Temporal Rules Discovery for Web Data Cleaning

Ziawasch Abedjan[§] Cuneyt G. Akcora[♯] Mourad Ouzzani[♯]
Paolo Papotti[♯] Michael Stonebraker[§]
[♯] Qatar Computing Research Institute, HBKU [§] MIT CSAIL
{cakcora,mouzzani,ppapotti@qf.org.qa} {abedjan,stonebraker@csail.mit.edu}

ABSTRACT

Declarative rules, such as functional dependencies, are widely used for cleaning data. Several systems take them as input for detecting errors and computing a "clean" version of the data. To support domain experts,in specifying these rules, several tools have been proposed to profile the data and mine rules. However, existing discovery techniques have traditionally ignored the time dimension. Recurrent events, such as persons reported in locations, have a duration in which they are valid, and this duration should be part of the rules or the cleaning process would simply fail.

In this work, we study the rule discovery problem for temporal web data. Such a discovery process is challenging because of the nature of web data; extracted facts are (i) sparse over time, (ii) reported with delays, and (iii) often reported with errors over the values because of inaccurate sources or non robust extractors. We handle these challenges with a new discovery approach that is more robust to noise. Our solution uses machine learning methods, such as association measures and outlier detection, for the discovery of the rules, together with an aggressive repair of the data in the mining step itself. Our experimental evaluation over real-world data from Recorded Future, an intelligence company that monitors over 700K Web sources, shows that temporal rules improve the quality of the data with an increase of the average precision in the cleaning process from 0.37 to 0.84, and a 40% relative increase in the average F-measure.

1. INTRODUCTION

With the increasing availability of web data, we are witnessing the proliferation of businesses engaged in automatic data extraction from thousands of web sources with the goal of gleaning useful information and intelligence about people, companies, countries, products, and organizations [30]. It is well recognized that the data cannot be used *as-is* because of errors that are in the sources themselves [15, 28, 29, 33] or that arise with automatic extractors [7, 13].

Proceedings of the VLDB Endowment, Vol. 9, No. 4

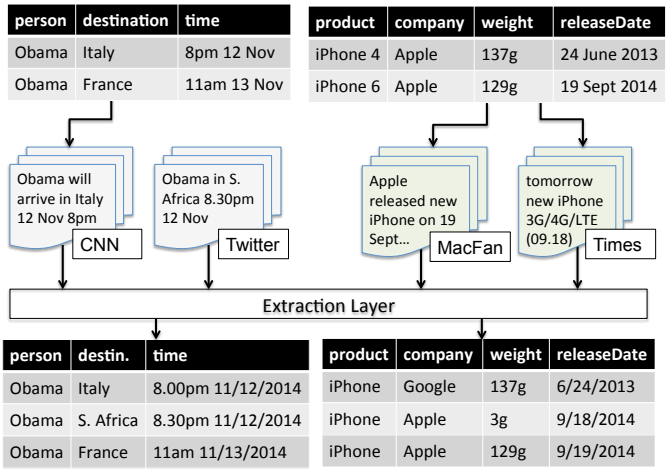


Figure 1: From top to bottom: real facts, their representation on the Web, and the extracted data.

Consider the example depicted on the left hand side of Figure 1. Obama attended a dinner in Italy, on Nov 12^{th} 2014 at 8 pm; this is a *real event* and is represented as a tuple in the relation at the top of the Figure. The information is correctly reported on a web page from *CNN* and a web data extractor identifies that a person ("Obama") was in a location ("Italy") at a certain time; this is an *extracted event* (relation at the bottom). However real events can be reported by multiple sources that may or may not agree on the details. In fact, another source reports Obama in South Africa on the same day. As it is unlikely that a person is reported in two different countries within 30 minutes, such a contradiction highlights a problem in the data. In this case, the event was extracted from a social media outlet that did not have a faithful knowledge about the real event. This happens in practice and is studied as the problems of *truth discovery* [33, 15, 28] or *fact-checking* [29]. By enforcing a rule over information from multiple sources, it is possible to gain understanding about the trustability of the sources and, ultimately, about the correct value of interest.

Consider another example for releases of products in the right hand side of Figure 1. The information reported by the source *Times* is a real event, but an error in the extractor led to an incorrect weight, namely *3g*, which in fact is a type of network supported by the phone. Detecting such problems can help identify faulty extractors [7, 13].

The two above examples highlight that identifying quality problems enables analysis over the sources, the data val-

ues, and the extractors. These analytics tasks usually rely on declarative rules (such as key constraints) for detecting problems in the data. For example, the fact that a product is always released by a company can be expressed with a functional dependency (FD), i.e., product → company. In the above example, the company releasing the phone cannot be both Google and Apple. Heuristics exploiting the redundancy are usually used to determine the correct value (the truth) [28]. However, there are other errors that can be identified only through *temporal functional dependencies*, which are FDs that restrict the rule on the temporal dimension [21]. For example, a domain expert may come up with a rule stating that a person cannot be reported arriving in two countries at the same time (person → destination in a 1-hour window), or that the same product cannot have different weights reported at the release date (product → weight in a 6-month window).

Coming up with these rules with the correct duration value for "same time" is not trivial. A conservative choice for this duration in a rule, such as "within a minute", leads to undetected errors in the Obama example. On the contrary, a high value for duration, such as "two days", does capture the problem, but would mark as errors all the tuples in the example, including the correct ones with Italy and France. Similar challenges arise for product release, a time window of one day for the weight would not capture the problem with *3g*. Moreover, durations depend on the entity at hand. For instance, Obama travels more frequently and faster than most people, so he should have a different temporal rule with a smaller time window.

Discovering constraints has been well studied in the literature [32, 1, 23, 8, 19]. However, a recurring assumption in these existing techniques is that data is either clean or, at worst, has a small amount of noise. Obviously, such assumptions do not hold for data extracted from the web due to the compounded effects of noise coming from the sources and errors made by the extractors. Moreover, even when it is possible to mine approximate dependencies over such dirty data, there is no algorithm to discover useful time-windows, or *durations*, to identify errors for the different events, e.g., a person is not reported traveling to two countries in a *1-hour* window. Without such a time dimension, rules are not usable, as discussed above.

In this paper, we present AETAS[1], our solution to overcome the above challenges by relying on two basic concepts: (i) the notion of approximation for the discovery of functional dependencies that hold for most of the data, and (ii) outlier detection techniques for the discovery of the durations. In a nutshell, we first create a set of approximate FDs that are valid in the smallest meaningful time interval. The dependencies are then ranked with an association measure, and validated by human experts. For each validated rule, we create a distribution of durations for all the objects in the data, e.g., how much time is observed within two consecutive destinations for every person, and mine it to compute the duration that identifies the lowest extreme values. This duration is then used as the time window for the rule to identify temporal outliers.

Our contributions are as follows:

1) We formulate the problem of discovering temporal functional dependencies for data cleaning (Section 2), and present techniques to discover approximate FDs based on statistical properties of the data (Section 3).

2) Given a rule, we mine the duration that lead to identifying temporal outliers. We tackle the problem of the sparseness of the data with value imputation, and reduce the noise by enforcing the rule in the smallest meaningful time bucket (Section 4). We also mine rules with constants (akin to conditional functional dependencies) such that specific durations can be used for specific entities.

3) We show over real and synthetic datasets that our techniques for approximate dependencies and duration discovery outperform alternative approaches in terms of quality. In particular, our durations lead to improvement in the data cleaning process compared to FDs, with an increase of the average precision in the repair of the temporal data from 0.37 to 0.84, and a 40% relative increase in the average F-measure (Section 5). Moreover, our technique discovers durations that lead to higher F-measure than the baselines, including the durations collected from a group of users.

We discuss related work in Section 6, and in Section 7 we draw some conclusions and list directions for future work.

2. RULE DISCOVERY FOR WEB DATA CLEANING

We first describe the kind of web data we are dealing with. We then define the syntax and semantics of temporal dependencies, give a definition of data cleaning, and define the problem of the rule discovery for web data cleaning.

From web pages to structured data. We are interested in event data collected from the Web by monitoring news media. Examples of such data include GDELT (gdeltproject.org) and Recorded Future (www.recordedfuture.com). Given a web page, the organization in charge of the event database runs extractors to produce structured data for different events. Examples of events include people traveling to destinations, company acquisitions, and occurrences of armed attacks. Figure 1 exemplifies data extracted from text in six web pages: three occurrences for event *PersonTravel* with person *Obama* as the only entity, and three occurrences for *ProductRelease* with product being the entity *iPhone*. In general, an entity may be an instance of a person, a location, a company, and so on. In the following, we assume that entity recognition from the text has been already performed. In addition to the event type specific attributes, *e.g.,* company, destination, all events have a timestamp attribute, such as time and releaseDate. We assume that all of these attributes may contain errors.

