Sifting Truths From Multiple Low-quality Data Sources

Zizhe Xie[1], Qizhi Liu[1], and Zhifeng Bao[2]

[1] State Key Lab. for Novel Software Technology,
Nanjing University, Nanjing, China
forrest0402@smail.nju.edu.cn,lqz@nju.edu.cn
[2] School of CSIT,RMIT University,Melbourne,Australia

Abstract. In this paper, we study the problem of assessing the quality of co-reference tuples extracted from multiple low-quality data sources and finding true values from them. It is a critical part of an effective data integration solution. In order to solve this problem, we first propose a model to specify the tuple quality. Then we present a framework to infer the tuple quality based on the concept of *quality predicates*. In particular, we propose an algorithm underlying the framework to find true values for each attribute. Last, we have conducted extensive experiments on real-life data to verify the effectiveness and efficiency of our methods.

Keywords: Data fusion · Data cleaning · Predicates

1 Introduction

Web data grow at an unprecedented pace following the increasing number of data sources, and people get opportunities to access a wide variety of information and viewpoints from multiple individual sources. Although where to find answers is not troublesome anymore, it remains a big challenge on how to sift true answers from multiple low-quality data sources [7].

Consider a person named Mary shown in Table 1. We collected four tuples for Mary from four sources. Tuple t_i is collected from D_i. Apparently, there are conflicts among the four tuples and we need to resolve conflicts among different salaries and affiliations.

Existing data fusion methods [2, 5, 6, 8] more or less take a voting approach, i.e., accumulating votes from various sources for each value on the same object and selecting the value with the highest vote. However, an inherent limitation of this model is that it is not able to find multiple true values, and it has to depend on a certain relationship among sources to work. Unlike existing methods which make the assumption that only one true value exists, in this paper, we aim to solve the same problem under a more relaxed assumption: multiple true values may exist, and propose a framework to find such true values. In particular, we have made the following contributions:

– We propose a novel data quality model which integrates several data quality criteria and we drop out the common assumption in previous work.

Table 1: **Entity instance** person **for Mary**

	Name	Salary	Research Area	Affiliation	Publication
t_1:	Mary	142k	Data integration, data cleaning	Amazon	Data integration
t_2:	Mary	120k	Data cleaning	Google	null
t_3:	Mary	88k	Knowledge management	AT&T Labs	A diagnostic tool for data errors
t_4:	Mary	88k	Information retrieve	null	null

- We propose the concept of *quality predicates* to differentiate true values from false values, which is able to work when there exist multiple true values from data sources.
- For a certain tuple, we propose an algorithm to infer its quality vectors based on its quality predicate(s). The time complexity of finding top-k true values in our method outperforms other rule-based methods.
- We experimentally verify the effectiveness and efficiency of our methods using two real datasets.

The remainder of this paper is organized as follows. Sec. 2 formally defines the quality model. Sec. 3 presents the algorithm to infer the *quality vectors* and find true values. Sec. 4 reports the experiments and we conclude at Sec. 5.

2 Model for Tuple Quality

In this section, we will introduce a model to specify the tuple quality.

2.1 Quality Predicates

There have been much work in assessing the tuple quality from plenty of semantic facets such as consistency [4], currency [3] and accuracy [1]. In this paper, we propose three types of *quality predicates* to evaluate tuple quality.

Priority. A *Priority relationship* denotes the relationship defined on attributes. If a set of values of one attribute is messy or impure, it is hard to find true values. However, if attribute values are pure i.e. most sources provide the same value, it is easy to find true values. In our model, we use Shannon entropy to calculate purity. For an attribute A_i: $H(A_i) = -\sum_{x \in X} p(x) \log_2 p(x)$, where X is the set of the classes of values in A_i (null is not a class), and $p(x)$ is the proportion of the number of x to the number of values from A_i. Besides, for the values of the same attribute, the higher the proportion of null values is, the harder the values are to acquire. We use $p_n(A_i)$ to represent the proportion of null values for A_i.

Definition 1. *(Priority predicate). For attributes A_i and A_j, if $\frac{H(A_i)}{1-p_n(A_i)} < \frac{H(A_j)}{1-p_n(A_j)}$, we define a priority predicate* $\mathrm{Prior}(A_i, A_j)$.

