, which contains 4 attributes A, B, C, and D, and follows a set, Ω, of two DD rules, DD_1 and DD_2, below:

$DD_1 : (A \rightarrow D, \{[0, 10], [0, 2]\})$, and

$DD_2 : (BC \rightarrow D, \{[0, 1], [0, 1], [0, 1]\})$.

In Table 6, for DD_1 $(A \rightarrow D, \{[0, 10], [0, 2]\})$, if two objects, such as s_2 and s_3, have attribute A satisfying the distance constraint $A.I = [0, 10]$ (i.e., $|s_2[A] - s_3[A]| = 10 \in [0, 10]$), then they must have similar values of attribute D (i.e., $|s_2[D] - s_3[D]| = 1 \in [0, 2]$). The case of DD_2 is similar.

DD [47] is quite useful for many real applications, such as fraud detection over transaction records (e.g., two transactions of a credit card within an hour must occur within 100 miles). DD can be also used for imputing missing attributes, as will be discussed in the next subsection.

The advantages of using DDs as the imputation approach In this paper, we use DDs as our imputation approach, which has following advantages. Compared with imputation methods requiring exact matching (e.g., editing rule [20]), DD-based imputation approach can tolerate differential differences (e.g., $\phi[A] = [0, 10]$ for DD_1 in Table 6) between attribute values, which can lead to a good imputation result even in sparse data sets [47]. Compared with the state-of-the-art constraint-based imputation approach [64] (requiring labelled data in data streams), DD-based imputation approach does not require any labelled data and imputes missing values via complete historical data records (i.e., data repository R). Specifically, many existing imputation approaches (e.g., the constraint-based approaches [52,64]) usually impute data based on incomplete data themselves only, which may lead to the imputation failure. For example, [52] requires that any two consecutive tuples cannot be missing at the same time. Nevertheless, imputation via DDs does not have such limitations. Moreover, imputation via DDs can lead to good query accuracy for skyline operator over incomplete data streams, which can be confirmed in Sect. 6.4.

3.2 Data imputation via DDs

Data imputation with one single DD $X \rightarrow A_j$. Given an incomplete object $o_i \in iDS$ with missing attributes A_j (for $1 \leq j \leq d$), a (complete) data repository R, and a single DD rule $DD_s \in \Omega$ in the form $X \rightarrow A_j$, our goal is to impute the missing attribute A_j in object o_i by utilizing R and DD_s.

Intuitively, if some object s_r from complete data repository R has attribute values $s_r[X]$ the same as or similar to that of incomplete object o_i, then, according to the DD_s rule, their values of attribute A_j should also be similar. In other words, we use attribute value $s_r[A_j]$ of complete object $s_r \in R$ as one possible imputed value of missing attribute $o_i[A_j]$ for incomplete object o_i.

In particular, given an incomplete object $o_i \in iDS$, if o_i has complete attributes X, then we can obtain all objects s_r from data repository R such that their attribute values of $s_r[X]$ satisfy distance constraints with $o_i[X]$ based on DD_s, that is, for each attribute $A_x \in X$, it holds that $|s_r[A_x] - o_i[A_x]| \in A_x.I$. Next, all the retrieved objects $s_r \in R$ will contribute their attribute values $s_r[A_j]$ (i.e., samples) to imputing the missing attributes $o_i[A_j]$ for object o_i.

Without loss of generality, we assume that all the imputed values $s_r[A_j]$ via objects $s_r \in R$ have equal chances to represent actual attribute value $o_i[A_j]$ of incomplete object o_i. Therefore, we will count the frequency, $v.freq$, of each distinct imputed value, v, for attribute $o_i[A_j]$, and then we can calculate the probability that the missing attribute $o_i[A_j]$ of incomplete object o_i equals to z (for some $s_r[A_j]$) from complete object s_r as: $Pr\{o_i[A_j] = z\} = \frac{z.freq}{\sum_{\forall v} v.freq}$.

Let us consider an incomplete object $o_5 = (70, 2, 2, -)$ in the example of Table 1. Based on a DD rule $DD_1 : (A \rightarrow D, \{[0, 10], [0, 2]\})$, we will find all objects from the data repository R in Table 6 whose attribute A values are within 10-distance from $o_5[A] = 70$, that is, falling into interval $[60, 80]$ $(= [70 - 10, 70 + 10])$. In Table 6, objects s_2 and s_3 from R will be selected (since both $s_2[A]$ and $s_3[A]$ are within interval $[60, 80]$). Then, we will use their attributes D values, $s_2[D]$ $(= 1)$ and $s_3[D]$ $(= 2)$, to impute the missing attribute $o_5[D]$ of incomplete object o_5. Thus, $o_5[D]$ will be imputed with two possible values 1 and 2, each with a probability 0.5 $(= \frac{1}{1+1})$.

Note that, in this paper, we do not use the imputed attributes to further estimate other missing attributes. We will leave this interesting topic as our future work.

Data imputation with multiple DDs In practice, we may have multiple DD rules with the same dependent attribute A_j over data repository R, for example, $X_1 \rightarrow A_j$, $X_2 \rightarrow A_j$, ..., and $X_l \rightarrow A_j$. Given an incomplete object o_i with missing attribute A_j, assume that attribute sets $X_1 \sim X_l$ from DD rules are all complete in object o_i. Then, we will utilize attributes $o_i[X_1]$, $o_i[X_2]$, ..., and $o_i[X_l]$ to impute the missing attribute $o_i[A_j]$ (via R and DDs). In other words, we can apply a combined DD rule, $X_1X_2 \ldots X_l \rightarrow A_j$, to efficiently impute $o_i[A_j]$. Here, if two attribute sets X_a and X_b share the same attributes A_y, then we will use the intersection of their intervals $A_y.I$ as the distance constraint in $X_1X_2 \ldots X_l \rightarrow A_j$.

Note that, one straightforward method is to use l individual DD rules to separately impute $o_i[A_j]$. However, this method may lead to low efficiency and, most importantly, biased estimates of attribute value $o_i[A_j]$ (due to the correlations among determinant attributes in $X_1 \sim X_l$). On the other hand, if we apply all attributes $X_1X_2 \ldots X_l$ to impute $o_i[A_j]$ (though

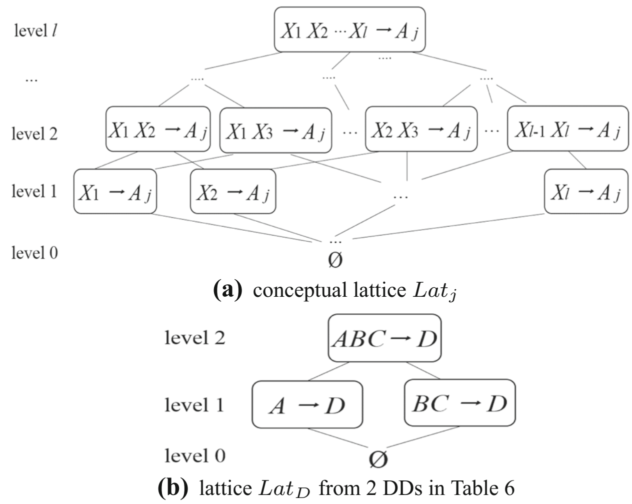


Fig. 2 Illustration of a conceptual lattice and its example

it is efficient), due to the limited number of samples in data repository R, it is possible that none of objects (samples) in R satisfy the distance constraints for all attributes $X_1X_2 \ldots X_l$, which cannot perform the imputation at all. Alternatively, in this paper, we will consider appropriate selection of attributes (e.g., a subset of $X_1X_2 \ldots X_l$) to impute attribute $o_i[A_j]$, making a balance between efficiency and accuracy.

Conceptual lattice Inspired by the reason above, in the sequel, we will propose a *conceptual lattice*, denoted by Lat_j (for $1 \leq j \leq d$), which can facilitate the decision of selecting DD rules for imputing the missing attribute A_j. Figure 2a shows the logical structure of the conceptual lattice Lat_j, which consists of $(l + 1)$ levels. Specifically, on level 0, we have an empty set, Ø, indicating that we cannot use any DD rules to infer attribute A_j; on level 1, we have l nodes, each corresponding to a DD rule DD_s: $X_s \rightarrow A_j$; on level 2, lattice nodes contain rules in the form of $X_aX_b \rightarrow A_j$; and so on. Finally, on level l, we have one node with a combined DD rule $X_1X_2 \ldots X_l \rightarrow A_j$. Figure 2b depicts the conceptual lattice Lat_D for the example in Table 6 (with 2 DDs).

DD selection via lattice Given a conceptual lattice Lat_j and an incomplete object o_i with missing attribute A_j (i.e., $o_i[A_j] =$"$-$"), we need to decide which (combined) DD rule from lattice Lat_j should be selected for imputing $o_i[A_j]$. Algorithm 2 illustrates the pseudocode of the DD selection algorithm, which traverses the lattice Lat_j in a breadth-first manner. Specifically, we start the traversal of the lattice Lat_j from level l to level 0 (line 1). Intuitively, higher level of lattice Lat_j involves more determinant attributes (e.g., level l has the largest number of attributes in $X_1X_2 \ldots X_l$), which will lead to more accurate imputation results and higher imputation efficiency (i.e., handling fewer candidates in R). Thus, here, we will start from higher level first.

When we access level lv, for each node with DD rule, $Y \rightarrow A_j$, on this level, we offline rank these DDs in increasing order of the imputation cost (defined as the expected number of possible samples from R). Intuitively, DDs with high ranks will have both low imputation cost and smaller imputation errors. Thus, for DDs on the same level, we will consider DDs with high ranks first. Then, we will check if this combined DD rule can be used for imputing $o_i[A_j]$ (lines 2–4). In particular, if some complete objects s_r in R satisfy the distance constraints with incomplete object o_i on attributes Y (i.e., the number of samples for imputation is nonzero, estimated from histograms), then we will terminate the loop and return the DD rule $Y \rightarrow A_j$ as the best DD rule for imputing the missing attribute $o_i[A_j]$ (lines 3–4). Note that, if multiple combined DD rules on the same level lv satisfy distance constraints, then we will only return the one with higher rank. If the lattice traversal descends to level 0, this indicates that none of DDs can be used for imputation. In this case, we can only apply a statistics-based method [35] to impute $o_i[A_j]$ with possible values of attribute A_j over R, following some probabilistic distribution, where the probability of each possible value can be calculated by the count of this value in attribute A_j over R divided by the size of R (lines 5–6). For instance, given a value set $\{0.1, 0.1, 0.1, 0.2\}$ on attribute A_j over data repository R (assuming R only having 4 complete data records), and an incomplete data object o_i with missing value on the attribute A_j, if no DD can be used for imputation $o_i[A_j]$, we will fill $o_i[A_j]$ with 0.1 and 0.2 with probabilities 0.75 and 0.25, respectively. Note that, if no DD can be used for imputing $o_i[A_j]$, we may not be able to use other imputation approaches (e.g., editing rule [20]) to impute $o_i[A_j]$.

Algorithm 2: DD Selection Using Conceptual Lattice

Input: Lattice Lat_j, incomplete object o_i with missing attribute A_j, and data repository R
Output: the best DD rule from Lat_j to do the imputation

```
1 for level lv = l to 0 do
2     for each node, Y → A_j, on level lv in increasing order of the imputation cost do
3         if the number of samples in R satisfying the distance constraints on attributes Y is not zero then
4             return the DD rule, Y → A_j, in this node
5     if lv = 0 then
6         apply a statistics-based method [35] to impute o_i[A_j] with possible values of attribute A_j over R
```

Once we select an appropriate (combined) DD rule, $Y \rightarrow A_j$, we can use this rule to impute the missing attribute, $o_i[A_j]$, of incomplete object o_i, similar to the aforementioned case of the data imputation with a single DD.

4 Pruning strategies

Problem reduction As mentioned in Sect. 2, it is not efficient, or even not feasible, to compute the skyline probability, $P_{Sky\text{-}iDS}(o_i^p)$, in Inequality (3) by enumerating an exponential number of possible worlds $pw(W_t)$. In order to speed up the efficiency, we will reduce our Sky-iDS problem over possible worlds to the one on uncertain objects. In particular, we will rewrite the skyline probability $P_{Sky\text{-}iDS}(o_i^p)$ as the probability that instances, o_{il}, of o_i^p are not dominated by other (imputed) objects o_j^p, which is given in an equivalent form below:

$$P_{Sky\text{-}iDS}(o_i^p) = \sum_{\forall o_{il} \in o_i^p} o_{il}.p \cdot \prod_{\forall o_j^p \in W_t \wedge o_j \neq o_i} (1 - Pr\{o_j^p \prec o_{il}\}). \quad (4)$$

Since it is still not efficient for stream processing to calculate the probability in Eq. (4) for every object $o_i^p \in W_t$, in this paper, we will provide pruning lemmas below to filter out false alarms (i.e., objects with low skyline probabilities) and reduce the search space of the Sky-iDS problem.

