# FAHES: A Robust Disguised Missing Values Detector

Abdulhakim A. Qahtan[*]
Qatar Computing Research Institute, HBKU
Doha, Qatar
aqahtan@hbku.edu.qa

Ahmed Elmagarmid
Qatar Computing Research Institute, HBKU
Doha, Qatar
aelmagarmid@hbku.edu.qa

Raul Castro Fernandez
MIT CSAIL
Cambridge, USA
raulcf@csail.mit.edu

Mourad Ouzzani
Qatar Computing Research Institute, HBKU
Doha, Qatar
mouzzani@hbku.edu.qa

Nan Tang
Qatar Computing Research Institute, HBKU
Doha, Qatar
ntang@hbku.edu.qa

## ABSTRACT

Missing values are common in real-world data and may seriously affect data analytics such as simple statistics and hypothesis testing. Generally speaking, there are two types of missing values: *explicitly missing values* (*i.e.,* NULL values), and *implicitly missing values* (*a.k.a.* disguised missing values (DMVs)) such as “11111111" for a phone number and “Some college" for education. While detecting explicitly missing values is trivial, detecting DMVs is not; the essential challenge is the lack of standardization about how DMVs are generated. In this paper, we present FAHES, a robust system for detecting DMVs from two angles: DMVs as detectable outliers and as detectable inliers. For DMVs as outliers, we propose a syntactic outlier detection module for categorical data, and a density-based outlier detection module for numerical values. For DMVs as inliers, we propose a method that detects DMVs which follow either missing-completely-at-random or missing-at-random models. The robustness of FAHES is achieved through an ensemble technique that is inspired by outlier ensembles. Our extensive experiments using real-world data sets show that FAHES delivers better results than existing solutions.

## CCS CONCEPTS

• **Information systems → Data cleaning**;

## KEYWORDS

Disguised Missing Value, Syntactic Outliers, Numerical Outliers, Syntactic Patterns

**ACM Reference Format:**
Abdulhakim A. Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. 2018. FAHES: A Robust Disguised Missing Values Detector. In *KDD ’18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3219819.3220109

| Source | Table | Column | DMVs |
|---|---|---|---|
| UCI ML | Diabetes | Blood Pressure | 0 |
| | adult | workclass | ? |
| | | education | Some College |
| U.S. FDA | Even Reports | EVENT_DT | 20010101, 20030101 |
| data.gov | Vendor Location | Ref_ID | -1 |
| data.gov | Graduation | Regents Num | s, - |
| data.gov.uk | Accidents 2015 | Junction Control | -1 |

**Table 1: Sample DMVs.**

## 1 INTRODUCTION

Real-world data is dirty and may misguide any data analytical task (*a.k.a.* garbage-in garbage-out), which may in turn lead to bad business decisions. Among the different types of errors, missing values constitute a challenging and well-recognized problem [23, 26], *e.g.,* for drug disease analysis [18]. Generally speaking, there are two types of missing values: explicit (*i.e.,* NULL values), and implicit (*a.k.a.* disguised missing values [26] (DMVs)), *e.g.,* “11111111” for a phone number. While explicit missing values are common in many data sets, DMVs are also widespread for various reasons, such as the user did not want to provide the correct information (*e.g.,* for survey forms), the correct value might not pass the system check constraints so a fake value is provided, the correct value is not available at the time of entry (*e.g.,* for creating a record in a hospital before receiving the insurance policy number), and so on.

Detecting explicit missing value is straightforward. Unfortunately, detecting DMVs is hard, since they can be either human (carefully) faked values, or system generated values. Obviously, there is no *global representation* for DMVs. Different persons and organizations will use different representations for DMVs. Consequently, it is impossible to define general rules for all data sets. DMVs in one table could represent legitimate values in other tables. In real applications, practitioners may have to write many customized rules to detect DMVs per table, which is a daunting task.

Before we present our solutions, let us show the prevalence of DMVs in real-world data sets. We have manually checked 100

[*] Corresponding author

*KDD ’18, August 19–23, 2018, London, United Kingdom*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220109

random tables from different repositories such as data.gov, data.gov.uk, FDA[1] and UCI ML[2], and we have found that more than 50% of these tables have DMVs. Some sample DMVs are given in Table 1. We make a couple of observations from this table. There are different DMVs for different numerical columns (*e.g.,* 0 for Blood Pressure and -1 for Ref_ID) – hard-coded rules must be data specific. Moreover, some DMVs are legitimate values (*e.g.,* "Some College" for education, and 20010101 for EVENT_ID) – they may pass all integrity constraint check.

In order to devise effective solutions, we categorize detectable DMVs into the following cases:

(1) Out of range data values, *e.g.,* disguise the missing values in an attribute that takes only positive values with a negative value.
(2) Outliers, *e.g.,* disguise the missing values with a very large value or a very small value such as replacing the missing age values with the value 1000.
(3) Strings with repeated characters or characters that are next to each other on the used keyboard, *e.g.,* replacing a phone number with 5555555555.
(4) Values with non-conforming data types, *e.g.,* disguising the missing strings with numerical values and vice versa.
(5) Valid values that are randomly distributed within the range of the data and used frequently in the data set.

In each of the above cases, DMVs can be treated as either outliers (cases 1-4) or inliers (case 5). For the case of sufficiently well disguised values which are inliers, *e.g.,* as in cases of a well thought fraud, detecting them is out of the scope of this work.

**Contributions.** In this paper, we present FAHES[3], an end-to-end system to deal with the above cases. We summarize our contributions below.

(i) We formally define the problem of DMV detection, introduce the architecture of FAHES, and present our DMVs ensembles method (Section 2).
(ii) For detecting DMVs as outliers:
  - We propose a syntactic outlier detection module (cases 1, 3 and 4) to capture those DMVs that are syntactic outliers or contain special patterns such as strings with repeated substrings (*e.g.,* abcabcabc) and numbers with incrementally increasing/decreasing digits (*e.g.,* 1234567) (Section 3).
  - We devise a numerical outlier detection module (cases 1 and 2) that best suits FAHES in terms of detection effectiveness (represented by precision and recall), time efficiency and minimal parameter setting. This module will detect DMVs that are far from the rest of the values in the Euclidean space (Section 4).
(iii) For detecting DMVs as inliers, we devise an algorithm for detecting DMVs that follow missing-completely-at-random (MCAR) or missing-at-random (MAR) models (case 5). The algorithm leverages some the basic idea of DiMaC [15] but significantly outperforms it for both in terms of efficiency and effectiveness (Section 5).

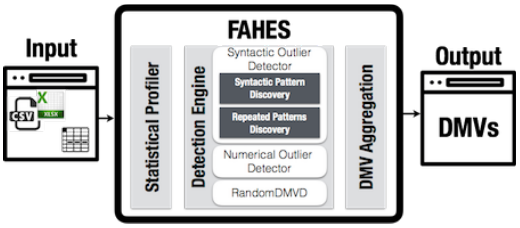

Input
CSV
XLSX
FAHES
Statistical Profiler
Detection Engine
Syntactic Outlier Detector
Syntactic Pattern Discovery
Repeated Patterns Discovery
Numerical Outlier Detector
RandomDMVD
DMV Aggregation
Output
DMVs

**Figure 1: FAHES Architecture.**

(iv) We have conducted extensive experiment using real-world data sets from diverse sources to show the wide applicability of FAHES. FAHES can detect DMVs with good precision and recall that cannot be achieved by other methods (Section 6).

We discuss related work in Section 7 and conclude in Section 8. Furthermore, a demo showing how FAHES works was presented in ICDE 2018 [28].