Temporal Functional Dependencies. We focus on a specific form of temporal functional dependencies similar to those described in [21]. We assume a total ordering on the time attribute t, and that there is a mapping $f()$ that linearizes the different time values into integers. For example, the value r[t] = (h,m,s) could be mapped to seconds via f(h,m,s) = 3600h + 60m + s. A *time interval* Δ is a pair with a minimum and a maximum value (for examples in hours), m and M, respectively, with $m \leq M$.

Given the pair $< U, t >$ with a fixed set $U = \{A_1, \ldots, A_n\}$ of event type attributes and the time attribute t, a tuple over $< U, t >$ is a set of $< r = \{A_1 : c_1, \ldots, A_n : c_n\}, t : c_t >$, where c_i is a constant. A relation I is a finite set of tuples over $< U, t >$.

[1]From "Omnia fert aetas", *Time cancels everything.*

Definition 1. *Let X, Y be two subsets of attributes from U, Δ a time interval, and π the permutation of rows of I increasing on the time value. A* temporal functional dependency *(TFD) over U is an expression $X \wedge \Delta \rightarrow Y$ that is satisfied if for all pairs of tuples $r_\pi, r_{\pi+1} \in I$, s.t. $r_{\pi+1}[t] - r_\pi[t] \in \Delta$, when $r_\pi[X] = r_{\pi+1}[X]$, it is the case that $r_\pi[Y] = r_{\pi+1}[Y]$.*

The subsets of attributes X and Y are referred to as left-hand side (LHS) and right-hand side attributes (RHS), respectively. When referring to values of X and Y attributes, we shall use the terms *reference value* and *attribute value*, respectively.

Example 1: The rule "a product cannot be released with two different weights in a time window of a year" defined over event *ProductRelease* can be stated as follows: product$\wedge$ (0, 1 year) $\rightarrow$ weight, where Δ is the pair $m = 0$ and $M =$ *1 year*. In Figure 1, *ProductRelease* events show conflicting weight values *3g* and *129g* on release dates *09/18/2014* and *09/19/2014* for product *iPhone*. □

While most of the entities for a given event abide by the same duration in a rule, some entities may require specific duration values. For example, in the case of *ProductRelease* events, new *iPhone* models are sometimes released with an interval of time shorter than a year, while for cars the interval is much longer (*e.g.,* BMW X5 car model is renewed every 6 years). Thus, in the same spirit of conditional function dependencies (CFDs) [5], we are also interested in TFDs that apply on subsets of tuples. We therefore extend the language to consider constant selections in the left-hand side, such as product["*iPhone*"] $\wedge$ (0, 8 months) $\rightarrow$ weight. This is equivalent to having views for specific entities and applying the TFD on the view induced by the selection.

Data Repairing. While TFDs can be used in multiple applications, such as database design, our focus is on data quality scenarios. Data repairing is the application we will use in the following to evaluate the quality of the discovered dependencies. Given a database instance I of schema $\mathbb{R}$ and a dependency φ, if I satisfies φ, we write $I \models \varphi$. If we have a set of dependencies Σ, $I \models \Sigma$ if and only if $\forall \varphi \in \Sigma, I \models \varphi$. A *repair* I' of an inconsistent instance I is an instance that satisfies Σ. A repair solution is not unique, as discussed in the following example.

Example 2: Consider a different instance D for *ProductRelease* and the FD d_1 : product $\rightarrow$ company. Value errors are reported in bold.

D	product	company	weight	releaseDate
t_1 :	iPhone	Apple	137g	**10am 6/24/2014**
t_2 :	iPhone	**Google**	129g	3pm 9/18/2014
t_3 :	iPhone	Apple	129g	4pm 9/19/2014

If we check the dependency over the data, we get the following pairs of violating tuples: (t_1,t_2),(t_2,t_3).
Two possible, alternative repairs are $R_1 - R_2$, as follows:

R_1	product	company	weight	releaseDate
t_1 :	iPhone	Apple	137g	**10am 6/24/2014**
t_2 :	iPhone	*Apple*	129g	3pm 9/18/2014
t_3 :	iPhone	Apple	129g	4pm 9/19/2014

R_2	product	company	weight	releaseDate
t_1 :	iPhone	*Google*	137g	**10am 6/24/2014**
t_2 :	iPhone	**Google**	129g	3pm 9/18/2014
t_3 :	iPhone	*Google*	129g	4pm 9/19/2014

Updates (in italic) in R_1 and R_2 make the new instance valid for d_1. An alternative repair strategy deletes tuple t_2 in R_1 or tuple t_1, t_3 in R_2.

Consider also a TFD d_2: product["*iPhone*"] $\wedge$ (0, 8months) $\rightarrow$ weight. A possible repair for violating tuples (t_1,t_2), (t_1,t_3) is by updating the value of weight for t_1 to *129g*, or to delete t_1. □

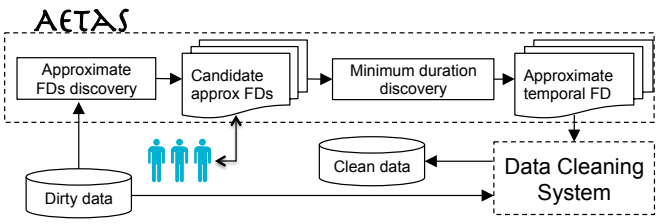


Figure 2: Architecture of the AETAS system.

Since the number of possible repairs is usually large and possibly infinite, a minimality principle is oftentimes used to identify desirable repairs for the *data cleaning problem* [22]: given a database I and a set of dependencies Σ, compute a *repair* I_r of I such that $I_r \models \Sigma$ (consistency) and their distance *cost*(I_r, I) is minimal (accuracy). Depending on the distance function, the desired repair is the one with the minimal number of cell updates, or the one with minimal number of tuple deletions. Computing minimal repairs is NP-hard to be solved exactly for FDs [4, 24] which led to several heuristics-based methods [10, 12, 16, 24].

Discovering temporal dependencies. Given a relational schema $\mathbb{R}$ and an instance I, the discovery problem for TFDs is to find all valid TFDs that hold on I. Since web data is noisy in nature, we are interested in the approximate version of the problem, i.e., find all valid TFDs, where a rule r is valid if its support has a value higher than a given threshold δ. To solve this problem, we developed AETAS, a system to discover TFDs from web data.

Figure 2 shows the architecture of the system and the main steps in our solution. Given a noisy dataset, we first discover approximate functional dependencies, i.e., traditional FDs that hold on most of the relation within a given atomic duration t_α. The use of the atomic duration removes the temporal aspect of the relation so that dependencies can be discovered purely in terms of record attributes.

Given a set of approximate FDs, we rank them according to their support to assist the user in their validation. A user can either reject a suggested approximate FD, or validate it as being a simple FD or a TFD. For a validated TFD, we then discover its corresponding time interval, including values that only hold for specific entities as we discussed previously. Since the data is dirty, we cannot just examine consecutive occurrences for each entity and collect the minimum duration. Therefore, we compute the distribution of the durations and mine it to identify the minimum duration that would eventually cut-off the outlying values, *i.e.,* data that is invalid. This minimum duration is then assigned to M and, together with default $m = 0$, define Δ for the approximate FD at hand.

Finally, FDs and TFDs are fed to a constraint based data cleaning system, which takes the rules and the noisy data as input and outputs a consistent updated dataset.

3. FD DISCOVERY OVER DIRTY DATA

Two main characteristics of web data prevent us from using traditional dependency discovery algorithms. First, most of these algorithms assume that the data is clean. As we work with dirty web data from multiple independent sources, this assumption does not hold. There have been some work to tolerate some dirtiness up to a certain threshold on the percentage of not conforming tuples [8, 19]. However, dirtiness in real (web) data is so high that the corresponding threshold leads to the discovery of very general rules that are not valid in practice. For example, our test dataset has noise up to 26% wrt the number of tuples (Table 1 in Section 5). A threshold of 26% leads to the discovery of several key constraints and multiple functional dependencies that do not hold semantically.

Second, temporal data contains reference values that change over time, such as Obama with correct values "Italy" and "France" at two different timestamps. Because of this temporal nature, traditional FDs do not apply over the relations with extracted data for many events.

