# VisClean: Interactive Cleaning for Progressive Visualization

Yuyu Luo[†] Chengliang Chai[†] Xuedi Qin[†] Nan Tang[‡] Guoliang Li[†]
[†]Department of Computer Science, Tsinghua University [‡]Qatar Computing Research Institute, HBKU
{luoyy18@mails., chaicl15@mails., qxd17@mails., liguoliang@}tsinghua.edu.cn, ntang@hbku.edu.qa

## ABSTRACT

Data visualization is crucial in data-driven decision making. However, *bad* visualizations generated from dirty data often mislead the users to understand the data and to draw wrong decisions. We present VisClean, a system that can progressively visualize data with improved quality through interactive and visualization-aware data cleaning. We will demonstrate two main features of VisClean: (1) *Easy-to-use:* the users can easily answer data cleaning questions through a novel GUI; and (2) *Cheap-to-clean:* the quality of bad visualizations can be significantly improved in a few interactions.

## 1. INTRODUCTION

Data visualization plays a key role in impacting strategic and operational decisions of today's data-driven businesses. However, data visualizations are not always correct and good. One common reason for such bad (uncertain) visualizations is that real-life data is often dirty.

**Example 1:** Table 1 shows publications from multiple sources. Dirty Citations cells are marked by red. Duplicated records are marked by the same color on attribute Id. Besides, it also contains unstandardized values such as "`Very Large Data Bases`" and "`VLDB`". The ground truth of Table 1 is given in Table 2. For example, $t_{123}$ in Table 2 is the consolidated record for tuples $t_1$, $t_2$, and $t_3$ in Table 1.

[An Incorrect Bar Chart.] Figure 1(a) is a bar chart about the #-total of Citations grouped by Venue. Due to various types of errors, the visualization is incorrect. For example, duplicated bars: "`SIGMOD Conf.`", "`ACM SIGMOD`", "`SIGMOD`", and "`SIGMOD'13`" should be merged, $t_2$ has an *outlier* at attribute Citations that affects the bar "`SIGMOD Conf.`", and $t_7$ has a missing Citations value that affects the bar "`VLDB`". □

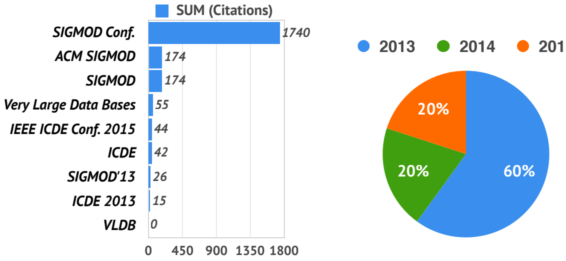
However, a visualization is not necessarily dirty, even if the data is dirty. Consider another example.

| Id | Year | Title (abbr.) | Venue | Affiliation | Citations |
|---|---|---|---|---|---|
| $t_1$ | 2013 | NADEEF | ACM SIGMOD | QCRI | 174.0 |
| $t_2$ | 2013 | NADEEF | SIGMOD Conf. | QCRI, HBKU | 1740 |
| $t_3$ | 2013 | NADEEF | SIGMOD | QCRI HBKU | 174.0 |
| $t_4$ | 2013 | KuaFu | ICDE 2013 | Microsoft | 15.0 |
| $t_5$ | 2013 | TsingNUS | SIGMOD'13 | Tsinghua | 13.0 |
| $t_6$ | 2013 | TsingNUS | SIGMOD'13 | THU | 13.0 |
| $t_7$ | 2014 | SeeDB | VLDB | Stanford Univ. | N.A. |
| $t_8$ | 2014 | SeeDB | Very Large Data Bases | Stanford | 55.0 |
| $t_9$ | 2015 | Elaps | ICDE | NUS | 42.0 |
| $t_{10}$ | 2015 | Elaps | IEEE ICDE Conf. 2015 | CS@NUS | 44.0 |

