

Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Data Science
Fachgebiet Datenbanken und Informationssysteme

Ordering datasets for efficient or effective error detection

<Bachelorarbeit>

im Studiengang Informatik

von

Xinyue Gong


Prüfer: Prof. Dr. Ziawasch Abedjan
Zweitprüfer: Prof. Dr. Sören Auer
Betreuer: Fatemeh Ahmadi

Hannover, 14.07.2023

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende <Bachelorarbeit> selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 14.07.2023



Xinyue Gong

Zusammenfassung

Die Datenbereinigung ist entscheidend für eine genaue Datenanalyse und das Training von Modellen. Obwohl auf diesem Gebiet bereits viel geforscht wurde, konzentrieren sich die aktuellen Ansätze meist auf die Bereinigung eines einzelnen Datensatzes. Die Erkennung von Fehlern in einem Datenset bleibt eine Herausforderung. Raha ist ein hochmodernes Fehlererkennungssystem, das verschiedene Fehlertypen in einem gegebenen Datensatz ohne jegliche Vorkonfiguration erkennen kann. Außerdem bietet es eine Strategiefilterfunktion, die Strategien für einen neuen Datensatz auf der Grundlage bereinigter Datensätze aus der Vergangenheit auswählt, wodurch die Fehlererkennung effizienter wird. In dieser Arbeit versuchen wir, die Filterfunktion von Raha zu nutzen, um Fehler in einem kleinen Datenset effizienter zu erkennen. Nach einer detaillierten Analyse der Filterstrategie von Raha entwickeln wir eine Scoring-Methode, welche die am besten geeigneten Datensätze identifiziert, die zum Filtern der anderen Datensätze im Set verwendet werden. Experimente zeigen, dass unsere Methode die Effizienz der Fehlererkennung erheblich verbessern kann, während sie das gleiche Maß an Effektivität beibehält wie die Erkennung ohne Filterung.

Abstract

Ordering datasets for efficient or effective error detection

Data cleaning is crucial for accurate data analysis and model training. Although plenty of research has been done in this field, current approaches mostly focus on the cleaning of a single dataset. The task of detecting errors in a data lake remains a challenge. Raha is a state-of-the-art error detection system that can detect various error types in a given dataset without any pre-configuration. It also provides a strategy filtering feature that prunes strategies for the new dataset based on cleaned datasets from the past, which makes error detection more efficient. In this thesis, we try to leverage Raha's filtering feature to detect errors in a small data lake more efficiently. After analyzing Raha's strategy filtering in detail, we come up with a scoring method that identifies the most suitable datasets that will be used to filter the other datasets in the lake. Experiments show that our method can greatly improve the efficiency of error detection while maintaining the same level of effectiveness as detection without filtering.

Contents

1	Introduction	1
1.1	Problem statement	2
1.2	Solution approach	2
1.3	Technical challenges	3
2	Fundamentals	5
2.1	Raha workflow	5
2.2	Automatic strategy generation	6
2.3	Strategy filtering	7
2.3.1	Preparations for filtering	7
2.3.2	Steps of filtering strategies for a new column	8
3	Methodology	11
3.1	A complete detection plan	11
3.2	The choice of historical datasets	12
3.2.1	Using only the filtered dataset for filtering	13
3.2.2	Using both starting and filtered datasets for filtering	14
3.3	The scoring method	15
3.3.1	From column similarity to dataset similarity	16
3.3.2	Dataset similarity score	17
3.3.3	Weight	18
3.3.4	The choice of starting datasets	19
4	Experiments	21
4.1	System modification and experiment setup	21
4.2	Experiment 1: Random orders	22
4.2.1	Efficiency comparison	23
4.2.2	Effectiveness comparison	24
4.2.3	Analysis of selected strategies	25
4.3	Experiment 2: Using the filtered datasets for filtering	27
4.3.1	Effectiveness comparison	27
4.3.2	Modified Option 2: selecting top k strategies	29
4.3.3	Comparison of selected strategies	30

4.4	Experiment 3: Different choices of the starting dataset	32
4.5	Experiment 4: Evaluation of our scoring method	33
4.5.1	Evaluation on Raha datasets	33
4.5.2	Evaluation on Kaggle datasets	34
5	Conclusion and future work		37

Chapter 1

Introduction

Businesses and organizations often gather data from different sources to make informed decisions or derive insightful conclusions. However, the quality of the data obtained through external sources is often questionable. Research shows that data errors are common even in spreadsheets obtained from trustworthy sources such as professional organizations (like KPMG) and scientific journals [9, 13].

Using inaccurate data for analysis can have negative consequences, leading to financial losses in business and potentially severe consequences in fields such as healthcare[5]. To ensure data quality, companies may use in-house experts or turn to consultants for data cleaning. When dealing with high-value datasets, it is worth investing in expensive consulting as the rewards will justify the cost[12]. But with a data lake, which contains a vast amount of datasets with unknown values, it's unrealistic to expect a data expert to manually set up various error-detection tools for each dataset. In such cases, an ML-based error detection model which requires minimal human involvement will be a better choice.

Although much research has been done on automated error detection[7, 6, 12, 4], current approaches mainly focus on detecting errors in a single dataset. Cleaning data within a data lake remains an unresolved challenge[8].

Raha[7], one of the state-of-the-art error detection systems, is a semi-supervised approach that can detect various error types in a given dataset. No pre-configuration is required because Raha automatically generates error detection strategies based on different algorithms and possible configurations. When dealing with a large dataset, hundreds or even thousands of strategies might be generated, and running all these strategies can cause an efficiency problem. If error detection is done on other datasets, it is possible to filter out irrelevant strategies for a new column based on historical columns.

Regarding Raha's strategy filtering, ideally, historical data should contain datasets from the same domain. But the experiment in the paper shows even

with a set of diverse datasets, Raha can still find similar columns and perform filtering to improve the runtime.

Since a data lake also stores diverse data from different domains, Raha seems to have good potential to detect errors in a data lake in an efficient way.

1.1 Problem statement

Suppose we would like to detect errors in a data lake with diverse datasets using Raha, the simplest way is to run normal detection on each dataset. But since Raha allows us to filter strategies for the new dataset based on historical datasets, we try to make error detection more efficient with strategy filtering.

Raha’s filtering algorithm will select the most promising strategies from historical datasets for a new dataset. As each dataset has different columns and provides different possible strategy sets, we can imagine the selected strategies for the new dataset will vary based on the historical datasets chosen, which will further lead to different detection results. If we run error detection on multiple datasets sequentially, the order of the input table can affect the detection results over all datasets because the same dataset may benefit from different historical datasets in another order.

Raha’s experiment shows that filtering can improve the runtime at a cost of slightly lower effectiveness. While we aim to detect errors in a data lake more efficiently, we strive for the same level of effectiveness as running detection without filtering. Therefore, the goal of this thesis is to figure out:

1. whether the order of input tables is impactful to Raha’s detection results,
2. how to identify a reasonable order for more efficient and effective error detection.

1.2 Solution approach

To develop an ordering method for Raha, the first step is to understand the working mechanism of Raha and get familiar with its code.

Although Raha can filter strategies, we notice that with the original implementation, it is impossible to run error detection with filtering on a new dataset directly. The filtering requires the complete strategy profiles of the new dataset, which can only be generated by running a normal detection. But we want to avoid the generation of all possible strategies and only run the selected strategies on the new dataset. Besides, Raha was designed to detect errors in a single dataset, while we aim to detect errors in multiple

datasets with a given order. To conduct our experiments, we need to modify the system.

With the modified Raha, we plan to generate some random orders to figure out whether the order of the input table is impactful to the detection results. But we soon realize that there are two additional factors to consider if we want to run error detection on multiple datasets using Raha. We must determine on which datasets we run normal detection (without filtering) and on which we activate Raha’s filtering feature. When we enable strategy filtering, we also need to decide which datasets should be used as historical datasets to filter the new dataset.

Regarding the choice of the historical datasets, we analyze the possible advantages and the problems of using “filtered” datasets (i.e. datasets, on which we enable strategy filtering) based on Raha’s working mechanism. We also conduct experiments to compare two options of whether we use such filtered datasets for filtering or not. We figure out that using filtered datasets may lead to excessive filtering which decreases the effectiveness of error detection.

Regarding the choice of the “starting” datasets (i.e. datasets, on which we run normal detection), as these datasets provide the possible strategy set for the others to select from, we propose a scoring method to identify the most suitable starting datasets based on dataset similarity and size. We evaluate our method on two groups of datasets and only use the identified datasets for filtering. In both cases, we achieve similar effectiveness over all datasets as detection without filtering, with a boost in efficiency by 27% and 59%.

1.3 Technical challenges

The main technical challenge is the modification of the system as mentioned in Section 1.2. In addition, we need to implement our scoring method and write utility methods that help us analyze the selected strategies on each column.

Chapter 2

Fundamentals

This chapter introduces the workflow of Raha, with a focus on its automatic strategy generation and strategy filtering algorithm. This chapter provides the necessary theory fundamentals for analyzing our research questions and designing experiments.

2.1 Raha workflow

Figure 2.1 shows Raha’s workflow of error detection without filtering. We do not discuss each step in great detail as our experiments have a focus on strategy filtering and further explanation can be found in Raha’s paper. Nevertheless, we need to understand the basic working mechanism for further analysis.