We introduce next how we model the data and then how we tackle the above problems with our approach for discovering approximate FDs.

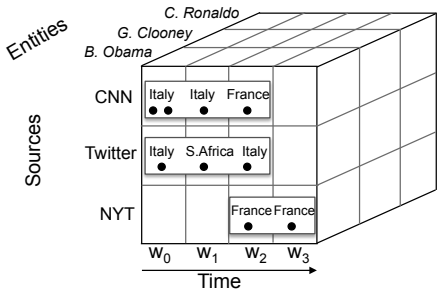


Figure 3: A dependency cube: w_0 is a 1-day time bucket, s_0 is source *CNN*, a_0 is entity *B. Obama*, {*Italy, S. Africa, France*} are attribute values.

Dependency cube. We start by considering all the possible dependencies with one attribute in the LHS and one attribute in the RHS. For each potential dependency $\mathsf{X} \rightarrow \mathsf{Y}$, we make the time dimension (attribute t) discrete by creating time buckets with the size of an atomic time duration. Given these time buckets, we define a *dependency cube* over four data dimensions: data sources S, time buckets W, reference values X, and attribute values Y. Figure 3 shows a dependency cube for the Obama example:

- The x axis is divided into homogeneous time buckets $w_i \in W$ (e.g., 1 hour).
- The z axis reports different reference values $x \in X$ (e.g., B. Obama, iPhone).
- The y axis reports different sources $s_i \in S$ (e.g., CNN).
- The reported cell values for a certain time bucket, source, and entity are attributes values $y \in Y$ (e.g., Italy, Apple).

The size of each dimension in this cube can be large. For example, Recorded Future continuously collects data from more than 0.7 Million web sources. However, due to how events are reported on the web, the data is very sparse and the size of the dependency cube is manageable in practice.

For a reference value $x \in X$, the sequence of values of Y reported by one source over time constitutes a stripe. For example, in Figure 3, source NYT reports two Y values for reference value $B.Obama$ in time buckets w_2 and w_3. For a reference value $x \in X$, the union of stripes from all sources constitutes a plate. For a time bucket w_i of a given dependency over R, we define the time slice R_i as the list of Y values for all X values reported within the time bucket w_i.

A potential dependency holds for the cube if, for each plate and for each stripe, it is true that $\mathsf{X} \rightarrow \mathsf{Y}$. If the dependency holds only for a specific plate for reference value x_i (i.e., a specific entity), then it is a constant dependency $\mathsf{X}[x_i] \rightarrow \mathsf{Y}$.

Implication discovery. Considering the aforementioned noise and temporal problems, we devise an algorithm that works with dirty, temporal data for the discovery of TFDs. We discover implications by first fixing an atomic time duration t_α such that we can mine the dependencies that hold within a time bucket. We observe that if a TFD holds for a certain duration Δ, it also holds for durations $\Delta' \leq \Delta$. The best bucket size is the smallest one that contains enough evidence to do the mining. Moreover, the value of the atomic duration cannot be more fine grained than the time granularity of the timestamps in the data. In our datasets, the granularity is up to milliseconds, but the data is too sparse to mine in such a small granularity, we therefore use $t_\alpha = 1\ hour$. The atomic duration t_α is application-dependent and is an input parameter for the algorithm.

Given a t_α value, we partition the data and create time slices $\mathcal{R} = \{R_1, R_2, ..., R_N\}$. Given a time slice R_i, we employ an association rule based method to detect 1-to-1 and many-to-1 implications. More specifically, we use *normalized pointwise mutual information* (NPMI) [6], a standard association measure in collocation extraction, for implication discovery. In a time slice, NPMI of a pair of reference-attribute values $x \in X$ and $y \in Y$ is defined as:

$$i(x,y) = ln(\frac{P(x,y)}{P(x) \times P(y)}) / -lnP(x,y)$$

where $P(x,y)$ is the joint probability of reference value x and attribute value y, and $P(x)$ is the marginal probability of reference value x. Intuitively, given a pair of outcomes x and y that belong to discrete random variables X and Y (assumed independent), the PMI quantifies the discrepancy between the probability of their coincidence given their joint distribution and their individual distributions. Its normalized version, NPMI, can have the following values: if x and y only occur together $i(x,y) = 1.0$, if x and y occur independently $i(x,y) = 0$, and -1 if they never co-occur. We use this score as an indicator of their correlation in the following.

We learn the implication $x \rightarrow y$ for a pair of values in the given time slice. In order to find the $X \rightarrow Y$ implication, we need to generalize the NPMI value over all value pairs of the two attributes. If the implication holds, we expect the NPMI value of each pair to be positive (i.e., the sign of the 1-to-1 implication) and, overall, $i(X,Y)$ close to 1.0. In fact, three factors can decrease NPMI values even in presence of real implications. First, multiple reference values can have the same attribute value in a given time slice (i.e., many-to-1 implication). For example, two persons can be in the same city at the same time. Second, because of a small bucket size, time slices may contain few instances about the same reference values. For example, in a bucket all events might be about Obama traveling to France. We employ a decision rule to overcome single reference value by assigning

$i(X,Y) = 1.0$ when $|X| = 1$. Third, dirty data can introduce different attribute values for the same reference value (i.e., 1-to-many occurrences), as in the example with Italy and S. Africa for Obama. As we assume dirtiness in the data, we need to tolerate this noise. Given these three possible causes, some value pairs have low NPMI values, thereby reducing the NPMI value of attributes, i.e., $i(X,Y) < 1.0$. This is a strong signal for the discovery of correlations and we exploit it in our algorithm.

From implications to dependencies. Given a list of NPMI values of multiple time slices, our next task is to decide whether the implication $X \rightarrow Y$ holds on enough time slices to be considered a dependency. Hence, we aggregate the expected value of NPMI values over time slices and compute a score. The score is then used to rank the output for user validation. Aggregation of NPMI values by expected value is weighted wrt the number of event instances (i.e., tuples) in time slices, such that an implication is penalized if it does not hold over time slices with large numbers of event instances.

To prune the number of results in the output, we also allow as input an optional user-defined significance threshold δ. In this case, we declare an implication to be a dependency if its aggregated score is higher than the threshold.

Example 3: Consider the case where an event R has instances distributed over a 36 hour period. Given $t_\alpha = 12\ hours$, we create three time slices R_1, R_2 and R_3, and compute their NPMI values to be 0.95, 0.3 and 0.5. Probabilities of event instances belonging to these time slices are $P(R_1) = 0.8$, $P(R_2) = 0.15$, and $P(R_3) = 0.05$. The expected NPMI value $E = 0.95 \times 0.8 + 0.3 \times 0.15 + 0.5 \times 0.05 = 0.83$. For $\delta = 0.7$, we assert that the implication $X \rightarrow Y$ holds. A value of 0.7 is usually used in practice [17]. □

Early Termination. If a threshold δ is defined, an implication can be declared to hold or to be pruned by considering a smaller number of slices. We thus stop NPMI computations if the remaining slices will not carry the expected score below or above the significance threshold δ. Consider the case when NPMI values of x out of n slices have been computed. In the best and worst cases, all the remaining slices can have NPMI values 1 or 0, respectively. If an implication does not hold even in the best case, or holds even in the worst case, we do not need to compute the NPMI values of the remaining slices. Otherwise we continue our computation. Formally, given a total of n slices for the implication $A \rightarrow B$, we terminate computations at the x^{th} slice with

$$\begin{cases} A \not\rightarrow B & \text{if } \delta > \sum_{j=1:x} i_j(A,B) \times P(j) + \sum_{k=x+1:n} i_k(A,B) \times P(k) \\ A \rightarrow B & \text{if } \delta \leq \sum_{j=1:x} i_j(A,B) \times P(j) \end{cases}$$

where δ is the significance threshold, $P(k)$ is the probability of an event instance being in the time slice k, and $i_k(A,B)$ is the NPMI value of the k^{th} slice.