Spatial pruning We first present an effective *spatial pruning* method, which utilizes the interval of each imputed attribute to rule out objects that can never be Sky-iDS answers (i.e., with zero skyline probabilities) over data stream.

Specifically, for each incomplete object o_i from data stream iDS, we use a *minimum bounding rectangle* (MBR), $o_i^p.MBR$, to represent its imputed object o_i^p. We denote $o_i^p.min$ and $o_i^p.max$ as minimum and maximum corners of MBR $o_i^p.MBR$, respectively, which have minimum and maximum possible coordinates on all attributes A_j in $o_i^p.MBR$.

Lemma 1 (Spatial Pruning) *Given two incomplete objects o_i and o' from incomplete data stream iDS, if $o'^p.min \prec o_i^p.max$ and $o'.exp \geq o_i.exp$ hold, then object o_i can be safely pruned.*

Proof Please refer to Appendix 9.1. □

As illustrated in Fig. 3a, (imputed) object o'^p dominates o_i^p, since the corner point, $o'^p.min$, of o'^p dominates that, $o_i^p.max$, of o_i^p. Moreover, since $o'.exp \geq o_i.exp$ holds, object o_i^p can never be the skyline in its lifetime and can be safely pruned (as given by Lemma 1).

Max-corner pruning Next, we present a *max-corner pruning* method, which uses the max-corner, $o_i^p.max$, of MBR $o_i^p.MBR$ to prune the false alarm.

Lemma 2 (Max-Corner Pruning) *Given two incomplete objects o_i and o' from incomplete data stream iDS, and a max-corner $o_i^p.max$ of the imputed object o_i^p, if $Pr\{o'^p \prec o_i^p.max\} \geq 1 - \alpha$ and $o'.exp \geq o_i.exp$ hold, then object o_i can be safely pruned.*

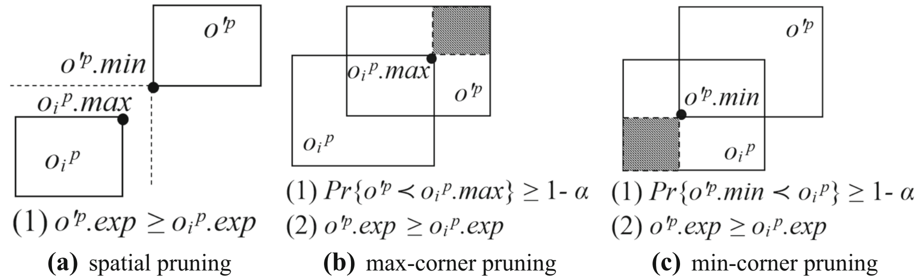


Fig. 3 Illustration of pruning strategies

Proof Please refer to Appendix 9.2. □

In Lemma 2, the probability $Pr\{o'^p \prec o_i^p.max\}$ is given by the probability that object o'^p falls into the shaded region w.r.t. max-corner $o_i^p.max$ (as shown in Fig. 3b). Intuitively, if $Pr\{o'^p \prec o_i^p.max\} \geq 1-\alpha$ holds, then object o_i^p is not dominated by o'^p with probability less than α, and in turn, the skyline probability, $P_{Sky\text{-}iDS}(o_i^p)$, of o_i^p is less than α. Moreover, object o'^p expires from pDS later than o_i^p. Thus, o_i^p cannot be a skyline in its lifetime and can be safely pruned.

Min-corner pruning Finally, we provide a *min-corner pruning* method, which uses min-corner, $o'^p.min$, of the MBR $o'^p.MBR$ to filter out object o_i^p with low skyline probability.

Lemma 3 (Min-Corner Pruning) *Given two incomplete objects o_i and o' from incomplete data stream iDS, and the min-corner, $o'^p.min$, of the imputed object o'^p, if $Pr\{o'^p.min \prec o_i^p\} \geq 1-\alpha$ and $o'.exp \geq o_i.exp$ hold, then object o_i can be safely pruned.*

Proof Please refer to Appendix 9.3. □

As an example in Fig. 3c, the probability $Pr\{o'^p.min \prec o_i^p\}$ in Lemma 3 is given by the probability that object o_i falls into the shaded region w.r.t. min-corner $o'^p.min$. Similar to Lemma 2, in Fig. 3c, object o_i^p is not dominated by o'^p with probability less than α (i.e., with low skyline probability), and o_i^p expires before o'^p. Thus, object o_i^p never has a chance to be the skyline during its lifespan and can be safely pruned.

Note that, for these three pruning rules, we will first apply the *spatial pruning* and then consider the *max-corner* and *min-corner* pruning rules if the *spatial pruning* fails.

5 Skyline processing on incomplete data stream

In this section, we will first propose a novel data synopsis, namely *skyline tree* (ST), which dynamically maintains Sky-iDS candidates over incomplete data stream iDS. Then, we will present index structures, $\mathcal{I}_j$, constructed over complete data repository R to facilitate missing data imputation. Next, we will discuss how to use data synopsis ST and indexes $\mathcal{I}_j$ to continuously monitor Sky-iDS query answers from incomplete data stream iDS, following the style of "imputation and query processing at the same time." Finally, we will provide cost models for index construction and parameter tuning.

5.1 Skyline tree

In this subsection, we will present the data structure of the skyline tree ST and then discuss properties of ST.

Data structure of the skyline tree In the sequel, we propose a multi-layer tree structure, namely *skyline tree* (ST), which is incrementally maintained over valid (imputed) objects (potential skyline candidates) $o_i^p \in W_t$ from incomplete data stream iDS. Intuitively, the skyline tree ST stores all possible Sky-iDS candidates over iDS that have chances to be skylines over time. If a Sky-iDS candidate (node) o_i^p on a layer of skyline tree ST expires, then its children (child nodes) o_c^p will become new skyline candidates.

Specifically, each node of the skyline tree ST corresponds to an (imputed) object, $o_i^p \in W_t$, which has one or multiple pointers pointing to its children o_c^p, such that: (1) each child o_c^p is dominated by its parent node o_i^p with probability greater than or equal to $(1-\alpha)$ (i.e., $Pr\{o_i^p \prec o_c^p\} \geq 1-\alpha$), and (2) o_c^p expires after o_i^p (i.e., $o_i^p.exp < o_c^p.exp$).

Moreover, for any two sibling nodes o_i^p and o_j^p on the same layer of the tree ST, they should dominate each other with probabilities less than $(1-\alpha)$, that is, (1) $Pr\{o_i^p \prec o_j^p\} < 1-\alpha$, and (2) $Pr\{o_j^p \prec o_i^p\} < 1-\alpha$.

Further, to obtain a tree structure, we use a virtual node (root) ∅ to point to all objects (skyline candidates) on the first layer of ST. In order to facilitate dynamic updates (e.g., deletions) in the streaming environment, for each layer of the ST tree, we will maintain the list of objects (nodes) o_i^p in non-descending order of their expiration times (i.e., $o_i^p.exp$).

Figure 4 illustrates a skyline tree ST over $W_8 = \{o_3, o_4, o_5, o_6, o_1, o_2\}$ in the example of Table 1, where $\alpha = 0.45$. This ST tree has 3 layers, $\{o_3^p\}$, $\{o_6^p, o_2\}$, and $\{o_1\}$. Consider objects (nodes) o_3^p and o_6^p in the tree structure. Node o_3^p is a parent of node o_6^p, since two conditions hold: (1) $Pr\{o_3^p \prec o_6^p\} = 0.6 \geq 1-\alpha = 0.55$, and (2) $o_3^p.exp < o_6^p.exp$.

Similarly, objects o_6^p and o_2 are sibling nodes on layer 2 of the ST tree. This is because (1) $Pr\{o_6^p \prec o_2\} = 0 < 1 - \alpha = 0.55$, and (2) $Pr\{o_2 \prec o_6^p\} = 0 < 1 - \alpha = 0.55$.

Moreover, objects o_4 and o_5 are not in the ST tree. This is because object o_4 (or o_5) is dominated by o_1 with probability 1 (in layer 3 of ST) and expires before o_1, which implies that o_4 (or o_5) can never be the skyline during its lifetime (i.e., always dominated by o_1 during the lifetime).

Properties of the skyline tree Next, we will provide the properties of the skyline tree ST.

Property 1 (Completeness) *The skyline tree ST contains all the objects o_i^p from iDS that have the chance to be skylines before they expire.*

Property 2 (No False Dismissals) *If an imputed object o_i^p is not on the first layer of the skyline tree ST over W_t, then o_i^p cannot be a skyline at current time stamp t.*

Property 3 (Superset of Sky-iDS Answers) *The set of objects o_i^p on the first layer of the skyline tree ST is a superset of Sky-iDS answers at current time stamp t.*

From the three properties above, we can see that the skyline tree ST contains a superset of Sky-iDS answers on the first layer of ST without any false dismissals. We will discuss later how to incrementally maintain this ST tree over incomplete data stream iDS. Please refer the proofs of these three properties to Appendix 10.

5.2 Cost-model-based indexes on data repository R for imputation

In this subsection, we will present indexes, $\mathcal{I}_j$, constructed from complete data repository R, which can facilitate quick imputation of missing attributes in data stream iDS.

Index structure In order to facilitate efficient data imputation, in this paper, we will devise d (i.e., the dimensionality of data sets, or the number of attributes in objects) effective indexes, $\mathcal{I}_j$ (for $1 \le j \le d$), each of which can help quickly access candidates s_r from data repository R and impute missing attributes $o_i[A_j]$. Specifically, given l DD rules $X_1 \to A_j$, $X_2 \to A_j$, ..., and $X_l \to A_j$ from Ω, we build an index $\mathcal{I}_j$ over those objects in R projected on attributes $U_j = X_1 \cup X_2 \cup \cdots \cup X_l$ as follows.

As illustrated in Fig. 5, we first divide the data space over attributes U_j into grid cells of equal size [30], where the side length of each cell is given by u. We will discuss later in Sect. 5.4 how to tune this parameter u, in light of our proposed cost model, for minimizing the imputation cost. Then, we insert each object $s_r \in R$ into a cell containing $s_r[U_j]$. Finally, we build an R*-tree [4] over those cells with objects, by invoking normal "insert" method. This way, the R*-tree over non-empty grid cells can be constructed, denoted as index $\mathcal{I}_j$, which can be used for imputing attribute A_j.

Note that, compared with directly using R*-tree [4] for imputation, our proposed index $\mathcal{I}_j$ can achieve better imputation cost. This is because, all objects in a non-empty grid cell are stored in a single leaf node in $\mathcal{I}_j$ (rather than multiple leaf nodes in the R*-tree), which incurs lower index traversal (DD imputation) cost than R*-tree.

Furthermore, each entry (MBR) e in nodes of index $\mathcal{I}_j$ is associated with a histogram, H_{U_j}, over attributes $U_j = X_1 \cup X_2 \cup \cdots \cup X_l$, which stores a summary of objects in e.

Histogram construction To build a histogram H_{U_j} for node e, we first divide each dimension $A_x \in U_j$ of the data space into λ intervals of equal size and obtain $\lambda^{|U_j|}$ buckets, denoted as buc_q (for $1 \le q \le \lambda^{|U_j|}$), where $|U_j|$ is the number of attributes in U_j (e.g., if $U_j = ABC$, then $|U_j| = 3$). Then, each bucket, buc_q, stores two items: (1) a COUNT aggregate, $buc_q.cnt$, of objects from R that fall into bucket buc_q; and (2) an interval, $buc_q.I = [buc_q.A_j^-, buc_q.A_j^+]$, of attribute values $s_r[A_j]$ for any objects $s_r \in R$ that fall into buc_q. Intuitively, the information stored in each bucket of the histogram can be used for spatial, max-corner, and min-corner pruning (as mentioned in Sect. 4).

As an example in Fig. 5, for data repository with attributes (A, B, C, D) and DD_1 and DD_2 in Table 6, index $\mathcal{I}_j$ over attributes $U_j = \{A, B, C\}$ contains a number of MBRs e, each of which is associated with a histogram, H_{ABC}. Given $\lambda = 2$, the histogram H_{ABC} has 8 ($= 2^{|ABC|} = 2^3$) buckets.