**Table 1: An Excerpt of DB Papers (Dirty)**

| Id | Year | Title (abbr.) | Venue | Affiliation | Citations |
|---|---|---|---|---|---|
| $t_{123}$ | 2013 | NADEEF | SIGMOD | QCRI | ***174.0*** |
| $t_4$ | 2013 | KuaFu | ICDE | Microsoft | 15.0 |
| $t_{56}$ | 2013 | TsingNUS | SIGMOD | Tsinghua | 13.0 |
| $t_{78}$ | 2014 | SeeDB | VLDB | Stantford Univ. | ***55.0*** |
| $t_{910}$ | 2015 | Elaps | ICDE | NUS | 43.0 |

**Table 2: An Excerpt of DB Papers (Ground Truth)**



(a) An Incorrect Bar Chart (b) A Correct Pie Chart

**Figure 1: Example Visualization on Table 1**

**Example 2:** [A Correct Pie Chart.] Figure 1(b) shows the proportion of the #-papers by Year and the result is not affected by dirty data, because the pie chart generated from the Table 1 is the same as the one from the Table 2. □

In practice, it is too expensive to clean the entire dataset. Intuitively, compared with cleaning the entire dataset, only cleaning visualization-aware data may be much cheaper. Hence, we study a new problem, *interactive cleaning for progressive visualization* (ICPV), to progressively improve the visualization quality by minimizing the cost of interacting with the user to clean the visualization-aware data.

**Challenges.** We face four main challenges. (C1) How to quantify the impact of data errors on visualization? (C2) What is the ideal user- and task-friendly interface to interact with the user? (C3) How to compute the expected benefit of cleaning data errors for a visualization? (C4) How to select the "most beneficial" question to minimize the human cost?

**The VisClean System.** We have addressed these research challenges in a recent research paper [5]. We propose to demonstrate VisClean with the following two main features: (1) *Easy-to-use:* the users can easily answer data

*Proceedings of the VLDB Endowment,* Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

cleaning questions through a novel GUI; and (2) *Cheap-to-clean:* the quality of bad visualizations can be significantly improved in a few interactions. The demonstration video is at `https://youtu.be/eqsw7L8iRFE`.

**Outline.** Section 2 introduces some background knowledge and gives an overview of VisClean. Section 3 discusses the demonstration scenarios. Section 4 presents some technical details that address the aforementioned challenges.

# 2. SYSTEM OVERVIEW

## 2.1 Background

### 2.1.1 Visualization Query

We use a SQL-like visualization query to visualize *bar/pie* charts. Given a query $Q$ on the relational dataset $D$, it will produce a *visualization*, denoted as $Q(D)$. For example, the bar chart in Fig. 1(a) is created by:

$Q_1(D)$ : **VISUALIZE** *Bar* **SELECT** *Venue, SUM(Citations)*
**FROM** *Table I* **GROUP BY** *Venue*

### 2.1.2 Visualization Meets Data Errors

We consider four types of common data errors that may affect the quality of visualizations.

(1) *Tuple-level duplicates* are tuples that refer to the same entity. For example, tuple $t_5$ and $t_6$ in Table 1 are tuple-level duplicates. Consider $Q_1(D)$ for Fig. 1(a). The Citations of $t_6$ is counted twice, which leads to an incorrect bar chart.

(2) *Attribute-level duplicates* (or synonyms) are variants for the same value, such as "`VLDB`", and "`Very Large Data Bases`". Consider $Q_1(D)$ again, all variants for "`SIGMOD`" entity (*e.g.,* "`ACM SIGMOD`", "`SIGMOD Conf.`", "`SIGMOD' 13`") should be normalized to "`SIGMOD`". Thus Citations of these tuples are incorrectly aggregated for the bar chart in Fig. 1(a).

(3) *Missing values.* Cell $t_7$[Citations] is a missing value. It affects the aggregation result of the "`VLDB`" group in Fig. 1(a).

(4) *Outliers.* Cell $t_2$[Citations] is an outlier. It greatly affects the aggregation result of "`SIGMOD Conf.`" group in Fig. 1(a).

