sensors

MDPI

Article

Answering the Min-Cost Quality-Aware Query on Multi-Sources in Sensor-Cloud Systems †

Mohan Li, Yanbin Sun, Yu Jiang and Zhihong Tian *

Cyberspace Institute of Advanced Technology, Guangzhou University, Guangzhou 510006, China; limohan@gzhu.edu.cn (M.L.); sunyanbin@gzhu.edu.cn (Y.S.); jiangyu@gzhu.edu.cn (Y.J.)

* Correspondence: tianzhihong@gzhu.edu.cn

† This paper is an extended version of our paper published in SCS 2018.

Received: 4 November 2018; Accepted: 11 December 2018; Published: 18 December 2018

Abstract: In sensor-based systems, the data of an object is often provided by multiple sources. Since the data quality of these sources might be different, when querying the observations, it is necessary to carefully select the sources to make sure that high quality data is accessed. A solution is to perform a quality evaluation in the cloud and select a set of high-quality, low-cost data sources (i.e., sensors or small sensor networks) that can answer queries. This paper studies the problem of min-cost quality-aware query which aims to find high quality results from multi-sources with the minimized cost. The measurement of the query results is provided, and two methods for answering min-cost quality-aware query are proposed. How to get a reasonable parameter setting is also discussed. Experiments on real-life data verify that the proposed techniques are efficient and effective.

Keywords: sensor-based systems; sensor-cloud systems; data quality; quality-aware query; source quality

1. Introduction

The quality of data can severely impact the data-driven applications. It is reported that data error rates of enterprises can be as high as 30% [1]. In some multi-sources applications, such as wireless sensor networks, Internet of Things or data fusion, data quality-aware sensing have been identified as one of the key concerns of sensor-based architecture [2]. In sensor-cloud systems which integrate hybrid networks [3], virtual sensors and their data can be queried by users via user interfaces. However, the quality of data returned by different data sources (i.e., virtual sensors) tends to be different due to different environments, and the quality of the historical data can be used as an index for generating plans of future queries. In some recent architectures, such as fog computing [4–6], edge devices are used to carry out a substantial amount of computation. The differences of computing capability and storage capacity between the devices also result in different quality of the data.

In these systems, a common fact is that there might be multiple sources providing the data of the same object, but the sources vary in data quality. Therefore, it is important to control and evaluate data quality. The quality of data and sources can be evaluated in the cloud based on classic data quality methods (which are not lightweight and therefore may be difficult to perform on sensors). Then, a set of data sources, which can provide relatively high quality data at low cost, will be selected to return the required data. In this way, we do not have to visit or wake up the rest of the data sources, and thus can save resources. Following example demonstrates the case of data sources with different data quality.

Example 1. *Three sources in Figure 1 provide weather conditions observed in the same place at the same time. However, these sources provide different observations of the same object. For instance, Source 1 claims that the temperature is 26 °C, but Source 2 and Source 3 claim that the temperature is 26.7 °C or 16 °C.*

Some relative quality constraints can be used to infer the relative quality of the observations. For example, if we know that the temperature sensor used by Source 2 possibly be a newer model of the sensor of Source 1, we can infer that the value of temperature provided by Source 2 tends to be more accurate than Source 1. Arcs with probability can represent the existence of relative quality constraints. In Figure 1, the arc from (temperature, 26.7 °C) to (temperature, 26 °C) indicates that there may exist a relative quality constraint that "the sensor of Source 2 provides higher quality temperature than the sensor of Source 1", and the probability of the existence of the constraint is 0.9.

According to relative quality constraints shown by Figure 1, the observations of temperature and wind speed of Source 3 might be less accurate than those of Source 1 and Source 2. That is, if we only access the data from Source 3, we might get inaccurate values. In this case, if the user is willing to pay a higher cost, perhaps we can access the other two data sources to obtain higher quality data.

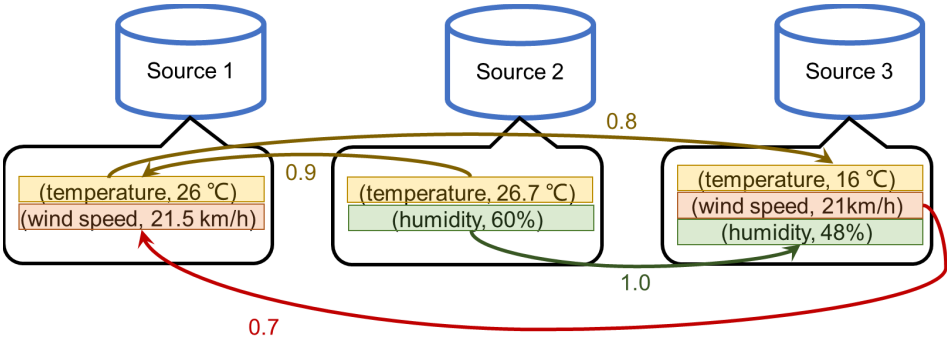


Figure 1. Data sources with different data quality.

When selecting sources for answering a query with relative quality constraints, both low cost and high quality are required. In this paper, we study the min-cost quality-aware query (MQQ for short) answering problem which aims to get high quality results from multi-sources with the minimized cost. This paper is extended based on the conference version of SCS 2018 [7]. The contributions of this paper are as follows.

1. A general definition of MQQ answering problem is provided.
2. The data quality measurement of quality-aware query on multi-sources is defined based on relative data quality constraints. Two methods for solving the min-cost quality-aware query answering problem are proposed.
3. If the users do not have enough knowledge of the data, it is often difficult for them to set the reasonable quality threshold. To deal with this problem, a method is proposed to automatically help the users to find a proper quality threshold.
4. The experiments on real-life data are conducted, which verifies the efficiency and effectiveness of the provided solutions.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 provides the problem definition. Section 4 gives the data quality measurement. Section 5 studies the solutions of MQQ answering problem. Section 6 discusses the strategies of setting a reasonable quality threshold. Section 7 shows the experimental results. Section 8 concludes the paper.

2. Related Work

Data quality. There is currently a lot of work on constraint-based data quality [8–22]. The problem of data quality evaluation is studied in these works. These constraint-based methods can be roughly categorized into two classes.

(1) The first class of methods can directly compute the quality score of a data set. Some constraints, such as functional dependencies, conditional functional dependencies, denial constraints and editing

rules, are used as data quality rules to detect errors in a given dataset [8–11]. Some systems use these kinds of rules to find the dirty data in a given data set [12], and the quality score of a data set can be defined as the proportion of clean data in the data set. Meanwhile, the trustworthy of data sources in data fusion can also be considered as the quality score of data sources [13].

(2) The second class consists of methods for computing relative data quality. The relative quality has been studied for many years, and a lot of rules are proposed to evaluate the relative completeness, accuracy, currency, etc. The relative data quality constraints can be derived based on their methods. For example, the relative accuracy constraints can determine the relative accuracy of different tuples [14]. Inclusion dependencies and conditional inclusion dependencies can determine the relative completeness of data set [15]. The currency rules can be used to determine the relative currency of different tuples or data sources [16,17]. Some techniques for data fusion and truth discovering also study how to derive relative data quality constraints [18–20]. These techniques can find the copy dependencies between different sources, and the copy dependencies can be used to derive the relative currency or completeness constraints of different sources [21,22].