## 2 OVERVIEW

In this section, we first define disguised missing values (DMVs) (Section 2.1). We then introduce the architecture of FAHES (Section 2.2). We close the section by introducing an ensemble method which will be used by each component of FAHES (Section 2.3).

### 2.1 Disguised Missing Values

Generally speaking, DMVs are values that are entered in a given table to replace missing values. The DMVs detection problem can be formulated as follows [26]: let $T = [t_{ij}]$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ be the true data table that is unknown, where $m$ and $n$ represent the number of tuples and attributes in $T$, respectively. Let $\tilde{T} = [\tilde{t}_{ij}]$ be the recorded (input) dirty table. We would like to construct a matrix $T'$ with Boolean entries that represent the cells with DMVs. That is, $T' = [t'_{ij}]$ where:

$$t'_{ij} = \begin{cases} 1 & \text{if } (t_{ij} = \phi \wedge \tilde{t}_{ij} \neq \phi) \\ 0 & \qquad\qquad otherwise \end{cases}$$

The symbol $\phi$ means that the cell is empty or 'null'.

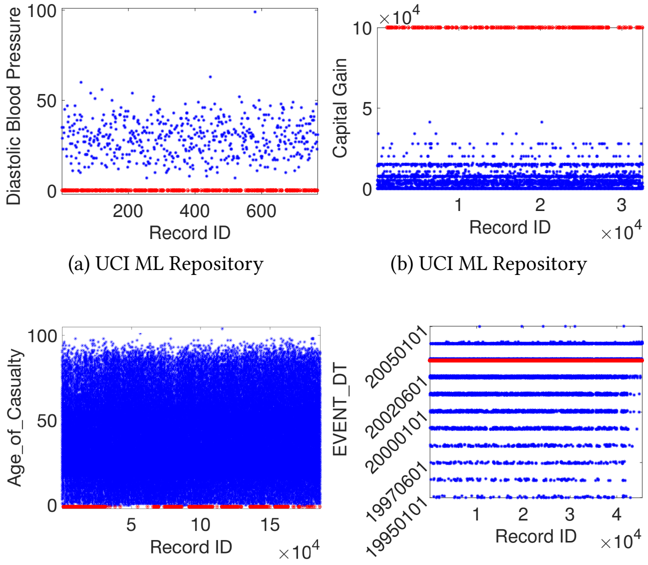
Note that, in practice, the number of DMVs for each attribute of one data set is small [14]. Also, we focus on detecting frequent DMVs since infrequent DMVs typically have little effect on analytics such as simple statistics, hypothesis testing, and regression models.

### 2.2 Architecture

Intuitively, it is unlikely to come up with a single method that could detect all different types of DMVs. Hence, we advocate a multi-pronged approach, where different methods are designed to detect DMVs based on their characteristics. Note that, this does not exclude the case that one DMV can be detected by different methods. Figure 1 shows the architecture of FAHES. The system contains three main components.

**Statistical Profiler.** It collects two types of statistics, which will be used by the detection engine: (1) *per table* statistics such as #-records and #-attributes, and (2) *per column* statistics such as #-empty (null) cells, #-numerical entries, #-strings and #-distinct values. For numerical values, we also count #-positive/negative entries. The profiler also stores the distinct values and their frequencies.

[1] https://open.fda.gov/downloads/
[2] https://archive.ics.uci.edu/ml/index.php
[3] From the Arabic word that means inspector.



(a) UCI ML Repository

(b) UCI ML Repository

(c) Road Safety (data.gov.uk)

(d) FDA

**Figure 2: DMVs can be outliers as in (a) and (b) or non-isolated values as in (c) and (d).**

**Detection Engine.** It contains the different modules for the different types of DMVs. For DMVs as outliers, we have a *syntactic outlier detector* which again contains a *syntactic pattern discovery* and a *repeated patterns discovery*, and a *numerical outlier detector*. For DMVs as inliers, RandomDMVD module detects DMVs that follow missing-completely-at-random (MCAR) or missing-at-random (MAR) model.

Figure 2 shows examples for different types of DMVs, using different data sets. The red dots indicate the repeated single DMV in different tuples. The DMVs in Figures 2(a,b) can be detected by the numerical outlier detection; the DMVs in Figure 2(c) can be detected using the syntactic detection module; and the DMVs in Figure 2(d) are only detectable using RandomDMVD.

**Aggregation.** As mentioned earlier, the same DMV may be detected by more than one module. The DMVs aggregator simply returns the union of all detected DMVs along with the module that detected them and the score they were assigned by that module.

**Setting Thresholds.** A general observation is that FAHES consists of multiple detection modules with threshold values required by each module. The score values generated by our modules are normalized to the interval $[0, 1]$ where values that are close to 1 give a strong signal about the value under test. A dynamic technique to set these thresholds is used by sorting the score values starting with the highest value and setting the threshold to the first large difference between two consecutive score values. Threshold values can also be adjusted based on the number of DMVs that could be reported. In all of the data sets that we checked manually, no attribute had more than 5 DMVs. Hence, if the used threshold reports many DMVs, a more strict threshold should be used. In all cases, strict thresholds that are $> 0.99$ work well for DMV detection.

## 2.3 DMV Ensembles

In order to improve the detection effectiveness of each detection module, we use a DMV ensemble approach that is inspired by outlier ensembles [2, 3] and methods to derive a strong classifier from many weak classifiers [30]. Both DMV detection and outlier detection are unsupervised problems since the data labels are not available. The theoretical justification for our approach can be derived in terms of the bias-variance tradeoff used in [3]. The difference is that our system for detecting DMVs is composed of three different modules where combining the score values of these modules is not feasible. We instead use DMV ensembles to improve the detection effectiveness of each individual component which will increase the overall detection effectiveness.

Each detection component generates a score value in the interval $[0, 1]$, where normal values take score values close to zero and DMVs have score values close to one. For a given value $v_i$, assume that the score $s'_i$ obtained for $v_i$ using the scoring function $f'$ has a corresponding optimal DMV score $s_i$ that can be obtained using an unknown function $f$. Based on the bias-variance analysis in [3], the mean integrated square that is used to quantify the accuracy of the scoring function $f'$ compared to the optimal unknown scoring function $f$ is defined as $MSE = \frac{1}{n}\sum_{i=1}^{n}\{s_i - s'_i\}^2$. Many techniques can be used to reduce the MSE of the computed score values such as bagging, bragging, wagging and subagging [7]. In [2], it was suggested to use subspace exploration for the methods that work on multidimensional data where the scoring function works on multiple subspaces to generate multiple sets of score values. For methods that require parameter setting, it is suggested to use multiple values for the parameter. In both cases, it was recommended to use the maximum score as a combination function to avoid the dilution from irrelevant subspaces or poor parameter values that would affect the averaging and the aggregation of the score values. We adopt the latter for scoring candidate DMVs.

# 3 DMVS AS SYNTACTIC OUTLIERS

As discussed earlier, disguising the missing values could be done using out of range values that do not conform syntactically with the regular values in a given attribute. Thus, learning the data patterns of each attribute could help in detecting such DMVs. In this section, we show how we discover syntactic patterns in a given attribute and then use them to detect non-conforming values. We also show how to tackle the special case of repeated patterns.

## 3.1 Syntactic Pattern Discovery

The syntactic pattern discovery module, SynPat for short, learns for each attribute a set of patterns that represent the values within that attribute. Discovering the syntactic structure for a set of values in an expressive and compact way is an NP-complete problem [12]. Discovering the optimal set of patterns is out of the scope of this paper. Instead, we focus on generating syntactic patterns that can be used to describe most of the values in a column and hence allows us to easily detect non-conforming values. Intuitively, the values that have non-conforming syntactic structures (syntactic outliers) compared to the structure of the majority of the values and appear frequently are likely to be DMVs.

