Mining Change Rules

Daniel Lindner
daniel.lindner@student.hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

Franziska Schumann
franziska.schumann@student.hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

Nicolas Alder
nicolas.alder@student.hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

Tobias Bleifuß
tobias.bleifuss@hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

Leon Bornemann
leon.bornemann@hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

Felix Naumann
felix.naumann@hpi.de
Hasso Plattner Institute
University of Potsdam
Germany

ABSTRACT

Changes in data happen frequently, and discovering how the changes interrelate can reveal information about the data and the transactions on them. In this paper, we define *change rules* as recurring patterns in database changes. Change rules embody valuable metadata and reveal semantic as well as functional relationships between versions of data. We can use change rules to discover formerly unknown relationships, anticipate data changes and explore anomalies if changes do not occur as expected.

We propose the CR-Miner algorithm, which dispenses the manual formulation of rules to uncover this hidden knowledge in a generic and domain-independent way. Given a dataset together with its past versions, we efficiently discover change rules and rank them according to their potential for a manual review. The experimental results confirm that our method finds change rules efficiently in big data: On a subset of Wikipedia infoboxes encompassing data from four years and different categories, we discover 4 456 change rules. Rules between changes from 48 706 tables of open-government data observed over the period of one year can be discovered within 33 minutes, and rules between about 2.5 million Wikipedia infoboxes from 153 templates within 77 minutes.

Figure 1: Example of a change rule between the Wikipedia infoboxes of Steven Whittaker and Efe Ambrose[1], two professional football players.

1 INTRODUCTION

Discovering salient patterns in data is a large field of interest in the area of knowledge discovery in databases [43]. Rather than investigating patterns only in a static view on a database, we raise the question of how *data changes* are interrelated. A prominent example of ever-changing data is the publicly maintained online encyclopedia Wikipedia – in particular, its infoboxes. Wikipedia *infoboxes* sum up information in a concise and structured way, unifying some properties that entities of a category share [39] – their schema. Albeit not always visibly connected by a direct link, many of its data changes are related.

For example, infoboxes in Wikipedia articles of professional football players usually feature properties that state the number of games they played for each team of their senior career. Figure 1 shows that Steven Whittaker and Efe Ambrose both played for the Hibernian F.C. concurrently, i.e., they often played in the same games. We see that the counters for senior club appearances on both player's Wikipedia pages were updated repeatedly around the same time, with Steven Whittaker's page always being the first.

Such a relationship is what we call a *change rule*. Since these infobox updates do not happen automatically but are carried out by volunteer editors, the changes do not happen simultaneously, but varying time spans may lie between one entry and the next. Our algorithm *CR-Miner* can find change rules where the related changes happen long apart from each other, as well as patterns that show smaller time intervals and can thus reveal patterns like controversies on Wikipedia [9], so-called edit-wars, that have been fought, e.g., over the music genre of a band. In some use-cases, instead of the most common items, items that appear rarely are of genuine interest. For example, changes that happen often and regularly are not worth investigating: when an event occurs every day, it is easy to predict, but one does not gain any insights from this prediction. Thus, we want to identify *rare change rules*, which differ from common patterns.

[1]https://en.wikipedia.org/wiki/Steven_Whittaker and https://en.wikipedia.org/wiki/Efe_Ambrose

The task of finding interesting relations and rules describing how data is connected is often tackled by employing association rule mining approaches [13]. Association rules were the first rules that could efficiently be discovered, and various algorithms for finding them in transactional databases have been proposed [29]. However, most of these algorithms are based on finding associated items that appear within the same transaction, i.e., at the same time. Temporal association rule mining is an extension of association rule mining, which incorporates the time dimension of patterns [10]. Both approaches are suited for static item sets, whereas our method helps investigate *changes* in data and their connections.

Change exploration is based on the insight that entries in databases are rarely static, and it aims to unlock valuable information from understanding data changes as a new dimension of data analysis [7]. Bleifuß et al. define one subproblem of change exploration as '*finding other values that experience similar change (in value) or correlated change (in time)*'. As part of the Janus project[3], this paper presents an algorithm to tackle this problem of finding change rules. By examining how changes in data are connected, one can find (hidden) rules, redundancies, underlying causes of change, and also predict future changes. For instance, in the example above with the two football players, a system could suggest to Wikipedia editors to update Efe Ambrose's counter whenever they update Steven Whittaker's counter. In this case, however, a manual check would still be necessary, because the changes have a common underlying cause (a game played by both players), but are not strictly dependent. At the same time, the change rule hints at information that is probably not even explicitly available on Wikipedia: the lineup of individual games.

An example of how change rules can reveal hidden relationships between changes is the following: When applied to a production database of a factory, CR-Miner can discover rules that resemble the production process, similar to process mining. Process mining techniques are used to model, analyze and optimize processes from event-based data [37]. In addition to conventional process discovery algorithms, which require a very specific input format to build a process model, CR-Miner can be applied to event logs and databases connected to the processes in question (e.g., an inventory), to discover hidden dependencies between process steps and various processes. Such detected change rules can also be an entry point into further investigation and process monitoring.

Part of the challenge of discovering interesting change rules is that databases can change in multiple ways: they can be updated, deleted, or new entries can be added over time. Moreover, all these types of changes can occur on different levels in a database: A whole table can be deleted, a column or row might be added, a single value in a field can change. And a field is also always part of a row, a column, and a table. Therefore, change rules should be regarded at multiple levels of granularity to discover the underlying rules. The problem is deepened in large databases, such as public databases, that do not have one underlying source of truth. Many different people might curate them, and some change rules might not always hold over time, or their change interval might vary. Together, this results in a large search space, which is why the automatic discovery of change rules also poses a computational problem. Thus, algorithms that efficiently navigate this vast search space are needed. The CR-Miner algorithm we present in this paper does not traverse all possible combinations of change rules, but prunes the search space early on, aided by lower and upper thresholds for support and confidence. In summary,

- we define the new notion of change rules,
- we propose the CR-Miner algorithm to discover such change rules efficiently,
- we illustrate how the discovered change rules can be reduced to the most interesting ones, and
- we show on two datasets from different domains how our approach scales and yields interesting rules.

The remainder of the paper is organized as follows: Section 2 discusses related work, Section 3 introduces our notation and change rule definitions, Section 4 continues with the description of our algorithm and how to obtain and rank change rules, Section 5 shows our experiments in detail, and Section 6 sums up our insights and concludes our work. All code and results are publicly available[4].

2 RELATED WORK

The problem of change rule mining is connected to (rare) itemset mining and association rule mining. They share the same essential problem and associated challenges, particularly regarding efficiency. Incorporating time-chronological order in rules is strongly related to temporal pattern mining, which aims to discover frequent subsequences by examining time intervals. Our work expands the field of change exploration, which investigates change as a dimension of data that needs to be understood. In the following, we discuss its related research areas in more detail.

2.1 Pattern and Rule Mining

Frequent itemset mining and association rule mining have been a field of interest for at least 25 years [13, 14]. Several algorithms, such as Apriori [2], Eclat [41], or Frequent Pattern-Growth [17], aim at finding sets of items that co-occur in many transactions of a transaction database. Association rule mining extends these approaches by finding rules within frequent itemsets, i.e., there is a direction of the relationship between the items [1]. Measures like confidence or lift [6] are assigned to these rules to compare their reliability and interestingness. There have been further improvements regarding the performance of the mining algorithms, multiple implementations run on distributed systems, and state-of-the-art frameworks incorporate these implementations [14, 23]. However, they share the limitation that they do not take the temporal dimension into account: Frequent itemsets and association rules are found only within transactions that are unordered sets, whereas we want to discover constraints that hold across a period of time points.

Temporal Pattern Mining addresses these limitations by discovering frequent sequences in transactions that contain ordered sets of events [33, 40, 42]. Initially, the problem of representing and reasoning about temporal knowledge was stated by Allen in 1983 [3]; his proposed 13 temporal relationships remain ubiquitous in the field until today. Since then, many works have focused on improved representations for obtaining these relationships [10, 18, 31] or more efficient mining algorithms [25, 28, 30, 32, 33, 42]. However, both association rule mining as well as temporal pattern mining approaches share the assumption that there are individual transactions containing sets (or sequences) of events (or items). It is further assumed that

[3] https://IANVS.org

[4] https://github.com/HPI-Information-Systems/mining-change-rules

there are many such transactions that share some of these items, so that patterns are defined as frequent if they appear in many transactions [33].