The two classes of methods can either directly compute the absolute quality score of one source or compute the relative quality score between two sources based on the observations. However, they do not take into consideration how to evaluate and guarantee the data quality of query results, and thus is different from our work.

Sensor-cloud. There are currently many studies on the architecture of sensor-cloud [3,23–29]. Alamri A. et al. [3] surveyed the early works of sensor-cloud. Madria S. et al. [23] proposed a new cloud of sensor (CoS) which was composed of virtual sensors built on top of physical wireless sensors rather than the device of traditional cloud. Based on the CoS, Olympus [24] is proposed by combining an information fusion and CoS-based decentralized WSAN virtualization model. Application decision processes are performed partly within physical sensors and partly within the cloud. Fazio M. et al. [25] proposed a cloud framework called Cloud4sens, which combines both data-centric and device-centric models to meet different needs of cloud clients. To improve the performance of cloud-based sensor system, Lyu Y. et al. [26] proposed a periodic-task based high-performance scheduling model with a cloud-supported caching mechanism for the multisensor gateway. Abdelwahab S. et al. [27] proposed a global architecture of Cloud of Things, and adopted a distributed sensing resource discovery and virtualization algorithms to deploy virtual sensor networks on top of a subset of the selected IoT devices. Zhu C. [28] proposed a multi-method data delivery scheme for green sensor-cloud among WSNs, cloud and users to achieve low delivery cost and less delivery time. Dinh T. et al. [29] proposed a location-based interactive model of IoT-cloud for mobile cloud computing applications, and the model provides sensing services on demand based on interests and locations of mobile users.

A common feature of these sensor-cloud architectures is that they combine some of the hybrid sensor networks to get a virtual sensor network, while using cloud services to handle the interaction of the virtual sensor network and end users. This ensures that the underlying hybrid sensor networks are transparent to the end users. The evaluation of data quality will consume a lot of computing resources (some evaluation problems require square or higher time overhead, or even be NP-hard problems), so it is difficult to be taken on the sensor side, and we need to deploy it in the cloud. There are three benefits. First, we can utilize the computing power of the cloud to quickly complete the evaluation of data quality. Second, the deploying it in the cloud can make the quality evaluation independent of the specific architecture of the underlying virtual sensor network. Third, since we will select better sensors to answer user queries after the evaluation of data quality, we can also save computing resources and power on the sensor side.

Quality-aware sensing. There are a lot of works to ensure some extra properties when executing queries in data-driven and sensor-based systems, such as security [30–34] or trustiness [35–37]. Quality-aware query has also been studied in some data-driven systems [38–40]. Lian X. et al. [38] formalized the RDF data by inconsistent probabilistic RDF graphs, and studied the quality-aware subgraph matching problem. Yeganeh N. K. et al. [39] provided a data quality-aware system

architecture which supports data quality profiling and data quality-aware SQL. The sample conditional data profile generated by the system can be used as a type of quality constraints in our work, but the work does not focus on how to get a best query result with cost and quality requirements. Wu H. et al. [40] studied the quality-aware query scheduling in wireless sensor networks. The provided framework can determine the target quality of each query. The work can be used as the basis for this study, but it does not consider optimization problems when there are multiple query results, and it also does not consider how to do quality-aware query in sensor-cloud systems.

3. A General Definition of MQQ Answering Problem

3.1. Preliminaries

First we provide some basic concepts which will be referred in the following sections.

Data sources. Let $\mathbb{S} = \{S_1, ..., S_n\}$ be a set of data sources, where $S_i = \{\phi_1, ..., \phi_{m_i}\}$ is the ith data source consisting of m_i observations. An observation $\phi = (s, o, v)$ is a triple consisting of a source s, an object o and a corresponding value v. For instance, $(S_2, humidity, 60\%)$ indicates that Source S_2 observes that the humidity is 60%. All the observations in a same source is with the same s.

Data quality constraints. Quality function hq is defined to represent data quality constraints. $hq(\phi, \phi') = p$ means that the quality of ϕ probably be *higher* than the quality of ϕ', and the existence probability of this constraint is p. For example, $hq(\phi, \phi') = 0.9$ means that the data quality of ϕ is higher than ϕ' with the probability 0.9. The quality constraints can be considered as the relative data accuracy or other relative data quality measurements [14–17]. These constraints can be used to evaluate the quality of the observations and sources. How to find and leverage these constraints will be addressed in the next section.

Access cost. Each data source $S_i \in \mathbb{S}$ has a access cost $cost(S_i) \in R^+$, indicating the cost of querying the data provided by S_i. High data quality sometimes means high access cost, thus MQQ is defined to try to balance the cost and quality.

3.2. Definition of MQQ Answering Problem

Quality-aware query. A quality-aware query Q is a query with data quality requirements. Q is in the form of $Q = (O_Q, \theta)$, which is a pair of an object set O_Q and a quality threshold θ. O_Q consists of the objects queried by Q. θ is a lower bound, which represents the requirement that the quality of the returned observations must be no less than θ.

MQQ answering problem. A MQQ aims to find the observations of O_Q from sources in $\mathbb{S}$ which satisfying data quality lower bound and with minimized access cost. Formally, the MQQ answering problem can be defined as follows.

Input: a quality-aware query $Q = (O_Q, \theta)$,

Output: a observation set Φ which satisfies the following conditions:

1. For each $o \in O_Q$, $\exists \phi \in \Phi$ such that ϕ is the observation of o, that is, Φ contains the observations of all queried objects,
2. Φ can satisfy the quality lower bound θ,
3. $\nexists \Phi'$ returned by $\mathbb{S}_{\Phi'} \subseteq \mathbb{S}$ such that Φ' satisfies condition (1) and (2), and $\sum_{S \in \mathbb{S}_{\Phi'}} cost(S) < \sum_{S \in \mathbb{S}_{\Phi}} cost(S)$.

As of now, we have not discussed how to calculate the data quality of query results. In the following sections we will first consider how to calculate the data quality of a set of observations, and then present the analysis of MQQ answering problem.

4. Measuring Data Quality Based on Quality Constraints

Ideally, each data source corresponds to a certain quality score (sometimes the score is called absolute quality score), then we can use certain quality scores to directly resolve subsequent

quality-aware query problems. However, in many cases, we can only get the relative quality rule of two data items [14–17]. For example, an accuracy rule $\psi = \forall t_1, t_2 \ \ \omega \rightarrow t_i \prec_{A_i} t_j$ can enable us to determine the relative accuracy of tuple t_i and t_j on attribute A_i. A uncertain currency rule $\psi = \forall t_1, t_2 \ \ (\omega \rightarrow t_i \prec_{A_i} t_j, cer(\psi))$ can provide the possibility that observations provided by one source will become obsolete earlier than another source. The semantic of a quality rule is that if the condition ω on the left-hand side is satisfied, the inequality of $t_i[A]$ and $t_j[A]$ on the right-hand side is established (with a probability or certainty). $t_i[A]$ and $t_j[A]$ are the values (i.e., observations) of two tuples on object A, thus the inequality of right-hand side indicate the relative data quality. Base on the rules, if two tuples satisfy ω, $hq(t_i[A], t_j[A]) = p$ can be derived from the right-hand side, where $t_i[A]$ and $t_j[A]$ correspond to the ϕ and ϕ' in the definition of data quality constraint in Section 3.1.