<table>
<tr><th>Class</th><th>Representative</th><th>Equivalent Regular Expression</th></tr>
<tr><td>Uppercase letters</td><td>$u$</td><td>A-Z</td></tr>
<tr><td>Lowercase letters</td><td>$l$</td><td>a-z</td></tr>
<tr><td>Digits</td><td>$d$</td><td>0-9</td></tr>
<tr><td>Space</td><td>$s$</td><td>space|tab|newline|carriage-return</td></tr>
<tr><td>Dot</td><td>$t$</td><td>.</td></tr>
<tr><td>Hash</td><td>$h$</td><td>#</td></tr>
<tr><td>Punctuation</td><td>$p$</td><td>: |; |?</td></tr>
<tr><td>Enclosures</td><td>$e$</td><td>[|(|{|]|)|}</td></tr>
<tr><td>Special Symbols</td><td>@|&|′|”|_| − |,</td><td>@|&|′|”|_| − |,</td></tr>
<tr><td>Other Symbols</td><td>$y$</td><td>All other symbols</td></tr>
<tr><td>Alphabet</td><td>$a$</td><td>$u|l$</td></tr>
<tr><td>Word</td><td>$w$</td><td>$a| - |\_|@|\&|, |.|'$</td></tr>
<tr><td>Word+</td><td>$v$</td><td>$w|d| - |h|t$</td></tr>
</table>

**Table 2: The set of atomic and compound classes used to build the syntactic pattern discovery.**

**Overview.** Discovering the syntactic structure of a given attribute requires developing a suitable syntactic representation of its values. In structured data sets, many attributes, such as ZIP codes and phone numbers, would have a dominant syntactic structure with a few (maybe none) values with syntactically different patterns. *A syntactic pattern* is a sequence of symbols (from a specific alphabet) that represents the characters in a given value. For example, a phone number with ten digits is represented by $d^{10}$ while a ZIP Code value could be represented by $d^5 - d^4$. Our proposed approach is to first construct an initial set of syntactic patterns and then aggregate them to have a small set of patterns. Afterwards, we consider the patterns that represents less than 1% of the distinct value in the attribute as non-conforming patterns.

**Generating initial syntactic patterns:** To construct the initial set of syntactic patterns, we define a set of atomic classes (alphabet) for the characters in the attribute values, which are presented in Table 2[4]. The special symbol class has no representative as each symbol represents itself in the resulting pattern. The atomic classes in Table 2 represent a partitioning of the text such that each data entry in a given table has a unique representation.

For each attribute, SynPat accepts as input the set of distinct values $\mathcal{D}$ encoded as strings and returns a set of patterns. The idea is to find a set of dominating patterns that represent the values in the given attribute. Any values that cannot be generated by one of the dominating patterns is considered a candidate DMV. Finding the set of dominating patterns involves discovering [24] (1) a set of syntactic patterns $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ and (2) a partitioning $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m\}$ of the set $\mathcal{D}$ such that partition $\mathcal{D}_i$ is generated by pattern $P_i$. Initially, SynPat tests the characters in the strings and replaces each character by its corresponding class representative from Table 2. The consecutive occurrences of the same class representative in each pattern are merged and the number of consecutive occurrences is stored. For example, the pattern for the value “Male” is “$ul^3$” which states that we have a single uppercase letter followed by three lowercase letters. We should note that we do not count the number of spaces, we use $s$ instead of $s^k$.

<table>
<tr><th>ID</th><th>Rule</th><th>Description</th></tr>
<tr><td>$r_1$</td><td>$class^k \rightarrow class^+$</td><td>Ignore the number of occurrences</td></tr>
<tr><td>$r_2$</td><td>$u|l \rightarrow a$</td><td>Replace the occurrence of $'u'$ or $'l'$ by $'a'$ = Alphabet</td></tr>
<tr><td>$r_3$</td><td>$d^k t d^m \rightarrow d^{k+m+1}$</td><td>Floating point numbers are considered digits</td></tr>
<tr><td>$r_4$</td><td>$u^k s \rightarrow u^{k+1}$<br>$s u^k \rightarrow u^{k+1}$</td><td>Merge classes $'u'$ and $'s'$</td></tr>
<tr><td>$r_5$</td><td>$l^k s \rightarrow l^{k+1}$<br>$s l^k \rightarrow l^{k+1}$</td><td>Merge classes $'l'$ and $'s'$</td></tr>
<tr><td>$r_6$</td><td>$a^k s \rightarrow a^{k+1}$<br>$s a^k \rightarrow a^{k+1}$</td><td>Merge classes $'a'$ and $'s'$</td></tr>
<tr><td>$r_7$</td><td>$a^k(-|\_|@|\&|, |.|') \rightarrow w^{k+1}$<br>$(-|\_|@|\&|, |.|')a^k \rightarrow w^{k+1}$</td><td>Alphabet characters that are separated by special symbols are replaced by $'w'$</td></tr>
<tr><td>$r_8$</td><td>$(d| - |h)d^m w^k \rightarrow v^{k+m+1}$<br>$w^k d^m (d| - |h) \rightarrow v^{k+m+1}$</td><td>Merge more classes that are likely to occur together</td></tr>
</table>

**Table 3: The set of rules that are used to reduce the number of discovered patterns. There are more rules in $r_3 - r_8$ where the constants are replaced by ‘+’. We omit them from the table due to space limitations.**

**Aggregating syntactic patterns:** For attributes with values that follow a well defined syntax, such as phone numbers and ZIP codes, the initial patterns could be enough to discover the main patterns and the non-conforming patterns. However, in other cases, such as for the attributes “employee name” and “product code”, the number of discovered patterns is large due to the differences in the entries length and the use of many characters that belong to different classes within the same value. In such cases, we aggregate *similar* patterns in order to reduce the number of discovered patterns and discover the set of dominating ones.

We use the rules in Table 3 to aggregate similar patterns for reducing the number of discovered patterns. This is an extended list of the cluster class hierarchy presented in [20]. We use the superscript sign "+" to represent any number of consecutive occurrences for the same class symbol. Selecting the aggregation rule that should be applied next is performed in a way that the number of remaining patterns is minimized. Please note that the pattern "$Any^+$" that could represent any value is not defined in our syntactic profiler. In this sense, it is guaranteed that our technique would converge to a local minimal number of patterns. In addition, some of the rules can only be applied if others have been already applied. The dependency graph shown in Figure 3 shows the different dependencies between the rules. Any rule that is not in the graph can be applied at any time since it does not have any dependencies. $r_6$ can only be applied after $r_2$, $r_7$ can only be applied after $r_2$ or $r_6$, and $r_8$ can only be applied after rule $r_7$.

The pattern aggregation process will continue until the number of discovered patterns becomes less than a given threshold $\gamma$, which is given a default value of 5, or the set of rules has been exhausted.

[4] This is a modified list from the syntax of the POSIX Java regex classes.