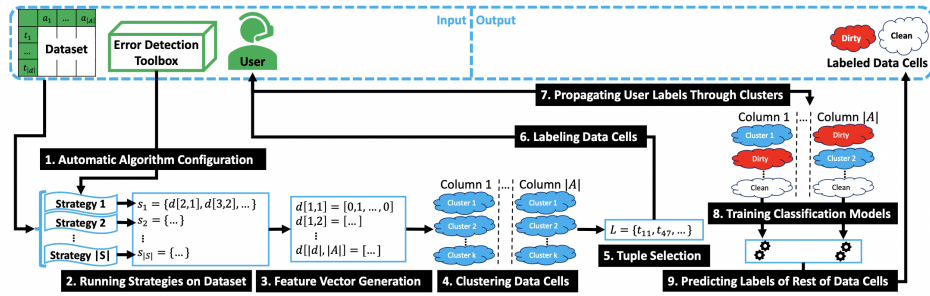


Figure 2.1: The workflow of Raha.

- **Step 1-4:** Raha generates and runs a set of error detection strategies to create feature vectors and build clusters.

No user involvement is required because Raha can generate strategies automatically. Details will be explained in Section 2.2.

Raha runs all the generated strategies, not to directly identify the errors in the given dataset, but to generate a feature vector for each data cell. This feature vector will be used for clustering data cells within each column. Data cells with the same or similar feature vector will be grouped into the same cluster. Raha applies hierarchical agglomerative clustering so that users do not need to specify the number of clusters.

- **Step 5-9:** Raha utilizes the labeled tuples and the column-based clusters to train a classification model for each column, and the trained classifiers predict the labels of the remaining data cells.

To limit user involvement, Raha only asks users to label a small number of tuples. Ideally, the sampled tuples should cover as many different clusters as possible over all the data columns. This tuple selection is clustering-based but not deterministic. With different sampled tuples, user labels will be propagated through different clusters, and the classifiers will be trained with different data. As a result, the performance of error detection may vary. Due to this probabilistic character of Raha, we will run error detection multiple times in our experiments to measure the effectiveness.

2.2 Automatic strategy generation

As mentioned in Section 2.1, Raha can automatically generate a set of error detection strategies. An error detection strategy is defined as a combination of an error detection algorithm and a configuration. Raha names a strategy in the format: [Algorithm type, [Configuration]].

Given a dataset, Raha generates and runs four types of error detection strategies:

- **Outlier detection (OD) strategies.** Both Histogram modeling and Gaussian modeling are used to detect outliers. For each algorithm, Raha has a predefined set of threshold values. A configuration for an OD strategy contains the algorithm name (Histogram or Gaussian) and a threshold value. The generated strategies are not column specified and will be run on the whole dataset. Raha utilizes dBoost^[10] to run OD strategies.
- **Pattern violation detection (PVD) strategies.** Instead of domain-specific data patterns, Raha leverages the bag-of-characters representation^[11] and uses the set of distinct characters which appear in data cells of a column as possible configurations. A PVD strategy can only be run on that specified column as the column name is part of the configuration.

As an example, suppose Raha generates a PVD strategy: [“PVD”, [“a1”, “-”]]. Running this strategy will mark all the data cells which contain “-” in column a1 as an error.

- **Rule violation detection (RVD) strategies.** Raha considers all pairs of columns as potential functional dependencies. A configuration contains the name of two columns. The order of column names is relevant as $a \rightarrow a'$ and $a' \rightarrow a$ are different functional dependencies.
- **Knowledge base violation detection (KBVD) strategies.** Given a knowledge base, each entity relationship inside that knowledge base is a possible configuration. A KBVD strategy will be run on the whole dataset as we try to find the columns which match the entities in the given relationship. Raha utilizes KATARA [2] to run KBVD strategies.

If we run normal detection without filtering, the above mentioned strategies will be generated and run on the dataset. As the size of the dataset grows, running all these generated strategies can cause a runtime problem. Although running more strategies can enrich the feature vectors and make them expressive, some features could be “irrelevant” and do not contribute to error detection. Therefore, Raha provides a strategy filtering feature that prunes irrelevant strategies based on cleaned datasets to make error detection more efficient.

2.3 Strategy filtering

Raha can filter strategies for the new dataset based on cleaned historical datasets. We explain the workflow of strategy filtering in detail as it is the focus of our research.

2.3.1 Preparations for filtering

To perform strategy filtering on a new dataset, Raha requires (1) the dataset profile of the new dataset and all the historical datasets, and (2) the evaluation profile of all the historical datasets. Therefore, there are two preparation steps for filtering:

- **Preparation 1: Profiling datasets.** Raha profiles both the new dataset and all historical datasets. When profiling a dataset, for each column Raha generates one dictionary which contains both character and value distribution in this column.
- **Preparation 2: Evaluating strategies.** For each historical dataset, Raha calculates for each column the precision, recall, and F1 score of every error detection strategy that was run on this dataset. The F1 score is relevant for the filtering.

2.3.2 Steps of filtering strategies for a new column

Raha filters and selects strategies based on the columns and groups them as a strategy set for the new dataset. Therefore, we explain the workflow of filtering strategies for a new column.

- **Step 1: Calculating column similarity.** For each column in the new dataset, Raha calculates all the pairwise cosine similarities between the respective column and every historical column (i.e. every column from all historical datasets). This similarity calculation is based on the column profiles generated in Preparation 1, i.e., character and value distribution.
- **Step 2: Scoring strategies.** For each column in the new dataset, imagine that every strategy from every historical column is a “candidate” that might be selected for this new column. Naturally, we want to select the best candidates, i.e. the strategies that might perform well on the new column. To find the best strategies for the new column, Raha will “adapt” (explained further in Step 2*) a strategy s from historical column j for the new column j and assign a score to the updated strategy s' . This score is the product of **column similarity** and the **F1 score** of strategy s on its historical column j .

$$score(s') = sim(d_{new}[:, j], d[:, j']) \times F(s, d[:, j']) \quad (2.1)$$

This score indicates how promising the strategy s' is for the new column. We call it **the promising score** in this thesis to distinguish it from the F1 score.

It is possible that multiple historical columns provide the same strategy s' for the new column.

For example, suppose we are selecting strategies for the new column a from two historical columns x and y . Column x provides a strategy s_1 and column y provides a strategy s_2 , but s_1 and s_2 lead to the same updated strategy s' on the new column a . In this case, Raha calculates the promising score based on both historical columns and keeps the higher one together with its historical column. Assume that the scores calculated based on column x and column y are 0.3 and 0.5, the promising score of strategy s' on the new column a will be 0.5, and this strategy will be considered as selected from column y .

- **Step 2*: Adapting strategies.** This step can be considered as a part of step 2. Each strategy from a historical column will be updated for the new column. While OD strategies and KBVD strategies remain the same, PVD and RVD strategies will be adapted.

The adaption of the PVD strategy is straightforward. Since the configuration of the PVD strategy contains both a column name and a character, Raha will adapt the column name, which means the historical column name will be replaced by the new column name.

The adaption of the RVD strategy requires us to find another column in the new dataset. Suppose a historical dataset A has an RVD strategy “a1->a2”. If this strategy is selected for column b1 in dataset B, Raha adapts this RVD strategy for dataset B by finding the most similar column to a2 in B. If the column found is b2, the adapted strategy will be “b1->b2”.

- **Step 3: Selecting the top-scored strategies.** After scoring each possible strategy provided by all historical columns, Raha sorts the strategy set S'_j . Instead of specifying a number indicating how many strategies should be picked per column, Raha selects the most promising strategy set S_j^* by applying a gain function used in the literature [1].

$$gain(S_j^*) = \sum_{s \in S_j^*} score(s) - \frac{1}{2} \sum_{s \in S_j^*} \sum_{s' \neq s \in S_j^*} |score(s) - score(s')| \quad (2.2)$$

This approach is threshold-free as Raha keeps adding the top-scored strategy to the promising strategy set until adding the next strategy leads to a decrease in gain.

Notice that the strategy selection process is also a “filtering” process. Concretely, some PVD strategies might be filtered out if the relevant character does not appear in the new column.

For example, if the data cells of the new column only contain numbers 1-9, while the next most promising strategy is a PVD strategy with the number “0”, we do not add this strategy to the selected strategy set. It is possible but meaningless to run this strategy on the new column as we know no data cell will be marked as an error. Raha even removes such non-informative features that are constant for all the data cells after generating feature vectors.

Chapter 3

Methodology

In this chapter, we try to analyze the questions we encounter during our research based on Raha’s working mechanism. During our analysis, we get inspiration for experiment design and we come up with a possible solution in the end.

3.1 A complete detection plan

As stated in Section 1.1, the goal of this thesis is to figure out whether the input order of datasets is impactful to Raha’s detection results and how we can identify a reasonable order for more efficient and effective error detection.

The possible order of a group of tables can simply be interpreted as the permutation. However, when using Raha for error detection on datasets, especially when we try to utilize the filtering feature, there are additional factors to consider beyond just the order.

1. Number of starting datasets

The starting datasets refer to the datasets on which we run normal detection at the beginning.

To filter strategies for the new dataset, Raha requires at least one historical dataset, which means we need to run normal detection on at least one dataset. But can we achieve a good result by only using one starting dataset?

In the strategy filtering impact analysis of Raha, the authors set up the experiment in such a way that for each run, only one dataset is considered as the new dirty dataset, and the rest of the datasets is the set of historical datasets. In other words, they filter one dataset based on many historical datasets. As we can imagine, more datasets provide more historical columns and more strategies to select from. It is therefore more likely for each column in the new dataset to find a good set of promising strategies from similar historical columns.