The setting for mining change rules is fundamentally different. As detailed in Section 3, we study the case of having data available in a single, time-ordered sequence of change events that cannot be meaningfully partitioned into individual transactions. This is the same scenario studied by frequent episode mining, in which a single event sequence is mined for frequently occurring parallel episodes (event sets) or serial episodes (event sequences) [4, 27]. There are however, several issues due to which episode mining approaches are not directly applicable to our problem: First, episode mining approaches are suited to discover patterns of arbitrary size but generally assume a moderate amount of input data [5, 27]. However, our dataset sizes are far larger than usual approaches for episode mining can handle. Our largest dataset contains more than 10 million event occurrences of close to 6 million different event types spread over more than 35 000 timestamps (see Table 2 in Section 5.1). Such a size dwarfs the usual datasets in episode mining, which contain only slightly more than 100 000 event occurrences of less than 8 000 event types [27]. We are aware of just one work that tackles similarly sized datasets, namely [5], although it should be mentioned that their datasets contain much fewer event types than ours. However, their approach uses an event type hierarchy that allows them to aggregate occurrences of different events to occurrences of a common category, leading to the inference of more coarse-grained events with high support counts. Second, their approach is distributed, whereas our approach uses a parallel setting on a single machine. And lastly, the authors use high support thresholds for their large datasets to prune the search space. This is not possible for our use-case, because we are not necessarily interested in patterns that occur very frequently, as those tend to be not interesting in a change exploration scenario.

Instead, we make the large dataset sizes manageable by restricting ourselves to rules of size two, introducing an additional time-constraint (minimum time between antecedent and consequent), and using an incremental generation of rule candidates. We show that the classical episode mining approach WINEPI [27] fails to efficiently process the datasets of our size in Section 5.3.2.

2.2 Rare Pattern Mining

Rare itemsets are an interesting type of pattern if one seeks events that happen infrequently compared to the overall event occurrences, i.e., events that stand out from the usual activities. In general, rare itemsets are harder to find than frequent itemsets: On the one hand, setting the minimum support to a relatively high value, as one usually does in frequent itemset mining, will exclude items that appear rarely. Setting the minimum support very low, on the other hand, creates a vast amount of candidate set combinations. This two-sided challenge has been coined as the *rare item problem* [26].

Most standard pattern or rule mining algorithms are highly inefficient when setting the minimum support threshold low due to the significantly increased search space. Furthermore, most of these algorithms are based on the assumption that all items have the same kind of properties when it comes to their occurrences, i.e., they assume similar overall frequencies throughout the data [26]. This assumption obviously does not hold true in every use case. For example, in some datasets we primarily see the occurrence of regular events that are not particularly compelling, but the events that appear rarely are the ones that are actually of interest. If two events always occur together, but they rarely occur in comparison to the other events in the dataset, this association will not be found by traditional pattern or rule mining algorithms.

In Section 5, we examine an open-government database that contains administrative data where many tables are updated daily, which does not mean that they are all semantically connected. An ordinary association rule mining algorithm would find a huge number of rules containing mostly the frequently updated tables and fail to discover rarer rules that are more interesting in this context. Our algorithm can find more salient rules, and we report on our results for the mentioned dataset in Section 5.4.2.

Specific algorithms have been proposed for finding rare itemsets and rare association rules [11, 15, 16, 26, 36, 38]. One technique to tackle the problem of rare association rule mining is to set upper boundaries as constraints on the frequency of the itemsets [20, 21, 34, 35], a strategy we employ in our algorithm as well.

A remaining challenge of rare association rule mining is to tell apart interesting rules from random noise [19], especially if no prior knowledge about the data is available to assess the results. In Section 4 we explain how we address this problem in an optional step of our algorithm by employing probability distributions.

2.3 Change Exploration

Data in databases are rarely static but can undergo many changes. The field of change exploration examines this change dimension of data [7]. Questions like '*When, how often, by whom, and, from what to what has the data been changed?*'are investigated because they can have interesting answers that hold information about more than just one entry in a database. For example, knowing data history can help to validate facts and to anticipate future changes [12]. *Change*, however, is in itself hard to describe and Bleifuß et al. tackle this challenge by proposing the *change cube*, a model to capture data changes [7]. Within the context of the Janus project[5], which aims to advance the research in the area of change exploration, we add to the above-mentioned inquires, by exploring the question: '*Which rules exist between data changes?*' In this paper, we propose a way of finding these rules in a relational database.

3 MODELLING CHANGE RULES

We describe what we regard as *changes* and introduce a notation for *change patterns* that facilitates understanding the nature of change rules and their analysis. Furthermore, we propose a definition of *change rules* in a time-window-based fashion that is based on the concept of association rules and can adapt standard measures like support and confidence accordingly.

3.1 Change Patterns

The *change-cube* is a generic model that captures change in four dimensions [7]. Each element in the change-cube is a quadruple of the form ⟨time, id, property, value⟩ or in brief ⟨t, id, p, v⟩, called a *change*. Such a change states that at a point in time t (*When?*) the property p (*Where?*) of an entity with the stable identifier id (*What?*) was changed to the new value v (*How?*). In the following, we assume these points in time t to appear within an overall

[5]Janus project page at www.IANVS.org

observed time-window $[1 \dots n]$, for which we denote timestamps as $t_k,\ k \in 1 \dots n$.

A *change pattern* C_Φ captures a set of changes by restricting one or multiple dimensions of the change-cube through a set of equality constraints Φ. Each equality constraint in Φ is of the form $d = x$, where d is one of the dimensions of the change-cube and x is a value that appears in the respective dimension. For example, $C_{\texttt{p=counter}}$ includes all changes that affect the property `counter`. A change pattern C_Φ *matches* a change c if all constraints in Φ hold for c. A change pattern p *occurs* at a timestamp t if and only if there is at least one change that matches p and $c.t = t$. We call the set of all timestamps at which a change pattern p occurs the *occurrences* of p.

For the purpose of this work, we allow constraints on two additional, derived dimensions: the table that a change belongs to and its change-type. The lowest granularity level on which we observe changes is at field-level (also known as cell-level), so a combination of id and property. For relational input, a *field* is part of a *row* (id) and a *column* (property) that are all in turn part of a *table*. The table dimension table allows restricting change-patterns to one table.

We can distinguish three different types of changes in the type dimension:

Insert Field i does not exist at time t_{k-1} but does exist at time t_k.

Update Field i exists at time t_{k-1} with a value of v and exists at time t_k with a value of w, where $v \neq w$.

Delete Field i exists at time t_{k-1} but not at time t_k.

A field *exists* if the field's entity (row) exists in the database and the field contains a non-NULL value (i.e., is not empty). As input data is not always stored in a database, in our experiments, we consider any of the following strings as NULL-values (including any white spaces): `"-"`, `"/"`, `"–"`, `"—"`, `"%"`, `"Null"`, `"null"`, `"NULL"`, and empty values.

Further, we do not consider constraints on the value or time dimension in this paper. This restriction increases the probability that we can observe matches of a change pattern at multiple points in time. That is desirable for our use case, as contemplating change rules makes sense only if we observe the involved change patterns multiple times. We can imagine other use cases where the data values and their "severity" of change are incorporated, but leave this to future work (Section 6). Thus, a change pattern C_Φ captures the location (i.e., a unique field, row, column, or table identifier) and the change type (insert, update, or delete) of a change.

3.2 Change Rules

We denote a general change rule as an implication between two change patterns C_i and C_j of the following form, where k and z are non-negative integers and $k + z \leq n$:

$$C_i \xRightarrow{[k,\ k+z]} C_j$$

A change rule holds, if whenever change pattern C_i matches a change at t_p, then the first subsequent match of C_j occurs at the earliest after k and at the latest after $k + z$ points in time. Further, we enforce the restriction that C_i does not match again before this match of C_j, because we assume the timewise closest changes to have the strongest relation.

Example. In Wikipedia articles of professional football players, the properties `caps1`, `caps2`, … in the infobox state each player's number of games for their first, second, … senior league team (Figure 1). Steven Whittaker and Efe Ambrose both played for the Hibernian F.C. at the same time, i.e., they often played in the same games. Editors happen to frequently update the counter for senior club appearances in the infobox of the former Scottish national player Steven Whittaker. Usually, within the next 17 hours Efe Ambrose's infobox is also updated. This rule can be expressed as:

$$C_{\texttt{id=SW, p=caps4, type=update}} \xRightarrow{[0,\ 17]} C_{\texttt{id=EA, p=caps6, type=update}}$$

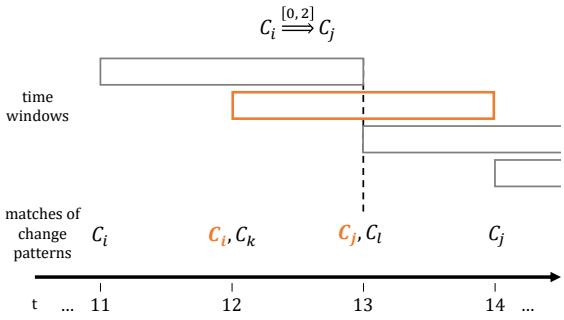


Figure 2: Example of a valid change rule occurrence and the corresponding time window.

As usual, we call the left-hand side of the implication *antecedent* and the right-hand side *consequent*. Our definition also covers the special cases of antecedent and consequent changing at the same point in time ($k = z = 0$), or with an interval of exactly u timestamps between them ($k = u,\ z = 0$).