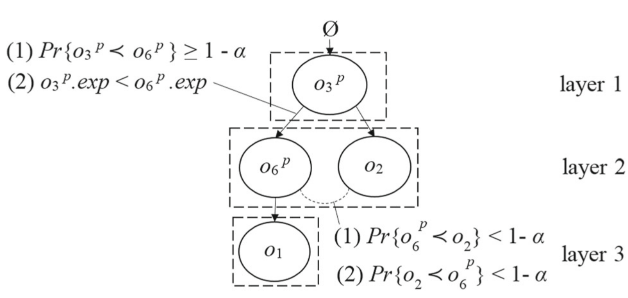


Fig. 4 An example of a skyline tree over incomplete data stream iDS at time stamp 8 (i.e., W_8) in Table 1 ($\alpha = 0.45$)

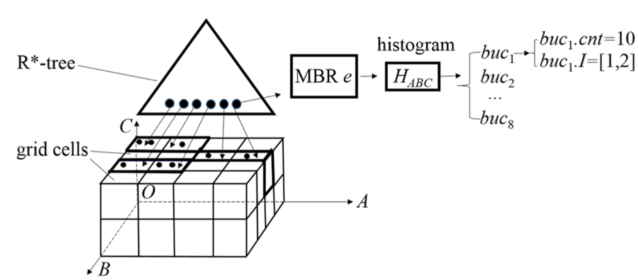


Fig. 5 The histogram associated with each MBR node $e \in \mathcal{I}_j$ w.r.t. $DD_1 : A \to D$ and $DD_2 : BC \to D$ in Table 6 ($\lambda = 2$)

Each bucket, buc_q, contains the number of objects in it (e.g., $buc_1.cnt = 10$ for bucket buc_1), and a value bound, $buc_q.I$ of attribute D for those objects in bucket buc_q (e.g., $buc_1.I = [1, 2]$).

Updates of index $\mathcal{I}_j$ Next, we consider how to maintain index $\mathcal{I}_j$ upon the appending of new objects for data repository R (though we consider R as static data set in our Sky-iDS problem). When a new complete object s_r comes in, we will insert this object s_r, by traversing indexes $\mathcal{I}_j$ from the root node to leaf nodes. During the index traversal, if we insert object s_r into an index node e, then we will: (1) increase the COUNT aggregate, $buc_q.cnt$, of bucket buc_q (containing s_r) in histogram H_{U_j} by 1; (2) update the minimum and maximum values of attribute A_j for the interval, $buc_q.I$, of bucket buc_q, and; (3) recursively insert s_r into one of children under node e. When we access a leaf node, we will insert object s_r into this leaf node (maintaining the index structure, if necessary) and update the information of a cell that contains object s_r.

Data imputation via indexes Next, we consider how to efficiently use indexes, $\mathcal{I}_j$, and DD rules, $X \to A_j$, to impute missing attribute A_j of an incomplete object $o_i \in iDS$. As discussed in Sect. 3.2, we will utilize the conceptual lattice to decide an appropriate DD rule $Y \to A_j$ (Algorithm 2) and then perform a range (aggregate) query over index $\mathcal{I}_j$ for attributes Y (note: range predicates on other attributes are wildcard $*$), where a query range Q is given by an MBR with $[o_i[A_x]-\epsilon_{A_x}, o_i[A_x]+\epsilon_{A_x}]$ on each dimension $A_x \in Y$.

Specifically, given a range query Q, we traverse index $\mathcal{I}_j$ over attributes Y (in the selected DD for imputation), starting from the root, $root(\mathcal{I}_j)$. When we encounter a non-leaf node e, we will check whether or not its children are intersecting with the query range Q (ignoring attributes other than Y). If the answer is yes, then we will access those intersecting children. When we encounter a leaf node e, we will obtain those cells intersecting with query range Q and retrieve objects $s_r \in R$ from cells that fall into Q.

After we retrieve all objects in the query range Q from $\mathcal{I}_j$, we can use their corresponding attribute A_j values (and confidences as well) to impute the missing attribute $o_i[A_j]$ of incomplete object $o_i \in iDS$.

As an example in Fig. 6, we can use index $\mathcal{I}_j$ (in Fig. 5) over attributes ABC to impute missing attribute D for an incomplete object o_i. In particular, with the help of a selected DD rule $DD_2 : BC \to D$, we can specify a query range:

$$Q = \left[o_i[B] - \epsilon_B, o_i[B] + \epsilon_B; o_i[C] - \epsilon_C, o_i[C] + \epsilon_C\right],$$

over attributes BC (wildcard "$*$" for other attribute A). In Fig. 6, we can obtain two nodes, e_2 and e_3, from index $\mathcal{I}_j$ intersecting with Q, each of which has four (projected) buckets, buc_q, intersecting with the query region Q. Correspondingly, we can retrieve the value bounds, $buc_q.I$, in these buckets buc_q to impute attribute D for incomplete object o_i. For example, in e_3, since all the 4 buckets are intersecting with Q, we can obtain lower/upper bounds of possible imputed attribute D w.r.t. e_3, that is, $[1, 4]$ ($= buc_1.I \cup buc_2.I \cup buc_3.I \cup buc_4.I$).

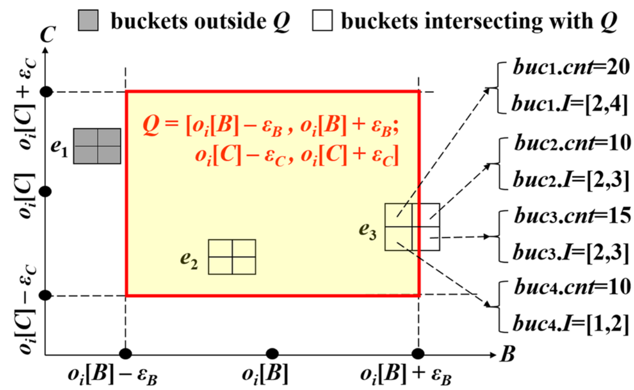


Fig. 6 The usage of index $\mathcal{I}_j$ for imputing $o_i[D]$ based on $DD_2 : BC \to D$

Object pruning via indexes As discussed in Sect. 4, we can apply spatial, max-corner, and min-corner pruning to filter out an (imputed) object o_i^p by using another object n^p, where the missing attributes in o_i^p and n^p are imputed by their possible values (inferred from data repository R). In the sequel, we will briefly discuss how to enable the pruning by traversing indexes $\mathcal{I}_j$ over R.

Specifically, when we access a level of index $\mathcal{I}_j$ for imputing attribute A_j of object o_i^p (or n^p), we can retrieve several possible value intervals of attribute A_j. Then, we can compute value boundaries, $buc_q.I$, of attributes A_j for object o_i^p (or n^p) and thus obtain corners $o_i^p.max$ and $n^p.min$, which can be used in the spatial pruning (as mentioned in Lemma 1). Similarly, we can also obtain COUNT aggregates, $buc_q.cnt$, for attribute A_j intervals from buckets buc_q and compute probabilities $Pr\{n^p \prec o_i^p.max\}$ and $Pr\{n^p.min \prec o_i^p\}$, which are used for max-corner and min-corner pruning (Lemmas 2 and 3), respectively. Similar to the pruning on the object level, we omit the pruning details via indexes.

5.3 Sky-iDS query processing algorithm

As discussed in Sect. 5.1 (Properties 1–3), the skyline tree ST always contains a superset of Sky-iDS query answers on its first layer. Therefore, in order to efficiently process Sky-iDS queries over incomplete data stream, one important issue is how to dynamically maintain this skyline tree ST in the streaming environment, upon object insertions and deletions. Then, we will discuss how to refine skyline candidates from (the first layer of) ST.

Algorithm 3: Insertion

Input: the skyline tree ST and a new object o_i
Output: the updated ST

1 $parentNode \leftarrow null$ `// parent node of` o_i^p `in` ST
2 $isPruned \leftarrow false$ `// whether` o_i^p `can be pruned`
3 $isAdded \leftarrow false$ `// whether` o_i^p `has been inserted`
4 **for** *each object* n^p *on layer* 1 **do**
5   **if** $Pr\{o_i^p \prec n^p\} \geq 1 - \alpha$ **then**
6     $isAdded \leftarrow true$ `// insert` o_i^p `into layer 1`
7   **if** $o_i^p.exp \geq n^p.exp$ **then**
8   replace n^p with o_i^p in ST `//` n^p `is pruned`
9   **else**
10   add o_i^p to the first layer of ST
11   move n^p and all its descendant nodes from their current layer L to layer $(L+1)$
12   let o_i^p be the parent node of n^p

13 **if** $isAdded = false$ **then**
14   queue $\mathcal{Q} \leftarrow$ all objects n^p on the first layer of ST
15   **while** $\mathcal{Q}$ *is not empty* **do**
16   remove n^p from $\mathcal{Q}$
17   **if** $Pr\{n^p \prec o_i^p\} \geq 1 - \alpha$ **then**
18   **if** $n^p.exp \geq o_i^p.exp$ **then**
19         $isPruned \leftarrow true$ `//` o_i^p `is pruned`
20   break; `// terminate the while loop`
21   **else**
        `// find the parent of` o_i^p
22   **if** $parentNode = null$ *or* $n^p.exp > parentNode.exp$ **then**
23           $parentNode \leftarrow n^p$
24   add all child nodes of n^p to $\mathcal{Q}$

25   **if** $isPruned = false$ **then**
26   **if** $parentNode = null$ **then**
      `//` o_i^p `is a skyline candidate`
27   add o_i^p to the first layer of ST
28   **else**
29   let o_i^p be the child node of $parentNode$

`// If` o_i^p `is inserted into` ST`, find children of` o_i^p `and use` o_i^p `to prune other objects in` ST
30 **if** $isPruned = false$ **then**
31   **for** *each object* n^p *from layer* $o_i^p.layer$ *to* $height(ST)$ **do**
32   **if** $Pr\{o_i^p \prec n^p\} \geq 1 - \alpha$ **then**
33   **if** $o_i^p.exp \geq n^p.exp$ **then**
34   remove n^p from layer $n^p.layer$
35   move up all descendant nodes o_c^p of n^p by $(o_c^p.layer - o_i^p.layer - 1)$ layer(s)
36   let o_i^p be the new parent for child nodes of n^p
37   **else if** $o_i^p.exp > par(n^p).exp$ **then**
38   let o_i^p be the new parent of n^p
39   move up n^p and all its descendant nodes o_c^p by $(o_c^p.layer - o_i^p.layer - 1)$ layer(s)
40   delete the edge between n^p and its old parent $par(n^p)$

5.3.1 Dynamic maintenance of the skyline tree

Insertion When a new object o_i arrives from incomplete data stream iDS, we will consider how to update the skyline tree ST with this (incomplete) object o_i. Specifically, Algorithm 3 illustrates the pseudocode to decide appropriate location to insert the imputed object o_i^p (if o_i is incomplete) and incrementally maintain the data structure of the ST index.

Basic idea In Algorithm 3, we initialize three variables, that is, $parentNode$, $isPruned$, and $isAdded$, which store the parent node of object o_i^p after the insertion, whether o_i^p can be pruned by some object in ST, and whether object o_i^p has been added to ST, respectively (lines 1–3). Then, we will find appropriate location in ST to insert object o_i^p, either on the first layer or on another layer pointed by a parent node, $parentNode$ (lines 4–29). Finally, we will update the ST index by removing those objects dominated by o_i^p and finding children of object o_i^p in ST (lines 30–40).

Finding the location to insert new object o_i^p First, we will check whether or not new (imputed) object o_i^p dominates any object n^p (i.e., $Pr\{o_i^p \prec n^p\} \geq 1 - \alpha$ holds) on layer 1 of ST (lines 4–5). If the answer is yes, then o_i^p can be inserted into layer 1 and the variable $isAdded$ is set to $true$ (line 6). Moreover, if object o_i^p expires after n^p (i.e., $o_i^p.exp \geq n^p.exp$), it indicates that n^p cannot be skyline any more (i.e., always dominated by o_i^p during its lifetime), and thus we replace n^p with o_i^p in ST (note: if there are duplicate objects o_i^p on the first layer, we will keep only one copy and merge their children; lines 7–8). Otherwise (i.e., $o_i^p.exp < n^p.exp$ holds; line 9), o_i^p should be a parent node of n^p. Therefore, we will add o_i^p to the first layer (line 10), move layers of n^p and all its descendant nodes from current layer L to $(L+1)$ (line 11), and let o_i^p point to n^p (line 12).

