CheetahER: An Accurate and Efficient Entity Resolution System for Heterogeneous Camera Data

SIGMOD 2020 Programming Contest Finalist Paper

Nan Deng[†], Wendi Luan[†], Haotian Liu[†], Bo Tang[†‡*]
[†]Department of Computer Science and Engineering, Southern University of Science and Technology
[‡]PCL Research Center of Networks and Communications, Peng Cheng Laboratory
{11711004@mail.,11712532@mail.,11613015@mail.,tangb3@}sustech.edu.cn

ABSTRACT

The SIGMOD Programming Contest 2020 raises a real-world entity resolution problem, which requires to identify product specifications from multiple e-commerce websites that represent the same real-world cameras. Entity resolution has been extensively studied and the general solution framework consists of two phases: *blocking* and *matching*. Most existing works focus on the matching phase, which trains (complex) models on large volumes of data and uses the models to decide whether a pair of descriptions refers to the same real-world object. However, training a high-quality model is difficult for the SIGMOD contest because there is only a limited amount of labeled data and the product specifications can be dirty and incomplete.

In this paper, we propose *CheetahER*, an accurate and efficient entity resolution system. Different from existing works, we focus on improving the effectiveness of the blocking phase, which is overlooked in both academia researches and industry systems, and propose a two-phase blocking framework to group the product specifications according to brand and model. The pre-processing and data cleaning procedures are also carefully designed to improve data quality. *CheetahER* ranks the 1st in accuracy among 53 teams and completes the task within 20 seconds. Even though some designs of *CheetahER* are specialized for camera datasets, its novel two-phase blocking framework and operators (i.e., merging and splitting) may generalize to other entity resolution tasks.

KEYWORDS

Data Integration, Entity Resolution, Two-phase Blocking

1 INTRODUCTION

Entity resolution, which identifies different records (e.g., web-pages and user names) referring to the same real-world entity, is an important task in the database community. In the SIGMOD Programming Contest 2020 [3], organizers introduce a real-world entity resolution problem: finding which camera specifications from 24 different e-commerce websites represent the same camera. Two types of datasets are provided to the participants: (i) camera specification datasets and (ii) ground-truth datasets. For a camera specification dataset, each camera specification is stored as a JSON file and an example is provided in Figure 1. All JSON files have a common attribute *page title* (highlighted in Figure 1) but the other attributes (not their values), such as *model* and *brand* (marked in gray), could be different in different specifications. This makes entity resolution difficult as the same camera can have different attributes in different JSON files. A ground-truth dataset is a CSV file, in which each row is a record that contains three fields, i.e., *left specification id*, *right specification id* and *label*. *label=1* indicates that the two specifications refer to the same camera. The participants are required to list all pairs of specifications that represent the same camera using a format similar to the ground-truth dataset. The contest ranks the participants by the accuracy of their results and measures accuracy using the F1 score, which is defined as

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}.$$

A general solution to the entity resolution problem consists of two steps, i.e., *blocking* and *matching*. The blocking step divides the dataset into a number of groups to reduce the complexity for pair-wise comparison. The matching step enumerates all possible pairs in each block and decides whether a pair of records (specifications in our case) refers to the same entity. Currently, both academia and industry focus on the matching step and developed many learning-based solutions that achieve good accuracy for pair-wise comparison [1, 2]. However, learning-based solutions is not suitable for the contest due to two challenges. (i) *Insufficient labeled data*, only 300k ground-truth pairs are given but there are 900M

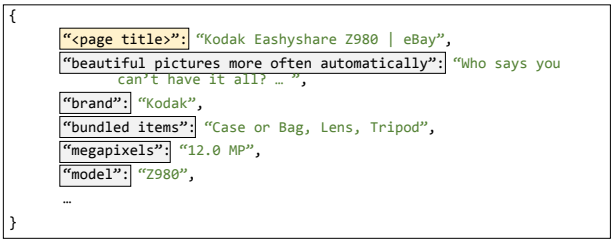

Figure 1: An example of the camera specification JSON file

Table 1: An illustration of the ground-truth dataset

Left specification ID	Right specification ID	Label
www.ebay.com//24887	www.ebay.com//56369	1
www.ebay.com//42902	www.mypriceindia.com//57	0

*Corresponding author: Bo Tang.

