Structured Object Matching across Web Page Revisions

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

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

Dmitri V. Kalashnikov
AT&T Labs – Research
dvk@research.att.com

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

Divesh Srivastava
AT&T Chief Data Office
divesh@att.com

***Abstract*—A considerable amount of useful information on the web is (semi-)structured, such as tables and lists. An extensive corpus of prior work addresses the problem of making these human-readable representations interpretable by algorithms. Most of these works focus only on the most recent snapshot of these web objects. However, their evolution over time represents valuable information that has barely been tapped, enabling various applications, including visual change exploration and trust assessment. To realize the full potential of this information, it is critical to match such objects across page revisions.**

In this work, we present novel techniques that match tables, infoboxes and lists within a page across page revisions. We are, thus, able to extract the evolution of structured information in various forms from a long series of web page revisions. We evaluate our approach on a representative sample of pages and measure the number of correct matches. Our approach achieves a significant improvement in object matching over baselines and over related work.

***Index Terms*—object matching, change exploration**

I. Tracking Changes on the Web

On the web, lists and tables serve as a structured, concise representation of information. Infoboxes are a standardized way to convey information about different topics on Wikipedia, so they arguably serve the same purpose. Many of those structured web objects are subject to frequent change, often because the represented information is dynamic and changes over time. Many other reasons exist for why an infobox, a list or a table changes, especially on the web: authors may enrich a table with additional information by appending new items, columns or rows; or, to condense data, less relevant parts might be deleted or moved to a different list or table. Even if the content stays unchanged, a table's format or the list's structure might be adjusted, for example, to highlight important aspects or to group related content. On parts of the web that are curated jointly by several authors, such as infoboxes, people may disagree about what the right content should be. This can go as far as to cause edit-wars [1]. Another undesired but highly prominent source of changes on the web is vandalism, which is deliberately destructive [2].

A. Challenges and goals

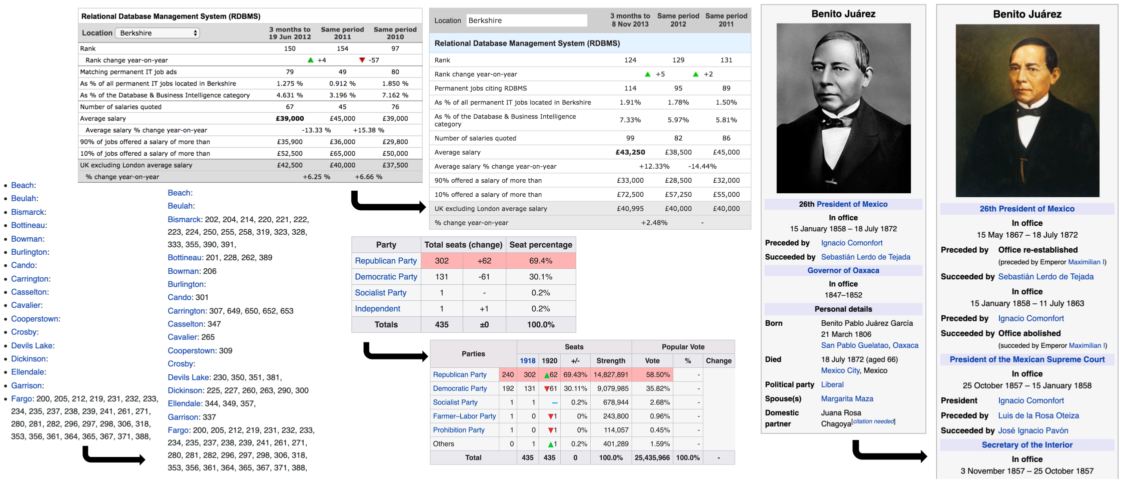
It is essential to obtain the history of individual objects (i.e., tables, lists or infoboxes) to study and explore these changes and their variety – we present several use-cases later in this section. The first hurdle is the retention of different document versions that contain the objects. For instance, the Internet Archive (www.archive.org) provides, to a limited extent, the history of crawled web pages. Some web projects, especially those with user participation, save the history of the changes themselves. The most prominent and important example is Wikipedia, which provides every version of every page as a document in Wikitext markup. In fact, for the approximately 2.7 million tables on more than a million Wikipedia articles, we have collected 40 million revisions.

To analyze this data, a new challenge arises: Given two consecutive versions of the same page, the two versions of Wikitext or HTML do not unambiguously reveal which structured object of the old page matches to which such object of the new page, or whether it matches any at all. The underlying problem is not web-specific, but arises frequently for evolving data, e.g., in data lakes or open data. In fact, our proposal of the *change-cube* as a universal data model to capture changes identifies populating those change-cubes as one of the fundamental challenges to enable change exploration in data and metadata [3].

Clearly, we can compare two document versions line by line, but that line-wise difference is purely syntactic and the resulting diff-set does not necessarily represent the intentions of the user. Line-wise differences assume that the (relative) positions of objects stay constant over their lifetime and this method is hence incapable of identifying objects, such as web tables, that have moved within the web page. Our aim is to link multiple versions of the same object over multiple revisions in a way that reflects the actual edits performed by users. Multiple object instances may be different versions of the same object; the *identity graph* connects all such object instances.

The problems of structured object matching become more challenging for web pages that contain many objects. The larger candidate spaces make incorrect matching decisions more likely, and such errors propagate easily. Another challenging case arises when individual objects contain scarce or volatile information. This case resembles the philosophical problem of the ship of Theseus: the question of how many timbers of a ship can be exchanged while it still remains to be the same ship [4]. Figure 1 shows examples for the wide variety of changes that objects on web pages can undergo.

When solving the problem of temporal object matching, that is, to construct the identity graph, we benefit from the



Relational Database Management System (RDBMS)			
Location: Berkshire	3 months to 19 Jun 2012	Same period 2011	Same period 2010
Rank	150	154	97
Rank change year-on-year	▲ +4	▼ -57	
Matching permanent IT job ads	79	49	80
As % of all permanent IT jobs located in Berkshire	1.275 %	0.912 %	1.850 %
As % of the Database & Business Intelligence category	4.631 %	3.196 %	7.162 %
Number of salaries quoted	67	45	76
Average salary	**£39,000**	£45,000	£39,000
Average salary % change year-on-year	-13.33 %	+15.38 %	
90% of jobs offered a salary of more than	£35,900	£36,000	£29,800
10% of jobs offered a salary of more than	£52,500	£65,000	£50,000
UK excluding London average salary	£42,500	£40,000	£37,500
% change year-on-year	+6.25 %	+6.66 %	

Location: Berkshire	3 months to 8 Nov 2013	Same period 2012	Same period 2011
Relational Database Management System (RDBMS)			
Rank	124	129	131
Rank change year-on-year	▲ +5	▲ +2	
Permanent jobs citing RDBMS	114	95	89
As % of all permanent IT jobs located in Berkshire	1.91%	1.78%	1.50%
As % of the Database & Business Intelligence category	7.33%	5.97%	5.81%
Number of salaries quoted	99	82	86
Average salary	**£43,250**	£38,500	£45,000
Average salary % change year-on-year	+12.33%	-14.44%	
90% offered a salary of more than	£33,000	£28,500	£32,000
10% offered a salary of more than	£72,500	£57,250	£55,000
UK excluding London average salary	£40,995	£40,000	£40,000
% change year-on-year	+2.48%	-	

- Beach:
- Beulah:
- Bismarck:
- Bottineau:
- Bowman:
- Burlington:
- Cando:
- Carrington:
- Casselton:
- Cavalier:
- Cooperstown:
- Crosby:
- Devils Lake:
- Dickinson:
- Ellendale:
- Garrison:
- Fargo: 200, 205, 212, 219, 231, 232, 233, 234, 235, 237, 238, 239, 241, 261, 271, 280, 281, 282, 296, 297, 298, 306, 318, 353, 356, 361, 364, 365, 367, 371, 388,

Beach:

Beulah:

Bismarck: 202, 204, 214, 220, 221, 222, 223, 224, 250, 255, 258, 319, 323, 328, 333, 355, 390, 391,

Bottineau: 201, 228, 262, 389

Bowman: 206

Burlington:

Cando: 301

Carrington: 307, 649, 650, 652, 653

Casselton: 347

Cavalier: 265

Cooperstown: 309

Crosby:

Devils Lake: 230, 350, 351, 381,

Dickinson: 225, 227, 260, 263, 290, 300

Ellendale: 344, 349, 357,

Garrison: 337

Fargo: 200, 205, 212, 219, 231, 232, 233, 234, 235, 237, 238, 239, 241, 261, 271, 280, 281, 282, 296, 297, 298, 306, 318, 353, 356, 361, 364, 365, 367, 371, 388,

Party	Total seats (change)		Seat percentage
Republican Party	302	+62	69.4%
Democratic Party	131	-61	30.1%
Socialist Party	1	-	0.2%
Independent	1	+1	0.2%
Totals	435	±0	100.0%