In the case that new object o_i^p has not been added to layer 1 (i.e., $isAdded = false$; line 13), we will utilize a queue, $\mathcal{Q}$, to search an appropriate parent node, $parentNode$, for this new object o_i^p (lines 14–24). Initially, we insert all objects on layer 1 of ST into the query $\mathcal{Q}$ (line 14). Each time we pop out one object, n^p, from queue $\mathcal{Q}$ (line 16). If n^p dominates o_i^p with probabilities greater than $(1-\alpha)$ and n^p expires after o_i^p, in this case, o_i^p can never be a skyline during its lifetime, that is, new object o_i^p should not be inserted into ST. Thus, we set variable $isPruned$ to $true$ and terminate the search loop (lines 17–20). When o_i^p cannot be pruned by n^p (as $n^p.exp < o_i^p.exp$ holds; line 21), we will set n^p as a temporary (best-so-far) parent, $parentNode$, of o_i^p, under one of the two conditions: (1) n^p is the first potential parent node we encounter (i.e., $parentNode = null$), or (2) n^p expires later than a best-so-far parent node, $parentNode$, of o_i^p (intuitively, o_i^p should be inserted under a parent node with the largest expiration time) (lines 22–23). Moreover, we will add children of node n^p to query $\mathcal{Q}$ for further searching (since these children may also be potential parent node of o_i^p in ST; line 24).

The loop of finding parent node $parentNode$ repeats, until queue $\mathcal{Q}$ becomes empty (line 15) or new object o_i^p can be pruned (line 20). If o_i^p cannot be pruned and $parentNode = null$ holds, it implies that no object can dominate o_i^p with high probability, and we can insert o_i^p into layer 1 as a skyline candidate (lines 25–27). On the other hand, if any parent node is found in variable $parentNode$, then we will let new object o_i^p be the child of $parentNode$ (lines 28–29).

Finding children of new object o_i^p and pruning objects in ST After we find the parent node of newly inserted object o_i^p in the ST index, we will next update the children of this new object o_i^p, as well as using o_i^p to prune/purge some (dominated) objects in ST (lines 30–40). That is, we will consider all objects, n^p, from the layer of o_i^p (i.e., $o_i^p.layer$) to the height of the ST index, and check whether o_i^p dominates n^p during n^p's lifetime (lines 31–33). If the answer is yes, then we will remove object n^p from ST, and let descendant nodes, o_c^p, of n^p be that of o_i^p (lines 34–36). Otherwise (i.e., n^p is not pruned), if o_i^p expires after the parent node, $par(n^p)$, of n^p, then we should let o_i^p be the new parent of n^p, move up n^p and all its descendants in ST and remove the link from old parent, $par(n^p)$, to n^p (lines 37–40).

Correctness of the insertion algorithm Please refer to the discussions about the correctness of the insertion algorithm in Appendix 11 in our technical report [43].

Algorithm 4: Deletion

```
  Input: the skyline tree ST and current timestamp t
  Output: the updated ST
1 for each expired object n^p on layer 1 do
2     remove object n^p from ST
3     move all descendant nodes of n^p from their current layer L to layer (L − 1)
```

Deletion At time stamp t, some objects o_i^p from incomplete data stream iDS are expired (i.e., $o_i^p.exp \leq t$). Algorithm 4 will remove all the expired objects from the skyline tree ST over iDS. In fact, we can prove that all the expired objects reside on layer 1 of ST (since objects on layers other than layer 1 will always expire after their parents). Since objects in each layer are sorted in ascending order of their expiration times, we will only check those expired objects on layer 1 (line 1). In particular, for each expired object n^p, we first remove it from ST (line 2) and move up all its descendant nodes o_c^p from their current layer L to layer $(L-1)$ (line 3).

Complexity analysis The object insertion in Algorithm 3 requires $O\big(|W_t| \cdot \frac{1-fanout(ST)^{height(ST)}}{1-fanout(ST)}\big)$ time complexity, where $|W_t|$ is the size of sliding window W_t (i.e., the number of descendants of o_i^p in index ST), $height(ST)$ is the height of the tree ST, and $fanout(ST)$ is the average number of children per node in ST. Similarly, the object deletion in Algorithm 4 needs $O\big(\theta \cdot \frac{1-fanout(ST)^{height(ST)}}{1-fanout(ST)}\big)$ time cost, where θ is the maximum number of expired objects on layer 1 of index ST.

Algorithm 5: Sky-iDS Refinement

```
   Input: the skyline tree ST, timestamp t, and a data stream W_t
   Output: the updated skyline answer set, A_t, at timestamp t
1  if there is no update with W_t at timestamp t then
2      return A_{t−1};
3  A_t = ∅;
4  if there is no new object added to W_t at timestamp t then
5      let A_t be A_{t−1} excluding all expired objects at timestamp t
       // objects in A_t are definitely skylines
6      let V be all objects on layer 1 of ST, but not in A_t
7  else
       // objects on layer 1 are potential skylines
8      let V be all objects on layer 1 of ST
9  for each object o_i^p ∈ V do
10     obtain a lower bound, lb_P(o_i^p), of probability P_{Sky-iDS}(o_i^p)
11     if lb_P(o_i^p) > α then
12         add o_i^p to A_t
13     else
14         compute exact Sky-iDS probability, P_{Sky-iDS}(o_i^p), of o_i^p
15         if P_{Sky-iDS}(o_i^p) > α then
16             add o_i^p to A_t
17 return A_t;
```

5.3.2 Sky-iDS refinement

After dynamic maintenance of the skyline tree ST over iDS, the first layer of ST always contains a superset of Sky-iDS answers at time stamp t, as guaranteed by Property 3 of ST (in Sect. 5.1). Thus, we will incrementally refine Sky-iDS candidates and return actual Sky-iDS query answers in a skyline answer set A_t.

Algorithm 5 provides the pseudocode of refining Sky-iDS candidates upon stream updates. In particular, if there is no update (insertion or deletion) at time stamp t, then skyline answers remain the same and we simply return skylines at previous time stamp $(t-1)$ in A_{t-1} (lines 1–2). In the case that there are deletions but no insertions, those objects in A_{t-1} (excluding expired objects) are still skylines at time stamp t. Thus, we add these non-expired objects in A_{t-1} to A_t, and objects on layer 1 of ST, but not in A_t, will form a candidate set V that should be refined (lines 3–6). On the other hand, if both insertions and deletions occur, then we will assign all objects on layer 1 to candidate set V (lines 7–8).

Next, we will refine objects o_i^p in the candidate set V by checking their Sky-iDS probabilities $P_{Sky\text{-}iDS}(o_i^p)$ (as given by Eq. (3); lines 9–16). Specifically, we will first calculate a lower bound, $lb\_P(o_i^p)$, of the skyline probability $P_{Sky\text{-}iDS}(o_i^p)$ (line 10). Here, the lower bound probability can be obtained by calculating the skyline probability of min-corner, $o_i^p.min$, of object o_i^p. If $lb\_P(o_i^p) > \alpha$ holds, object o_i^p will definitely be a skyline, and we add o_i^p to the skyline answer set A_t (lines 11–12). Otherwise, we need to compute exact skyline probability, $P_{Sky\text{-}iDS}(o_i^p)$, of o_i^p, and add o_i^p to A_t if the Sky-iDS probability is greater than α (lines 13–16).

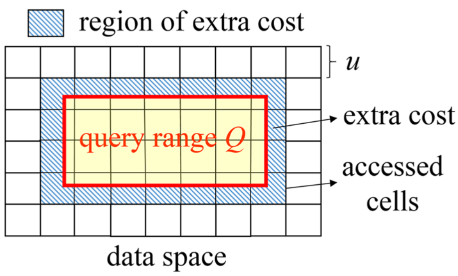


Fig. 7 Derivation of the cost model

Finally, we return actual Sky-iDS query answers in set A_t (line 17).

Correctness of the refinement algorithm Please refer to discussions on the correctness of the refinement algorithm in Appendix 11 in our technical report [43] (Lemmas 7 and 8).

Complexity analysis Algorithm 5 has $O(|W_t| \cdot \theta)$ time complexity in the worst case, where $|W_t|$ is the number of valid objects in sliding window W_t at time stamp t, and θ is the number of new objects per time stamp in data stream. At time stamp t, we need to update skyline probabilities of (at most $|W_t|$) objects on the first layer of the skyline tree ST, due to the insertion of at most θ new objects and the deletion of at most θ expired objects. Therefore, the worst-case refinement cost is given by $O(|W_t| \cdot \theta)$. Note that, in practice, the expected number of objects on the first layer of ST is much smaller than $|W_t|$. From our experiments over real/synthetic data sets (as discussed later in Sect. 6.3), the average number of objects on layer 1 of ST is about 2.8–11.76% of $|W_t|$. Thus, the refinement algorithm (Algorithm 5) is empirically quite efficient in the average case.

5.4 Cost model for parameter tuning

We provide a cost model to tune the parameter u (i.e., the side length of each cell in the grid) for index $\mathcal{I}_j$ over R (discussed in Sect. 5.2). The basic idea is to derive a cost model for the total cost (Fig. 7), Cost, to access the grid (w.r.t., parameter u). Then, we take the derivative of $Cost$ to u, and let it be 0, that is, $\frac{\partial Cost}{\partial u} = 0$, in order to find the optimal u that minimizes $Cost$. For the details, please refer to Appendix 12 in our technical report [43].

6 Experimental evaluation

6.1 Experimental settings

Real/synthetic data sets We evaluate the performance of our Sky-iDS approach on both real and synthetic stream data. Specifically, for real data, we use Intel lab data[1], UCI gas sensor data for home activity monitoring[2], Antallagma time series data for trading goods[3], and Pump sensor data for predictive maintenance[4], denoted as $Intel$, Gas Bid, and $Pump$, respectively. $Intel$ data are collected every 31 sec from 54 sensors deployed in Intel Berkeley Research lab on Feb. 28-Apr. 5, 2004, including 2.3 million readings. Gas data contain 919,438 sensory instances from 8 MOX gas sensors, a temperature and humidity sensor. Bid data contain 882K operation transactions between buyers and sellers from Jan. 2014 to Jun. 2016. $Pump$ has 220K data, collected from 52 sensors on Apr. 1-Aug. 31, 2018. We extract 4 attributes from $Intel$ data: temperature, humidity, light, and voltage; 10 attributes from Gas data: resistance of sensors 1-8, temperature, and humidity; 8 attributes from Bid data: price_sd, price_mean, price_max, price_min, mean, max, min, sd; and 10 attributes from $Pump$: sensor_01-sensor_10. We normalize all the attributes of each real data set to an interval [0, 10]. We obtain DD rules (as depicted in Table 7), by scanning all complete objects s_r in data repository R and all possible combinations of any two determinant/dependent attributes in the data schema [47] and selecting the ones with minimum interval for each dependent attribute A_j.

For synthetic data, we generate data repository R and incomplete data stream iDS as follows. Following the convention [7], we generate three types of d-dimensional data sets: $Uniform$, $Correlated$, and $Anti\text{-}correlated$, which correspond to different data distributions. Specifically, we first generate 5,000 seeds following uniform, correlated, or anti-correlated distribution [7]. Then, based on these seeds, we produce the remaining data objects, following DD rules as depicted in Table 7.

For real/synthetic data above, given a missing rate ξ (i.e., the probability that objects in the sliding window have missing attributes), for each incomplete object, we randomly set m out of d attributes to "–" (i.e., missing attributes) and obtain incomplete data stream iDS. Table 8 depicts the average number of instances per incomplete object for both real and synthetic data, where $m = 1$ and $\xi = 0.3$.

Competitor We compare our Sky-iDS approach with six competitors, namely $DD + skyline$, $mul + skyline$, $con + skyline$, $DD + skyline\_tree$, $mul + skyline\_tree$, and $con + skyline\_tree$. Note that, many existing works (e.g., [31,40]) for skyline on uncertain data are for static uncertain databases and require offline building an index and online traversing the index, which is not efficient for the stream sce-

1 http://db.csail.mit.edu/labdata/labdata.html.

2 http://archive.ics.uci.edu/ml/datasets/gas+sensors+for+home+activity+monitoring.

3 https://www.kaggle.com/abkedar/times-series-kernel.

4 https://www.kaggle.com/nphantawee/pump-sensor-data/version/1.

