Front. Comput. Sci.
DOI 10.1007/s11704-012-2002-5

RESEARCH ARTICLE

Linking temporal records

Pei LI (✉)[1], Xin Luna DONG[2], Andrea MAURINO[1], Divesh SRIVASTAVA[2]

1 University of Milan-Bicocca, Milan 20126, Italy
2 AT&T Labs-Research, Florham Park, NJ 07932, USA

Abstract Many data sets contain *temporal records* which span a long period of time; each record is associated with a time stamp and describes some aspects of a real-world entity at a particular time (e.g., author information in DBLP). In such cases, we often wish to identify records that describe the same entity over time and so be able to perform interesting longitudinal data analysis. However, existing record linkage techniques ignore temporal information and fall short for temporal data.

This article studies linking temporal records. First, we apply *time decay* to capture the effect of elapsed time on entity value evolution. Second, instead of comparing each pair of records locally, we propose clustering methods that consider time order of the records and make global decisions. Experimental results show that our algorithms significantly outperform traditional linkage methods on various temporal data sets.

Keywords temporal data, record linkage, data integration

1 Introduction

Record linkage takes a set of records as input and discovers which records refer to the same real-world entity. It plays an important role in data integration, data aggregation, and personal information management, and has been extensively studied in recent years (see [1,2] for recent surveys). Existing techniques typically proceed in two steps: the first step compares the similarity of each pair of records, deciding if they match or not; the second step clusters the records accordingly, with the goal that records in the same cluster refer to the same real-world entity and records in different clusters refer to different ones.

In practice, a data set may contain *temporal records* over a long period of time; each record is associated with a time stamp and describes some aspects of a real-world entity at that particular time. In such cases, we often wish to identify records that describe the same real-world entity over time and so be able to trace the history of that entity. For example, DBLP[1)] lists research papers over many decades; we wish to identify individual authors such that we can list all publications by each author. Other examples include medical data that keep patient information over tens of years, customer-relationship data that contains customer information over years, and so on; identifying records that refer to the same entity enables interesting longitudinal data analysis over such data [3].

Although linking temporal records is important, to the best of our knowledge, traditional techniques ignore the temporal information in linkage. Thus, they can fall short for such data sets for two reasons. First, the same real-world entity can evolve over time (e.g., a person can change their phone number and address) and so records that describe the same real-world entity at different times can contain different values; blindly requiring value consistency of the linked records can thus cause false negatives. Second, it is more likely to find highly similar entities over a long time period than at the same time (e.g., having two persons with highly similar names in the same university over the past 30 years is more likely than at the same time) and so records that describe different entities at different times can share common values;

Received January 01, 2012; accepted February 02, 2012

E-mail: pei.li@disco.unimib.it

1) http://www.dblp.org/

blindly matching records that have similar attribute values can thus cause false positives. We illustrate the challenges with the following example.

Example 1 Consider records that describe paper authors in Table 1; each record is derived from a publication record at DBLP (we skip some co-authors for brevity). These records describe 3 real-world people: r_1 describes E_1: *Xin Dong*, who was at *R. Polytechnic* in 1991; $r_2 - r_6$ describe E_2: *Xin Luna Dong*, who moved from *Univ of Washington* to *AT&T Labs*; $r_7 - r_{12}$ describe E_3: *Dong Xin*, who moved from *Univ of Illinois* to *Microsoft Research*.

If we require high similarity on both name and a liation, we may split entities E_2 and E_3, as records for each of them can have different values for a liation. If we require only high similarity of name, we may merge E_1 with E_2 as they share the same name, and may even merge all three entities.