We now define the *occurrence* of a change rule over time windows with the goal of counting numbers of occurrences. A change rule $C_i \xRightarrow{[k,\ k+z]} C_j$ has a *valid* occurrence in time window $w = [p, p + k + z]$, if

- C_i matches at point in time p,
- C_j matches at q with $p + k \leq q \leq p + k + z$ but not in $[p, q)$ and
- C_i does not match in $(p, q]$ either.

All other occurrences are *invalid*. For example, in Figure 2 there is only one occurrence of $C_i \xRightarrow{[0,\ 2]} C_j$ that is valid and, hence, counted. The pertinent points in time are highlighted in the figure. Only for the matches of C_i at t_{12} and C_j at t_{13} all mentioned criteria are fulfilled. The other occurrences are invalid, because the antecedent either matches again before the consequent is observed (C_i at t_{11} then again, C_i at t_{12} before C_j at t_{13}), or the consequent matches earlier in time after the antecedent (C_j at t_{13} after C_i at t_{12} and before C_j at t_{14}).

3.3 Support and Confidence

In accordance with the definition of *support* for itemsets, we define the support $\sup(C_i)$ of a change pattern C_i as the number of points in time for which this pattern matches at least one change. For given k and z, let W be the set of all time windows in which a change rule could occur, i.e., $W[k + z] \subseteq \{[1, k + z + 1], [2, k+z+2], \ldots, [n-k-1, n], [n-k, n]\}$. The support of a change rule $C_i \xRightarrow{[k,\ k+z]} C_j$ is defined by the number of time windows in which it has a valid occurrence. For the relative support, this is divided by the number of overall time windows $|W| = n - k$. For example, to determine the support of the change rule

$$C_{\texttt{id=SW, p=caps4, type=update}} \xRightarrow{[0,\ 17]} C_{\texttt{id=EA, p=caps6, type=update}}$$

we count every valid time window where Efe Ambrose's counter is updated within 17 hours after Whittakers's counter has been updated and there are no further updates to both counters in the meantime.

The *confidence* of a change rule is defined as

$$\text{conf}(C_i \overset{[k,\,k+z]}{\Longrightarrow} C_j) = \frac{\text{sup}(C_i \overset{[k,\,k+z]}{\Longrightarrow} C_j)}{\text{sup}(C_i)}$$

It quantifies how often we see a valid occurrence of a change rule compared to the number of matches of the antecedent change pattern.

In our real-life example, this amounts to: How often do we see a valid occurrence of the change rule stating that Ambrose's senior appearance counter is being updated within 17 hours of an update to Steven Whittaker's counter (31 times) compared to the overall number of updates of Whittaker's counter (32 times)? Both support and confidence are essential measures that are used in our algorithm (Section 4).

3.4 Search Space Complexity

For fixed k and z, the search space for change rules is defined by the number of combinations of change patterns, as well as the possible time windows. Let m be the number of all change patterns; then the amount of possible change pattern combinations overall is $m \cdot (m-1)$.

The total number of possible antecedent changes to examine for any point in time t_k and the consequent changes occurring on any of the following z points in time is: $\sum_{l=0}^{z} |C_k| \cdot (|C_{k+z}| - b_z)$, where $b_z = |C_k \cap C_{k+z}|$, i.e., b_z is the number of changes c_i that occur at both points in time. There are n time points at which any change rule can appear in our overall time frame (considering that consequent changes happening at the same point in time are also covered by our definition above). As we need to calculate the support of the found candidate change rules in order to validate whether they fulfill the specified thresholds, we observe a space of $\sum_{k=1}^{n} \sum_{l=0}^{z} |C_k| \cdot (|C_{k+l}| - b_z)$. Thus, in the worst case, we have to consider $n \cdot (z+1) \cdot m^2$ candidate change rules. Because $z \leq n$, the search space for the algorithm lies in $O(n^2 \cdot m^2)$.

We have discussed the search space a naïve algorithm would have to traverse to find all change rules. Of course, some examined elements never change (at least not in the finite observed time frame). Thus, it makes more sense to regard only the changes we actually encounter in the transactions instead of all the possible rules. Moreover, one cannot make discover any reliable rules for changes that appear only once or twice in the entire dataset. Thus, some change rules can be excluded early in the search process. Section 4 explores in detail how our algorithm finds change rules efficiently and ranks them according to their interestingness.

4 THE CR-MINER ALGORITHM

Our approach consists of two steps: First, we find change rules that satisfy a minimum and maximum support and a minimum confidence – described in Section 4.2. For each change rule, we create a histogram that states how often the consequent occurred for each specific number of points in time after the antecedent. These histograms are necessary to calculate an interestingness measure that we use to rank the potentially large set of discovered change rules. To be more precise, in a second step, we use probability density functions based on these histograms to compare the change rules and rank them by their Jensen-Shannon divergence to the calculated average distribution. The details of this second step are described in Section 4.3.

4.1 Input Format

For the change rule mining step, the points in time when a change pattern C_i occurred are assigned to each change pattern. Our algorithm expects an index `inputIndex` from the change patterns to their corresponding sorted points in time and the size of the time window z. The index needs to be created once per dataset. Since it is easy and inexpensive to create, we omit a detailed description. Further input parameters are minimum confidence `min_conf` and minimum and maximum support `min_sup` and `max_sup`, which limit the output to change rules of the desired quality.

4.2 Change Rule Mining

The aim of the mining step is to efficiently find valid occurrences of change rules in order to obtain rules that fulfill confidence and support thresholds and to provide histograms of the occurrences. In Section 3.2, we introduced the concept of change rules defined over time windows. One interpretation of the problem is the identification of time windows with valid change rule occurrences. We must ensure that our algorithm combines the latest possible occurrence of an antecedent with the earliest possible occurrence of a consequent, as we have seen in Figure 2.

The main idea of CR-Miner is the following: For each point in time, it combines the occurrences of consequents with antecedents that occurred within the last z points in time, including the current point in time. We call these antecedents *active antecedents*.

First, our algorithm filters the input index of change patterns such that their number of occurrences is within the specified support range. Due to the Apriori property, change patterns with support lower than the minimum support threshold cannot be part of a change rule that reaches the threshold, and our algorithm can discard them. On the other hand, a change pattern C_i with a high support can be part of a change rule $C_i \overset{[k,\,k+z]}{\Longrightarrow} C_j$, where C_j has a low support. If C_i is the antecedent, the confidence is not high, and would not meet the minimum confidence. For the inverse change rule $C_j \overset{[k,\,k+z]}{\Longrightarrow} C_i$, with C_i as the consequent, it would be questionable whether an occurrence of C_j really implies the occurrence of C_i. However, our algorithm could not discard the rule based on support and confidence. Thus, our algorithm also removes change patterns with support higher than the maximum support threshold.

After this initial pruning of change patterns, our algorithm builds an inverse index `time2pattern` of consequent change patterns with the input index and the remaining set of consequents. This inverse index points from a point in time to the consequents that occur at that point in time. Table 1 gives an overview over the central data structures in our approach, including this one.

For each point in time, Algorithm 1 is executed once to create all possible candidates at this point in time. First, in line 1, we update the set of active antecedents `actives`, such that each antecedent is mapped to the number of points in time that passed since its most recent occurrence. E.g., for Figure 2, only the second occurrence of C_i is part of this map from point in time t_{12} onwards. If the occurrence of an antecedent is older than allowed by the window size, it is removed from the active antecedents.

Table 1: Data structures used in Algorithm 1.

Name	inputIndex	time2pattern	actives	pruned	histograms
Content	$C_i \rightarrow [t_5, t_{11}, t_{12}]$ $C_j \rightarrow [t_4, t_{13}]$	$t_{11} \rightarrow C_i$ $t_{12} \rightarrow C_i, C_k$	$C_i \rightarrow 3$ $C_j \rightarrow 5$	$C_i \rightarrow \{C_k\}$ $C_l \rightarrow \{C_j, C_k\}$	$(C_i, C_j) \rightarrow [5, 2, 0, 3]$ $(C_k, C_j) \rightarrow [6, 1, 2, 6]$
Description	an index from change patterns to points in time at which they occur	an index from points in time to all consequent change patterns that occur there	an index from antecedents to the time span since its last occurrence within the window	a set of pruned combinations	current histograms of change rules

Our algorithm now starts to iterate over all consequent change patterns at that point in time (line 2) and combines them with active antecedents in line 4 if the combination has not already been pruned (line 3). Our algorithm skips the consequent occurrence if there has already been a consequent occurrence since the antecedent occurrence (line 5). In doing so, it ignores the second occurrence of C_j in Figure 2. Our algorithm uses the remaining occurrences of the consequent and the antecedent to find an upper bound on the future occurences of the candidate rule in lines 7–9. Based on this upper bound in combination with the past occurrences of this rule (line 10), the algorithm tests if it is possible that the current change rule fulfills the confidence and support thresholds, as shown in lines 11–13. If this is not the case, our algorithm prunes this combination of antecedent and consequent and does not consider it in the future (line 16). Otherwise, we update the histogram of this change rule (line 14). Finally, our algorithm filters the discovered rules to fulfill minimum support and confidence and saves their histograms.