Example 4: Consider the scenario of three time slices in Example 3. After computing the NPMI value of R_1 as 0.95, we can terminate computations for a significance threshold of $\delta = 0.7$ because the expected value $E = 0.95 \times 0.80 = 0.76$ is already above δ. □

Data: A relation R of attributes A, B; atomic time length t_α; (threshold δ)
Result: A score for the dependency

```
1  npmi = 0, current := 0;
2  Create time buckets R = {R_1, ..., R_n} of R with t_α;
   // Iterate on each slice;
3  foreach R' ∈ R do
4      current ← current + |R'|;
5      v = 0;
       // Decision rule;
6      if |R'.B| = 1 then
           // Add an NPMI value of 1.0;
7          v = 1.0;
8      else
9          foreach a ∈ R'.A do
10             foreach b ∈ R'.B do
11                 i(a,b) = ln(P(a,b)/(P(a)*P(b))) / -ln(P(a,b));
12                 v = v + P(a,b) × i(a,b);
       // Add expected value of the slice
13     npmi = npmi + v × |R'|/|R|;
       // Termination;
       // 1.  Dependency will not hold;
14     if δ ≠ null ∧ δ > npmi + (|R|-current)/|R| then
15         return 0;
       // 2.  Dependency will hold;
16     if δ ≠ null ∧ δ ≤ npmi then
17         return 1;
18 return npmi;
```

Algorithm 1: Implication detection with NPMI.

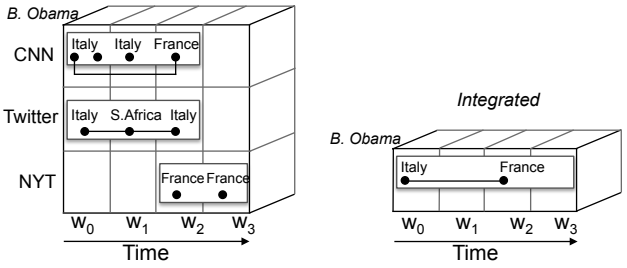
Algorithm. We now give a description of our approximate dependency discovery algorithm. Given two attributes A and B from an event R, we use Algorithm 1 to compute their NPMI score, or to find whether a dependency holds over the two attributes, if a threshold is given. The algorithm takes as input an atomic duration t_α, two attributes A, B from a relation R, and an optional significance threshold δ. The algorithm is called twice for each direction, namely $A \rightarrow B$ and $B \rightarrow A$. If an implication is found for the attributes, only one, or both of these dependencies may hold.

The algorithm starts by time bucketing the relation into smaller relations (Line 2). From Line 3, we find the strength of the implication within the slice. If the sub-relation contains a single attribute value, the decision rule in Line 2 assigns a NPMI value of 1.0 to the slice. Otherwise, we compute NPMI values of each (a,b) pair in Line 11. Line 12 adds the NPMI value to the expected value of the slice.

Once NPMI values of all pairs have been computed, the NPMI value of the slice is added to the expected value of the whole dependency in Line 13. If δ is defined, we check the termination conditions in Lines 14 and 16. In Line 14, we compute the NPMI value for the ideal case where all the remaining slices will be 1.0. Similarly, Line 16 checks whether the expected value is already above the threshold. In both cases, we stop the computations early if the condition holds and return 0 or 1 accordingly. If δ is not defined, we return the NPMI value for the dependency to be used for ranking and user's consumption. Once the user has selected the dependencies that are TFDs, we process then for duration discovery, as described in the next Section.

4. TIME DURATION DISCOVERY

Given an approximate FD $X \wedge \Delta \rightarrow Y$ with Δ_t=(0,t_α) for an event, the goal of duration discovery is to expand the atomic duration t_α to the correct minimum duration M in Δ. In the ideal case, there exists one and only one Δ such that no reference value $x \in X$ can change its attribute value $y \in Y$ within a time interval $(t_y, t_y + m)$, where t_y is the reported time of value y.



(a) Baseline durations (b) Projected durations

Figure 4: Stripes integration for duration discovery.

A naive approach for duration discovery is to take each stripe of a dependency cube, and find the time it takes for an attribute value to change from $y_1 \in Y$ to $y_2 \in Y$, i.e., $t_{y_2} - t_{y_1}$. This results in a list of time difference values. Then, the minimum value among all the time differences can be chosen as the M in Δ. This approach is shown in Figure 4(a) for a single plate, where value changes in stripes are highlighted.

A major assumption in the naive approach is that web sources correctly report attribute values. When the data comes from non-authoritative web sources, this assumption can easily be broken. A more robust approach is to exploit the evidence coming from multiple sources such that the accuracy of an attribute value can be "verified". The idea is to first repair the data within a time bucket with the evidence coming from multiple sources. We then compute durations on a "clean" integrated stripe, as in Figure 4(b). Below, we first describe how to repair data in the buckets and then give the full discovery algorithm.

Repair step. Given an approximate FD $A \rightarrow B$, we create a plate p for each reference value $a \in A$, and the plate is partitioned into time buckets of size t_α. Each bucket $w_n \in p$ has a time slice of attribute values R_n reported by sources where $|R_n| \geq |distinct(R_n)| \geq 1$. A new stripe I is created, where the results of the integration will be reported. In a bucket, if there exists a b' such that $mode(R_n) = b'$, then the corresponding w_n bucket for I is updated with b'. If there is no majority, the value in bucket w_{n-1} is assigned to w_n for I.

Figure 5 shows a window repair for three sources. In the figure, the value *Italy* from source *Twitter* is less frequent than *France*, so it is not in the result. Although our repair approach uses a simple majority voting scheme, any repair algorithm can be plugged into the system, for example by using truth discovery algorithms [15, 28, 29, 33] or by involving domain experts.

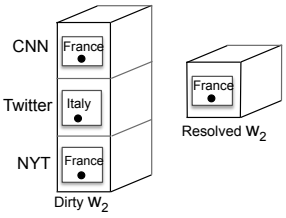


Figure 5: Repair step.

Time durations. Given the integrated plate, we compute a distribution of time durations between consecutive, distinct attribute values for every reference value. Figure 4(b) shows time durations on I, the stripe with the outcome of the repair step. Even with repaired values in time buckets, reference values have varying durations for the same temporal dependency. Two factors impact the observed durations:

- **Dirty data.** Sources can report conflicting values in two consecutive buckets that cannot be detected by local repairs. This problem raises many short durations. For example, the *Twitter* stripe in Figure 4(a).

- **Reporting frequency.** Although a reference value changes its value in the real world, sources may not report it. In our dataset, only a small set of entities, such as political leaders, have their changes reported frequently. This leads to some durations that are longer than the real time windows between two occurrences of an event.

As a result of these factors, time durations constitute a non-uniform distribution $D(x, y)$, with a range of $[t_\alpha, |W|]$. Our goal is to mine a duration that would remove the outlying values from this distribution.

Data: A dependency cube C for $A \rightarrow B$, a cut-off value c with $1 \leq c \leq 100$
Result: A time duration M

```
1  Define D(A,B) to be an empty duration list;
2  foreach plate p ∈ C do
3      Define I to be an empty stripe with |I| equals to
       the # of buckets in p;
4      foreach non-empty bucket w_i ∈ p with time slice R_i
       do
5          b' ← Mode(R_i);
6          if b' is not null then
7              update bucket w_i ∈ I with b';
8          else
9              update bucket w_i ∈ I with value in w_{i-1};
10     l = 0;
11     for i=0:length(I) do
12         if value(w_i) ≠ value(w_l) then
13             add i − l to D(A, B);
14             l = i;
15 return percentile(D(A, B), c);
```

Algorithm 2: Duration discovery for a temporal rule.

Algorithm. Taking into account the above factors, we propose an approach for time duration discovery in Algorithm 2. A dependency cube C for an approximate functional dependency $A \rightarrow B$ is given as input as well as a cut-point $1 \leq c \leq 100$ for the identification of the duration that removes outliers. We use as default value of 10 for the cut-point, as this is a common value used for trimming of outliers (e.g., interdecile range). We also show in the experimental study how this parameter affects the results.

In a nutshell, the algorithm first corrects the erroneous attribute values reported by the sources for each plate in an integrated stripe I (Lines 3-10), and then adds the durations over I to a duration list (Lines 11-15). The output is the minimum time duration value M that removes outlying durations for the time dependency $A \rightarrow B$.

The algorithm iterates over each plate (entity) in the cube (Line 2). For each plate, we create a new, empty integrated stripe (Line 3). In the time slice for each bucket in the plate, depending on the source quality, sources can agree or disagree on the attribute value. To alleviate the problem of sources with poor quality, we employ a repair step (Line 5). In a simple analysis, if there is a single most frequent value, this is assigned to the integrated stripe (Line 7). If a majority cannot be determined, the values are ignored and the value imputation is done with the previous values in the strip (from the Occam's razor principle) (Line 9).