By Definition 1, a priority relationship is a total order on A $= (A_1, ..., A_l)$. Let $P_{score}(A_i) = \frac{H(A_i)}{1-p_n(A_i)}$. Assuming $P_{score}(A_1) < ... < P_{score}(A_l)$, for any two adjacent $P_{score}(A_i) < P_{score}(A_j)$, we can define a *priority predicate* $\text{Prior}(A_i, A_j)$.

Example 1. In Table 1, we can define three *priority predicates*: Prior(Salary, Research Area), Prior(Salary, Publication) and Prior(Publication, Affiliation).

Status. A *status relationship* is the relationship among the values of the same attribute. The quality of an attribute value changes along with that attribute value.

Definition 2. *(Status predicate). A status predicate* $\text{Stat}(A_i)$ *is in the form* $(\forall t_i, \forall t_j)(P(t_i^{(A_k)}, t_j^{(A_k)}) \wedge \phi(t_i, t_j))$ *representing that the values of* A_i *satisfying a certain condition are worse than the others.*

A *status predicate* φ is defines on two tuples (it is less likely that a predicate that involves more tuples in real life), and it checks whether $t_i^{(A_k)}$ and $t_j^{(A_k)}$ satisfy the condition defined by P. We then introduce the predicates we use.

For numberic values, we define $P_1(v_1, v_2)$ ($P_2(v_1, v_2)$) to denote v_1 is bigger (less) than v_2. For string typed values, we define $P_3(v_1, v_2)$ ($P_4(v_1, v_2)$) to denote v_1 is longer (shorter) than v_2. We also define $P_5(v_1, v_2)$ ($P_6(v_1, v_2)$) to represent v_1 is more (less) detailed than v_2 in term of their information entropy.

Note that ϕ in *status predicates* is a condition that t_i and t_j must comply. We define $\phi(t_i, t_j) = f_1(t_i^{(A_m)}, t_j^{(A_m)}) \wedge ... \wedge f_l(t_i^{(A_n)}, t_j^{(A_n)})$ where $f_i(v_1, v_2)$ is either $v_1 = v_2$ or $v_1 \neq v_2$.

Example 2. In Table 1, a tuple with a higher salary is of better quality. Hence, we can define φ_4 : $\text{Stat}(\mathsf{Salary}) = (\forall t_i, \forall t_j)(P_{lt}(t_i^{(\mathsf{Salary})}, t_j^{(\mathsf{Salary})}))$ to represent a tuple with lower salary is of lower quality.

Interaction. *Interaction* is the relationship among values of different attributes in one tuple. For example, in Table 1, some values are null, which indicates they are of low quality.

Definition 3. *(Interaction predicate). An interaction predicate* $\text{Inter}_\delta(A_1, ..., A_l)$ *represents that when a tuple satisfies condition* δ*, its values of attribute* $A_1, ..., A_l$ *are of low quality.*

For an *interaction predicate*, $\delta = (\forall t_i)(P_1'(t_i^{A_m}, c_1) \wedge ... \wedge P_l'(t_i^{A_n}, c_l))$ where $P_i'(v, c)$ can be any predefined predicate. For *interaction predicate*, we add four more predicates.

For the values of string type, we define $P_7(v, c)$ and $P_8(v, c)$ to represent v_1 contains or not contains c. For both numberic values, we define $P_9(v, c)$ and $P_10(v, c)$ to represent v_1 is equal or not equal to c.

The second argument c is in the range of the dataset. c usually represents a single word or a punctuation. In addition, no matter what schema a instance has, we can define a universal *interaction predciate* φ_5 : $\text{Inter}_{(\forall t_i)(P_7(t_i^{(A_i)}, null))}(A_i)$ for every attribute.

3 Deducing Tuple Quality

In this section, we present an algorithm to deduce the *quality vectors* of tuples by *quality predicates*. *Quality vectors* are a n-ary vector representing the quality of a tuple, where the real number in each dimension represents the quality of corresponding attribute value in the tuple.

Our method to find true values is based on the voting approach and assumes that multiple true values exist. For each attribute, if the attribute is labeled as "multi-valued", we will return all the attribute values with non-negative values in *quality vectors*. If the attribute is labeled as "time-sensitive", we will return the attribute value with the highest value in *quality vectors*.

The runtime of our algorithm has the square relation with the size of the tuples. Finding top-k candidates is a polynomial problem after deducing *quality vectors*, while it is NP-hard in a chase-like algorithm [1].