Table 1 Records from DBLP

ID	name	a liation	co-authors	year
r_1	Xin Dong	R. Polytechnic Institute	Wozny	1991
r_2	Xin Dong	Univ of Washington	Halevy, Tatarinov	2004
r_3	Xin Dong	Univ of Washington	Halevy	2005
r_4	Xin Luna Dong	Univ of Washington	Halevy, Yu	2007
r_5	Xin Luna Dong	AT&T Labs-Research	Das Sarma, Halevy	2009
r_6	Xin Luna Dong	AT&T Labs-Research	Naumann	2010
r_7	Dong Xin	Univ of Illinois	Han, Wah	2004
r_8	Dong Xin	Univ of Illinois	Wah	2007
r_9	Dong Xin	Microsoft Research	Wu, Han	2008
r_{10}	Dong Xin	Univ of Illinois	Ling, He	2009
r_{11}	Dong Xin	Microsoft Research	Chaudhuri, Ganti	2009
r_{12}	Dong Xin	Microsoft Research	Ganti	2010

Despite the challenges, temporal information presents additional evidence for linkage. First, record values typically transition *smoothly*. In the motivating example, person E_3 moved to a new affiliation in 2008, but still had similar co-authors from previous years. Second, record values seldom change *erratically*. In our example, $r_2, r_7, r_3, r_8, r_{10}$ (time ordered) are very unlikely to refer to the same person, as a person rarely moves between two affiliations back and forth over many years (however, this can happen around transition time; for example, entity E_3 has a paper with the old affiliation information after he moved to a new affiliation, as shown by record r_{10}). Third, in case we have a fairly complete data set, such as DBLP, records that refer to the same real-world entity often (but not necessarily) observe *continuity*; for example, one is less confident that r_1 and $r_2 - r_6$ refer to the same person given the big time gap between them. Exploring such evidence would require a global view of the records with the time factor in mind.

This article studies linking temporal records and makes three contributions. First, we apply *time decay*, which aims to capture the effect of time elapse on entity value evolution (Section 3). In particular, we define *disagreement decay*, with which value difference over a long time is not necessarily taken as a strong indicator of referring to different real-world entities; we define *agreement decay*, with which the same value with a long time gap is not necessarily taken as a strong indicator of referring to the same entity. We describe how we learn decay from labeled data and how we apply it when computing similarity between records.

Second, instead of comparing each pair of records locally and then clustering, we describe three temporal clustering methods that consider records in time order and accumulate evidence over time to enable global decision making (Section 4). Among them, *early binding* makes eager decisions and merges a record with an already created cluster once it computes a high similarity; *late binding* instead keeps all evidence and makes decisions at the end; and *adjusted binding* in addition compares a record with clusters that are created for records with later time stamps.

Finally, we apply our methods on a European patent data set and two subsets of the DBLP data set. Our experimental results show that applying decay in traditional methods can improve the quality of linkage results, and applying our clustering methods can obtain results with high precision and recall (Section 5).

This article focuses on improving quality (precision and recall) of linking temporal records. We can further improve efficiency of linkage by applying previous techniques such as canopy [4] to create small blocks of records that are candidates for temporal linkage.

This article is an extended version of a previous conference paper [5]. We describe in detail more alternative solutions, including two additional decay learning algorithms, and the probabilistic adjusting algorithm for adjusted binding and present experimental results comparing them. We also give detailed proofs and describe more experimental results showing the effectiveness and robustness of our results. Note that in our work we assume the existence of temporal information in the temporal records. Recent work [6] discussed how to derive temporal ordering of records when time stamps are absent; combining our techniques with techniques in [6] would be interesting future work.

2 Overview

This section formally defines the temporal record linkage

problem (Section 2.1) and provides an overview of our approach (Section 2.2).

2.1 Problem definition

Consider a domain $\mathcal{D}$ of object entities (not known a-priori) where each entity is described by a set of attributes $\mathbf{A} = \{A_1, \ldots, A_n\}$ and values of an attribute can change over time (e.g., affiliation, business addresses). We distinguish *single-valued* and *multi-valued* attributes, where the difference is whether an attribute of an entity can have single or multiple values at any time. Consider a set $\mathbf{R}$ of records, each is associated with a time stamp and describing an entity in $\mathcal{D}$ at that particular time. Given a record $r \in \mathbf{R}$, we denote by $r.t$ the time stamp of r and by $r.A$ the value of attribute $A \in \mathbf{A}$ from r (we allow `null` as a value). Our goal is to decide which records in $\mathbf{R}$ refer to the same entity in $\mathcal{D}$.