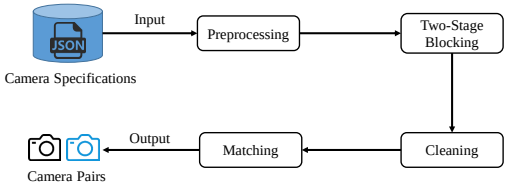

Figure 2: System architecture for *CheetahER*

possible specification pairs. The limited size of the ground-truth dataset makes it difficult to train high quality models. (ii) *Poor data quality*, the specifications are unstructured data with different attributes. We believe that the two challenges are also general for many real-word entity resolution problems.

Our system, *CheetahER*, is designed to overcome the aforementioned challenges. Different from existing works, we focus on the blocking step and introduce two block operations: *merging* and *splitting*. The merging operation merges two or more blocks into one while the splitting operation splits one block into multiple blocks. A complete set of rules is also designed to control the execution of the block operations. *CheetahER* achieves an F1 score of 98% and runs within 20 seconds. We are also honored as finalist, ranking top-5 on the world-wide leaderboard [4]. Details of *CheetahER* can be found in our open-sourced code[1] .

2 SYSTEM ARCHITECTURE

As illustrated in Figure 2, *CheetahER* has four components: *prepossessing*, *two-stages blocking*, *cleaning* and *matching*. The prepossessing step loads data into memory, extracts useful information and organizes the specifications in structured form. The blocking step indexes the specifications and clusters similar specifications into a group, and the cleaning step adjusts the group assignment of incorrectly classified specifications. Finally, the matching step enumerates all possible specifications pairs in each block as the result. That is, for a block of size n, $n(n-1)/2$ matched pairs will be produced.

3 PREPROCESSING

Brand and *model* can identify a specific camera, and thus the preprocessing step retrieves attributes from which *brand* and *model* could be extracted. The *page title* attribute is present for JSON files and it often contains information about brand and model. For example, the *page title* attribute of spec "www.wexphotographic.com//626" is "Samsung WB350F Digital Smart Camera ...", which includes its brand "Samsung" and model "WB350F". *Brand* and *model* may also appear independently in other attributes. However, the *model* attribute can be ambiguous. e.g. "0002724284400" and "Camera" are model attributes in specifications "www.buzzillions.com//854.json" and "www.ebay.com//60127.json", respectively. Thus, we only keep the *page title* and *brand* attributes in this step.

4 TWO-STAGE BLOCKING

An illustration of our two-stage blocking method is provided in Figure 3, which consists of two phases, *brand blocking* and *model blocking*. We elaborate the two phases as follows.

[1] https://github.com/LUUUAN/EntityMatching_SIGMOD_2020_Contest

4.1 Brand-based Blocking

We first divide the camera specifications into different blocks according to their brands. As shown in Figure 3, brand-based blocking consists of two steps: *grouping* and *merging*.

Grouping by Brand. In this step, we first extract the brand names from the *brand* attribute in the JSON files. Although the *brand* attribute only exists in a small portion of the specifications, it helps to obtain a set of brands that could cover most camera specifications. We then utilize the *page title* attribute in each JSON file to extract more brand information. A camera specification will be grouped into a brand block if the brand appears in the specification's *page title* attribute. For example, the JSON file in Figure 1 has attribute *brand* with value "Kodak", then we are able to create a block with brand name "Kodak". All JSON files that have "Kodak" as a substring in its *page title* attribute will be grouped into this block. We generate blocks for all encountered brands, e.g., "Canon", "Cannon", "Fuji", "Fujifilm", as shown in Figure 3. These blocks will go through the merging step to improve accuracy.

Merging. Block merging is used to merge different blocks that correspond to the same brand. As shown in Figure 3, we obtain blocks with name "Canon" and "Cannon" after grouping but "Cannon" is apparently a typo of the correct spelling "Canon". Different blocks can also be created for the same brand due to alias, "Fuji" and "Fujifilm" in Figure 3 for example. To handle spelling error and alias, we introduce two criteria for merging the brand blocks.