Parties	Seats				Popular Vote		
	1918	1920	+/-	Strength	Vote	%	Change
Republican Party	240	302	▲62	69.43%	14,827,891	58.50%	-
Democratic Party	192	131	▼61	30.11%	9,079,985	35.82%	-
Socialist Party	1	1	–	0.2%	678,944	2.68%	-
Farmer–Labor Party	1	0	▼1	0%	243,800	0.96%	-
Prohibition Party	1	0	▼1	0%	114,057	0.45%	-
Others	0	1	▲1	0.2%	401,289	1.59%	-
Total	435	435	0	100.0%	25,435,966	100.0%	-

Benito Juárez	
26th President of Mexico	
In office 15 January 1858 – 18 July 1872	
Preceded by	Ignacio Comonfort
Succeeded by	Sebastián Lerdo de Tejada
Governor of Oaxaca	
In office 1847–1852	
Personal details	
Born	Benito Pablo Juárez García 21 March 1806 San Pablo Guelatao, Oaxaca
Died	18 July 1872 (aged 66) Mexico City, Mexico
Political party	Liberal
Spouse(s)	Margarita Maza
Domestic partner	Juana Rosa Chagoya[citation needed]

Benito Juárez	
26th President of Mexico	
In office 15 May 1867 – 18 July 1872	
Preceded by	Office re-established (preceded by Emperor Maximilian I)
Succeeded by	Sebastián Lerdo de Tejada
In office 15 January 1858 – 11 July 1863	
Preceded by	Ignacio Comonfort
Succeeded by	Office abolished (succeded by Emperor Maximilian I)
President of the Mexican Supreme Court	
In office 25 October 1857 – 15 January 1858	
President	Ignacio Comonfort
Preceded by	Luis de la Rosa Oteiza
Succeeded by	José Ignacio Pavón
Secretary of the Interior	
In office 3 November 1857 – 25 October 1857	

Fig. 1: Examples of infobox, list and table revisions.

observation that changes typically happen gradually, so that the context of changed elements can be used to help identify elements over time. We model this task as a matching problem, where for every new document version, we either match an object instance to a previous history or consider it to be a new object, if no sufficiently similar history exists.

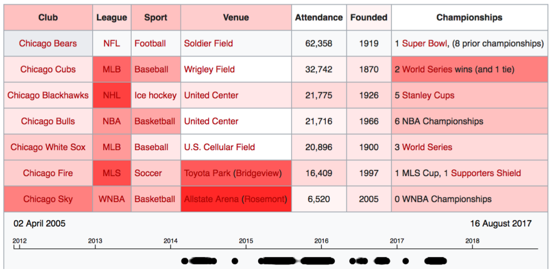
Example 1: Consider common Wikipedia pages like "List of awards and nominations received by . . . " for musicians or actors. These pages usually contain a separate table for each award, such as a table for "MTV Award". At first glance, these tables may seem quite static, but in fact, they are extended whenever awards are won, and they are changed, e.g., with an update of the result from "nominated" to "won", with a layout revision, with new links that are inserted, etc. Many of these tables on a page also look very similar, because they all have the same schema, and common award titles (like "best actor") exist for different events. This, together with the small sizes of the tables, makes matching difficult. Similar considerations apply to lists and infoboxes.

The identity graph makes the object history available as a new dimension of each element, because for each object it tells us about previous and subsequent versions. A very first application could be to overlay an object with a heatmap to show the volatility of each cell (see Figure 2).

B. Use cases

Knowledge about matching objects across web page revisions can serve many different use cases in the context of web mining. We briefly outline a few, partly taken from the recent overview of web table usage [5]. Generally, knowledge of the history of an object enriches the knowledge about and understanding of the contained information.

1) Understanding web tables: There are many approaches to enhance knowledge of individual web tables, such as generating their title [6], generating column headers [7], or finding subject columns [8]. All of these approaches make use of table content, headers and surrounding text and data.



Club	League	Sport	Venue	Attendance	Founded	Championships
Chicago Bears	NFL	Football	Soldier Field	62,358	1919	1 Super Bowl, (8 prior championships)
Chicago Cubs	MLB	Baseball	Wrigley Field	32,742	1870	2 World Series wins (and 1 tie)
Chicago Blackhawks	NHL	Ice hockey	United Center	21,775	1926	5 Stanley Cups
Chicago Bulls	NBA	Basketball	United Center	21,716	1966	6 NBA Championships
Chicago White Sox	MLB	Baseball	U.S. Cellular Field	20,896	1900	3 World Series
Chicago Fire	MLS	Soccer	Toyota Park (Bridgeview)	16,409	1997	1 MLS Cup, 1 Supporters Shield
Chicago Sky	WNBA	Basketball	Allstate Arena (Rosemont)	6,520	2005	0 WNBA Championships

Fig. 2: Browser plugin displaying the history of Wikipedia tables highlighting frequently changing cells.

Providing more such data, and in particular different versions of such data gives these machine learning approaches a richer input set, which has recently been proven useful for fact-extraction [9] and key discovery [10]. For the latter task, we show the benefit of having a table's version history available in our case study in Section V-E.

2) Entity linking: Linking entities across different locations on the web is a popular but difficult task, aiming to identify entity representations in different data sources representing the same real-world entity [11], [12]. As data on the web is updated independently, old and new representations exist simultaneously. With knowledge of an object's history, linking approaches can make use of the higher similarity of records that represent data about the same point in time. Records sharing the same values for large parts of their lifetime allow a more confident linking.

3) Search in web data: Researchers have identified the rich nature of web tables and the problem of making available this data through advanced search interfaces. In particular, such search systems cannot assume fixed schemata and user knowledge about how entities are described [13]. Being able to index past data, and using past data to build more general models can improve the search experience. For instance, the authors of [14] propose using a wide set of "clues" about a table's schema and content to map search queries to specific tables and columns. Extending these clues to past versions of tables has the potential to improve the results.

4) Data quality: The ability to identify object changes, and knowledge about matching entities, allows propagating changes across sources, improving their timeliness. For example, WiClean [15] uses frequent change patterns of the Wikipedia link graph to help users perform complete and consistent updates. Further, mining change patterns can point to unexpected changes or to expected changes that did not occur, thus enabling quality assessment of the data contained by the object [3] or past changes can hint at the most likely fixes for constraint violations [16].

C. Contributions and overview

Our main contributions toward tracking the history of structured web objects are the following:

- A definition of the generalized problem of temporal object matching and three specific problem instances: table, infobox and list matching.
- A general content- and context-based solution combining multiple similarity-based matching stages to achieve both high precision and high recall.
- Application of this general solution to the three matching problems of tables, infoboxes and lists. For infobox- and table-matching we achieve F_1-measure above 99%, and for list-matching above 98%.
- Manually annotated gold-standards for all three problems, which we use for our empirical evaluation and make publicly available for repeatability.

The remainder of this paper is structured as follows: We present related work in Section II, a formal problem definition in Section III, and our matching process in Section IV. We evaluate our results in Section V and conclude in Section VI.

II. Related Work

We discuss a variety of related work: structured datasets with history, related techniques and matching problems, and Wikipedia change analysis.

1) Related corpora: Wikipedia provides access to its entire version history, allowing us to track very fine-grained changes. A variety of datasets that also deal with (semi-) structured content have been extracted from the web and Wikipedia before. Multiple corpora of web tables [17], [18] provide extracts of static versions of tables on the web and have since been subject to extensive research [5]. The infobox history dataset WHAD [19] comprises structured information on Wikipedia, namely the changes of infoboxes. However, there is no matching problem within one infobox instance, as each property of an infobox has a unique identifier and property labels are drawn from standardized templates. Additionally, the authors do not consider the fact that a Wikipedia article can contain multiple infoboxes, effectively ignoring any matching problem.

2) Related problems and techniques: Korn et al. propose a table matching approach as part of their work on fact-extraction [9]. We experimentally compare our table matching solution to their work. A problem related to table matching, which also aims to combine several semantically related tables, is table union search or table stitching [20]–[22]. That problem is orthogonal to our problem, as these works do not consider the time dimension and changing data, but instead try to find additional data that is not already present in the table.

For our problem as well as for the table union search, tables that match do not necessarily share the same schema. Given multiple schemata, the vast work on schema matching [23] attempts to match semantically corresponding attributes and can help to perform the actual integration of the information contained in the tables. Especially in web tables, the schema is often opaque, so in many cases only instance-based techniques [24], [25] are applicable.

Record linking aims to identify which specific records in a set of records refer to the same entity. This problem has been extensively studied [26]. Recently, changes in attributes over time have been considered, for linking temporal records [12], [27]–[30]. However, this body of work typically assumes a well-defined and static schema, which does not apply to web tables and their version history, because of their opaque schema and column headers, which change over time.

3) Analyzing changes in Wikipedia: The content of Wikipedia has been the subject of much research [31]. Both the evolution of content [32] and of the page link graph [33] including the links' anchor texts [34] have been studied. Specifically, the study of content evolution can help detect conflicts [35] or controversy that may result in edit-wars [1]. The edit histories serve as input to event-extraction [36] and are also valuable for trust assignment [37]. Our approach can provide a better understanding of what has really changed, which should benefit many of these studies.