Applying quality predicates. *Priority predicates* are used for distinguishing the quality of two similar attribute values by the quality of another attribute value. Hence we apply a priority predicate $Prior(A_i, A_j)$ for two tuples t_1 and t_2 when $\boldsymbol{q}_{t_1}^{(A_j)} = \boldsymbol{q}_{t_2}^{(A_j)}$ and $\boldsymbol{q}_{t_1}^{(A_i)} > \boldsymbol{q}_{t_2}^{(A_i)}$. In this case, if only $t_1^{(A_j)} \neq t_2^{(A_j)}$, we consider $t_1^{(A_j)}$ is of better quality than $t_2^{(A_j)}$. Because priority relationship is a total order relationship on all attributes, we can sort *priority predicates* by their P_{score} of the first attribute in descending order and traverse them in sequence.

Status predicates are used for distinguishing the values in different status. We apply them in the comparison between two tuples i.e. we extract all distinct pairs of tuples and apply *status predicates* on them. During the comparison between t_1 and t_2, if they satisfy a *status predicate* $\mathrm{Stat}(A_k) = (\forall t_1, \forall t_2)(P(t_1^{(A_k)}, t_2^{(A_k)}) \wedge \phi(t_1, t_2))$, we decrease η from $\boldsymbol{q}_{t_1}^{(A_k)}$.

The *interaction predicate* is defined on w.r.t. a single tuple. For a *interaction predicate* $\mathrm{Inter}_\delta(A_1, ..., A_l)$, if a tuple t_1 satiesfies δ, we decrease η from $\boldsymbol{q}_{t_1}^{(\mathrm{A})}$ where $\mathrm{A} = (A_1, ..., A_l)$.

Influence factor. When applying *quality predicates*, we need to change the values in the corresponding *quality vectors*. We use the word "influence factor" to represent the degree that the *quality vectors* being changed. For simplicity, we set all influence factors when applying *status predicates* and *interaction predicate* to the same value, and set influence factors to 1 when applying *priority predicates* because that is enough to distinguish two different facts of equal quality. We will discuss the influence of different η in Section 4.

Execution order. When applying *quality predicates*, is different applying order leads to a different result? The answer to this question is affirmative. *Status predicates* and *interaction predicates* act on the attribute values that are immutable. Therefore, they can be applied without disturbing each other. However, *priority predicates* act on the variable value of *quality vectors*. Therefore, *priority predicates* should be applied in the last after we inferred *quality vectors*.

Algorithm 1: Deducing the quality vectors of tuples

```
1  function DTQ (M);
   Input  : a specification M = (D, e, I_s, P_P, P_S, P_I)
   Output: the quality vectors Q of I_s
2  I_e ← PARTITION(I_s, e);
3  Q ← ∅;
4  for I_{e_i} ∈ I_e do
5      Q_{e_i} ← INIT(I_{e_i});
6      for φ_s ∈ P_S do
7          for (t_i, t_j) ∈ I_{e_i} do
8              APPLYSTAT(t_i, t_j, φ_s, Q_{e_i});
9          end
10     end
11     for φ_i ∈ P_I do
12         for t_i ∈ I_{e_i} do
13             APPLYINTER(t_i, φ_i, Q_{e_i});
14         end
15     end
16     for φ_c ∈ TOPSORT(P_P) do
17         for I ∈ SORTBYCLASS(I_{e_i}) do
18             for (t_i, t_j) ∈ I do
19                 APPLYPRIOR(t_i, t_j, φ_c, Q_{e_i});
20             end
21         end
22     end
23     Q ← Q ∪ Q_{e_i};
24 end
25 return Q;
```