As long as our algorithm investigates each possible combination of change patterns as antecedents and consequents, these steps are independent of each other. Hence, our algorithm splits the input index into partitions and executes the described approach in parallel for all combinations of partitions to improve the overall performance.

Algorithm 1: Candidate combination for a point in time.

Input: min_conf: minimum confidence,
min_sup: minimum absolute support,
t_i: the current point in time
Data structures: see Table 1
Output: updated histograms for point in time t_i

```
1  update actives
2  foreach c_i ∈ time2pattern[t_i] do
3      candidates := actives \ (pruned[c_i] ∪ {c_i})
4      foreach a_i ∈ candidates do
5          if ∃t_k ∈ inputIndex[c_i] : t_i − t_k ≤ actives[a_i]
6              continue
7          remain_ci := |{t_k | t_k ∈ inputIndex[c_i] ∧ t_k > t_i}|
8          remain_ai :=
             |{t_k | t_k ∈ inputIndex[a_i] ∧ t_k > t_i}|
9          remain_rule := min {remain_ci, remain_ai}
10         histogram := histograms[a_i, c_i]
11         possible_sup := ∑ histogram + remain_rule
12         possible_conf := possible_sup / |inputIndex[a_i]|
13         if possible_sup ≥ min_sup ∧ possible_conf ≥
             min_conf
14             histogram.add(actives[a_i])
15         else
16             pruned[c_i] := pruned[c_i] ∪ {a_i}
```

Still, our algorithm has to combine change patterns C_i as antecedents and consequents that occur within a window of size z for each point in time. Because the algorithm filters change patterns that are not within the support range, the number of change patterns that need to be considered is not m but decreases to a number we refer to as m_f. Only the a average active antecedents are combined with the consequents. In Figure 3, we have seen that the majority of change patterns occur infrequently. As a result, $a \ll m$, if the window size z is not too large. The complexity of our algorithm is thus in $O(n \cdot m_f \cdot a)$. Only if all change patterns occur frequently and no support thresholds are applied, it is in $O(n \cdot m^2)$.

The set of parameters should always be adjusted to the specific dataset and use-case. In general, the minimum and maximum support thresholds should not be set too far apart, as this can lead to the discovery of arbitrary rules, in which a rare event implies a frequent event with a high confidence. To account for this, we included a maximum support threshold and used narrow support intervals in our experiments. Also, we have found that a minimum confidence of 0.9 yields meaningful rules for our use-case (see Section 5.2 for all parameters). Smaller support thresholds and a high minimum confidence shrink the algorithm's search space per point in time, ease pruning, and are thus beneficial for the run-time. The chosen window size and gap k depend entirely on the granularity of data and the posed question. For example, if the window size is kept very small, the algorithm finds only rules including changes that happen shortly after one another. To choose the window size, considering a realistic time span between supposedly interrelated events can be a starting point.

4.3 Finding Change Rules with Unusual Distributions

As is the case for association rule mining, searching for change rules often leads to large result sets even when employing a high minimum support. Not all of these discovered change rules are necessarily of interest. There are some *interestingness measures* commonly used in association rule mining, such as support, confidence, entropy or lift, and many other measures have been proposed for specific use cases [22]. As described in Section 4.2, we use support and confidence for early pruning of the search space. In a real-world scenario, a manual review step would likely follow. However, as the number of rules can grow quadratically with the size of the dataset, it is usually infeasible to review them all manually, even when prior knowledge about the data is available, but especially when there is no point of reference to assess the results. Thus, we present a ranking method that can be employed as an optional step to the result set obtained with

CR-Miner to account for change rules that are possibly the most interesting ones to review.

Determining which change rules are *interesting* is always a matter of context. Similar to the premise of rare association rule mining (section 2.2), we want to find change rules that are different from the dominant patterns in the dataset. I.e., we are interested in change rules whose distributions of occurrence deviates from any domain-specific norm or random co-occurring patterns. Thus, our ranking method is based on the distribution of change rules in the dataset to find those that differ from the majority. This is an optional post-processing step of our algorithm and might not be needed for every dataset. In our experiments, we show how such a ranking helps interpret the discovered change rules in the Socrata dataset (Section 5). We provide a possible approach to determine unusual change rules without incorporating domain knowledge. As we observed one predominant distribution shape within the discovered rules in this specific dataset, we describe a solution to find rules differing from this average rule. For a dataset with multiple common patterns of change rule distributions, another approach can be to cluster rules by their distributions and to apply distance measures within the resulting clusters.

4.3.1 Change Probability Distributions. We use probability distribution functions to describe the *shape* of a change rule, i.e., how often can we observe the consequent after the antecedent for each number of points in time between k and $k+z$. The probability distribution function pdf is always tied to antecedent C_i and consequent C_j of a change rule $C_i \xRightarrow{[k,\,k+z]} C_j$. The function's input is a variable $a \in [k, k+z]$ for which the pdf returns the probability that consequent C_j occurs exactly a points in time after antecedent C_i, given that C_j actually occurs. The probability distribution enables us to compare change patterns and therefore quantify differences between them.

$$\mathrm{pdf}_{C_i,C_j}(a) = \frac{histogram_{C_i,C_j}[a]}{\sum_{s=k}^{k+z} histogram_{C_i,C_j}[s]}, \quad k \le a \le k+z$$

Eventually, each change rule is described by a probability distribution – a prerequisite for the next step.

4.3.2 Distributional Difference Measure. The probability distributions quantify how two change patterns occur together within a defined time frame. They do not classify a change relationship as being interesting, common or merely random. We observe that the distributions of most captured (in particular frequent) change patterns in a database look similar, and that there is one dominant pattern. This dominant pattern might reflect the specific domain or the frequent mutual co-occurrence of change rules. E.g., two Wikipedia infoboxes can change daily, but the changes are not necessarily semantically dependent on each other. If this random chance of mutual occurrences dominates change data, it is reflected in most change distributions. The same might be true for a domain, where a logging service updates tables in static time intervals.

We express this dominant pattern with a probability distribution, too: we average all change rules into a general change distribution. The number of all change rules is denoted as d.

$$\mathrm{pdf}_{general}(a) = \frac{\sum_{C_i,C_j} \mathrm{pdf}_{C_i,C_j}(a)}{d}, \quad k \le a \le k+z$$

As we want to analyze large amounts of data, an efficient calculation of the general probability distribution is required. We do this by approximating the general distribution from a sampled subset. We draw one rule for each antecedent C_i into this subset. The general distribution resembles the average change rule and should therefore incorporate dominating properties.

To find diverging distributions, we employ the Jensen-Shannon divergence, which is a measure used to quantify the difference of two probability distributions [24]. We use it to quantify the difference between the general distribution and the probability distribution of a change rule. A large difference score expresses a change distribution that is very different from the dominating norm and vice versa.

The final result we obtain is a ranking of all rules by their Jensen-Shannon divergence score. This ranking helps determine which rules should be manually reviewed for further use. In our experiments we show that this is indeed the case: We have discovered 20 832 change rules in total which could be reduced to 309 salient ones.

Table 2: General statistics of all datasets. The table shows the number of timestamps (# T), event types (# ET) and event counts (# EC) per dataset.

Dataset	Category	#T	#ET	#EC
Wikipedia	Education	35 062	1 101 267	1 529 207
	Geography	35 062	6 153 201	6 684 065
	Media	35 062	4 087 946	6 651 898
	Military	35 062	479 347	841 905
	Politics	35 062	1 651 499	2 488 533
	Sports	35 062	5 879 497	10 847 686
	Transport	35 062	13 174	17 813
Socrata	–	359	55 122	4 373 371

5 EXPERIMENTS

After introducing our concept of change rules and describing our approaches to mine and rank them, this section explores our approach in practice. We employ our algorithm on two different real-world datasets to demonstrate its overall performance and show exemplary results of the change rules we find for unary change patterns.

5.1 Data and Preprocessing

General statistics about all datasets are reported in Table 2. In the following, we elaborate on how we extracted and prepared the data from both the Wikipedia and Socrata data source.

5.1.1 Wikipedia. Our first dataset comprises the changes of Wikipedia infoboxes, an example of which is shown in Figure 1. Wikipedia infoboxes contain data in the form of key-value pairs and belong to a particular template type, e.g., *infobox person*. Bleifuß et al. [8] matched versions of the same infoboxes in Wikipedia articles and, as a result, identified each infobox with a unique and stable key. We consider changes from January 1, 2015, to December 31, 2018 at an hourly granularity, ignoring initial inserts of infobox values, e.g., upon the creation of a new infobox or the addition of a field.

We transform these infobox versions to a relational structure through the following modelling: infobox template types translate to tables, the keys of the infoboxes serve as row identifiers, and the keys of an infobox value as column identifiers. Due to renaming of infobox template types, restructurings and merges

of infobox template types, or simply due to vandalism, infoboxes can use different templates throughout their lifetime (even without changes to the rest of the infobox). To be able to assign these infoboxes to one table, we always use the last infobox template type in the change history of a given infobox as the table identifier.