III. Formal Problem Definition

While we gave an intuition for the *identity graph* in Section I, we now provide a definition of the graph and formally define the addressed problem itself. First, we introduce some of the terms used in the paper: Every change to a web page – *context* in general – represents a *revision*. The result of a revision is a new *version* of that context. Each version of a context contains several *object instance*s (such as tables, infoboxes or lists). These object instances can be new versions of previously *identified object*s or represent *new objects*. For example, object instances are shown in Figure 3 by colored circles. In Version 1, the blue, red and yellow circles are new versions of previously identified objects of Version 0, while the green circle represents a new object.

Definition 1: The *identity graph* of a sequence of context versions is a graph that satisfies the following conditions:

- Every object instance contained in the context versions is represented as a node.

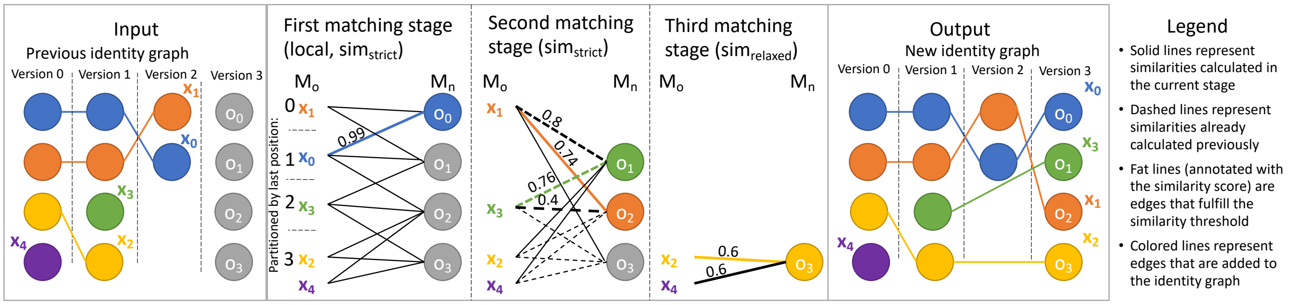

Input
Previous identity graph
Version 0
Version 1
Version 2
Version 3
First matching stage (local, sim_strict)
Second matching stage (sim_strict)
Third matching stage (sim_relaxed)
Mo
Mn
Partitioned by last position:
0.99
0.8
0.74
0.76
0.4
0.6
0.6
Output
New identity graph
Legend
Solid lines represent similarities calculated in the current stage
Dashed lines represent similarities already calculated previously
Fat lines (annotated with the similarity score) are edges that fulfill the similarity threshold
Colored lines represent edges that are added to the identity graph

Fig. 3: An illustrative example that shows how the different matching stages interact.

- Every edit of an object instance that results in another object instance is represented as an identity edge. As there is only one object instance for “creation” or “deletion” of an object, there are no incoming respectively outgoing identity edges in these cases.

Identity graphs are linear graphs (there is at most one incoming and one outgoing edge for every node), because each object can occur only once in each context version. We refer to the connected components in the identity graph as (previously) *identified objects*, and all object instances that are part of this connected component the *object’s versions*. Two adjacent (object instance) nodes in an identity graph may occur in non-consecutive context versions (such as the green object in Version 1 and Version 3 of Figure 3). The identity graph carries information about the physical nature of an object; in the editing process a user does not physically merge or split an object.

We can now state the problem of *temporal object matching* as follows: Given a list of versions $r_1, \ldots, r_n$ of contexts, e.g., web pages in our case, where each version r_i contains a set of object instances of interest (e.g., infoboxes, lists or tables), the problem of *temporal object matching* is to construct the identity graph for these object instances (so to construct all edges in the output graph of Figure 3). In this paper, we solve three instances of this problem, namely temporal matching of tables, lists and infoboxes. As the context for each of these objects are pages, we use “pages” instead of “context” in the following section. The method nevertheless applies to other contexts, such as domains in open data lakes as we show in Section V-B, where different subdomains can act as context.

IV. Matching Objects over Time

This section explains our general solution to the temporal matching problem. Its input is a list of web page versions – either snapshots that have been crawled, or specifically from Wikipedia the complete edit history as a set of XML files. Through parsing, we obtain for every page version a (possibly empty) list of parsed object nodes. From those we extract a number of features for each of the object versions. Now we can perform the actual matching, which performs a matching step for each page change: we extract a list of all objects contained in the page that are of the object type (infobox, list, or table) that is currently matched. This list, which corresponds to the gray nodes o_0 to o_3 shown in Figure 3, in combination with the previously identified objects of the same type, constitutes the input for the actual matching step.

In each matching step, i.e., for each page revision, it is necessary to decide whether the objects therein are versions of previously identified objects or entirely new objects. It is not sufficient to consider only objects of the previous version, because objects can be deleted and then restored several revisions later (such as the green object that reappears in Version 3). This step-wise approach that matches object versions with only the information available up until this point, makes it suitable for an online scenario in which new versions arrive incrementally. This section first explains this matching step in general before we discuss the exact implementation for the three object types that we consider in this paper.

A. Matching step overview

Our approach is based on measures of similarity between objects and new object instances. We first match objects that have very high similarity and then allow larger changes (i.e., lower similarity) for the remaining objects. Within each stage, we find a maximum-weight matching.

1) Multi-stage matching: The matching process is divided into three stages:

1) a local search,
2) a strict search, and
3) a relaxed search,

each based on measures of similarity between objects. Within each stage, only elements that have not yet found a match are considered further, and we find a maximum-weight matching. The first stage is mainly a performance optimization; the second stage obtains high precision matches, and the third stage increases recall.

Each matching stage $s_i = (\lambda, \mathrm{sim}, \theta)$ is defined by

1) a candidate generation function $\lambda : (X = \{\mathbf{x_1}, \ldots, \mathbf{x_m}\}, O = \{o_1, \ldots, o_n\}) \rightarrow \mathcal{P}(\{\mathbf{x_1}, \ldots, \mathbf{x_m}\} \times \{o_1, \ldots, o_n\})$,
2) a similarity function $\mathrm{sim} : (x, o) \rightarrow \mathbb{R}$ and
3) a similarity threshold $\theta \in \mathbb{R}$.

The candidate generation function determines a set of possible matching pairs (for which to evaluate the similarity function).

```
Algorithm 1: Matching step
  Data: previous objects $\{\mathbf{x_1}, \ldots, \mathbf{x_m}\}$,
        new object versions $[o_1, \ldots, o_n]$
  Parameters: Matching stages $[s_1, \ldots, s_n]$, past-version
              window size $k$, decay factor $\varphi$
  Result: list of objects $O$ that contains previously
          identified objects (possibly with new versions)
          and newly identified objects
1 $M_o \leftarrow \{\mathbf{x_1}, \ldots, \mathbf{x_m}\},\ M_n \leftarrow \{o_1, \ldots, o_n\},\ O \leftarrow M_o$
2 for $(\lambda, \mathrm{sim}, \theta) \in [s_1, \ldots, s_n]$ do
3     $C \leftarrow \lambda(M_o, M_n)$
4     $\mathcal{G} \leftarrow (M_o \cup M_n, \{(\mathbf{x}, o) \in C \mid \hat{\mathrm{sim}}_{k,\varphi}(\mathbf{x}, o) \geq \theta\})$
5     for $(\mathbf{x}, o) \in$ matching$(\mathcal{G}, \hat{\mathrm{sim}}_{k,\varphi})$ do
6         $\mathbf{x} \leftarrow [\mathbf{x}, o]$, $M_o \leftarrow M_o \setminus \mathbf{x}$, $M_n \leftarrow M_n \setminus o$
7 return $O \cup \{[o] \mid o \in M_n\}$
```

This function takes two arguments: the set X of identified objects and a list O of new object instances. The default candidate generation function considers all remaining, non-matched pairs of previously identified objects and new object instances: $\lambda(X, O) = \{(\mathbf{x}, o) \mid \mathbf{x} \in X, o \in O\}$. The similarity function sim maps pairs of object versions to similarity scores. The similarity threshold θ determines the minimum similarity score that can still create a match; Section V-C explains the influence and choice of these thresholds.

Algorithm 1 iterates over all stages (in our case exactly three stages) in Line 2. For each stage, in Line 3 the candidate generation λ creates all pairs of previously identified objects and new object instances to be considered in this stage. The algorithm then constructs a bipartite graph, where each of these pairs serves as an edge with a weight based on sim (Line 4), if this similarity score is not smaller than the threshold θ. To be more precise, the graph construction is based on an extended similarity function $\hat{\mathrm{sim}}_{k,\varphi}$, which also considers past object versions and which is formally defined next, in Section IV-A2. Finally, Line 5 finds a maximum-weight matching using the Hungarian algorithm [38] and iterates over its edges. The matching breaks ties based on the two ranking functions $\downarrow_{\mathrm{LT}}$ and $\downarrow_{\mathrm{POS}}$, which are explained in Section IV-A3 below. For each match in the assignment, the algorithm assigns the candidate's object instance to the object $\mathbf{x}$ and marks both of them as matched (Line 6). Objects for which no matching partner has been found, end up in the pool for potential matches in the next stage. Finally, once all stages are executed, all remaining object instances are considered to be newly created objects (Line 7).