As the number of datasets grows, the idea of only using one dataset to filter becomes more unrealistic. But since our experiments are conducted on a small group of datasets, we assume it is possible to use only one dataset to filter the others. We try to stick to this minimum number in our experiments and solution design as it seems more efficient. If it sacrifices the effectiveness of error detection, we may raise the number.

2. The choice of historical datasets

When we run error detection on multiple datasets in sequential order, we might naturally come to the idea that all the previous datasets should be treated as historical datasets to the next dataset.

For example, suppose we decide to run error detection on three datasets with the order: $A \rightarrow B \rightarrow C$. After we run normal detection on A, A is the historical dataset for B. After we run filtering detection on B, both A and B are historical datasets for C. However, as explained in Section 2.3.2, the idea of strategy filtering is to select the most promising strategies from the historical datasets. In other words, the strategy set for the new dataset can be considered as a subset of the strategy set from historical datasets. If B's strategy set is only a subset of A, is it still meaningful to use both A and B to filter the new dataset C? Will we benefit more by also using B for filtering? In fact, there are three possible options regarding the choice of historical datasets for C, which will be further discussed in Section 3.2.

With the above discussion, we realize that we are not simply looking for an "order" of datasets in our research. Suppose we have a group of datasets [A, B, C, D, E]. Even if we decide to run error detection in this given order, our plan is not complete yet.

A complete detection plan could be: Run error detection in the order $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$. On A and B we run normal detection (which makes the order of A and B irrelevant). Starting from the detection for C, we enable the filtering feature and always use all the previously detected datasets as historical datasets.

We design an experiment (Section 4.2) to answer the first research question by generating some random plans and comparing the runtime and F1 score of error detection. If there are noticeable differences, we can consider that the "order" is impactful.

3.2 The choice of historical datasets

The discussion in Section 3.1 leaves us with an open question and we reformulate it into the following example:

Suppose we have datasets [A, B, C]. We run normal detection on A and use A to filter strategies for B. Since B does not generate strategies but only selects and adapts strategies from A, we call B a “**filtered**” dataset.

Assume that A generated 500 strategies and B selected 60 strategies from A. To detect errors in C using the filtering feature, we have the following three options regarding the choice of historical datasets:

1. Only use A (500 strategies).
2. Only use B (60 strategies).
3. Use both A and B (500+60 strategies).

Which of these three options is the best?

3.2.1 Using only the filtered dataset for filtering

We might first rule out the second option where we only use the filtered dataset B for filtering. These 60 strategies are selected for B because they seem to be promising, but there is no guarantee that they will work well on B.

Consider an extreme case if there is no error in B at all, these 60 strategies will have an F1 score of zero. As explained in section 2.3.2, the filtering is based on the promising score, which is the product of column similarity and the strategy’s historical F1 score. If the historical F1 score on B is zero, the strategy will have a promising score of zero and will not be selected for the new dataset. So C will end up with no strategy in this case, even though many strategies might work well on C. If we use A or both A and B for filtering, we can avoid such situations where a “promising” strategy becomes “useless” only because it fails to find errors in a filtered dataset.

In general, we can ignore such an order where we only use the last detected dataset to filter the new one. We can imagine that orders like “A (normal detection) -> B (filter with A) -> C (filter with B) -> D (filter with C)...” **usually lead to fewer and fewer strategies** for the new dataset. As explained in Section 2.1, Raha does not use the selected strategies to detect errors but to generate feature vectors for clustering. If we have an extremely small number of strategies, the feature vector may not be expressive enough, leading to a potential decrease in the effectiveness of error detection.

Notice that there is one extreme case in which the new dataset **might have “more” strategies** than the historical dataset. Raha counts strategies with distinct names and the strategy name is in the format [Algorithm type, [Configuration]].

As explained in Section 2.2, PVD strategies have a configuration like [“column name”, “character”]. Suppose we have a filtered dataset X with

only one column x (which contains the character “1” in its data cells) and only one strategy s : $["PVD", ["x", "1"]]$. If we use X to filter dataset Y which has two columns $y1$ and $y2$ (both columns also contain the character “1”), Raha might select strategy s for both columns in Y , which leads to two different strategies $s1'$: $["PVD", ["y1", "1"]]$ and $s2'$: $["PVD", ["y2", "1"]]$. These two strategies have different names and will be considered as different strategies. So after filtering, we have two strategies for Y while dataset X , which can be considered as the strategy provider, has only one strategy.

3.2.2 Using both starting and filtered datasets for filtering

The remaining question is whether Option 1: A (500 strategies) or Option 3: $A+B$ (500+60 strategies) is better.

Intuitively, we might prefer Option 3 for the following reasons:

1. **B can provide new strategies.**

Although the 60 strategies can be considered a “subset” of the 500 strategies, B can have “new” strategies due to the adaption of RVD strategies. If an RVD strategy $["RVD", ["a1", "a2"]]$ is selected for column $b1$ in B , this strategy might be adapted to $["RVD", ["b1", "b2"]]$, which could be an even more promising strategy for C .

2. **B can provide more columns or even “better” columns.**

The idea of Raha’s filtering is to select promising strategies from similar historical columns and we assume strategies selected from a more similar column may lead to a better result. So if B has columns that are more similar to the columns in C than A , C will benefit from filtering with both A and B since C can find “better” strategies from “better” columns.

With the above analysis, we believe filtering with both datasets A and B can lead to a more “precise” strategy set for dataset C . While it is true that this strategy set contains the most effective strategies from the most similar historical columns in A and B , it does not necessarily mean we will achieve better effectiveness than using only A to filter.

Consider the following example:

B has a strategy s' for column $b1$, and s' was originally strategy s from column $a1$ in A .

Selecting s or s' for the new column $c1$ will lead to the same strategy, and the adapted strategy on $c1$ will be represented by s'' . As explained in Section 2.3.2, Raha calculates the promising score of strategy s'' based on both historical columns and keeps the higher score and its historical column.

After filtering, suppose strategy s'' is selected and has a good promising score of 0.8 with historical column $b1$. While we do not know what the

score calculated based on a_1 is, it must be lower than 0.8. That’s why Raha is choosing s' from b_1 rather than s from a_1 . Since the promising score calculated based on s' is higher, it seems that we are choosing a more promising strategy from a better column. But the truth is: s' was originally adapted from s , so s' and s are in essence the same strategy.

The promising score is the product of column similarity and F1 score, so the reason for strategy s' having a higher score than s could be: (1) column c_1 is more similar to b_1 than a_1 ; (2) s' on b_1 has a better F1 score than s on a_1 .

As mentioned above in reason 2, we would prefer to select strategies from a more similar column. But when this “more similar column” is from a filtered dataset, we are selecting strategies from a very limited set that only contains the strategies which “seem to be promising” based on some other historical datasets. So it is questionable how much we can gain from this “more similar column”.

In addition, although seeing the same strategy working even better on another column will give us more confidence and thus make this strategy seem more “promising”, we should realize this does not make the strategy “better”. We might be more and more optimistic about a few strategies, believing they are so good that we ignore many others which could also be helpful. We design an experiment (Section 4.3) to compare the differences between whether we use the filtered datasets for filtering or not.

If we only use the starting datasets on which we run normal detection for filtering, the order of datasets becomes irrelevant. Instead of an order, we are looking for the most suitable datasets to start with, which will later be used to filter the rest of the datasets.

We decide to first focus on the choice of starting datasets. Because even if we use every detected dataset for filtering, we will face the same problem of choosing a starting point as it is part of the order.

3.3 The scoring method

We design an experiment (Section 4.4) to see whether the different choice of the starting dataset is impactful to the result. As we can imagine, this choice must be important.

Intuitively, we won’t choose a dataset that only contains a few columns to filter all the other datasets that contain many more columns. Consider the extreme case when we use a dataset with only one column to filter, all the columns of the other datasets will be forced to select strategies from this single historical column despite a potentially bad column similarity. This inspires us: we are probably looking for a dataset that can be considered to have a good similarity to the rest of the datasets. To this end, we propose a scoring method to find such a dataset.

3.3.1 From column similarity to dataset similarity

In the previous sections of this chapter and in Chapter 2, our discussion has remained at the level of column similarity. The idea of filtering is to select and adapt strategies for a new column from similar historical columns, and the calculation of the promising score is also partly based on column similarity.

Because Raha filters based on the columns, one idea might be to **split the datasets into columns**. Instead of using a whole dataset to filter another, we use one column to filter some other columns that have a high column similarity to this one. We split and clean the columns in a dataset separately and later group the cleaned columns so that we have a cleaned dataset.

One problem with this idea is that, by splitting a dataset, we can not benefit from RVD and KBVD strategies anymore. RVD strategies may find errors in columns where functional dependencies are violated while KBVD strategies may find errors in columns that match the entities in a relationship from the knowledge base. As we do not want to ignore the (hidden) relationship between columns in a dataset, we decide to keep the dataset as a whole. Besides, when users are asked to label a sampled tuple but only see a split column, they might have problems identifying whether this is an error or not because they need information from other columns of the original tuple.

Since we decide to detect errors in a whole dataset and use one dataset to filter another, calculating column similarity is not enough anymore. We wonder how similar the two given datasets are. In other words, we need a method to calculate the dataset similarity.