The first criteria utilizes the regular rules. For two brand blocks with brand name A and B, if A is the prefix of B (e.g. "Fuji" and "Fujifilm"), then block A should be merged with block B. The other criteria is based on the Levenshtein distance for strings[5].

$$\mathrm{lev}_{A,B}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{A,B}(i-1,j)+1 \\ \mathrm{lev}_{A,B}(i,j-1)+1 \\ \mathrm{lev}_{A,B}(i-1,j-1)+1_{(A_i \neq B_j)} \end{cases} & \text{otherwise,} \end{cases}$$

Here, A_i and A_j are the sub-strings of A and B with first i and j characters, respectively. Merging brand blocks with a small Levenshtein distance helps to tackle spelling errors, e.g., "Canon" and "Cannon".

4.2 Model Blocking within Brand Blocking

In this step, we further divide each brand block into multiple blocks based on the camera model. Model blocking consists of three phases, i.e., *grouping*, *merging* and *splitting*.

Grouping. Model names cannot be easily extracted as the brand names do. On the one hand, there are many specifications whose model name is missing from the *model* attribute; on the other hand, the value of the *model* attribute can be ambiguous. For instance, in a specification with id "www.buzzi-llions.com/872", the value of the *model* attribute is "15820728", but its actual model is "F100fd" with brand "Fujifilm" by scrutinizing the data.

According to our observation, there are two patterns for the name of camera models: (i) model names usually consist of only alphabet, number, space and crossbar; (ii) model names can be constructed by a combination of prefix and postfix, e.g., "EOS 1Ds". Based on these observations, we define a suite of regular rules to extract a set of model names from the "page title" attribute within

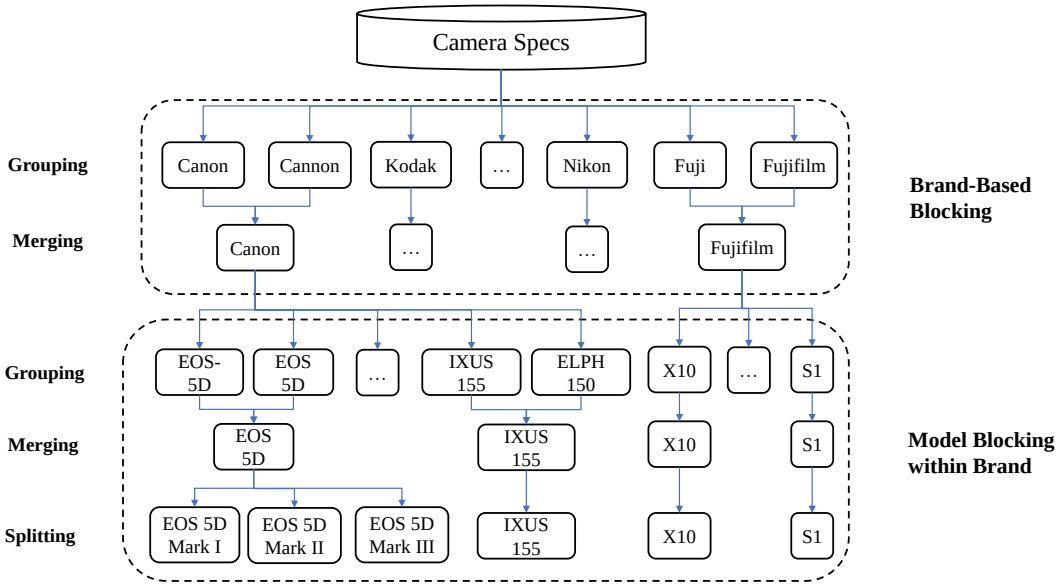

Figure 3: An example of two-stage blocking

each brand block. Similar to brand-based blocking, we can further group specifications in each brand block into several model blocks, using the model name set extracted previously. For instance, the JSON file shown in Figure 1 will be classified into block "Kodak" in the brand-based blocking step. In the model blocking step, its model name "Z980" can be extracted using regular rules described above. As a result, it will be grouped into the block which contains all camera manifestations that has "Kodak" and "Z980" in their *page title* attribute. Similarly, as shown in Figure 3, the brand block "Canon" will be further divided into model blocks with name "EOS 5D", "IXUS 155", etc.

