Matching Roles from Temporal Data

Why Joe Biden is not only President, but also Commander-in-Chief

LEON BORNEMANN, Hasso Plattner Institute, University of Potsdam, Germany

TOBIAS BLEIFUSS, Hasso Plattner Institute, University of Potsdam, Germany

DMITRI V. KALASHNIKOV, Unaffiliated, USA

FATEMEH NARGESIAN, University of Rochester, USA

FELIX NAUMANN, Hasso Plattner Institute, University of Potsdam, Germany

DIVESH SRIVASTAVA, AT&T Chief Data Office, USA

We present role matching, a novel, fine-grained integrity constraint on temporal fact data, i.e., ⟨subject, predicate, object, timestamp⟩-quadruples. A *role* is a combination of subject and predicate and can be associated with different objects as the real world evolves and the data changes over time. A *role matching* states that the associated object of two or more roles should always match across time. Once discovered, role matchings can serve as integrity constraints to improve data quality, for instance of structured data in Wikipedia [3]. If violated, role matchings can alert data owners or editors and thus allow them to correct the error. Finding all role matchings is challenging due both to the inherent quadratic complexity of the matching problem and the need to identify true matches based on the possibly short history of the facts observed so far.

To address the first challenge, we introduce several blocking methods both for clean and dirty input data. For the second challenge, the matching stage, we show how the entity resolution method Ditto [27] can be adapted to achieve satisfactory performance for the role matching task. We evaluate our method on datasets from Wikipedia infoboxes, showing that our blocking approaches can achieve 95% recall, while maintaining a reduction ratio of more than 99.99%, even in the presence of dirty data. In the matching stage, we achieve a macro F1-score of 89% on our datasets, using automatically generated labels.

CCS Concepts: • **Information systems** → **Data cleaning**.

Additional Key Words and Phrases: Data Quality, Entity Matching, Blocking, Change Exploration

1 MOTIVATION: ROLES IN FACT DATA

Many data sources can be represented as a set of facts – knowledge graphs are usually even defined as containing sets of facts. Facts are basic pieces of information that assign a specific object to the property of an entity and are commonly represented as RDF-triples. Once such data evolves and version history is tracked, facts can be annotated with a timestamp that shows when they were

Authors' addresses: Leon Bornemann, leon.bornemann@hpi.de, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany; Tobias Bleifuß, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany, tobias.bleifuss@hpi.de; Dmitri V. Kalashnikov, Unaffiliated, USA, dmitri.vk@acm.org; Fatemeh Nargesian, University of Rochester, Rochester, New York, USA, fnargesian@rochester.edu; Felix Naumann, Hasso Plattner Institute, University of Potsdam, Germany, felix.naumann@hpi.de; Divesh Srivastava, AT&T Chief Data Office, USA, divesh@research.att.com.
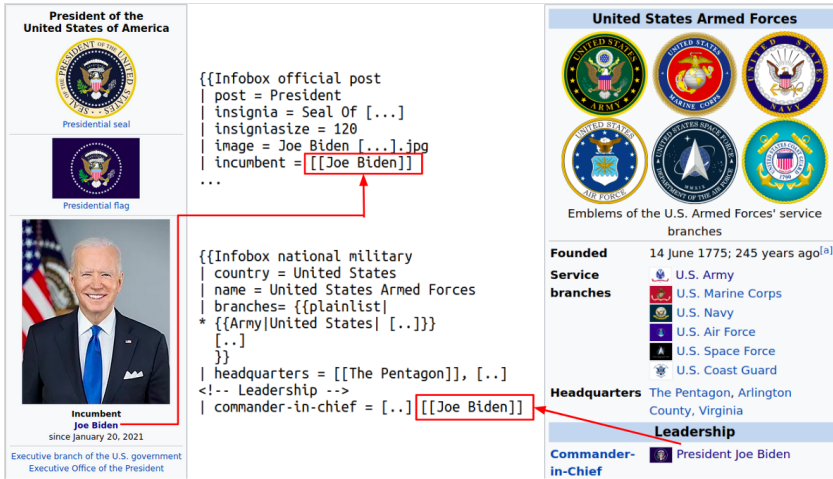
Fig. 1. A role matching in Wikipedia: The president of the United States (currently Joe Biden) is both the incumbent of the role "President of the United States", as well as commander in chief of the US armed forces.

created or updated. It is common to simply append the timestamp as a fourth element, resulting in quadruples [4]. Table 1 presents a few temporal facts that can be extracted from Wikipedia.

When browsing a set of facts, one can observe that many objects appear multiple times. For example, many facts refer to the same important person, such as `Joe Biden`. While many of these recurring mentions are coincidental, there are also those that are the result of real-world constraints. For example, the US Constitution states that the `US-President` is also the `commander-in-chief` of the `US-Armed Forces` [1]. In other words, `Joe Biden` currently fulfills multiple roles, that of the US-president and that of US Armed Forces commander in chief. Intuitively, a role can be defined as the combination of a subject and a predicate. Figure 1 shows how this is reflected in the corresponding Wikipedia infoboxes, a common dataset of facts. The observed equality of the recorded objects (in red boxes) is not coincidental, but is due to the above described constraint, which states that these two roles should always have the same associated object at any given point in time. We call two roles for which such an equality constraint holds a *role matching*.

There are many more role matchings that we can find in Wikipedia pages on education, politics, and sports. For example, we discovered in our experiments that two branches of a particular

Table 1. Temporal facts extracted from Wikipedia

Role		Object	Timestamp
Subject	**Predicate**		
USA	president	Barack Obama	2009-01-20
		Donald Trump	2017-01-20
		Joe Biden	2021-01-20
US-Armed Forces	commander-in-chief	Barack Obama	2009-01-20
		Donald Trump	2017-01-20
		Joe Biden	2021-01-20
...	...	...	...
Jill Biden	spouse	Joe Biden	2002-11-09

university always share the same mascot[1], that the colors of the left and right arm in the uniform of a specific soccer club should always match[2], and that the home states of the two candidates for the 2012 presidential election in the US should be the same in all pages on the elections in the individual states[3]. Knowledge about such role matchings is beneficial for two main use-cases.

Data quality and data cleaning. Role matchings serve as metadata that help capture relationships in the data. They can be used to discover and repair data quality issues in the existing or future data. For example a fact for a matched role might be updated later than its matching partner or it might receive an erroneous update.

Inferring missing data. In many data sources, parts of the data might not be available all the time. This can be due to network or system errors, or temporary deletions. This effect is especially prevalent in Wikipedia data, where many pages are subject to vandalism, or properties in infoboxes are renamed and thus appear to be deleted. Role matchings can provide the missing values and thus complete the historical data.

In our manually annotated gold standards (see Section 7), we observed that 24% of all true role matchings reveal erroneous values and 33% of all true role matchings can infer missing values. However, the automatic discovery of role matchings is not an easy task. Clearly, not every time two roles share the same object at a certain point in time should they continue to do so in the future. For example, `Jill Biden` is currently married to `Joe Biden`, who is also the president of the United States. However, matching these two roles is clearly wrong: Once a new president is elected, that person does not automatically become Jill Biden's spouse. Fortunately, we can use the dynamic nature of the data to our advantage: We know that elements of a true role matching should show a similar temporal behavior.

We formally define the problem of role matching in Section 3, but give an intuition already here: Given a set of roles and their version histories, find all pairs of roles that form a role matching, meaning that these roles should (almost) always have the same value at every point in time. In contrast to the traditional *entity matching* task, which considers static pairs of records, *role matching* must additionally consider role lineages, i.e., the series of insertions, updates, and deletions of facts. The fickle nature of Wikipedia edits, leading to long delays between changes of matching roles, poses an additional obstacle, which we address by specific solutions to both stages of traditional solutions: candidate selection (blocking) and candidate comparison (matching). This paper focuses on solutions to the blocking stage. The goal of the blocking stage is to reduce the number of candidates that need to be considered, while maintaining most of the actual matches in the selected subset.

In this work, we present and experimentally compare several blocking methods for the problem of role matching. Because errors in the data may lead to true role matchings not being retained in the blocking stage, we present blocking methods for both clean and dirty data, where the latter are relaxed variants of the blocking methods that work on clean data. Our blocking methods maintain high recall for true role matchings, while drastically reducing the search space. For the matching stage, we adapt the schema-agnostic matching technique Ditto [27] and show that it outperforms other approaches, achieving a satisfactory performance for the task of role matching. In summary, our contributions are:

- The introduction of the novel problem of role matching.

---

[1] https://en.wikipedia.org/?curid=1373604 and https://en.wikipedia.org/?curid=1443633
[2] https://en.wikipedia.org/?curid=1007441
[3] https://en.wikipedia.org/wiki/2012_United_States_presidential_election_in_Utah, https://en.wikipedia.org/wiki/2012_United_States_presidential_election_in_Florida, https://en.wikipedia.org/wiki/2012_United_States_presidential_election_in_Kansas, ...

- The conception and experimental comparison of various blocking strategies for both clean and dirty input data.
- A proof of concept solution for the matching stage of role matching that adapts Ditto to the task of role matching.
- An experimental evaluation of our solution on five different datasets extracted from Wikipedia infoboxes, including two manually annotated gold standards of 3,000 role matching candidate pairs in total.

In the remainder of the paper, Section 2 discusses related work and Section 3 formally introduces the role matching problem. Next, Sections 4 and 5 introduce blocking methods for clean and dirty data, respectively. Afterwards, Section 6 explains how Ditto [27] can be used to solve the matching stage. Subsequently, we evaluate our methods in Section 7 and conclude in Section 8.

2 RELATED WORK

There are two areas of prior research that are related, namely *data matching* and *truth discovery*.

Data Matching. Data matching, also known as entity matching or duplicate detection, considers matching representations of the same entities. Numerous surveys excellently summarize state of the art [9, 12, 30], in particular also about blocking [31]. Sort-based blocking techniques, such as the sorted neighborhood method [20], are ill-suited for our data, which has many same values. Instead, we focus on standard blocking methods [17], which generate candidate pairs based on sameness of values, but here across multiple timestamps instead of multiple attributes.

Existing techniques for data matching range from score-based methods for matching large knowledge bases [6, 16, 24] to active learning approaches [11, 33] that require only a limited amount of training data, to fully supervised approaches [15, 22, 27]. While these solutions assume static data, there are also approaches that link records in temporal data [7, 8, 35]. Typically, the challenges in temporal record linkage lie in the fact that different snapshots of data are available without knowledge about the underlying changes to the contained entities. A commonly applied strategy is to use a decay function to model the probability that entities have changed over time [21, 25]. Our setting is significantly different: We assume perfect knowledge about which changes occurred at which point in time to which entity. The goal of role matching is not to find multiple representations of the same entity, but instead discover roles that can have quite different subject entities, but that always share an associated object.