In practice, quality rules are either discovered from a training data set [41,42], or written by the domain experts. For example, the owner of the sensor network may provide the knowledge that "*model A is a newest model of the temperature sensor used in my network, which can provide more accurate data than other models in 90% of cases*". Based on the knowledge, if we have a set of temperature data $\{26, 27, 28\}$, and we find that 27 is provided by a sensor of *Model A*, 26 and 28 is provided by sensors of *Model B*, we will know that $hq((Model A, temperature, 27), (Model B, temperature, 26)) = 0.9$.

To deal with the relative quality constraints, we propose a data source quality score based on the semantic of possible worlds.

4.1. Quality Graph

As defined before, quality constraints are represented by the quality function hq. $hq(\phi, \phi') = p$ indicates that the quality constraint exist with probability p. Thus, uncertain graphs [43,44] can be used to model the constraints, that is, observations are the nodes in the graph, and relative quality constraints are the weighted arcs.

Definition 1 (quality graph)**.** *Given an object o, the observation set Φ_o containing all the observations corresponding to o, and the data quality function corresponding to Φ_o, the quality graph $G_o = (V, A)$ is defined as follows.*

1. *Each node $v_\phi \in V$ corresponds to an observation in Φ_o.*
2. *For each pair (ϕ, ϕ'), if $hq(\phi, \phi') \neq null$, there exists an arc from ϕ to ϕ' in A, the weight of the arc (ϕ, ϕ') is $hq(\phi, \phi')$.*

In the definition, $hq(\phi, \phi') = null$ means that hq is not defined at (ϕ, ϕ'). Figure 2 is the quality graph of the object "temperature" in Figure 1. For the arcs in G_o, the semantic of the arc weight is the probability of the arc's existence. For example, if the weight of an arc a is 0.9, the probability of a's existence is 0.9, and the probability of non-existence is $1 - 0.9 = 0.1$. Therefore, possible worlds [45] can be used to describe different situations of data quality. Figure 3 shows the the four possible worlds corresponding to Figure 2, and the probabilities of each possible worldc are 0.02, 0.18, 0.08, and 0.72, respectively.

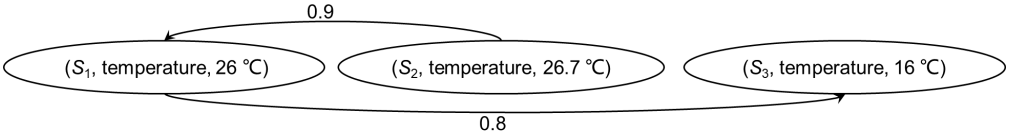


Figure 2. The direct graph of the observations of temperature in Figure 1.

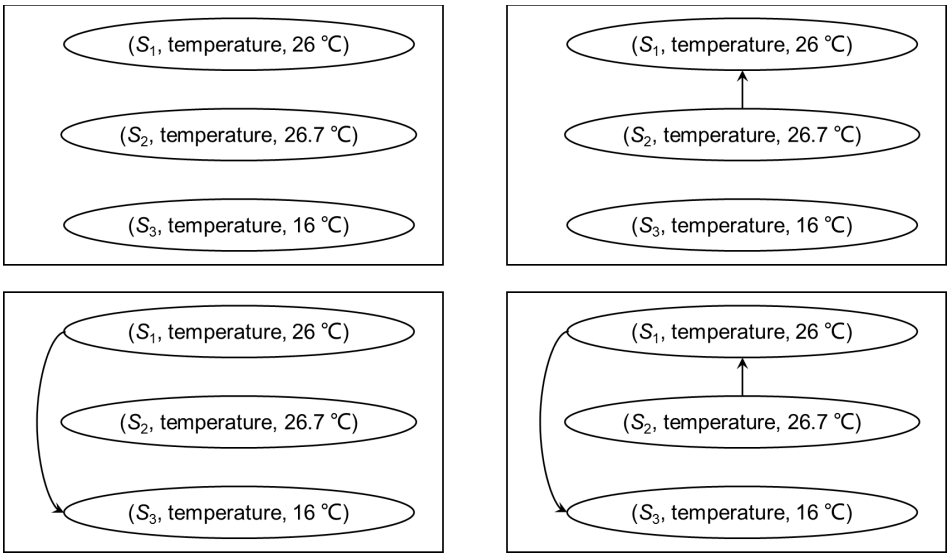


Figure 3. The possible worlds of a quality graph.

4.2. Data Quality Score

The data quality score of an observation ϕ of the object o_ϕ is defined as the probability that the quality of ϕ is NOT lower than any other observations of o_ϕ. We use G_{o_ϕ} to denote the quality graph of o_ϕ. In any possible world, the quality of ϕ is higher than the quality of ϕ' if there exists an arc from ϕ to ϕ'. Therefore, the quality score of ϕ is the probability that the in-degree of v_ϕ equals to 0, where v_ϕ is the node corresponding to ϕ.

Definition 2 (quality score of an observation)**.** *Let W be the set of all the possible worlds corresponding to G_{o_ϕ}. $dq(\phi) = \sum_{w \in W} P_w \times f(w)$ is the quality score of Φ, where $f(w) = 1$ if node v_ϕ has an in-degree of 0 in w, otherwise $f(w) = 0$.*

According to the semantics of the arcs in the quality graph, a naive idea to calculate the quality score of a node is to enumerate all possible worlds and check in which possible worlds the node's in-degree is 0. However, if we assume that the existence of the arcs are independent of each other, the quality score of ϕ is the product of the non-existence probabilities of all the in-arcs of ϕ, that is, $dq(\phi) = \prod_{a \in in(v_\phi)} (1 - weight(a))$, where $in(v_\phi)$ is the in-arc set of v_ϕ, and $weight(a)$ is the weight of arc a in G_{o_ϕ}. Furthermore, we assume the observations are independent of each other, the quality of an observation set Φ can be similarly defined.

Definition 3 (quality score of observation set)**.** *Given an observation set Φ, the quality of Φ is the probability that for each observation $\phi \in \Phi$, the quality of ϕ is not lower than any other observations, that is,*

$$dq(\Phi) = \prod_{\phi \in \Phi} dq(\phi). \tag{1}$$

Given a quality-aware query $Q = (O_Q, \theta)$, the query result of Q is a set of observations. Therefore, the data quality of the query results is the same as the quality of the observation set. More precisely, condition (2) (i.e., "Φ can satisfy the quality lower bound θ") in the definition of MQQ answering problem can be rewritten by "$dq(\Phi) \geq \theta$".