We can first calculate the pairwise column similarity. Although we might come up with some new ideas about how to measure column similarity, we stick to Raha's approach and calculate the cosine similarity based on the character and value distribution in the columns. As the authors explained in Raha's paper, the syntactic similarity between two columns can be captured by the similarity of character distribution and the semantic similarity can be captured by the overlap of the actual values, i.e., the value distribution. This representation helps us find syntactic and semantic data errors and seems reasonable.

Now we need to calculate the dataset similarity based on the column similarity.

Consider the example where we have two datasets A and B.

- Column set of A: $Col(A) = \{a_0, a_1, \dots, a_m\}$
- Column set of B: $Col(B) = \{b_0, b_1, \dots, b_n\}$

A simple idea for calculating the dataset similarity between A and B could be **to add up all the pairwise column similarities**.

$$Sim(A, B) = \sum_{a_i \in Col(A), b_j \in Col(B)} sim(a_i, b_j) \quad (3.1)$$

While this idea seems naive, it might be a good choice.

If Raha first finds the most similar historical column and selects the most effective strategies from this column, we should only focus on the maximum column similarity. For example, suppose $sim(a1, b1) = 0.5$, $sim(a1, b2) = 0.7$, we can ignore the similarity between $a1$ and $b1$ when selecting strategies for $a1$, because the more similar column $b2$ will be preferred. But the reality is, Raha scores the strategy based on both column similarity and the strategy's F1 score on the historical column. All the possible strategies for the new column will be assigned a promising score first, then Raha selects the most promising strategies using a gain function. As this is a threshold-free approach, we do not know how many strategies will be selected. This filtering mechanism leads to two results: (1) The selected strategies may come from multiple historical columns. (2) The selected strategies may not come from the most similar column.

In the above example, although $sim(a1, b2) = 0.7$ is greater than $sim(a1, b1) = 0.5$, we can not assume that $a1$ will select strategies from $b2$ instead of $b1$. If there is no error in $b2$, no strategy from $b2$ can be selected as the F1 score is zero. In this case, Raha will select strategies from $b1$. It is also possible that Raha selects strategies from both $b1$ and $b2$ when for example two strategies have the same promising score.

As long as the similarity between the new column and the historical column is not zero, strategies of this historical column might be selected. That's why using the sum of pairwise column similarity might be a good idea. It treats each similarity as "possible". The value of similarity also implies how good the possible choice is. When two columns have a similarity of zero, it is impossible to filter using these two columns. Selecting strategies from a more similar column is always better than from a less similar one.

3.3.2 Dataset similarity score

We proposed a possible representation for dataset similarity in Section 3.3.1. Now we try to score each dataset based on this idea.

Suppose we have a data lake $\mathcal{L} = \{D_1, D_2, D_3, \dots\}$ and we have calculated the pairwise dataset similarity.

For each dataset, we calculate a "similarity score" by summing up its similarity to the rest of the datasets in the lake. Take the dataset D_1 in the lake as an example:

$$SimilarityScore(D_1) = \sum_{D_k \in \mathcal{L}, D_k \neq D_1} Sim(D_1, D_k) \quad (3.2)$$

Another idea is to calculate the average similarity, which means we divide the sum by the number of datasets in the lake minus one. As we are interested in finding the dataset with the highest score and either choice leads to the same result, we use the sum for simplicity.

We believe the dataset with the highest score is the one that has a relatively good similarity to the others. We hope that all the rest of the datasets can find some columns with good similarity so that the error detection can achieve relatively good effectiveness. But since our objective is to make error detection “more efficient and effective”, we consider introducing a weight to this similarity score which might improve the efficiency of error detection.

3.3.3 Weight

The weight should reflect the efficiency of choosing one dataset to filter the others. The feature extraction process on large datasets is time-consuming, so a large dataset can be considered as “inefficient”. Naturally, we would prefer to use smaller datasets to filter the larger ones.

Consider the following example where two datasets (A and B) in a data lake have a close similarity score:

- $SimilarityScore(A) \approx SimilarityScore(B) = 100$
- $\#Tuple(A) = 1000$
- $\#Tuple(B) = 7000$
- $\#Tuple(lake) = 14000$

If we use A to filter the remaining datasets in the lake, we will be using one dataset to filter about 93% [= (14000 - 1000) / 14000] of tuples in the data lake. If we use B to filter, we will be using one dataset to filter only 50% of tuples. In this case, we should prefer A to B because this seems more efficient.

We consider using this percentage as a weight to punish the larger dataset. The weight for a dataset D in a data lake \mathcal{L} can be represented by:

$$Weight(D) = \frac{\#Tupel(\mathcal{L}) - \#Tupel(D)}{\#Tupel(\mathcal{L})} \quad (3.3)$$

Further, by multiplying the similarity score and the weight, we can calculate the final score for D:

$$FinalScore(D) = SimilarityScore(D) \times Weight(D) \quad (3.4)$$

We calculate the final score for A and B in our example: $FinalScore(A) = 93 > FinalScore(B) = 50$. We notice that A now has a much higher

score than B since it has a great advantage in the aspect of efficiency. If two datasets have the same number of tuples, applying this weight will not change the preference between them.

This weight can be meaningful because it reflects efficiency, but it is debatable whether we should apply this weight to all the datasets. If we apply this weight only in cases like the above where we have a few similar “top candidates”, this weight can help us improve efficiency without sacrificing effectiveness. But if we apply this weight to all the datasets, it is possible that the “top datasets” with a high similarity score can not keep the top ranking anymore, and instead, a “somewhat both efficient and effective” dataset will be preferred. This trade of effectiveness for efficiency is not what we want to achieve. As long as we are utilizing the filtering feature of Raha, we have already improved the efficiency to a certain extent, so we should probably prioritize effectiveness in our solution design.

3.3.4 The choice of starting datasets

As discussed in Section 3.1, the minimum number of starting datasets is one, but using more starting datasets may improve the effectiveness of error detection. The ideal number of starting datasets can be a question and will certainly depend on the size of the data lake. Without a large number of datasets, we can not conduct further experiments to find a method to identify the optimal number. However, with the consideration of prioritizing effectiveness as mentioned in Section 3.3.3, we suggest raising the minimum number of starting datasets to two.

While the dataset with the highest similarity score is valuable and should be chosen as the starting dataset, we might also want to run normal detection on the dataset with the lowest similarity score.

There are two possible reasons why a dataset achieves the lowest score:

1. **This dataset has relatively fewer columns.** The similarity score of a dataset is the sum of all the relevant dataset similarities, which are the sum of pairwise column similarities. If a dataset has few columns, there will be fewer summands that can contribute to the calculation of the similarity score.
2. **The columns of this dataset are somewhat “unique”.** If the columns of this dataset are not similar to most columns from the other datasets, most summands are relatively smaller, which will lead to a low score. If this is the case, we should be more cautious about this dataset. It might not be able to benefit much from filtering because it can not find enough similar columns from other datasets. Therefore, we should run normal detection on this dataset as filtering is more likely to cause a significant decrease in the effectiveness of error detection.

Based on the discussion in Section 3.3, we design an experiment (Section 4.5) to evaluate our proposed scoring method.

Chapter 4

Experiments

This chapter includes all the experiments inspired by the discussion in Chapter 3.

4.1 System modification and experiment setup

The original Raha generates and runs all the strategies for the new dataset before filtering the strategies. We modify the system¹ so that Raha filters the strategies first and only runs the selected strategies on the new dataset.

There are four types of error detection strategies. RVD and PVD strategies contain column names in the configuration and will be run on the relevant columns. OD strategies contain only the model name and threshold value in the configuration, but we record the relevant new columns for each selected OD strategy and later run the strategies only on the recorded columns. KBVD strategies will be applied to the whole new dataset no matter for which new columns they are selected. Because KBVD strategies will try to find columns that match the entity relationship, we can not run KBVD strategies on a single column. A knowledge base can contain thousands of entity relationships, but in our experiment, we only include one relationship that is relevant to our datasets.

As the default parameter setting for error detection, we set the user labeling budget to 20 and use Gradient Boosting^[3] as the classification model. While we stick to the hierarchical clustering approach, using the default cosine similarity metric and the average linkage method, we add one extra column with all ones to the generated feature vectors to avoid having all zero vectors. That might be problematic in calculating the distance in cosine similarity. But the extra feature will not be counted in the feature number of each column. This modification will not affect the performance of error detection.

¹<https://github.com/citrusqwq/raha-ordering>

To measure effectiveness, we report precision, recall, and F1 score. Because Raha’s tuple sampling is probabilistic, we calculate the average of 10 runs. To measure efficiency, we report strategy runtime because it dominates the runtime of error detection and the goal of filtering is to speed up this feature extraction process. We also report the number of selected strategies, the number of extracted features, and the filtering runtime in some experiments.

In addition to reporting the effectiveness and efficiency of individual datasets, we also report the precision, recall, and F1 score over all datasets based on the accumulated number of output errors, true positives, and actual errors in each dataset.

Our experiments are mostly conducted on the five datasets (as shown in Table 4.1) that are available in Raha’s project repository. The number of strategies refers to the number of automatically generated strategies if a normal detection is run on the dataset. We rely on this small group of datasets to analyze and we will test our proposed method on another group of datasets in the last experiment. We run all the experiments on a MacBook Air with 8-core CPU and 16GB memory.