The infobox template types are rather fine-grained, so our mapping to the relational model would create 8 263 tables, whereas we do not expect rules between completely unrelated infoboxes. To reduce the search space for the change rule mining and lower the probability of finding random rules, we grouped several tables, i.e., infobox template types, by topics. For this purpose, we ranked the infobox template types by the number of infoboxes and manually assigned a topic to the top 200 of them, if a topic was evident considering the template name. Thus, we derived the seven categories *education*, *geography*, *media*, *military*, *politics*, *sports*, and *transport*. Overall, these categories include over 2.4 million infoboxes of 153 infobox template types. Table 3 shows the number of infobox types per category.

Filtering all changes by these categories, results in a final dataset containing almost 30 million occurrences of nearly 20 million change patterns. As we can see in Figure 3(i), the dataset is sparse – the majority of change patterns occurs infrequently, and the number of change patterns decreases for higher frequencies. 84 % of change patterns occur once, and only 66 599 change patterns have at least 20 occurrences in four years.

5.1.2 Socrata. For the Socrata dataset, we obtained table histories by downloading daily snapshots of the published tables over the course of one year, from November 1, 2019, to November 2, 2020. For each table, we then generated stable identifiers for columns and rows using a matching approach by Bleifuß et al. [8] and are thus able to extract changes on field-level granularity.

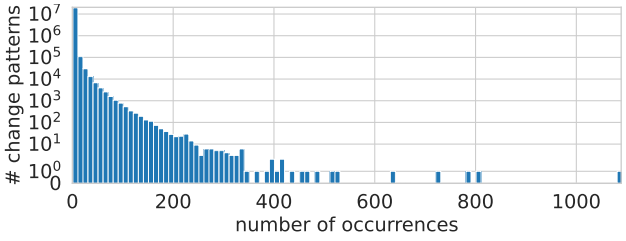
The final dataset that we use for our evaluation consists of 48 706 tables containing on average 18 columns and 416 rows. Because the API was unavailable on some days, the dataset contains snapshots for 359 timestamps; we simply ignore missing days. Whenever any value of a field has been changed, the dataset contains a new version of the respective table.

First of all, we extracted changed fields and the change types by comparing the subsequent versions of the tables. We then aggregated the set of changes on a column granularity and obtained 22 456 464 occurrences of 583 145 change patterns – Figure 3 shows their distribution. As we can see, the vast majority of change patterns occur fewer than ten times. Except for these infrequent change patterns, the number of occurrences per change pattern is within one order of magnitude for all days in the dataset. Thus, the data is denser than the Wikipedia data.

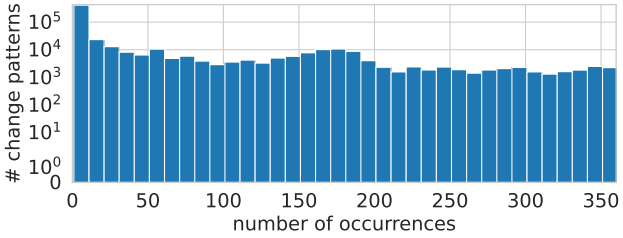
For the Socrata dataset, we performed an additional step of preprocessing based on two observations. On the one hand, insertions or deletions of rows always affect all columns. On the other hand, there are changes within multiple columns of a table on the same day. As a result, there are multiple change patterns with the same occurrences. By grouping these change patterns, we reduce our input data from 583 145 to 52 565 change patterns. This is a reduction by 91 %, without an information loss.

5.2 Experimental Setup

To ensure reproducibility, we state the parameters and hardware we chose to retain the presented results.



(i) Wikipedia



(ii) Socrata, aggregated on column-level granularity

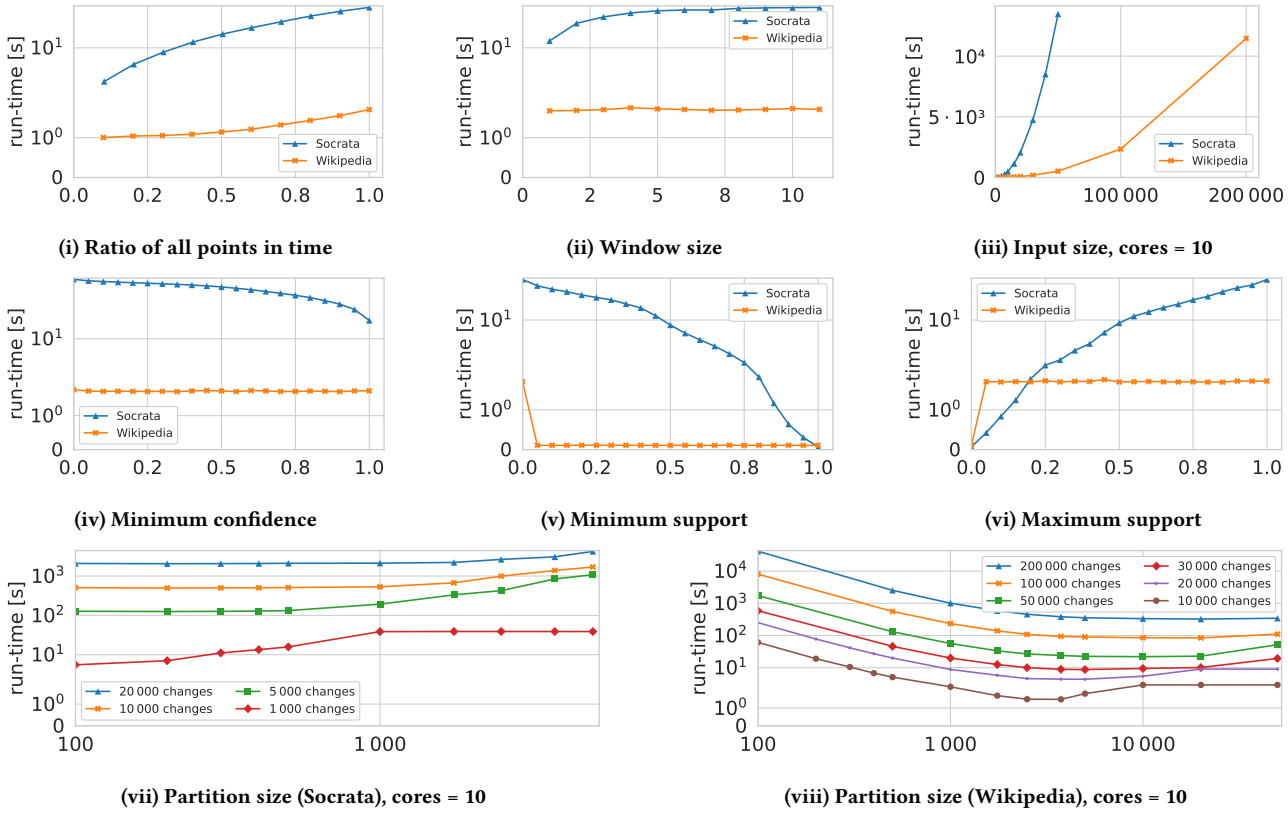
Figure 3: Distribution of change pattern occurrences in the two datasets (bin width = 10 occurrences).

5.2.1 Algorithm Performance Experiments. The experiments were performed on a server equipped with an Intel Xeon E5-2630 processor with 10 cores/20 threads and 64 GB RAM. We implemented the CR-Miner algorithm in Python and executed it with a Python 3.7.6 interpreter.

The algorithm's relevant performance dimensions are confidence and support thresholds, the number of points in time, window size, input size, and partition size. The last dimension affects only performance, whereas the others influence the results, too. We used predefined subsets of change patterns with their occurrences and fixed the parameters that were not part of the current experiment. Unless stated otherwise, we used as defaults an input sample with 1 000 change patterns that are randomly sampled from the respecting dataset and consider changes at all points in time of the input dataset, a window size of 11, minimum support of 0, maximum support of 1, minimum confidence of 0.9, and partition size of 200 in a single-core setup. For the experiments on input size and partition size, we used ten cores. Reported times are the average run-time of three executions.

5.2.2 Obtaining Change Rules. For the Wikipedia dataset, we perform the rule mining only for changes within the same category. To avoid random rules, we only consider change patterns that occur at least 20 times and at most at 45 % of the days. Moreover, we group the change patterns according to their occurrence frequency. We define frequency intervals with a size of $\#days \cdot 0.05$ and apply our algorithm to change patterns with frequencies within the same interval. I.e., we use the interval boundaries divided by the 35 064 hours of the four years from 2015 to 2018 as relative support thresholds. The window size is 24 hours.