Definition 1 (Temporal record linkage) Let $\mathbf{R}$ be a set of records, each in the form of $(x_1, \ldots, x_n, t)$, where t is the time stamp associated with the record, and $x_i, i \in [1, n]$, is the value of attribute A_i at time t for the referred entity.

The *temporal record linkage* problem clusters the records in $\mathbf{R}$ such that records in the same cluster refer to the same entity over time and records in different clusters refer to different entities.

Example 2 Consider the records in Table 1, where each record describes an author by name, a liation, and co-authors (co-authors is a multi-valued attribute) and is associated with a time stamp (year). The ideal linkage solution contains 3 clusters: $\{r_1\}, \{r_2, \ldots, r_6\}, \{r_7, \ldots, r_{12}\}$.

2.2 Overview of our solution

Our record-linkage techniques leverage the temporal information in two ways.

First, when computing record similarity, traditional linkage techniques reward high value similarity and penalize low value similarity. However, as time elapses, values of a particular entity may evolve; for example, a researcher may change affiliation, email, and even name over time (see entities E_2 and E_3 in Example 1). Meanwhile, different entities are more likely to share the same value(s) with a long time gap; for example, it is more likely that we observe two persons with the same name within 30 years than at the same time. We thus define *decay* (Section 3), with which we can reduce the penalty for value disagreement and reduce the reward for value agreement over a long period. Our experimental results show that applying decay in similarity computation can improve over traditional linkage techniques.

Second, when clustering records according to record similarity, traditional techniques do not consider the time order of the records. However, time order can often provide important clues. In Example 1, records $r_2 - r_4$ and $r_5 - r_6$ may refer to the same person even though the decayed similarity between r_4 and r_6 is low, because the time period of $r_2 - r_4$ (year 2004-2007) and that of $r_5 - r_6$ (year 2009-2010) do not overlap; on the other hand, records $r_2 - r_4$ and r_7, r_8, r_{10} are very likely to refer to different persons even though the decayed similarity between r_2 and r_{10} is high, because the records interleave and their occurrence periods highly overlap. We propose temporal clustering algorithms (Section 4) that consider time order of records and can further improve linkage results.

3 Time decay

This section introduces *time decay*, an important concept that aims at capturing the effect of time elapsing on value evolution. Section 3.1 defines decay, Section 3.2 describes how we learn decay, and Section 3.3 describes how we apply decay in similarity computation. Experimental results show that by applying decay in traditional linkage techniques, we can already improve the results.

3.1 Definition

As time goes by, the value of an entity may evolve; for example, entity E_2 in Example 1 was at *UW* from 2004 to 2007, and moved to *AT&T Labs* afterwards. Thus, different values for a single-valued attribute over a long period of time should not be considered as a strong indicator of referring to different entities. We define *disagreement decay* to capture this intuition.

Definition 2 (Disagreement decay) Let Δt be a time distance and $A \in \mathbf{A}$ be a single-valued attribute. *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.

On the other hand, as time goes by, we are more likely to observe two entities with the same attribute value; for example, in Example 1 entity E_1 occurred in 1991 and E_2 occurred in 2004-2010, and they share the same name. Thus, the same value over a long period of time should not be considered as strong indicator of referring to the same entity. We define *agreement decay* accordingly.

Definition 3 (Agreement decay) Let Δt be a time distance

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.