2) Glance through the rear-view mirror: Many of the changes we observe for Wikipedia are relatively short-lived, because they are quickly reverted. For this reason a "look through the rear-view mirror" allows matches with objects from revisions further in the past, even if the object was not present in the meantime. In fact, we observed such cases of re-appearing objects 2,343 times for infoboxes, 2,741 times for lists, and 4,879 times for tables in our gold standard alone. When comparing a new object instance with a previously identified object, we compare it not only with the latest version of that object, but with the k latest (non-empty) versions. We try to keep k small, because the time for similarity calculation grows linearly with k, and because objects can be subject to a topical shift, potentially leading to false positives. However, even a small $k = 5$ affords robustness against short-lived changes like vandalism (see Section V-C for an experiment on choosing k).

The similarity measure should also reflect the fact that a new object instance resembles only the object of a previous revision, and should give priority to similarities with more recent object versions. For this reason, we weight the similarity with a decay-factor that scales exponentially with the number of different object versions that have occurred in the meantime. For an object $\mathbf{x}$ that has the object versions $x_0, \ldots, x_n$ (with x_0 as the first version and x_n as the most recent), we define the similarity to a new object version o as the maximum similarity of the last k versions weighted by the corresponding power of the decay factor $\varphi \in [0, 1]$:

$$\hat{\mathrm{sim}}_{k,\varphi}(\mathbf{x} = [x_0, \ldots, x_n], o) = \max_{i=0 \to \min(n,k)-1} \varphi^i \,\mathrm{sim}(x_{n-i}, o)$$

3) Meta-data similarity: A common phenomenon in Wikipedia is that a user unintentionally duplicates parts of an article or even the entire article. This results in two (or more) object instances that both have exactly the same similarities to a previous object. As the content of both object instances is the same, the decision of which objects to match, can rely only on meta-data, such as the position of the object instance in its environment. If several object instances have high similarity with only a small difference between them, our system prefers matches that minimize the difference between the previous position and the position of the new object instance.

$$\downarrow_{\mathrm{POS}} (\mathbf{x} = [x_0, \ldots, x_n], o) = |\,\mathrm{position}(x_n) - \mathrm{position}(o)|$$

An inverse problem occurs when such duplicated objects are deleted at a later point in time. In this case, two existing objects have exactly the same similarity to a single new object instance, and it is necessary to decide which one remained and which one was deleted. We use a heuristic: if in doubt, we match the object with a longer lifetime, because longer and therefore more comprehensive histories are preferable.

$$\downarrow_{\mathrm{LT}} (\mathbf{x} = [x_0, \ldots, x_n], o) = \mathrm{lifetime}(\mathbf{x})$$

B. Object matching details

Now that we have explained the basic matching procedure, we can define the exact features, similarity measures and stages for the object matching.

1) Object features: To compare the content of different tables, lists, or infoboxes, we create a bag-of-words representation of their content. In order to not let elements (such as cells or items) with long content dominate this representation, we truncate element values after 10 words. The bag-of-words approach has the advantage that it is robust against larger changes, such as table layout changes and list reorderings; a representation that considers the position of words within the object would change much more when layout changes occur. Furthermore, we add all appropriate hierarchical section titles or HTML headings of the surrounding sections to the bag-of-words. For object version o, we create one vector $\vec{v_o}$ where every dimension represents the number of occurrences for one specific word either in the object or in its section headers. Besides those content-features, we also consider certain metadata of the object. The position of the object is the position-rank of the object among all objects of the same type on the page (in the order the objects appear in the page source).

2) Similarity measures: For object matching, we calculate similarities of the bag-of-words vectors based on a generalized Jaccard similarity coefficient, also called Ruzicka similarity [39], of two multisets. This Ruzicka similarity $\text{sim}_{\textit{strict}}$ is defined as the sum of the respective lower frequencies (across both objects) for each word (occurring in at least one of the two objects) divided by the sum of the respective higher frequencies for each word.

$$\text{sim}_{\textit{strict}}(\vec{v} = {\scriptstyle (v_1,\ldots,v_n)}, \vec{w} = {\scriptstyle (w_1,\ldots,w_n)}) = \frac{\sum_i \min(v_i, w_i)}{\sum_i \max(v_i, w_i)}$$

Due to its normalizing denominator, the Ruzicka similarity penalizes object size changes. As many objects grow or shrink over time, we introduce a relaxed version of this similarity measure, an element-wise containment $\text{sim}_{\textit{relaxed}}$ that we apply in our last matching stage. In comparison to Ruzicka, this similarity measure uses the token count of the smaller object as the normalizing denominator.

$$\text{sim}_{\textit{relaxed}}(\vec{v} = {\scriptstyle (v_1,\ldots,v_n)}, \vec{w} = {\scriptstyle (w_1,\ldots,w_n)}) = \frac{\sum_i \min(v_i, w_i)}{\min(\sum_i v_i, \sum_i w_i)}$$

Some tokens appear in many objects and are of little help in deciding whether an object should be the successor of another object. To account for such noise, we use a token weighting that is similar to the idea of inverse document frequencies (IDFs). More precisely, we multiply the token frequencies by the inverse of the number of new or previously identified objects containing this token, whichever is larger. As a result, a token that appears in up to one previously identified object and up to one new object version is not given less weight. For instance, a token that appears in three previously identified and five new object versions would be weighted $\frac{1}{5}$.

3) Object matching stages: We have observed that in many cases the position of an infobox, a list or a table changes only slightly, so many good matches can be found efficiently by a local search. Thus, the *first stage* searches for very good match partners locally, i.e., in a very limited spatial context of the previously identified object, on the revised page. This stage avoids the calculation of similarities of all possible candidates, rather calculating the similarity for a constant number of neighbors for each identified object. For infoboxes, lists and tables, the position-rank of the object among all objects of the same type on the same page defines its neighborhood. In particular, we say that a new object instance is in the neighborhood of a previously existing object when the absolute difference between their position-ranks on the page does not exceed a certain threshold. Hence, the first stage candidate generation function is defined as $\lambda_{\theta_{\text{POS}}}(X, O) = \{(\mathbf{x}, o) \mid \mathbf{x} \in X, o \in O, \downarrow_{\text{POS}} (\mathbf{x}, o) \leq \theta_{\text{POS}}\}$. That is, it considers only such pairs, where the position of the new object instance is sufficiently close to the last position of the previous objects. For our experiments, we say that an object is in the neighborhood of a previous object if the absolute difference of their positions is ≤ 2 (Section V-C shows that this is a good choice).

In the *second stage* all remaining object pairs (of previously identified objects and new object versions) are compared using the relatively strict similarity measure $\text{sim}_{\textit{strict}}$ and are again sorted accordingly. The similarity measure is strict to anticipate matches of high quality (preferring precision over recall) and it strictly penalizes larger changes. Both this second stage and the last stage use the default candidate generation function. While this stage therefore has quadratic complexity in theory, in practice (as we have observed) it is very efficient as many objects are already matched during the first stage. Since only objects with sufficient similarity are matched here, additional objects can remain for the third stage.

Finally, the *third stage* uses the relaxed similarity measure $\text{sim}_{\textit{relaxed}}$ to allow larger changes in the objects, in a manner similar to the second stage. At this point, all objects with similar partners according to the stricter similarity measure are already matched. While the previous stages focus on high precision, the last stage optimizes for a higher recall.

4) Illustrative example: Figure 3 shows how the different stages interact. In the first stage, we need to compute similarities for each object (left side) to a maximum of three new object instances (right side) assuming a neighborhood of $\theta_{\text{POS}} = 1$. Only one similarity score fulfills the threshold for stage 1 and we add the edge $(x_0{,}o_0)$ to the identity graph. Note that for calculating each of the similarities past versions of an object also need to be considered. In the second stage, all except for four similarities are already computed and can be reused. In this case, four similarity scores fulfill the (lower) threshold and according to the maximum-weight matching we match $(x_1{,}o_2)$ and $(x_3{,}o_1)$. For the last stage, only two similarity scores remain to be calculated, and both equally fulfill the similarity threshold. Due to the lifetime-tiebreaker $\downarrow_{\text{LT}}$ we match $(x_2{,}o_3)$.

V. Evaluation

In the following, we evaluate our approach in two main dimensions: quality and performance. Prior to this, we provide the most interesting figures about our observations and statistical findings.