	#Columns	#Tupels	#Strategies	#Cells with error
Beers	11	2410	511	4362
Flights	7	2376	309	4920
Hospital	20	1000	776	509
Movies_1	17	7390	1246	7675
Rayyan	11	1000	686	948

Table 4.1: Dataset characteristics.

4.2 Experiment 1: Random orders

In this experiment, we try to figure out whether the order of input tables is impactful to the detection results.

As discussed in Section 3.1, to run error detection on multiple datasets, we need a complete detection plan which contains more than the order of input tables. We have five datasets, the minimum number of starting datasets is one and the maximum number is four, so we generate four random orders and run normal detection on the first one to four datasets. For Order 1 and Order 2, we use all the previously detected datasets for filtering. For Order 3 and Order 4, we do not use filtered datasets for filtering.

- Order 1: [**rayyan**’->’beers’->’flights’->’movies_1’->’hospital’]
- Order 2: [**beers**’, **flights**’->’hospital’->’rayyan’->’movies_1’]
- Order 3: [**movies_1**’, **flights**’, **rayyan**’, ‘hospital’, ‘beers’]

- Order 4: ['hospital', 'flights', 'rayyan', 'beers', 'movies_1']

4.2.1 Efficiency comparison

Figure 4.1 shows the comparison of the overall strategy runtime (in seconds) of four orders. To explain the huge differences in runtime, we also report the number of strategies run on each dataset in Table 4.2. If normal detection is run on the dataset, the number is colored green.

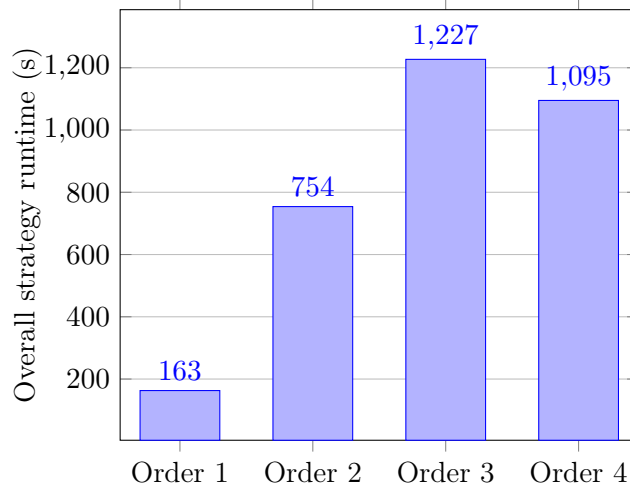


Figure 4.1: Runtime comparison of four orders.

	Beers	Flights	Hospital	Movies_1	Rayyan
Order 1	44	23	21	28	686
Order 2	511	309	51	65	30
Order 3	59	309	48	1246	686
Order 4	511	309	776	73	686

Table 4.2: Strategy number comparison of four orders (on each dataset).

Order 1 is the most efficient plan in comparison, as we only run normal detection on one dataset, which is also small in terms of size (11 columns, 1000 tuples). Only a small set of strategies is run on the other datasets, which improves the runtime significantly.

The runtime has an increasing tendency as the number of normal detection datasets grows. But by comparing Order 3 and Order 4, we know it is not the number but the size of normal detection datasets that matters more.

Movies_1 is the largest dataset (17 columns, 7390 tuples). Without strategy filtering, it generates 1246 strategies. In Order 4, we run normal

detection on four datasets and use them to filter Movies_1, which is more efficient than using three datasets (while one of them is Movies_1) for filtering in Order 3.

Based on the above analysis, we can infer that given a data lake, the most efficient error detection plan is to run normal detection on the smallest dataset and use it to filter all the rest of the datasets.

Furthermore, if we compare the selected strategy number of Beers and Movies_1, it seems that the more normal detection datasets we use for filtering, the more strategies will the filtered datasets select. But we find an exception on the dataset Hospital with Order 2 and Order 3. Because of this exception, we can not draw any conclusion on the relationship between the number of normal detection datasets and the number of selected strategies yet.

4.2.2 Effectiveness comparison

Table 4.3 shows the precision, recall, and F1 score of error detection with these four orders. If normal detection is run on the dataset, the number is colored green.

	Beers			Flights			Hospital			Movies_1			Rayyan			Overall		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
Order 1	0.99	0.83	0.90	0.73	0.55	0.59	0.69	0.21	0.30	0.87	0.85	0.86	0.83	0.76	0.79	0.85	0.74	0.79
Order 2	0.99	0.98	0.98	0.82	0.84	0.83	0.81	0.34	0.46	0.88	0.79	0.83	0.74	0.80	0.76	0.87	0.84	0.85
Order 3	0.97	0.87	0.92	0.85	0.78	0.81	0.72	0.27	0.37	0.86	0.90	0.88	0.81	0.73	0.76	0.87	0.83	0.85
Order 4	0.97	0.99	0.98	0.84	0.81	0.82	0.94	0.61	0.73	0.79	0.83	0.80	0.79	0.78	0.78	0.84	0.85	0.84

Table 4.3: Effectiveness comparison of four orders.

Raha’s tuple sampling is probabilistic, so we calculate the average of 10 iterations to measure the effectiveness as mentioned in Section 4.1. But if we compare the green-colored numbers on the dataset Rayyan or Flights, we notice there can be a difference of up to 6% in terms of recall, 4% in terms of precision, and 3% in terms of F1 score, even though we are running normal detection in all these cases.

Since we do not have a known tolerance range, it can be difficult for us to tell whether the effectiveness suffers from strategy filtering or not. One idea might be to use the same set of tuples to avoid fluctuations in results due to the randomness in tuple sampling. But Raha’s tuple sampling process is also clustering-based. Filtering strategies will lead to different clustering results, so we can not specify a set of tuples and have to live with the randomness.

Although it is difficult to identify the slight decrease in effectiveness caused by filtering, the bad result with Order 1 is obvious. The effectiveness drops slightly on Movies_1 and Beers, and significantly on Flights and Hospital. It seems that using one dataset to filter is not enough.

Order 2, Order 3, and Order 4 have achieved similar precision, recall, and F1 score over all datasets. By comparing the filtered results of Movies_1,

we can conclude that filtering with more datasets does not necessarily lead to a better result.

Overall, the comparison of efficiency and effectiveness clearly shows that the order of input tables is impactful to the detection results.

4.2.3 Analysis of selected strategies

We further analyze the selected strategies on Movies_1 with Order 4 to deepen our understanding of Raha's filtering feature. Table 4.4 shows the selected strategies for each column.

Column Name	Number of Strategies	Historical Column (number)	Strategy Type (number)
id	36	hospital.zip(1) hospital.phone(35)	PVD(1) KBVD(1) OD(34)
name	27	hospital.name(1) beers.city(26)	PVD(1) OD(26)
year	35	rayyan.article_jvolumn(2) rayyan.article_jcreated_at(1) hospital.provider_number(32)	PVD(1) OD(34)
release date	29	hospital.address_1(1) beers.ounces(28)	RVD(9) PVD(5) OD(15)
director	27	hospital.name(1) beers.city(26)	PVD(1) OD(26)
creator	27	hospital.name(1) beers.city(26)	PVD(1) OD(26)
actors	27	beers.city(27)	PVD(1) OD(26)
full cast	27	beers.city(27)	PVD(1) OD(26)
language	27	hospital.city(1) beers.city(26)	OD(27)
country	35	beers.ibu(1) beers.state(34)	PVD(1) OD(34)
duration	31	beers.ounces(12) flights.sched_dep_time(2) flights.act_dep_time(17)	RVD(10) PVD(4) OD(17)
rating value	11	flights.sched_dep_time(1) beers.abv(10)	OD(11)
rating count	35	hospital.phone(35)	KBVD(1) OD(34)

review count	26	beers.city(1) hospital.type(25)	OD(26)
genre	26	beers.city(26)	OD(26)
filming locations	26	beers.city(26)	OD(26)
description	27	hospital.measure_name(1) beers.city(26)	PVD(1) OD(26)

Table 4.4: Selected strategies for Movies_1.

We use all the other four datasets to filter Movies_1, and we notice that the historical columns are indeed from different datasets. Interestingly, the most frequently selected column seems to be beers.city. This column does contain a rich variety of (city) names. Columns like name, director, creator, actors, full cast, language, genre, and filming locations in Movies_1 can also be considered as columns that contain “names”. So it somehow makes sense that beers.city is selected for all these columns. The character distribution of these columns can be very similar.

Based on the demonstrated example we also noticed that when Raha selects strategies from two or three historical columns for some columns, it often selects a great number of strategies from one column and only one strategy from another. We take the selected strategies for the column year as an example (as shown in Figure 4.2) to explain this filtering result.

The “sf_score” means “similarity*f1_score”, which is considered as the promising score in this thesis. The “prf” shows the strategy’s precision, recall, and f1 score on this new column.

The strategies selected from rayyan.article_jvolumn have the highest promising score of 0.882. The 32 strategies selected from hospital.provider_number have a lower promising score of 0.871. The only strategy selected from rayyan.article_jcreated_at has the lowest promising score of 0.603. Raha only selects one strategy from this column because the gain function has reached a local maximum after adding this strategy to the strategy set.

The reason why the selected strategies from the same historical column all have the same promising score is that all the selected strategies have an F1 score of one. The three scores 0.882, 0.871, and 0.603 indicate the column similarities.