According to the definitions, decay satisfies two properties. First, decay is in the range of [0,1]; however, $d^{\neq}(A, 0)$ and $d^{=}(A, 0)$ are not necessarily 0, since even at the same time their value-match does not necessarily correspond to record-match and vice versa. Second, decay observes *monotonicity*; that is, for any $\Delta t < \Delta t'$ and any attribute A, $d^{\neq}(A, \Delta t) \leqslant d^{\neq}(A, \Delta t')$ and $d^{=}(A, \Delta t) \leqslant d^{=}(A, \Delta t')$. Whereas our definition of decay applies to all attributes, for attributes whose values always remain stable (e.g., birth-date), the disagreement decay is always 0, and for those whose values change rapidly (e.g., bank-account-balance), the disagreement decay is always 1.

Example 3 Figure 1 shows the curves of disagreement decay and agreement decay on attribute address learned from a European patent data set (described in detail in Section 5).

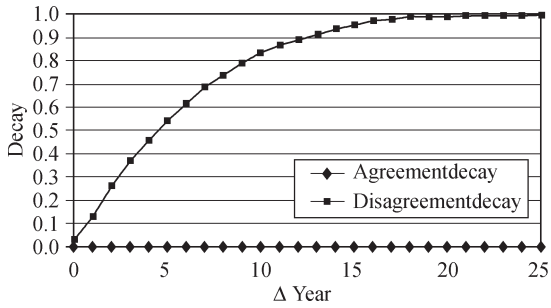

Fig. 1 Decay curves for Address

We observe that (1) the disagreement decay increases from 0 to 1 as time elapses, showing that two records differing in affiliation over a long time is not a strong indicator of referring to different entities; (2) the agreement decay is close to 0 everywhere, showing that in this data set, sharing the same address is a strong indicator of referring to the same entity even over a long time; (3) even when $\Delta t = 0$, neither the disagreement nor the agreement decay is exactly 0, meaning that even at the same time an address match does not definitely correspond to record match or mismatch.

3.2 Learning decay

Decay can be specified by domain experts or learned from a labeled data set, for which we know if two records refer to the same entity and if two strings represent the same value.[2)] For simplification of computation, we make three assumptions. 1. *Value uniqueness:* at each time point an entity has a single value for a single-valued attribute. 2. *Closed-world:* for each entity described in the labeled data set, during the time period when records that describe this entity are present, each of its ever-existing values is reflected by some record and the change is reflected at the transition point. 3. *Correctness:* values in each record reflect the truth in real world. The data sets in practice often violate the assumptions. In our learning we can resolve value-uniqueness conflicts with domain experts. Our experimental results show that the learned decay does lead to good linkage results even when the latter two assumptions are violated, as various kinds of violations in the real data often cancel out each other in affecting the learned curves. In addition, we relax the closed-world assumption and propose probabilistic decay, but our experiments show that it obtains very similar results to deterministic decay.

Consider attribute A and time period Δt. We next describe three ways to calculate decay for A and Δt according to the labels, namely, *deterministic decay*, *single-count decay* and *probabilistic decay*.

3.2.1 Disagreement decay

By definition, disagreement decay for Δt is the probability that an entity changes its A-value within time Δt. So we need to find the valid period of each A-value of an entity.

Consider an entity E and its records in increasing time order, denoted by $r_1, \ldots, r_n, n \geqslant 1$. We call a time point t a *change point* if at time t there exists a record $r_i, i \in [2, n]$, whose A-value is different from r_{i-1}. Additionally r_1 is always a change point. For each change point t (associated with a new value), we can compute a *life span*: if t is not the final change point of E, we call the life span of the current A-value a *full* time span and denote it by $[t, t_{next})$, where t_{next} is the next change point; otherwise, we call the life span a *partial* time span and denote it by $[t, t_{end} + \delta)$, where t_{end} is the final time stamp for this value and δ denotes one time unit (in Example 1, a unit of time is 1 year). A life span $[t, t')$ has length $t' - t$, indicating that the corresponding value lasts for time $t' - t$ before any change in case of a full life span, and that the value lasts *at least* for time $t' - t$ in case of a partial life span. $\bar{L}_f$ denotes the bag of lengths of full life spans, and $\bar{L}_p$ the bag of partial life spans.