5. Methods for Solving MQQ Answering Problem

As is defined in Section 3, if Φ is a valid output of MQQ answering problem, Φ should satisfy three conditions, i.e., (1) Φ covers all the objects in O_Q, (2) $dq(\Phi)$ is no less than θ, and (3) the access

cost of finding Φ is minimized. Please note that if condition (2) is not considered, MQQ answering the problem is an equal problem of "weighted set cover" which is a NP-hard problem [46].

In this section, we first provide a search and prune method to accurately solve the MQQ answering problem, then provide an approximate method to try to get a feasible (but not necessarily optimal) solution quickly.

5.1. The Search and Prune Method

The main idea is that we first find the observation set with highest quality score, then we continually replace one observation in current set by an unused observation of the same object with a lower or equal quality score, to see if the new set after replacement can satisfy the quality threshold at a lower cost. The replacement step is repeated until the quality threshold cannot be satisfied or there is no less-cost solution.

Figure 4 shows the search tree of this process. Each node in the tree represents an observation set, and each child corresponds to a substitution of its parent. Since we replace exactly one observation each time, for each node $\Phi = \{\phi_1, ..., \phi_k\}$ in the search tree, if $\Phi' = \{\phi'_1, ..., \phi'_k\}$ is the child of Φ, Φ' has exactly one observation different from Φ. That is, $\exists j$ s.t. $\phi_j \neq \phi'_j$, and ϕ_i is the same as ϕ'_i for any other $i \neq j$. Moreover, we have $dq(\phi_j) \geq dq(\phi'_j)$.

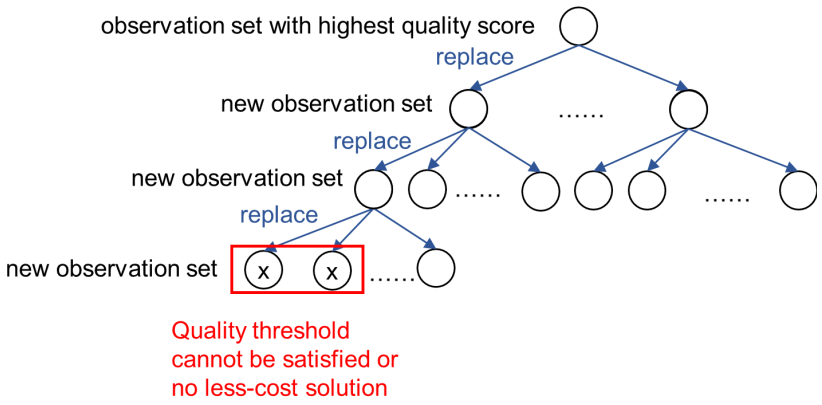


Figure 4. The main idea of the search and prune method for MQQ answering.

To ensure that the search can finally enumerate all possible solutions, we sort all the observations associated with the same object based on their quality scores. The current observation is hereby replaced by the next observation in the sorted sequence to generate a new observation set in the search tree. For each node Φ in the search tree, Φ may have 0 to $|O_Q|$ child nodes. The total number of nodes in the full search tree is exponential, thus we need an effective pruning strategy to speed up the process.

Prune strategy. Since we always replace one observation by another observation with an equal or lower quality score, the quality score of a parent node is always not lower than its children. Theorem 1 formalizes this conclusion.

Theorem 1. *For any node Φ and its child Φ' in the search tree, $dq(\Phi) \geq dq(\Phi')$.*

Proof. Let $\Phi = \{\phi_1, ..., \phi_k\}$ and $\Phi' = \{\phi'_1, ..., \phi'_k\}$. As we have discussed before, $\exists j$ s.t. $\phi_j \neq \phi'_j$, $dq(\phi_j) \geq dq(\phi'_j)$, and for any $i \neq j$, ϕ_i is the same as ϕ'_i. According to the definition of quality score of observation set, $dq(\Phi) = dq(\phi_j) \times \prod_{i \neq j}(dq(\phi_i))$, and $dq(\Phi') = dq(\phi'_j) \times \prod_{i \neq j}(dq(\phi'_i))$. Therefore, $dq(\Phi) \geq dq(\Phi')$. □

For any node Φ and its child Φ', $dq(\Phi') \leq \theta$ if $dq(\Phi) \leq \theta$. Thus, if the quality threshold θ cannot be satisfied by Φ, it also cannot be satisfied by any descendant node of Φ. Thus the subtree rooted at Φ can be pruned off.

Let O_Q be the object set queried by Q, the time complexity of the worst case is still exponential, i.e., $O(|O_Q|^{|\mathbb{S}|})$. It is still unbearable in the condition of high efficiency requirement. Therefore, we propose an approximate search strategy which can quickly provide a feasible solution when the user is willing to relax the requirements for the cost.

5.2. Approximate Search

When a specified data source is asked to return multiple observations, the access cost only needs to be paid once. Thus, a replacement that adds data sources necessarily reduces the quality and increases the cost at the same time. For node Φ and its child Φ', we have Theorem 2, where $\mathbb{S}_\Phi$ and $\mathbb{S}_{\Phi'}$ are the source sets returning Φ and Φ' respectively, $cost(\Phi) = \sum_{S \in \mathbb{S}_\Phi} cost(S)$ is the cost of Φ.

Theorem 2. *$cost(\Phi) \leq cost(\Phi')$ and $dq(\Phi) \geq dq(\Phi')$ if $S \notin \mathbb{S}_\Phi$ and $\mathbb{S}_{\Phi'} = \mathbb{S}_\Phi \cup \{S\}$.*

Proof. According to Theorem 1, $dq(\Phi) \geq dq(\Phi')$. Since $\mathbb{S}_{\Phi'} = \mathbb{S}_\Phi \cup \{S\}$, $cost(\Phi') = \sum_{S' \in \mathbb{S}} cost(S') + cost(S) = cost(\Phi) + cost(S)$, we have $cost(\Phi) \leq cost(\Phi')$. □

According to Theorem 2, an idea to quickly get a feasible solution is to drop the substitution that add new data sources. That is, for any node Φ, only the observations returned by $\mathbb{S}_\Phi$ are used to do the replacement. More precisely, for any node Φ and its child Φ', we stipulate that a replacement should ensure either $\mathbb{S}_{\Phi'} = \mathbb{S}_\Phi$ or $\mathbb{S}_{\Phi'} = \mathbb{S}_\Phi \backslash \{S\}$, where $S \in \mathbb{S}_\Phi$. The search is terminated when the quality threshold cannot be satisfied or there is no less-cost solution.

It is easy to observe that this search strategy only considers the subset of the data sources which provide the observations with the best quality score. The solution returned by this strategy necessarily satisfies the conditions (1) and (2) in the definition of MQQ answering problem, but does not necessarily satisfy the condition (3) (i.e., the cost may not be the lowest).

5.3. Discussions

Please note that the above search methods do not have much restrictions on the definition of data quality score. The only restriction is that the quality score should not reduce after replacement. More concretely, our data quality definition for the observation set is in a multiplicative form, but other forms (e.g., average or maximum) of data quality metrics which guarantee the quality score not reduce after replacement, are applicable to the search strategy in this section. If there are any other reasonable quality metrics which may not based on the semantic of possible worlds but can ensure that the quality score do not reduce after replacement, our search method can also be applied to these quality metrics with a few modifications.