Data matching has also been studied in the context of change exploration, where the matching of structured entities across time has been looked at [5]. However, the suggested technique uses a bag of words-based approach, thus requiring that the entities to match are collections of values, which is not the case for roles.

Truth discovery. The research area of truth discovery deals with the problem of finding the true object for a given role from a set of sources that may claim different objects to be true [26]. While there are many solutions to the problem [14, 18, 32, 36], quite a few works deal with specialized variants of the problem, such as incorporating dependencies between different sources [13] or incorporating labelled truths [37]. Particularly related are truth discovery algorithms that consider data that evolve over time [14, 28, 38]. While the underlying use-case of truth discovery – ensuring data quality – is similar to role matching, the setting of truth discovery is significantly different: it assumes and requires several observations for the same fact to be present in the data. Instead of using known redundancies (the presence of multiple observations of the same fact) the task of role matching generally assumes that every fact is present only once in the data and discovers equality constraints between two different roles. Truth discovery could be applied as an upstream task to resolve disagreements before doing role matching.

3 ROLE MATCHING IN TEMPORAL DATA

We first introduce the notation used in this paper, after which we formally introduce the role matching problem.

Preliminaries and Notation. To denote a time-specific fact f (in the following called fact for brevity), we use the RDF-inspired quadruple notation $f = < s, p, o, t >$, where s, p, o correspond to the subject, predicate, and object as in RDF and t is a timestamp when this fact f is valid. We use $\mathcal{F}(S, P, O, T)$ to refer to a relation containing many such facts, with $\{S, P, T\}$ being a key of this relation. We model timestamps as integers and denote the set of all observable timestamps, by $TS = \pi_T(\mathcal{F})$. We call the combination of subject and predicate a *role* (denoted $r = (s, p)$). We define the lineage of a role in the following:

Definition 3.1. Given a role $r = (s, p)$, its *role lineage* L_r is defined as the relation of all object-timestamp pairs that were ever recorded for r: $L_r = \pi_{O,T}(\sigma_{S=s \wedge P=p}(\mathcal{F}))$.

We use $L_r[t]$ to denote the object associated with role r at timestamp t: $L_r[t] = \pi_O(\sigma_{T=t}(L_r))$. Due to our key constraint, this set can contain at most one object, so we overload notation and use $L_r[t]$ to also denote this object. In the following, we also refer to the object associated with a role at a certain timestamp as that role's *value* at that point in time. For simplicity of notation, we assume that $\mathcal{F}$ contains an observation for every role r at every timestamp $t \in TS$ and that the timestamps $t \in TS$ are equidistant. If there is no data about r at point in time t, then we assign $L_r[t] = \bot$ (which is not part of the regular domain of $L_r[t]$) and assume that $\mathcal{F}$ also contains this. We say that there is a *change* to r at timestamp t, if $L_r[t] \neq \bot$ and either the initial insert happened at t ($\forall t' < t \; L_r[t'] = \bot$), or there was an update at timestamp t: $\exists i, i < t \wedge L_r[i] \neq L_r[t] \wedge L_r[i] \neq \bot \wedge \forall j, i < j < t : L_r[j] = \bot$. We do not consider deletions or re-insertions of the previously deleted value to be changes.

We now define two concepts that are used later in the paper, namely change sequences and transition sets. Intuitively, the change sequence of a role r is the sequence of changes to r without considering the explicit timestamps of the changes:

Definition 3.2. The *change sequence CQ* of a role r is defined as $CQ(r) = \pi_O(\tau_T(\sigma_{isChange(T)}(L_r)))$.

For example, given the data shown in Table 1, the change sequence for the role (USA, president) is [Barack Obama, Donald Trump, Joe Biden]. Given the notion of a change sequence, we define the notion of the transition set:

Definition 3.3. Given the change sequence $CQ(r) = [v_1, v_2, ..., v_n]$ of role r, the *transition set TRS* of r is defined as $TRS(r) = \{(v_i \rightarrow v_{i+1}) \mid i \in \{0, ..., n-1\}\}$.

The Role Matching Problem. As mentioned in the introduction, a role matching is an integrity constraint that holds between two roles r_1 and r_2 if they should always have the same value. We can thus define the role matching problem in the following:

Definition 3.4. Given a set of roles R, the *role matching problem* asks for those pairs of roles $(r_1, r_2) \in R \times R$, where r_1 and r_2 form a role matching.

Considering all pairs in $R \times R$ is infeasible in practice, as there are just too many to consider and computational resources are limited. An overview over the different stages for role matching is displayed in Figure 2. As discussed in Section 1, we use the same task split as for entity matching to solve the problem of role matching. We distinguish blocking methods for clean and dirty data, because dirty input data requires more lenient (relaxed) blocking methods. While these are expected to have a higher recall, they also return more candidate pairs, leading to a large number of candidates
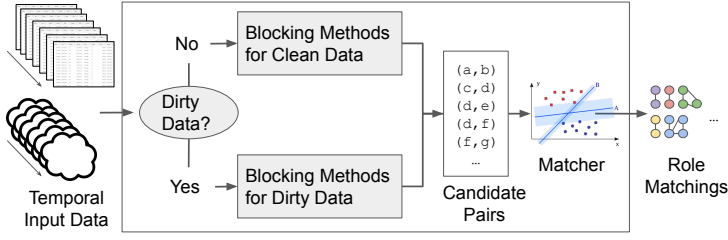
Fig. 2. The role matching pipeline.

that are passed on to the matcher. In the following, we explain how the different subtasks of role matching can be solved. When denoting a blocking method, we always assume that R is the set of input roles and that r_1 and r_2 are two roles from that set. For every blocking method, we describe the set of pairs that are retained in the blocking stage and thus passed on to the matching stage. We now describe how blocking for clean (Section 4) and dirty (Section 5) input data can be realized. Afterwards, we describe how related work from the area of entity matching, namely Ditto [27], can be employed in the matching stage (Section 6).

4 BLOCKING FOR CLEAN DATA

In use cases where we expect clean data, we can apply strict blocking methods, which we present in this section. This applies, for example, to relational databases where automatic checks can ensure the consistency of values and transactions can update several values at the same recorded timestamp. In the following, we first define the blocking methods, after which we discuss their efficient implementation.

4.1 Blocking Method Definitions

Assuming clean data, a first approach is to limit the scope of candidates to those roles whose lineages are equivalent:

Blocking Method 1. *Exact Match* (EM) considers all pairs of roles where the two roles agree on every value at every point in time in the observed time period: $EM(R) = \{(r_1, r_2) \in R \times R \mid \forall t \in TS \, L_{r_1}[t] = L_{r_2}[t]\}$.

However, this blocking technique fails to consider important effects that happen when data is observed over a long period of time. EM demands, that the values of two roles r_1 and r_2 must always match at every point in time. This implicitly assumes that two roles must always exist at the same time because if $(r_1, r_2) \in EM$ then $L_{r_1}[t] = \bot$ implies $L_{r_2}[t] = \bot$. This is highly unrealistic in practice because some data entries may be created earlier than others, data providers may experience server or network issues, or individual parts of the data can be temporally missing. For example, common role matchings on Wikipedia are that books of the same series should share the same genre. However, articles on Wikipedia are not created or updated systematically. For example, the article for the book *Red Sun of Darkover*[4] was inserted on September 23, 2008, whereas the article for *Renunciates of Darkover*[5] was created a month later on October 21, 2008. It is easy to imagine that the same effect would also occur in database systems, as typically only few records are created at exactly the same time.

---

[4]https://en.wikipedia.org/?curid=19433458
[5]https://en.wikipedia.org/?curid=19869298

What follows is that the unobservability of a role r at timestamp t (denoted as $L_r[t] = \bot$) should not prevent matches to other roles that are observable at timestamp t. We also refer to $\bot$ as the wildcard-value. We use that notion to define value compatibility:

Definition 4.1. The values of two roles r_1 and r_2 are *compatible* at timestamp t, denoted $L_{r_1}[t] \widehat{=} L_{r_2}[t]$, if $L_{r_1}[t] = L_{r_2}[t] \lor L_{r_1}[t] = \bot \lor L_{r_2}[t] = \bot$

The idea is intuitive: if the actual true value of a role is unknown at a timestamp t (denoted by $L_r[t] = \bot$), then that role can, in principle, match any other role at t. This definition corresponds to the *open world assumption* that is commonly used in knowledge bases, stating that the absence of a fact cannot be used to assume that such a fact is untrue. Given the notion of value compatibility, we can introduce role compatibility:

Definition 4.2. The *role compatibility RC* is defined as the fraction of time that two roles r_1 and r_2 are compatible:
$$RC(r_1, r_2) = \frac{|\{t \in TS \mid L_{r_1}[t] \widehat{=} L_{r_2}[t]\}|}{|TS|}$$

Given a pair of roles $p = (r_1, r_2)$ we also say that p is a fully compatible pair of roles, if $RC(r_1, r_2) = 1.0$. The idea of the next blocking method is to return the set of all roles that are fully compatible. However, a remaining issue is that if data density (meaning the number of non-wildcard values) is low, candidate pairs may be fully compatible even without sharing a single non-wildcard value. For example, if the only observations of non-wildcard values in L_{r_1} are in the first half of the observable time period and the only observations of non-wildcard values in L_{r_2} are in the second half of the observable time period, then (r_1, r_2) is fully compatible. To filter such candidates, we employ an additional filter that requires that roles r_1 and r_2 share at least one transition:

Blocking Method 2. *Compatibility-Based Role Blocking* (CBRB) considers all pairs of roles that have a compatibility of 1.0 and share at least one transition: $CBRB(R) = \{(r_1, r_2) \in R \times R \mid RC(r_1, r_2) = 1.0 \land TRS(r_1) \cap TRS(r_2) \neq \emptyset\}$.

4.2 Efficient Computation of CBRB

While EM can be implemented via a simple group-by query, the solution for CBRB is more complex. Conventional index-structures, such as hashtables or B^+-trees do not solve the problem, because they are unable to model wildcards matching any other value. Naturally, checking all pairs of roles for their compatibility is infeasible. That is why we present a specialized pruning strategy next.