Deterministic decay To learn $d^{\neq}(A, \Delta t)$, we consider all full life spans and the partial life spans with length of at least Δt (we cannot know for others if the value will change within

[2)] In case there is no label for whether two strings represent the same value, we can easily extend our techniques by considering value similarity.

Δt). We compute the decay as

$$d^{\neq}(A, \Delta t) = \frac{|\{l \in \bar{L}_f | l \leqslant \Delta t\}|}{|L_f| + |\{l \in L_p | l \geqslant \Delta t\}|}. \quad (1)$$

We give details of the algorithm in Algorithm LearnDisagreeDecay (Algorithm 1). We can prove that the decay it learns satisfies the monotonicity property.

Proposition 1 Let A be an attribute. For any $\Delta t < \Delta t'$, the decay learned by Algorithm LearnDisagreeDecay satisfies $d^{\neq}(A, \Delta t) \leqslant d^{\neq}(A, \Delta t')$.

Algorithm 1 LearnDisagreeDecay($\bar{C}, A$)

Input: $\bar{C}$ Clusters of records in the sample data set, where records in the same cluster refer to the same entity and records in different clusters refer to different entities.
A Attribute for learning decay.
Output: Disagreement decay $d^{\neq}(\Delta t, A)$.

```
1: L̄_f = φ; L̄_p = φ;
2: for each C ∈ C̄ do
3:    sort records in C in increasing time order to r_1, ..., r_|C|;
4:    // Find life spans
5:    start = 1;
6:    while start ⩽ |C| do
7:       end = start + 1;
8:       while r_start.A = r_end.A and end ⩽ |C| do
9:          end + +;
10:      end while
11:      if end > |C| then
12:         insert r_|C|.t − r_start.t + δ into L̄_p; // partial life span
13:      else
14:         insert r_end.t − r_start.t into L̄_f; // full life span
15:      end if
16:      start = end;
17:   end while
18: end for
19: // learn decay
20: for Δt = 1, ..., max_{l∈L̄_f ∪ L̄_p} {l} do
21:   d^≠(A, Δt) = |{l∈L̄_f | l⩽Δt}| / (|L_f| + |{l∈L_p | l⩾Δt}|)
22: end for
```

Proof In LearnDisagreeDecay, as Δt increases, $|\{l \in \bar{L}_f | l \leqslant \Delta t\}|$ is non-decreasing while $|\{l \in \bar{L}_p | l \geqslant \Delta t\}|$ is non-increasing; thus, $\frac{|\{l \in \bar{L}_f | l \leqslant \Delta t\}|}{|L_f| + |\{l \in L_p | l \geqslant \Delta t\}|}$ is non-decreasing. □

Example 4 Consider learning disagreement decay for affiliation from the data in Example 1. For illustrative purposes, we remove record r_{10} as its affiliation information is incorrect. Take E_2 as an example. As shown in Fig. 2, it has two change points: 2004 and 2009. So there are two life spans: $[2004, 2009)$ has length 5 and is full, and $[2009, 2011)$ has length 2 and is partial.

