Modeling Entity Evolution for Temporal Record Matching

Yueh-Hsuan Chiang
University of Wisconsin
Madison
1210 W Dayton Street
Madison, WI 53706, USA
yhchiang@cs.wisc.edu

AnHai Doan
University of Wisconsin
Madison
1210 W Dayton Street
Madison, WI 53706, USA
anhai@cs.wisc.edu

Jeffrey F. Naughton
University of Wisconsin
Madison
1210 W Dayton Street
Madison, WI 53706, USA
naughton@cs.wisc.edu

ABSTRACT

Temporal record matching recognizes that if the entities represented by the records change over time, approaches that use temporal information may do better than approaches that do not. Any such temporal matching method relies at its heart on a temporal model that captures information about how entities evolve. In their pioneering work, Li *et al.* used an efficiently computable model that simply tries to predict if an attribute is expected to change over a given time interval. In our work, we propose and evaluate a more detailed model that focuses on the probability that a given attribute value reappears over time. The intuition here is that an entity might change its attribute value in the way that is dependent on its past values. In addition, our model considers sets of records (rather than simply pairs of records) to improve robustness and accuracy. Experimental results show that the resulting approach improves both accuracy and resistance to noise while incurring a minimal overhead.

1. INTRODUCTION

Record matching takes a collection of records and determines which records belong to the same real world entity (see [8, 12] for recent surveys). The vast majority of work in record matching has assumed that the records come with no temporal information. In practice, however, we often have temporal information in the form of time stamps associated with records. Examples include author records in DBLP [2], donor records in federal campaign finance data sets [3], and time stamps in tweets [4].

It has been shown that approaches that use temporal information can do better than approaches that do not, especially when the entities described by a data set may change or evolve their attribute values over time [14]. Any such temporal matching method relies at its heart on a temporal model that captures information about how entities evolve. This immediately gives rise to the question: "what kind of model should one use?" It should be simple enough to be easy to construct, yet powerful enough to be effective and robust in the presence of noise.

In their pioneering work, Li *et al.* used an efficiently computable model that simply tries to predict if an attribute is expected to change over a given time interval [14]. Because it only models "change" and "no change" it, for example, cannot tell the difference between an attribute that changes from one value to another chosen at random, and an attribute that changes only between values drawn from a small, entity-dependent set. Their model is also a "point to point" model that considers pairs of records rather than sets of records.

Table 1: Records of the same author from DBLP.

rid	eid	name	affiliation	co-authors	year
r_1	e_1	Lei Wang	Xidian University	Licheng Jiao	1999
r_2	e_1	Lei Wang	Xidian University	Licheng Jiao	2000
r_3	e_1	Lei Wang	Xidian University	Licheng Jiao	2001
r_4	e_1	Lei Wang	Xi'an Univ. of Tech.	Yinling Nie, Weike Nie, Licheng Jiao	2005
r_5	e_1	Lei Wang	Xi'an Univ. of Tech.	Yinling Nie, Weike Nie, Licheng Jiao	2006
r_6	e_1	Lei Wang	Xi'an Univ. of Tech.	Liya Wang, Yinling Nie	2006
r_7	e_1	Lei Wang	Xidian University	Jiaji Wu, Licheng Jiao, Li Zhang, Guangming Shi	2007
r_8	e_1	Lei Wang	Beijing U. of A&A	Zheng Wang, Chen Yang, Li Zhang, Qiang Ye	2009
r_9	e_1	Lei Wang	Xidian University	Licheng Jiao, Jiaji Wu, Guangming Shi	2009
r_{10}	e_1	Lei Wang	Xidian University	Licheng Jiao, Jiaji Wu, Guangming Shi, Yanjun Gong	2010

We present and evaluate a more complex model. It is not "point to point"; rather, when processing records in increasing temporal order, and deciding whether or not to add a new record to a growing cluster, it uses all the records in the cluster to answer the question. That is, rather than saying "given record r_1, does record r_2 refer to the same entity?", it says "given a cluster C of records already determined to refer to a single entity, does a new record r refer to the same entity?"

Furthermore, our model is able to learn patterns more detailed than just "change/no change." In particular, our model focuses on the probability of a value re-appearing over time. The intuition here is that an entity might change its attribute values in a way that is dependent on or predicted by its past values. One possibility is that some of an entity's attributes might change over time yet have an affinity for a small set of values that are associated with the entity. For example, if a person's location attribute has taken on the values {Madison, Silicon Valley, Taiwan} in the past, then these values may be more likely to appear in this person's records in the future than, say, "France." In other cases, it is also possible that an evolving attribute never changes its value back to a previously seen value. As we will show, by learning probabilities of recurrence, our model is able to make better predictions than those made by simply predicting "change/no change".

EXAMPLE 1.1. *Consider a real world example from DBLP shown in Table 1, which contains several publication records associated with the same author — Lei Wang. From his publication records, we observe that Lei Wang changed both his affiliation and co-authors over time, but in both attributes there exist underlying affinities: Lei Wang returned to Xidian University in 2007 after leaving there in 2001, and over time he also had certain reappearing co-authors*

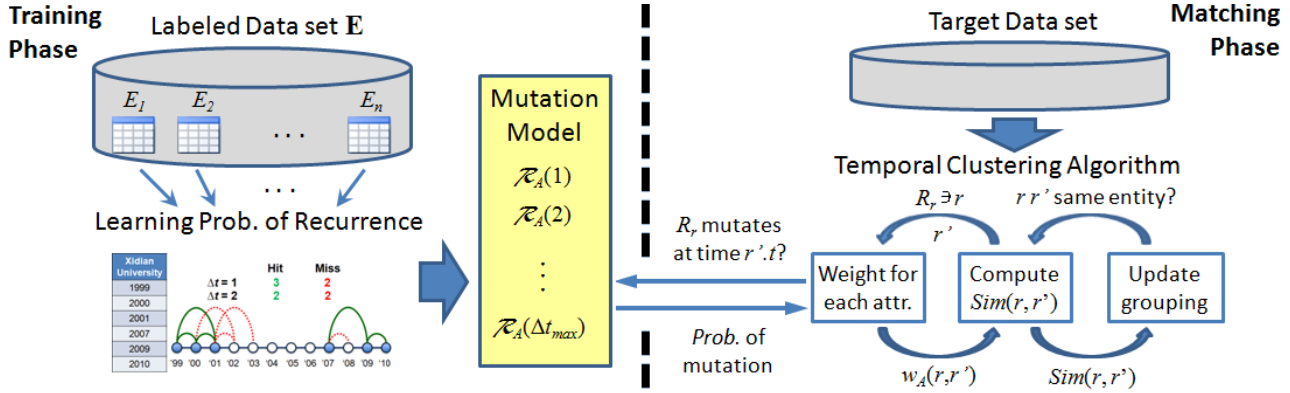


Figure 1: How the proposed model works in the training and matching phases of a temporal matching framework.

even though the coauthor set itself evolved over time. Thus there is potential for a model that considers the recurrence of values to provide better insight than one that does not. □

At a high level, shown in Figure 1, our approach to record matching is the following. We begin with a training data set, which consists of a set of records and their associated entities. We train a temporal model with this data set, attempting to capture useful information about how the entities in the data set evolve. Later, when we are solving the target record matching problem over an unlabeled data set, we use this temporal model as input to the similarity computations performed in the course of this matching. Intuitively speaking, the temporal model tells us how to weight similarities on various attributes. For example, if the temporal model suggests that similarity on a given attribute is not very reliable due to entity evolution, then the result of the similarity computation will be given a low weight. In this paper we focus on the definition of a new temporal model and use it along with previously proposed temporal clustering techniques.

Returning to the training phase of our model, there are many different things we could try to learn. A simple approach would be to merely learn how frequently values change in each attribute of the record set under consideration (this is the approach in [14]). More complicated approaches could focus on either evolution within entities, or global evolution among entities, or both. We consider evolution within entities, because, as we will see in our experiments, if one has good predictions about intra-entity evolution, one can draw useful conclusions about inter-entity evolution.

Our approach for intra-entity evolution is to look at the entire sequence of records corresponding to each entity in the labeled data set and to calculate the probability for each time period that the entity corresponding to the records is likely to change its value to one not seen previously. We call this a "mutation model," and it allows us to answer the question "given that two records with time stamps disagree on some attribute A, how likely is that they refer to the same entity that has evolved to change its A-value?" (If the answer is "it is likely" then we would view this disagreement on attribute A as unconvincing and give it low weight in the similarity computation.)

Perhaps surprisingly, this mutation model can also help answer the question "given two records with different time stamps from two different entities, how likely is it that they will evolve to share a value on some attribute?" As a partial intuition for this, note that for every pair of entities that evolve to share attribute values, at least one of them must have evolved to change attribute values — which is captured by the mutation model.