For the Socrata dataset, we employ a minimum support of 0.05 and a maximum support of 0.45, given a window size of 11 days. Furthermore, the constraint that antecedent and consequent must stem from the same Socrata domain, which mostly matches geographic regions, was included. To check how many change rules with unusual distributions are found, we rank the obtained



(i) Ratio of all points in time **(ii) Window size** **(iii) Input size, cores = 10**

(iv) Minimum confidence **(v) Minimum support** **(vi) Maximum support**

(vii) Partition size (Socrata), cores = 10 **(viii) Partition size (Wikipedia), cores = 10**

Figure 4: Run-times w.r.t. dimensions. If not stated otherwise, input size = 1 000 change patterns, minimum support = 0, maximum support = 1, minimum confidence = 0.9, partition size = 200, cores = 1.

change rules by their deviation from the sampled general probability distribution using the Jensen-Shannon divergence score. We then determine a cut-off by manual revision. For both datasets, the minimum confidence enforced is 0.9 and the gap $k = 0$ points in time.

5.3 Run-time Performance Experiments

The run-time experiments are divided into two parts. We first investigate the impact of different parameter settings and input data sizes on the performance of CR-Miner. Afterwards we compare CR-Miner to WINEPI [27], an algorithm for mining serial episodes and show that WINEPI is unable to handle large datasets.

5.3.1 CR-Miner Performance in Different Settings. We visualize CR-Miner's run-time regarding the different dimensions in Figure 4. The first experiment examines the influence of the number of points in time. As we can see in Figure 4(i), there is an approximately linear dependency between the number of points in time and the run-time for both datasets. In Section 4, we explained that our algorithm iterates over the points in time once for each combination of input partitions. Thus, we expect this relationship.

The size of the sliding window also affects performance. Figure 4(ii) illustrates that the behavior of CR-Miner differs when applied to different datasets. For the Socrata dataset, wider windows lead to increasing run-times, as expected. The window size influences the number of *active antecedents*, which our algorithm combines with the occurrences of consequents. This set of active antecedents grows with larger window sizes. Nevertheless, frequent change patterns will always be present in this set once a specific window size is reached. Thus, the curve flattens, and the run-time does not differ considerably for large window sizes. For the Wikipedia dataset, the run-time is roughly constant. The reason for this is the sparsity of the dataset: infoboxes do not change at high frequencies, and the number of active antecedents barely increases for time windows covering several hours.

Another dimension is the input data size. We define it as the number of change patterns whose occurrences we need to consider. Its relationship to the run-time can be seen in Figure 4(iii). For both datasets, the plot shows a quadratic dependency. Disregarding pruning, CR-Miner has to combine all occurrences of all change patterns within the specified window size. As the number of these combinations increases quadratically with the number of change patterns, we expect this behavior.

Quality thresholds, which are enforced for the resulting change rules, also affect the performance. When investigating the minimum confidence, we observe a negative correlation for the Socrata dataset, as seen in Figure 4(iv). CR-Miner uses the minimum confidence for pruning, i.e., it avoids combinations of consequents and active antecedents for a given point in time. For this dataset, pruning is effective. Due to the previously discussed sparsity of the Wikipedia dataset, pruning does not have a noticeable effect.

Figures 4(v) and 4(vi) show the experimental results for the support thresholds. Both thresholds are used for pruning and filter the input of CR-Miner, too. In general, run-times decrease

when a higher minimum support is required. For the Socrata dataset, this results in an approximately quadratic decrease, with matches our observations regarding input size. As seen before, the run-time of CR-Miner when applied to the Wikipedia dataset is constant for most of the maximum support thresholds. Only for extremely low values, the run-time is significantly higher. Again, the cause for this is the data distribution: In Section 5.1, we mentioned that over 80 % of the change patterns have only a single occurrence, and no change pattern has a support higher than 0.1. Thus, when enforcing higher thresholds, the input dataset is empty. Regarding maximum support, we observe the inverse behavior for both datasets.

Finally, the partition size is a pure performance parameter of our algorithm and does not affect the results. Figure 4(vii) shows the run-time for different input and partition sizes when applied to the Socrata dataset. For all input sizes, larger partition sizes lead to longer run-times for two reasons. First, change patterns are not equally correlated for different combinations of partitions. Thus, the algorithm runs longer for some combinations, and with fewer partitions, the work is less evenly distributed. Second, we use ten cores in this experiment. Not all of them are used at a specific partition size, in particular when there are fewer combinations of partitions than cores. The graph shows an extreme example of this behavior for the input size 1 000: Once the partition size equals the input size, all work is done by one core. Yet, for this small input size, a partition size of 200 performs slightly better than other choices.

The results for the Wikipedia dataset, as shown in Figure 4(viii), are different. Very small partition sizes cause a significant increase in run-time. CR-Miner iterates over all points in time for each combination of partitions. Their number is higher due to the hourly granularity, and as the change pattern occurrences are sparse, this leads to overhead. For large partition sizes, the behavior resembles the Socrata dataset. Still, there is a reasonable partition size for all input sizes, which lies around 10 000. In conclusion, the choice of a suitable partition size depends on the characteristics of the dataset.

5.3.2 Comparative Experiments against WINEPI. As mentioned in Section 2, the task of mining change rules is similar to episode mining [27]. However, existing techniques do not scale to the sizes of our datasets. We show this by comparing against an implementation of WINEPI [27][6]. We tweaked WINEPI to search only for episodes of size 2 to allow for a fair comparison to CR-Miner, which also only discovers change rules with two elements. Both algorithms are implemented in Python, were run on the same server, and were given the same filtering parameters (minimum support and confidence) as detailed in Section 5.2.2. Additionally, CR-Miner was also given the maximum support parameter, which WINEPI does not support. Main memory was limited to 64 GB and the algorithms were terminated after 24 hours. As WINEPI is unable to process datasets of our size, we randomly sampled the event types from our datasets with different sampling rates.

Figure 5 shows the results for the Socrata and the Wikipedia datasets. Run-time for the wikipedia dataset was obtained by processing all categories (see Table 2) individually and summing up the individual run-times. The figure shows that WINEPI already takes longer than 24 hours to complete for 10 % of the Socrata dataset and 1 % of the Wikipedia dataset. Furthermore, as WINEPI materializes all candidates before validation, it runs out of memory for any sample of 40 % or higher of the Socrata dataset (5 % or higher for Wikipedia). This demonstrates the superiority of CR-Miner, which is able to handle the full datasets in a comparably short amount of time, even with just a single thread.

[6] https://github.com/HPI-Information-Systems/winepi_serial_length2

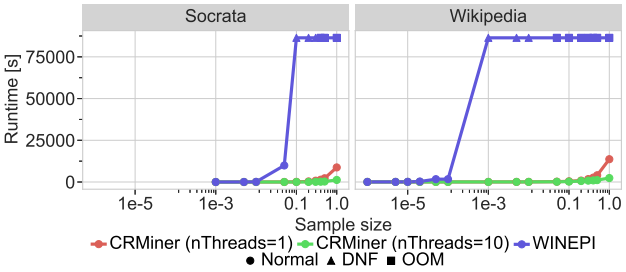


Figure 5: Run-time of all approaches. Datapoints where an approach was not able to finish in time are plotted as 24 hours. The shapes show whether the process did not finish in time (DNF) or ran out of memory (OOM).

Table 3: Wikipedia infobox types and change rules per category.

		# Change Rules	
Category	# Infobox Types	overall	external
Education	8	23	0
Geography	18	81	0
Media	33	951	62
Military	6	171	4
Politics	17	649	18
Sports	51	17 165	4 372
Transport	14	5	0

5.4 Discovered Change Rules

In the following, we examine the discovered change rules for both datasets. The results differ in terms of their interpretability: Changes in Wikipedia infoboxes can usually be explained without further domain knowledge, whereas the Socrata data lacks this advantage due to its administrative and numeric nature.

5.4.1 Change Rules in Wikipedia. Within the seven categories, our algorithm discovered more than 19 000 change rules in 77 minutes. As we can see in Table 3, these rules are not equally distributed between the categories: The vast majority belongs to the *sports* category. We distinguish *internal* and *external* rules, where antecedent and consequent are change patterns of the same and different infobox instances, respectively. It seems natural that multiple values of the same infobox are frequently modified simultaneously and create many uninteresting change rules. Thus, we do not consider internal rules in all further results and distinguish between the total and external count in Table 3. After excluding those internal rules, 4 456 change rules between different infobox instances, i.e., external rules, remain.

The *sports* category still provides the vast majority of external rules, namely 4 372 rules. One typical example of rules in this category is the one between the football players *Steven Whittaker* and *Efe Ambrose*, as introduced earlier. Whenever Whittaker's number of matches played for the fourth club in his career is updated, Ambrose's number of matches for his sixth club is updated with a confidence of 0.97 within 17 hours. With an absolute

support of 31 occurrences for this change rule, the rule holds in 31 out of 32 cases. Both counters refer to the same club, the Edinburgh-based *Hibernian F.C.*, and both players signed there from 2017 on. Thus, they mostly participated in the same football matches, and their number of matches are updated accordingly.