When implementing the above two strategies, the efficiency can be further optimized in two aspects.

1. For the two search strategies, it is very important to efficiently get the next data source to do the replacement. To this end, $|O|$ ordered lists can be maintained, and the ith ordered list stores the high-to-low ordering of the sources that can provide object o_i. A element of the ordered list is a quad of the form $(S, \phi, dq(\phi), next)$, where $next$ is a pointer to the next quad in the ordered list. Thus, during the search, we can use $dq(\phi)$ to calculate the quality score of the current search tree node, and use $next$ to get the next data source that can be used to do the replacement in constant time.
2. Different branches of the search tree may have the same node. In implementation, we can first check to see if the current node already has a duplicate that has been processed. The node is

discarded if the answer is yes. This operation can avoid repeating process and can save a lot of time.

6. A Method for Selecting the Quality Threshold

When writing a MQQ $Q = (O_Q, \theta)$, if the end user does not have much knowledge about the data, it might be difficult to set a proper threshold θ. Therefore, we propose a method that can automatically determine the reasonability of a threshold, and can help the user to find a proper threshold.

6.1. Determining the Reasonability of a Threshold

There are two cases that the quality threshold is not set properly.

1. The quality threshold is set too high such that no observation set can satisfy the threshold.
2. The quality threshold is set too low, which is likely to cause large search space, and to lead to unendurable query latency.

Therefore, determining the reasonability of a threshold can be divided into two steps.

6.1.1. Deal with High Threshold

First we need to determine whether the threshold is too high. A quality threshold θ is deemed to be too high if we cannot find any data source which can provide the queried observations and meet the quality threshold. This is a trivial problem since we just need to check whether the root node of the search tree can satisfy θ. As we have discussed in Section 5, the root node corresponds to the highest quality score. Therefore, all the nodes in the search tree cannot satisfy θ if the root node fail to satisfy θ. If the time to construct a search tree node can be considered as constant, then the time complexity to determine whether the threshold value is too high is $O(1)$.

6.1.2. Deal with Low Threshold

If the threshold is not too high, we need to determine whether it is too low. Since a low threshold will make the query to be processed slowly, the user should be reminded to raise the threshold in this case. We found that when using accurate search, the search time may rise by more than twofold when the threshold is decreased by 0.01 in some cases.

To determine whether the threshold is too low is much difficult. The first difficulty is that we cannot precisely define what is "too low". To deal with this difficulty, we can use historical data to estimate the query latency and let the user to determine whether the threshold is too low. The second difficulty is that it is hard to precisely know the query latency without any prior knowledge since the data needed by different queries are also different. However, it is easy to observe that the query delay is mainly affected by the size of search space, thus the estimate of query latency can be converted to the estimate of the search space size.

Estimate of the search space size. We provide a method to estimate the size of search space based on the idea of estimating the layer number of the tree using depth-first search. The process is as follows. We expand the search tree from top to bottom in the way similar to Section 5, but the difference is that we do not generate a full search tree. We only do the replacement in each layer once, that is, generate only one new node in each layer. The process of the expanding stops if the new node cannot meet the quality threshold requirement. The height of the tree at the time point that the process stopped is the estimate of the search tree. For example, if we only generate the left most nodes in the search tree shown by Figure 4, the height of the tree is estimated to be 4 because the node generation of the search tree stops at the fourth layer. Let h denote the real height of the search tree, and $\hat{h}$ denote the estimated height of search tree, then the search space size (i.e., the upper bound of the total number of nodes of the search tree) can be estimated to be $|O_Q|^{\hat{h}-1}$ since each node has at most $|O_Q|$ children.

A correspondence table of search space size and time overhead can be maintained based on the historical data. Alternatively, other models of search space size and query latency also can be built

based on some machine learning methods. Based on the models, the estimate of the search space size can be converted to the estimate of the query latency.

However, the accuracy of the estimate is determined by the accuracy of the estimate of the height of the search tree, and this estimate has a lot to do with the position of the nodes generated for each layer. For instance, in Figure 4, if we generate the rightmost (rather than the leftmost) node in each layer, the estimate of the height of the search tree is at least 5.

Since the accurate search needs to traverse all branches, it is not difficult to observe that we only generate a path in the search tree when doing the estimation. The real height h is the length of the longest path in the search tree, and the estimate of the height must be no more than the real height. Formally, we have

$$\hat{h} \leq h. \tag{2}$$

Based on this conclusion, we can infer that the larger the estimate, the closer it is to the real height. To find a better estimate of the height, we propose two generation strategies for nodes, which are applicable to different scenarios.

Strategy 1 (*FixedPos*). This strategy generates the node in a fixed position each time. Concretely, a location k is selected before the estimation, and the kth data source is replaced to obtain a new child node each time. This strategy applies to situations where pruning is not considered and there is some domain knowledge of the query. For example, when we know in advance that "the kth object in the query has the most replaceable data sources", searching down at position k is more likely (not necessarily) to get a higher search tree than other positions. $\hat{h}$ tends to be closer to h.

Strategy 2 (*RandPos*). This strategy generates the node in a random position each time. When pruning is considered or there is no domain knowledge of the query, if we do not perform an accurate search, we cannot know how to use depth-first search can get the most accurate estimate. Therefore, we can randomly selecte a k_i ($0 < k_i \leq |O_Q|$) for layer i, and the k_ith data source is replaced to obtain a new child node in layer $i+1$. If time permits, we can also perform multiple random strategies and pick the largest (and therefore the most accurate) estimate.

After $\hat{h}$ is obtained, the query latency can be estimated based on the table or model mentioned above. Then, the estimated query latency can be reported to the user, and the user can determine whether the latency is acceptable. If not acceptable, the threshold is marked as too low.

6.2. Finding a Reasonable Threshold

If the threshold is considered to be unreasonable, we can give some suggestions to help the user correct the threshold.

1. If the threshold is considered to be too high, we only need to return the quality score of the root node of the search tree and tell the user that there is no query result higher than this score.
2. If the threshold is considered to be too low, we need to return a set of candidates to guide the user in adjusting the threshold. As stated in Theorem 1, when the search tree is expanded to the lower layer, dq is not increased (i.e., the value of dq only decreases or keeps unchanged). Therefore, in the estimation process, the layer where dq changes can be recorded and converted into a query latency. That is, we ignore those layers whose dq is unchanged and record the layers whose dq decreases. The historical data can be used to learn the possible query latency regarding the number of layers. The recorded can be given to the user as a candidate set to help the user to adjust the threshold setting.