**Algorithm 1:** SyntacticPatternDiscovery

```
Input: $\mathcal{D}$: distinct values in attribute $A$,
       $\mathcal{R}$: set of aggregation rules,
       $D_g$: rules dependency graph, $\gamma = 5$
Output: $\mathcal{P} = \{\langle R_g, q\rangle\}$: minimal set of patterns that represent $\mathcal{D}$ and
        the number of values represented by each pattern
        $\mathcal{R}_s$: the sequence of rules used to get $\mathcal{P}$
1  begin
2      $\mathcal{P} = \phi$, $\mathcal{R}_s = \phi$
3      for $str \in \mathcal{D}$ do
4          $R_g = RegEx(str)$
5          if $\langle R_g, q\rangle \in \mathcal{P}$ then
6              $\mathcal{P}(\langle R_g, q\rangle) = \langle R_g, q + 1\rangle$
7          else
8              $\mathcal{P}.append(\langle Rg, 1\rangle)$
9      $i = 1$
10     while $(not(\mathcal{R}.empty()))$ do
11         $\mathcal{P}_i = \mathcal{P}$, $\mathcal{R}_{s_i} = \phi$
12         while $(|\mathcal{P}_i| > \gamma)$ and $(not(\mathcal{R}.empty()))$ do
13             $\mathcal{P}_{temp} = \phi$
14             $r_m = \arg\min_{r\in\mathcal{R}}(|\mathcal{P}_i|)$
15             if $not(predecessors(r_m).executed())$ then
16                 execute($predecessors(r_m)$)
17             for $\langle R_g, q_j\rangle \in \mathcal{P}_i$ do
18                 $R'_g \leftarrow$ apply rule $r_m$ on $R_g$
19                 if $\langle R'_g, q_k\rangle \in \mathcal{P}_{temp}$ then
20                     $\mathcal{P}_{temp}(\langle R'_g, q_k\rangle) = \langle R'_g, q_k + q_j\rangle$
21                 else
22                     $\mathcal{P}_{temp}.append(\langle R'_g, q_j\rangle)$
23             $\mathcal{R}.remove(r_m)$
24             $\mathcal{R}_{s_i}.append(r_m)$
25             $\mathcal{P}_i = \mathcal{P}_{temp}$
26         $increment(i)$
27     $[\mathcal{P}, k] = get\_best\_ptrns\_list\{\mathcal{P}_j,\ \ j = 1, \ldots, i\}$
28     return $[\mathcal{P}, \mathcal{R}_{s_k}]$
```

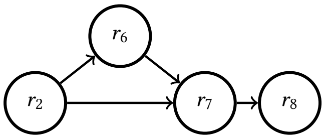


**Figure 3: The dependency between the rules in Table 3.**

The parameter $\gamma$ controls the expressiveness/compactness of the discovered patterns. Small value for $\gamma$ will force more aggregation rules to be applied which will merge more patterns in most of the cases and vice versa. There are other cases such as the "description" or "URL" attributes in which the values have many different patterns that are hard to aggregate.

Given a pattern $P_i$, we assign it two values, a contribution defined as $C_i = \frac{q_i}{|\mathcal{D}|}$, where $q_i$ is the number of values in $\mathcal{D}$ that are represented by the pattern $P_i$, and a DMV score defined as $SynSc(P_i) = 1 - C_i$. The process of discovering syntactic patterns is presented in Algorithm 1. For each attribute, the algorithm starts by constructing the initial set of patterns by replacing each character in the attribute values by its corresponding atomic classes from Table 2 (lines 2-8). The algorithm then performs pattern aggregation (lines 10-27) by applying the rules in Table 3 in a way that minimizes the set of discovered patterns. After generating the minimal set of patterns $\mathcal{P}_i$ (*i.e.*, $|\mathcal{P}_i| < \gamma$) using the sequence of rules $\mathcal{R}_{s_i}$, we iterate over the other rules to generate another minimal set of patterns. Each time, we iterate over the set of rules that have not been applied yet. If the rule that will be applied in the current iteration needs other rules to be executed first (based on the dependency graph in Figure 3), we apply these rules (even if they have been removed from $\mathcal{R}$ in a previous outer iteration) before applying the current selected rule. Upon termination, the algorithm checks and returns the set of patterns that contains the smallest number of patterns with a score value close to 1.

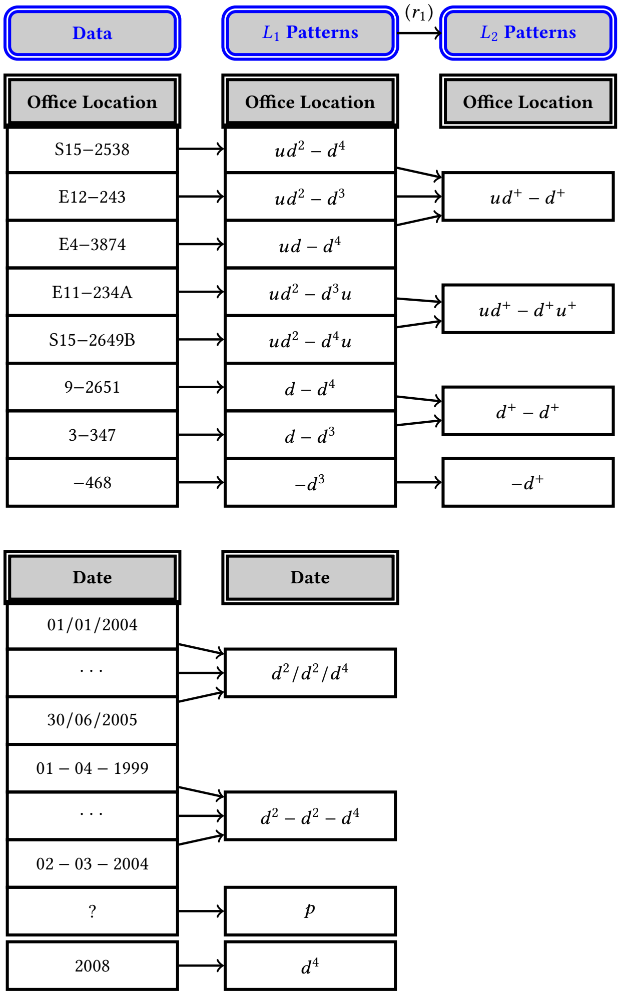


**Figure 4: Example of syntactic pattern discovery.**

Figure 4 represents a simple example of applying the syntactic patterns discovery output to find non-conforming data patterns. After constructing multiple sets of patterns and selecting the set of patterns that best describes the values in the attribute, we need to generate the pattern that corresponds to each value in $D$ by applying the same set of rules that are used to construct the set of patterns. The values that have patterns with a *SynSc* value greater than a given threshold that is close to 1 are reported as DMV candidates.

The submodule *get_best_ptrns_list* iterates over the discovered lists of patterns. For each list, it sorts the patterns in descending order based on the number of distinct values that can be represented by that pattern. The patterns that represent less than 1% of the distinct values of the attribute are considered syntactic outlier patterns. The list with the minimum number of syntactic outlier patterns is considered as the best pattern list for discovering the DMVs.

### 3.2 Repeated Pattern Identification

A simple way to disguise missing values is to use strings that contain repeated patterns. For example, by pushing the same key on a keyboard multiple times or entering the same sequence of characters multiple times. The repeated pattern identification module defines a similarity function between the consecutive characters within a given string. For each attribute, the set of distinct values $\mathcal{D}$ are processed by this module which would return a list $\mathcal{L}$ of pairs $< value, score >$. The score takes values in the interval $[0, 1]$. A small score value means that the string does not have repeated patterns, whereas a large value means that the string contains many repeated patterns.

Our target is not just to find the repeated substrings but to define a scoring function that generates higher scores for longer values with subtrings that are repeated more frequently. For that reason, we define the scoring function $rep(S)$ in terms of the number of occurrences of the repeated substring in $S$, the length of the substring, the length of $S$, and the length of the longest string in the attribute that $S$ belongs to. Let $M$ be the length of the longest string in attribute $A$. Given an attribute $A$, we define $rep(S)$ for $S \in A$ as

$$rep(S) = \frac{|s| * t}{|S|} * \frac{|s| * t}{|M|} \tag{1}$$

where $M$ is the longest value in $A$, $s$ the substring that appears repeatedly in $S$, and $t$ is the number of times that $s$ was repeated. The first term in Equation (1) determines the ratio of the occurrences of the repeated substring to the length of the string, whereas the second term gives higher score values for long strings that contain repeated substrings. Short strings that include repeated substrings are likely to exist in real data unlike long strings. Thus, we need to consider the minimum length of the string $S$ that should be tested to reduce the number of false positives. The process of detecting DMVs, based on repeated pattern, is then to take the values that have a score close to 1.