This is just one example of a majority of change rules in the *sports* category that capture correlating numbers of matches or points of multiple athletes. These correlations result in numerous rules, because rules capture pair-wise relations. To facilitate further investigation, we cluster change rules as subgraphs of infoboxes, where the rules are the edges. Thus, an infobox is in a cluster together with all other infoboxes that are connected by a change rule. This results in 176 clusters containing more than two infoboxes, and the same number of clusters with two infoboxes being interrelated. The majority of these clusters solely contain athletes that compete in the same team or league. E.g., Steven Whittaker and Efe Ambrose are part of a group of 13 interrelated players – all of them playing for Hibernian F.C. Other clusters also contain players of specific football clubs, occasionally combined with their coaches, or competitors in specific leagues, e.g., race drivers participating in the same series. For instance, clusters contain 13 DTM or 11 WTCC racing drivers, and numerous football and rugby teams. Only one cluster in the sports category does not consist of athletes and instead contains three infoboxes about basketball clubs' 2017/18 teams that competed in the same league.

The category with the second-highest number of external rules is *media*, for which we found 62 change rules. Applying the same clustering as for the sports category results in 12 clusters – eleven with two infoboxes, three with three articles, and one with nine infoboxes. This last cluster contains rules between genre updates of nine songs by the band *Talking Heads*. There has been a multi-article edit war concerning the genre of the songs, which shows in rare but repeating changes matching the pattern in the period of October 2016 to November 2018. The remaining clusters in the media category can similarly be explained by edit wars affecting multiple articles' infoboxes, except for three change rules that have different TV series as consequent and antecedent. These rules, which each involve two infoboxes, contain an update of the infobox's number of episodes. E.g., the Indian TV series *Agar Tum Saath Ho* and *TV Ke Uss Paar* premiered on October 3, 2016. Their episodes aired on the same days until their final episodes were broadcast on March 11, and March 25. Thus, editors updated the respective Wikipedia infoboxes in similar intervals.

$$C_{\texttt{id=Agar...},\ \texttt{p=num\_episodes},\ \texttt{type=update}} \overset{[0,\ 16]}{\Longrightarrow} C_{\texttt{id=TV...},\ \texttt{p=num\_episodes},\ \texttt{type=update}}$$

Nearly all remaining change rules in the *military* and *politics* categories are caused by edit wars as well. The only exception is a cluster of three infoboxes concerning the 2018 election of the US senate in California, this state's governor and lieutenant governor. These three elections happened on the same date, and during counting, their results were updated in the same periods.

Concluding, not a single change cluster contains infoboxes that miss a naturally explicable interrelation, and our approach performs well for this dataset and a series of domains. Besides edit wars, our algorithm solely found rules with real-world events causing data changes in multiple places, emphasizing that our approach yields change rules that fulfill the definition and, more importantly, provide semantic insights to the dataset.

5.4.2 Change Rules in Socrata. Overall, we discovered 20 832 change rules in the Socrata dataset in 33 minutes. The nature of this dataset makes it difficult to assess the quality of these change rules manually without further domain knowledge. Manual review of such a large number of change rules is infeasible, so we applied our ranking method to the obtained change rules, and determined a cut-off threshold, which reduced the number of interesting results to 309. We describe our ranking method in more detail in Section 5.5

We illustrate a problem when manually evaluating change rules in the Socrata dataset, for which we have no further domain knowledge, by examining the rank 8 change rule. The rule's antecedent and consequent refer to the table *Child Care & Youth Camp Licensing Program* by the Connecticut Office of Early Childhood[7]. The data is part of the licensing process for child care programs and youth camps to meet state regulations. The table contains 15 100 rows and 49 columns, e.g., the license number, the director, and the attention (for addresses). The change rules themselves state that an insert in the `Director` column is always followed by two deletes; one delete each in the `LicenseNumber` and `Attention` columns:

$$C_{\texttt{p=Director},\ \texttt{type=insert}} \overset{[1,\ 1]}{\Longrightarrow} C_{\texttt{p=LicenseNumber},\ \texttt{type=delete}}$$

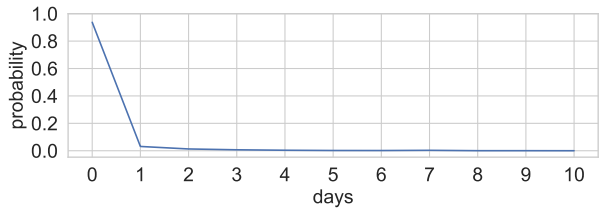
$$C_{\texttt{p=Director},\ \texttt{type=insert}} \overset{[1,\ 1]}{\Longrightarrow} C_{\texttt{p=Attention},\ \texttt{type=delete}}$$

Figure 6(ii) shows the probability distribution of the rule. It reveals that the consequent always follows on day 1 after the antecedent's occurrence. The change rule does not tell us how many fields change within the columns or if they all refer to the same row. It only states that *something* changes in the respective columns. Note that a change of type insert requires a former *null* value in an existing field or the addition of an entirely new field that is also part of a new row. One might hypothesize that there are several reasons to insert a new director. Either a new director with a new organization might apply for the licensing, or an existing organization appoints a new director and notifies the authority. While in the first scenario, an insert is reasonable, the second scenario poses the question of why the field is not just updated. The dataset owner might create a new one instead of an update, because of the application process design, legal requirements, or the table's rights-management. The following consequent's changes of `LicenseNumber` and `Attention` might be part of the same row, but this is not necessarily so. Being part of the same row would imply an underlying connection between `LicenseNumber`, `Attention` and `Director` of the same entry, which is reasonable. This exemplary change rule illustrates an inherent problem of evaluating found change rules without further domain knowledge or ground truth.

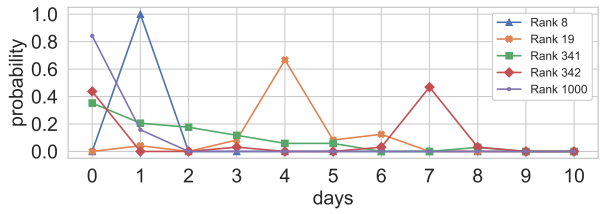
5.5 Ranking Change Rules

From the Socrata dataset we obtained 20 832 change rules in total, which we ranked according to their Jensen-Shannon divergence score as described in section 4.3. The general probability distribution for change rules we obtained from the dataset is shown in Figure 6(i). It shows that antecedent and consequent of a change rule occur primarily concurrently. Following the ranking's objective of finding rules that differ from the dominating norm, we expect top-ranked rules to have distribution maxima on day 1 or later to call them interesting (and not on day 0). This is indeed the case for the top 309 change rules in our ranking, i.e., they are distributed differently than the general probability distribution. We therefore conclude that the ranking works as desired,

[7] https://data.ct.gov/Business/Child-Care-Youth-Camp-Licensing-Program-Data/h8mr-dn95



(i) General probability distribution



(ii) Probability distributions of selected rules

Figure 6: Experiment results of the change rule ranking on the Socrata dataset.

and change rules with deviating distributions can be obtained through it.

In more detail, we see three ranges of change rules within the ranking of the change rules discovered in the Socrata dataset. The first range consists of consistently unusually distributed rules ($1 < K \leq 309$). The second range is imprecise ($309 < K \leq 354$), i.e., we observe both change rules with distributions that differ from the dominant pattern and some with similar distributions. The third range ($K > 354$) contains only change rules whose distributions are similar to the general one, and are thus not interesting in this context. Figure 6(ii) shows corresponding examples. Top-ranked change rules of the first range (e.g., rules that are ranked 8th and 19th) are characterized by peaks on day 1 or later. However, several limitations have to be considered. Our ranking approach depends on the existence of a dominant pattern. While we always found such a pattern in our datasets, this might not be generalizable. Also, an occurrence of multiple dominant patterns is conceivable. Further investigation on other datasets will be required.

To evaluate the ranking approach, we *injected* occurrences of synthetic change rules into the Socrata dataset and each of the Wikipedia categories. To this end, we randomly chose changes of five antecedent change patterns out of all points in time. For each antecedent, we chose consequent changes that always occur at one or two fixed timespans later, according to random support and confidence values. After applying CR-Miner and ranking the change rules, we automatically tested whether the injected change rules (i) were discovered with correct histograms, and (ii) are ranked high. Criterion (i) was indeed fulfilled for all injected rules across the datasets. For evaluating Criterion (ii), we use the mean and median rank of the injected rules. I.e., if all five injected rules are ranked at the top five positions, mean and median rank are both 3. This was indeed the case for four Wikipedia categories; for another two, at least the median rank is 3. For the other datasets and injected rules, the mean rank reaches up to 16.6, and the median rank up to 10. Thus, most of the injected rules are ranked high, and the proposed ranking is feasible to highlight rules with unusual distributions.

6 CONCLUSION AND FUTURE WORK

This paper introduced the concept of change rules, including formal definitions and showing their relationship to previously defined types of rules and rules. Furthermore, we proposed the CR-Miner algorithm to discover such change rules and showed how the results could be reduced to the most interesting rules even without expert knowledge about the dataset. We applied our approach to two real-world datasets and presented the results.