After the repair process, the algorithm works on the integrated stripe I and extracts time durations between different consecutive attribute values (Lines 11-16). Parameter l in Line 11 records the first point in time when the stripe reports an attribute value. In the following windows, the source may report the same value, or change it. If the value changes, the l parameter is used to compute the time difference between the two different attribute values.

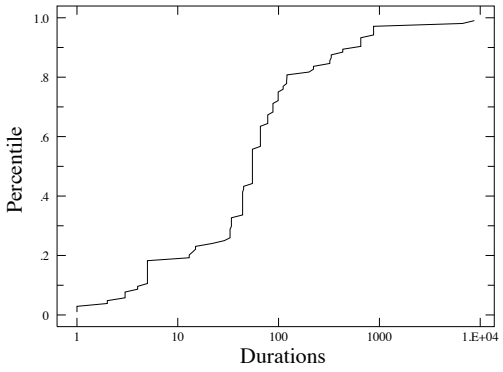


Figure 6: Duration discovery with percentile plot.

With multiple time durations from multiple integrated stripes, we use a trimming (truncation) function, namely the c^{th} percentile, to compute the duration M. The intuition is that trimming identifies outlying values (*trimmed minima*), and we are after the duration that identifies such outliers. For example, in the probability distribution of time durations, the 10^{th} percentile specifies the time duration value at which the probability of the time durations is less than or equal to 0.1. We report an example of a minimum duration of six hours (x axis) discovered with the 10^{th} percentile (y axis) in Figure 6.

Timestamps or Values? It is worth observing that the algorithm above aligns the timestamps and then compares values to perform the analysis. An alternative approach is to align the values, after they have been ordered, and then perform the counting of the durations. We will show in the experimental section that an algorithm that relies on values for alignment performs worse than the one we propose based on time alignment. In particular, we implemented a variant of sequence alignment from the bioinformatics domain [26]. Taking all stripes from a plate, we align attribute values of each pair of stripes. The alignment process creates two temporary stripes that are the aligned versions of the input pair; the temporary stripes both report the same value at a given time, or one of them reports nothing (i.e., reports a *null* value) whereas the other reports an attribute value. The alignment approach mines durations between value changes only when the change is reported by both stripes.

Conditional Durations. As we mentioned earlier, some TFDs may only apply to a subset of entities because some, usually popular, entities have more frequent attribute changes at smaller time frames. To discover the corresponding durations, we track the duration sequences of a specific entity and compute its duration by mining M only with values from their plate. As the minimum duration is computed based on only the sequences that refer to a specific entity, this entity has to be popular, i.e., there must be at least some observations to compute a distribution. As it is common in statistics, we require 30 observations for the computation of the percentile. Therefore, we compute constant rules only for entities with at least 30 durations in their plate. For instance, while 24 hours is the minimum duration that removes outliers for the majority of persons in our person travel dataset, persons such as *Vladimir Putin* or *Ban Ki Moon* should have smaller minimum durations, and this is reflected with their constant rules.

5. EXPERIMENTS

In the following, we first study the performance of our solutions and compare them to baseline alternatives using a real dataset provided by Recorded Future. We then study our algorithms in depth with synthetic data.[2]

We measure the effectiveness of both implication and duration discoveries. We also measure the execution time needed by the algorithms. Experiments were conducted on a Linux machine with 24 1.5GHz Intel CPUs and 48GB of RAM. All algorithms have been implemented in Java with Heap size set to 12GB.

Algorithms. For the implication discovery, we compare our proposal (Section 3) to CORDS [20], a state of the art algorithm for the discovery of approximate dependencies. We test both methods for time slices, therefore they do not have to deal with the time dimension. For the duration discovery, we test the following algorithms:

- REPAIR-OUTLIERS (RO), our method reported in Section 4 where the durations collection is performed over a unified view of every plate. These "clean" durations are then used for mining the minimum duration M that isolates outliers.

- NO REPAIR-OUTLIERS (NR), a variant of our algorithm where we do not perform repair; we collect all durations over the stripes to mine M. This method shows the role of the repair.

- ALIGNMENT-OUTLIERS (AL), a variant of sequence alignment in genomics [26] (Section 4). This is an alternative method that trusts values more than time, as the former are used for alignment.

- NO REPAIR-PROBABILITY (NP), an adaptation of the disagreement decay from the duration discovery algorithm in [27]. Disagreement decay is the probability that an entity changes its value within time Δt. For an increasing Δt, the probability of decay $0 \leq p \leq 1$ also increases. The authors use a probability distribution D for various Δt values [27]. We use a probability cut-point δ_c, such that we select the smallest $\Delta t'$ that satisfies the condition $D(t') \geq \delta_c$ as the duration for our temporal dependency.

[2]The annotated real-world data and the program to generate synthetic data can be downloaded at `https://github.com/Qatar-Computing-Research-Institute/AETAS_Dataset`

Event	# Atts	# Ground Rules	Rules Coverage	Rule Annotated Over Data	# Annotated Tuples	% Errors
Acquisition	3	4	1.00	acquired company → acquirer company	217	26
Company Employees #	2	2	0.50	company → employees number	198	26
Company Meeting	5	6	0.80	company → meeting type	179	17
Company Ticker	3	2	0.67	ticker → company	1,906	4
Credit Rating	4	6	1.00	company → new rank	150	8
Employment Change	7	11	0.86	person → company	186	14
Insider Transaction	22	210	0.95	insider → company	150	0
Natural Disaster	2	1	0.50	location → natural disaster	250	10
Person Travel	6	2	0.67	person → destination	372	21
Political Endorsement	2	1	0.50	endorser → endorsee	199	11
Product Recall	5	12	0.67	product → company	216	5
Voting Result	2	2	1.00	location → winner	215	10

Table 1: Events, correct rules, and annotated rule used in the real data evaluation.

5.1 Real Data

Dataset. We obtained a 3-month snapshot of data extracted by Recorded Future, a leading web data analytics company. The dataset has about 188M JSON documents with a total size of about 3.9 TB. Each JSON document contains *extracted events* defined over entities and their attributes. An entity can be an instance of a person, a location, a company, and so on. Events have also attributes. In total, there are 150M unique event instances excluding meta-events such as co-occurrence.

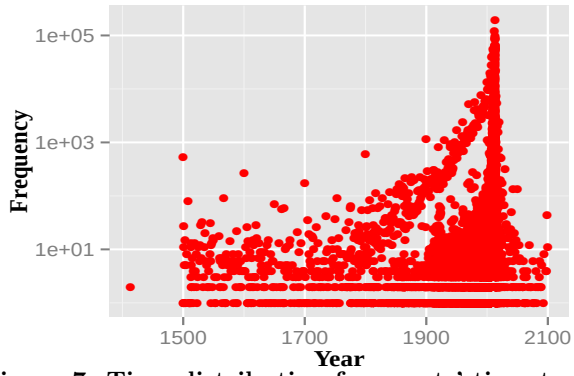


Figure 7: Time distribution for events' timestamp.

The data contains events from circa 15^{th} to 21^{st} century. Figure 7 shows events across years, where each point corresponds to the number of events in a day. Most of the events occur around Fall 2014, because the data is mostly around the 3 months covered by the snapshot. Starting from January 2015 onward, the reported events are future forecasts.

Metrics. We crafted ground rules for all events to evaluate the discovery algorithms. These rules are dependencies that are semantically correct. We report their number for every event in Table 1, where we also report as "Coverage" the ratio of the distinct number of attributes in the set of rules to the total number of attributes for that event. We then use this ground truth to evaluate the precision and recall over the top-k dependencies returned by the algorithms.

In the case of TFDs, crafting also their ground durations is more challenging since there are rarely clear duration values that can be set. We argue that it is not correct to compare the discovered duration value versus an arbitrary manually set ground truth; different persons may come up with different durations. We also show that a combination of these arbitrary values does not lead to the best duration value. Distance between the discovered duration and the real one can be better measured in a controlled environment with synthetic data, as we discuss in Section 5.2. We therefore evaluate the quality of the duration discovery on real data in a different way. Instead of measuring the distance between durations, we evaluate the quality of a duration by measuring its effect in a target application. In particular, for a TFD, we run it multiple times in the same data cleaning tool with different durations, and measure the quality of the obtained repairs. We use BigDansing [22] , a data cleaning system that can handle TFDs with the repair semantics discussed in Section 2.