4.2.1 Constructing Role Trees. It is possible to efficiently discover the set of fully compatible role pairs with the help of a specialized tree-like traversal method, role trees, which function similar to standard search trees, but are able to handle wildcards. To explain how role trees function, we first consider the simpler case of no wildcards being present in the data. Given a set of roles R, and a timestamp t, we partition the roles $r \in R$ according to $L_r[t]$, because only roles that have the same values at t can be fully compatible. Given a timestamp t and a value v, let $P_v^t = \{r \mid L_r[t] = v\}$ denote the partition of roles r that have value v at timestamp t. By recursively splitting partitions (using different timestamps) we obtain a tree of partitions that we call a role tree. Figure 3 illustrates an example. Initially (in the root node) the value at every timestamp is still unknown, and thus all boxes are colored white. Once a split at a timestamp is performed, the values at that particular timestamp are examined in all role lineages and different values are depicted as different colors. Roles can be fully compatible with each other only if they are in the same leaf node.

In theory, a role tree can reach a maximum height of $|TS|$, but many branches will be much smaller, because we need to split nodes only until they contain few enough roles, so that a pairwise comparison is no longer expensive. Note that a role tree is not just a simple prefix-tree, because t is
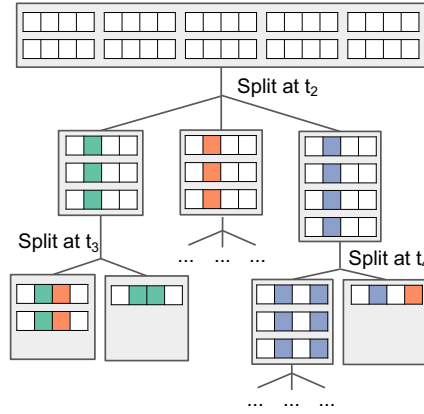
Fig. 3. Visualization of a role tree when no wildcards are present. Role lineages are represented as arrays, where one box depicts one timestamp. Colors represent different values (at specific timestamps).
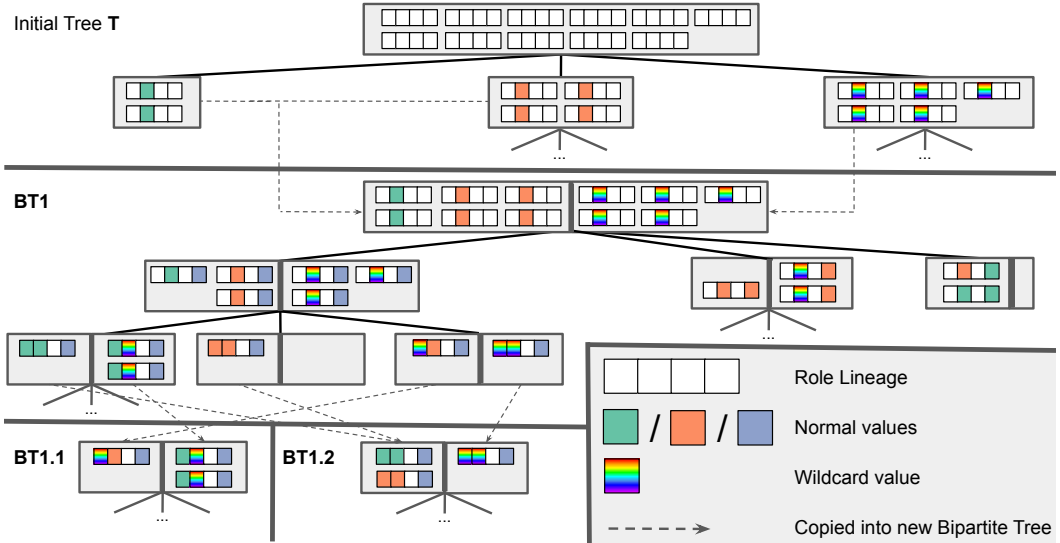
Fig. 4. A role tree to efficiently construct the compatibility graph. Wildcard values are depicted by the rainbow-boxes. Every wildcard partition introduces one or two new bipartite role trees per level, making the overall data structure a forest. Only roles that are in the same leaf-node of a tree can be compatible to each other.

chosen independently in every node. This freedom allows us to select a timestamp that is suitable for splitting the current node into many small partitions. An example can be seen in the second level in Figure 3, where the first node is split on t_3 and the third on t_4. We discuss the mechanism to choose a suitable split-timestamp t in Section 4.2.2.

With wildcards values ("⊥"), the problem of determining whether two roles are fully compatible becomes more complicated, because wildcards are compatible with any other value. Thus, they need to be given special consideration: In any level of the role tree, roles in the wildcard-partition $P_\perp^t$ can still be fully compatible to all other roles in all other partitions at that level. Depending on

the size of $P_\perp^t$, this may result in a prohibitively large number of candidates to check, especially at early levels. A simple solution would be to also add all roles in $P_\perp^t$ to all other partitions on the same level and proceed recursively. However, this is inefficient for two reasons: First, every role in the wildcard partition is now processed m times (where m is the number of partitions). Second, a match between two roles within $P_\perp^t$ is now also found (and thus computed) m times. These problems are amplified once the next level is partitioned, because once again roles in the wildcard partitions are added to every other partition on that level, leading to a tree quickly growing in width, but not shrinking enough in the size of the leaf-nodes.

Our solution to this problem is to process the wildcard partition as if it were a non-wildcard partition and additionally create an instance of a new, specialized traversal method, which we call the bipartite role tree, or bipartite tree in short. A bipartite role tree takes two sets $R, \tilde{R}$ of roles as the input and discovers all fully compatible pairs $(r, \tilde{r})$ with $r \in A, \tilde{r} \in B$. It does *not* consider pairs where both roles are from the same set (this is done in the original, non-bipartite role tree). Whenever there is a level in the original role tree in which there is a wildcard node, our procedure creates a new bipartite role tree with R containing all roles in the non-wildcard partitions (of that level) and $\tilde{R}$ containing the roles in the wildcard partition.

Figure 4 illustrates the creation of bipartite trees. In the example, we start with a normal role tree, which is split on timestamp 2, creating a wildcard-node (on the right-hand side of the figure). This leads to the creation of BT1, whose root node consists of all role lineages in the first level of T, shown by the dashed lines in Figure 4. The root node of BT1 is then split on timestamp 4. Because no role lineage has a wildcard value at timestamp 4, there is no separate bipartite tree created at this level. However, when the left-most level 1 node of BT1 is split on timestamp 1, a new wildcard node is created, leading to the creation of BT2 and BT3. Wildcard nodes within a bipartite tree always spawn two new bipartite trees, because the roles in the left part of the wildcard node need to be compared to all roles in the right part of the non-wildcard partitions (BT2 in Figure 4) and vice-versa (BT3 in Figure 4). Like before, only roles that are in the same leaf-node can be fully compatible.

4.2.2 Choosing suitable split-timestamps. Both regular and bipartite role trees must choose a suitable split-timestamp t for a given input set of roles R or a pair of role sets $(R, \tilde{R})$ in the bipartite case, respectively. Naturally, it is beneficial to choose t in such a way, that the trees quickly converge to small partitions. We can (greedily) model this by comparing the number of pairwise comparisons that would be necessary to determine the set of fully compatible role pairs before and after the split. The larger the difference, the better the split. To facility further notation, let $NP(R)$ denote the number of pairwise combinations of roles in set R:

$$NP(R) = |\{(r_1, r_2) \in R \times R \,|\, r_1 \neq r_2\}| = \frac{|R| \cdot (|R| - 1)}{2} \tag{1}$$

Furthermore, let $V_t = \{L_r[t] \,|\, r \in R \wedge L_r[t] \neq \perp\}$ be the set of non-wildcard values at t. We can describe the number of pairs we need to consider after a split on t for the normal role tree with Equation (2):

$$S_{R,t} = NP(P_\perp^t) + |P_\perp^t| \cdot |R| + \sum_{v \in V_t} NP(P_v^t) \tag{2}$$

The reduction in the number of pairs that need to be considered is described by:

$$RED(R, t) = 1 - \frac{S_{R,t}}{NP(R)} \tag{3}$$

The ideal split-timestamp is the one that minimizes $S_{R,t}$ and thus maximizes $RED(R, t)$. Given two sets of roles R and $\tilde{R}$ in the bipartite case, it is beneficial to separate R and $\tilde{R}$ as much as possible.

Given a timestamp t and a value v, we use P_v^t to denote all $r \in R$ where $L_r[t] = v$ (the left side of the partition). Equivalently $\tilde{P}_v^t$ denotes the right side of the partition (all $\tilde{r} \in \tilde{R}$ where $L_{\tilde{r}}[t] = v$). Here, we can describe the number of comparisons after a split on t with Equation (4):

$$S_{(R,\tilde{R}),t} = |P_\perp^t| \cdot |\tilde{R}| + |R \setminus P_\perp^t| \cdot |\tilde{P}_\perp^t| + \sum_{v \in V_t} (|P_v^t| \cdot |\tilde{P}_v^t|) \tag{4}$$

Here, the reduction in the number of pairs that need to be considered can be described by:

$$RED(R, \tilde{R}, t) = 1 - \frac{S_{(R,\tilde{R}),t}}{|R| \cdot |\tilde{R}|} \tag{5}$$

Once again the ideal split-timestamp minimizes $S_{(R,\tilde{R}),t}$ and thus maximizes $RED(R, \tilde{R}, t)$.

While finding the timestamp that minimizes $S_{R,t}$ or $S_{(\tilde{R},R),t}$ is possible, it has a complexity of $|TS| \cdot |R|$ or $|TS| \cdot (|R| + |\tilde{R}|)$ respectively, because all role lineages in the current partition need to be considered at every candidate timestamp to find it. This would be just as expensive as actually executing every split, and is thus not a viable solution. If the data is evenly distributed across time, simply picking a random timestamp t to split on can work well. However, this is rarely the case in practice, which is why we sample roles from R and timestamps from TS and compute $S_{R,t}$ or $S_{(\tilde{R},R),t}$ on the sample respectively. We then use the t that optimally splits the sample as our split timestamp. In our experiments, we observed that a sampling rate of 10% for fields and timestamps proved effective. Sometimes, a node contains mostly pairwise compatible roles. In such a case, $RED(R, t)$, or $RED(R, \tilde{R}, t)$ respectively, becomes very small. This can lead to cases where the algorithm dedicates a lot of runtime to repeatedly splitting off a very small fraction of R. In such a case, it is usually better to simply check all pairs in the current node instead of further partitioning it. Thus, we abort the partitioning of a role tree node if $RED(R, t)$, or $RED(R, \tilde{R}, t)$ respectively, is below a threshold δ_{minRED}. In our experiments, we discovered that a setting of $\delta_{minRED} = 0.1$ worked well.