4.3. EXPERIMENT 2: USING THE FILTERED DATASETS FOR FILTERING27

```

On new column movies_1.year, 35 strategies are applied:
["OD", ["histogram", "0.3", "0.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "1.7"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.029, 0.184, 0.05]
["OD", ["histogram", "0.5", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["PVD", ["year", "1"]], hc: rayyan.article_jcreated_at, sf_score: 0.603, prf:[0.027, 0.821, 0.052]
["OD", ["histogram", "0.7", "0.7"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "1.0"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.029, 0.184, 0.05]
["OD", ["histogram", "0.3", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.1", "0.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.1"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "3.0"]], hc: rayyan.article_jvolumn, sf_score: 0.882, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.7"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "1.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.029, 0.184, 0.05]
["OD", ["histogram", "0.5", "0.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.1"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "1.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.029, 0.184, 0.05]
["OD", ["histogram", "0.1", "0.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "2.7"]], hc: rayyan.article_jvolumn, sf_score: 0.882, prf:[0.042, 0.047, 0.045]
["OD", ["histogram", "0.1", "0.7"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "2.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.047, 0.068, 0.056]
["OD", ["histogram", "0.1", "0.1"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.7"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.7", "0.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.7", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.7", "0.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.7"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.3"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.1"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "2.0"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.029, 0.184, 0.05]
["OD", ["histogram", "0.7", "0.1"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["gaussian", "2.5"]], hc: hospital.provider_number, sf_score: 0.871, prf:[0.039, 0.053, 0.045]
["OD", ["histogram", "0.1", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]

```

Figure 4.2: Selected strategies for column year.

4.3 Experiment 2: Using the filtered datasets for filtering

In this experiment, we try to figure out whether using the filtered datasets for filtering is a better option than only using the normal datasets.

We generate a random order(normal detection run on the first dataset): ['beers', 'rayyan', 'flights', 'hospital', 'movies_1'] and compare the error detection results of these two options:

- #1: Using only beers to filter all the other datasets.
- #2: Using all the previously detected datasets to filter the next dataset.

4.3.1 Effectiveness comparison

Table 4.5 shows the effectiveness comparison of these two options.

	Beers			Rayyan			Flights			Hospital			Movies_1		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
#1	0.99	0.99	0.99	0.72	0.79	0.75	0.82	0.77	0.79	0.76	0.32	0.44	0.77	0.79	0.77
#2	0.99	1.00	1.00	0.75	0.76	0.75	0.81	0.78	0.79	0.51	0.19	0.27	0.62	0.73	0.66

Table 4.5: Effectiveness comparison of two options.

We do not compare the results on Beers and Rayyan because normal

detection is run on Beers and only Beers is used to filter Rayyan in both situation #1 and #2.

For the filtering on Flights, #1 uses one dataset (Beers) while #2 uses two datasets (Beers, Rayyan). The results of both options are similar. But on the last two datasets Hospital and Movies_1, the effectiveness of #2 is worse than #1. This difference can be explained by comparing the feature number on each column as shown in Table 4.6.

Flights			Hospital			Movies_1		
C	#1	#2	C	#1	#2	C	#1	#2
0	30	28	0	21	15	0	42	31
1	27	23	1	25	14	1	27	26
2	11	8	2	5	2	2	43	32
3	16	16	3	0	0	3	36	37
4	11	11	4	3	3	4	35	30
5	12	22	5	5	9	5	26	23
6	22	12	6	12	22	6	21	20
			7	25	18	7	28	24
			8	12	17	8	26	15
			9	20	20	9	19	18
			10	32	23	10	24	32
			11	9	9	11	21	20
			12	10	9	12	19	14
			13	15	10	13	11	10
			14	8	5	14	11	0
			15	0	4	15	15	4
			16	5	10	16	6	1
			17	11	6			
			18	4	1			
			19	0	0			

Table 4.6: Number of column features comparison of two options.

On Flights, the number of generated features on each column is similar in both situations #1 and #2. But on Hospital and Movies_1, #2 generates fewer features than #1 on most columns. As we know Raha later builds clusters based on the feature vectors, the poor performance of #2 can be attributed to the decrease in feature numbers. In fact, #2 selects 7 fewer strategies than #1 on Hospital and 30 fewer strategies on Movies_1. We assume that the difference in feature numbers can be traced back to the difference in strategy selection. #2 might have filtered out some helpful strategies on some columns.

4.3.2 Modified Option 2: selecting top k strategies

We conduct a further experiment to see whether the drop in effectiveness is due to missing strategies. We first record the number of strategies selected for each column in #1. Then we try #2 again with one modification: instead of using the threshold-free gain function to decide how many strategies we select for each column, we select the top k strategies where k equals the recorded number on that column in #1. Table 4.7 and Table 4.8 are the comparisons of #1 and modified #2.

	Flights			Hospital			Movies_1		
	P	R	F	P	R	F	P	R	F
#1	0.82	0.77	0.79	0.76	0.32	0.44	0.77	0.79	0.77
#2*	0.81	0.76	0.78	0.68	0.35	0.44	0.76	0.79	0.77

Table 4.7: Effectiveness comparison with modified Option 2.

Flights			Hospital			Movies_1		
C	#1	#2*	C	#1	#2*	C	#1	#2*
0	30	30	0	21	21	0	42	42
1	27	27	1	25	24	1	27	25
2	11	11	2	5	5	2	43	43
3	16	16	3	0	0	3	36	37
4	11	11	4	3	3	4	35	35
5	12	12	5	5	5	5	26	26
6	22	22	6	12	12	6	21	21
			7	25	25	7	28	27
			8	12	13	8	26	26
			9	20	20	9	19	19
			10	32	34	10	24	25
			11	9	9	11	21	21
			12	10	10	12	19	19
			13	15	15	13	11	10
			14	8	8	14	11	11
			15	0	0	15	15	15
			16	5	5	16	6	7
			17	11	12			
			18	4	4			
			19	0	0			

Table 4.8: Number of column features comparison with modified Option 2.

With this modification, we achieve similar performance and a similar feature number on each column. So the decrease in effectiveness is caused by missing helpful strategies.

4.3.3 Comparison of selected strategies

We decide to compare the selected strategies on a single column and we use column `movies_1.year` as an example (Figure 4.3 and Figure 4.4).

```
On new column movies_1.year, 29 strategies are applied:
["OD", ["histogram", "0.7", "0.5"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["id", "year"]], hc: beers.ounces, sf_score: 0.540, prf:[0.0, 0.0, 0.0]
["RVD", ["year", "name"]], hc: beers.ounces, sf_score: 0.542, prf:[0.023, 0.884, 0.044]
["OD", ["histogram", "0.9", "0.5"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["year", "rating_value"]], hc: beers.ounces, sf_score: 0.542, prf:[0.022, 0.853, 0.044]
["OD", ["histogram", "0.5", "0.9"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["year", "full_cast"]], hc: beers.ounces, sf_score: 0.542, prf:[0.023, 0.884, 0.044]
["RVD", ["rating_value", "year"]], hc: beers.ounces, sf_score: 0.534, prf:[0.005, 0.174, 0.009]
["OD", ["histogram", "0.7", "0.7"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["director", "year"]], hc: beers.ounces, sf_score: 0.541, prf:[0.024, 0.611, 0.047]
["RVD", ["full_cast", "year"]], hc: beers.ounces, sf_score: 0.528, prf:[0.091, 0.005, 0.01]
["PVD", ["year", " " ]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.9"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["year", "rating_count"]], hc: beers.ounces, sf_score: 0.542, prf:[0.023, 0.884, 0.045]
["OD", ["histogram", "0.9", "0.7"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["PVD", ["year", "1"]], hc: beers.ounces, sf_score: 0.540, prf:[0.027, 0.821, 0.052]
["OD", ["histogram", "0.7", "0.9"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.9"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.1", "0.7"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.5"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["year", "id"]], hc: beers.ounces, sf_score: 0.542, prf:[0.023, 0.884, 0.044]
["RVD", ["rating_count", "year"]], hc: beers.ounces, sf_score: 0.520, prf:[0.016, 0.163, 0.029]
["OD", ["histogram", "0.5", "0.7"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.7"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.5"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.1", "0.9"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["PVD", ["year", "0"]], hc: beers.ounces, sf_score: 0.541, prf:[0.033, 0.705, 0.062]
["OD", ["histogram", "0.1", "0.5"]], hc: beers.ounces, sf_score: 0.543, prf:[1.0, 1.0, 1.0]
["RVD", ["year", "director"]], hc: beers.ounces, sf_score: 0.542, prf:[0.023, 0.884, 0.044]
```

Figure 4.3: Selected strategies for column `year` with #1.

```
On new column movies_1.year, 5 strategies are applied:
["RVD", ["year", "rating_count"]], hc: rayyan.article_jcreated_at, sf_score: 0.600, prf:[0.023, 0.884, 0.045]
["OD", ["histogram", "0.5", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.7", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.1", "0.9"]], hc: hospital.provider_number, sf_score: 0.871, prf:[1.0, 1.0, 1.0]
```

Figure 4.4: Selected strategies for column `year` with #2.

When we use all the previously detected datasets to filter `Movies_1(#2)`, Raha only selects 5 strategies. These strategies are also selected in #1 (marked in red). The only difference is that the selected strategies in #2 have a better promising score (calculated based on a more similar column). As shown in Figure 4.3, there are many other strategies that can achieve a good F1 score on the new column `movies_1.year`, but they are somehow filtered out in #2. We assume that in #2, only these 4 OD strategies have achieved an F1 score of one on column `hospital.provider_number`.