A. Basic statistics

To determine the quality of our matching, we have created a gold standard of infobox, list and table matchings by hand. This gold standard as well as the following statistics are based

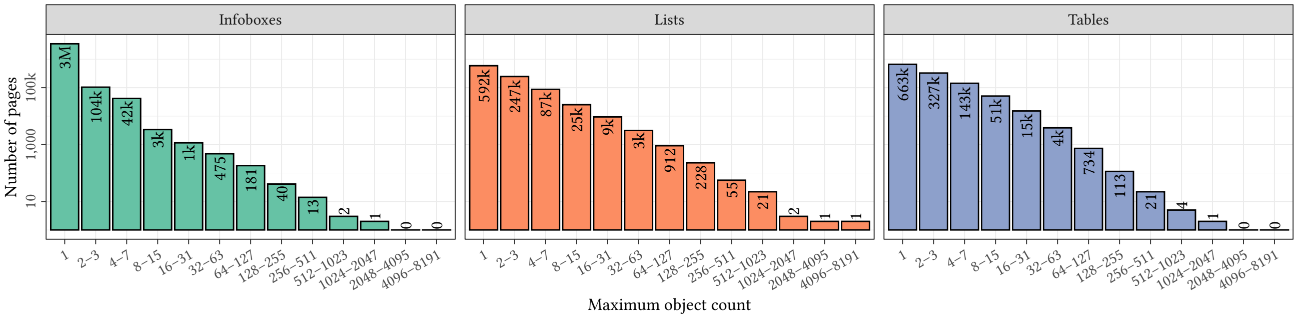


Fig. 4: Histogram of the maximum number of simultaneously existing object instances per page. This is also the minimum number of objects on that page, because each of those object instances must belong to its own object.

on the English Wikidump of 01.09.2019. We furthermore reconstructed and labeled the history for a number of web tables from the Dresden Web Table Corpus (DWTC) [40], which are not from Wikipedia. Since general web tables do not maintain their own history, we used the Internet Archive to retrieve page histories. Both our gold standard and output dataset are available at our project website www.IANVS.org.

As can be seen in Figure 4, the vast majority of web pages contain only a few tables, lists or infoboxes (we omitted the even larger number that do not contain any such objects at all). Because a random sample of pages would, thus, mostly select pages where not much matching is required, we used a stratified sample for our Wikipedia gold standard. As can also be seen in Figure 4, most of the lists and tables (but not infoboxes) exist on pages with more than one list or table on that page. Out of the pages that contain at least one table at any point in time, we selected 15 random pages that never contain more than one table, another 15 random pages that at some point contain more than one but always fewer than four tables, and so on up to 15 random pages with up to 64 tables. On these selected 90 pages we identified 1,445 tables, with a total of 16,919 distinct table versions. We repeated this process for infoboxes, where we again selected 90 pages for which we identified 812 infoboxes, with a total of 9,747 distinct infobox versions, and finally again 90 pages for lists, where we identified 1,648 lists, with a total of 16,919 distinct list versions. For each version of each object of each of these object types we manually created all correct matches. Unless otherwise mentioned, the results in this section are based on this Wikipedia gold standard.

The average object in our gold standard is re-inserted 1.78 times, deleted 2.28 times, and updated 10.33 times. Of the 1.78 re-inserts, 0.10 are fresh objects, i.e., the object's content is different from any previous version, which means 1.68 of the inserts restore previously existing object versions that were deleted at some prior point in time. For the 10.33 updates, the ratio of fresh and old versions is 8.82 fresh versus 1.51 updates that restore previously existing versions. While the vast majority of objects is never deleted or deleted only once, there is a larger skew in the distribution of deletes. One table about Melbourne's climate was deleted 38 times during its lifetime, an infobox about Leprosy 74 times, and list on Archimedes 110 times.

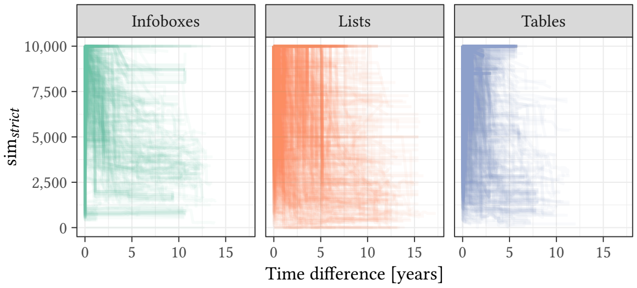
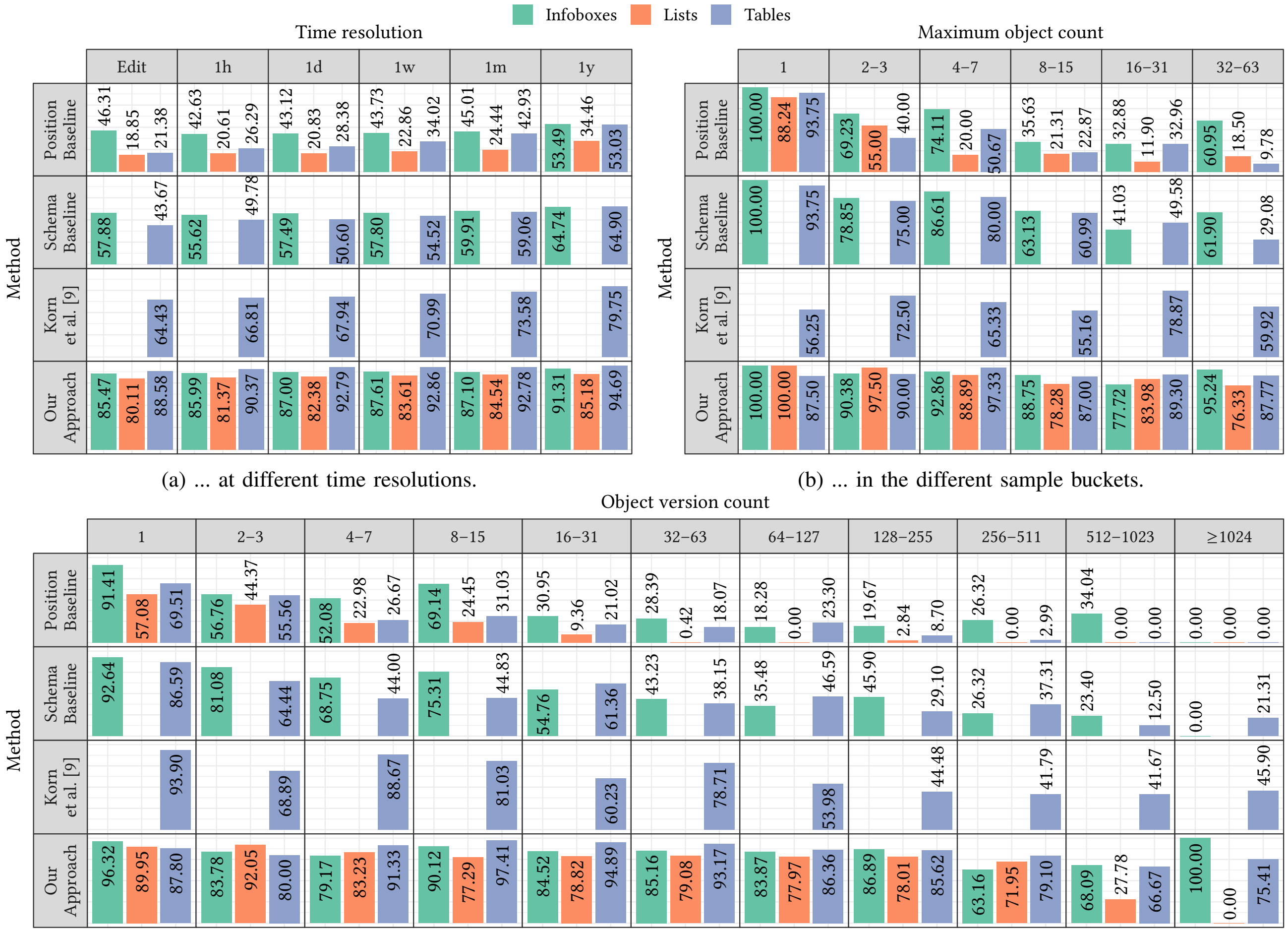


Fig. 5: Similarity sim_{strict} for each object version and the object's first version. Each line represents one object.

From the time an object is created until it is deleted (or until the end-date of the dataset), the average in our gold standard is 3.62 years. In 97.0% of that time, the object is truly part of the page, while in the remaining 38.95 days the object is (temporarily) deleted. During that time, 21.7% of all tables either grow or shrink in the number of columns, and 30.0% grow or shrink in the number of rows. However, 62.1% of all tables retain their original size throughout their lifetime. Of all lists, 26.6% grow or shrink in the number of items, and 37.1% of all infoboxes grow or shrink in their schema.

While for most page revisions, the objects do not move or move only slightly in their positions, there are page revisions for which infoboxes/lists/tables move by up to 28/45/46 positions, respectively. We observe that objects rather move down on the page (9.8%) than they move up (6.9%). The distribution of object movements in relation to other objects of the same type on the page and especially the number of object versions with the same position as the previous version (83.3%), are already an indicator of how well a baseline that uses only the object position as an identifier can perform.