Merging. Model block merging is similar to brand block merging but the merge conditions are different. We can't directly apply the brand block merging rules due to two reasons: (i) the similarity between brand blocks is defined using the Levenshtein distance and does not work for models, e.g., the Levenshtein distances between "EOS 50D" and "EOS 5D" is only one but they are different models; (ii) the same camera model can have different model names in different regions. For example, "IXY 140", "ELPH 150" and "IXUS 155" are names of the same model when they are sold in Japan, America and elsewhere, respectively.

We propose two criteria for model merging to tackle the aforementioned problems. Firstly, some e-commerce websites tend to write all possible names of a model in its *page title*, i.e., models names like "IXUS 155" and "ELPH 150" may appear in a single *page title* attribute values. In our example, this camera specification will be classified into both "IXUS 155" and "ELPH 150" model blocks. As a heuristic method, we merge two model blocks if their common specifications is greater than a user-given threshold. This merging condition can be formally specified as follows: Given two brand blocks A and B with $A \neq B$, if $|A \cap B| > e$, in which e is a threshold parameter, then the two brand blocks can be merged. The other model merging criteria is based on the format of the model names. The same model name can be expressed in various forms, for instance, the model "Canon EOS 5D" can have name "EOS-5D", "EOS5D", or even a simple name "5D". In this case, we ignore the noncontributory prefixes and postfixes and merge them together.

The pseudo-code for model block merging is shown in Algorithm 1 and the merging threshold e is 3 by default. We will show that the model merging strategies are effective and can significantly improve recall in the experiments in Section 5.

Algorithm 1: Model Block Merging

Input: Model blocks M_{t_1}, M_{t_2}, ..., M_{t_n} in brand block B_t
Output: Model blocks M_{t_1}, M_{t_2}, ..., M_{t_k}

```
for i=1:n do
    for j=i+1:n do
        if |M_ti ∩ M_tj| > e then
            Merge M_ti and M_tj
    end
end
```

Splitting. After the brand blocking and model blocking steps, some specifications referring to different entities could be put into the same block. We found that this is because some models need to be further distinguished by their generations. For example, "Canon EOS 5D Mark II" will be blocked to "EOS 5D" by the previous steps, losing its generation information "Mark II". As a result, "Canon EOS 5D Mark II" will be paired with "Canon EOS 5D Mark I" and "Canon EOS 5D Mark III" when generating the solution. This will severely degrade the precision of the final result. So we design a set of regular rules like ".*Mark [(I)|(II)|(III)|(IV)].*" to identify "Mark II" from "Canon EOS 5D Mark II", then extract them and generate new blocks for such records.

5 BLOCK CLEANING

The block cleaning step aims to remove accessories (e.g., lens and bags) that should not contribute to the final result. Accessories may contain several brands and models, and thus may be assigned to multiple blocks and paired with specifications referring to real camera instances, which hampers the precision of the result. For instance, "New Wide Angle Macro Lens for Canon EOS Digital Rebel Camera XTi T3i T4i 18 55mm | eBay" describes a camera lens but is assigned to model "XTi", "T3i" and "T4i".

To solve this problem, a revert list is built to record the model blocks a specification has been assigned to. We detect and delete accessories according to the length of the reverted list. That is, we regard a specification as accessory if it is assigned to a large number of model blocks. Setting a good length threshold is crucial for the effectiveness of cleaning. If the length threshold is too small, some pairs that ought to be matched will be discarded. For example, in "Canon EOS 400D Digital Rebel XTi 10 1MP Digital SLR Camera Silver | eBay", "XTi" is an alias of "EOS 400D". On the other hand, if the length threshold is too large, some accessories may not be identified. We found the optimal length threshold to be 3 by traversing all possible values. An exhaustive search is acceptable for 2 reasons: (i) the maximal length of the reverted list is 6 for all specifications in the dataset, and thus the search space is not large; (ii) the cleaning process is very efficient and running block cleaning under a threshold is very fast.