Table 7 The tested real/synthetic data sets and their DD rules

Data sets	DD rules
Intel	$voltage \rightarrow temperature$, {[0, 0.001], [0, 0]}
	$voltage \rightarrow humidity$, {[0, 0.001], [0, 0]}
	$voltage \rightarrow light$, {[0, 0.001], [0, 0]}
	$light \rightarrow voltage$, {[0, 0.001], [0, 9.89]}
Gas	$resistance4 \rightarrow resistance1$, {[0, 0.001], [0, 1.77]}
	$resistance3 \rightarrow resistance2$, {[0, 0.001], [0, 2.615]}
	$resistance2 \rightarrow resistance3$, {[0, 0.001], [0, 2.79]}
	$resistance5 \rightarrow resistance4$, {[0, 0.001], [0, 2.39]}
	$resistance4 \rightarrow resistance5$, {[0, 0.001], [0, 2]}
	$resistance1 \rightarrow resistance6$, {[0, 0.001], [0, 0.38]}
	$resistance3 \rightarrow resistance7$, {[0, 0.001], [0, 1]}
	$temperature \rightarrow resistance8$, {[0, 0.001], [0, 0.06]}
	$resistance8 \rightarrow temperature$, {[0, 0.001], [0, 0.07]}
	$resistance8 \rightarrow humidity$, {[0, 0.001], [0, 0.43]}
Bid	$price\_max \rightarrow price\_sd$, {[0, 0.001], [0, 5.73]}
	$price\_max \rightarrow price\_mean$, {[0, 0.001], [0, 4.58]}
	$price\_mean \rightarrow price\_max$, {[0, 0.001], [0, 6.58]}
	$price\_max \rightarrow price\_min$, {[0, 0.001], [0, 2.96]}
	$sd \rightarrow mean$, {[0, 0.001], [0, 3.5]}
	$sd \rightarrow max$, {[0, 0.001], [0, 3.21]}
	$mean \rightarrow min$, {[0, 0.001], [0, 2.11]}
	$max \rightarrow sd$, {[0, 0.001], [0, 2.06]}
Pump	$sensor\_06 \rightarrow sensor\_01$, {[0, 0.001], [0, 0]}
	$sensor\_06 \rightarrow sensor\_02$, {[0, 0.001], [0, 0]}
	$sensor\_06 \rightarrow sensor\_03$, {[0, 0.001], [0, 0]}
	$sensor\_06 \rightarrow sensor\_04$, {[0, 0.001], [0, 0]}
	$sensor\_08 \rightarrow sensor\_05$, {[0, 0.001], [0, 0]}
	$sensor\_07 \rightarrow sensor\_06$, {[0, 0.001], [0, 0.206]}
	$sensor\_01 \rightarrow sensor\_07$, {[0, 0.001], [0, 0.73]}
	$sensor\_07 \rightarrow sensor\_08$, {[0, 0.001], [0, 0.6]}
	$sensor\_01 \rightarrow sensor\_09$, {[0, 0.001], [0, 0.65]}
	$sensor\_08 \rightarrow sensor\_10$, {[0, 0.001], [0, 0]}
Uniform Correlated Anti-correlated	$B \rightarrow A$, {[0, 0.001], [0, 0.01]}
	$C \rightarrow B$, {[0, 0.001], [0, 0.01]}
	$D \rightarrow C$, {[0, 0.001], [0, 0.01]}
	$E \rightarrow D$, {[0, 0.001], [0, 0.01]}
	$F \rightarrow E$, {[0, 0.001], [0, 0.01]}
	$G \rightarrow F$, {[0, 0.001], [0, 0.01]}
	$H \rightarrow G$, {[0, 0.001], [0, 0.01]}
	$I \rightarrow H$, {[0, 0.001], [0, 0.01]}
	$J \rightarrow I$, {[0, 0.001], [0, 0.01]}
	$A \rightarrow J$, {[0, 0.001], [0, 0.01]}

nario. Therefore, we compare with the existing work [18] on skyline over uncertain data streams. The details of the six baseline methods are as follows (please refer to [18,56,63] for more implementation details).

- $mul + skyline$: this baseline first imputes the missing attribute values via *multiple imputation* [44] and then performs skyline query processing over imputed data streams via the algorithm in [18]. We implement the *multiple imputation*, by first obtaining 20 possible imputed

Table 8 Average number of instances per incomplete object for real/synthetic data sets (with $\xi = 0.3$ and $m = 1$)

Data sets	Average No. of object instances
$Intel$	14
Gas	19
Bid	23
$Pump$	11
$Uniform$	27
$Correlated$	25
$Anti\text{-}correlated$	27

values for each missing attribute A_j via Markov chain and prior distribution of attribute A_j in complete objects of R, and then computing the final imputed value by averaging the 20 imputed values [56];

- $mul+skyline\_tree$: this baseline first imputes the missing attribute values via *multiple imputation* [44] (with the same implementation as the $mul+skyline$) and then performs skyline query processing via the skyline tree over imputed data streams in our work;
- $con + skyline$: this baseline first imputes the missing attribute values via a *constraint-based imputation method* [63] and then uses the skyline query processing method in [18];
- $con+skyline\_tree$: this baseline first imputes the missing attribute values via a *constraint-based imputation method* [63] and then performs skyline query processing via the skyline tree over imputed data streams in our work;
- $DD + skyline$: this baseline first imputes the missing attribute values via DD rules and data repository R and then conducts the skyline query over imputed data streams via the algorithm in [18];
- $DD+skyline\_tree$: this baseline first imputes the missing attribute values via DD rules and data repository R and then performs skyline query via the skyline tree over imputed data streams in our work.

Measures In our experiments, we will report maintenance and query times of our proposed Sky-iDS approach, which are the CPU times to incrementally maintain the skyline tree ST (as discussed in Sect. 5.3.1; including the missing data imputation via $\mathcal{I}_j$) and to retrieve actual Sky-iDS query answers (by refining candidates on the first layer of ST, as mentioned in Sect. 5.3.2), respectively.

Parameter settings Table 9 depicts the parameter settings of our experiments, where default parameter values are in bold. In each set of experiments, we will vary one parameter, while setting other parameters to their default values. We ran our experiments on a machine with Intel(R) Core(TM) i7-6600U CPU 2.70 GHz and 32 GB memory. All algorithms were implemented by C++.

Table 9 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
The number, $\lvert W_t \rvert$, of valid objects in iDS	5K, 10K, **20K**, 40K, 50K, 80K
The size, $\lvert R \rvert$, of data repository R	40K, 80K, **120K**, 160K, 200K
The number, θ, of new objects per time stamp in iDS	10, 20, **30**, 40, 50, 100
The number, m, of missing attributes	**1**, 2, 3
The missing rate, ξ, of incomplete objects in iDS	0.1, 0.2, **0.3**, 0.4, 0.5

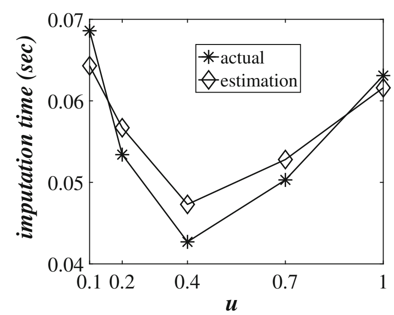
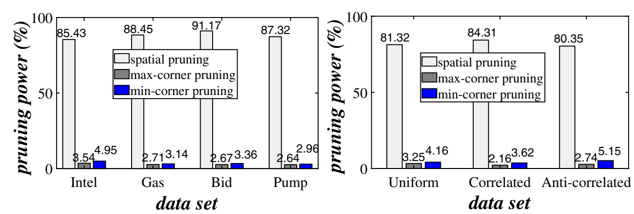


Fig. 8 Cost model verification for the imputation cost ($Uniform$)



(a) real data **(b)** synthetic data

Fig. 9 Pruning power evaluation over real/synthetic data sets

6.2 Verification of the cost model

We first verify our cost model in Sect. 5.4, by comparing the estimated and actual data imputation time over $Uniform$ data set, w.r.t. different side lengths, u, of cells in index $\mathcal{I}_j$, where $u = 0.1, 0.2, 0.4, 0.7$, and 1, and $|R| = 120K$. From the experimental results in Fig. 8, we can see that our estimated imputation cost can closely approximate the trend of actual imputation cost, which confirms the correctness of our proposed cost model for estimating the imputation cost. As a result, we can use our cost model to select the best value of side length u of cells, that minimizes the imputation cost. In

Fig. 8, the optimal u value is about 0.4, which matches with the u selection based on our cost model and thus indicates the effectiveness of our cost model.

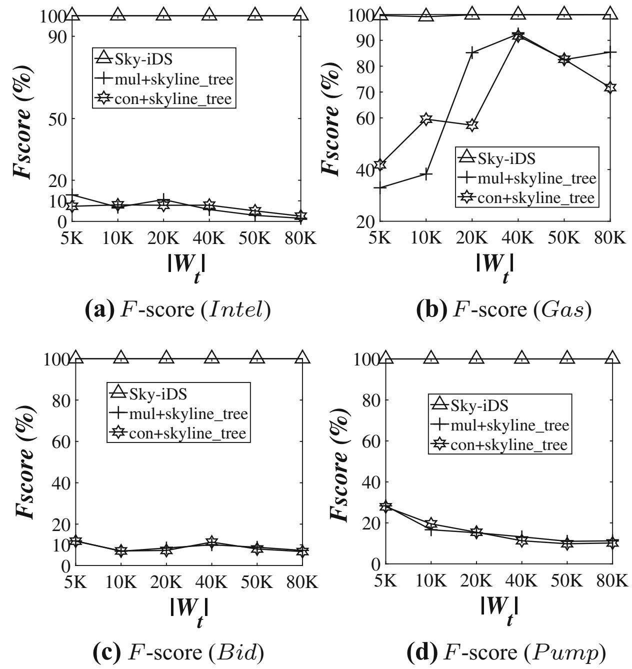
The verification results of the cost model for other data distributions (e.g., $Correlated$ and $Anti\text{-}Correlated$) are similar, and therefore omitted here.

6.3 Effectiveness of Sky-iDS pruning methods

Figure 9 demonstrates the percentages of objects that are pruned by our three pruning rules, spatial pruning, max-corner pruning, and min-corner pruning, over real/synthetic data sets, where parameters of synthetic data sets are set to their default values. As mentioned in Sect. 4, we will first apply the spatial pruning, followed by max-corner and min-corner pruning rules (if the spatial pruning fails). From figures, we can see that the spatial pruning can significantly prune most of data objects for both real and synthetic data sets (i.e., 85.43%-91.17% for real data sets and 80.35%-84.31% for synthetic data sets). Then, the max-corner and min-corner pruning rules can further reduce the Sky-iDS search space. To be specific, the max-corner pruning rule can further prune 2.64%-3.54% and 2.16%-3.25% of objects from real and synthetic data, respectively, whereas the min-corner pruning rule can further filter out 2.96%-4.95% and 3.62%-5.15% of objects in real and synthetic data, respectively. Overall, our proposed three pruning methods can together prune 92.92%-97.2% and 88.24%-90.09% of data objects in real and synthetic data sets, respectively, which indicates the effectiveness of our proposed Sky-iDS approach. Note that, from our experimental results, the first layer of our proposed *skyline tree ST* (as mentioned in Sect. 5.1) contains only 2.8%-11.76% of objects in the sliding window W_t, which confirms the effectiveness of our *skyline tree* and shows the efficiency of our Sky-iDS refinement algorithm (Sect. 5.3.2).

6.4 The effectiveness of Sky-iDS queries

In this subsection, we compare the effectiveness of our proposed Sky-iDS approach with that of $mul + skyline\_tree$ and $con + skyline\_tree$ over four real data sets (i.e., $Intel$, Gas, Bid, and $Pump$), in terms of the F-$score$. Note that, since $DD + skyline$ and $DD + skyline\_tree$ use the same DD-based imputation method as our Sky-iDS approach, they have the same F-score as our Sky-iDS approach. Thus, we will not report the effectiveness of $DD + skyline$ and $DD + skyline\_tree$ here. Similarly, since they have the same F-score as $mul + skyline\_tree$ and $con + skyline\_tree$, we will not report the effectiveness of $mul + skyline$ and $con + skyline$, respectively. Specifically, for each (complete) real data set, we first randomly select some objects as incomplete based on the missing rate ξ and then mark m out of d random attribute(s) as missing in the selected objects. This



(a) F-score ($Intel$) **(b)** F-score (Gas)

(c) F-score (Bid) **(d)** F-score ($Pump$)

Fig. 10 The Sky-iDS effectiveness versus the number, $|W_t|$, of valid objects in iDS

way, we can know the groundtruth of actual skyline query answers from complete real data and test the accuracy of the three approaches over (masked) incomplete data sets, in terms of the F-$score$ defined as follows.

$$F\text{-}score = 2 \times \frac{recall \times precision}{recall + precision}, \tag{5}$$

where *recall* is given by the number of actual skyline answers in our Sky-iDS query results divided by the total number of actual skyline answers in complete data sets, and the *precision* can be calculated by the total number of actual skyline answers in our Sky-iDS query results divided by the total number of objects returned by our Sky-iDS approach.