4.2.3 CBRB - Algorithm. Algorithm 1 shows the recursive algorithm to construct a (bipartite) role tree, thus creating the blocking for CBRB. Note that the (bipartite) role trees do not need to be explicitly materialized: we use them only as a traversal method, not as a data structure. The procedure *RoleTree* serves as the entry to the program, chooses a split-timestamp t and calls itself recursively for every partition. The subprocedure *BipartiteRoleTree* is used to discover compatible roles in bipartite role trees as described above.

Different recursive calls to *RoleTree* and *BipartiteRoleTree* can easily be parallelized, because they are independent of each other. Thus, we fork a new thread if the new input sizes $|R|$ or $|R| + |\tilde{R}|$ are larger than a certain threshold θ_{fork}. Similarly, the pairwise computations (lines 35-37) are also parallelized if their number exceeds a certain number. For that purpose, we do not actually compute calls to the PAIRWISE procedure in the same process, but instead add them to a list of tasks until a certain batch size θ_{batch} is reached. Once the number of pairs to compute exceeds θ_{batch}, a new asynchronous computation is triggered that executes all accumulated calls to PAIRWISE. In our experiments in Section 7.1 we found that setting $\theta_{fork} = 30$ and $\theta_{batch} = 900$ worked well. Using the settings specified in this section, CBRB computed the blocking for our largest dataset containing more than 300,000 roles in less than half an hour (using 32 threads).

Note that role trees and bipartite role trees can of course also be materialized to efficiently generate candidates for new, previously unseen roles. While it is not a focus of our work, it is worthy to note that (bipartite) role trees can also be built incrementally and are thus also applicable to scenarios where new observations for new timestamps are constantly coming in. In such a case, the leaf-nodes just need to be further split on the new timestamps as the new data arrives.

input : Set of Roles R, timestamps T used by parent-partitions (initially empty), δ_{minRED}
output : All fully compatible role pairs

1 **PROCEDURE** *RoleTree(R, T)*:
2   **if** $T \neq TS \wedge |R \times R|$ *is too large* **then**
3       $t \Leftarrow$ sample best split-timestamp in $TS \setminus T$;   /* see EQ. (2) */
4   **if** $RED(R, t) \geq \theta_{minRED}$ **then**
5          $V_t \Leftarrow \{v \mid L_r[t] = v \ \wedge v \neq \bot, r \in R\}$;   /* all values at t */
6   **for** $v \in V_t$ **do**
7             $P_v \Leftarrow \{r \mid r \in R \wedge L_r[t] = v\}$;   /* current partition */
8   *RoleTree(P_v, T ∪ {t})*;
9          $P_\bot \Leftarrow \{r \mid r \in R \wedge L_r[t] = \bot\}$;   /* wildcard partition */
10   *RoleTree(P_\bot, T ∪ {t})*;
11   *BipartiteRoleTree(P_\bot, R \setminus P_\bot, T ∪ {t})*;
12   **else**
13   *Pairwise(R, R)*;
14   **else**
15   *Pairwise(R, R)*;
16 **PROCEDURE** *BipartiteRoleTree(R, $\tilde{R}$, T)*:
17   **if** $|T| \neq |TS| \wedge |R \times \tilde{R}|$ *is too large* **then**
18       $t \Leftarrow$ sample best split-timestamp in $TS \setminus T$;   /* see EQ (4) */
19   **if** $RED(R, t) \geq \theta_{minRED}$ **then**
20          $V_t \Leftarrow \{v \mid L_r[t] = v \ \wedge v \neq \bot, r \in R \cup \tilde{R}\}$;
21   **for** $v \in V_t$ **do**
22             $P_v \Leftarrow \{r \mid r \in R \wedge L_r[t] = v\}$;
23             $\tilde{P}_v \Leftarrow \{\tilde{r} \mid \tilde{r} \in \tilde{R} \wedge L_{\tilde{r}}[t] = v\}$;
24   *BipartiteRoleTree(P_v, $\tilde{P}_v$, T ∪ {t})*;
25          $P_\bot \Leftarrow \{r \mid r \in R \wedge L_r[t] = \bot\}$;
26          $\tilde{P}_\bot \Leftarrow \{\tilde{r} \mid \tilde{r} \in \tilde{R} \wedge L_{\tilde{r}}[t] = \bot\}$;
27   *BipartiteRoleTree(P_\bot   , $\tilde{R} \setminus \tilde{P}_\bot$, T ∪ {t})*;
28   *BipartiteRoleTree(R \setminus P_\bot, $\tilde{P}_\bot$, T ∪ {t})*;
29   *BipartiteRoleTree(P_\bot   , $\tilde{P}_\bot$, T ∪ {t})*;
30   **else**
31   *Pairwise(R, $\tilde{R}$)*;
32   **else**
33   *Pairwise(R, $\tilde{R}$)*;
34 **PROCEDURE** *Pairwise(R, $\tilde{R}$)*:
35   **for** $(r, \tilde{r}) \in \{(r, \tilde{r}) \in R \times \tilde{R} \mid r \neq \tilde{r} \wedge r \,\hat{=}\, \tilde{r}\}$ **do**
36   **if** $TRS(r) \cap TRS(\tilde{r}) \neq \emptyset$ **then**
37   **output** $(r, \tilde{r})$;   /* fully compatible pair found */

Algorithm 1: CBRB – Compatibility-based Role Blocking

5 BLOCKING FOR DIRTY DATA

While we can expect the prior approach to retain all role matching candidates in clean data, the same is not true if there are data quality issues. In crowd-edited data sources, such as Wikipedia,

Candidate Pair	EM	CBRB	CQM	TSM	VSM	RM^0.8
1	✓	✓	✓	✓	✓	✓
2	✗	✓	✓	✓	✓	✓
3	✗	✗	✓	✓	✓	✓
4	✗	✗	✗	✓	✓	✗
5	✗	✗	✗	✗	✓	✓
6	✗	✗	✗	✗	✗	✓
7	✗	✓	✗	✗	✗	✓

Fig. 5. Examples of which types of candidate pairs are retained by which blocking methods. Like in Figure 3, a rainbow represents the non-existence of an observation, which CBRB interprets as a wildcard. Checkmarks signify that a candidate is retained by a blocking method.

common data quality issues include vandalism, inconsistent formattings, delayed updates, or actual disagreements about what is correct. In the presence of such data quality issues, it is unrealistic to expect to retain most role matchings using CBRB. Thus, we present new blocking methods that further relax the search space. Analogously to Section 4, we first define the blocking methods for dirty data and then discuss how they can be efficiently implemented.

5.1 Blocking Method Definitions

To illustrate the differences between the blocking methods that we already presented in Section 4 and the ones that we present in the following, Figure 5 shows what example candidate pairs are retained by which blocking method. We first present a relaxation to CBRB by introducing value decay and subsequently discuss several relaxations to EM.

5.1.1 CBRB with Decaying Values. The first relaxation that we consider is the notion of value decay. This relaxation can be used with CBRB and deals with the issue that data items can become outdated and erroneously show outdated values. Treating such outdated values as the true current value can prevent CBRB from finding semantically correct role matchings, especially if the usual update frequency for a role is low. This can be remedied, by transforming the underlying data using a monotonically decreasing decay function ω:

$$L_r[t]' = \begin{cases} \omega(r, t, t') > 0 : & L_r[t] \\ otherwise : & \bot \end{cases} \tag{6}$$

Here, we denote r as the role, t as the current timestamp, and t' as the timestamp of the last change to r before t. The reasoning is that the longer there is no change to a role, the less certain we are about the correctness of the current value, which is modelled by a decrease in the value returned by ω. If the result reaches zero, the recorded value is deemed too uncertain and replaced with a wildcard, which is allowed to match any other value.

Naturally, some roles change their values more frequently than others and thus their values should decay more quickly. A natural, data-driven way to model the decay function is to use the distribution of time intervals between observed consecutive changes of a role. Let $T_r = [t_1, ..., t_k]$ be the sorted sequence of timestamps at which r changed. Further, let $time(t_i, t_j)$ denote the absolute

time difference between timestamps t_i and t_j. We denote $TI_r = \{time(t_i, t_{i+1}) \mid i \in \{1, \ldots, k-1\}\}$ as the multiset of time differences from one change to the next. The probability that the value of r has decayed at timestamp t (with the timestamp of the last change to r before t being t') is described by the cumulative distribution function:

$$C_r(t', t) = \frac{|\{d \in TI_r \mid d \le time(t', t)\}|}{|TI_r|} \tag{7}$$

In order to turn this probability into a deterministic binary decision, we use a threshold $\beta \in [0, 1]$ in the decay function $\omega(r, t, t') = \beta - C_r(t', t)$. The larger β, the more certain we want to be that the current value of a role r is outdated in order to actually flag it as decayed (and replace it with a wildcard). Using the notion of decay, we can introduce a relaxed form of CBRB:

Blocking Method 3. *Compatibility-Based Role Blocking with Decay* (CBRB$_\beta$) functions in the same way as CBRB, with the difference that before the blocking is executed, the lineage of every role $r \in R$ is transformed as described in Equation 6 using our decay function ω with β as the threshold.

5.1.2 Relaxations to EM. While applying a decay function can deal with cases where a few values in a lineage are outdated for a longer period of time, it does not help if a role is updated regularly, but with delays. For such cases, it makes sense to merely demand that that changes (new value assignments) to the two roles happen in the same order, without considering the exact timestamps. This is realized by blocking by the change sequence:

Blocking Method 4. *Change Sequence Match* (CQM) considers all pairs of roles, where the change sequences (see Definition 3.2) of the two candidate roles are equal: $CQM(R) = \{(r_1, r_2) \in R \times R \mid CQ(r_1) = CQ(r_2)\}$.

While CQM tolerates changes that happen out of sync, it still requires all changes to happen in the same order, which can still be too strict, for example in the presence of edit wars. A blocking method that keeps the notion of a change, but relaxes their order is the idea of blocking by transition sets:

Blocking Method 5. *Transition Set Match* (TSM) considers all pairs of roles where the transition set (see Definition 3.3) of r_1 is equivalent to the transition set of r_2: $TSM(r) = \{(r_1, r_2) \in R \times R \mid TRS(r_1) = TRS(r_2)\}$.