6 MATCHING

The two-stage blocking with split and merge operators has classified the specifications into blocks quite accurately. Thus, the matching procedure can be made very simple–enumerating all possible pairwise combination within each block as the result. Therefore, for a block with size n, a total $n(n-1)/2$ matched pairs will be generated in the final result.

7 EXPERIMENTAL RESULTS

We implemented *CheetahER* using C++ and conducted the experiments on a machine with 4x Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz and 16 GB memory. To test the gain of block merging and splitting, we disable them and create two variants of *CheetahER*. The accuracy results are reported in Figure 4, and the precision and recall scores are acquired from contest committee.

The results show that block merging significantly improves recall, i.e., from 0.88 to 0.97. This is because more specification pairs referring to the same camera can be identified when block merging puts them into the same block. On the other hand, block splitting improves precision, from 0.93 to 0.99. This is because block splitting avoids generating false positive specification pairs that do not represent the same entity. Combining block merging and block splitting, the *CheetahER* achieves an F1 score of 0.98. In the meantime, *CheetahER* is also efficient, running the entire processing piepiline within 20 seconds.

8 CONCLUSION

In this work, we develop an entity resolution engine, *CheetahER*, for the SIGMOD Programming Contest 2020. Different from popular methods that heavily rely on machine learning, *CheetahER* focuses on blocking. We design a comprehensive blocking pipeline that involves brand blocking, model blocking, block splitting, block merging and block cleaning by considering properties of the problem and dataset. Our experiment results show that *CheetahER* is accurate an efficient, achieving an F1 score of 0.98 and running within 20 seconds on a standard CPU machine. We think the success of *CheetahER* shows that it is crucial to consider the characteristics of the problem and data in practical data mining problems such as entity resolution. Despite that machine learning based solutions are highly successful for many problems, *CheetahER* is an example that simple rule-based solutions are still valuable if they are properly guided by insights from data.

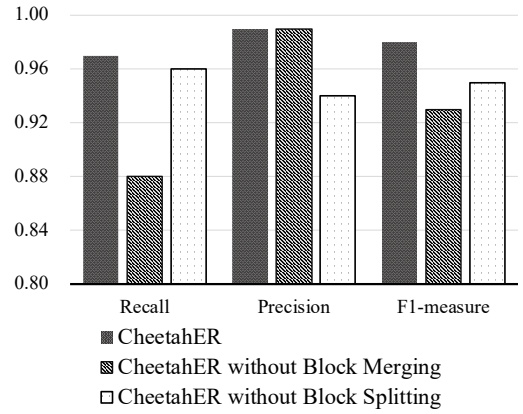

Figure 4: Accuracy of the *CheetahER* variants

9 ACKNOWLEDGMENTS

This work was supported by the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. JCYJ20180302174301157), the Education Department of Guangdong (Grant No. 2020KZDZX1184), the National Science Foundation of China (NSFC No. 61802163), and PCL Future Regional Network Facilities for Large-scale Experiments and Applications (PCL2018KP001).

REFERENCES

[1] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *PVLDB* 9, 12 (2016), 1197–1208. https://doi.org/10.14778/2994509.2994535

[2] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 19–34. https://doi.org/10.1145/3183713.3196926

[3] Database Research Group of the Roma Tre University. [n.d.]. ACM SIGMOD 2020 Programming Contest. [EB/OL]. http://www.inf.uniroma3.it/db/sigmod2020contest/task.html.

[4] Database Research Group of the Roma Tre University. 2017. *ACM SIGMOD 2020 Programming Contest Leaderboard.* Retrieved July 13, 2020 from http://www.inf.uniroma3.it/db/sigmod2020contest/leaders.html

[5] Wikipedia. 2020. Levenshtein distance — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/wiki/Levenshtein_distance [Online; accessed 13-July-2020].