For this validation, we manually created ground truth for a large sample of the data. We randomly picked 12 event types and discovered rules over their corresponding instance datasets. For each selected rule, we sampled 1% of the tuples, making sure that (i) at least 150 tuples comprising 5 different entities were selected, (ii) both popular and rare entities were selected, and (iii) at most 150 tuples were annotated for one single entity. Each tuple has been manually validated with sources such as Twitter accounts for persons, LinkedIn official web pages for companies, and stock brokers. Table 1 gives the details about the selected events, representative rules, and size of the samples.

Given the ground data and the results of the cleaning, we follow common practices from the data cleaning literature [3, 10] to evaluate the quality of the obtained repair. We count as *correct detections* the updates in the repair that correctly identify dirty values. This corresponds to measuring the effectiveness of repairs based on delete-semantics, where tuples with errors are removed. We count as *correct changes* the updates in the repair that are equal to the original values in the ground truth. Based on these two metrics, we can then measure precision $P = \frac{|changes \cap errors|}{|changes|}$, which corresponds to correct changes in the repair, recall $R = \frac{|changes \cap errors|}{|errors|}$, which corresponds to coverage of the errors, and F-measure $F = \frac{2 \times (P \times R)}{(P+R)}$.

Results. We start with evaluating the discovery of approximate FDs. Table 2 shows the obtained precision and recall values. We rank the approximate FDs based on their scores and compute the precision and recall @k= {1,3,5}, where k is the number of dependencies evaluated after they are ordered with decreasing scores. On average, the precision and recall @k of the NPMI-sorted results is significantly higher than the CORDS-sorted results. The significance is most obvious @$k = 1$. On average, the NPMI scoring approach clearly yields better results than the baseline. This is not surprising, since CORDS was designed to discover approximate FDs on relational databases and it is reported to re-

Event	NMPI						CORDS					
	k=1		k=3		k=5		k=1		k=3		k=5	
	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec	Prec	Rec
Acquisition	1	0.25	0.66	0.5	0.8	1	1	0.25	0.66	0.5	0.6	0.75
Company Employees #	1	1	0.5	1	0.5	1	0	0	0.5	1	0.5	1
Company Meeting	1	0.14	1	0.42	1	0.7	1	0.14	0.66	0.28	0.6	0.42
Company Ticker	1	0.5	0.33	0.5	0.4	1	0	0	0	0	0.2	0.5
Credit Rating	1	0.16	1	0.5	1	0.83	1	0.16	1	0.5	1	0.83
Employment Change	0	0	0.66	0.18	0.6	0.27	0	0	0.33	0.09	0.4	0.18
Insider Transaction	1	0	1.0	0.01	1.0	0.02	0	0	0	0.66	0.01	0.02
Natural Disaster	1	1	0.5	1	0.5	1	1	1	0.5	1	0.5	1
Person Travel	0	0	0.33	0.5	0.4	1	0	0	0.33	0.5	0.4	1
Political Endorsement	1	1	0.5	1	0.5	1	0	0	0.5	1	0.5	1
Product Recall	0	0	0.66	0.17	0.8	0.33	0	0	0.66	0.17	0.8	0.33
Voting Result	1	0.5	1	1	1	1	1	0.5	1	1	1	1
Avg	0.75	0.38	0.68	0.57	0.71	0.76	0.42	0.17	0.57	0.50	0.62	0.67

Table 2: Precision/Recall of approximate FD discovery for sample events over 3 months of data.

quire a sample of size between 1k and 2k pairs [20]. Such amounts of data are not always available when the data is chunked into time buckets.

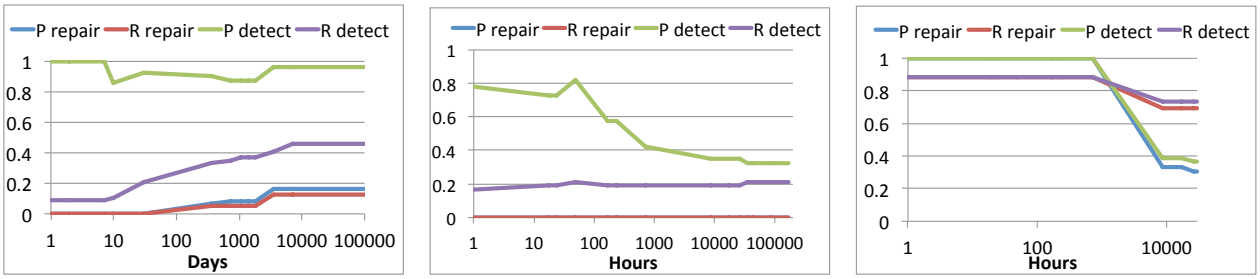
We analyze next how different duration values for the same rule impact the quality of the repairs. In Figure 8(a), the event is Acquisition. Since a company is usually acquired only once, the considered rule is a FD as it is demonstrated by the improvement in the quality of the repair for both precision and recall when the duration exceeds 3600 days (10 years). The explanation is that for smaller values, the rules cannot detect errors. As expected, the results in terms of detection (delete semantics) are much better than the ones that consider the modification of problematic values (update semantics). In particular, there is not enough redundancy in the data to find the correct update for the repair. In Figure 8(b), the event reports the number of employees for a company. As this information changes over time, different durations lead to different quality results in the repair. Intuitively, if the time is small, precision is favored over recall, and the other way around with large values. This is the behavior we observed for all events with temporal characterization. Also in this case, the detection has much better performance than the metric considering also the values of the updates. Finally, Figure 8(c) reports a case where there is a clear point in which precision falls to low values when the duration increases. In this case, the duration is too large and covers several changes of employment for a person, thus several correct values are detected as problematic.

In Table 3, we report the discovered minimum duration M and the cleaning quality results with the REPAIR-OUTLIERS (RO) and NO REPAIR-PROBABILITY (NP) approaches. We compare them against (i) the results obtained using a FD, (ii) the average of the durations suggested by three domain experts, (iii) the best duration value for the rule, selected with the previous study (as in Figure 8). The first three rules do not depend on time since they have only one correct reference value in our dataset. Hence, the FDs perform best for this case. The duration discovery algorithm was not able to find a duration that is large enough to make the TFD perform better. This is because our dataset mainly contains events that happened within three months and the discovery approach subsequently suffers from the limited timespan when identifying these larger durations. Interestingly, values for the remaining events change over time. In these cases, the durations discovered with our RO approach always lead to better a F-measure value than the FDs and the alternative approach NP. Moreover, in several cases we are able to achieve the same precision and recall of the best duration from the previous study. The other TFD approaches NR and AL performed similarly to RO on these datasets with failures in some cases. For example AL discovers a duration of 0 for Voting Result, while NR fails with the events with higher noise rate, such as Company Employees #. We shall elaborate on the differences of the TFD approaches in more detail in the next subsection. Finally, the average of the durations collected from the three domain experts show poor results in terms of F-measure, with the exception of the Person Travel case. This confirms that manually crafting the correct durations for data cleaning is a hard problem to be tackled top-down; a bottom-up approach that mines the data leads to more useful results.

Event	Entity	Conditional		Global	
		M	F	M	F
Company Emp #	Wal-Mart	24	**0.82**	24	**0.82**
Company Emp #	Tesco	27	**1.0**	24	**1.0**
Company Meet.	Val. Pharm. Int.	217	**0.68**	336	**0.68**
Company Meet.	Wal-Mart	45	**0.57**	336	**0.57**
Credit Rating	Tysons Foods	53	**1.0**	48	**1.0**
Credit Rating	NY Method. Hosp.	72	**0.66**	48	0.0
Emp. Change	Sean Moriarty	3168	**1.0**	24	**1.0**
Emp. Change	Rodney Reid	12	**1.0**	24	**1.0**
Natural Disaster	Argentina	45	**0.57**	24	**0.57**
Natural Disaster	England	7	**0.67**	24	0.6
Person Travel	C. Ronaldo	24	**0.71**	48	0.69
Person Travel	Lady Gaga	26	**0.73**	48	0.69
Pol. Endorsement	Ron Paul	96	0.52	48	**0.54**
Pol. Endorsement	Sarah Palin	20.5	**0.75**	48	**0.75**
Product Recall	vehicle	108	**0.76**	177	**0.76**
Product Recall	cars	32	**1.0**	177	0.89
Voting Result	Afghanistan	24	**0.76**	24	**0.76**
Voting Result	United States	6	0.33	24	**0.86**

Table 4: Comparison of F-measure results for conditional and global TFDs.