The precise definition and computation of this mutation model is somewhat complex. However, our experiments show that the greater insight provided by the mutation model is useful, leading to substantially better precision and recall, along with lower sensitivity to noisy or missing data, than the previously proposed frequency of change model.

Of course, such a model does not come for free, so a reasonable question is: is this more detailed model worth it? Does it take a long time to compute? Does it lead to better clustering decisions? Does it have any other good properties? In the rest of this paper, we will show (a) how to compute the proposed model in a way that is not expensive, and (b) how it can be used to make better matching decisions, and (c) that it is more robust to noise.

In more detail, we make the following contributions:

- We introduce the notion of *mutation*, which describes the situation when an entity changes an attribute to a value that has not appeared in that attribute in the history of that entity.
- We describe an algorithm that learns and uses statistics about attribute value recurrence to model entity evolution based on the notion of mutation. Our model is able to handle both single-valued and multi-valued attributes.
- We demonstrate how to utilize entity-dependent information to improve matching accuracy when records are processed in increasing temporal order.
- We propose an effective approximation of our model that reduces computational overhead while producing equivalent matching quality.
- We analyze the matching accuracy and robustness to noise of our model by a set of experiments on several real world temporal matching tasks. The results show that our model improves both matching accuracy (up to 40% in F-1 score) and robustness to noisy data over the state-of-the-art temporal model, while producing minimum computational overhead (less than 5%).

The paper is organized as follows. The next section defines the problem of temporal record matching and reviews necessary background. Section 3 describes our temporal model and its learning process. Section 4 introduces how to apply our temporal model in similarity computation. Section 5 presents our experiments. Section 6 reviews related work, and Section 7 discusses future work and draws conclusions.

2. PRELIMINARIES

This section gives the problem definition and briefly reviews the state of the art technique for temporal record matching proposed in [14].

2.1 Definitions

We now define the problem of temporal record matching.

DEFINITION 2.1. (TEMPORAL RECORD MATCHING) *Consider a domain $\mathcal{D}$ of entities (not known a-priori) and a set* R *of records. Each record $r \in$ R is of the form of $\langle x_1, ..., x_n, t\rangle$, where t is the time stamp of the record r, and each x_i, $1 \leq i \leq n$, is the value of attribute A_i at time t for the associated entity in domain $\mathcal{D}$. The goal of* temporal record matching *is to find a clustering of the records in* R *that satisfies the following properties: 1) records in the same cluster refer to the same entity in domain $\mathcal{D}$, and 2) records in different clusters refer to different entities in domain $\mathcal{D}$.* □

As the entities described by a temporal data set may change or evolve their attribute values over time, a temporal matching technique must deal with ambiguity caused by this evolution. One of such ambiguity is *within-entity temporal disagreement*, or *temporal disagreement* for short:

DEFINITION 2.2. (WITHIN-ENTITY TEMPORAL DISAGREEMENT). *Within-entity temporal disagreement arises when two records referring to the same entity have dissimilar values on one attribute because over time their associated entity evolves its state on that attribute.* □

For example, a person may change his or her address and phone number, so two records referring to the same person at different times may disagree on those attributes.

A second kind of ambiguity is *between-entity temporal agreement*, or *temporal agreement* for short:

DEFINITION 2.3. (BETWEEN-ENTITY TEMPORAL AGREEMENT). *Between-entity temporal agreement arises when two records referring to two different entities have identical or highly similar values on one attribute because over time one of the entities evolved to have the same value in some attribute as that previously held by the other.* □

Returning to our example, it is possible that one person might get the phone number of a second person when the first person takes over the second person's office.

As a result, a temporal matching algorithm must handle how entities might evolve over time to resolve temporal disagreement and agreement.

2.2 Notation

We will use the following terms. For attribute values, we use the term "A-value" to describe a value for attribute A, $r.A$ and $r.t$ to denote the A-value and the time stamp of record r respectively, and $v \in A$ to denote v is an A-value.

To describe the attributes and the time stamps of an entity or a list of records, we use the following notation. Let E be an entity and $\langle r_1, ..., r_n\rangle$ be a list of records associated with E. We use $E.A$ to denote the set of A-values of E, which is the union of A-values of its associated records:

$$E.A = \{r_1.A, r_2.A, ..., r_n.A\}. \quad (1)$$

If A is a set-valued attribute, then $E.A$ is defined as the union of elements of the A-values of E's associated records:

$$E.A = \{v | v \in r.A \wedge r \in E\}. \quad (2)$$

We use $E.t$ to denote the ordered list of time stamps of E given records $r_1, ..., r_n$, sorted in increasing temporal order:

$$E.t = \langle r_1.t, r_2.t, ..., r_n.t\rangle. \quad (3)$$

2.3 Review of Temporal Record Matching

Existing techniques for non-temporal record matching typically consist of two main components: a similarity measure to determine how likely it is that two records refer to the same real world entity, and a clustering algorithm that groups records based on their similarity. Li *et al.* [14] proposed the first solution for temporal record matching. It consists of three main components — a similarity measure, a *temporal model*, and a *temporal clustering algorithm*. Here we review the two temporal components in more detail:

Temporal model: Typically, the similarity between two records is computed based on attribute value similarities. However, if entities evolve over time, attributes that change become less reliable indicators for record matching. In view of this, Li *et al.* proposed building a temporal model that learns how entities evolve in a labeled data set and use it to determine the importance of each attribute in record similarity computations as follows:

$$sim(r, r') = \frac{\sum_{A \in \mathbf{A}} w_A(r, r') \cdot sim_A(r.A, r'.A)}{\sum_{A \in \mathbf{A}} w_A(r, r')} \quad (4)$$

where sim_A is the similarity metric for attribute A and w_A is the weighting function derived from a temporal model. Intuitively, attributes that are less "stable" are less trusted as indicators in matching.

Temporal clustering: In [14], the term *temporal clustering* was presented to describe clustering algorithms that process records in increasing temporal order. One can view a cluster of records all with time stamp previous to time t as representing the history of its associated entity up to time t.

As the main focus of this paper is on temporal modeling, we will omit reviewing temporal clustering algorithms, focusing instead on the temporal modeling approach — the time decay model — proposed in [14].

2.3.1 The Time Decay Model

The time decay model [14] attempts to capture the effect of time elapsing on value evolution. It consists of two components. The first is *disagreement decay*, which handles the ambiguity caused by within-entity temporal disagreement:

DEFINITION 2.4. (DISAGREEMENT DECAY). *Let Δt be a time interval and $A \in \mathbf{A}$ be a single-valued attribute. The disagreement decay of A over time Δt, denoted by $d^{\neq}(A, \Delta t)$, is the probability that an entity changes its A-value within time Δt.* □

The second component is *agreement decay*, which handles the ambiguity caused by between-entity temporal agreement:

DEFINITION 2.5. (AGREEMENT DECAY). *Let Δt be a time interval and $A \in \mathbf{A}$ be an attribute. The agreement decay of A over time Δt, denoted by $d^{=}(A, \Delta t)$, is the probability that two different entities share the same A-value within time Δt.* □

When computing the similarity between two records, the weight of each attribute value is determined by the complement of the agreement- and disagreement-decay on that attribute. Specifically, consider two records r and r'. The weight w_A of their similarity

on attribute A is computed as follows:

$$w_A(r,r') = \begin{cases} 1 - d^{=}(A,\Delta t) & sim_A(r,r') > \theta_h \\ 1 - d^{\neq}(A,\Delta t) & sim_A(r,r') < \theta_l \\ 1 - sim_A(r,r') \cdot d^{=}(A,\Delta t) & \text{otherwise} \\ \quad -(1 - sim_A(r,r')) \cdot d^{\neq}(A,\Delta t) & \end{cases} \tag{5}$$

where $sim_A(r,r')$ denotes the A-value similarity between r and r', Δt is the time difference between r and r', and θ_l and θ_h are thresholds indicating low and high similarity respectively.

3. OUR TEMPORAL MODEL

This section describes our temporal model — the mutation model. During the discussion, we will assume that, as proposed in [14], a clustering algorithm that processes records in increasing temporal order will be used.

3.1 Approach Overview

At the most basic level, our mutation model answers the following question: consider a cluster C of records that have been determined to correspond to the same entity, and a record r with time stamp $r.t$ greater than that of any record in C. Suppose that r also corresponds to the entity associated with C. For a given attribute A of r, what is the probability that the value in $r.A$, say v, does not appear in attribute A in any record in C? As a bit of terminology, if v does not appear in A in any record of C, then we call r a "mutant record," attempting to capture the notion that the entity in question has changed substantially. Hence our goal can be restated as computing the probability that r is a mutant record.