## 4 DMVS AS NUMERICAL OUTLIERS

As we mentioned earlier, there are cases where the missing values are replaced by out of range values, which can be seen as outliers. In the Pima Indians Diabetes data set [21], many values in the attribute diastolic blood pressure were replaced by 0. In the adult data set [21], many of the missing values in the "capital-gain" attribute are replaced by 99999. Both of these can be seen as outliers (see Figures 2(a, b)). Outlier detection is a well studied problem and many methods have been developed. However, selecting a robust outlier detection method that requires minimal user interaction is still a challenging problem. Moreover, existing outlier detection techniques detect outliers that are not necessarily DMVs.

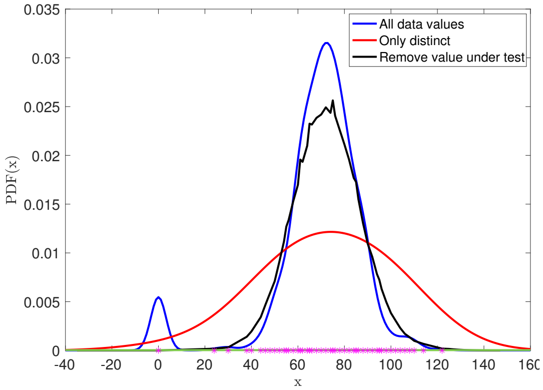


**Figure 5: The effect of the included data in the PDF estimation on correctly detecting the outliers.**

**Algorithm 2:** DENSITY-BASEDOUTLIERDETECTION

```
Input: $\mathcal{D}$: set of distinct values in attribute $A$,
  $D_f$: distinct frequent values in $\mathcal{A}$
Output: $O_f$: set of values that represent outliers
begin
    $O_f = \phi$
    $h = compute\_bandwidth(\mathcal{D}, K)$
    $\tau, f_{max} = compute\_threshold(h, \mathcal{D}, K)$
    for $x$ in $\mathcal{D}_f$ do
        $X = \mathcal{D}/\{x\}$
        $h_0 = compute\_bandwidth(X, K)$
        for $i \in \{0, \ldots, 4\}$ do
            $h = h_0 - (i * 0.2 * h_0)$
            $f_i(x) = \frac{1}{|X|h} \sum_{x_i \in X} n_{x_i} K\left(\frac{x - x_i}{h}\right)$
            $g_i(x) = \frac{f_{max} - f_i(x)}{f_{max}}$
        $g_m(x) = \max\{g_i(x), \quad i = 1, \ldots, 4\}$
        if $g_m(x) > \tau$ then
            $O_f$.append($x$, $g_m(x)$)
    return $O_f$
```

Detecting DMVs that appear as outliers raises the following challenges: (1) the values that are used to replace the missing values are used frequently, which could mislead many outlier detection tools; (2) the set of DMVs does not equal the set of outliers, *i.e.,* there are many outliers that are not DMVs and vice versa; and (3) different outlier detection methods detect different sets of outliers [13].

**Outlier Detection for DMVs.** We implemented a modified version of the outlier detector proposed in [29]. Detecting outliers using probability density functions has been shown to outperform other popular outlier detection methods in terms of time efficiency and detection effectiveness [19, 29]. The simplicity of parameter setting for this method is another advantage. According to [29], outliers fall in areas with low density. A good probability density function (PDF) estimator could reveal very useful information about the data distribution of a given attribute which could then be utilized to detect outliers that represent DMVs. Values with small PDF values are candidates outliers. However, the DMVs are usually used frequently, which increase their PDF values as shown in Figure 5 (the blue curve). Thus, conventional outlier detectors will be unable to report such values as outliers.

A possible solution for the above problem is to ignore the repetition of the values within the data set and estimate the PDF using the distinct values only (red curve in Figure 5). However, this solution is impractical since it hides important information about the data distribution. For this reason, we propose to ignore the duplicates from the data set before applying the outlierness test only for the value under test but not for the other values. The estimated PDF curve is not smooth as shown by the black curve in Figure 5. However, the estimated density shows clearly that the value 0 is an outlier which was not detectable when estimating the density function using the full data set or the set of unique values.

Algorithm 2 presents our method for detecting the outliers that represent DMVs. For each numerical attribute, we remove the value that we estimate the PDF for, and extract the set $X$ of values that are different from that value (line 6). The bandwidth value $h$ (also called the smoothing parameter of the PDF) is computed based on the data set $X$ and the kernel function $K$ (line 7). This value controls the smoothness of the PDF curve. Large $h$ values over-smooth the PDF curve and hide a lot of useful information whereas small $h$ values increase the fluctuation of the PDF curve and show misleading behavior of the density function in many cases. In this paper, we use the popular Gaussian kernel, which is defined as follows $K(x) = \frac{1}{2\pi} e^{-\left(\frac{x-\hat{\mu}}{2\hat{\sigma}^2}\right)}$, where $\hat{\mu}, \hat{\sigma}$ are the sample mean and the sample standard deviation, respectively. For setting the bandwidth $h$, we use the normal rule $h = 1.06\hat{\sigma}n^{-\frac{1}{5}}$, where $\hat{\sigma}$ is the sample mean and $n$ is the number of samples. This bandwidth value works very well when the data follows a normal distribution. However, it over-smooths the PDF curve in the case of multimodal density functions, which will make many outliers undetectable. Let $h_0$ be the bandwidth value computed using the normal rule, we use multiple $h$ values ($h \in \{h_0 - (i * 0.2 * h_0),\ \ i = 0, \ldots, 4\}$) to estimate the PDF at any value of the frequent values in each attribute. It has been shown in [10] that using $h < 0.2 * h_0$ undersmoothes the density function curve significantly and gives incorrect estimation of the PDF. The PDF at the value $x$ is then computed in line 10 where $n_{x_i}$ represents the frequency of the value $x_i$ in the attribute which is stored by the statistical profiler.

Let $f_i(x)$ be the PDF of the value $x$ estimated using the bandwidth $h_i = h_0 - (i * 0.2 * h_0),\ \ i = 0, \ldots, 4$, $g_i(x) = \frac{f_{max} - f_i(x)}{f_{max}}$ is the score value based on the PDF and $g_m = \max\{g_i,\ \ i = 0, \ldots, 4\}$. The value $x$ is reported as a candidate DMV if its score $g_m$ is greater than a given threshold $\tau$. Qahtan et. al. in [29] proposed to estimate the PDF at a set $M = \{m_0, m_1, \ldots, m_{q-1}\}$ of points that are uniformly distributed within the range of the data to be used as an approximation of the PDF curve. We compute $f_{max} = \max_{m \in M} f(m)$. The threshold is set automatically using a technique similar to the one in [29] by computing the average PDF values $\bar{f} = \frac{\sum_{i=0}^{q-1} f(m_i)}{q}$. The threshold $\tau$ is chosen to be $\tau = 0.99\bar{f}$, which is more restrictive than the one used in [29]. This setting works well in our problem as it reduces the number of false positives.

# 5 DMVS AS INLIERS

To better understand how to detect DMVs that are not outliers (syntactic or numeric), we first discuss how to statistically model missing data. This would then help us build a detector, RandomD-MVD, for DMVs that take valid values.