7. Experimental Results

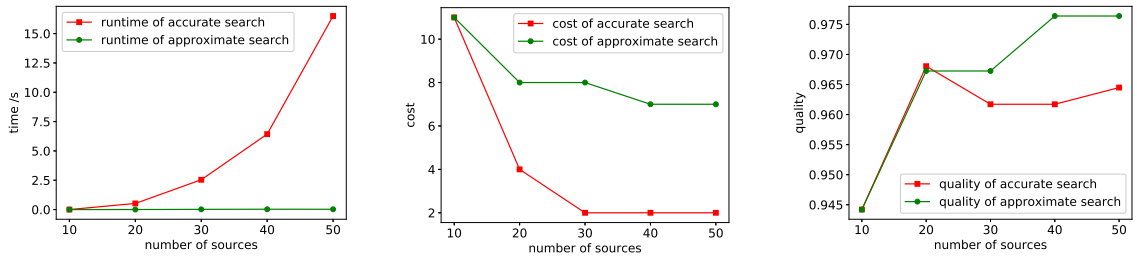
We conduct the experiments on a real-life data set. The codes are written in Python and run on a machine with i5 2.50 GHz Intel CPU and 8 GB of RAM. The dataset consists of air quality data generated by a network of 56 sources located in Krakow, Poland. The dataset is from https://airly.eu/. The available object set is {temperature, humidity, pressure, pm1, pm25, pm10}. Since the data set does

not provide the cost of data sources, we random assign each source a cost. In practical applications, the cost can be the bandwidth, power consumption, or the money of getting data, etc. We first compared the two different search methods (i.e., the accurate search and the approximate search). After that, we studied the performance of the two strategies of search space size estimation.

7.1. Performance of the Search Methods

7.1.1. Varying $|\mathbb{S}|$

A parameter that has a significant impact on efficiency and performance is the number of data sources, i.e., $|\mathbb{S}|$. Figure 5 shows the experimental results of runtime, cost and quality under different $|\mathbb{S}|$. Here we fixed $|O_Q|$ to 4 and the quality threshold to 0.95.



(a) runtime (b) cost (c) quality

Figure 5. Experimental results when varying $|\mathbb{S}|$.

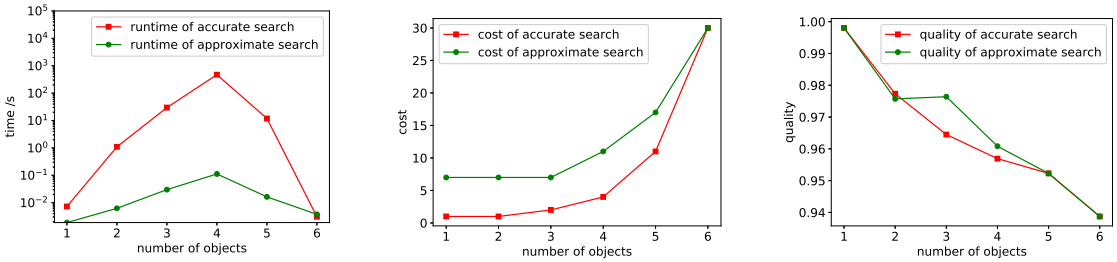
Runtime. We compared the runtime of accurate search and approximate search. The worst case of accurate search is $O(|O_Q|^{|\mathbb{S}|})$. This theoretical analysis is consistent with the experimental results. The efficiency of approximate search is much higher than that of accurate search. When the number of data sources is 10, the time cost of the two is similar, but as the number of data sources increases, the time cost of the accurate search increases exponentially, while the time cost of the approximate search is basically not change much because it controls the choice of the data sources.

Cost. We compared the cost of the result of the query returned by the accurate search and the approximate search. As the number of available data sources increases, the costs of the results of the two methods are both declining. This is because that the more data sources we can access, the more likely we are tantamount to discover data sources with high-quality and low-cost. Overall, the cost of the accurate search results is substantially lower than that of the approximate search, because the accurate search returns the optimal solution.

Quality. The quality of the optimal solution returned by accurate search is usually closer to the quality threshold and therefore lower than the quality of the result of the approximate search. Please note that when the data source is 10, there is no result that satisfies the quality threshold. To comprehensively compare the two methods, when implementing the two methods, we require that both methods return the highest quality result in this case. At this time, the results of the two methods are the same, and the quality score is 0.945.

7.1.2. Varying $|O_Q|$

When varying $|O_Q|$, the relative merits of the two methods in different aspects are the same as the situation of changing $|\mathbb{S}|$, but the trends of runtime, cost and quality are different. We fixed $|\mathbb{S}|$ to 56 (i.e., we use all the available data sources) and the quality threshold to 0.95. We change $|O_Q|$ from 1 to 6. Figure 6 shows the experimental results.



(a) runtime (b) cost (c) quality

Figure 6. Experimental results when varying $|O_Q|$.

Runtime. Since the runtime of the accurate search varies greatly, the vertical axis of Figure 6a uses a logarithmic scale. As can be seen from the results that the time cost of accurate search is much higher than the approximate search. As $|O_Q|$ increases, the runtime increases first and then decreases. This is because when $|O_Q|$ is large enough, the result that satisfies the quality threshold is gradually reduced (the quality score is the probability multiplication, thus more observations will make the probability smaller). Based on the pruning strategy of Theorem 1, the search space is also gradually reduced. When $|O_Q| = 3$ and 4, since the combination of sources and values that satisfy the quality threshold will explode, the number of nodes in the search tree generated by accurate search also increases, resulting in a sharp increase in runtime. Meanwhile, since the sources that can be used for the extension of the search tree are pre-constrained in approximate search, the number of search tree nodes increases slower, and the runtime is considerably less than the accurate search.

Cost. As $|O_Q|$ increases, the cost increases, because in order to find the records of all objects in O_Q, it is often necessary to access more data sources. Overall, the cost of accurate search is lower than the cost of approximate search, but when $|O_Q| = 6$, the two are the same, because no result meets the quality threshold at this time, both methods return the result with highest quality score.

Quallity. As $|O_Q|$ increases, the quality of the results decreases. Accurate search sacrifices quality to reduce costs while ensuring that the quality threshold is satisfied, so sometimes the resulting quality scores are lower than the approximate search. For example, in the case of $|O_Q| = 3$ or $|O_Q| = 4$, accurate search can find the solution that with lowest cost, but the quality is also lower than the solution returned by approximate search.

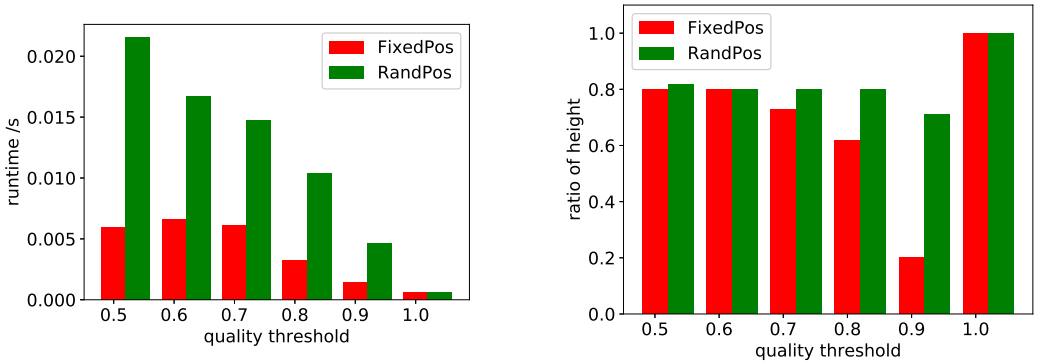
7.2. Evaluation of Search Space Size Estimation

The most important step in adjusting the inappropriate quality threshold is to estimate the size of the search space and the height of the search tree. We have experimented with the two estimation strategies (i.e., *FixedPos* and *RandPos*) proposed in Section 6.1. The experimental results are shown in Figure 7. In the experiments, $|\mathbb{S}|$ and $|O_Q|$ are set to be the maximum values (i.e., 56 and 6 respectively), and the quality threshold θ varies from 0.5 to 1.