We will explain later how this is used in temporal matching, but the intuition is that if it is likely that r is a mutant record, then the fact that v does not match any A value in C should not be construed to suggest that r and C do not refer to the same entity. In addition, if we are expecting a mutation based C, then we are not expecting to see a C record with a matching A-value, so if we see a matching A-value it is possibly from temporal agreement.

Our main idea is to calculate, for each attribute, the probability that the value appearing in that attribute will recur after one, two, three, ... time units. So if C has only one record in it, say r_1, with time stamp $r_1.t$, and $r.t - r_1.t = \Delta t$, then the probability p that the value $r_1.A$ will reappear in Δt time units is the complement of the probability that r is a mutant record, and $1 - p$ is our desired probability.

Extending this to multiple records in C requires combining the "predictions" of each of the records in C (since they are all at different temporal distances from r). We use a heuristic, described below, to do so.

3.2 Entity Mutation

This section introduces our concept of entity mutation. We say that an entity has had a mutation if an attribute changes to a value that has not been seen in the last Δt time units, where Δt is a tunable parameter.

DEFINITION 3.1. (ENTITY MUTATION). *Let E be an entity and A be an attribute. We say that E has a* mutation *on attribute A at time t if E changes its A-value at time t to a value v such that none of the records associated with E in the past Δt time units from time t has v as its A-value.* □

If A is a set-valued attribute, then the mutation condition is defined as E changing its A-value to v such that none of the elements in v appears in any A-values in E in the past Δt time units from time t.

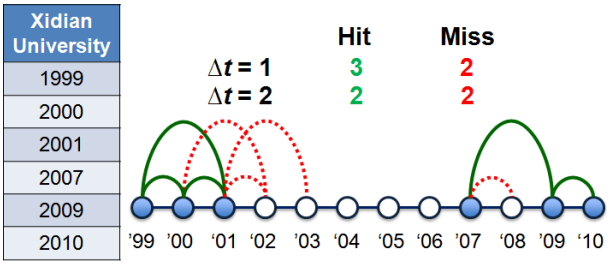


Figure 2: Example of learning attribute value recurrence where each solid / dashed line represents a recurrence his / miss respectively.

Based on the concept of mutation, we further define the notions of *mutant record* and *mutation point* of an entity as follows.

DEFINITION 3.2. (MUTATION POINT / MUTANT RECORD). *Let Δt be a time interval and $A \in \mathbf{A}$ be an attribute, which can be either single-valued or multi-valued. Let r be a record associated with a real world entity E with time stamp $r.t$ and value $r.A = \{r.a_1, ..., r.a_n\}$ on attribute A. We call r a* mutant record *of entity E on attribute A within time Δt if none of the records in E Δt time units ago share a common value with r on attribute A, and we call $r.t$ a* mutation point *of entity E.* □

As a special case, if Δt is set to infinity, then mutant records of one entity on a single-valued attribute are those records containing a value on that attribute that has not appeared in any record associated with the same entity previously.

EXAMPLE 3.1. *Consider an example from DBLP shown in Table 1. If we look at the affiliation attribute and set the time interval Δt to be less than 7 years, then r_1, r_4, r_7, r_8 are mutant records of entity e_1 on the affiliation attribute, and time points 2005, 2007, and 2009 are the mutation points of entity e_1 respectively. However, if we set Δt to be greater than or equal to 7 years, then only r_1, r_4 and r_8 are mutant records of entity e_1. If we look at the co-authors attribute, then r_1 and r_4 would be the only mutant records on the co-author attribute if Δt is between 2 to 4 years. In addition, if Δt is greater than 4 years, then only the first record r_1 would be the mutant record on the co-author attribute.* □

We now introduce the mutation function $\mathcal{M}_A(R,t)$:

DEFINITION 3.3. (MUTATION FUNCTION). *Let A be an attribute, $R = \{r_1, ..., r_n\}$ be a list of records associated with an entity E, and t a specified time. The mutation function $\mathcal{M}_A(R,t)$ returns the probability of entity E having a mutant record on attribute A at time t given the record history R of entity E.* □

While an upper-bound time interval Δt could be included in the mutation function, here we omit this parameter and set it to infinity for simplicity. This makes the mutation function $\mathcal{M}_A(R,t)$ return the probability that the entity associated with the input list R of records has an A-value at time t that is different from all the A-values in R.

In the rest of this section, we will introduce how we learn the mutation function from a training data set based on statistics about attribute value recurrence. How to apply the mutation function in similarity computation will be described in the next section (Section 4).

3.3 Attribute Value Recurrence

To learn the mutation function, we collect the statistics of attribute value recurrence from the given training data set. Here we first introduce the notion of *recurrence hit* and *miss*:

DEFINITION 3.4. (Δt RECURRENCE HIT). *We say that a value v has a Δt* recurrence hit*, or Δt* recurrence *for short, if v satisfies following conditions: 1) value v occurs in some entity E on attribute A, and 2) v recurs Δt time units later in the same entity E.* □

DEFINITION 3.5. (Δt RECURRENCE MISS) *We say that a value v has a Δt* recurrence miss *on attribute A if v satisfies following conditions: 1) value v occurs in some entity E on attribute A, and 2) v does not recur in E over the next Δt time units.* □

EXAMPLE 3.2. *Consider an example from DBLP shown in Table 1 and look at the value 'Licheng Jiao' on the co-authors attribute. We can see that value 'Licheng Jiao' appears in year 1999, 2000, 2001, 2005, 2006, 2007, 2009, and 2010. By the definition of recurrence, 'Licheng Jiao' has five 1-year recurrences, three 2-year recurrences (1999 to 2001, 2005 to 2007 and 2007 to 2009), and several other recurrences.* □

Now we describe how we learn the statistics of attribute value recurrence. Without loss of generality, we may assume all attributes are set-valued attributes by viewing each single-valued attribute as a set-valued attribute having a single element, and we will use the notation described in Section 2.2.

Given a training data set $\mathbf{E}$, where records are grouped according to their associated entities, we learn the statistics of value recurrence on attribute A as follows:

Step 1: For each entity $E = \langle r_1, r_2, ...\rangle$ with its records sorted in increasing temporal order, maintain the *occurrence history* $E.t_{v\in A} \subseteq E.t$ for each value $v \in E.A$ that appears in E on attribute A. Each occurrence history $E.t_{v\in A} = \langle t_1, t_2, ...\rangle$ is an ordered list of time stamps sorted in increasing temporal order, where each time stamp describes a time point when entity E has value v on attribute A. For example, consider the entity shown in Table 1. Its occurrence history of value 'Licheng Jiao' on the co-authors attribute would be $\langle$1999, 2000, 2001, 2005, 2006, 2007, 2009, 2010$\rangle$.

Step 2: For each occurrence history $E.t_{v\in A}$, identify and fill in possible missing occurrences. For each consecutive pair of time stamps t_i, t_{i+1} in $E.t_{v\in A}$, if entity E does not have any records with time stamp within (t_i, t_{i+1}), then we assume that entity E has its A-value equal to v from time t_i to t_{i+1} and insert all possible time stamps $t_i < t < t_{i+1}$ into $E.t_{v\in A}$. For example, consider the example in Table 1. Since 'Licheng Jiao' is listed in the co-author attribute of entity e_1 in 2001 and 2005, and e_1 has no records from 2002 to 2004, we shall not exclude the possibility that e_1 in fact worked with 'Licheng Jiao' from 2001 to 2005. Therefore, the missing occurrences of 'Licheng Jiao' between 2002 and 2004 should be filled in.

Step 3: Count and aggregate the numbers of hits $h^A_{\Delta t}$ and misses $m^A_{\Delta t}$ of Δt-recurrences on attribute A from the updated occurrence histories of all attribute values for each $0 < \Delta t \leq t_{max}$. This can be done by using a sliding window that iterates through each occurrence history.

Step 4: For each Δt, smooth the hit and miss counts of its recurrence by applying a Gaussian filter with variance in proportional to Δt.

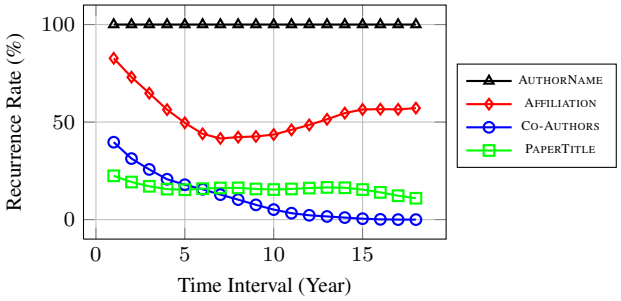


Figure 3: The results of learning the probability of Δt recurrence from the DBLP-Ambiguous data set on different attributes. Note that both paper title and co-authors attributes are processed as set-valued attributes.