Experiments confirm that our algorithm scales quadratically with its input size, whereas it benefits from a stricter confidence threshold and support constraints and sparse data. For the Socrata dataset of 35 107 change patterns on a column granularity with 53 occurrences on average over a period of 359 days, 309 rare change rules with support between 0.05 and 0.1 and confidence of at least 0.9 could be discovered within 33 minutes using ten cores. For the Wikipedia dataset of almost 20 million change patterns with occurrences within four years, over 19 000 change rules with support between 0.05 and 0.45 and confidence of at least 0.9 were discovered within 77 minutes. In an incremental scenario, our algorithm can avoid having to re-process the whole dataset. This is possible by serializing intermediate histograms and running Algorithm 1 for new points in time. However, there might be pruning rules that are no longer applicable.

The optional post-processing step of our algorithm, which compares the probability distributions of found change rules, is able to rank rules precisely and can help to heavily reduce the change rules to review. In our experimental setup, we could reduce the total number of results of more than 20 832 rules to 309 interesting ones to review for the Socrata dataset.

With the explorative nature of our work, open questions and challenges remain in this field of research. We now name some of the more relevant issues as potential next steps building upon our contribution.

We have focused on the binary event of a change, i.e., whether a change of value in an entity occurred or not. It shall be interesting to consider the value itself – before and after the change event in various use cases. Thus, the severity of a change can be measured and, for instance, used as a weight in the algorithm. Moreover, we have concentrated our work on finding change rules where the antecedent as well as the consequent consist of one change pattern. However, our proposed framework would also allow multiple change pattern as antecedent, increasing both the search space and the potential interesting rules to find.

Furthermore, we can imagine possible change rules not only at the same levels of granularity, as we have explored in this paper, but also rules across different granularity levels. For example, a *field update* in one table can lead to a *row insert* in another table. Traversing those, adds another layer of complexity to the number of candidate change rules.

While we have mined the data for change rules that fulfil certain support and confidence thresholds over the entire time frame, some rules might occur only during a specific period (e.g., a specific season or only as long as two football players play for the same club) or show a dynamic pattern. We see potential in combining or approach with knowledge from sequential rule mining to investigate on those dynamic change rules.

ACKNOWLEDGMENTS

We thank Divesh Srivastava and Dmitri V. Kalashnikov for their continued support of our change exploration endeavors in the context of the Janus project (www.IANVS.org).

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 207–216.

[2] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*. 487–499.

[3] James F Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26, 11 (1983), 832–843.

[4] Xiang Ao, Ping Luo, Chengkai Li, Fuzhen Zhuang, and Qing He. 2015. Online frequent episode mining. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 891–902.

[5] Xiang Ao, Haoran Shi, Jin Wang, Luo Zuo, Hongwei Li, and Qing He. 2019. Large-scale frequent episode mining from complex event sequences with hierarchies. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 4 (2019), 1–26.

[6] Sikha Bagui, Jiri Just, and Subhash C. Bagui. 2009. Deriving strong association mining rules using a dependency criterion, the lift measure. *International Journal of Data Analysis Techniques and Strategies (IJDATS)* 1 (2009), 297–312. Issue 3.

[7] Tobias Bleifuß, Leon Bornemann, Theodore Johnson, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2018. Exploring Change: A New Dimension of Data Analytics. *PVLDB* 12, 2 (2018), 85–98.

[8] Tobias Bleifuß, Leon Bornemann, Dmitri V. Kalashnikov, Felix Naumann, and Divesh Srivastava. 2021. Structured Object Matching across Web Page Revisions. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 1284–1295.

[9] Siarhei Bykau, Flip Korn, Divesh Srivastava, and Yannis Velegrakis. 2015. Fine-grained controversy detection in Wikipedia. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 1573–1584.

[10] Yi-Cheng Chen, Wen-Chih Peng, and Suh-Yin Lee. 2015. Mining temporal patterns in time interval-based data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 27, 12 (2015), 3318–3331.

[11] Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey Ullman, and Cheng Yang. 2001. Finding interesting associations without support pruning. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 13 (2001), 64 – 78.

[12] Xin Luna Dong, Anastasios Kementsietsidis, and Wang-Chiew Tan. 2016. A Time Machine for Information: Looking Back to Look Forward. *SIGMOD Record* 45, 2 (2016), 23–32.

[13] Philippe Fournier-Viger, Chun-Wei Lin, B. Vo, Tin C. Truong, Ji Zhang, and H. Le. 2017. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 4 (2017), e1207.

[14] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. 2019. A Survey of Parallel Sequential Pattern Mining. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13, 3, Article 25 (2019).

[15] Michael Hahsler. 2006. A Model-Based Frequency Constraint for Mining Associations from Transaction Data. *Data Mining and Knowledge Discovery* 13 (2006), 137–166.

[16] Jiawei Han and Y. Fu. 1995. Discovery of Multiple-Level Association Rules from Large Databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*.

[17] Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining Frequent Patterns without Candidate Generation. *SIGMOD Record* 29, 2 (2000), 1–12.

[18] Po-Shan Kam and Ada Wai-Chee Fu. 2000. Discovering temporal patterns for interval-based events. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*. Springer Verlag, 317–326.

[19] Yun Sing Koh and Sri Devi Ravana. 2016. Unsupervised Rare Pattern Mining: A Survey. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10, 4, Article 45 (2016).

[20] Yun Sing Koh and Nathan Rountree. 2005. Finding Sporadic Rules Using Apriori-Inverse. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*. 97–106.

[21] Yun Sing Koh, Nathan Rountree, and Richard O'Keefe. 2006. Mining Interesting Imperfectly Sporadic Rules. In *Advances in Knowledge Discovery and Data Mining (PAKDD)*. Springer Verlag, 473–482.

[22] P. Lenca, B. Vaillant, Patrick Meyer, and S. Lallich. 2007. Association Rule Interestingness Measures: Experimental and Theoretical Studies. In *Quality Measures in Data Mining*. 51–76.

[23] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. 2008. PFP: Parallel FP-Growth for Query Recommendation. In *Proceedings of the ACM Conference on Recommender Systems (RECSYS)*. 107–114.

[24] J. Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information Theory (TOIT)* 37, 1 (1991), 145–151.

[25] Ming-Yen Lin, Suh-Yin Lee, et al. 2005. Fast discovery of sequential patterns through memory indexing and database partitioning. *Journal of Information Science and Engineering (JISE)* 21, 1 (2005), 109–128.

[26] Bing Liu, Wynne Hsu, and Yiming Ma. 1999. Mining Association Rules with Multiple Minimum Supports. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. Association for Computing Machinery, 337–341.

[27] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. 1997. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery* 1, 3 (1997), 259–289.

[28] F. Masseglia, F. Cathala, and P. Poncelet. 1998. The PSP approach for mining sequential patterns. In *European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*. 176–184.

[29] Bhabesh Nath, Dhruba K Bhattacharyya, and Ashish Ghosh. 2013. Incremental association rule mining: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 3 (2013).

[30] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. 2005. Discovering frequent arrangements of temporal intervals. In *IEEE International Conference on Data Mining (ICDM)*. 354–361.

[31] Dhaval Patel, Wynne Hsu, and Mong Li Lee. 2008. Mining relationships among interval-based events for classification. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 393–404.

[32] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Jianyong Wang, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 2004. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 16, 11 (2004), 1424–1440.

[33] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 1–17.

[34] Laszlo Szathmary, Amedeo Napoli, and Petko Valtchev. 2007. Towards Rare Itemset Mining. *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI)* 1, 305–312.

[35] L. Szathmary, Petko Valtchev, A. Napoli, and R. Godin. 2012. Efficient Vertical Mining of Minimal Rare Itemsets. In *International Conference on Concept Lattices and Their Applications (CLA)*. 269–280.

[36] Feng Tao, Fionn Murtagh, and Mohsen Farid. 2003. Weighted Association Rule Mining Using Weighted Support and Significance Framework. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. Association for Computing Machinery, 661–666.

[37] Wil M.P. van der Aalst. 2016. *Process Mining - Data Science in Action, Second Edition*. Springer. https://doi.org/10.1007/978-3-662-49851-4

[38] Ke Wang, Yu He, and Jiawei Han. 2003. Pushing support constraints into association rules mining. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 15, 3 (2003), 642–658.

[39] Wikipedia. 2021. *Wikipedia Help Page on Wikipedia Infoboxes*. https://en.wikipedia.org/wiki/Help:Infobox

[40] Chung-Ching Yu and Yen-Liang Chen. 2005. Mining sequential patterns from multidimensional sequence data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 17, 1 (2005), 136–140.

[41] Mohammed J. Zaki. 2000. Scalable Algorithms for Association Mining. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 12, 3 (2000), 372–390.

[42] Mohammed J Zaki. 2001. SPADE: An efficient algorithm for mining frequent sequences. *Machine learning* 42, 1 (2001), 31–60.

[43] Mohammed J. Zaki and Wagner Meira Jr. 2020. *Data Mining and Analysis: Fundamental Concepts and Algorithms* (2 ed.). Cambridge University Press.