## 5.1 Statistical Modeling of Missing Data

In general, missing data follows one of three models [4, 22]: missing-completely-at-random (MCAR,) missing-at-random (MAR), or not-missing-at-random (NMAR). In the MCAR model, the missing values are randomly distributed across all records. They do not depend on the recorded values or on the missing values. An example is shown in Table 4 (b). In the table, the probability that the salary value is missing does not depend on the recorded or on the missing values [22]; we can see that: $P(Salary = null \mid position = Manager) = P(Salary = null \mid position = Secretary) = P(Salary = null \mid position = Technician) = P(Salary = null) = 0.5$. In the MAR model, the missing values are randomly distributed within one or more sub-samples of the records. The fact that the data values are missing depend only on the recorded values but not on the missing values. In other words, the probability that a given attribute contains MAR values could depend on any of the observed values [4, 22]. An example of MAR values is shown in Table 4 (c); the salary values are missing when the position takes the value "Technician" is missing, *i.e.*, $P(Salary = null \mid position = Technician) = 1$ whereas $P(Salary = null \mid position = Manager) = 0$ and $P(Salary = null \mid position = Secretary) = 0$. In the NMAR model, the missing data depends on the missing values themselves. An example is shown in Table 4 (d); $P(Salary = null \mid Salary < 2500) = 1$ whereas $P(Salary = null \mid Salary \geq 2500) = 0$. Detecting DMVs that follow the NMAR model is hard to impossible.

## 5.2 RandomDMVD

Detecting missing values that are disguised using frequent valid values is clearly hard. Under the MCAR or MAR models, detecting the values that replace the missing values could be achieved by removing one of the frequent values in a given attribute and testing if the resulting missing cells follow either models. If the resulting empty cells follow either models then that value is likely to be a DMV. This process could be repeated over the most frequent values in each attribute and the values that satisfy the MCAR/MAR assumption are reported as DMV candidates. This technique has been studied in [14, 15]. In the following, we describe a new approach for detecting such DMVs that leverages some of the concepts from [14].

For detecting the DMVs that follow MCAR/MAR models, we assume that a value $v$ in attribute $A_i$ is a DMV if $\tilde{T}_{A_i=v}$ contains a subset that represents a good sampling of the original $\tilde{T}$, where $\tilde{T}_{A_i=v} = \sigma_{T_{A_i=v}}$. We compare the data distribution of $\tilde{T}_{A_i=v}$ and $\tilde{T}$ to generate a score for each frequent value and sort the values based on their scores. Values that have scores close to 1 are considered DMVs. We use the mutual information between the distribution in $\tilde{T}_{A_i=v}$ and $\tilde{T}$ as a metric for comparing the data distribution. The mutual information is shown to be a good metric to discover if

| ID | Position | Salary $ |
|---|---|---|
| E1 | Manager | 3500 |
| E2 | Secretary | 2200 |
| E3 | Manager | 3600 |
| E4 | Technician | 2400 |
| E5 | Technician | 2500 |
| E6 | Secretary | 2000 |

(a)

| ID | Position | Salary $ |
|---|---|---|
| E1 | Manager | null |
| E2 | Secretary | 2200 |
| E3 | Manager | 3600 |
| E4 | Technician | null |
| E5 | Technician | 2500 |
| E6 | Secretary | null |

(b)

| ID | Position | Salary $ |
|---|---|---|
| E1 | Manager | 3500 |
| E2 | Secretary | 2200 |
| E3 | Manager | 3600 |
| E4 | Technician | null |
| E5 | Technician | null |
| E6 | Secretary | 2000 |

(c)

| ID | Position | Salary $ |
|---|---|---|
| E1 | Manager | 3500 |
| E2 | Secretary | null |
| E3 | Manager | 3600 |
| E4 | Technician | null |
| E5 | Technician | 2500 |
| E6 | Secretary | null |

(d)

**Table 4: The different models for the missing data (a) correct data (b) missing values follow MCAR model (c) missing values follow MAR model (d) missing values follow NMAR model.**

$\tilde{T}_{A_i=v}$ contains a subset $T_s$ that represents a good sampling of $\tilde{T}$ [8]. Let $\boldsymbol{v} = (v_1, \ldots, v_s)$ be a tuple, the mutual information between the values $v_1, \ldots, v_s$ in a given table $\tau$ is computed as:

$$I_\tau(v_1, v_2, \ldots, v_s) = \log \frac{P_\tau(v_1, v_2, \ldots, v_s)}{\prod_{i=1}^{s} P_\tau(v_i)}, \tag{2}$$

where $P_\tau(v_1, v_2, \ldots, v_s)$ is the ratio between the number of tuples that contains the values $v_1, v_2, \ldots, v_s$ in $\tau$ and the size of $\tau$. The function $I_\tau$ takes values in the interval $(-\infty, \infty)$. If the subtable $\tilde{T}_{A_i=v}$ represents a good sampling of the table $\tilde{T}$ then the average mutual information of the values in $\tilde{T}_{A_i=v}$ will be close to the average mutual information of the values in $\tilde{T}$. After computing the mutual information between the values that belongs to $\tilde{T}_{A_i=v}$ with respect to $\tilde{T}_{A_i=v}$ and $\tilde{T}$, we define a normalized score value that measures how good the sample $\tilde{T}_{A_i=v}$ is *w.r.t.* $\tilde{T}$ as follows:

$$S = \frac{1}{|\boldsymbol{v} \in \tilde{T}_{A_i=v}|} \sum_{\boldsymbol{v} \in \tilde{T}_{A_i=v}} \frac{1}{1 + |I_{\tilde{T}}(\boldsymbol{v}) - I_{\tilde{T}_{A_i=v}}(\boldsymbol{v})|}. \tag{3}$$

The score $S$ takes values in the interval $[0, 1]$ with values close to 1 when the subtable $\tilde{T}_{A_i=v}$ represents a good sampling of the table $\tilde{T}$, which indicates that the value $v$ is highly likely a DMV.

Note that the subspace in which the subset of $\tilde{T}_{A_i=v}$ represents a good sample of $\tilde{T}$ is unknown, thus requiring the examination of all possible subspaces. However, this process would be costly in terms of time complexity as it requires exponential time complexity with respect to the number of attributes. To tackle this issue, we propose the two following approaches.

In the first approach, we use the ratio between the number of distinct values to the number of tuples in each attribute as an indicator for the attribute to be included in the score computation process. For attribute $A_i$, let $\rho_{A_i}, \rho'_{A_i}$ be the ratios between the distinct values and the number of tuples in table $\tilde{T}$ and in subtable $\tilde{T}_{A_i=v}$, respectively. An attribute $A_j$ will help in discovering if $\tilde{T}_{A_i=v}$ contains a good sample of $\tilde{T}$ if $\Gamma_j = \frac{\rho'_{A_j}}{\rho_{A_j}}$ is close to 1. In our search for the best subspace, we sort the attributes using the distance between their $\Gamma$ value and 1, select the first two attributes and compute the DMV score $S$. We add more attributes and recompute the DMV score $S$ until including all the attributes and we select the maximum score value as recommended by [2].

In the second approach, we simply use a multilevel index of the data. Since computing $P_\tau(v_1, v_2, \ldots, v_s)$ will be done frequently, the multilevel table index helps in reducing the running time significantly. The index contains the set of distinct values together with the subtable that is produced by selecting the records that include the value in the original table. This index has a linear space complexity with respect to the number of attributes in each table. However, it reduces the running time of RandomDMVD significantly, as shown in our experiments.