If we further compare the selected strategies on column `hospital.provider_number` in #1 and #2 (Figure 4.5 and Figure 4.6), we notice that our assumption is correct. Only 4 OD strategies have a good F1 score in #2.

4.3. EXPERIMENT 2: USING THE FILTERED DATASETS FOR FILTERING31

```

On new column hospital.provider_number, 25 strategies are applied:
["OD", ["histogram", "0.1", "0.9"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.9"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.7", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["RVD", ["provider_number", "name"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.1", "0.5"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["RVD", ["city", "provider_number"]], hc: beers.ounces, sf_score: 0.552, prf:[0.041, 1.0, 0.08]
["RVD", ["name", "provider_number"]], hc: beers.ounces, sf_score: 0.574, prf:[0.054, 1.0, 0.103]
["OD", ["histogram", "0.7", "0.9"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.5"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.9"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.3", "0.9"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[1.0, 1.0, 1.0]
["PVD", ["provider_number", "0"]], hc: beers.ounces, sf_score: 0.574, prf:[0.015, 0.536, 0.03]

```

Figure 4.5: Selected strategies for column provider_number with #1.

```

On new column hospital.provider_number, 25 strategies are applied:
["OD", ["histogram", "0.3", "0.9"]], hc: rayyan.article_jcreated_at, sf_score: 0.596, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.1", "0.9"]], hc: rayyan.article_jcreated_at, sf_score: 0.596, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.9"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.7", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["RVD", ["provider_number", "name"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.1", "0.5"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.3", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["RVD", ["city", "provider_number"]], hc: beers.ounces, sf_score: 0.552, prf:[0.041, 1.0, 0.08]
["RVD", ["name", "provider_number"]], hc: beers.ounces, sf_score: 0.574, prf:[0.054, 1.0, 0.103]
["OD", ["histogram", "0.5", "0.5"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.9", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["PVD", ["provider_number", "0"]], hc: beers.ounces, sf_score: 0.574, prf:[0.015, 0.536, 0.03]
["RVD", ["provider_number", "city"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.7", "0.9"]], hc: rayyan.article_jcreated_at, sf_score: 0.596, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.5", "0.9"]], hc: rayyan.article_jcreated_at, sf_score: 0.596, prf:[1.0, 1.0, 1.0]
["OD", ["histogram", "0.9", "0.5"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]
["OD", ["histogram", "0.5", "0.7"]], hc: beers.ounces, sf_score: 0.576, prf:[0.0, 0.0, 0.0]

```

Figure 4.6: Selected strategies for column provider_number with #2.

Meanwhile, by comparing the other OD strategies and their F1 score on the new column, take the red strategy as an example, we notice that surprisingly the same strategy has a different F1 score.

The reason behind this difference is that Raha uses dBoost to run OD strategies which “flags suspicious fields in database tuples”. This means OD strategies are not run on a single column separately. As mentioned in Section 4.1, after filtering, we run an OD strategy on a column subset, which contains only those columns that have selected this strategy. So if the same strategy is selected for different columns in #1 and #2, i.e. the strategy is run on different column subsets, the strategy output might be different.

In this experiment, using all the datasets for filtering leads to a decrease in effectiveness. But this does not prove that using the filtered dataset for filtering has a negative effect. In some cases, #1 and #2 can have similar results or #2 can be better than #1. This shows that filtering with filtered datasets is complicated.

4.4 Experiment 3: Different choices of the starting dataset

In this experiment, we compare the different choices of the starting dataset. Using only one dataset to filter all the other datasets is efficient, and we wonder if it is possible to achieve good effectiveness at the same time. We compare the F1 score on each dataset when using different datasets for filtering (as shown in Figure 4.7).

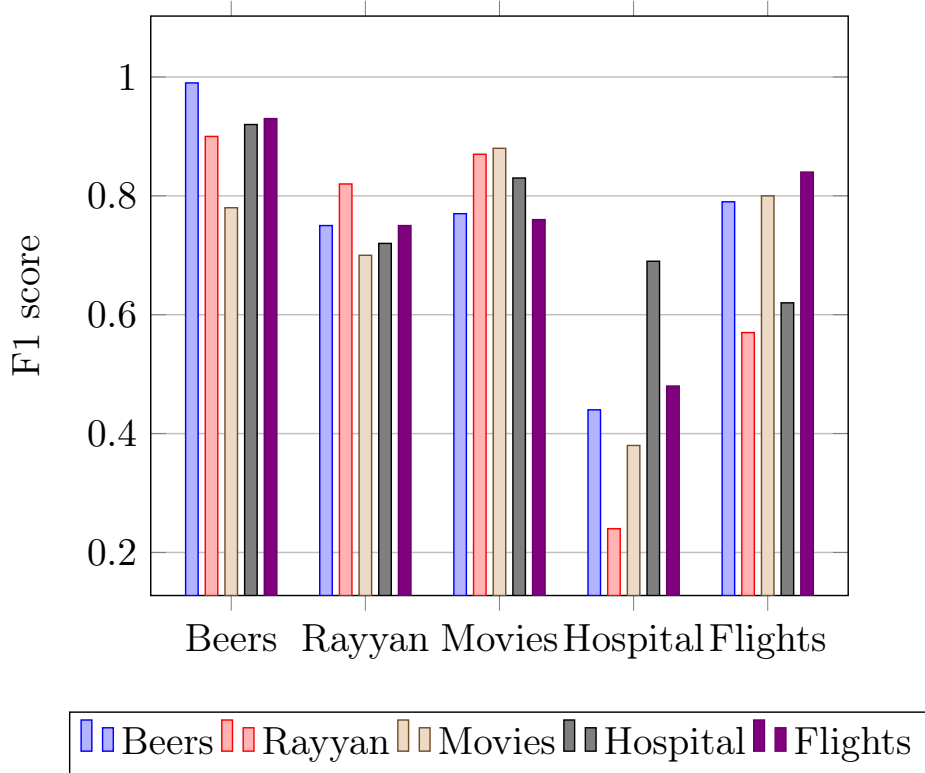


Figure 4.7: Comparison of different starting dataset.

The bars in a particular color show the results of using a particular dataset to filter. For example, from left to right, the blue bars show the F1 score of running normal detection on Beers, followed by the F1 score of Rayyan, Movies, Hospital, and Flights when we use Beers to filter them.

If we compare the bars in groups of 5, we can see how effective another dataset can filter this dataset. The highest bar is the baseline (when we run normal detection on this dataset). For example, considering the first 5 bars in different colors as Group Beers, using Movies to filter Beers has the worst effectiveness while using the other three datasets shows a similar result.

From this bar chart, we can conclude that none of the datasets is “perfect” and can filter all the rest of the datasets with good effectiveness. Because

4.5. EXPERIMENT 4: EVALUATION OF OUR SCORING METHOD 33

none of the datasets can filter Hospital effectively and Hospital does not work well on Flights.

Hospital has a rich variety of columns and many columns contain typos by random injection of the character “x” which can not be found in other datasets. So this could be the reason why no dataset can filter Hospital.

For Beers, Rayyan, and Movies, using different datasets for filtering leads to relatively similar results if compared with Flights. Flights is the dataset with the lowest column number (7) and four of them are actual/scheduled departure/arrival time. Such time columns with hidden FD are also special and can hardly find similar columns in other datasets.

If we can only pick one from these five datasets to be the starting dataset, Hospital might be the best choice. It can filter Beers, Rayyan, and Movies effectively, and the effectiveness on itself is guaranteed by normal detection. On Flights the decrease in F1 score is not “slight” but at least not as significant as using other datasets to filter Hospital. We assume that if we raise the number of starting datasets to two and run normal detection on both Hospital and Flights, we could achieve a good F1 score on each dataset.

4.5 Experiment 4: Evaluation of our scoring method

In this experiment, we evaluate our proposed scoring method in Section 3.3 on two groups of datasets.

4.5.1 Evaluation on Raha datasets

We evaluate our method first on the five Raha datasets that were introduced in Section 4.1.

Our scoring method requires us to first calculate a similarity score for each dataset. If the top candidates have a close similarity score, we apply a weight that is based on its tuple number to punish the larger datasets. We run normal detection on the dataset with the highest final score and the lowest similarity score and use both of them to filter the other datasets.

Table 4.9 shows the similarity score, weight, and final score of each dataset.

	Beers	Hospital	Rayyan	Movies_1	Flights
Similarity Score	191.88	293.03	203.62	285.30	129.85
Weight	0.83	0.93	0.93	0.48	0.83
Final Score	159.26	272.36	189.25	136.57	108.08

Table 4.9: Scoring Raha datasets.

In this group of datasets, Hospital achieves the highest similarity score

and Flights the lowest. Somehow we find the same datasets that we wanted to choose as mentioned in Experiment 3. We notice that Movies_1 also has a high similarity score, but from Experiment 1 we already know this dataset is large and inefficient. If we apply the weight to Hospital and Movies_1, Movies_1 has a much lower final score, so we stick to Hospital.

We use Hospital and Flights to filter the rest of the datasets. Table 4.10 and Table 4.11 show the effectiveness and efficiency of our detection plan compared with running error detection without filtering on all datasets.