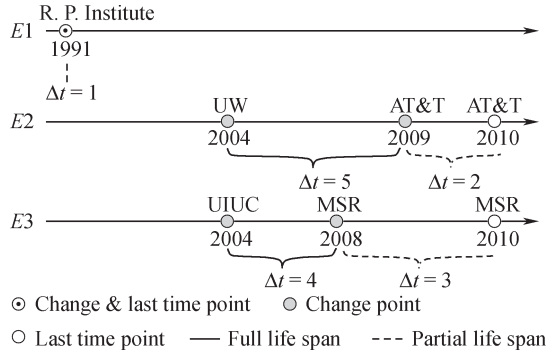

Fig. 2 Learning $d^{\neq}(\text{affiliation}, \Delta t)$

After considering other entities, we have $\bar{L}_f = \{4, 5\}$ and $\bar{L}_p = \{1, 2, 3\}$. Accordingly, $d^{\neq}(\text{affiliation}, \Delta t \in [0, 1]) = \frac{0}{2+3} = 0$, $d^{\neq}(\text{affiliation}, \Delta t = 2) = \frac{0}{2+2} = 0$, $d^{\neq}(\text{affiliation}, \Delta t = 3) = \frac{0}{2+1} = 0$, $d^{\neq}(\text{affiliation}, \Delta t = 4) = \frac{1}{2} = 0.5$, and $d^{\neq}(\text{affiliation}, \Delta t \geqslant 5) = \frac{2}{2} = 1$.

Single-count decay We consider an entity at most once and learn the disagreement decay $d^{\neq}(A, \Delta t)$ as the fraction of entities that have changed their A-value within time Δt. In particular, if an entity E has full life spans, we choose the shortest and insert its length l to $\bar{L}_f$, indicating that E has changed its A-value in time l; otherwise, we consider E's partial life span and insert its length l to $\bar{L}_p$, indicating that E has not changed its A-value in time l, but we do not know if it will change its A-value after any longer time. We learn disagreement decay using Eq. (1).

Proposition 2 Let A be an attribute. For any $\Delta t < \Delta t'$, the decay learned using *Single-count decay* satisfies $d^{\neq}(A, \Delta t) \leqslant d^{\neq}(A, \Delta t')$.

Proof In *single-count decay*, as Δt increases, $|\{l \in \bar{L}_f | l \leqslant \Delta t\}|$ is non-decreasing while $|\{l \in \bar{L}_p | l \geqslant \Delta t\}|$ is non-increasing; thus, $\frac{|\{l \in \bar{L}_f | l \leqslant \Delta t\}|}{|L_f| + |\{l \in L_p | l \geqslant \Delta t\}|}$ is non-decreasing. □

Example 5 Consider learning disagreement decay for affiliation from the following data: entity E_1 has two full life spans, $[2000, 2005)$, $[2005, 2009)$, and one partial life span $[2009, 2011)$; entity E_2 has one partial life span $[2003, 2010)$. For E_1, we consider its shortest full life span, which has length 4; for E_2, we consider its partial life span, which has length 7. Therefore, we have $\bar{L}_f = \{4\}$, and $\bar{L}_p = \{7\}$. Accordingly, $d^{\neq}(\text{affiliation}, \Delta t \in [0, 3]) = \frac{0}{1+1} = 0$, $d^{\neq}(\text{affiliation}, \Delta t \in [4, 7]) = \frac{1}{1+1} = 0.5$, $d^{\neq}(\text{affiliation}, \Delta t \geqslant 8) = \frac{1}{1+0} = 1$.

For comparison, on the same data set LearnDisagreeDecay learns the following disagreement decay: $d^{\neq}(\text{affiliation}, \Delta t \in [0, 3]) = 0$, $d^{\neq}(\text{affiliation}, \Delta t = 4) = \frac{1}{2+1} = 0.33$, $d^{\neq}(\text{affiliation}, \Delta t \in [5, 7]) = \frac{2}{2+1} = 0.67$, $d^{\neq}(\text{affiliation}, \Delta t \geqslant 8) = \frac{2}{2+0} = 1$. So the decay learned by LearnDisagreeDecay is smoother.