# 6 EVALUATION

To evaluate FAHES, we compare its detection effectiveness, in terms of of precision and recall, with two baseline methods. The first is DiMaC [14, 15], which is designed to detect the DMVs that follow the MCAR/MAR models. DiMaC reports a value $v$ in attribute $A$ as DMV if $\sigma_{T_{A=v}}$ contains an embedded unbiased sample (EUS) of the table $T$. The correlation between the values in $\sigma_{T_{A=v}}$ and $T$ is used to measure the goodness of the sample. The second is *dBoost* [27], which can detect both numerical and syntactic outliers. dBoost applies a set of transformations to the data in each attribute to find a set of rules that describe the bulk of the data in that attribute. Values that do not conform with the found set of rules are flagged as outliers.

We also tested the local outlier factor (LOF) [6] and the local outlier correlation integral (LOCI) [25], which are two popular outlier detection methods. However, due to their poor results as they are not designed to detect DMVs and because of space limitation, we omit their results in this evaluation.

In this evaluation, we used 32 data sets from public and private data repositories. The variations in the data sources used in the evaluation reflect the generality of our solution for detecting the DMVs. We manually annotated the DMVs in each data set.

**UCI-ML repository:** It includes around 399 data sets which are mainly used by the ML community. We used 4 tables from this repository. The average table size is 53047 tuples and 11 attributes.

**data.gov:** It contains more than 200$k$ open data sets from different US government agencies on domains such as education, finance, environment and health care. We used 10 tables from this repository. The average table size is 3825 tuples and 13 attributes.

**data.gov.uk:** It is the UK counterpart of the previous repository with over 30$k$ data sets. We used 3 tables from this repository. The average table size is 194696 tuples and 23 attributes.

**mass.gov:** Mass.gov is a website that provides access to open data in the state Massachusetts. We used 6 tables from this repository. The average table size is 43391 tuples and 22 attributes.

**MIT DWH:** The MIT data warehouse is a private repository that includes 2,400 tables. We used 9 tables from this repository from

| **Source** | **# Tables** | **DiMaC** | | **dBoost** | | **FAHES (NE)** | | **FAHES** | | **Syntactic Outliers DMVs** | | **Numerical Outliers DMVs** | | **Random DMVs** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **P** | **R** | **P** | **R** | **P** | **R** | **P** | **R** | **P** | **R** | **P** | **R** | **P** | **R** |
| UCI ML-Repo. | 4 | 0.17 | 0.333 | 0.004 | 0.286 | 0.327 | 0.857 | 0.384 | **0.952** | 0.528 | 0.905 | 0.833 | 0.238 | 0.375 | 0.429 |
| data.gov | 10 | 0.242 | 0.644 | 0.038 | 0.356 | 0.321 | 0.911 | 0.484 | **0.978** | 0.620 | **0.978** | 0.667 | 0.0444 | 0.596 | 0.689 |
| data.gov.uk | 3 | 0.075 | 0.217 | 0.005 | 0.652 | 0.222 | **0.870** | 0.385 | **0.870** | 0.667 | **0.870** | 0.750 | 0.130 | 0.192 | 0.217 |
| MIT DWH | 9 | 0.045 | 0.214 | 0.0005 | 0.077 | 0.135 | 0.571 | 0.371 | **0.929** | 0.500 | 0.714 | 0.800 | 0.286 | 0.188 | 0.214 |
| mass.gov | 6 | 0.235 | 0.388 | 0.003 | 0.2 | 0.333 | 0.575 | 0.522 | **0.725** | 0.633 | 0.475 | 0.500 | 0.038 | 0.532 | 0.413 |

**Table 5: The detection effectiveness of the evaluated methods when applied to detect the DMVs on different data sets.**

the subset of tables which are available to MIT researchers. The average table size is 11760 tuples and 14 attributes.

## 6.1 Detection Effectiveness

The detection effectiveness of the different methods is measured in terms of

$$precision(P) = \frac{\#true\ DMVs\ detected}{\#reported\ DMVs}$$

and

$$recall(R) = \frac{\#true\ DMVs\ detected}{total\ number\ of\ DMVs}.$$

The number of DMVs is the number of distinct values that are used to disguise the missing values (we ignore the frequency of the values when counting the DMVs). We performed several experiments to show the ability of FAHES to detect the different types of DMVs. We summarize all the results in Table 5.

The first experiment compares the precision and recall of FAHES with the two baseline methods, namely DiMaC and dBoost. DiMaC outperforms dBoost in all of the data sets except for those in data.gov.uk since most of the DMVs in the latter are syntactic outliers which are easily detectable by dBoost. Since dBoost is designed to detect outliers, it reported many outliers that are not DMVs. FAHES outperforms both methods in terms of precision and recall for all data sets. This experiment shows that detecting DMVs requires special handling where out-of-the-shelf solutions would perform poorly.

The second experiment breaks down the different components of FAHES to show how each component behaves. While the numeric outlier detection module has better precision, its recall is the lowest as it is restricted only to numerical attributes. Many of the DMVs we encountered in our data sets are categorical. The syntactic outlier detector has the best recall and a good precision compared with the other modules in FAHES. However, for data sets from mass.gov, its recall is not that good. RandomDMVD shows better effectiveness than DiMaC which detects the same type of DMVs. The better effectiveness is due to the better subspace selection and the score combination function where we use the maximum instead of the average used by DiMaC. This experiment shows clearly that the three modules of FAHES are important to have a robust system for detecting DMVs.

The third experiment shows the importance of applying ensemble technique in improving the system's quality. In this experiment, we run FAHES with the best parameter setting without DMVs ensembles (FAHES (NE) in Table 5) for each repository and compare the results with those obtained using FAHES with DMVs ensembles. The results clearly shows that FAHES with DMVs ensembles outperforms FAHES (NE) since the best parameter setting works fine for some data sets within each repository but not for all data sets. Overall, it is important to note that since the number of DMVs to be validated is small, high recall must be achieved at the expense of precision.

| **Data set** | **Rows** | **Columns** | **DiMaC** | **dBoost** | **FAHES** |
|---|---|---|---|---|---|
| Pima Ind. Dia. | 768 | 8 | 213 | 1.29 | 0.039 |
| Adult {UCI} | 32561 | 15 | (+1) h | 102 | 11.070 |
| DOE H. School Perf. | 437 | 18 | 694 | 2.77 | 0.297 |
| SFO Museum Exhib. | 1242 | 16 | (+1) h | 4.77 | 0.767 |
| Avg. Daily Traf. Counts | 1279 | 9 | 42.3 | 2.22 | 0.197 |
| Website Analytics | 3366 | 10 | (+1) h | 5.53 | 0.638 |
| Employee Directory | 15946 | 21 | (+1) h | 87 | 10.5 |
| Accidents 2015 | 140056 | 32 | (+1) h | 1958 | 431 |

**Table 6: Running time (sec) for DiMaC, dBoost and FAHES when detecting the DMVs on different repositories ((+1) h means the method took more than an hour to report DMVs).**

## 6.2 Detection Efficiency

Another important factor of the usability of a given method in detecting DMVs is its running time. In this experiment, we show the running time incurred by the evaluated methods when detecting the DMVs on specific data sets. DiMaC has quadratic time complexity *w.r.t.* the number of attributes. It has quadratic time *w.r.t.* number of tuples in the worst case analysis. Its running time also depends on the number of distinct values in each attributes. The running time in FAHES is dominated by the RandomDMVD module. It has linear time complexity *w.r.t.* number of attributes and $O(N \log N)$ *w.r.t.* number of tuples due to the multilevel index, which speeds up the calculation of the score in Eq.(3).