Step 5: Construct the recurrence function $\mathcal{R}_A(\Delta t)$ based on the following equation:

$$\mathcal{R}_A(\Delta t) = \begin{cases} 1 & \Delta t = 0 \\ \dfrac{h^A_{\Delta t}}{(h^A_{\Delta t} + m^A_{\Delta t})} & 1 \leq \Delta t \leq t_{max} \\ \dfrac{h^A_{\Delta t_{max}}}{(h^A_{\Delta t_{max}} + m^A_{\Delta t_{max}})} & \Delta t \geq t_{max} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

We set $\mathcal{R}_A(\Delta t) = \mathcal{R}_A(\Delta t_{max})$ for all $\Delta t > \Delta t_{max}$ for simplicity.

EXAMPLE 3.3. *Consider Table 1 again, where we would like to learn the recurrence function of the affiliation attribute. Suppose we are currently processing value "Xidian University" and would like to count the recurrence hits and misses for one- and two-year time intervals. Then we will learn $(3, 2)$ and $(2, 2)$ recurrence hits and misses for one- and two-year time intervals respectively. Figure 2 shows the occurrence history of value "Xidian University" and its recurrence hits and misses for one- and two-year time intervals.* □

Figure 3 shows the curves of recurrence rates on four different attributes: author's name, author's affiliation, paper title (segmented into terms) and co-authors, learned from the DBLP-Ambiguous data set (described in detail in Section 5). We observe that 1) the proposed model is able to learn different evolution types; 2) authors in this data set never change their names; 3) authors are more likely to move back to their previous affiliations after publishing papers for many years; 4) the probabilities of working with the same scholar and working on similar topics decays over time, but their speed of these decays differ.

We give the algorithm for learning attribute value recurrence in Algorithm 1.

3.4 Capturing Entity Mutation

We now discuss how to construct the mutation function $\mathcal{M}_A(R, t)$. We start by considering single values and then extend the idea to consider multiple values.

Consider the case where we compute part of the mutation function by looking at only one value v on attribute A given a list R of records associated with the same real world entity. We use $\mathcal{M}_{v\in A}(R, t)$ to denote the simplified mutation function that only returns the probability of the entity described by R *not* having its A-value equal or contain v at time t. To compute such an entity-dependent probability, we first consider all records in R that have

Algorithm 1 LEARNVALUERECURRENCE($\mathbf{E}$, A, Δt_{max})

Input:

- $\mathbf{E}$, a set of entities, where each entity $E = \langle r_1, ..., r_{|E|} \rangle$ consists of a list of records sorted in increasing temporal order.
- A, the attribute of interest.
- Δt_{max}, the maximum time interval considered.

Output:

- $\mathcal{R}_A(\Delta t)$, the recurrence function.

// Initialize the counters for Δt recurrence hit / miss.
for $\Delta t = 1$ to t_{max} **do**
  $h^A_{\Delta t} \leftarrow 0, m^A_{\Delta t} \leftarrow 0$
end for
$C^A \leftarrow \{\langle h^A_1, m^A_1 \rangle ..., \langle h^A_{\Delta t_{max}}, m^A_{\Delta t_{max}} \rangle\}$
for all $E \in \mathbf{E}$ **do**
  $E.A \leftarrow \{\}$ // The set of ever-occurred A-values in E.
  $E.t \leftarrow \langle\rangle$ // The list of ever-occurred time stamps in E.
  for all $r_i \in E$ **do**
    $E.t = E.t \cup r_i.t$
    for all $v \in r_i.A$ **do**
      if $v \notin E.A$ **then**
        $E.A \leftarrow E.A \cup v$
        $E.t_{v \in A} \leftarrow \langle\rangle$ // Init the occurrence history for v.
      end if
      // Update the occurrence history.
      $E.t_{v \in A} \leftarrow E.t_{v \in A} \cup r_i.t$
    end for
  end for
  for all $v \in E.A$ **do**
    // Functions are defined in Algorithm 2. and 3.
    $E.t_{v \in A} \leftarrow$ UPDATEHISTORY($E.t_{v \in A}, E.t$)
    // Update the recurrence counts ($h^A_{\Delta t}$ and $m^A_{\Delta t}$) in C^A.
    $C^A \leftarrow$ UPDATECOUNTS($E.t_{v \in A}, C^A, \Delta t_{max}, \max(E.t)$)
  end for
end for
// Construct the recurrence function $\mathcal{R}_A(\Delta t)$.
$\mathcal{R}_A \leftarrow \{\langle 0, 1 \rangle\}$
$\mathcal{R}_A \leftarrow \mathcal{R}_A \cup \{ \langle \Delta t, \frac{h^A_{\Delta t}}{(h^A_{\Delta t} + m^A_{\Delta t})} \rangle | \ 1 \leq \Delta t \leq \Delta t_{max} \}$
$\mathcal{R}_A \leftarrow \mathcal{R}_A \cup \{ \langle \Delta t, \frac{h^A_{\Delta t_{max}}}{(h^A_{\Delta t_{max}} + m^A_{\Delta t_{max}})} \rangle | \ \Delta t > \Delta t_{max} \}$
$\mathcal{R}_A \leftarrow \mathcal{R}_A \cup \{ \langle \Delta t, 0 \rangle | \ \Delta t < 0 \}$
return $\mathcal{R}_A$

their A-value equal to v and time stamps earlier than t. Then, we view each such record r as a "predictor" and invoke the recurrence function with time interval $t - r.t$ to obtain the probability that value v reappears at time t given that v appears at time $r.t$. Finally, we combine all the probabilities estimated by each predictor record.

How to best combine the predictions from the predictor records is an interesting question. The combination should be a probability (between zero and one); also, the combination probability that a value appears plus the combination probability that a value does not appear should equal one. A possible approach that satisfies this would be to average the probabilities from all the predictors. But this is somehow unsatisfying; for one thing, a sum tends to lessen the impact of variations in small probabilities (for example, if most

Algorithm 2 UPDATEHISTORY($E.t_{v \in A}, E.t$)

Input:

- $E.t_{v \in A}$, time stamps of v's occurrence of E in increasing temporal order.
- $E.t$, time stamps of entity E in increasing temporal order.

Output:

- $E.t_{v \in A}$, the updated time stamps of v's occurrence.

for all $t_i \in E.t_{v \in A}$ **do**
  // If true, then E has no records with
  // its time stamp between t_{i-1} and t_i.
  if $i > 1$ and $\neg \exists t \in E.t$ s.t. $t_{i-1} < t < t_i$ **then**
    // Insert the missing time stamps of occurrence.
    for $t = t_{i-1} + 1 \rightarrow t_i - 1$ **do**
      $E.t_{v \in A} \leftarrow E.t_{v \in A} \cup t$
    end for
  end if
end for
return $E.t_{v \in A}$

Algorithm 3 UPDATECOUNTS($E.t_{v \in A}, C^A, \Delta t_{max}, t_{E_{latest}}$)

Input:

- $E.t_{v \in A}$, time stamps of v's occurrence of entity E in increasing order.
- $C^A = \{\langle h^A_1, m^A_1 \rangle ..., \langle h^A_{\Delta t_{max}}, m^A_{\Delta t_{max}} \rangle\}$, $h^A_{\Delta t}$ and $m^A_{\Delta t}$ are the Δt-recurrence hit and miss counters respectively.
- Δt_{max}, the maximum time difference considered.
- $t_{E_{latest}}$, the latest time stamp of entity E.

Output:

- C^A, the updated C^A.

// update the recurrence hits $h^A_{\Delta t}$ and misses $m^A_{\Delta t}$
// stored in C_A for all possible Δt.
for all $t \in E.t_{v \in A}$ **do**
  for $\Delta t = 1$ to Δt_{max} **do**
    if $t + \Delta t \leq t_{E_{latest}}$ **then**
      if $t + \Delta t \in E.t_{v \in A}$ **then**
        $h^A_{\Delta t} \leftarrow h^A_{\Delta t} + 1$
      else
        $m^A_{\Delta t} \leftarrow m^A_{\Delta t} + 1$
      end if
    end if
  end for
end for
return C^A

of the probabilities are close to one, the impact of an outlier small probability changing from say 0.01 to 0.02 is negligible.)

Accordingly, instead we use the product of the probability that all of the predictors will say "no" (the value will not recur) normalized by the sum of the probability that all predictors will say "yes" and the sum of the probability that all predictors will say "no." We

express this more precisely below.

$$\mathcal{M}_{v\in A}(R,t) = \frac{\prod\limits_{r\in R_{v,t}} (1-\mathcal{R}_A(t-r.t))}{\prod\limits_{r\in R_{v,t}} \mathcal{R}_A(t-r.t) + \prod\limits_{r\in R_{v,t}} (1-\mathcal{R}_A(t-r.t))} \quad (7)$$

$$R_{v,t} = \{r | r \in R \wedge r.t < t \wedge v \in r.A\} \quad (8)$$

The denominator of the above equation contains two products: the left product computes the probability of all observer records in $R_{v,t}$ saying that value v will reappear at time t, while the other product on the right computes the probability of all observer records in $R_{v,t}$ saying that value v will not reappear at time t.