Efficiency. We first compare the runtime of the two strategies. The time cost of *RandPos* is overall slightly higher than *FixedPos*. When the threshold changes, the time cost of *RandPos* and *FixedPos* decreases(that is, the efficiency rises). This is explained by the fact that the higher the threshold, the earlier the search process stops. At the same time, we found that the time cost of both strategies is much lower than the time cost of accurate search and approximate search. Even if the quality threshold is set very low, the two strategies can complete the entire estimation process within 30 ms. With the same settings, accurate search and approximate search may take tens of minutes to return results. Therefore, the time cost for estimating the size of the search space is negligible compared to the time the query is executed. In other words, if we can perform the estimation before the query starts, we can assume that this will add little extra time cost.

Effectiveness. We compare the estimated height and true height of the search tree. The experimental results are shown in Figure 7b, where *ratio of height* is defined as $\hat{h}/h$. Since $\hat{h} \leq h$, the ratio of height

is in $[0, 1]$. This value indicates the accuracy of the estimate. It can be observed in the experimental results that the performance of *RandPos* is relatively stable compared to *FixedPos*. When the threshold changes, the performance of *RandPos* is stable (basically kept above 0.8), while *FixedPos* may fluctuate depending on the fixed position to be replaced. Concretely, some of the branches in the search tree may be pruned very early. For example, some objects may have fewer candidate values, then *FixedPos* may have no value to replace very early on these objects. Or, some attributes fail to satisfy the quality threshold after a few steps of replacement, so the corresponding branches will be pruned early. When the threshold is set to 0.9, the situation that the replacement of a single location cannot satisfy the quality threshold is very easy to occur, so the estimated accuracy is not stable enough. When the threshold is 1, the search tree height is 1 because there is no query result can meet the threshold, so the height ratio of *FixedPos* and *RandPos* is 1, that is, both of the two strategies can accurately estimate the height of the search tree.



(a) runtime (b) ratio of height

Figure 7. Experimental results of search space size estimation.

In summary, the time overhead of the two strategies is much lower than the execution of the search, and the estimated search tree height is basically close to the real situation. Therefore, if we can perform the estimation operation before the query starts, it can effectively avoid the unreasonable threshold setting while adding little time overhead.

8. Conclusions and Future Work

This paper studies the problem of answering MQQ query on multi-sources in sensor-cloud systems. First, a general definition of MQQ answering problem in the multi-source environment is given. After that, the quality measurement is provided based on relative quality constraints, and two methods for solving the MQQ answering problem are proposed. To deal with the case that it is difficult to set the reasonable quality threshold, a method to automatically check the resonablility of the threshold is proposed.

In future work, we will study how to get more types of quality scores of different sources by the methods of data mining or machine learning, and will explore other forms of quality-preserving queries.

Author Contributions: Algorithm design, M.L. and Z.T.; Making charts, M. L.; Manuscript writing, M.L., Z.T. and Y.J.; Literature search, Y.S. and Y.J.; Data collection, Y.S.; Data analysis, Y.S. and Y.J.

Funding: This research was funded by National Natural Science Foundation of China (No. 61702220, 61702223, 61871140, 61572153) and the National Key Research and Development Plan (Grant No. 2018YFB0803504).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rahm, E.; Do, H.H. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.* **2000**, *24*, 3–13.
2. Lazaridis, I.; Han, Q.; Yu, X.; Mehrotra, S.; Venkatasubramanian, N.; Kalashnikov, D.V.; Yang, W. Quasar: Quality aware sensing architecture. *ACM SIGMOD Rec.* **2004**, *33*, 26–31. [CrossRef]
3. Alamri, A.; Ansari, W.S.; Hassan, M.M.; Hossain, M.S.; Alelaiwi, A.; Hossain, M.A. A survey on sensor-cloud: Architecture, applications, and approaches. *Int. J. Distrib. Sens. Netw.* **2013**, *9*, 917923. [CrossRef]
4. Wang, T.; Zhang, G.; Liu, A.; Bhuiyan, M.Z.A.; Jin, Q. A Secure IoT Service Architecture with an Efficient Balance Dynamics Based on Cloud and Edge Computing. *IEEE Internet Things J.* **2018**. [CrossRef]
5. Wang, T.; Zhou, J.; Liu, A.; Bhuiyan, M.Z.A.; Wang, G.; Jia, W. Fog-based Computing and Storage Offloading for Data Synchronization in IoT. *IEEE Internet Things J.* **2018**. [CrossRef]
6. Wang, T.; Zhang, G.; Bhuiyan, M.Z.A.; Liu, A.; Jia, W.; Xie, M. A novel trust mechanism based on Fog Computing in Sensor–Cloud System. *Future Gener. Comput. Syst.* **2018**. [CrossRef]
7. Li, M.; Jiang, Y.; Sun, Y.; Tian, Z. Answering the Min-cost Quality-aware Query on Multi-sources in Sensor-Cloud Systems. In Proceedings of the The Fourth International Symposium on Sensor-Cloud Systems (SCS 2018), Melbourne, Australia, 11–13 December 2018.
8. Fan, W.; Geerts, F. Foundations of data quality management. *Synth. Lect. Data Manag.* **2012**, *4*, 1–217. [CrossRef]
9. Rammelaere, J.; Geerts, F.; Goethals, B. Cleaning Data with Forbidden Itemsets. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 897–908.
10. Fan, W.; Geerts, F.; Jia, X.; Kementsietsidis, A. Conditional Functional Dependencies for Capturing Data Inconsistencies. *ACM Trans. Database Syst.* **2008**. [CrossRef]
11. Fan, W.; Geerts, F.; Wijsen, J. Determining the currency of data. *ACM Trans. Database Syst.* **2012**, *37*, 25. [CrossRef]
12. Chu, X.; Ilyas, I.F.; Papotti, P. Holistic data cleaning: Putting violations into context. In Proceedings of the IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, Australia, 8–12 April 2013; pp. 458–469.
13. Rekatsinas, T.; Joglekar, M.; Garcia-Molina, H.; Parameswaran, A.; Ré, C. SLiMFast: Guaranteed results for data fusion and source reliability. In Proceedings of the 2017 ACM International Conference on Management of Data, Chicago, IL, USA, 14–19 May 2017; ACM: New York, NY, USA, 2017; pp. 1399–1414.
14. Cao, Y.; Fan, W.; Yu, W. Determining the relative accuracy of attributes. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 22–27 June 2013.
15. Fan, W.; Geerts, F. Relative information completeness. *ACM Trans. Database Syst.* **2010**, *35*, 27. [CrossRef]
16. Li, M.; Li, J.; Cheng, S.; Sun, Y. Uncertain Rule Based Method for Determining Data Currency. *IEICE Trans. Inf. Syst.* **2018**, *101*, 2447–2457. [CrossRef]
17. Li, M.; Li, J. A minimized-rule based approach for improving data currency. *J. Comb. Optim.* **2016**, *32*, 812–841. [CrossRef]
18. Dong, X.L.; Gabrilovich, E.; Murphy, K.; Dang, V.; Horn, W.; Lugaresi, C.; Sun, S.; Zhang, W. Knowledge-based trust: Estimating the trustworthiness of web sources. *Proc. VLDB Endow.* **2015**, *8*, 938–949. [CrossRef]
19. Dong, X.L.; Berti-Equille, L.; Srivastava, D. Integrating conflicting data: The role of source dependence. *Proc. VLDB Endow.* **2009**, *2*, 550–561. [CrossRef]
20. Lin, X.; Chen, L. Domain-aware multi-truth discovery from conflicting sources. *Proc. VLDB Endow.* **2018**, *11*, 635–647.
21. Dong, X.L.; Berti-Equille, L.; Srivastava, D. Truth discovery and copying detection in a dynamic world. *Proc. VLDB Endow.* **2009**, *2*, 562–573. [CrossRef]
22. Dong, X.L.; Berti-Equille, L.; Hu, Y.; Srivastava, D. Global detection of complex copying relationships between sources. *Proc. VLDB Endow.* **2010**, *3*, 1358–1369. [CrossRef]
23. Madria, S.; Kumar, V.; Dalvi, R. Sensor cloud: A cloud of virtual sensors. *IEEE Softw.* **2014**, *31*, 70–77. [CrossRef]
24. Santos, I.L.; Pirmez, L.; Delicato, F.C.; Khan, S.U.; Zomaya, A.Y. Olympus: The cloud of sensors. *IEEE Cloud Comput.* **2015**, *2*, 48–56. [CrossRef]