Table 6 shows the running time in seconds for the three evaluated methods. Each experiment is run for a maximum of one hour and report "(+1) h" if the method does not report the results in less than an hour. The running time depends on the number of tuples, attributes and frequent values. From the results, we can see that FAHES is three and more orders of magnitude faster than DiMaC depending on the size of the table. The running time of dBoost is also 5-33 times larger than the running time of FAHES.

# 7 RELATED WORK.

We categorize the related work as follows.

Disguised missing values. The DMVs problem was first introduced in [26]. A specific type of DMVs that follows missing-at-random (MAR) model was studied in [14, 15]. We leverage this model for our detection module for inlier DMVs but we outperform it both in terms of efficiency and effectiveness.

*Outlier Detection.* The literature abounds with outlier detection methods which differ in their view of what outliers are and the way to find them. Statistical-based approaches assume the data is extracted from a given distribution with unknown parameters and proceed to find the missing parameters. Data points that do not fit in the statistical model are reported as outliers. For example, in a normal distribution, a data point $p$ is reported as an outlier if it deviates more than $3\sigma$ from the mean $\mu$ [11]. A distance-based outlier detection method was proposed in [17]. In this method, an object $p$ in a data set $D$ is a $DB(\pi, d_{min})$-outlier if at least percentage $\pi$ of the objects in $D$ lies within a distance greater than $d_{min}$ from $p$. Many variants of the distance-based outlier detection method have been proposed as well, *e.g.,* [5, 31]. The main problem in this approach is that it requires prior knowledge about the data application in order to efficiently set the parameters $(\pi, d_{min})$. The high computational time is also another problem with this approach. Another popular set of approaches is density-based, where a given data point is reported as an outlier if the density in its neighborhood is too different from the densities around its neighbors. Local Outlier Factor [6] and Local Correlation Integrals [25] are examples of density-based outlier detection techniques. Outlier detection using probability density functions has been shown to outperform other popular outlier detection methods in terms of time efficiency and detection effectiveness [19, 29]. The simplicity of parameter setting for this method is another advantage. The numerical outlier methods proposed in this paper use a modified version of the work in [29].

*General error detection.* As pointed out by [1], real-world data is dirty, there are many different types of data errors, and more importantly, existing tools are not robust enough to capture most errors in real-world datasets. Even with the recent advancements of general-purpose data cleaning tools [9, 16], robustly detecting data errors is still a long-standing research problem. FAHES made a firm step for detecting DMVs, which can be easily deployed by existing data cleaning tools.

## 8 CONCLUSION AND FUTURE WORK

We presented FAHES, an end-to-end system to detect DMVs from two different angles: DMVs as detectable outliers and as detectable inliers. For the former case, we proposed a syntactic outlier detection module for categorical data, and a density-based outlier detection module for numerical values. For the latter case, we proposed a method for detecting DMVs that follow either MCAR/MAR models. We also applied an ensemble method to strengthen the results delivered by these different modules. Our extensive experiments using real-world data sets show that FAHES deliver results that are way better than existing solutions in detecting DMVs.

One future work we are planning to perform is to improve FAHES to detect the DMVs that are generated randomly within the range of the data. For example, when a child tries to create an account on a domain that has a minimum age restriction, the child fakes her age with a random value that allows him to create the account. Such random fake values are hard, if not impossible, to detect. Moreover, although DMVs are the focus of this paper, there are more types of errors are found in the wild. Many of the principles and techniques we have used to detect DMVs can be leveraged to detect other types of errors, so a natural next step is to extend the infrastructure we have built to detect those. This opens new challenges related to the robust identification of errors that could be interpreted differently by different modules.

## REFERENCES

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: Where are we and what needs to be done? *PVLDB*, 9(12):993–1004, 2016.

[2] C. C. Aggarwal. Outlier ensembles: Position paper. *SIGKDD Explor. Newsl.*, 14, 2013.

[3] C. C. Aggarwal and S. Sathe. Theoretical foundations and algorithms for outlier ensembles. *SIGKDD Explor. Newsl.*, 17:24–47, 2015.

[4] P. Allison. *Missing Data.* SAGE Publications Inc., 2001.

[5] F. Angiulli and F. Fassetti. Detecting distance-based outliers in streams of data. In *CIKM'07*, 2007.

[6] M. Breunig, H. Kriegel, R. Ng, and J. Sander. Lof: Identifying density-based local outliers. In *SIGMOD'00*, 2000.

[7] P. Bühlmann. Bagging, subagging and bragging for improving some prediction algorithms. *Recent advances and trends in nonparametric statistics*, pages 927–961, 2002.

[8] K. W. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Comput. Linguist.*, 16:22–29, 1990.

[9] M. Dallachiesa, A. Ebaid, A. Eldawy, A. K. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, pages 541–552, 2013.

[10] A. Eckert-Gallup and N. Martin. Kernel density estimation (kde) with adaptive bandwidth selection for environmental contours of extreme sea states. IEEE Monterey OCEANS 2016 MTS, 2016.

[11] D. H. Freedman, R. Pisani, and R. Purves. *Statistics.* W. W. Norton & Company., 1978.

[12] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37:302–320, 1978.

[13] V. J. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22:85–126, 2004.

[14] M. Hua and J. Pei. Cleaning disguised missing values: A heuristic approach. In *KDD'07*, 2007.

[15] M. Hua and J. Pei. DiMaC: A disguised missing data cleaning tool. In *KDD'08*, 2008.

[16] Z. Khayyat, I. F. Ilyas, A. Jindal, S. Madden, M. Ouzzani, P. Papotti, J. Quiané-Ruiz, N. Tang, and S. Yin. Bigdansing: A system for big data cleansing. In *SIGMOD*, pages 1215–1230, 2015.

[17] E. Knorr and R. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB'98*, 1998.

[18] Y. Kwak, Y. Yang, and S. Park. Missing data analysis in drug naive alzheimer's disease with behavioral and psychological symptoms. *Yonsei Med J.*, 54:825–831, 2013.

[19] L. J. Latecki, A. Lazarevic, and D. Pokrajac. Outlier detection with kernel density functions. MLDM'07, 2007.

[20] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish. Regular expression learning for information extraction. EMNLP'08, 2008.

[21] M. Lichman. UCI machine learning repository, 2013.

[22] R. Little and D. Rubin. *Statistical Analysis with Missing Data, 2nd Edition.* WILEY, 2002.

[23] J. Luengo, S. García, and F. Herrera. On the choice of the best imputation methods for missing values considering three groups of classification methods. *Knowl. Inf. Syst.*, 32:77–108, 2011.

[24] S. Padhi, P. Jain, D. Perelman, O. Polozov, S. Gulwani, and T. D. Millstein. Flashprofile: Interactive synthesis of syntactic profiles. *CoRR*, abs/1709.05725, 2017.

[25] S. Papadimitriou, H. Kitagawa, P. Gibbons, and C. Faloutsos. Loci: Fast outlier detection using the local correlation integral. In *ICDE'03*, 2003.

[26] R. Pearson. The problem of disguised missing data. *SIGKDD Explor. Newsl.*, 8:83–92, 2006.

[27] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden. Outlier detection in heterogeneous datasets using automatic tuple expansion.

[28] A. Qahtan, A. Elmagarmid, M. Ouzzani, and N. Tang. FAHES: Detecting disguised missing values. ICDE'18, 2018.

[29] A. Qahtan, X. Zhang, and S. Wang. Efficient estimation of dynamic density functions with an application to outlier detection. In *CIKM'12*, 2012.

[30] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. Snorkel: Rapid training data creation with weak supervision. 11(3):269–282, 2017.

[31] B. Sheng, Q. Li, W. Mao, and W. Jin. Outlier detection in sensor networks. In *MobiHoc'07*, 2007.