Note that the overall goal is to improve matching by deciding weights for attributes. The important thing here is the relative weights of the attributes, not the absolute values of the weights.

Intuitively, the general form of the mutation probability $\mathcal{R}_A(R,t)$ can be computed by considering all possible values $v \in R.A$ (using the notation defined in Equation 2) that have appeared in at least one of the records in R:

$$\mathcal{M}_A(R,t) = \prod_{v\in R.A} \mathcal{M}_{v\in A}(R,t) \quad (9)$$

The computation of mutation function could be expensive. This can be somewhat amortized by caching the result and reusing it for each R during the clustering process. When R is updated by including one additional record, the result can be incrementally updated by further maintaining the occurrence history of different values for each cluster.

3.5 An Approximation of Mutation Function

While caching improves the computational cost of our algorithm, in some cases it may still be too high. Therefore, we propose the following approximation, which essentially treats all different values in $R.A$ as the same value instead. This relaxes the need for maintaining and iterating through the occurrence history of each attribute value. Again, without loss of generality, we assume each attribute is a set-valued attribute by treating each single-valued attribute as a set-valued attribute with one value.

$$\mathcal{M}_A(R,t) = \frac{\prod\limits_{r\in R_t} (1-\mathcal{R}_A(t-r.t))^{|r.A|}}{\prod\limits_{r\in R_t} \mathcal{R}_A(t-r.t)^{|r.A|} + \prod\limits_{r\in R_t} (1-\mathcal{R}_A(t-r.t))^{|r.A|}} \quad (10)$$

$$R_t = \{r | r \in R \wedge r.t < t\} \quad (11)$$

where $|r.A|$ is the number of values on attribute A in r.

The above approximation looks very similar to the single-valued mutation function (Equation 7) as it treats different values of $R.A$ as the same value. In addition, the iterating of each individual value $v \in R.A$ is replaced by simply counting the number of elements in $R.A$. Note that as we will later discuss in this section, the full version (Equation 9 and 7) and the approximated version (Equation 10) have identical numerators. The difference is in the denominator, where the approximated version has a value no greater than that of the full version. In addition, when the target attribute is a single-valued attribute, the two versions will be identical. Our experiments (described in Section 5.5) show that the two versions of mutation functions do not have noticeable differences in result quality, but the approximated version can substantially improve running time.

Here we further discuss the difference between the full version and the approximated version of the mutation function.

We first look at the numerators of the two versions. In the original mutation function, the numerator is the product of the numerators of $\mathcal{M}_{v\in A}(R,t)$ of all different values in $R.A$, which can be rewritten as follows:

$$\prod_{v\in R.A} \prod_{r\in R_{v,t}} (1-\mathcal{R}_A(t-r.t)) = \prod_{r\in R_t} \prod_{v\in r.A} (1-\mathcal{R}_A(t-r.t)) \quad (12)$$

where both sides of the above equation enumerate all possible values on attribute A in R which time stamp is greater than t. Since the values of any record r on attribute A will have the same time stamp, the above equation can be simplified further:

$$\prod_{r\in R_t} \prod_{v\in r.A} (1-\mathcal{R}_A(t-r.t)) = \prod_{r\in R_t} (1-\mathcal{R}_A(t-r.t))^{|r.A|} \quad (13)$$

where the right hand side of the equation gives the numerator of the approximated mutation function (Equation 10), which further shows that both the full version and the approximated version of the mutation function have identical numerator.

Now we turn our focus on their denominators. In the full version of the mutation function, its denominator is the product of the denominators of the $\mathcal{M}_{v\in A}(R,t)$, which can be expended and rewritten as follows:

$$\prod_{v\in R.A} \left(\prod_{r\in R_{v,t}} \mathcal{R}_A(t-r.t) + \prod_{r\in R_{v,t}} (1-\mathcal{R}_A(t-r.t)) \right) = \prod_{v\in R.A} \prod_{r\in R_{v,t}} \mathcal{R}_A(t-r.t) + \prod_{v\in R.A} \prod_{r\in R_{v,t}} (1-\mathcal{R}_A(t-r.t))) + \text{ remaining terms.} \quad (14)$$

where on the right hand side, the first term is the product of all possible recurrence rates $\mathcal{R}_A(t-r.t)$ reported by the observer records in R and the second term is the product of the complements of all possible recurrence rates $(1-\mathcal{R}_A(t-r.t))$ reported by the observer records in R. These two terms forms the denominator of the approximated mutation function.

3.6 Application to Fuzzy Clustering

In fuzzy clustering algorithms, the probability $p(r \in R)$ of each record belonging to a cluster is "fuzzy" (between 1 to 0) in the same sense as fuzzy logic. This probability is determined at run time by a fuzzy clustering algorithm, and a record could be put into multiple clusters with differing probabilities for each. To extend our model to work for fuzzy clustering algorithms, we further weight the importance of each observer record based on the probability $p(r \in R)$ of each record belonging to its cluster. Below is the modified version of the mutation function that works for fuzzy clustering techniques:

$$\mathcal{M}_{v\in A}(R,t) = \frac{\prod\limits_{r\in R_{v,t}} p(r\in R)(1-\mathcal{R}_A(t-r.t))}{\prod\limits_{r\in R_{v,t}} p(r\in R)\mathcal{R}_A(t-r.t) + \prod\limits_{r\in R_{v,t}} p(r\in R)(1-\mathcal{R}_A(t-r.t))} \quad (15)$$

$$\mathcal{M}_A(R,t) = \prod_{v\in R.A} \mathcal{M}_{v\in A}(R,t) \quad (16)$$

where we reuse the previous notation $R_{v,t}$ defined in Equation 8.

Similarly, we extend the approximated version of the mutation function as follows:

$$\mathcal{M}_A(R,t) = \frac{\prod_{r\in R_t} p(r\in R)(1-\mathcal{R}_A(t-r.t))^{|r.A|}}{\prod_{r\in R_t} p(r\in R)\mathcal{R}_A(t-r.t)^{|r.A|} + \prod_{r\in R_t} p(r\in R)(1-\mathcal{R}_A(t-r.t))^{|r.A|}} \quad (17)$$

and again, the definition of R_t can be found in Equation 11.

4. APPLYING OUR TEMPORAL MODEL

We now describe how we utilize the proposed mutation model to weight the importance of each attribute in similarity computations. The weight is computed based on the entity-dependent probability of temporal agreement and disagreement. Then, we will introduce how we compute the similarity between two records.

We will use the following example task during the discussion.

EXAMPLE 4.1. *Consider a pair of records r and r'. We would like to compute their similarity based on attribute value similarities. We use $sim_A(r.A, r'.A)$ to denote their value similarity on attribute A. Without loss of generality, we assume that r' has a time stamp no later than the time stamp of r, and let $\Delta t = r'.t - r.t$ be the time difference between r and r'. In addition, we assume that record r is currently grouped into cluster C_r that consists of a list $R_r = \{r_1, ..., r_n\}$ of n records.* □

Given two records r and r', we ask the following two questions: 1) if r and r' have dissimilar values on attribute A, what is the probability that the value $r.A$ in the entity described in cluster C_r does not recur at time $r'.t$? 2) if r and r' have similar values on attribute A, what is the probability that the value $r.A$ does not recur in the entity described by cluster C_r at time $r'.t$? To answer the above questions, we utilize the value-based mutation function $\mathcal{M}_{v\in A}$ defined in Equation 7 to determine the probability of all values in $r.A$ *not* reappearing at time $r'.t$:

$$\mathcal{M}_{v\in r.A}(R_r, r'.t) = \prod_{v\in r.A} \mathcal{M}_{v\in A}(R_r, r'.t) \quad (18)$$

If the approximate version is used, then we use Equation 10 to approximate $\mathcal{M}_{v\in r.A}(R_r, r'.t)$.

To answer the first question, if the value in $r.A$ is not likely to reappear at time $r'.t$, the fact that $r.A$ and $r'.A$ do not have similar values should not be construed as suggesting that r and r' do not refer to the same entity. As a result, a lower weight will be assigned to attribute A.

On the other hand, in the second question, if we are expecting that the values in $r.A$ will not reappear at time $r'.t$, then we are not expecting to see a record belonging to C_r having its A-value similar to $r.A$ at time $r'.t$. So, if $r.A$ and $r'.A$ match in this case, it is likely from between-entity temporal agreement, and thus we should still assign a lower weight to A.