	Hospital			Flights			Movies_1			Rayyan			Beers			Overall		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
No filtering	0.95	0.60	0.73	0.84	0.78	0.81	0.79	0.87	0.82	0.83	0.71	0.76	0.99	1.00	1.00	0.85	0.86	0.85
Our plan	0.94	0.58	0.71	0.83	0.82	0.82	0.78	0.86	0.81	0.73	0.75	0.74	0.95	0.95	0.94	0.82	0.85	0.84

Table 4.10: Effectiveness evaluation on Raha datasets.

	Hospital	Flights	Movies_1	Rayyan	Beers	Overall
No filtering	161	65	1053	57	148	1484
Our plan	156	60	711	44	101	1072
(+filtering runtime)	(+0)	(+0)	(+4)	(+2)	(+2)	

Improvement			32%	19%	30%	27%
-------------	--	--	-----	-----	-----	-----

Table 4.11: Runtime evaluation on Raha datasets.

With an improvement of 27% in efficiency, we achieve a similar effectiveness compared with the baseline, except for a slight decrease on Beers. We consider that our scoring method makes error detection more efficient while retaining its effectiveness.

4.5.2 Evaluation on Kaggle datasets

We also evaluate our method on another group of datasets containing 7 datasets from Kaggle^[2]. Errors are generated using Jenga^[3]. These 7 datasets are also from different domains (books, countries, covid, and movies).

We apply our method on this group and the score of each dataset is shown in Table 4.12.

	Disney	Imdb	Covid_1	Covid_2	Cou_1	Cou_2	Best_sellers
Similarity Score	139.64	278.36	168.28	219.98	297.26	421.91	162.50
Weight	0.83	0.70	0.89	0.86	0.93	0.95	0.84
Final Score	115.73	196.05	149.97	189.80	275.99	399.58	136.08

Table 4.12: Scoring Kaggle datasets.

Our detection plan is to run normal detection on Cou_2 (highest score) and Disney (lowest score) and use them to filter the rest of the datasets.

²<https://www.kaggle.com>

³<https://github.com/schelterlabs/jenga>

4.5. EXPERIMENT 4: EVALUATION OF OUR SCORING METHOD 35

Table 4.13 and Table 4.14 show the effectiveness and efficiency of our detection plan.

	Cou_2			Disney			Imdb			Best_sellers		
	P	R	F	P	R	F	P	R	F	P	R	F
No filtering	0.59	0.41	0.47	0.77	0.57	0.63	0.57	0.46	0.50	0.55	0.34	0.38
Our plan	0.55	0.52	0.53	0.77	0.55	0.63	0.85	0.43	0.55	0.80	0.23	0.34
	Covid_1			Covid_2			Cou_1			Overall		
	P	R	F	P	R	F	P	R	F	P	R	F
No filtering	0.89	0.82	0.85	0.79	0.78	0.77	0.56	0.45	0.48	0.62	0.52	0.56
Our plan	1.00	0.78	0.87	1.00	0.60	0.75	0.82	0.18	0.29	0.78	0.44	0.56

Table 4.13: Effectiveness evaluation on Kaggle datasets.

	Cou_2	Disney	Imdb	Best_sellers
No filtering	37	13	88	13
Our plan	35	11	24	4

Improvement			73%	69%
-------------	--	--	-----	-----

	Covid_1	Covid_2	Cou_1	Overall
No filtering	12	62	18	243
Our plan	3	18	4	99

Improvement	75%	71%	78%	59%
-------------	-----	-----	-----	-----

Table 4.14: Runtime evaluation on Kaggle datasets.

We ignore the strategy filtering runtime as it is $< 1s$ on each dataset. On this group of datasets, we improve the overall efficiency by 59%. While our plan achieves the same F1 score over all datasets compared with the baseline, our plan is better in terms of precision and worse in recall. We achieve a good F1 score on almost every dataset except Cou_1 because of the low recall. We have better precision on each dataset and on Covid_1 and Covid_2 we even achieve 100% precision.

We randomly select two datasets (Best_seller, Covid_1) to be the starting datasets as a “random” plan and compare our plan with it as shown in Table 4.15 and Table 4.16.

The random plan achieves good effectiveness on Cou_1 where our plan fails. But our plan works better on Imdb and Covid_2 and shows better effectiveness over all datasets. Our plan also shows better efficiency, especially on Imdb.

Our scoring method finds a plan which has better precision, worse recall, and better efficiency. It makes sense that strategy filtering can improve the precision of error detection because we are filtering out some irrelevant

	Cou_2			Disney			Imdb			Best_sellers		
	P	R	F	P	R	F	P	R	F	P	R	F
Our plan	0.55	0.52	0.53	0.77	0.55	0.63	0.85	0.43	0.55	0.80	0.23	0.34
Random	0.70	0.50	0.56	0.71	0.57	0.61	0.67	0.37	0.46	0.43	0.39	0.40
	Covid_1			Covid_2			Cou_1			Overall		
	P	R	F	P	R	F	P	R	F	P	R	F
Our plan	1.00	0.78	0.87	1.00	0.60	0.75	0.82	0.18	0.29	0.78	0.44	0.56
Random	0.90	0.82	0.85	0.88	0.60	0.70	0.59	0.42	0.46	0.63	0.48	0.54

Table 4.15: Effectiveness comparison with a random plan.

	Cou_2	Disney	Imdb	Best_sellers
Our plan	35	11	24	4
Random	15	7	40	14
	Covid_1	Covid_2	Cou_1	Overall
Our plan	3	18	4	99
Random	11	22	6	115

Table 4.16: Runtime comparison with a random plan.

strategies that can be “misleading”. Meanwhile, with fewer strategies, we may lose some important features that could help us find more errors.

Chapter 5

Conclusion and future work

To detect errors in a data lake using Raha more efficiently, we proposed a scoring method that identifies the most suitable starting datasets which will then be used to filter the rest of the datasets in the lake. We conducted tests on two groups of datasets and found that our method was able to achieve nearly identical levels of effectiveness as error detection without filtering, while also increasing efficiency by up to 59%.

This good result gives us some confidence in our method, but we must acknowledge that our method has not been thoroughly tested and we are uncertain about its effectiveness on other groups of datasets. We also have to admit that we are not “ordering” the datasets as we intended at the beginning. Instead, we only select the starting datasets and ignore the subsequent order for simplicity.

The unanswered questions in our thesis can be directions for future work:

1. Is there an optimal number of starting datasets?

Currently, we are utilizing two datasets to filter through our small data lake, which contains up to seven datasets. We are unsure whether increasing this number will improve the effectiveness.

Furthermore, if we have a larger data lake, should we try to use more datasets to filter the others? Or should we divide the datasets in the whole lake into several small groups?

2. How can we take advantage of the filtered datasets and find a subsequent order?

We believe the filtered datasets can also be useful as they can provide new columns and new strategies. But we figured out that in some cases, using filtered datasets can lead to a significant decrease in effectiveness because some helpful strategies are filtered out. However, by selecting top k strategies instead of using the gain function, we achieved better results.

This gain function is a threshold-free approach. We should stick to it if we can not identify the optimal strategy number on a column. But we may set a minimum strategy number. For example, if we believe each column needs at least 10 strategies, while the gain function selects only 5, then we force Raha to select 5 more top-scored strategies. This adjustment could prevent Raha from excessively filtering out strategies that might be helpful, which might also solve the problem of low recall using our method.

We select the starting dataset based on dataset similarities. Maybe we can identify a subsequent order using the same idea. For example, if we run normal detection on the highest-scored dataset and the lowest-scored dataset, the next dataset to filter might be the one that is most similar to the highest-scored dataset (or to both starting datasets).

In practice, it can be difficult to benefit from strategy filtering. Raha's filtering is based on column similarity and strategy evaluation on historical datasets. Without a clean version of a dataset, we can not generate the required strategy evaluation that Raha requires for filtering. We might prepare some cleaned datasets (with their dirty version) to detect errors in a new data lake with strategy filtering, but unless we manually clean the newly detected dataset in this lake, we can only rely on the "starting datasets".

Bibliography

- [1] Z. Abedjan, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, and M. Stonebraker. Dataxformer: A robust transformation discovery system. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1134–1145. IEEE, 2016.
- [2] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1247–1261, 2015.
- [3] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.
- [4] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 International Conference on Management of Data*, pages 829–846, 2019.
- [5] I. F. Ilyas, X. Chu, et al. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends® in Databases*, 5(4):281–393, 2015.
- [6] M. Mahdavi and Z. Abedjan. Baran: Effective error correction via a unified context representation and transfer learning. *Proceedings of the VLDB Endowment*, 13(12):1948–1961, 2020.
- [7] M. Mahdavi, Z. Abedjan, R. Castro Fernandez, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Raha: A configuration-free error detection system. In *Proceedings of the 2019 International Conference on Management of Data*, pages 865–882, 2019.
- [8] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment*, 12(12):1986–1989, 2019.
- [9] R. R. Panko. What we know about spreadsheet errors. *Journal of Organizational and End User Computing (JOEUC)*, 10(2):15–21, 1998.

- [10] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden. Outlier detection in heterogeneous datasets using automatic tuple expansion. 2016.
- [11] H. Schütze, C. D. Manning, and P. Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008.
- [12] P. Wang and Y. He. Uni-detect: A unified approach to automated error detection in tables. In *Proceedings of the 2019 International Conference on Management of Data*, pages 811–828, 2019.
- [13] M. Ziemann, Y. Eren, and A. El-Osta. Gene name errors are widespread in the scientific literature. *Genome biology*, 17(1):1–3, 2016.