### 2.1.3 Single Questions vs. Composite Questions

For each type of data errors, the **single questions** are:

(1) *Tuple-level duplicates:* "*Are $t_1$ and $t_2$ duplicates*", where tuples $t_1$ and $t_2$ are possible duplicates.

(2) *Attribute-level duplicates: Are value $v_1$* (e.g., `VLDB`) *and $v_2$* (e.g., `VLDB'18`) *the same? If so, which value should be used?*

(3) *Missing values: Which possible repair a missing value (attribute a of tuple t) should take?*

(4) *Outliers: Is value v in tuple t an outlier? If so, which (possible) repair should it take?*

We use $T$-, $A$-, $M$-, and $O$-question to denote the above four types of questions respectively.

**Composite Questions.** In practice, it is hard for a user to precisely answer a *single question* without enough context. Consequently, interactive data cleaning systems typically provide more information to the user, beyond only one data error [2]. Hence, we propose to use a graph model [5] to holistically organize *single questions.*

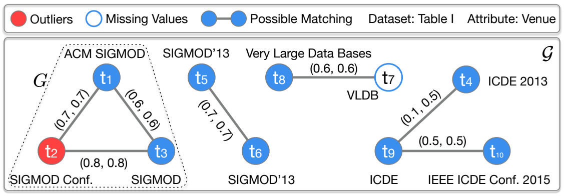


**Figure 2: A Sample ERG and CQG**

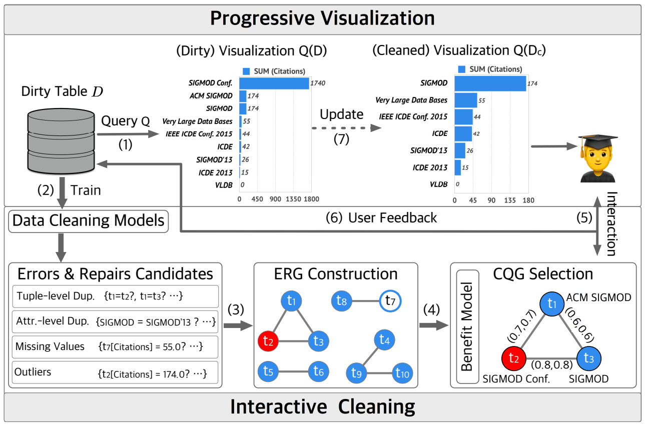


**Figure 3: System Overview**

*Errors-and-Repairs Graph (ERG).* All errors and possible repairs are modeled as an undirected weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Each vertex $v_i \in \mathcal{V}$ is a tuple, and possibly has a red label that means the tuple has an outlier (*i.e.,* an $O$-question), and hollow indicates a missing value (*i.e.,* an $M$-question). Each edge $(v_i, v_j) \in \mathcal{E}$ with a weight $(p_{ij}^t, p_{ij}^a)$ denotes that $v_i$ and $v_j$ are possible tuple-level duplicates (*i.e.,* a $T$-question) or attribute-level duplicates (*i.e.,* an $A$-question) with $p_{ij}^t$ or $p_{ij}^t$ probability.

For example, graph $\mathcal{G}$ in Fig. 2 is a sample ERG for Fig. 1(a). It has 10 vertices, one contains an $O$-question ($t_2$), and another has an $M$-question ($t_7$). Note that it just shows edges that are weighted (either $p_{ij}^t$ or $p_{ij}^a$) between 0.5 and 0.8. It has 7 edges to denote 7 $T$-questions and 7 $A$-questions. Particularly, the edge $(t_4, t_9)$ denotes that entity $t_4$ and $t_9$ are tending to match with probability 0.1, while the attribute values (*i.e.,* Venue) for $x$-axis on $t_4$ and $t_9$ match with probability 0.5.

Instead of using *single questions* on an ERG to interact with the user, we propose to use **composite questions** to provide more context for the user, which is also considered in [1]. A **composite questions**, namely *composite questions graph* (CQG), is a connected induced subgraph $G$ of an ERG $\mathcal{G}$. For instance, an induced subgraph $G$ in Fig. 2 is a CQG.