Figure 5 shows how the contents of all objects in our gold-standard develop over time. More precisely, it shows the sim_{strict} similarity of each object version compared to the first version of that object. In general, the similarity is expected to decrease over time, but it can also rise if the object content



(a) ... at different time resolutions.

(b) ... in the different sample buckets.

(c) ... for different number of object versions.

Fig. 6: Overall matching performance as the accuracy of correctly matched object histories ...

becomes more similar to its original version. While there are some objects that stay almost unchanged throughout their lifetime, other objects rapidly change within the first few days of their existence. One reason for this could be that authors copy & paste other objects as templates and then adjust the content. Another reason is that tables can capture static content (stable real-world facts, such as place-of-birth) and dynamic content (such as current-address).

To validate our approach, we took a random sample of 100 pages that contain at least two tables in the DWTC and retrieved the history for those 32 of the 100 pages that were indexed by the Internet Archive. This validation dataset provides insight on how our approach generalizes to general web tables, which are not from Wikipedia. Furthermore, we used a second validation dataset from the domain of open data lakes to show that our approach generalizes beyond web objects. This consists of 2,722 datasets that were published on Socrata (https://dev.socrata.com/data/) in the subdomains of Chicago and Utah and for which we tracked any changes over the course of the last year.

B. Matching quality

In the following, we report results about the quality of our identity graph construction for tables, infoboxes and lists. In Figures 6a–6c, we report the accuracy as the fraction of objects, for which every version was correctly matched. There are gaps in the plot where methods do not apply to specific object types. Figure 6a gives an overview of the overall performance of our approach in comparison to two baselines and a related work approach. The *position baseline* matches two objects (infoboxes, lists, tables), if they appear in the same position on the same page (they have the same position-rank on that page). The *schema baseline* matches infoboxes and tables based on their schema using a single threshold for sim_{strict} in combination with our tiebreakers. We also implemented the related approach by Korn et al. [9] using TableMiner+ [8] to detect subject columns, which are a prerequisite of that approach. Lists do not have a schema, so the schema baseline does not apply to them, just like Korn et al. [9] on any other object type but tables. Figure 6a shows the results for a matching not only when every single edit

TABLE I: Matching results for both baselines, related work and our approach on two validation datasets: DWTC web table and Socrata open data lake.

		Object Accuracy	Precision	Recall	F_1
DWTC	Position baseline	38.5 %	66.4 %	75.6 %	70.7 %
	Schema baseline	76.1 %	88.4 %	93.1 %	90.7 %
	Korn et al. [9]	60.6 %	85.8 %	88.4 %	87.1 %
	Our approach	**89.9 %**	**92.0 %**	**92.4 %**	**92.2 %**
Socrata	Schema baseline	94.7 %	99.3 %	99.3 %	99.3 %
	Korn et al. [9]	96.1 %	99.5 %	99.1 %	99.3 %
	Our approach	**98.2 %**	**99.7 %**	**99.6 %**	**99.7 %**

is available (first column), but also for lower time resolutions, simulating how the matching behaves if the page versions were captured only at certain frequencies (e.g., once a month). As will be described later, with low temporal resolution (larger intervals between snapshots) the individual matching steps become more difficult. At the same time, there are fewer matches necessary, so the overall result is typically better.

For the 725 web tables from DWTC (Table I), the positional baseline performs significantly worse, even though the random, non-stratified sample should even favor that baseline due to the higher probability of smaller pages on which the tables can naturally move less. The lower time resolutions of the Internet Archive crawls explain at least parts of the worse results for all three approaches. Our approach still performs best on this corpus. Table I also shows the result on the Socrata dataset. For this experiment, we hide the stable ID that Socrata provides for each dataset and measured how well the approaches can reconstruct the correct matching. Each subdomain (Chicago, Utah) serves as one context, but as there is no order available, we disabled all spatial features and could not compare against a positional baseline. Overall the results are very good for all approaches. This is not too surprising because the tables are much larger and contain more evidence for matches, so the dataset is less difficult. For some datasets, like the Socrata dataset, our positional features might not be applicable. Therefore, we also ran a experiment for which we disabled all spatial features and compared the overall matching performance for each object type at edit level. The matching performance decreased by 1.3% for lists and tables and only by 0.1% for infoboxes. This is not too surprising, as the positional features mostly act as tie-breakers. Nevertheless, they allow significant runtime improvements as shown in Section V-D.

Like Figure 6a, Figures 6b and 6c show the matching accuracy as the fraction of correctly matched objects. Figure 6b shows how the different matching approaches perform in the strata that we selected in our sampling. Especially the baselines suffer with a higher object count, because there is a much higher chance for object movements and similar schemata on one page. Note, that even an object with only one version can be matched incorrectly, if this one version is not recognized as an individual object (either matched to a previously existing object or to more object versions). Figure 6c confirms the expected behavior that more object versions make it harder to match every version correctly. In this case, the number of versions is not artificially controlled as in the time resolution experiment, but naturally, objects are edited with a different frequency.

TABLE II: Infobox, list, and table matching results for baselines, related work and our approach at different time resolutions.

	Time Resolution	Edit	1h	1d	1w	1m	1y
		Infobox matching					
Position baseline	Precision [%]	86.2	89.4	87.3	85.7	85.1	83.3
	Recall [%]	86.5	89.7	87.7	86.2	85.8	85.2
	F_1 [%]	86.4	89.5	87.5	85.9	85.4	84.2
Schema baseline	Precision [%]	90.2	92.5	91.2	90.1	88.9	87.1
	Recall [%]	90.4	92.8	91.5	90.5	89.4	88.4
	F_1 [%]	90.3	92.6	91.3	90.3	89.2	87.8
Our approach	Precision [%]	**99.4**	**99.5**	**99.4**	**99.3**	**99.0**	**97.9**
	Recall [%]	**99.6**	**99.7**	**99.7**	**99.7**	**99.6**	**99.1**
	F_1 [%]	**99.5**	**99.6**	**99.6**	**99.5**	**99.3**	**98.5**
		List matching					
Position baseline	Precision [%]	75.7	78.2	79.6	79.6	78.4	64.0
	Recall [%]	76.5	79.2	80.7	80.8	79.9	67.3
	F_1 [%]	76.1	78.7	80.1	80.2	79.1	65.6
Our approach	Precision [%]	**98.6**	**98.7**	**98.8**	**98.8**	**98.5**	**96.4**
	Recall [%]	**98.7**	**98.8**	**99.0**	**98.9**	**98.7**	**96.7**
	F_1 [%]	**98.6**	**98.7**	**98.9**	**98.8**	**98.6**	**96.6**
		Table matching					
Position baseline	Precision [%]	87.1	91.0	90.9	91.1	89.8	80.8
	Recall [%]	87.5	91.4	91.3	91.6	90.4	82.4
	F_1 [%]	87.3	91.2	91.1	91.3	90.1	81.6
Schema baseline	Precision [%]	92.3	94.3	94.1	94.2	92.8	86.3
	Recall [%]	92.4	94.4	94.2	94.3	92.9	86.8
	F_1 [%]	92.3	94.4	94.2	94.2	92.9	86.5
Korn et al. [9]	Precision [%]	95.5	96.9	97.5	98.3	98.4	97.9
	Recall [%]	91.8	94.1	94.4	94.7	93.7	89.3
	F_1 [%]	93.6	95.5	96.0	96.5	96.0	93.4
Our approach	Precision [%]	**99.5**	**99.6**	**99.6**	**99.5**	**99.3**	**98.6**
	Recall [%]	**99.6**	**99.7**	**99.7**	**99.7**	**99.5**	**99.2**
	F_1 [%]	**99.6**	**99.6**	**99.7**	**99.6**	**99.4**	**98.9**

Table II shows the results for precision, recall and F-measure of individual edges in the identity graph for the infobox, list and table matching. Here, precision is the fraction of the output edges that are correct, i.e., appear in the gold standard. Recall is the fraction of correct edges that appear in the output. F-measure is the harmonic mean of precision and recall. For our evaluation, we consider only non-trivial edges: We call a matching between two object versions of two consecutive page versions *trivial*, if (i) the number of

TABLE III: Error analysis for the identity matching.