The Sky-iDS effectiveness versus the number, $|W_t|$, of valid objects in *iDS* Figure 10 shows the query accuracy of our Sky-iDS approach and other two competitors (i.e., $mul + skyline\_tree$ and $con + skyline\_tree$) over the four real data sets, where $|W_t| = 5K$, $10K$, $20K$, $40K$, $50K$ and $80K$, and other parameters follow their default values in Table 9. From figures, we can see that our Sky-iDS approach can achieve high F-$score$ over real data sets with different $|W_t|$ values (i.e., close to 100%), which significantly outperforms $mul + skyline\_tree$ and $con + skyline\_tree$.

The Sky-iDS effectiveness versus the missing rate, ξ, of objects in *iDS* Figure 11 demonstrates the query accuracy evaluation between our Sky-iDS approach and its competitors (i.e., $mul + skyline\_tree$ and $con + skyline\_tree$) over

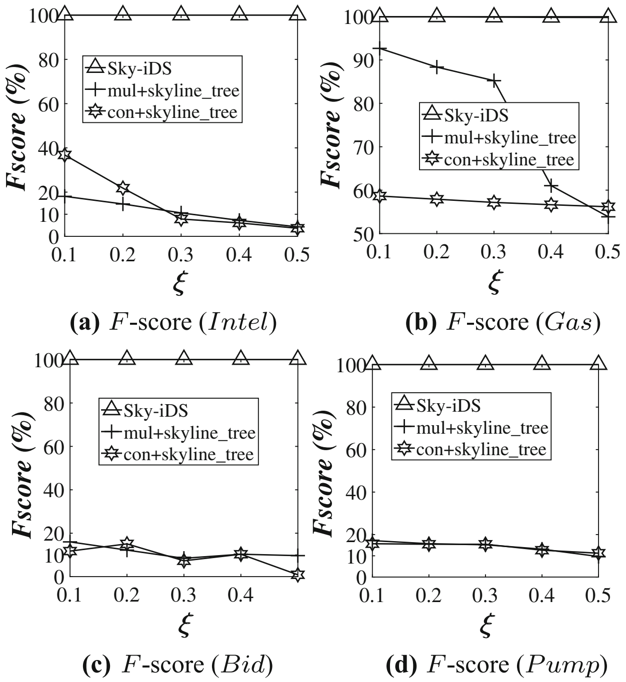


Fig. 11 The Sky-iDS effectiveness versus the missing rate, ξ, of objects in iDS

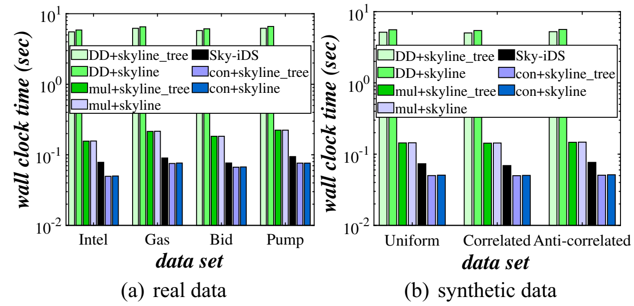


Fig. 12 The efficiency versus real/synthetic data sets

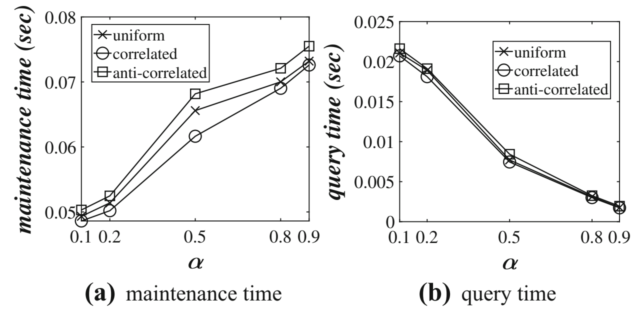


Fig. 13 The efficiency versus probabilistic threshold α

four real data sets, where missing rate ξ varies from 0.1 to 0.5, and other parameters are set to their default values in Table 9. As shown in figures, as the increase in the ξ, the F-*scores* of $mul + skyline\_tree$ and $con + skyline\_tree$ decrease smoothly. This is reasonable, since multiple imputation [44] and constrained-based imputation methods [63] may lead to higher imputation errors with higher missing rate ξ. Nevertheless, Fig. 11 shows that our Sky-iDS approach can still achieve high F-*score* (close to 100% even when $\xi = 0.5$) for all real data sets, which confirms the effectiveness of our Sky-iDS approach.

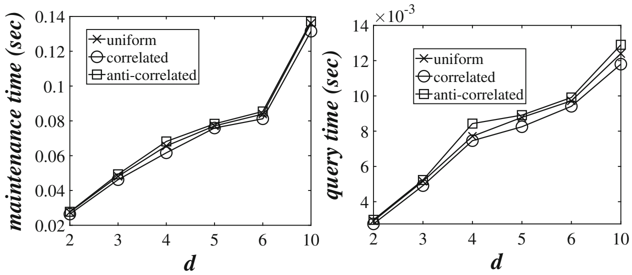
The experimental results with respect to *recall* and *precision* are similar and thus will not be reported here.

6.5 The efficiency of Sky-iDS queries

The Sky-iDS efficiency versus real/synthetic data sets Figure 12 illustrates the performance of our Sky-iDS algorithm, $DD + skyline$, $mul + skyline$, $con + skyline$, $DD + skyline\_tree$, $mul+skyline\_tree$, and $con+skyline\_tree$ over both real and synthetic data sets, where parameters of synthetic data sets are set to default values. We report the overall *wall clock time* of each approach, which includes both maintenance and query times. From experimental results, our Sky-iDS approach outperforms $DD + skyline$ and $DD+skyline\_tree$ algorithms by 2 orders of magnitude, has lower cost than the $mul + skyline$ and $mul + skyline\_tree$ approach, and slighter higher cost than the $con + skyline$ and $con + skyline\_tree$ approach, in terms of the wall clock time. The reason that our Sky-iDS approach is better than $DD + skyline\_tree$ and $DD + skyline$ is as follows. When Sky-iDS performs the imputation (via indexes over data repository R) and skyline processing (via skyline tree) at the same time, Sky-iDS can early prune incomplete objects on the level of index nodes. In contrast, $con + skyline$ and $DD + skyline\_tree$ need to impute incomplete objects to their instance level, by obtaining all samples from data repository R. Thus, our Sky-iDS approach outperforms $DD + skyline\_tree$ and $DD + skyline$ by two orders of magnitude, which verifies the efficiency of the "imputation and query processing at the same time" style of our Sky-iDS approach. Moreover, the experimental results show that our proposed Sky-iDS approach is comparable to $mul+skyline$, $mul+skyline\_tree$, $con+skyline$, and $con+skyline\_tree$, in terms of the efficiency, however, our Sky-iDS approach incurs much higher accuracy, as confirmed by Figs. 10 and 11.

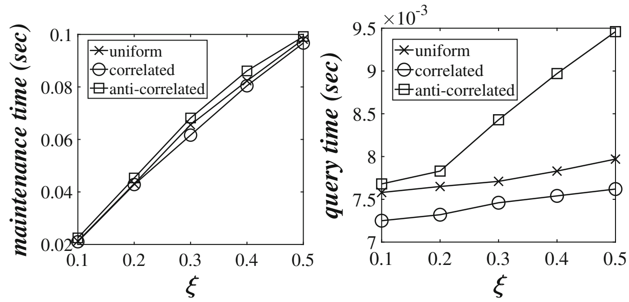
Below, we will test the robustness of our Sky-iDS approach by varying different parameters over synthetic data sets.

The Sky-iDS efficiency versus probabilistic threshold α Figure 13 shows the effect of the skyline probability threshold α on the Sky-iDS performance over three synthetic data, where $\alpha = 0.1, 0.2, 0.5, 0.8,$ and 0.9 and other parameters are set to default values. From figures, the maintenance time is low (less than 0.222 *sec*) and increases linearly for larger α over the three data sets, which shows good per-



(a) maintenance time **(b)** query time

Fig. 14 The efficiency versus dimensionality d



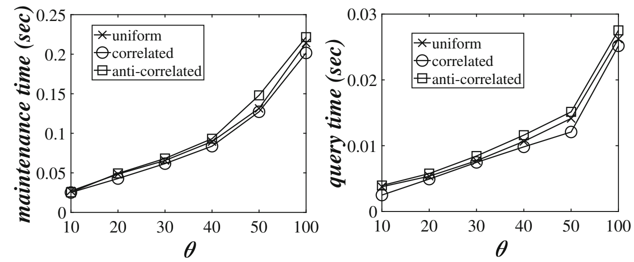
(a) maintenance time **(b)** query time

Fig. 15 The efficiency versus missing rate ξ

formance of our Sky-iDS approach to impute incomplete objects via indexes and incrementally maintain the skyline tree ST. Moreover, in Fig. 13b, when α increases, the query time decreases (due to the lower cost to incrementally refine skyline candidates on the first layer of ST) and remains small (i.e., 0.0251∼0.0275 sec). Thus, the experimental results confirm the efficiency of our Sky-iDS approach against different α values.

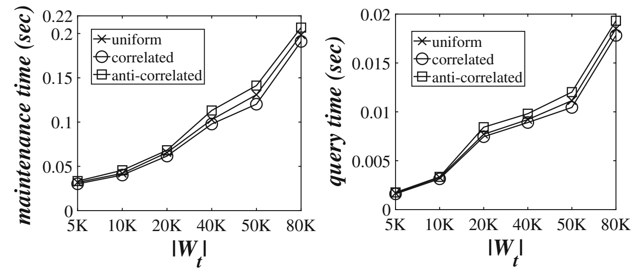
The Sky-iDS efficiency versus dimensionality *d* Figure 14 reports the performance of our Sky-iDS approach over synthetic data sets, by varying the number, d, of attributes in objects from 2 to 10, where other parameters are by default. As shown in Fig. 14a, with the increase in dimensionality d, the maintenance time increases. This is because, the maintenance time includes the data imputation cost via R*-tree and update time of the ST index. With higher dimensionality d, the imputation cost via R*-tree becomes higher, due to the "dimensionality curse" problem [5]; similarly, the updates of ST need to check the dominance relationships by considering more attributes, which incurs more time cost. Thus, the maintenance cost increases for larger d, nevertheless, remains low (i.e., less than 0.137 sec).

Since higher dimensionality d may lead to more skylines, the query cost to refine more candidates on layer 1 of ST is also increasing (as shown in Fig. 14b). Nonetheless, for different dimensionality d, the query time is small (i.e., less than 0.013 sec).



(a) maintenance time **(b)** query time

Fig. 16 The efficiency versus the number, θ, of new objects per time stamp in iDS

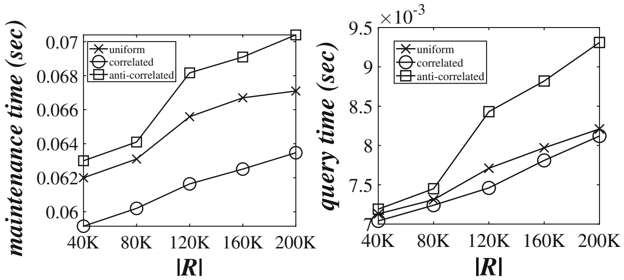


(a) maintenance time **(b)** query time

Fig. 17 The efficiency versus the number, $|W_t|$, of valid objects in iDS

The Sky-iDS efficiency versus the missing rate, ξ, of incomplete objects in *iDS* Figure 15 evaluates the Sky-iDS performance with different missing rates, ξ, of incomplete objects in iDS, where $\xi = 0.1$, 0.2, 0.3, 0.4, and 0.5, and default values are used for other parameters. As shown in Fig. 15a, as the increase in ξ, the maintenance time increases linearly for all three data sets. This is reasonable, since more incomplete objects will need more imputation cost. Similarly, in Fig. 15b, when ξ increases, the query time also becomes larger for all three data sets. In particular, in Fig. 15b, the $Correlated$ and $Anti\text{-}correlated$ data sets always need the minimum and maximum query time. This is because, under "the larger, the better" semantics, the $Anti\text{-}correlated$ data usually have more skylines than the $Correlated$ data [7]. Nevertheless, the time costs for both maintenance and query processing are still low (i.e., less than 0.1 sec and 0.0095 sec, respectively).