While TSM is more lenient than CQM, it discards candidate pairs where one of the roles underwent an erroneous revert. By demanding only value set equality, without considering any order in which the values were assigned to the roles, such cases can be covered:

Blocking Method 6. *Value Set Match* (VSM) considers all pairs of roles where the set of all values that were assigned to r_1 is equivalent to the set of all values that were assigned to r_2: $VSM(R) = \{(r_1, r_2) \in R \times R \mid \Pi_O(L_{r_1}) = \Pi_O(L_{r_2})\}$

While these methods relax the search space to different degrees, all of them require the roles to fully match on a specific grouping key and only relax how strictly that grouping key is constructed. For some data errors, it is necessary to instead relax the *extent* to which roles match. For example, none of the above methods would retain a candidate pair, where one role assumes a value that the other one does not. Such cases can be covered by what we call the relaxed match method (RM), which is a relaxation of EM:

Blocking Method 7. *Relaxed Match* (RM$^\gamma$) considers all pairs of roles where the two roles have the same value for a relative number of timestamps of at least γ:
$RM^\gamma(R) = \{(r_1, r_2) \in R \times R \mid \frac{|\{t \in TS \mid L_{r_1}[t] = L_{r_2}[t]\}|}{|TS|} \ge \gamma\}$

For this blocking strategy, γ serves as a parameter to tune the tradeoff between recall on the one hand and false positive rate as well as runtime on the other. Such a parametrized relaxation could also be applied to CBRB, but in our experiments we found that a relaxed notion of CBRB is not superior to RM.

For most of the blocking methods for dirty data, it is simple to find an efficient implementation. $CBRB_\beta$ requires no additional effort, as it just applies a preprocessing step, before the algorithm for CBRB is applied (as detailed in Section 4.2). CQM, TSM and VSM can be implemented by a simple GROUP-BY query. However, RM^γ is more complicated to efficiently implement because it no longer requires an exact match on a grouping key, which is why we discuss its implementation in the following.

5.2 Efficient Computation of RM^γ

To efficiently implement RM^γ, we use prior work on the problem of domain search. Zhu et al. define the problem as [39]: Given a collection of domains (sets of values) D, a query domain Q, and a threshold $\gamma \in [0, 1]$, return the set of domains $D' \subseteq D$ that contain at least $100 * \gamma\%$ of the values in Q. For this purpose, set containment is defined as: $SC(Q, X) = |X \cap Q|/|Q|$. Thus, D' is defined as: $D' = \{X \in D | SC(Q, X) \geq \gamma\}$. The authors present LSHEnsemble, which uses an ensemble of MinHash functions to create data sketches that can be efficiently indexed and queried. It can be used to implement the blocking method RM^γ in the following way: For every role r, we create a set of string values D_r that we call the role domain of r. D_r is created by taking every observable timestamp t, and concatenating t with $L_r[t]$, thus creating a string that represents exactly the value of r at that timestamp: $D_r = \{\text{``}t.v\text{''}|t \in TS \land L_r[t] = v\}$

We then treat every D_r as a domain, and solve the problem of RM^γ by querying every D_r (with threshold γ) against the set of all role domains. By concatenating timestamp and value, we ensure that multiple occurrences of the same value are recognized as a match by LSHEnsemble only if they also appear at the same timestamp, which is precisely what we need for RM^γ. The returned set of role domains by LSHEnsemble corresponds exactly to those roles that match the query role r for at least $100 * \gamma\%$.

A weakness of this method is that it is necessary to materialize a string for every role for every observable timestamp $t \in TS$. This can become prohibitively expensive if TS is large, which is the case if the data were observed over a long period of time and the granularity of the recorded timestamps t is small. We experimentally show this disadvantage in Section 7.3.5.

6 SOLVING THE MATCHING STAGE

While the focus of this paper is dedicated to the blocking stage to find eligible candidates, the problem of identifying true role matchings among these candidates must also be solved. On a technical level, the problem of role matching is equivalent to that of entity matching, but instead of multiple attributes that describe an entity, we have only a single attribute, for which we observe multiple values over time.

We comparatively evaluate three different approaches. The first is a modification of the matching process of MinoanER [16], which subsequently applies four rules. For the task of role matching, we need to modify this process, because it is designed for the clean-to-clean entity resolution use-case (every entity is expected to match at most one other entity) and assumes that most entities have a distinctive name property. These assumptions do not hold for role matching and a 1-to-1 application of MinoanER would hurt its performance. Therefore, we drop the *Name Matching Rule (R1)* and *Reciprocity Matching Rule (R4)* and modify the two remaining rules (*Value Matching Rule (R2)* and *Rank Aggregation Rule (R3))*, by computing their corresponding scores (the value and neighbor similarity). For details about these scores we refer the reader to the original paper [16].

Given two thresholds, a candidate is then recognized as a match if either of the two scores exceeds the corresponding threshold.

The second approach uses the Magellan system's classifiers [23] – traditional machine learning models, such as decision trees and random forests, that require hand-crafted feature tables containing numeric features. We generate these in the following way: The schema of our feature table contains the class label, as well as one attribute for every observable timestamp t. Given a candidate (r_1, r_2), the feature for t is defined as the edit distance $editDist(L_{r_1}[t], L_{r_2}[t])$ between the lineages' values at that timestamp. It is notable that this results in much redundant information for candidates where both roles only change their values occasionally. However, the requirement for all candidates to share a common schema is hard to satisfy without such redundancies.

The third matching approach, Ditto [27], is based on language models. It avoids the aforementioned problem of redundancy, because it does not the require same schemata for its candidates. Instead, it transforms the two entities of a candidate into string representations, concatenates them using special separators and then uses them to fine-tune a pretrained language model to classify pairs of entities as matches or non-matches. Ditto can be used to create a matcher for the role matching problem similarly. Given a role lineage L_r, we create its attributes in the following way: For every time $t \in TS$ where there is a change in L_r (including the first insert), we create an attribute named t and assign it the value $L_r[t]$. Additionally, we create a special attribute that stores the duration for which the value v was valid in r. Finally, we also include the subject and predicate IDs as attributes for every role. The attributes of role r are then serialized to a string exactly like they would be for the task of entity matching. For the details of the serialization, we refer the reader to the original paper [27]. Our experiments in Section 7.4 show that this approach can already yield high precision matchings, while maintaining reasonable recall for our datasets.

7 EVALUATION

We evaluate our approach on several datasets from Wikipedia. Our implementations and datasets are publicly available[6]. We give a brief overview of our dataset extraction and preparation steps in Section 7.1. Then, we discuss the creation of two different gold standards in Section 7.2 and report the performance of the proposed blocking methods on them in Section 7.3. Finally, we present the experimental setup and the results for the matching stage in Section 7.4.

7.1 Data Sets

We use the historical data that is provided by the Wikimedia Foundation[7], choosing more than 16 years between early 2003 and late 2019. We consider solely data stored in infoboxes, which are collections of key-value pairs for a specific entity (a Wikipedia page). Thus, a role r is the combination of the Wikipedia page as the subject and the property in the infobox as the predicate. Infoboxes are organized by templates, such as *infobox book, infobox football player*, etc., which act as categories. To obtain larger datasets and thus the potential to discover more role matchings, we manually assigned 36 templates of the top 100 templates into five broader categories: `Education (ED)`, `Football (FO)`, `Military (MI)`, `Politics (PO)`, and `TV and Film (TV)`.

An issue of this dataset is that details of the Wikipedia markup syntax can make two data points that are semantically the same look very different, for example because users add special formatting or extra icons and images. To mitigate this problem, we preprocess values containing Wikilinks (links to other Wikipedia pages): For every key-value pair that contains at least one Wikilink, we

---

Table 2. Basic Statistics about the datasets. *Avg. #DV* denotes the average number of distinct values in a lineage, *Avg. Data Density* denotes the average share of non-wildcard values in a lineage (with standard deviations).

Dataset	#roles	Avg. #DV	Avg. Data Density
ED	73998	4.22 (2.16)	0.58 (0.15)
FO	325933	7.59 (11.64)	0.50 (0.17)
MI	92338	5.46 (5.47)	0.61 (0.16)
PO	66684	4.35 (2.65)	0.50 (0.16)
TV	217521	5.47 (6.82)	0.54 (0.17)

create an additional, separate key-value pair per contained Wikilink whose value is just the target of the link, thus removing any boilerplate or formatting instructions.

To reduce the impact of vandalism, which frequently appears in Wikipedia [2], and to achieve reasonable runtime for the RM^γ, we aggregate the time granularity of the observed data to weekly snapshots using the majority value in that week. The resulting lineages have 870 data points (one for each week). Some key-value pairs on Wikipedia are not of interest to us, either because they are created accidentally or as vandalism, or because they are completely static, meaning they never change. Thus, we filter the obtained lineages by demanding that they must contain at least two distinct (non-wildcard) values before 2016-05-07. Furthermore, we demand that there must be at least one value observation after 2016-05-07. This is relevant for the matching stage in Section 7.4, because we use the time period after 2016-05-07 to automatically generate labels for candidate pairs. For all experiments and statistics in Sections 7.1–7.3, we consider only the time period between the beginning (2003-01-04) and 2016-05-07. Table 2 presents basic statistics of the datasets. It reveals that there are not many distinct values per lineage on average and that the data density (meaning the share of non-wildcard values) is only slightly higher than 50% for most datasets. However, the variance for the number of distinct values is high, showing that while there are some roles that rarely change, others change more frequently.

7.2 Gold Standards

To judge the recall of the different suggested blocking methods, we require knowledge about true (semantically valid) role matchings. For that purpose, we manually annotated pairs of roles from the different Wikipedia datasets. When trying to annotate data, two issues arise. First, it is extremely unlikely to sample true positives from a uniform sample of the set of all pairs. Thus, we limited the set of pairs that we look at to those candidates (r_1, r_2), where $\exists t \in TS, L_{r_1}[t] = L_{r_2}[t]$, i.e., a value agreement in at least one timestamp. In the following, we also refer to this set of candidates as *1TVA*. Second, if one were to uniformly sample pairs from 1TVA, the result set would be dominated by candidate pairs that agree on frequently occurring values, such as "1", "NA", "−", etc. Such agreements would mostly be by chance, once again leading to few true positives. Also, such true positives would not be a very diverse representation of the types of role matchings that we can find in the datasets because the chance of selecting a pair of roles that agree on a rarely occurring value is very low. Therefore, we used a different procedure to sample our first gold standard.