Putting it all together, we use the following equation to determine the weight of each attribute:

$$w_A(r,r') = \begin{cases} 1 + \vartheta_{\mathcal{M}} \cdot (1 - \mathcal{M}_{v\in r.A}(R_r, r'.t)) & sim(r.A, r'.A) \geq \theta_A \\ 1 - \vartheta_{\mathcal{M}} \cdot \mathcal{M}_{v\in r.A}(R_r, r'.t) & \text{otherwise} \end{cases} \quad (19)$$

where $\vartheta_{\mathcal{M}}$ is a parameter that controls the importance of the mutation function, and θ_A again is a learned threshold indicating high similarity on attribute A.

Here we have used the heuristic that the weights given to the attributes where the two records agree on their values are always higher than the weights given to the attributes where the two records disagree. This heuristic ensures that the final record similarity will never be dominated by several low similarity attributes if at least one high similarity attribute is observed.

Finally, the temporal similarity between two records r and r' is defined to be the weighted average of their attribute value similarities based on the weights derived from our mutation model:

$$sim(r,r') = \frac{\sum_{A\in\mathbf{A}} w_A(r,r') \cdot sim(r.A, r'.A)}{\sum_{A\in\mathbf{A}} w_A(r,r')} \quad (20)$$

5. EXPERIMENTAL EVALUATION

This section describes our experimental setup and results.

5.1 Experiment Settings

5.1.1 Data Sets

To evaluate the matching quality of different approaches, we consider three subsets from two real world data sets: a benchmark of European patent data [1] and the DBLP data set [2]. Two of the tasks, Euro-Patent and DBLP-WW, are the tasks used in Li's work [14].

DBLP-Ambi matching task: We created a labeled subset from DBLP by selecting 21 different groups of records which contain 2664 records in total. Each group contains multiple authors sharing the same name. Examples are "Ajy Gupta", "Li Zhang", "Rajesh Kumar", "Min-Soo Kim", "Arnab Roy", and so forth. Again, in this matching task, the author's affiliation information is manually filled in for each record.

Euro-Patent matching task: From the patent data, which is also used in [14], we extracted inventor records with attributes name and address; the time stamp of each record is the patent filing date. The benchmark involves 359 inventors of French patents. For purpose of comparison, we also followed the practice in [14] of using only last name initial and first name for each inventor.

DBLP-WW matching task: From the DBLP data, we used the labeled "Wei Wang" subset also used in [14] containing 738 records. The data provider has manually identified 18 authors for ground truth and filled in the author affiliation attribute for each record. One important property of this matching task is that all authors have the same name, *Wei Wang*.

Table 2 briefly summarizes the matching tasks used in our experiment.

5.1.2 Implementation

We consider the following approaches for handling temporal agreement and disagreement:

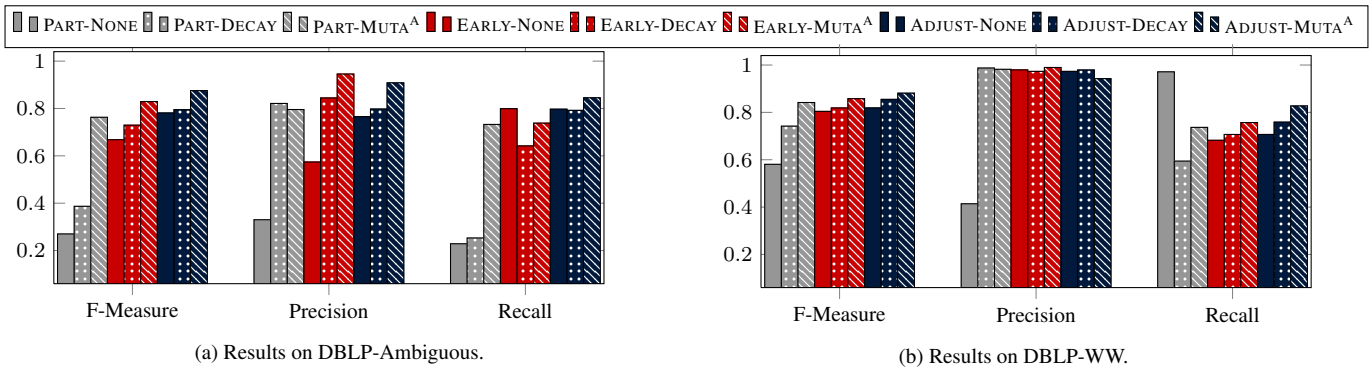
- -NONE: This approach does not use any strategy to handle temporal agreement and disagreement.
- -DECAY: This is the time decay model proposed in [14].
- -MUTA$^{\text{A}}$: This is the approximate version of our proposed model (Eq. 10, described in Section 3).
- -MUTA$^{\text{F}}$: This is the full version of -MUTA$^{\text{A}}$ (Eq. 7 and Eq. 18, described in Section 3 and Section 4). This model is only used for analyzing the effectiveness of our proposed approximation.

Recall that in general a temporal matching approach requires both a temporal model and a clustering algorithm. In our experiments we tried three different clustering algorithms, testing each with all four temporal models. The clustering algorithms used were:

- PART: The partition algorithm proposed in [10], a single-stage approach that does not cluster in any temporal order.

Table 2: Temporal matching tasks

Name	Total # Records	# Pairwise Record Relations	Attributes	Years	Note
DBLP-Ambi	2664	3.5 million	author's name, affiliation (manually filled), paper title, co-authors	1987 - 2012	258 authors share only 21 names; most entities evolve.
DBLP-WW	738	0.3 million	author's name, affiliation (manually filled), co-authors	1991 - 2010	All authors share the same name; most entities evolve. also used in [14]
Euro-Patent	1871	1.7 million	author first name initial, author last name, affiliation	1978 - 2003	Fewer entities evolve; also used in [14].



(a) Results on DBLP-Ambiguous.

(b) Results on DBLP-WW.

Figure 4: Result quality on DBLP matching tasks.

- EARLY: The early binding algorithm [14], a single-stage temporal clustering approach that considers records in increasing temporal order and uses a single pair of records when making cluster merging decisions.
- ADJUST: The adjusted binding algorithm [14], the leading clustering approach in that paper, a multi-stage temporal clustering approach which uses an EM-style fuzzy clustering to iteratively refine the temporal clustering results.

All the techniques were implemented as single-threaded Java programs, and we ran the experiments on a Linux machine with a 2.40 GHz Intel CPU and 1GB of RAM.

Similarity measure: We applied edit distance similarity [13] for all scalar attributes. For multi-valued attributes, we did not use the regular Jaccard similarity [11] because the records in our matching tasks rarely share more than one or two elements on the multi-valued attributes we considered (e.g., co-authors). As a result, the regular Jaccard similarity usually reports a low similarity score even when two records show substantial match on the multi-valued attributes in our matching tasks. Instead, we use the following variation on Jaccard similarity, where the denominator is the size of the smaller set:

$$sim_A(r.A, r'A) = \frac{|r.A \cap r'.A|}{min(|r.A|, |r'.A|)} \quad (21)$$

This variation returns the percentage of the smaller set covered by the larger.

Models and thresholds learning: We used a cross-validation process to learn the temporal models and the thresholds for each matching task. We used a three-fold cross-validation, which divides the data set into three disjoint partitions with nearly equal size. We learned temporal models and thresholds from one partition at a time and used them to test with the rest of the data set.

Parameter settings: For the time decay model and the temporal clustering algorithms proposed in [14], we followed the parameter settings that were used in their experiment. For the parameters that used in our model, we set $\theta_{\mathcal{M}} = 0.5$ (used in Eq. 19) in our experiments.

5.1.3 Metrics

We compared pairwise matching decisions with the ground truth and measured the quality of the result by *precision* (P), *recall* (R), and *F-measure* (F). We denote the set of false positive pairs by F_N, the set of false negative pairs by F_N, and the set of true positive pairs by T. Then, $\mathrm{P} = \frac{|T|}{|T|+|F_P|}$, $\mathrm{R} = \frac{|T|}{|T|+|F_N|}$, and $\mathrm{F} = \frac{2\mathrm{PR}}{\mathrm{P+R}}$.

5.2 Matching Accuracy

Here we explore and discuss the matching accuracy of different approaches on the three matching tasks.

5.2.1 DBLP-Ambiguous Matching Task

In this matching task, 258 entities share only 21 different names, and most author entities evolve on all other attributes: affiliations, paper titles, and co-authors. As a result, the key to high matching accuracy is to resolve the ambiguity on name by handling evolution on all other attributes. Figure 4a shows the matching accuracy of different clustering algorithms paired with different temporal models. Note that since we do not profile MUTA^F, three clustering algorithms with three temporal models gives nine approaches. Higher is better in this graph.