Infoboxes		**Our matching** TP/TN	FN	FP	FP∧FN	∑
Position baseline	TP/TN	18,634	26	20	40	18,720
	FN	113	2	0	7	122
	FP	134	0	60	0	194
	FP∧FN	2,690	3	0	4	2,697
	∑	21,571	31	80	51	21,733

Lists		**Our matching** TP/TN	FN	FP	FP∧FN	∑
Position baseline	TP/TN	33,090	55	16	276	33,437
	FN	447	6	0	14	467
	FP	777	0	136	0	913
	FP∧FN	9,376	62	0	159	9,597
	∑	43,690	123	152	449	44,414

Tables		**Our matching** TP/TN	FN	FP	FP∧FN	∑
Position baseline	TP/TN	35,438	19	14	52	35,523
	FN	337	3	0	0	340
	FP	437	0	82	0	519
	FP∧FN	4,557	9	0	57	4,623
	∑	40,769	31	96	109	41,005

objects in the two page versions stays almost constant (at most one object added/deleted), (ii) all or all except for one objects have the same content and same context, and (iii) the object content and context are unchanged. Our approach can provide an almost perfect matching if every single edit is available. Lower time resolutions initially have only a minor influence on all four solutions. However, starting with a resolution of one year, it becomes clear that the problem becomes much more difficult and all solutions make more mistakes.

We provide a more detailed error analysis by considering different cases that might appear when we compare the gold standard to our matching output. We distinguish three error types, concentrating on an individual object version and its predecessor in the gold standard and the output. If the object version has a predecessor only in the gold standard, we are missing an edge, so this is a false negative. On the other side, if the object version has a predecessor only in the output, this is a false positive. If the object version has a predecessor in both the output and the gold standard, but they are different, then this is a wrong match, resulting in both a false positive and a false negative. Table III uses this error classification to provide a more detailed comparison of our approach against the position baseline. Our matching drastically reduces the absolute number of every error type (between factor 2 for infobox FP and factor 53 for infobox FP∧FN). Furthermore, the table shows that for most cases where both our matching and the position baseline output a wrong result, the error types remain the same. Still, there are a total of 86 cases for infoboxes, 347 cases for lists, and 85 cases for tables, where the position baseline is correct, but our matching is wrong.

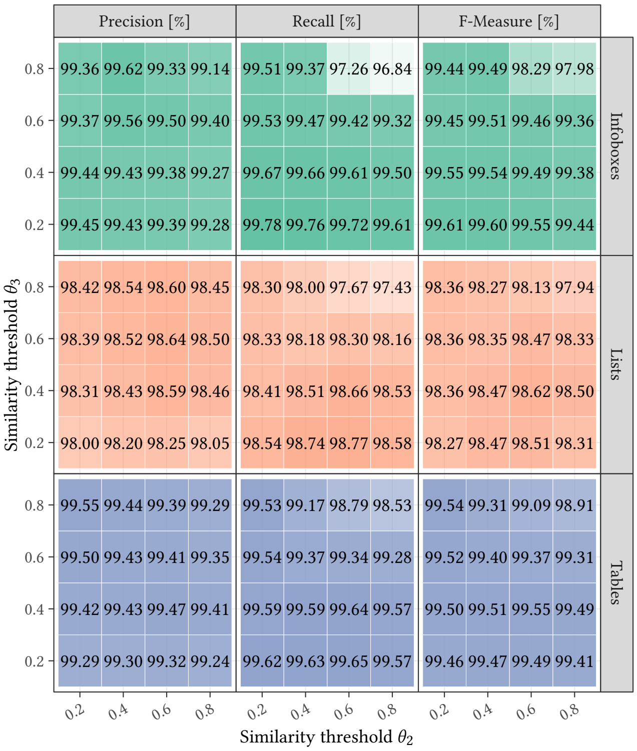


Fig. 7: Effect of different similarity threshold on precision, recall and F-measure for Wikipedia infoboxes, lists and tables.

C. Parameter choice

In this research area, matching approaches typically have many parameters for their similarity measures and candidate set selection strategies. While our approach is no exception, we justify our choice of parameter values with the following experiments, which show that our approach is highly robust, and most parameters can influence only runtime performance.

Figure 7 shows the effect of different similarity thresholds on precision, recall, and F-measure for the three considered object types. While the overall influence of these parameters is low, higher thresholds generally result, as expected, in lower recall and higher precision. The best overall F-measure is achieved at $\theta_2 = 0.6$ and $\theta_3 = 0.4$, which we used for all experiments.

Figure 8 shows the distribution of the number of object candidates that have a very high similarity depending on the maximum position difference between them. Most objects that have a very high similarity also have a very low difference in position. Once a maximum position difference of 2 is exceeded, the number of candidates grows only very slowly. On the other hand, the gold standard reveals that candidates with a high similarity and also a high difference in position are mostly non-matches. Thus, we allow a maximum position difference of only $\theta_{\text{POS}} = 2$ in our first matching phase.

Figure 9 shows the number of candidates that are additionally considered, when not only the most recent object version

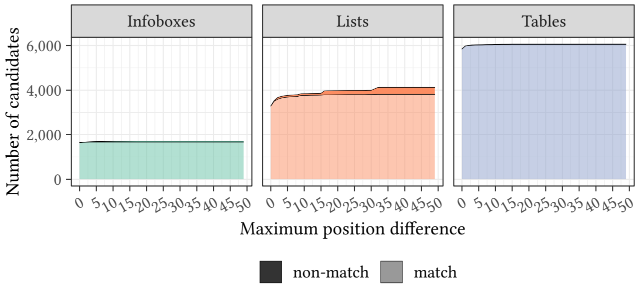


Fig. 8: Total number of object candidates for which $\mathrm{sim}_{strict} >= 0.99$ for different limits of absolute position difference θ_{POS}.

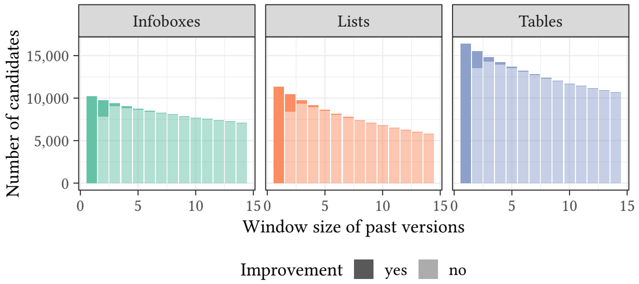


Fig. 9: Number of new candidates for different window sizes k of previous object versions that improve or not improve previous sim_{strict}.

is considered, but also a number k of past versions. While for smaller window sizes it is worthwhile to keep past versions, larger window sizes increase the number of candidates for which the similarity has to be calculated, but do not contribute to higher similarity scores. Hence, we suggest to keep k small and we use $k = 5$ in our experiments.

Finally, Figure 10 shows the effect of weighting similarity scores by an inverse object frequency. While for object pairs that should be matched the similarity mostly stays almost the same, the similarity of all other pairs drops significantly. This shows that the weighted similarity is the better similarity measure here, because it allows an easier distinction between matches and non-matches.

D. Runtime performance

For the following experiments, we report results only for tables, but as previously seen, the different object types behave quite similarly and tables are generally the most complex objects and, thus, require the most matching time. Figure 11a shows the general distributions of table matching runtimes for all pages in our gold standard. We can see that the matching steps take a short time, even without the first matching stage. However, the first matching stage clearly improves the runtime, especially on the long tail of matching times. The median runtime per page version drops from 6.2ms to 4.2ms and the 90th percentile drops from 55.7ms to 11.9ms. The total runtime of all matchings for the gold standard tables drops from 5.2 minutes to 1.6 minutes.

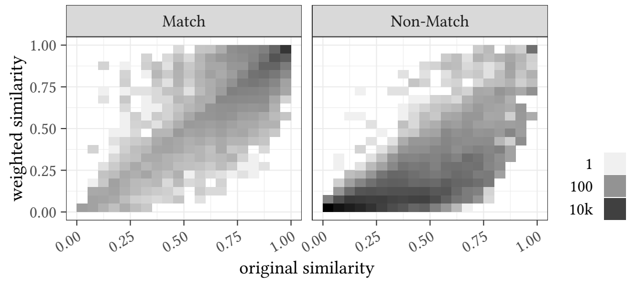
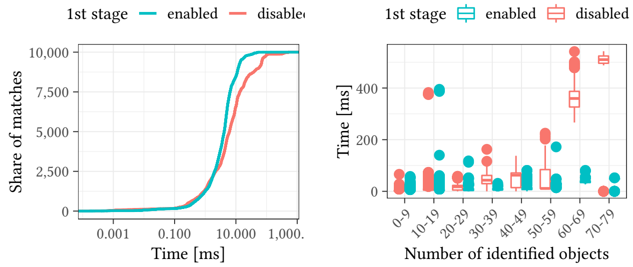


Fig. 10: Effect of weighting similarity scores by an inverse object frequency. While for true matches the weighting has little effect, the similarity scores for non-matches drop significantly.



(a) Cumulative distribution of matching runtimes. The x-axis uses a logarithmic scale.

(b) Box and whiskers plot of the distributions of matching runtimes.

Fig. 11: Effect of first stage on matching runtimes.

Figure 11b shows the effect of the first matching stage in more detail: especially for pages that contain more tables, and thus higher candidate counts, the first matching stage dramatically improves efficiency. Instead of a quadratic scaling in the number of tables, we observe a more linear behavior.