## 2.2 Overview

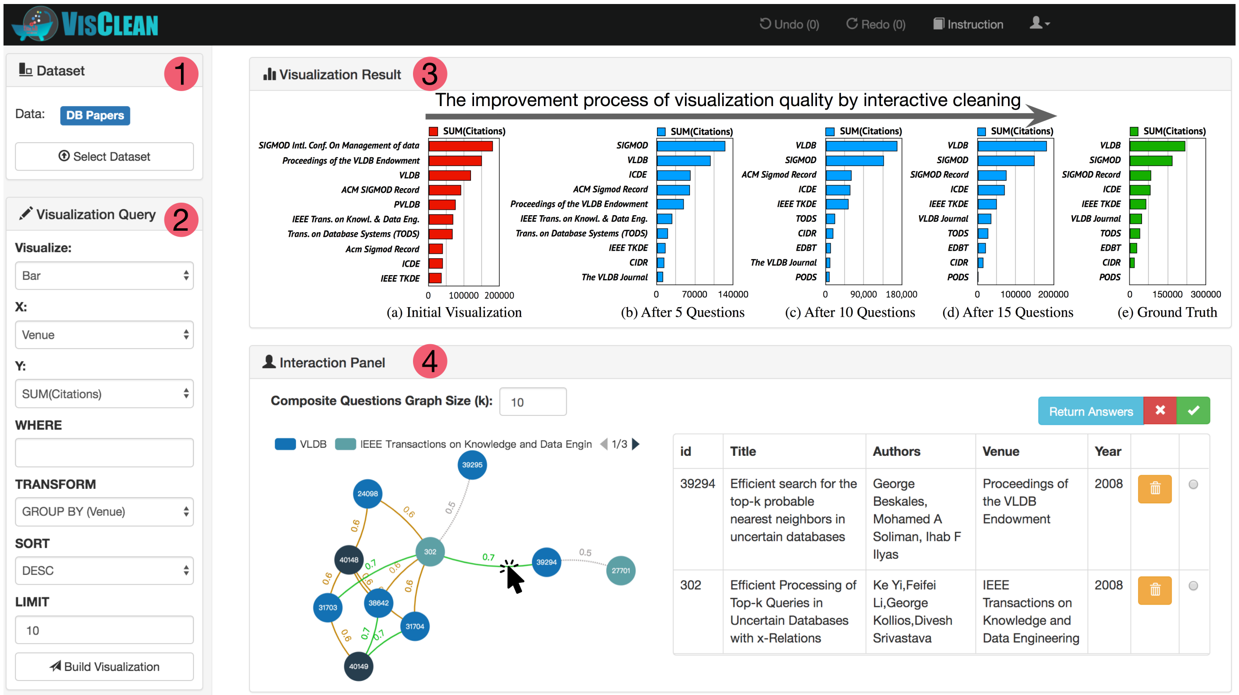
The overview of VisClean for solving the ICPV problem is shown in Fig. 3, with the following steps.

***Step 1: Visualization Specification.*** The user needs to specify a visualization query on a specific dataset.

***Step 2: Initialization.*** VisClean first needs to run off-the-shelf data cleaning tools to detect different types of errors and generate possible repairs.

***Step 3: Errors-and-Repairs Graph Construction.*** It organizes the detected data errors and possible repairs by a graph.

***Step 4: CQG Selection.*** It selects the "most beneficial" composite questions to interact with the user in each iteration.