From the results, we first observe the pure non-temporal technique PART-NONE only produces 27.01% in F-1 score. However, when being paired with a temporal model, PART can produce a higher matching accuracy: a 38.68% in F-1 score when paired with the time decay model and a 76.28% in F-1 score when paired with our proposed model — MUTA^A. Second, applying a temporal model to a clustering algorithm in general improves F-1 score: our model MUTA^A improves 10% to 50% in F-1 score compared with no temporal model while the time decay mode — DECAY improves from 2% to 12% in F-1 score as compared with no temporal model.

The readers might wonder at this point why a non-temporal matching technique might produce a lower precision than a temporal matching technique. It is because without considering temporal

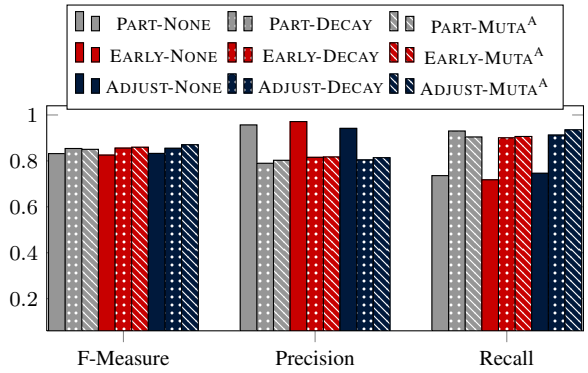


Figure 5: Result quality on the Euro-Patent matching task.

information, a non-temporal matching technique might incorrectly merge two records which in fact refer to two different entities caused by between-entity temporal agreement. This contributes to its low precision.

In addition to the case where authors might move back to their "home affiliations", we also observed the following cases where our mutation model does better than the time decay model: 1) an author might have out-of-date affiliation associated with his / her publications for one or two years when he / she moves from one affiliation to another; 2) publications of the same author are usually associated with different co-authors over time while many of his / her co-authors re-appear in that author's publication history over time.

Comparing our mutation model with the time decay model, our approach improves 40% in matching accuracy over the time decay model when a non-temporal clustering technique is used and improves 8% to 10% in matching accuracy over the time decay model when temporal clustering approaches are used.

5.2.2 *DBLP-WW Matching Task*

The DBLP-WW matching task evaluates how each approach handles the special case when all author entities in the same data set share the same name. In addition, author entities also evolve on all other attributes. Figure 4b compares the result qualities of different clustering techniques paired with different models for handling evolution and ambiguity.

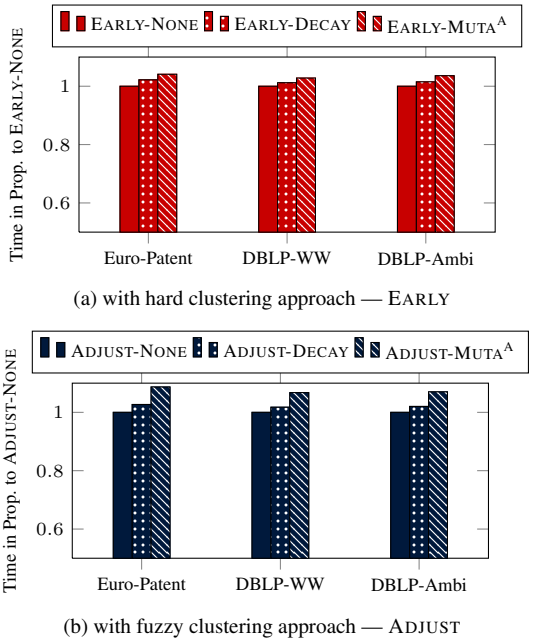
Similar to what we have seen in the DBLP-Ambiguous matching task, applying a temporal model will improve matching accuracy over applying no temporal model: The proposed MUTA^A improves 5% to 25% in F-1 score over using no temporal model, while the state-of-the-art model — DECAY — improves 2% to 15% in F-1 score over using no temporal model.

Compared with the state-of-the-art temporal model — DECAY, our model MUTA^A improves up to 10% in F-1 score in the DBLP-WW matching task.

5.2.3 *Euro-Patent Matching Task*

Unlike the previous two matching tasks, there are relatively few entities described in the Euro-Patent matching set whose attributes evolve over time. This situation has two implications. First, as this matching task has fewer evolving entities, all approaches have less trouble separating records belonging to the same entity. Second, because fewer entities evolve over time, there are relatively fewer temporal clues that any temporal model can utilize. Figure 5 shows the matching accuracy of different approaches.

First, as we have discussed, since there are fewer evolving entities, the non-temporal solution — PART-NONE — performs relatively better than it did in the previous two matching tasks: 83.17% in F-1 score with 95.65% precision and 73.58% recall. Second, temporal cluster approaches alone without using a temporal model do not bring noticeable improvement on accuracy ($\leq$ 1% improvement). Third, we observe that while not many temporal clues are available in this matching task, using a temporal model still brings some improvement in matching accuracy: our model MUTA^A improves 2% to 4% in F-1 score over using no temporal model while the decay model improves 2% to 3% in F-1 score over using no temporal model.



(a) with hard clustering approach — EARLY

(b) with fuzzy clustering approach — ADJUST

Figure 6: Running time comparison.

5.2.4 *Summary*

When most entities described in the data set evolve over time, our proposed model improves 10% to 50% in matching accuracy over using no temporal model while the state-of-the-art temporal model improves 2% to 15% over using no temporal model. When only a few of the entities described in the data set evolve over time, the temporal models provide a limited improvement (2% to 4%) in accuracy over using no temporal model.

Compared with the state-of-the-art temporal model, our model improves up to 40% in F-1 score when a non-temporal clustering algorithm is used, and up to 10% in F-1 score when a temporal clustering algorithm is used.

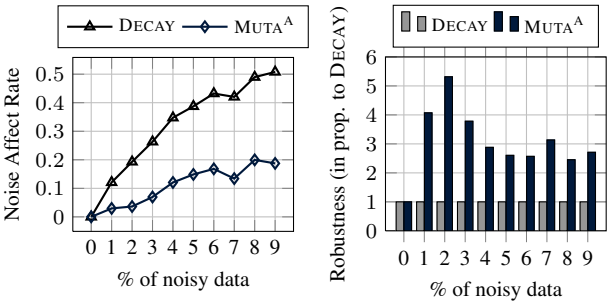
5.3 Results on Running Time Efficiency

We analyzed the running time efficiency of different temporal models by running them with different clustering approaches. As different clustering techniques spend different amounts of time finishing different matching tasks, we treat the running time of using no temporal model as 1.0 and report the running time of using temporal models in proportion to that.

The results in Figure 6 show that both DECAY and MUTA^A introduce acceptable running time overhead compared to a solution without using any temporal model. Compared with DECAY, our model MUTA^A introduces 2% running time overhead when running with a hard clustering algorithm (EARLY) and 5% running time overhead when running with a fuzzy clustering algorithm (ADJUST). The difference between hard- and fuzzy- clusterings is due to the

Table 3: Attributes and the types of their simulated noise.

Attribute	Considered types of simulated noise
Name	typo
Affiliation	typo, missing value, incorrect reference
Co-authors	typo, missing value
Publication Year	numerical error



(a) Noise affect rate.

(b) Robustness to noise.

Figure 7: Results on robustness to noise.

fact that clusters in the fuzzy clustering tend to have more frequent updates. As a result, the caching strategy used in our model will become less effective.

5.4 Results on Noise Tolerance

To test the impact of noise on different temporal models, we simulated the following and added them to the original data sets:

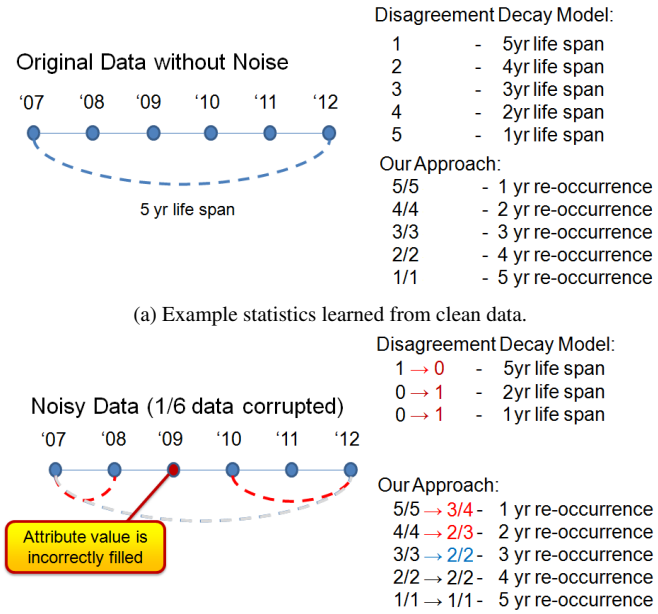
- **typo**: simulated by randomly replacing a small number of characters by their keyboard neighbors. Characters include numbers.
- **missing value**: simulated by deleting the whole value.
- **incorrect reference**: simulated by replacing the original value by a completed random value.
- **numerical error**: simulated by adding or subtracting a small number in proportion to the original value.