E. Case study: Natural key discovery

In prior work, we were able to directly benefit from the knowledge about a table's version history. When discovering natural keys in Wikipedia tables [10], the object matching approach discussed here gave us access to the tables' version histories, which we used to engineer temporal features. These features exploit the fact that keys are usually static in nature and do not change as much over time as other columns do, and some key properties need to hold on all versions. A single snapshot of a table cannot reveal this knowledge about the dynamic behavior of columns. For example, a column might be unique in the current snapshot but previously contained many duplicates, effectively ruling it out as a key. Our experiments showed that these temporal features raised the F-measure of our classifier by 4.5 percentage points on average [10], showing that knowledge about a table's version history is indeed valuable for downstream tasks.

VI. Conclusions

We have presented an approach that can track objects on different versions of web pages over time. This matching allows us to identify changes locally and calculate meaningful differences between objects. We empirically demonstrated that our approach is applicable to various object types, in particular tables, lists, and infoboxes. It shows significant improvements over baselines and related work.

As future work, we plan to classify the extracted changes to better understand and explore the large number of changes. For example, we want to distinguish between changes that affect the presentation of data and those that reflect a change in the actual semantics of the data value. Another classification could distinguish between meaningful or correct changes on the one hand and destructive changes, such as vandalism or edit wars, on the other. Furthermore, future work could apply the presented methods to other structured and semi-structured elements of evolving corpora.

References

[1] S. Bykau, F. Korn, D. Srivastava, and Y. Velegrakis, "Fine-grained controversy detection in Wikipedia," in *Proceedings of the International Conference on Data Engineering (ICDE)*, 2015, pp. 1573–1584.

[2] M. Potthast, B. Stein, and R. Gerling, "Automatic vandalism detection in Wikipedia," in *Proceedings of the European Conference on Information Retrieval (ECIR)*, 2008, pp. 663–668.

[3] T. Bleifuß, L. Bornemann, T. Johnson, D. V. Kalashnikov, F. Naumann, and D. Srivastava, "Exploring change – a new dimension of data analytics," *PVLDB*, vol. 12, no. 2, pp. 85–98, 2018.

[4] Plutarch, *Theseus*, ser. Plutarch's lives, vol. 1, english translation by Bernadotte Perrin. 1914.

[5] M. Cafarella, A. Halevy, H. Lee, J. Madhavan, C. Yu, D. Z. Wang, and E. Wu, "Ten years of webtables," *PVLDB*, vol. 11, no. 12, pp. 2140–2149, 2018.

[6] B. Hancock, H. Lee, and C. Yu, "Generating titles for web tables," in *Proceedings of the International World Wide Web Conference (WWW)*. ACM, 2019, pp. 638–647.

[7] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu, "Understanding tables on the web," in *Proceedings of the International Conference on Conceptual Modeling (ER)*. Springer, 2012, pp. 141–155.

[8] Z. Zhang, "Effective and efficient semantic table interpretation using TableMiner+," *Semantic Web*, vol. 8, no. 6, pp. 921–957, 2017.

[9] F. Korn, X. Wang, Y. Wu, and C. Yu, "Automatically generating interesting facts from Wikipedia tables," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2019, pp. 349–361.

[10] L. Bornemann, T. Bleifuß, D. V. Kalashnikov, F. Naumann, and D. Srivastava, "Natural key discovery in Wikipedia tables," in *Proceedings of the International World Wide Web Conference (WWW)*, 2020, pp. 2789–2795.

[11] O. Hassanzadeh, K. Q. Pu, S. H. Yeganeh, R. J. Miller, L. Popa, M. A. Hernández, and H. Ho, "Discovering linkage points over web data," *PVLDB*, vol. 6, no. 6, pp. 445–456, 2013.

[12] P. Li, X. L. Dong, A. Maurino, and D. Srivastava, "Linking temporal records," *PVLDB*, vol. 4, no. 11, pp. 956–967, 2011.

[13] G. Limaye, S. Sarawagi, and S. Chakrabarti, "Annotating and searching web tables using entities, types and relationships," *PVLDB*, vol. 3, no. 1, pp. 1338–1347, 2010.

[14] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *PVLDB*, vol. 5, no. 10, pp. 908–919, Jun. 2012.

[15] S. Goldberg, T. Milo, S. Novgorodov, and K. Razmadze, "WiClean: A system for fixing wikipedia interlinks using revision history patterns," *PVLDB*, vol. 12, no. 12, pp. 1846–1849, 2019.

[16] T. Pellissier Tanon, C. Bourgaux, and F. Suchanek, "Learning how to correct a knowledge base from the edit history," in *Proceedings of the International World Wide Web Conference (WWW)*, 2019, pp. 1465—-1475.

[17] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "WebTables: exploring the power of tables on the web," *PVLDB*, vol. 1, no. 1, pp. 538–549, 2008.

[18] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *Proceedings of the International Conference Companion on World Wide Web*, 2016, pp. 75–76.

[19] E. Alfonseca, G. Garrido, J. Delort, and A. Peñas, "WHAD: Wikipedia historical attributes data - historical structured data extraction and vandalism detection from the Wikipedia edit history," *Language Resources and Evaluation*, vol. 47, no. 4, pp. 1163–1190, 2013.

[20] X. Ling, A. Y. Halevy, F. Wu, and C. Yu, "Synthesizing union tables from the web," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013, pp. 2677–2683.

[21] F. Nargesian, E. Zhu, K. Q. Pu, and R. J. Miller, "Table union search on open data," *PVLDB*, vol. 11, no. 7, pp. 813–825, 2018.

[22] O. Lehmberg and C. Bizer, "Stitching web tables for improving matching quality," *PVLDB*, vol. 10, no. 11, pp. 1502–1513, 2017.

[23] Z. Bellahsene, A. Bonifati, and E. Rahm, *Schema Matching and Mapping*. Berlin, Heidelberg: Springer, 2011.

[24] J. Kang and J. F. Naughton, "On schema matching with opaque column names and data values," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2003, pp. 205–216.

[25] A. Bilke and F. Naumann, "Schema matching using duplicates," in *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005, pp. 69–80.

[26] N. Koudas, S. Sarawagi, and D. Srivastava, "Record linkage: similarity measures and algorithms," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2006, pp. 802–803.

[27] Y.-H. Chiang, A. Doan, and J. F. Naughton, "Tracking entities in the dynamic world: A fast algorithm for matching temporal records," *PVLDB*, vol. 7, no. 6, pp. 469–480, 2014.

[28] ——, "Modeling entity evolution for temporal record matching," in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2014, pp. 1175–1186.

[29] P. Christen and R. W. Gayler, "Adaptive temporal entity resolution on dynamic databases," in *Advances in Knowledge Discovery and Data Mining (PAKDD)*, 2013, pp. 558–569.

[30] Y. Hu, Q. Wang, D. Vatsalan, and P. Christen, "Improving temporal record linkage using regression classification," in *Advances in Knowledge Discovery and Data Mining (PAKDD)*. Springer, 2017, pp. 561–573.

[31] M. Mesgari, C. Okoli, M. Mehdi, F. Å. Nielsen, and A. Lanamäki, "The sum of all human knowledge: A systematic review of scholarly research on the content of Wikipedia," *Journal of the Association for Information Science and Technology*, vol. 66, no. 2, pp. 219–245, 2015.

[32] A. Ceroni, M. Georgescu, U. Gadiraju, K. D. Naini, and M. Fisichella, "Information evolution in Wikipedia," in *Proceedings of the International Symposium on Open Collaboration (OpenSym)*, 2014, pp. 24:1–24:10.

[33] L. S. Buriol, C. Castillo, D. Donato, S. Leonardi, and S. Millozzi, "Temporal analysis of the wikigraph," in *International Conference on Web Intelligence (WI)*, 2006, pp. 45–51.

[34] N. Kanhabua and W. Nejdl, "On the value of temporal anchor texts in Wikipedia," in *Proceedings of the SIGIR Workshop on Temporal, Social and Spatially-aware Information Access (TAIA)*, 2014.

[35] A. Kittur, B. Suh, B. A. Pendleton, and E. H. Chi, "He says, she says: conflict and coordination in Wikipedia," in *Proceedings of the International Conference on Human Factors in Computing Systems (SIGCHI)*, 2007, pp. 453–462.

[36] M. Georgescu, N. Kanhabua, D. Krause, W. Nejdl, and S. Siersdorfer, "Extracting event-related information from article updates in Wikipedia," in *Advances in Information Retrieval (ECIR)*. Springer, 2013, pp. 254–266.

[37] B. T. Adler, K. Chatterjee, L. De Alfaro, M. Faella, I. Pye, and V. Raman, "Assigning trust to Wikipedia content," in *Proceedings of the International Symposium on Wikis (WikiSym)*, 2008, pp. 26:1–26:12.

[38] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[39] M. M. Deza and E. Deza, "Similarities and distances for numerical data," in *Encyclopedia of distances*. Springer, 2009, pp. 298–301.

[40] J. Eberius, M. Thiele, K. Braunschweig, and W. Lehner, "Top-k entity augmentation using consistent set covering," in *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 2015.