(a) Initial Visualization (b) After 5 Questions (c) After 10 Questions (d) After 15 Questions (e) Ground Truth

Data: DB Papers

Visualize: Bar

X: Venue

Y: SUM(Citations)

WHERE

TRANSFORM GROUP BY (Venue)

SORT DESC

LIMIT 10

Composite Questions Graph Size (k): 10

| id | Title | Authors | Venue | Year | | |
|---|---|---|---|---|---|---|
| 39294 | Efficient search for the top-k probable nearest neighbors in uncertain databases | George Beskales, Mohamed A Soliman, Ihab F Ilyas | Proceedings of the VLDB Endowment | 2008 | 🗑 | ○ |
| 302 | Efficient Processing of Top-k Queries in Uncertain Databases with x-Relations | Ke Yi,Feifei Li,George Kollios,Divesh Srivastava | IEEE Transactions on Knowledge and Data Engineering | 2008 | 🗑 | ○ |

Figure 4: A Screenshot of VisClean

***Step 5: User Interaction***. The user interacts with our graph-based GUI to provide feedback.

***Step 6: Repair Errors and Update Visualization***. After getting feedback from the user, VisClean will repair data errors and refresh the data visualization. Afterwards, the user feedback might be used by each used data cleaning model to find new errors/possible repairs *(Step 2)*. The process iterates until a user budget for the number of interactions is used up.

## 3. DEMONSTRATION OVERVIEW

In this section, we showcase the main features of VisClean. For more details, please refer to our demonstration video (`https://youtu.be/eqsw7L8iRFE`).

**Datasets.** We have collected hundreds of real-world datasets whose semantics is easy to understand by the general audience. We will use a `DB Papers` dataset for a running example. This dataset contains 50,483 papers (13,915 distinct papers), and its excerpt is shown in Table 1. We also encourage the audiences to try their own datasets using VisClean.

**End-to-End Experience.** Figure 4 is a screenshot of the front-end of VisClean. The user can do interactive cleaning for progressive visualization with the following steps.

*(1) Dataset and Visualization Specification.* The user can upload a dataset by clicking the button "Select Dataset" (see Fig. 4–①). Then, the user can create a visualization query, *e.g.,* a bar chart with top-10 Venues ranking by total Citations, in Fig. 4-②. The corresponding visualization will be shown in Fig. 4-③. For example, the red bar chart (histogram) is the initial visualization before interactive cleaning.

*(2) Back-end Algorithms.* VisClean performs ***Step 2–Step 4*** in the back-end. The selected CQG will be shown in Fig. 4-④.

*(3) Interactive Cleaning.* Next, the user can interact with the CQG to provide feedback for cleaning visualization. For each edge, the user can *confirm* (resp. *split*) an edge to denote its associated two vertices are (resp. *not*) the same tuple- or attribute-level entity. For each vertex, the user can either *approve* or *reject* the outliers and missing values repair candidates. In Fig. 4-④, the user clicks edge (302, 39294), it will show more details about the corresponding two tuples using table UI to help the user to make the decision. For example, the user clicks ✖ button to split the edge to represent tuple 302 and tuple 39294 are not the duplicates. For more details, please refer to our demonstration video.

*(4) Repair Errors and Update Visualization.* Then, VisClean will refresh the visualization on the cleaner version dataset (by repairing data errors). The updated visualization will be displayed in Fig. 4-③ (those blue bar charts).

The above steps *(2)-(4)* iterate until the budget is used up. Then, the user can browse and export the cleaned version of the visualization and the corresponding dataset.

**Running Example.** We can see the initial red bar chart (Fig. 4-③-(a)) is very dirty. Obviously, there are many synonymous bars (*e.g.,* `VLDB` and `PVLDB`) due to the attribute-level duplicates. Besides, there are also some potential tuple-level duplicates, outliers and missing values in the original data for the bar chart. After 5 CQG questions, the visualization (Fig. 4-③-(b)) is improved a lot. For example, many synonymous bars (*e.g.,* `VLDB` and `PVLDB`) are merged as `VLDB` (user picks `VLDB` as the standard value), and some tuple-level duplicates are removed by the entity matching model. After asking 10 CQG questions, the visualization (Fig. 4-③-(c)) is significantly improved, which is already quite similar to the ground truth (Fig. 4-③-(e)). For example, the order of top-5 Venues are the same as the ground truth. The *Conference* `PODS` first appears among the top-10 bars, in part because VisClean standardizes many attribute-level duplicates referring to the `PODS` entity, *e.g.,* standardize `PODS`, `In`