25. Fazio, M.; Puliafito, A. Cloud4sens: A cloud-based architecture for sensor controlling and monitoring. *IEEE Commun. Mag.* **2015**, *53*, 41–47. [CrossRef]
26. Lyu, Y.; Yan, F.; Chen, Y.; Wang, D.; Shi, Y.; Agoulmine, N. High-performance scheduling model for multisensor gateway of cloud sensor system-based smart-living. *Inf. Fusion* **2015**, *21*, 42–56. [CrossRef]
27. Abdelwahab, S.; Hamdaoui, B.; Guizani, M.; Znati, T. Cloud of things for sensing-as-a-service: Architecture, algorithms, and use case. *IEEE Internet Things J.* **2016**, *3*, 1099–1112. [CrossRef]
28. Zhu, C.; Leung, V.C.; Wang, K.; Yang, L.T.; Zhang, Y. Multi-method data delivery for green sensor-cloud. *IEEE Commun. Mag.* **2017**, *55*, 176–182. [CrossRef]
29. Dinh, T.; Kim, Y.; Lee, H. A Location-Based Interactive Model of Internet of Things and Cloud (IoT-Cloud) for Mobile Cloud Computing Applications. *Sensors* **2017**, *17*, 489. [CrossRef]
30. Wang, Y.; Tian, Z.; Zhang, H.; Su, S.; Shi, W. A Privacy Preserving Scheme for Nearest Neighbor Query. *Sensors* **2018**, *18*, 2440. [CrossRef]
31. Tian, Z.; Cui, Y.; An, L.; Su, S.; Yin, X.; Yin, L.; Cui, X. A Real-Time Correlation of Host-Level Events in Cyber Range Service for Smart Campus. *IEEE Access* **2018**, *6*, 35355–35364. [CrossRef]
32. Tan, Q.; Gao, Y.; Shi, J.; Wang, X.; Fang, B.; Tian, Z.H. Towards a Comprehensive Insight into the Eclipse Attacks of Tor Hidden Services. *IEEE Internet Things J.* **2018**. [CrossRef]
33. Yu, X.; Tian, Z.; Qiu, J.; Jiang, F. A Data Leakage Prevention Method Based on the Reduction of Confidential and Context Terms for Smart Mobile Devices. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 11. [CrossRef]
34. Jiang, F.; Fu, Y.; Gupta, B.B.; Lou, F.; Rho, S.; Meng, F.; Tian, Z. Deep Learning based Multi-channel intelligent attack detection for Data Security. *IEEE Trans. Sustain. Comput.* **2018**. [CrossRef]
35. Chen, J.; Tian, Z.; Cui, X.; Yin, L.; Wang, X. Trust architecture and reputation evaluation for internet of things. *J. Ambient Intell. Humaniz. Comput.* **2018**, 1–9. [CrossRef]
36. Zhihong, T.; Wei, J.; Yang, L. A transductive scheme based inference techniques for network forensic analysis. *China Commun.* **2015**, *12*, 167–176.
37. Zhihong, T.; Wei, J.; Yang, L.; Lan, D. A digital evidence fusion method in network forensics systems with Dempster-shafer theory. *China Commun.* **2014**, *11*, 91–97.
38. Lian, X.; Chen, L.; Wang, G. Quality-aware subgraph matching over inconsistent probabilistic graph databases. *IEEE Trans. Know. Data Eng.* **2016**, *28*, 1560–1574. [CrossRef]
39. Yeganeh, N.K.; Sadiq, S.; Sharaf, M.A. A framework for data quality aware query systems. *Inf. Syst.* **2014**, *46*, 24–44. [CrossRef]
40. Wu, H.; Luo, Q.; Li, J.; Labrinidis, A. Quality aware query scheduling in wireless sensor networks. In Proceedings of the Sixth International Workshop on Data Management for Sensor Networks, Lyon, France, 24 August 2009; ACM: New York, NY, USA, 2009; p. 7.
41. Chu, X.; Ilyas, I.F.; Papotti, P. Discovering denial constraints. *Proc. VLDB Endow.* **2013**, *6*, 1498–1509. [CrossRef]
42. Kruse, S.; Naumann, F. Efficient discovery of approximate dependencies. *Proc. VLDB Endow.* **2018**, *11*, 759–772. [CrossRef]
43. Zou, Z.; Gao, H.; Li, J. Discovering Frequent Subgraphs over Uncertain Graph Databases Under Probabilistic Semantics. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 25–28 July 2010; pp. 633–642.
44. Zou, Z.; Li, J.; Gao, H.; Zhang, S. Frequent Subgraph Pattern Mining on Uncertain Graph Data. In Proceedings of the 18th ACM Conference on Information and Knowledge Management, Hong Kong, China, 2–6 November 2009; pp. 583–592.
45. Abiteboul, S.; Kanellakis, P.; Grahne, G. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.* **1991**, *78*, 159–187. [CrossRef]
46. Garey, M.R.; Johnson, D.S. *Computers and Intractability. A Guide to the Theory of NP-Completeness*; WH Freeman and Co.: San Francisco, CA, USA, 1979.