7.2.1 Diverse Gold Standard (DGS). We created DGS by repeating the following procedure until we reached our sample size:

(1) Randomly choose a timestamp $t \in TS$.
(2) Group all roles in the dataset by their value at timestamp t.
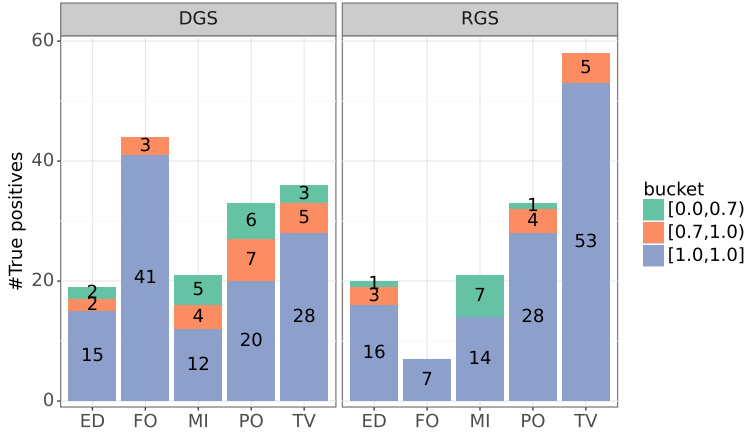(3) Randomly choose one of the above groups with size > 1.

Fig. 6. Distribution of true positives into the three buckets of the stratified sample for both gold standards.

(4) Randomly choose one of the pairs in that group.

This procedure is biased towards pairs occurring in smaller groups (meaning their values are rarer). This increases the chances of sampling true role matchings and gives us a more diverse set of role matchings. We first used this procedure to obtain 100 candidate pairs per dataset (thus 500 in total), which we annotated. On this sample we observed that 55.5% of all true role matchings were fully compatible and that no true role matching had a role compatibility (see Definition 4.2) of less than 70%. However, we still observed very few true role matchings (only 27 across all datasets), so this was not enough of a sample to evaluate the suggested blocking methods. Thus, we extended this sample to a stratified sample in the following way: The sampling was continued until the sample consisted of 100 pairs per dataset for each of the following compatibility ranges: low ($[0, 0.7)$), high ($[0.7, 1)$), and full ($[1, 1]$). In the following we also refer to these as the *bottom*, *middle* and *top* bucket. Thus, DGS amounts to a total of 1,500 annotated pairs (300 per dataset).

7.2.2 Representative Gold Standard (RGS). While DGS gives us a good overview over what different kinds of role matchings we can expect to find in the data, it has one drawback: Because it was sampled non-uniformly, it cannot be used to estimate the performance on the entire dataset. In order to sample a new gold standard uniformly, we used the information we gained from DGS to further reduce the set of input pairs. In DGS, more than 90% of all true positives (r_1, r_2) have at least 95 timestamps at which they agree on the same value, and they agree on at least two different values. In the following, we will refer to this set as 95TVA&2DVA. We thus uniformly sampled a stratified sample, using the same buckets as for DGS, from 95TVA&2DVA to obtain RGS. While this filter may in theory miss matches below these thresholds, in Section 7.3.3 we argue that this is unlikely.

7.2.3 Basic Statistics for DGS and RGS. The distribution of true positives for DGS and RGS is reported in Figure 6. The figure shows that a large majority of true positives that we annotated are indeed fully compatible (the $[1.0, 1.0]$-buckets). In total, the bucket of fully compatible candidates contains 76% of all true positives in DGS and 86% of all true positives in RGS, showing that the notion of compatibility is indeed useful to identify true role matchings.

However, while we sampled from all buckets equally, the size of these buckets is not equal in the underlying population. To determine the bucket sizes in the underlying population, we sampled 100,000 candidate pairs uniformly from 95TVA&2DVA. In this sample, we found that 17.7% of

Table 3. Recall of all non-parametrized blocking methods on our manually annotated gold standards in percent.

Dataset	DGS					RGS				
	EM	CBRB	CQM	TSM	VSM	EM	CBRB	CQM	TSM	VSM
ED	36	**78**	57	73	73	41	47	**67**	**67**	**67**
FO	88	93	90	93	**95**	**100**	**100**	**100**	**100**	**100**
MI	23	**57**	33	33	33	18	**85**	44	69	69
PO	45	60	54	60	**63**	37	62	66	76	**78**
TV	41	**77**	66	75	75	7	50	78	**99**	**99**
MIA	52	**75**	65	71	72	27	59	68	80	**81**
MAA	47	**73**	60	67	68	41	69	71	**82**	**82**

Table 4. Reduction ratios (in percent) for all non-parametrized blocking methods, as well as RM^γ. The single transition overlap filter (as defined in Section 4) was additionally applied to all blocking methods.

Dataset	EM	CBRB	CQM	TSM	VSM	$RM^{0.88}$
ED	**99.99997**	99.99987	99.99974	99.99957	99.99935	99.99976
FO	**99.99997**	99.99979	99.99633	99.99550	99.99342	99.99712
MI	**99.99994**	99.99925	99.99904	99.99828	99.99821	99.99871
PO	**99.99991**	99.99982	99.99964	99.99953	99.99948	99.99959
TV	**99.99999**	99.99998	99.99990	99.99980	99.99968	99.99988
MAA	**99.99996**	99.99974	99.99893	99.99853	99.99803	99.99901

all pairs are in the bottom bucket, 70.5% of all pairs are in the middle bucket and 11.8% of all pairs are in the top bucket. Thus, in the following evaluation, we report weighted recall on RGS, meaning that true positives are weighted by the relative size of their respective buckets in the underlying population. For example, this means that for the algorithm performance in terms of recall, recognizing a true positive from the middle bucket is as important as recognizing seven true positives from the top bucket. Because the bottom and middle buckets represent the case of dirty data (in clean data, true role matchings would be expected to be fully compatible) and RGS emphasizes the importance of these buckets, this gold standard essentially models the case of dirty data, whereas DGS is closer to the case of clean data.

7.3 Performance on the Gold Standards

For every blocking method, we report *recall* (also known as pairs completeness), meaning the relative number of true positives that a method retains, and *reduction ratio*, meaning the number of candidate pairs that a blocking method filters out, relative to the number of all pairs [10]. We first report the results for the non-parametrized methods and subsequently discuss the parametrized methods $CBRB_\beta$ and RM^γ.

7.3.1 Non-Parametrized Methods. Table 3 shows the recall of all non-parametrized blocking methods on DGS and the weighted recall on RGS. The table shows that CBRB achieves the highest macro- and micro-average recall in the diverse gold standard. However, this is not the case in the representative gold standard. Here, not being able to recognize true positives from the middle bucket greatly hurts CBRB's performance, dropping its weighted macro-average recall to 69%. This is expected as CBRB was designed to work for clean data, and RGS emphasizes the dirty parts of
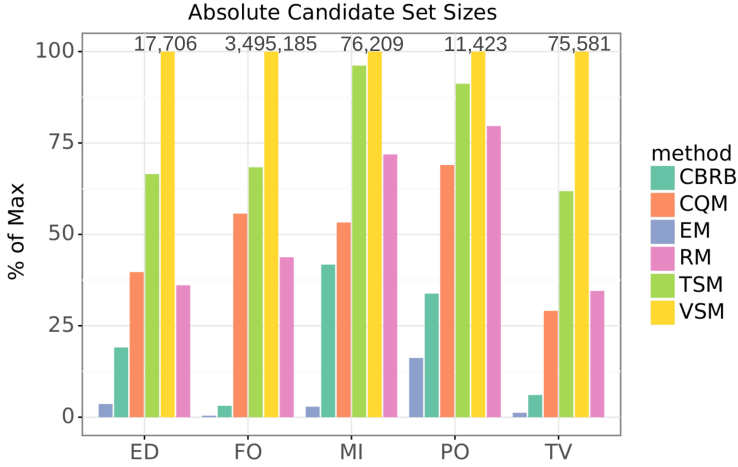
Fig. 7. Absolute number of candidates that the non-parametrized blocking methods retain, relative to the maximum that any of them retain per dataset. RM was set to achieve the same recall as TSM on RGS.

the data. On RGS, the two blocking methods TSM and VSM perform similarly in terms of recall, with VSM performing slightly better for the MI and PO datasets. As expected, EM is too strict as a blocking method and while CQM performs better on average than CBRB in RGS, it does not reach the same recall levels as TSM or VSM.

To enable a fair comparison for the reduction ratios, the transition filter used by CBRB, as specified in Section 4, was also applied to all other blocking methods. This filter does not reduce recall in either gold standard, and thus can be applied without disadvantage. However, we did observe that, as suspected, this filter is especially valuable for CBRB and less fruitful for the other methods. The filter reduces the output size of CBRB by an additional 88% on average, whereas for the other methods this results only in a moderate additional reduction between 22% for EM and 39% for VSM.

Table 4 shows for every dataset the reduction ratios of all non-parametrized methods wrt. the set of all pairs. The only aggregate that the table reports is the macro-average, because the micro-average would be dominated by FO and TV due to their large sizes. For a convenient comparison, the parametrized method RM^γ is also included in the table with $\gamma = 0.88$, which results in the same recall as the best performing methods on RGS (which are TSM and VSM). While at first glance, the results in Table 4 look mostly indistinguishable from each other, it is important to keep in mind that even a seemingly minor increase in reduction ratio (for example at the third decimal) can drastically lower the absolute number of candidates to check because the set of all pairs is very large. Figure 7 visualizes the difference between the methods more closely. While VSM retains almost 3.5 million candidates for FO and TSM still retains more than 2.3 million candidates, CBRB retains just a little more than 100,000 candidates, making it clearly superior in terms of reduction ratio. As a general rule of thumb: A difference of 0.0001% in reduction ratio already leads to a difference in retained pairs of 2,000 for the PO dataset and 50,000 for the FO dataset.

7.3.2 Parametrized Methods. For the two parametrized methods, namely $CBRB_\beta$ and RM^γ, we can tune the tradeoff between recall and reduction ratio. Specifically for RM^γ, we can achieve arbitrarily high recall levels, but naturally, some settings will result in prohibitively large candidate sets returned by the blocking. Because of the prohibitively large size for some parameter settings,
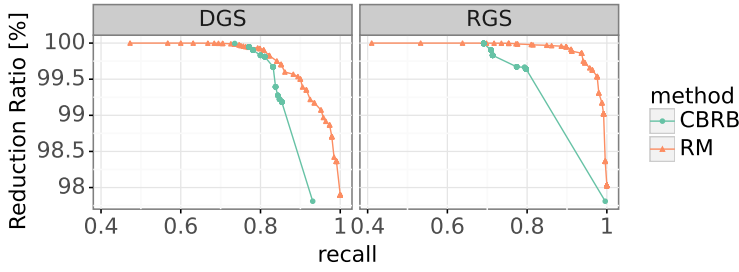
Fig. 8. Tradeoff between reduction ratio and recall (macro-average for both)

the reduction ratios that are presented in the following were estimated on a large sample drawn from the set of all pairs for every dataset.