Pods, and PODS 2018. Therefore, the total Citations of PODS increases. After 15 iterations, we can see that the total Citations of each bar (Fig. 4-③-(d)) increases and is very similar to the one generated by the ground truth dataset. Part of the reason is that lots of synonymous bars (*e.g.,* VLDB'18 and Very Large Data Bases) are merged as the standard bar (*i.e.,* VLDB). Note that, we need to ask hundreds of questions to clean the entire dataset to produce the ground truth.

# 4. BEHIND THE SCENES

## 4.1 Error Detection and Repair Generation

$\mathbb{Q}_T$*: Questions for Tuple-Level Duplicates.* We first train an entity matching model (we use random forests [4]). For each tuple pair (*e.g.,* $t_1$ and $t_3$), the entity matching model will provide a matching probability. Then we use the active learning techniques to generate a set of tuple pairs $\mathbb{Q}_T$, *i.e.,* those uncertain pairs with probability close to 0.5.

$\mathbb{Q}_A$*: Questions for Attribute-Level Duplicates.* We incorporate two strategies. The first strategy leverages the entity matching cluster to detect attribute-level duplicates and generate string transformations. We implement this strategy based on the existing techniques, *i.e., GoldenRecordCreation* [3]. However, it's hard for the above method to detect those attribute-level duplicates across two different entity matching clusters. Thus, we propose to run a *string similarity join* algorithm on the column for $x$-axis (*e.g.,* Venue) to generate a set of similar string pairs for detecting attribute-level duplicates. Hence, we can combine the above two strategies to generate a set of $A$-questions $\mathbb{Q}_A$.

$\mathbb{Q}_O$*: Questions for Outliers Repairing.* We use kNN [6] to detect outliers on the column for $y$-axis (*e.g.,* Citations), which computes an outlier score for each value $v$. The score is defined as the $k-th$ smallest absolute difference between all other values and $v$. Then we sort all the values based on the scores in descending order. Next, we select those values with the largest scores as the $O$-questions $\mathbb{Q}_O$.

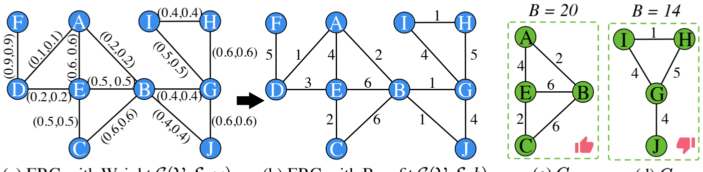
$\mathbb{Q}_M$*: Questions for Missing Values Imputation.* Given a tuple with a missing value on the column for $y$-axis (*e.g.,* Citations), we can use the above method for outliers repairing to generate a set of $M$-questions $\mathbb{Q}_M$ for imputing missing values.

The above detected errors and possible repairs will be organized as an ERG. Next, we introduce how to select the "most beneficial" CQG from ERG.

## 4.2 Composite Questions Graph Selection

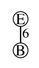
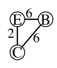
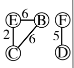
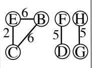
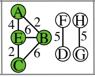
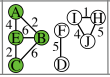
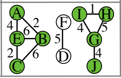
### 4.2.1 Estimation-based Benefit Model

We first discuss how to measure the benefit on an edge of a CQG. Since the user can *confirm/split* an edge of a CQG to provide new training data to enhance the quality of data cleaning models and thus improve the quality of visualization. More specifically, for each edge, we enumerate all possible user operations in this edge, *i.e., confirm/split* the edge with possibility $\mathbf{P}^Y/\mathbf{P}^N$. For each type of user operation, it provides useful feedback to repair data errors and retrain data cleaning models. Then we can derive a new version of dataset $D_c$ from $D$. At last, we can update the visualization on $D_c$. Thus, we can measure the visualization distance **dist** before/after cleaning the data using distance function (*e.g.,* Earth Mover distance). The larger the distance is, the larger the estimated benefit is. Next, we discuss how to quantify the benefit of a CQG. We use the sum of the total benefit of each edge to estimate the benefit of a CQG. Thus, given a CQG $G(V, E)$, its estimated benefit can be computed as: $\mathcal{B}(G) = \sum_{i=1}^{|E|}(\mathbf{P}_i^Y \mathbf{dist}_i^Y + \mathbf{P}_i^N \mathbf{dist}_i^N)$.