While the minimum durations from Table 3 can be applied for all the entities, we report in Table 4 minimum duration values for popular entities over all events. In our approach, the specific minimum duration for each entity can be computed before the aggregation of the stripes. For popular entities, these values can lead to better cleaning results. For example, while the discovered minimum duration for Person Travel is 48 hours (Table 3), conditional rules for pop-



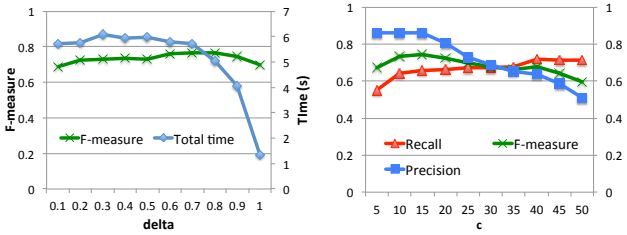
(a) Acquisition (FD) (b) Company Employee # (TFD) (c) Employment Change (TFD)

Figure 8: Cleaning results with different durations.

Event	TFD Semantic (RO)				TFD Semantic (NP)				FD semantic			Humans		Best possible F			
	M	P	R	F	M	P	R	F	P	R	F	M	F	M	P	R	F
Acquisition	48	1.0	0.08	0.16	840	0.92	0.21	0.34	0.96	0.46	**0.62**	-	-	-	0.96	0.46	0.62
Company ticker	24	0.43	0.25	0.31	1	0.69	0.14	0.23	0.96	1.0	**0.98**	-	-	-	0.96	1.0	0.98
Insider transaction	24	1.0	1.0	**1.0**	264	1.0	1.0	**1.0**	1.0	1.0	**1.0**	-	-	-	1.0	1.0	1.0
Company Employees #	24	0.74	0.17	**0.27**	1344	0.37	0.17	0.23	0.24	0.19	0.22	1016	0.23	48	0.73	0.20	0.31
Company Meet.	336	0.94	0.5	**0.65**	5k	0.4	0.54	0.46	0.38	0.53	0.44	4560	0.46	720	0.88	0.53	0.67
Credit Rating	48	0.6	0.75	**0.67**	72	0.56	0.75	0.64	0.18	0.66	0.29	4680	0.55	24	0.69	0.75	0.72
Employment Change	24	1.0	0.88	**0.94**	14k	0.39	0.73	0.51	0.37	0.8	0.51	12k	0.5	≤720	1	0.88	0.94
Natural Disaster	24	0.8	0.5	0.62	29	0.8	0.5	0.62	0.51	0.91	0.65	255	**0.86**	[168:500]	0.93	0.78	0.86
Person Travel	48	0.61	0.82	0.7	72	0.59	0.84	0.69	0.42	0.93	0.58	36	**0.73**	24	0.92	0.85	0.88
Political Endorsement	48	1.0	0.59	**0.74**	216	0.85	0.65	0.73	0.52	0.88	0.65	1200	0.68	[24:70]	1.0	0.59	0.74
Product Recall	177	0.9	0.9	**0.9**	7,033	0.41	0.9	0.56	0.38	0.9	0.53	352	**0.9**	[100:400]	0.9	0.9	0.9
Voting Result	24	1.0	0.6	**0.75**	816	0.79	0.71	**0.75**	0.31	0.9	0.59	4440	0.57	720	0.83	0.75	0.79

Table 3: Precision and recall of the error detection based on duration discovery approaches (M in hours).

ular entities yield higher F-measure than the unconditional TFDs. Interestingly, there is a case where the conditional TFD performs worse that the non-conditional one. Since in the US there can be multiple elections in different states in the same day, the algorithm mines a very low duration of 6 hours. This suggests that Voting Result extractors can be revised to consider American states, instead of one country.



(a) Early termination δ (b) Cut off point c with RO

Figure 9: Study of the input parameters averaged over 9 temporal events: (a) Execution time and F-measure for the top-5 rules, (b) Precision, Recall, and F-measure of the repair for RO.

Parameters. In the above experiments the early termination threshold δ and the cut-off point c for trimming and duration discovery were set to 0.7 and 10%, respectively. We report in Figure 9(a) how different values for δ affect the F-measure of the top-5 dependencies discovered with our method. We observe that aggressive pruning leads to faster execution, but to a loss in the quality of the results. However, an early termination with δ=0.7 reduces the execution from 52 to 6 seconds and preserves the quality.

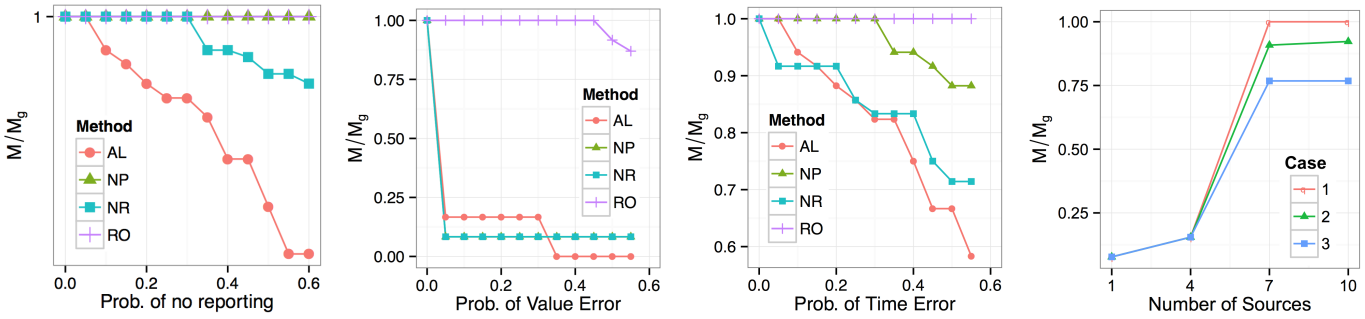
Figure 9(b) shows that the discovery algorithm behaves as expected with respect to the cut-off parameter c: low cut-off points lead to high precision and higher values lead to higher recall. This property allows the user to tune the discovery for their target application requirements. Interestingly, increasing values for c show similar behavior for both RO and NP, and the default value of $c = 10$ is close to the max F-measure value for both methods.

Execution times. AETAS's runtime is dominated by the time needed for reading the data from a database. For the largest dataset, CompanyTicker with more than one million tuples, and without early termination, AETAS took a total time of 53 seconds from which 52 seconds were spent to identify the implications and 1 second to discover the minimum duration for a chosen implication. With early termination, the process takes less than 2 seconds. The dependency cube is also easy to maintain in memory as we handle one cube per FD at a time and its size is bound by the number of tuples. Also, when imputing missing timestamps, we do not materialize their values in the stripe, and we implicitly maintain the time sequence for a non changing value.

5.2 Synthetic Data

The goal of the experiments with synthetic data is to analyze how the discovery algorithms perform wrt different properties of the data.

Dataset. In each scenario, we generate S sources with information over events for O objects for T timestamps. The generation of the values follows a TFD with a given M_g. Each tuple for a source has an attribute for the entity (reference value), an attribute value for the TFD, and a timestamp. For example, for TFD $name \wedge \Delta \rightarrow position$ with Δ=[0, 2] would generate tuples such as *(Jay, worker, 1), (Jay, worker, 2), (Jay, manager, 3), (Jay, manager, 4),*



(a) Reporting error P_r (b) Value error P_e (c) Time error P_t (d) Multiple sources

Figure 10: Ratio of the mined duration M wrt the golden duration M_g with different kinds of errors.

(Jay, manager, 5), (Jay, manager, 7), and (Jay, clerk, 8). For each timestamp and entity, a source has a probability P_r of not reporting the current value in a tuple, a probability P_h of changing the value (with the current duration up to a given maximum value), a probability P_e of reporting a wrong value, and a probability P_t of reporting a wrong timestamp. We run the different discovery algorithms on the union of the data from all sources. All experiments have been carried out for 1000 reference values and the probability of changing the value P_h was set to 0.2.

Metrics. In this controlled environment, we know the properties of our generative model, such as the golden values for M_g. We can thus measure the quality of the mining as the ratio of the discovered duration M to the input duration M_g. A ratio of 1 shows that the method has correctly mined M.