Table 3 lists the types of noise we added to different attributes in our matching tasks.

We analyze the noise tolerance of different temporal modeling approaches by comparing their learned model, which essentially can be described as a sequence of numeric values, from the clean data set and the one from the noisy data set and measuring the difference. We here define the noise affect rate as the difference in proportion to the original value sequence. Let $V_c = \langle v_{c,1}, ..., v_{c,t} \rangle$ and $V_n = \langle v_{n,1}, ..., v_{n,t} \rangle$ be two sequences of values learned by the same temporal model from the original clean data set and the noisy data set respectively, where t is the maximum time interval we considered. We define the noise affect rate as the sum of absolute difference between each pair of values over the sum of the values of the clean value sequence V_c:

$$\text{NOISEAFFECTRATE}(V_c, V_n) = \frac{\sum_{i=1}^{t} |v_{c,i} - v_{n,i}|}{\sum_{i=1}^{t} v_{c,i}} \quad (22)$$

Figure 7 compares the noise affect rates and the robustness against noise of the time decay model [14] (DECAY) and the proposed mutation model (MUTA^A). The robustness against noise is defined as the inverse of the noise affect rate, and the figure shows the normalized result where we treat the robustness of DECAY as 1.00.



(a) Example statistics learned from clean data.

(b) Example statistics learned from noisy data.

Figure 8: Learning from clean and noisy data.

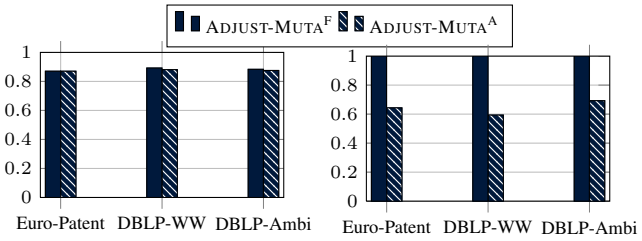
From the results shown in Figure 7, we observe that our proposed model has better noise resistance than the previously proposed model. Here are reasons that might contribute to this result. First, our model does not impose a continuous life span segmented by change points. This improves robustness to noise as any continuous life span could be easily corrupted by noise into multiple shorter life spans. Second, since our mutation model looks at multiple records when making its predictions, it is more robust than the decay model, which only looks at a single record.

Overall, our mutation model is 2X to 5X more robust against noise than the existing time decay model.

EXAMPLE 5.1. *Consider an example shown in Figure 8a where we have a value with 5-year life span in the original data. In this data, the disagreement decay model will simply learn one sample of a 5-year life span. However, suppose the attribute value of one record is incorrectly represented as shown in Figure 8b. Then the observed life span pattern will be corrupted and thus the disagreement decay model learns one sample of 2-years and one sample of 1-year life spans instead:* $1/6$ *records corrupted introduces significant differences in the learned result. In contrast, since our approach learns the mutation function based on the statistics about each individual value recurrence, such incorrect data only introduces a limited effect. In the clean data, our approach learns a* $\{5/5, 4/4, 3/3, 2/2, 1/1\}$ *of 1- to 5-year recurrence rates, while in the noisy data, our model learns a* $\{3/4, 2/3, 2/2, 2/2, 1/1\}$ *of 1- to 5-years recurrence rate:* $1/6$ *records corrupted introduce a 11.67% (less than 1/6) bias in the recurrence rate learned from surrounding records of the corrupted one.* □

5.5 Effectiveness of Approximation

Here we compare the result quality and running time between the full version of the mutation function and the approximated version of mutation function discussed in Section 3.4 by applying them to different temporal matching tasks and measuring the resulting quality and running time. Since the running time varies with the size of matching tasks, we treat the running time of the full version



(a) Result quality comparison (in F-1 score).

(b) Running time comparison (in prop. to ADJUST-MUTA^{F}).

Figure 9: Full vs. approximate versions of our model.

ADJUST-MUTA^{F} as 1.0 and report the running time of the approximated version in proportion to ADJUST-MUTA^{F}. The results are shown in Figure 9.

Our experimental results show that when using the approximated version of the mutation function, we can save around 30% to 40% in running time without losing noticeable level of result quality: the approximated version finishes 2664 records in 1320 seconds while the full version finishes in 1905 seconds.

6. RELATED WORK

[7, 9, 18] proposed different record matching / deduplication techniques, but their techniques all assume value difference are due to different representations of the same value and record values do not change over time. Several temporal data models [15] and temporal knowledge discovery paradigms [16] have been proposed in the past. However, their main focus are not on record matching. Yakout *et al.* proposed behavior based linkage [19] based on periodical behavior patterns of each entity. While our approach is based on the statistics about event recurrences, it does not try to find periodic entity evolution patterns. [17] proposed a recurrence based approach to learn and recognize complex temporal sequence, but their model is based on neural networks and Hebbian learning rules. Both [5], [6], and [14] proposed the notion of time decays, but their definition and application of time decay are different. [5] and [6] use time decay to reduce the effect of older tuples on data analysis. Li *et al.* [14] is the work closest to our work here, and we have discussed it in detail inline where appropriate.

7. CONCLUSION

We have proposed a new temporal model for handling entity evolution. Unlike existing models, our model focuses on the probability of a value re-appearing over time. Our experimental results on various real world temporal matching tasks show that our model improves both matching accuracy (up to 10% to 40%) and noise resistance (2X to 5X) over the state-of-the-art model while introducing minimal running time overhead ($\leq 2\%$).

Substantial room for future work remains. For example, entity evolution breaks the assumptions of blocking techniques [8]. An evolution aware blocking operator is needed for matching temporal records at large scale. Also, our model is of course only one point in a spectrum of models that seek to be more sophisticated in modeling evolution — it would be interesting to explore whether more complex models can yield better matching quality with acceptable overhead. Finally, in some sense temporal matching works by recognizing that the temporal dimension in a matching problem has special properties. It would be interesting to see if other dimensions can also be exploited — in particular, a special treatment of spatial information might be useful.

8. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under grant no. IIS-1018792.

9. REFERENCES

[1] Academic patenting in Europe (APE-INV). http://www.esf-ape-inv.eu/.

[2] The DBLP computer science bibliography. http://www.informatik.uni-trier.de/ ley/db/.

[3] Fec-standardizer - an experiment to standardize individual donor names in campaign finance data. https://github.com/cjdd3b/fec-standardizer.

[4] Twitter - an online social networking and microblogging service. https://twitter.com/.

[5] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. *Journal of Algorithms*, 59(1):19–36, 2006.

[6] G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu. Forward decay: A practical time decay model for streaming systems. In *IEEE 25th International Conference on Data Engineering (ICDE)*, pages 138–149. IEEE, 2009.

[7] P. Domingos. Multi-relational record linkage. In *Proc. of the KDD-2004 Workshop on Multi-Relational Data Mining*. KDD, 2004.

[8] A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, 2007.

[9] I. Fellegi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, pages 1183–1210, 1969.

[10] O. Hassanzadeh, F. Chiang, H. Lee, and R. Miller. Framework for evaluating clustering algorithms in duplicate detection. *Proceedings of the VLDB Endowment*, 2(1):1282–1293, 2009.

[11] P. Jaccard. *Distribution de la Flore Alpine: dans le Bassin des dranses et dans quelques régions voisines*. Rouge, 1901.

[12] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 802–803. ACM, 2006.

[13] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[14] P. Li, X. Dong, A. Maurino, and D. Srivastava. Linking temporal records. *Proceedings of the VLDB Endowment*, 4(11):956–967, 2011.

[15] G. Ozsoyoglu and R. Snodgrass. Temporal and real-time databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 7(4):513–532, 1995.

[16] J. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):750–767, 2002.

[17] D. Wang and M. A. Arbib. Complex temporal sequence learning based on short-term memory. *Proceedings of the IEEE*, 78(9):1536–1543, 1990.

[18] W. Winkler. Methods for record linkage and bayesian networks. Technical report, Statistical Research Division, US Census Bureau, Washington, DC, 2002.

[19] M. Yakout, A. Elmagarmid, H. Elmeleegy, M. Ouzzani, and A. Qi. Behavior based record linkage. *Proceedings of the VLDB Endowment*, 3(1-2):439–448, 2010.