(a) ERG with Weight $\mathcal{G}(\mathcal{V}, \mathcal{E}, w)$ (b) ERG with Benefit $\mathcal{G}(\mathcal{V}, \mathcal{E}, b)$ (c) $G_1$ (d) $G_2$

$k = 4$

| Iteration | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ··· |
|---|---|---|---|---|---|---|---|---|
| Edge $(v, v')$ | (B, E) | (B,C) | (D, F) | (G, H) | (A, E) | (G, I) | (G, J) | ··· |
| Collection $\mathcal{C}$ | {B, E} | {B, C, E} | {B, C, E}<br>{D, F} | {B, C, E}<br>{D, F} {G, H} | {A, B, C, E}<br>{D, F} {G, H} | {A, B, C, E}<br>{D, F} {G, H, I} | {A, B, C, E}<br>{D, F} {G, H, I, J} | ··· |
| Subgraph $G$ |  |  |  |  |  |  |  | ··· |

(e) Iterating Edges Phase

**Figure 5: Example of CQG Selection**

### 4.2.2 CQG Selection Algorithm

After we know how to estimate the benefit of a CQG, we now aim to select the "most beneficial" CQG from the ERG. Given an ERG $\mathcal{G}(\mathcal{V}, \mathcal{E}, b)$ and an integer $k$, the optimal CQG selection problem aims to find a connected subgraph $G$ of $\mathcal{G}$ with $k$ vertices ($k$-subgraph) such that the total edge weight (*i.e.,* benefit) is maximum.

The straightforward approach is to enumerate all connected subgraphs with $k$ vertices, compute total benefit of each subgraph, and then select the subgraph with maximum total weight as the optimal CQG. Unfortunately, we have proved that the *Optimal CQG Selection* problem is NP-hard [5]. Thus, we devise a greedy algorithm [5] to select the CQG effectively and efficiently. The basic idea is that we select subgraphs with large benefits as candidates greedily and incrementally. For the convenience of elaboration, Fig. 5 is an example for our algorithm. Firstly, we compute each edge's benefit based on *Estimation-based Benefit Model* and get $\mathcal{G}(\mathcal{V}, \mathcal{E}, b)$ (Fig. 5(b)). Since we want to incorporate edges with large benefit into CQG, we select the unvisited edge with maximum benefit in each iteration to greedily construct the candidate CQG set. For instance, in iteration 1, we select edge $(B, E)$ and put its two endpoints into a vertex set (see Fig. 5(e)). After iterating all edges, we construct two $k$-subgraphs (Fig. 5(c) and Fig. 5(d)) and we select the subgraph with maximum benefit as the target CQG (*i.e.,* Fig. 5(c)).

# 5. REFERENCES

[1] M. Bergman, T. Milo, and et al. Query-oriented data cleaning with oracles. In *SIGMOD*, 2015.

[2] X. Chu, J. Morcos, and et al. KATARA: a data cleaning system powered by knowledge bases and crowdsourcing. In *SIGMOD*, 2015.

[3] D. Deng and et al. Unsupervised String Transformation Learning for Entity Consolidation. In *ICDE*, 2019.

[4] P. Konda, S. Das, and et al. Magellan: Toward building entity matching management systems. *PVLDB*, 2016.

[5] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li. Interactive cleaning for progressive visualization through composite questions. In *ICDE, 2020.*

[6] S. Ramaswamy and et al. Efficient algorithms for mining outliers from large data sets. In *SIGMOD*, 2000.