Figure 8 visualizes the trade-off between recall and reduction ratio for the parametrized methods. It reports the macro-averages over all datasets. The figure shows that RM^γ is superior to $CBRB_\beta$ in all settings that result in recall greater than 80%. This is not unexpected, as the decay function is designed for the presence of outdated values, whereas a general relaxation like RM^γ is helpful in more scenarios. However, a closer look at the individual datasets revealed that for the MI dataset, $CBRB_\beta$ was actually superior to RM^γ, showing that applying decay has merit in certain cases. Overall though, the figures clearly show that RM^γ is the preferred method for high-recall values. For RGS, it can achieve 95% recall, while maintaining a reduction ratio above 99.99%.

A closer look at Table 4 reveals that for RGS, RM^γ is also superior to TSM and VSM. If the γ parameter is set for a target (macro-average) recall of 82% for RGS (which is what TSM and VSM can achieve), RM^γ actually surpasses both TSM and VSM in terms of reduction ratio, achieving a reduction ratio of 99.9990%. For DGS, RM^γ remains inferior to CBRB in terms of reduction ratio: CBRB achieves 99.9997% reduction ratio at 73% recall, whereas RM^γ achieves a reduction ratio of only 99.9993% for that recall setting.

7.3.3 Impact of the TVA filter on RGS. As detailed in Section 7.2, RGS was sampled using a filter, which requires candidates to agree on 95 timestamps. Therefore, there is a theoretical possibility that true-positives with low agreement exist but do not appear in our gold standard. However, experimental evidence suggests that this is not the case. Figure 9 shows the impact of stricter filters on the recall in RGS. It can be seen that even a much larger agreement count of 364 does not impact recall in RGS for any method. In fact, the lowest TVA-agreement of any true positive in RGS is 218 and there are only 3 true positives in the range [218,364] (which none of the methods find). Thus, it is reasonable to assume that 95 as a threshold does not remove a relevant amount of true positives. We confirmed this by unformly sampling another 500 pairs (100 per dataset) with agreement counts in [1,94] and found no true positives among them.

7.3.4 Impact of Missing Values. Because missing values are an important aspect of temporal data, especially in Wikipedia, we additionally investigate how well the blocking methods perform in such cases. For this purpose, we randomly replace changes in the data with wildcards with probabilities of up to 50%. Because we observed that the transition filter, which we previously applied, starts to affect recall once more missing values are introduced, we replace it with a less strict filter that requires only one value agreement at any point in time (1TVA) for these experiments. Figures 10 and 11 show the impact of missing values on recall and reduction ratio. We can see that all non-parametrized blocking methods except CBRB achieve worse recall with an increasing number of missing values. The recall of CBRB however, increases with more missing values as it is specifically
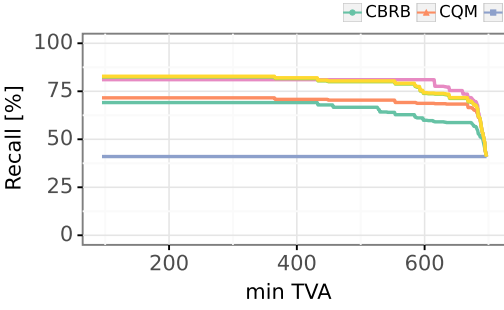
Fig. 9. Impact of TVA on recall (macro-avg) in RGS. RM was set to achieve the same recall as TSM (at TVA=95).
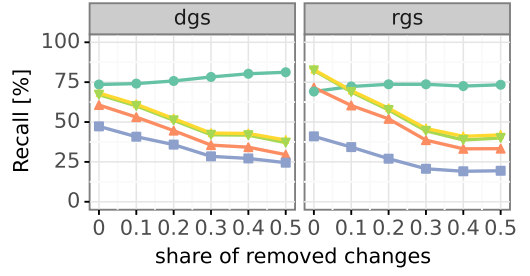
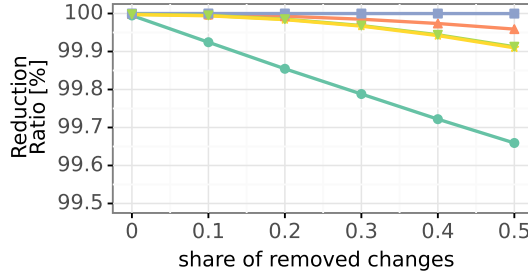Fig. 10. Impact of missing data on recall (Macro-Avg).

Fig. 11. Impact of missing data on reduction ratio.
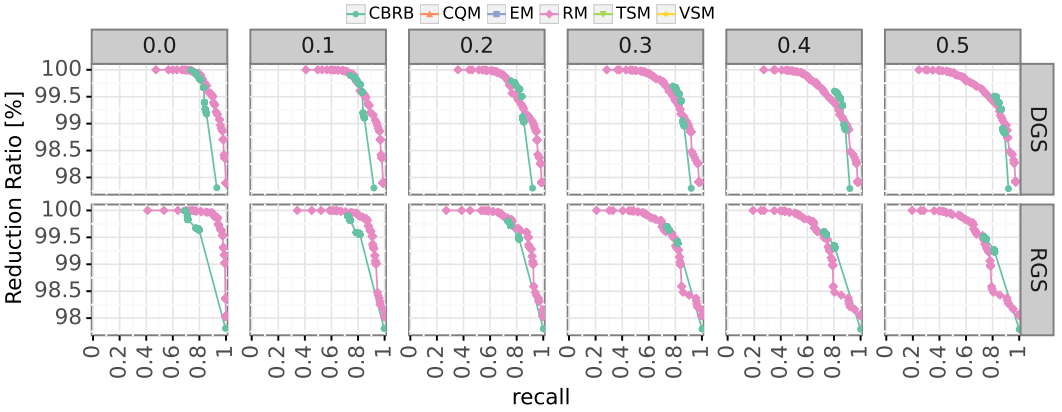

Fig. 12. Tradeoff between recall and reduction ratio with a varying fraction of missing values.

able to handle such situations. While the reduction ratios of all methods decrease with more missing values, it is notable that CBRB suffers the most. This is expected, as more missing values lead to more wildcards and thus more accidental matches in CBRB.

The results for the parametrized methods are shown in Figure 12. Once again, we see that the more values are missing, the better $CBRB_\beta$ becomes in comparison to RM^γ. However, it is important

(a) ... the input size.
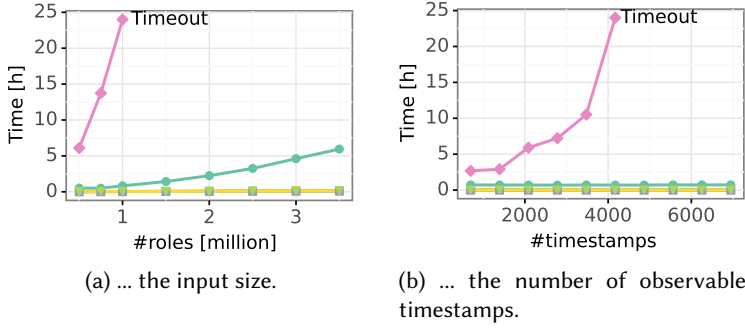
(b) ... the number of observable timestamps.

Fig. 13. Runtime impact of ...

to note that the more data is missing, the less impactful the decay function is, leaving $CBRB_\beta$ with fewer tunable settings than RM^γ.

7.3.5 Scalability. We investigate two different dimensions in terms of scalability: the number of roles in the input and the number of observable timestamps. To enable a fair comparison for the following experiments, the threshold parameter of RM^γ was set so that RM^γ achieves the same recall as TSM (the best of the other methods) on RGS. Both RM^γ and CBRB were run with 32 threads. The experiments were executed on a server running Ubuntu 20.04 LTS with two Intel Xeon E5-2650 2.00 GHz CPUs and 256 GB RAM. Executions were terminated if they exceeded a runtime of 24 hours (marked with "Timeout" in figures).

To evaluate the blocking methods on a larger number of roles, we synthetically extended our largest dataset (F0) by resampling existing roles and randomizing the values that a resampled role changes to. The randomization is weighted according to the value distribution in the original dataset, meaning that frequently occurring values are more likely to be sampled. Figure 13a visualizes our results. It shows that EM, CQM, TSM, and VSM, in the following called grouping-based methods, are very fast to compute. CBRB takes significantly longer than the grouping-based methods, but shows only a moderate runtime increase with the number of roles. RM^γ, however quickly exceeds 24 hours without terminating, showing that it struggles with larger data sets. It should be noted that datasets of many million roles are not an artificial use-case. Knowledge bases, such as Wikidata [19] or Yago [34], contain tens of millions of entities and thus many times more roles.

To scale the number of observable timestamps, we keep all roles in the F0 dataset as they are, but stretch the time periods until values are changed, leading to the same temporal behavior but over a longer period (more observable timestamps). Figure 13b shows that, as expected, the grouping-based methods as well as CBRB are not affected by having more observable timestamps. The runtime of RM^γ on the other hand grows with the number of timestamps, because it materializes all values of all roles at every timestamp.

7.3.6 Summary. Overall, the different blocking methods behave as expected. More relaxed methods offer higher recall, but result in a larger candidate set. The evaluation has shown that respecting not just the order, but also the exact timestamps at which roles assume values is beneficial. The set-based methods TSM and VSM are inferior to methods that respect exact timestamps, such as CBRB for DGS (the clean data case) and RM^γ for RGS (the dirty data case). For clean data in particular, the evaluation has shown that the notion of compatibility is very beneficial for retaining role matchings as opposed to demanding exact matches: CBRB greatly increases the recall in comparison to EM. Furthermore, the more missing values are present, the better CBRB becomes in

comparison to the other methods. For dirty data with few missing values however, RM^γ is superior. In terms of scalability, RM^γ is much worse than the other methods, as it already exceeds a runtime of 24 hours for an input of one million roles.

Taking all of the above into consideration, the ideal blocking method to use depends on the desired recall, computational capabilities of the matcher, the size of the input data as well as other properties of the data, such as the amount of missing values. In general, RM^γ offers a flexible tradeoff between recall and reduction ratio and achieves high reduction ratios even for large recall values. Our experiments showed that it can reach 95% recall on RGS, while maintaining a reduction ratio of more than 99.99%, making it a good choice in standard scenarios. However, it does not scale well with the number of roles or observable timestamps, so that other methods may need to be chosen in such scenarios. If the input data is mostly clean or contains many missing values, CBRB is a more scalable alternative to RM^γ. If the input data is mostly dirty, but contains few missing values, TSM is the better alternative, as it also scales very well and deals better with dirty data than CBRB.