The Sky-iDS efficiency versus the number, θ, of new objects per time stamp in *iDS* Figure 16 varies the number, θ, of newly arriving objects per time stamp from 10 to 100, where default values are used for other parameters. In Fig. 16a, when θ becomes larger, the maintenance time increases smoothly for all the three data sets. This is because, the skyline tree ST is updated with more new objects per time stamp, which requires more time to impute missing attributes and maintain skyline answers (as discussed in Algorithm 5 of Sect. 5.3.2). Similarly, in Fig. 16b, the query time also increases with more new objects per time stamp (due to



(a) maintenance time **(b)** query time

Fig. 18 The efficiency versus the size, $|R|$, of data repository

higher refinement cost). Nevertheless, both maintenance and query costs remain low (i.e., 0.201$\sim$0.221 *sec* for dynamic maintenance and 0.0251$\sim$0.0275 *sec* for retrieving skyline answers).

The Sky-iDS efficiency versus the number, $|W_t|$, of valid objects in *iDS* Figure 17 shows the Sky-iDS performance with different numbers, $|W_t|$, of valid objects in stream iDS, where $|W_t|$=$5K$, $10K$, $20K$, $40K$, $50K$, and $80K$, and other parameters are set to their default values. For larger $|W_t|$ value, both maintenance and query times increase, but remain low (less than 0.2067 *sec* and 0.01932 *sec*, respectively, even when $|W_t| = 80K$). This is reasonable, with more valid objects in iDS, we need more efforts to maintain ST index with the imputed objects and conduct the refinement over more Sky-iDS candidates.

The Sky-iDS efficiency versus the size, $|R|$, of data repository *R* Figure 18 illustrates the influence of the size, $|R|$, of data repository on the performance of our Sky-iDS approach. From figures, with larger $|R|$, the maintenance time increases smoothly, since more objects in R are included for data imputation. On the other hand, due to more possible imputed attribute values (resulting from larger $|R|$), the query cost to refine Sky-iDS candidates in ST requires more time cost. Nonetheless, both time costs are low (i.e., around 0.0704 *sec* for the maintenance, and 0.00931 *sec* for the query cost, even when $|R| = 200K$). The experimental results indicate the scalability of our Sky-iDS approach against large $|R|$.

We also did experiments on other parameters (e.g., the number, m, of missing attributes). We do not report similar experimental results here. For interested readers, please refer to Appendix 13 in our technical report [43]. In summary, our Sky-iDS approach can achieve robust and efficient performance under various parameter settings.

7 Related work

Stream processing Existing works on data streams studied many query types, including the keyword search [42], top-k query [10,14], join [13,23], aggregate queries [19,55], nearest neighbor queries [6,28], skyline queries [18,30,54], event detection [67], and so on. These works usually assume that the underlying data (e.g., either certain or uncertain) are complete. Thus, the proposed techniques for complete data streams cannot be directly applied to our Sky-iDS problem over incomplete data stream.

Differential dependency Differential dependency (DD) [47] is a useful tool for data imputation [51], data cleaning [41,46], data repairing [24,48,49,58,59], and so on. Song et al. [50,51] used DD to impute the missing attributes via extensive similarity neighbors with the same determinant attributes. Prokoshyna et al. [41] detected records violating DD rules and cleaned those inconsistent records. Song et al. [46] cleaned the dirty time stamps in data stream based on temporal constraints. Moreover, DD can be also used for constraint-based data repairs over texts [24], events [58,59], and graphs [48].

Many existing works on imputation methods, such as editing rule [20], multiple imputation [44], smoothing-based imputation method [26], constraint-based imputation method [64], or regression-based imputation approach [62], usually impute data based on incomplete data themselves only. However, for sparse (incomplete) data sets (i.e., with many missing attributes), it is rather difficult to accurately and unbiasedly impute data attributes. For example, the supervised imputation approaches (e.g., [64]) usually require labelled data, which is not trivial how to online obtain the labelled stream data in the streaming environment. Moreover, the rule-based imputation approaches (e.g., editing rule [20]) usually requires exact matching, and we may not obtain possible candidates for missing values, especially in sparse data set. In contrast, our DD-based imputation approach utilizes an external source, a complete data repository R, for imputing missing attributes from incomplete data stream, which can avoid lacking of (unbiased) samples, tolerate differential differences between attribute values, and does not require any labelled data. Thus, our DD-based imputation approach can achieve unbiased and more accurate data imputation, compared with existing works.

Skyline queries The skyline query was proposed by Borzsony et al. [7]. Afterward, there are many relavant works on skyline and its variants, for example, skyline queries over certain data [3,8,9,16,29,39,45,54,65] and that on uncertain data [18,31,34,40,66].

In the literature, Khalefa et al. [27] re-defined the skyline operator over static incomplete database. In particular, they ignore the missing attributes during the dominance checking between two incomplete objects. Based on this new skyline definition, Gao et al. [21] and Miao et al. [36] further explored a variant of the skyline query, k-skyband query, which obtains those objects that are dominated by at most

k objects in incomplete data set. However, by neglecting incomplete dimensions, the resulting skylines may be biased (compared with skylines on all attributes). For example, given two objects, $o_1 = (2, 4)$ and $o_2 = (1, 9)$, with two dimensions, according to [7], o_1 and o_2 cannot dominate each other. In this scenario, if the first dimension of o_1 is missing, that is, $o_1 = (-, 4)$, based on [27], o_2 dominates o_1 (by neglecting the first missing attribute for dominance checking; the larger, the better), which may lead to biased skyline result (i.e., o_1 is not included).

The previous work [18] directly assumed that objects from data streams are uncertain, thus, skyline queries are directly conducted over uncertain objects. In contrast, we consider skyline queries over incomplete data streams and turn incomplete objects into complete ones via *differential dependencies* (DDs) [47] (rather than ignoring missing attribute for dominance checking), which will result in unbiased skylines with high confidences. Most importantly, our work follows the style of "imputation and query processing at the same time", which is more challenging than conducting skyline queries directly over uncertain objects and cannot borrow previous techniques for skyline computations to solve our Sky-iDS problem.

Stream outlier detection and repair Existing works on stream outlier detection and repair can be classified into two categories, smoothing-based [26] and constraint-based [52,63,64] approaches. Without distinguishing normal data and outlier, [26] modified almost all data values, which may not be the best way to clean (repair) the outlier. To overcome this drawback, Song et al. [52] proposed an approach to detect the outlier values within a sliding window and then updated the outlier values based on a speed constraint s with minimum and maximum speed changes s_{min} and s_{max}), respectively. Zhang et al. [63] refined this speed-constraint approach by detecting and modifying smaller errors by narrowing the speed intervals s via probability distributions of speeds and speed changes. However, [52,63] cannot repair outliers for data sets with consecutive errors between any two sequential data records. To solve this problem, Zhang et al. [64] proposed a supervised approach based on some labelled data on data stream. Note that, the constraint-based approaches [52,63,64] detected outliers with speed change beyond the acceptable speed constraint s, which have different semantics from the skyline operator in this paper (i.e., skylines are records with maximum values on at least one attributes among all data within a sliding window). Nevertheless, in our experiments, we implemented a baseline method based on [63] and compared our imputation method with [63]. Specifically, [64] cannot be used as the imputation method for our Sky-iDS problem, since it is not trivial how to online obtain the labelled stream data in the streaming environment.

Since these works [26,52,63,64] focus on detecting the outlier values with the high (abnormal) change rates (speeds) w.r.t. the near normal values, they cannot be applied to solve our Sky-iDS problem, which retrieves data objects not dominated by other objects in a sliding window.

Incomplete data management There are some previous works on incomplete data management, for example, how to model incomplete data [2,32], how to index incomplete data [38], and so on. Miao et al. [37] did a comprehensive survey about incomplete data management. In order to obtain complete data, some studies imputed the missing attributes by applying rule-based (exact matching over all dimensions) [20], statistical-based (exact matching over partial dimensions) [35], filter-based [57], pattern-based [60], or analysis-based [44] imputation methods. For example, [60] imputed the missing attributes in streams by finding the k most similar patterns from l time series. However, if the same attributes from l time series are all missing, then this method cannot accomplish the imputation. Royston [44] is to create multiple complete (imputed) versions of data sets and combine all these versions to impute the missing attributes. However, these generated data versions may introduce many erroneous imputed values, which may not be able to provide a stable imputation result. For [20,35,57], although they can achieve explicit imputation results, they may not successfully impute the missing data, due to the sparseness of data sets [47]. In contrast, in this paper, we use DDs [47] and a complete data repository R to impute the missing attributes.

To our best knowledge, no prior works studied the problem of conducting data imputation (via DDs) and skyline query answering, at the same time, on incomplete data in the streaming environment.

8 Conclusions

In this paper, we study an important problem, Sky-iDS, of monitoring the skylines over incomplete data stream, which is useful in many real-world applications such as sensory data monitoring. In order to efficiently impute the missing attributes and conduct Sky-iDS queries, we propose effective data synopses and *skyline tree* (ST) indexes to facilitate the data imputation via *differential dependency* (DD) rules and skyline computations, , at the same time. We also design effective pruning strategies to greatly reduce the Sky-iDS search space over the stream and propose efficient Sky-iDS algorithms to perform "imputation and query processing at the same time" over incomplete data stream. Extensive experiments have demonstrated the efficiency and effectiveness of our proposed Sky-iDS processing approaches on both real and synthetic data sets under different parameter settings.

Acknowledgements Xiang Lian is supported by NSF OAC No. 1739491 and Lian Startup No. 220981, Kent State University. We thank the anonymous reviewers for the useful suggestions.

Appendix

9 Proofs of Lemmas for pruning strategies

9.1 Proof of Lemma 1

Proof As shown in Fig. 3a, since $o'^p.min$ is the minimum corner of the imputed object o'^p, it holds that imputed samples of o'^p is dominating $o'^p.min$, that is, $o'^p \preccurlyeq o'^p.min$. Similarly, we also have $o_i^p.max \preccurlyeq o_i^p$. Due to lemma assumption that $o'^p.min \prec o_i^p.max$, by dominance transition, we can derive $o'^p \preccurlyeq o'^p.min \prec o_i^p.max \preccurlyeq o_i^p$. Thus, we have $Pr\{o'^p \prec o_i^p\} = 1$ (or $Pr\{o'^p \prec o_{il}\} = 1$ for any instance $o_{il} \in o_i^p$). According to Eq. (4), it holds that $P_{Sky\text{-}iDS}(o_i^p) = 0$. Moreover, since $o'.exp \geq o_i.exp$ holds (i.e., object o' expires after o_i^p from lemma assumption), it indicates that o_i^p can never be the skyline due to the existence of object o'^p. Hence, object $o_i^p \in iDS$ can be safely pruned, which completes the proof. □

9.2 Proof of Lemma 2

Proof From Eq. (4), we can derive a probability upper bound as follows.

$$P_{Sky\text{-}iDS}(o_i^p) \leq \sum_{\forall o_{il} \in o_i^p} o_{il}.p \cdot (1 - Pr\{o'^p \prec o_{il}\})$$
$$= 1 - \sum_{\forall o_{il} \in o_i^p} o_{il}.p \cdot Pr\{o'^p \prec o_{il}\}. \quad (6)$$

Since $o_i^p.max \preccurlyeq o_{il}$ ($o_{il} \in o_i^p$) and $Pr\{o'^p \prec o_i^p.max\} \geq 1-\alpha$ hold, we have $Pr\{o'^p \prec o_{il}\} \geq Pr\{o'^p \prec o_i^p.max\} \geq 1-\alpha$. By substituting this probability into Eq. (6), we can obtain: $P_{Sky\text{-}iDS}(o_i^p) \leq 1 - \sum_{\forall o_{il} \in o_i^p} o_{il}.p \cdot (1-\alpha) = \alpha$. Moreover, since $o'.exp \geq o_i.exp$ holds, o_i^p always has the skyline probability less than α during its lifetime, due to the existence of object o'. Thus, object o_i can be safely pruned. □

9.3 Proof of Lemma 3

Proof Similar to the proof of Lemma 2, since $o'^p \preccurlyeq o'^p.min$ and $Pr\{o'^p.min \prec o_i^p\} \geq 1-\alpha$ hold, we have $Pr\{o'^p \prec o_i^p\} \geq Pr\{o'^p.min \prec o_i^p\} \geq 1-\alpha$. By substituting this probability into Eq. (6), we can obtain: $P_{Sky\text{-}iDS}(o_i^p) \leq 1 - Pr\{o'^p \prec o_i^p\} = \alpha$. Thus, since object o_i expires before object o' (i.e., $o'.exp \geq o_i.exp$), object o_i always has the skyline probability lower than α during its lifetime. Hence, object o_i can be safely pruned. □

10 Proofs of properties for skyline tree ST

10.1 Proof of Property 1 of ST

Proof We can prove this property by showing that no such an imputed object o_i^p exists, where o_i^p is a valid object not within skyline tree ST but is actually a skyline or may become a skyline later.