Algorithm. We now present the main driver of DTQ. Given M, it first partitions I_s into $\mathbf{I}_e = (I_{e_1}, ..., I_{e_q})$ by entity set $\boldsymbol{e} = (e_1, ...e_q)$. For each I_{e_i}, it initializes an empty $|I_{e_i}| \times n$ *quality matrix* $\mathbf{Q}_{e_i} = (\boldsymbol{q}_{t_1}, ..., \boldsymbol{q}_{t_{|I_{e_i}|}})$, where n presents the number of attributes and each row in $\mathbf{Q}_{e_i}$ represents a *quality vector* of a tuple. Then it starts to apply *status predicates* to deduce *quality vectors*. Assuming $\varphi_s = \text{Stat}(A_k)$, APPLYSTAT will subtract η from $\boldsymbol{q}_{t_i}^{(A_k)}$ if $t_i^{(A_k)}$ and $t_j^{(A_k)}$ satisfy φ_s. Next, DTQ applies *interaction predicates* to further update $\mathbf{Q}_{e_i}$. It traverses each tuple of I_{e_i}. Assuming $\varphi_i = \text{Inter}_\delta(A_1, ..., A_k)$, If t_i satisfies condition δ, then APPLYINTER will substracts η from $\boldsymbol{q}_{t_i}^{(\boldsymbol{A})}$, where $\boldsymbol{A} = (A_1, ..., A_k)$. Finally, it applies *priority predicates*. Assuming $\varphi_c = \text{Prior}(A_1, A_2)$, if $t_i^{(A_2)} = t_j^{(A_2)}$, APPLYPRIOR will make the comparison between $t_i^{(A_1)}$ and $t_j^{(A_1)}$ and plus 1 to $\mathbf{Q}_{e_i}^{(A_2)}$ of the tuple with greater value on A_1. Notice that before extracting pairs of tuples, it first sorts I_{e_i} in ascending order by the quality of A_2 and groups I_{e_i} by the quality of A_2. We use I to denote a group of tuples that have the same value on A_2.

4 Experimental Study

Using both real-life data and synthetic data, we conducted three sets of experiments to evaluate: (1) the effectiveness of algorithm DTQ on the real dataset, compared with the algorithms of [8, 2]; (2) the relationship between the trustworthiness of sources and recall; and (3) the influence of η;

Experimental setting. We used two real-life datasets[3] (Book and Flight). In our experiment, we used FOIL to discover the *quality predicates*. We manually filtered the predicates of low quality. For all datasets, we set $\eta = 15$ and $\epsilon = 0.1$.

We implemented the following, all in C#: (1) DTQ; (2) TruthFinder [8]; and (3) a truth discovery algorithm Sim [2]. All experiments were conducted on a 64bit Windows Intel Core(TM) i5-3470 CPU with 16GB of memory and 1000GB of storage. Each experiment was repeated 5 times and the average is reported.

Exp-1: Effectiveness of DTQ. Using real life data Book and Flight we evaluated the effectiveness of DTQ compared with Sim [2] and TruthFinder [8].

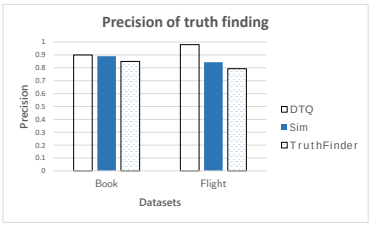

Fig. 1: Results of truth finding.

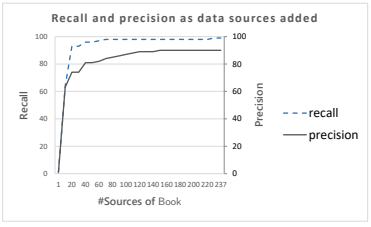

Fig. 2: Applying DTQ on Book.

We tested how many true values were correctly derived by DTQ. As shown in Figure 1, according to the gold standard, DTQ achieved 90% precision while Sim achieved 89% precision and TruthFinder achieved 85% of precision on Book.

We also ran DTQ on Flight to further verify its effectiveness. In Figure 1, DTQ get the highest precision on all datasets. In [6], 16 methods including 1 baseline methods, 4 web-link based methods, 3 IR based methods, 7 bayesian based methods and AccuCopy were used to find true values on Flight. Among them, AccuCopy get the highest precision of 96.0%. However, DTQ get a higher precision than AccuCopy. It also can be seen from Figure 1 that DTQ worked well on Book and Flight.

Exp-2: Trustworthiness of sources. Using real life data Book and Flight, we applied DTQ to calculate the trustworthiness of each data source and sort data

[3] All datasets can be download from http://lunadong.com/fusionDataSets.htm

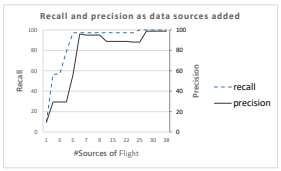

Fig. 3: Applying DTQ on Flight.

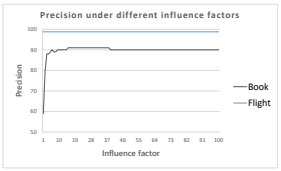

Fig. 4: η may affect the precision of DTQ.