7.4 Matching Stage

To evaluate the matching stage, we require a different way of obtaining annotated data, because even after manually labelling 3,000 role matching candidates, we have an insufficient number of true positives available to properly train and test the matchers produced by Magellan or Ditto [27]. Instead, we use the latter 20% of the available time period (2016-05-07 – 2019-09-07) that we previously split off (see Section 7.1) to automatically label candidate pairs. We call that time period the test period. For this experiment, a candidate (r_1, r_2) is labelled as a correct role matching if r_1 exactly matches (as defined by blocking method EM) r_2 during the test period. Furthermore, we consider only candidates for which we observe at least one change to either r_1 or r_2 in the test period, reducing the likelihood that r_1 and r_2 match by chance. While this is an imperfect substitute for actual human-labeled ground truth, it gives us the opportunity to evaluate the matching stage on a large scale. As a blocking method, we used RM^γ, with γ set to achieve an average recall of 80%. Notably, the result set for the F0 set was still too large to be processed by Ditto in reasonable time, which is why we adjusted the threshold for this dataset to $\gamma = 1.0$, which is equivalent to the EM method. After blocking, we executed a 3-1-1 split on the candidate pairs, as is common in related work on Entity Matching [27]. This means, we reserved 60% of the data for training, 20% for validation, and 20% for testing. The performance of all approaches is compared on the test set.

The Ditto models were trained for 20 epochs using the RoBERTa language model [29] with all optimizations suggested by the authors, namely domain knowledge, TF-IDF summarization, and data augmentation [27]. For Magellan [23], we report only the performance of the random forest classifier, which performed best for all datasets. For MinoanER [16], we report the F1-score for the optimal setting of the two threshold parameters. The achieved F1-scores of all approaches are reported in Table 5. They show that Ditto outperforms the other methods, achieving a macro-F1 score of 89%, which is 17 percentage points higher than the second best method. It only falls slightly to MinoanER for the football (F0) dataset. A closer inspection of that dataset revealed that there are many input pairs in both training and test data where the two roles are different parts of uniform-colors of the same football team, for example the colors for arms and legs. Ditto struggles to separate true positives from false positives because the colors sometimes match by chance in the test period. This is something that we only rarely observed in the other datasets, and having human-annotated data available could alleviate these issues.

In our experiments, we observed that Ditto can also be tuned to find role matchings with high precision, while maintaining a satisfactory recall, which would likely be most important in a data cleaning scenario. For example, if we demand 95% precision as our target, Ditto can still achieve a

Table 5. F1-Scores (in percent) of all matching approaches.

Dataset	MinoanER	Magellan	Ditto
ED	67	64	**87**
FO	**80**	74	78
MI	51	58	**97**
PO	57	72	**88**
TV	77	92	**95**
Macro-Avg	66	72	**89**

macro-recall of 67%, which still corresponds to thousands of role matchings that can be used to improve data quality of Wikipedia.

8 CONCLUSIONS

We have introduced the problem of role matching in temporal data. A role is defined as the combination of subject and predicate. A role matching is a novel integrity constraint that states that two roles should always refer to the same object (i.e., have the same value). The task of role matching is to identify, among all pairs of roles, exactly those roles that are in a role matching. Like entity matching, role matching can be divided into the blocking and matching stages. We have conceived, discussed, and implemented seven different blocking strategies for role matching in both clean and dirty data. To evaluate these methods, we created two manually annotated gold standards of 1,500 annotated pairs each. The evaluation shows that the usage of temporal information is key to both obtaining a high recall and reducing the set of pairs to check efficiently. For CBRB, the best performing blocking method for the case of clean data or data with many missing values, we have introduced the key notion of compatibility and presented an efficient search strategy to compute its result set. All presented blocking methods except for RM^y scale well with the number of timestamps and roles, making them applicable to large datasets. For the matching stage, our experiments show that the schema-agnostic entity matcher Ditto [27] outperforms a modified version of MinoanER [16] and traditional supervised learning approaches [23]. On automatically generated labels, Ditto manages to achieve a Macro-F1 of 89% and can also maintain decent recall for higher precision settings.

For future work, one could assemble a larger human-annotated gold standard to test whether Ditto works as well on human-labelled data as it did for our automatically generated labels and improve the matching stage otherwise. Furthermore, role matching could be applied to different data sources, such as data extracted from textual web pages, to see whether new challenges arise in such a setting. Finally, the problem of incrementally discovering and maintaining role matchings in a perpetually evolving system could be studied.

REFERENCES

[1] 1787. U.S. Constitution article II, § 2, cl. 1.

[2] B. Thomas Adler, Luca De Alfaro, Santiago M Mola-Velasco, Paolo Rosso, and Andrew G West. 2011. Wikipedia vandalism detection: Combining natural language, metadata, and reputation features. In *International Conference on Intelligent Text Processing and Computational Linguistics*. Springer, 277–288.

[3] Enrique Alfonseca, Guillermo Garrido, Jean-Yves Delort, and Anselmo Peñas. 2013. WHAD: Wikipedia historical attributes data - Historical structured data extraction and vandalism detection from the Wikipedia edit history. *Language Resources and Evaluation* 47, 4 (2013), 1163–1190.

[4] 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.

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

[6] Christoph Böhm, Gerard de Melo, Felix Naumann, and Gerhard Weikum. 2012. LINDA: Distributed Web-of-Data-Scale Entity Matching. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)* (Maui, Hawaii, USA) *(CIKM '12)*. Association for Computing Machinery, New York, NY, USA, 2104–2108. https://doi.org/10.1145/2396761.2398582

[7] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F Naughton. 2014. Modeling entity evolution for temporal record matching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1175–1186.

[8] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F Naughton. 2014. Tracking entities in the dynamic world: A fast algorithm for matching temporal records. *PVLDB* 7, 6 (2014), 469–480.

[9] Peter Christen. 2012. The data matching process. In *Data matching*. Springer, 23–35.

[10] Peter Christen and Karl Goiser. 2007. Quality and complexity measures for data linkage and deduplication. In *Quality measures in data mining*. Springer, 127–151.

[11] Victor Christen, Peter Christen, and Erhard Rahm. 2019. Informativeness-based active learning for entity resolution. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 125–141.

[12] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2020. An overview of end-to-end entity resolution for big data. *Comput. Surveys* 53, 6 (2020), 1–42.

[13] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. 2009. Integrating conflicting data: the role of source dependence. *PVLDB* 2, 1 (2009), 550–561.

[14] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. 2009. Truth discovery and copying detection in a dynamic world. *PVLDB* 2, 1 (2009), 562–573.

[15] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *PVLDB* 11, 11 (2018), 1454–1467.

[16] Vasilis Efthymiou, George Papadakis, Kostas Stefanidis, and Vassilis Christophides. 2019. MinoanER: Schema-agnostic, non-iterative, massively parallel resolution of web entities. (2019), 373–384.

[17] I. Fellegi and A. Sunter. 1969. A theory of record linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969).

[18] Alban Galland, Serge Abiteboul, Amélie Marian, and Pierre Senellart. 2010. Corroborating information from disagreeing views. In *Proceedings of the International Conference on Web Search and Data Mining (WSDM)*. 131–140.

[19] Armin Haller, Axel Polleres, Daniil Dobriy, Nicolas Ferranti, and Sergio José Rodríguez Méndez. 2022. An Analysis of Links in Wikidata. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, Vol. 13261. Springer, 21–38.

[20] Mauricio A. Hernández and Salvatore J. Stolfo. 1995. The Merge/Purge Problem for Large Databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 127–138.

[21] Yichen Hu, Qing Wang, Dinusha Vatsalan, and Peter Christen. 2017. Improving temporal record linkage using regression classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 561–573.

[22] Robert Isele and Christian Bizer. 2012. Learning Expressive Linkage Rules Using Genetic Programming. *PVLDB* 5, 11 (jul 2012), 1638–1649. https://doi.org/10.14778/2350229.2350276

[23] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *PVLDB* 9, 12 (2016), 1197–1208.

[24] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani. 2013. SIGMa: Simple Greedy Matching for Aligning Large Knowledge Bases. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)* (Chicago, Illinois, USA) *(KDD '13)*. Association for Computing Machinery, New York, NY, USA, 572–580. https://doi.org/10.1145/2487575.2487592

[25] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. 2011. Linking temporal records. *PVLDB* 4, 11 (2011), 956–967.

[26] Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. 2016. A survey on truth discovery. *Knowledge Discovery and Data Mining (SIGKDD) Explorations Newsletter* 17, 2 (2016), 1–16.

[27] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *PVLDB* 14, 1 (2020), 50–60. https://doi.org/10.14778/3421424.3421431

[28] Yaliang Li, Qi Li, Jing Gao, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. 2015. On the discovery of evolving truth. In *Proceedings of the International Conference on Knowledge discovery and data mining (SIGKDD)*. 675–684.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). http://arxiv.org/abs/1907.11692

[30] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The Four Generations of Entity Resolution.* Morgan & Claypool Publishers.

[31] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *Comput. Surveys* 53, 2, Article 31 (March 2020), 42 pages.

[32] Jeff Pasternack and Dan Roth. 2010. Knowing what to believe (when you already know something). In *International Conference on Computational Linguistics (COLING)*. 877–885.

[33] Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active learning for large-scale entity resolution. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. 1379–1388.

[34] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. 2020. YAGO 4: A Reason-able Knowledge Base. In *Proceedings of the Extended Semantic Web Conference (ESWC)*, Vol. 12123. Springer, 583–596.

[35] Mohamed Yakout, Ahmed K Elmagarmid, Hazem Elmeleegy, Mourad Ouzzani, and Alan Qi. 2010. Behavior based record linkage. *PVLDB* 3, 1-2 (2010), 439–448.

[36] Xiaoxin Yin, Jiawei Han, and S Yu Philip. 2008. Truth discovery with multiple conflicting information providers on the web. *IEEE Transactions on Knowledge and Data Engineering (TKDE)* 20, 6 (2008), 796–808.

[37] Xiaoxin Yin and Wenzhao Tan. 2011. Semi-supervised truth discovery. In *Proceedings of the International World Wide Web Conference (WWW)*. 217–226.

[38] Zhou Zhao, James Cheng, and Wilfred Ng. 2014. Truth discovery in data streams: A single-pass probabilistic approach. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*. 1589–1598.

[39] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. LSH Ensemble: Internet-scale domain search. *PVLDB* 9, 12 (2016), 1185–1196.