First, assume that the object o_i^p is a current skyline. According to Definition 6, we can obtain $P_{Sky\text{-}iDS}(o_i^p) > \alpha$. By substituting this probability into Eq. (6), we have $\sum_{\forall o_{il} \in o_i^p} o_{il}.p \cdot Pr\{n^p \prec o_{il}\} < 1-\alpha$, that is, $Pr\{n^p \prec o_i^p\} < 1-\alpha$. Thus, no object t^p in ST dominates o_i^p with probability not smaller than $(1-\alpha)$, and then object o_i^p should be on the first layer of ST.

Second, assume that the object o_i^p is dominated by some objects $n^p \in ST$, and may become the skyline after these objects n^p expire (i.e., $n^p.exp < o_i^p.exp$). In this case, object n^p should be the child of one of these objects n^p, since $Pr\{n^p \prec o_i^p\} \geq 1-\alpha$ and $n^p.exp < o_i^p.exp$. Therefore, the ST index contains all the objects $o_i^p \in pDS$ that have the chance to be skylines before they expire. □

10.2 Proof of Property 2 of ST

Proof Given an imputed object $o_i^p \in ST$, if it is not on the first layer of ST, o_i^p will be dominated by its non-empty parent node (object) $n^p \in ST$ with probability $Pr\{n^p \prec o_i^p\} \geq 1-\alpha$. By substituting this probability into Eq. (6), we can obtain $P_{Sky\text{-}iDS}(o_i^p) \leq 1 - \sum_{\forall o_{il} \in o_i^p} o_{il}.p \cdot Pr\{n^p \prec o_{il}\} = 1 - Pr\{n^p \prec o_{il}\} \leq \alpha$, that is, $P_{Sky\text{-}iDS}(o_i^p) \leq \alpha$, which violates the Sky-iDS definition in Definition 6. Hence, object o_i^p cannot become a skyline before its parent node expires from stream iDS. □

10.3 Proof of Property 3 of ST

Proof According to *Property 2*, we can get objects n^p not on the first layer all have the skyline probabilities not bigger than α ($P_{Sky\text{-}iDS}(n^p) \leq \alpha$). So, current skyline objects must be all on the first layer of ST, in other words, the set of objects on the first layer of ST is a superset of Sky-iDS answers. □

References

1. Aberer, K., Hauswirth, M., Salehi, A.: Infrastructure for data processing in large-scale interconnected sensor networks. In: MDM (2007)

2. Antova, L., Koch, C., Olteanu, D.: From complete to incomplete information and back. In: SIGMOD (2007)
3. Awasthi, A., Bhattacharya, A., Gupta, S., Singh, U.: K-dominant skyline join queries: extending the join paradigm to k-dominant skylines. In: ICDE (2017)
4. Beckmann, N., Kriegel, H., Schneider, R., Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. In: SIGMOD (1990)
5. Berchtold, S., Keim, D., Kriegel, H.: The x-tree: an index structure for high-dimensional data. In: VLDB (1996)
6. Bohm, C., Ooi, B.C., Plant, C., Yan, Y.: Efficiently processing continuous k-nn queries on data streams. In: ICDE (2007)
7. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE (2001)
8. Bousnina, F., Elmi, S., Chebbah, M., Tobji, M., HadjAli, A., Yaghlane, B.: Skyline operator over tripadvisor reviews within the belief functions framework. In: ICDE (2017)
9. Chan, C., Jagadish, H.V., Tan, K., Tung, A., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: SIGMOD (2006)
10. Choudhury, F.M., Bao, Z., Culpepper, J.S., Sellis, T.: Monitoring the top-m rank aggregation of spatial objects in streaming queries. In: ICDE (2017)
11. Cranor, C., Johnson, T., Spataschek, O., Shkapenyuk, V.: Gigascope: a stream database for network applications. In: SIGMOD (2003)
12. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB (2007)
13. Das, A., Gehrke, J., Riedewald, M.: Approximate join processing over data streams. In: SIGMOD (2003)
14. Das, G., Gunopulos, D., Koudas, N., Sarkas, N.: Ad-hoc top-k query answering for data streams. In: VLDB (2007)
15. Das Sarma, A., Lall, A., Nanongkai, D., Xu, J.: Randomized multi-pass streaming skyline algorithms. In: VLDB (2009)
16. Dellis, E., Seeger, B.: Efficient computation of reverse skyline queries. In: VLDB (2007)
17. Dhanabal, L., Shantharajah, S.P.: A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. In: IJARCCE (2015)
18. Ding, X., Lian, X., Chen, L., Jin, H.: Continuous monitoring of skylines over uncertain data streams. Inf. Sci. **184**, 196–214 (2012)
19. Dobra, A., Garofalakis, M., Gehrke, J., Rastogi, R.: Processing complex aggregate queries over data streams. In: SIGMOD (2002)
20. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. In: VLDB (2010)
21. Gao, Y., Miao, X., Cui, H., Chen, G., Li, Q.: Processing k-skyband, constrained skyline, and group-by skyline queries on incomplete data. Expert Syst. Appl. **41**, 4959–4974 (2014)
22. Golab, L., Özsu, T.: Issues in data stream management. In: ACM SIGMOD Record (2003)
23. Hammad, M.A., Aref, W.G., Elmagarmid, A.K.: Query processing of multi-way stream window joins. In: VLDB (2008)
24. Hao, S., Tang, N., Li, G., He, J., Ta, N., Feng, J.: A novel cost-based model for data repairing. In: ICDE. IEEE (2017)
25. Igbe, O., Darwish, I., Saadawi, T.: Distributed network intrusion detection systems: an artificial immune system approach. In: CHASE. IEEE (2016)
26. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: ICDE (2001)
27. Khalefa, M., Mokbel, M., Levandoski, J.: Skyline query processing for incomplete data. In: ICDE (2008)
28. Koudas, N., Ooi, B.C., Tan, K., Zhang, R.: Approximate nn queries on streams with guaranteed error/performance bounds. In: VLDB (2004)
29. Lee, J., Hwang, S.: Toward efficient multidimensional subspace skyline computation. In: VLDB (2014)
30. Li, X., Wang, Y., Li, X., Wang, Y.: Parallelizing skyline queries over uncertain data streams with sliding window partitioning and grid index. In: KAIS (2014)
31. Lian, X., Chen, L.: Monochromatic and bichromatic reverse skyline search over uncertain databases. In: SIGMOD (2008)
32. Libkin, L.: Incomplete information and certain answers in general data models. In: PODS (2011)
33. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the sky: efficient skyline computation over sliding windows. In: ICDE (2005)
34. Liu, M., Tang, S.: An effective probabilistic skyline query process on uncertain data streams. In: EUSPN/ICTH (2015)
35. Mayfield, C., Neville, J., Prabhakar, S.: Eracer: a database approach for statistical inference and data cleaning. In: SIGMOD (2010)
36. Miao, X., Gao, Y., Chen, L., Chen, G., Li, Q., Jiang, T.: On efficient k-skyband query processing over incomplete data. In: DASFAA (2013)
37. Miao, X., Gao, Y., Guo, S., Liu, W.: Incomplete data management: a survey. Front. Comput. Sci. **2018**(12), 4–25 (2018)
38. Ooi, B.C., Goh, C.H., Tan, K.: Fast high-dimensional data search in incomplete databases. In: VLDB (1998)
39. Papadias, D., Tao, Y., Fu, G., Seeger, B.: An optimal and progressive algorithm for skyline queries. In: SIGMOD (2003)
40. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: VLDB (2007)
41. Prokoshyna, N., Szlichta, J., Chiang, F., Miller, R.J., Srivastava, D.: Combining quantitative and logical data cleaning. In: PVLDB (2015)
42. Qin, L., Yu, J.X., Chang, L.: Scalable keyword search on large data streams. In: VLDB (2011)
43. Ren, W., Lian, X., Ghazinour, K.: Skyline Queries Over Incomplete Data Streams (Technical Report). arXiv:1909.11224 (2019)
44. Royston, P.: Multiple imputation of missing values. Stata J. **4**, 227–241 (2004)
45. Sarkas, N., Das, G., Koudas, N., Tung, A.: Categorical skylines for streaming data. In: SIGMOD (2008)
46. Song, S., Cao, Y., Wang, J.: Cleaning timestamps with temporal constraints. In: PVLDB (2016)
47. Song, S., Chen, L.: Differential dependencies: Reasoning and discovery. In: TODS (2011)
48. Song, S., Cheng, H., Yu, J.X., Chen, L.: Repairing vertex labels under neighborhood constraints. In: PVLDB (2014)
49. Song, S., Liu, B., Cheng, H., Yu, J.X., Chen, L.: Graph repairing under neighborhood constraints. In: VLDBJ (2017)
50. Song, S., Sun, Y., Zhang, A., Chen, L., Wang, J.: Enriching data imputation under similarity rule constraints. In: TKDE (2018)
51. Song, S., Zhang, A., Chen, L., Wang, J.: Enriching data imputation with extensive similarity neighbors. In: VLDB (2015)
52. Song, S., Zhang, A., Wang, J., Yu, P.S.: Screen: stream data cleaning under speed constraints. In: SIGMOD (2015)
53. Srivastava, J., Cooley, R., Deshpande, M., Tan, P.: Web usage mining: Discovery and applications of usage patterns from web data. In: SIGKDD (2000)
54. Tao, Y., Papadias, D.: Maintaining sliding window skylines on data streams. In: TKDE (2006)
55. Tatbul, N., Zdonik, S.: Window-aware load shedding for aggregation queries over data streams. In: VLDB (2006)
56. Van Buuren, S.: Multiple imputation of discrete and continuous data by fully conditional specification. Stat. Methods Med. Res. **16**, 219–242 (2007)
57. Vijayakumar, N., Plale, B.: Prediction of missing events in sensor data streams using kalman filters. In: sensorKDD (2007)
58. Wang, J., Song, S., Zhu, X., Lin, X.: Efficient recovery of missing events. In: PVLDB (2013)
59. Wang, J., Song, S., Zhu, X., Lin, X., Sun, J.: Efficient recovery of missing events. In: TKDE (2016)

60. Wellenzohn, K., Böhlen, M.H., Dignös, A., Gamper, J., Mitterer, H.: Continuous imputation of missing values in streams of pattern-determining time series. In: EDBT, pp 330–341 (2017). https://doi.org/10.5441/002/edbt.2017.30
61. Xue, W., Luo, Q., Chen, L., Liu, Y.: Contour map matching for event detection in sensor networks. In: SIGMOD (2006)
62. Zhang, A., Song, S., Sun, Y., Wang, J.: Learning individual models for imputation. In: ICDE (2019)
63. Zhang, A., Song, S., Wang, J.: Sequential data cleaning: a statistical approach. In: SIGMOD (2016)
64. Zhang, A., Song, S., Wang, J., Yu, P.S.: Time series data cleaning: from anomaly detection to anomaly repairing. In: VLDB (2017)
65. Zhang, S., Mamoulis, N., Cheung, D.: Scalable skyline computation using object-based space partitioning. In: SIGMOD (2009)
66. Zhang, W., Lin, X., Zhang, Y., Wang, W., Yu, J.X.: Probabilistic skyline operator over sliding windows. In: ICDE (2009)
67. Zhou, X., Chen, L.: Event detection over twitter social media streams. In: VLDB (2014)