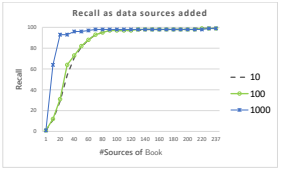

Fig. 5: η affects the convergence rate of recall.

sources by their quality in descending order. For a data source D_i, we calculate the trustworthiness of it as $\lambda_i = \sum_{t \in D_i} d(\boldsymbol{q}_t)/|D_i|$, where t is the tuple pertaining to D_i and $|D_i|$ is the size of D_i. By the rank of data sources, we evaluated (a) the change of recall and precision as data sources added; and (b) the distribution of the trustworthiness of data sources.

Recall. As shown in Figure 2 and Figure 3, we tested the recall as data sources added. In the experiments, we used preference model $d_h(\mathbf{u}) = \mathbf{u}^{(1)} + ... + \mathbf{u}^{(n)}$ to convert a vector to a real number. From the figures, the recall is definitely monotonically increasing as data sources added. All datasets show that true values lie in a few datasets. The curve in Figure 3 is not as steep and smooth as that in Figure 2 because the number of data sources is small and we only used a few *quality predicates* to distinguish the trustworthiness of data sources. In conclusion, by scoring data sources, we can use a small amount of data sources to get a same or even better result. Last, an effective rank of data sources verifies the effectiveness of DTQ.

Precision. Figure 2 and Figure 3 also report the precision of DTQ as data sources added. The change of precision is not definitely monotonically increasing. Before the recall reaches the peak, adding more data sources means a larger precision, and the figure of precision during this phase is similar to that of recall. After recall remains stable, more data sources mean more noises, thus the precision may start to shock. In practice, if we can use a small amount of data sources to get a high recall, we can then use a small amount of data sources to get a good precision.

Exp-3: Influence factor η. Intuitively, we should not choose an extremely small value because we use it to filter tuples and a tiny value will not have an effect. We evaluated integer η from two aspects and we set η for all *quality predicates* the same value for simplicity.

Influence on precision. We first tested the influence of different size of η on the precision of DTQ using Book and Flight. As shown in Figure 4, we ran DTQ when η in the range of 1 to 100. For Book, when $\eta = 1$, the precision is only 59%. When $1 < \eta < 15$, the precision has a slight wobble. When $\eta \geq 15$, the precision remains 91%. For Flight, the size of η does not influence the result of DTQ. From observation, we found that whether η has an influence depends on *quality predicates*. In practice, this situation is common and there are always

imperfect *quality predicates*. Hence, we need to avoid a very large η as well as an extraordinary small η. We set $\eta = 15$ for all our experiments.

Influence on recall. We then tested the influence of different η in *interaction predicates* on the recall of DTQ on Book. As shown in Figure 5, we executed DTQ and recorded the recall when $c = 10, 30, 100, 200, 1000$ separately. We found that with the increase in η, the convergence rate of recall is getting slow, which means we need more data sources to achieve an equal result.

5 Conclusion

This paper studied how to the estimate tuple quality and find true values among multiple low-quality data sources. We measured tuple quality by three types of *quality predicates* including *priority predicates*, *status predicates* and *interaction predicates*. These *quality predicates* are in a simple form and can be found automatically by existing methods. By the quality model, we can assess the quality of attribute values and the quality of tuple, and thereby find the true values via *quality vectors*.

In future, we would like to explore how the trustworthiness of data sources affect the accuracy of the true values.

Acknowledgements. This work is supported by the National Natural Science Foundation of China (Grant No. 61572247,61373130).

References

1. Y. Cao, W. Fan, and W. Yu. Determining the relative accuracy of attributes. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 565–576. ACM, 2013.
2. X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: the role of source dependence. *Proceedings of the VLDB Endowment*, 2(1):550–561, 2009.
3. W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. *ACM Transactions on Database Systems (TODS)*, 37(4):25, 2012.
4. W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *Proceedings of the VLDB Endowment*, 3(1-2):173–184, 2010.
5. A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 131–140. ACM, 2010.
6. X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava. Truth finding on the deep web: Is the problem solved? In *Proceedings of the VLDB Endowment*, volume 6, pages 97–108. VLDB Endowment, 2012.
7. Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, and J. Han. A survey on truth discovery. *SIGKDD Explor. Newsl.*, 17(2):1–16, Feb. 2016.
8. X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the web. *Knowledge and Data Engineering, IEEE Transactions on*, 20(6):796–808, 2008.