Results. To evaluate the influence of errors and sparseness in the data, we created different scenarios by varying different parameters. We first tested each type of error, namely missing values, wrong values, and wrong timestamps in isolation, i.e., we varied the probability of the error at hand from 0 to 0.6 and fixed the others probabilities to zero. We had 12 experiments for each error type by varying (i) the number of reporting sources from 2 to 5, and (ii) M_g to 7, 12, 17. The maximum duration (the time an event holds) was fixed at $M_g + 5$. From these experiments, we collected 12 M/M_g ratios, and took their median. We then tested the role of sources and skewed error rates for our method. In this experiment, we considered 1, 4, 7, and 10 sources, all of them with P_r=0.1 and P_t=0.2, and three cases. In case 1, the first source has P_e=0.1, and at each step we add three new sources with P_e values 0.1, 0.3, and 0.5. Similarly, in case 2 (resp. 3), the first source has P_e=0.2 (resp. 0.3), and at each step we add three sources with P_e values 0.2, 0.4, and 0.6 (0.3, 0.5, and 0.7 resp.).

Figure 10 shows the overall results. We see that RO performs better than the baselines with all error types. In Figure 10(a), it is easy to see that both RO and NP are robust to missing values, AL performs poorly because it cannot align stripes when values do not match, and the absence of integration leads to several missing values for NR. With errors in the values (Figure 10(b)) RO is the only one able to perform well with high percentages of noise, while the other methods experience a big drop in performance. In Figure 10(c)), we see that RO is robust to errors in the timestamps and computes better durations that the others (NP drops in performance at 0.3). Finally, Figure 10(d) shows that increasing the number of sources leads to improvement in the mining. The combination of missing values, errors in the timestamps, and very erroneous sources make the problem more challenging. In particular, a useful duration is discovered starting with seven sources in all cases.

6. RELATED WORK

Our work is related to two main areas, namely, dependency discovery and temporal data management.

In the context of constraints discovery, FDs attracted the most attention. TANE is a representative for the schema-driven approach to discovery [19], while FASTFD is an instance-driven approach [32]. Recently, DFD has also been proposed with improvements in performance [1]. While all these methods are proven to be valid over clean data, few solutions have been proposed for discovery over noisy data. An extension in this direction is the discovery of approximate FDs that hold for a subset of a relation with respect to a given threshold [23]. A similar extension has been proposed to mine approximate TFDs [11]. The major drawback of approximate FDs on noisy data is that from a certain threshold of noise on, such as 26% in our real world data scenario, the results of the discovery approach will mix up useful approximate FDs with actual non-dependent columns.

Another aspect of discovering constraints is to measure their importance according to a scoring function. CORDS [20], which we use as baseline for approximate FDs discovery, uses statistical correlations for each column pair to score possible FDs. In conditional functional dependencies (CFDs) discovery, other measures have been proposed, including support, which is defined as the percentage of the tuples in the data that match the pattern tableaux (the constants) and χ^2 test [8, 14]. Song et al. introduced the concept of differential dependencies [31] by extending FDs with differential functions, which are dependencies that change over time. They also mine dependencies, but they have focused on identifying dependencies on clean data only.

Integration and cleaning with temporal data [2, 9, 25, 27] is also of interest. The related approaches can benefit from our algorithms. The PRAWN integration system [2] can use our TFDs to detect errors, while the record-linkage systems for temporal data can exploit our repair-based duration discovery for their mining of temporal behavior. In fact, their goal is to identify records that describe the same entity over time and understanding how long a value should hold is critical for their algorithms. In particular, we adapted the disagreement decay discovery algorithm from [27] to our setting and indeed it can be applied for minimum duration dis-

covery. From the experimental study, it is clear that our algorithm does better because of the improved robustness wrt the noise in the data. Notice that, differently from [27], noisy data cannot be clustered with good results. We therefore decided to go directly to the tuple-pair comparisons in the cleaning step, and this aggressive cleaning is supported by the experimental results with low execution times and good results in terms of quality. Another application for our rules is truth discovery [15, 28, 29, 33].

7. CONCLUSION

We presented AETAS, a system for the discovery of approximate temporal functional dependencies. At the core of the system are two modules that exploit machine learning techniques to identify approximate dependencies and their durations from noisy web data. As we have shown in the experimental study, traditional FDs lead to poor results when used on a temporal dataset in a data cleaning system. On the contrary, temporal dependencies can improve the quality of the data; our system is able to discover TFDs with minimal interactions with the users and with better results than alternative methods.

As a future direction, we plan to mine TFDs that identify *large* extreme values over the duration distributions, i.e., outlying durations that are too long for a certain event. For example, in many countries politicians have a maximum number of mandates for a certain position. We also plan to extend our duration discovery algorithm with more sophisticated methods for temporal outlier detection [18].

8. ACKNOWLEDGMENTS

This research was supported by Qatar Computing Research Institute (QCRI). The research in this paper used data kindly provided by Recorded Future.

9. REFERENCES

[1] Z. Abedjan, P. Schulze, and F. Naumann. DFD: efficient functional dependency discovery. In *CIKM*, pages 949–958, 2014.

[2] B. Alexe, M. Roth, and W.-C. Tan. Preference-aware integration of temporal data. *PVLDB*, 8(4):365–376, 2014.

[3] G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB*, 3(1):197–207, 2010.

[4] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, pages 143–154, 2005.

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[6] G. Bouma. Normalized (pointwise) mutual information in collocation extraction. In *GSCL*, pages 31–40, 2009.

[7] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti. Extraction and integration of partially overlapping web sources. *PVLDB*, 6(10):805–816, 2013.

[8] F. Chiang and R. J. Miller. Discovering data quality rules. *PVLDB*, 1(1):1166–1177, 2008.

[9] Y.-H. Chiang, A. Doan, and J. F. Naughton. Modeling entity evolution for temporal record matching. In *SIGMOD*, pages 1175–1186, 2014.

[10] X. Chu, I. F. Ilyas, and P. Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469, 2013.

[11] C. Combi, P. Parise, P. Sala, and G. Pozzi. Mining approximate temporal functional dependencies based on pure temporal grouping. In *ICDMW*, pages 258–265, 2013.

[12] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. NADEEF: A Commodity Data Cleaning System. In *SIGMOD*, pages 541–552, 2013.

[13] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *PVLDB*, 7(10):881–892, 2014.

[14] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *IEEE TKDE*, 23(5):683–698, 2011.

[15] A. Galland, S. Abiteboul, A. Marian, and P. Senellart. Corroborating information from disagreeing views. In *WSDM*, pages 131–140, 2010.

[16] F. Geerts, G. Mecca, P. Papotti, and D. Santoro. The LLUNATIC data-cleaning framework. *PVLDB*, 6(9):625–636, 2013.

[17] A. Gelman and J. Hill. *Data analysis using regression and multilevel/hierarchical models*. Cambridge U. Press, 2006.

[18] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. *Outlier Detection for Temporal Data*. Synthesis Lectures on Data Mining and Knowledge Discovery. Morgan & Claypool Publishers, 2014.

[19] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[20] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658, 2004.

[21] C. S. Jensen, R. T. Snodgrass, and M. D. Soo. Extending existing dependency theory to temporal databases. *IEEE Trans. Knowl. Data Eng.*, 8(4):563–582, 1996.

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

[23] J. Kivinen and H. Mannila. Approximate inference of functional dependencies from relations. In *ICDT*, pages 129–149, 1995.

[24] S. Kolahi and L. V. S. Lakshmanan. On approximating optimum repairs for functional dependency violations. In *ICDT*, pages 53–62, 2009.

[25] F. Li, M. Lee, W. Hsu, and W. Tan. Linking temporal records for profiling entities. In *SIGMOD*, pages 593–605, 2015.

[26] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics*, 11(5):473–483, 2010.

[27] P. Li, X. L. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011.

[28] X. Li, X. L. Dong, K. Lyons, W. Meng, and D. Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB*, 6(2):97–108, 2012.

[29] J. Pasternack and D. Roth. Making better informed trust decisions with generalized fact-finding. In *IJCAI*, pages 2324–2329, 2011.

[30] S. Truvé. A white paper on temporal analytics. www.recordedfuture.com/assets/RF-White-Paper.pdf.

[31] S. Song, L. Chen, and H. Cheng. Efficient determination of distance thresholds for differential dependencies. *IEEE Trans. Knowl. Data Eng.*, 26(9):2179–2192, 2014.

[32] C. M. Wyss, C. Giannella, and E. L. Robertson. FastFDs: A heuristic-driven, depth-first algorithm for mining functional dependencies from relation instances. In *DaWaK*, pages 101–110, 2001.

[33] B. Zhao, B. I. Rubinstein, J. Gemmell, and J. Han. A bayesian approach to discovering truth from conflicting sources for data integration. *PVLDB*, 5(6):550–561, 2012.