Probabilistic decay We remove the *closed-world* assumption; that is, each value change is reflected by a record at the change point. In particular, considering a full life span $[t, t_{next})$ we assume the last time we see the same value is $t', t \leqslant t' \leqslant t_{next}$. We assume the value can change at any time from t' to t_{next} with equal probability $\frac{1}{t_{next}-t'+1}$. Thus, for each $t_0 \in [t', t_{next}]$, we insert length $t_0 - t$ into $\bar{L}_f$ and annotate it with probability $\frac{1}{t_{next}-t'+1}$. If we denote by $p(l)$ the probability for a particular length l in $\bar{L}_f$, we compute the disagreement decay as

$$d^{\neq}(A, \Delta t) = \frac{\sum_{l \in \bar{L}_f, l \leqslant \Delta t} p(l)}{\sum_{l \in \bar{L}_f} p(l) + |\{l \in \bar{L}_p | l \geqslant \Delta t\}|}. \quad (2)$$

Proposition 3 Let A be an attribute. For any $\Delta t < \Delta t'$, the decay learned using *Probabilistic decay* satisfies $d^{\neq}(A, \Delta t) \leqslant d^{\neq}(A, \Delta t')$.

Proof In *probabilistic decay*, as Δt increases, $\sum_{l \in \bar{L}_f, l \leqslant \Delta t} p(l)$ is non-decreasing while $|\{l \in \bar{L}_p | l \geqslant \Delta t\}|$ is non-increasing; thus, $\frac{\sum_{l \in \bar{L}_f, l \leqslant \Delta t} p(l)}{\sum_{l \in \bar{L}_f} p(l) + |\{l \in \bar{L}_p | l \geqslant \Delta t\}|}$ is non-decreasing. □

Example 6 Consider learning disagreement decay for affiliation from the data set in Example 1. Again, we remove record r_{10} for illustrative purpose. Take E_2 as an example. Its first affiliation has full life span [2004, 2009), and its last time stamp is 2007. We consider the change can occur in any year from 2007 to 2009, each with probability $\frac{1}{2009-2007+1} = \frac{1}{3}$. So we insert length 3, 4, 5 into $\bar{L}_f$, each with probability $\frac{1}{3}$. Similarly, we insert length 3, 4 for entity E_3, each with probability 0.5.

Eventually, we have $\bar{L}_f = \{3(\frac{1}{3}), 4(\frac{1}{3}), 5(0.33), 3(0.5), 4(0.5)\}$, and $\bar{L}_p = \{1, 2, 3\}$. Accordingly, $d^{\neq}(\mathsf{a}, \Delta t \in [0, 1]) = \frac{0}{2+3} = 0$, $d^{\neq}(\mathsf{a}, \Delta t = 2) = \frac{0}{2+2} = 0$, $d^{\neq}(\mathsf{a}, \Delta t = 3) = \frac{0.5+0.33}{2+1} = 0.28$, $d^{\neq}(\mathsf{a}, \Delta t = 4) = \frac{0.33+0.33+0.5+0.5}{2} = 0.84$, and $d^{\neq}(\mathsf{a}, \Delta t \geqslant 5) = \frac{0.33+0.33+0.33+0.5+0.5}{2} = 1$.

Recall that for the same data set LearnDisagreeDecay learns the following decay: $d^{\neq}(\mathsf{a}, \Delta t \in [0, 3]) = 0$, $d^{\neq}(\mathsf{a}, \Delta t = 4) = 0.5$, and $d^{\neq}(\mathsf{a}, \Delta t \geqslant 5) = 1$. Thus, the curve learned by LearnDisagreeDecay is less smooth.

Experimental results (Section 5) show that these three methods learn similar (but more or less smooth) decay curves, and their results lead to similar linkage results.

3.2.2 Agreement decay

By definition, agreement decay for Δt is the probability that two different entities share the same value within time period Δt. Consider a value v of attribute A. Assume entity E_1 has value v with life span $[t_1, t_2)$ and E_2 has value v with life span $[t_3, t_4)$. Without losing generality, we assume $t_1 \leqslant t_3$. Then, for any $\Delta t \geqslant \max\{0, t_3 - t_2 + \delta\}$, E_1 and E_2 share the same value v within a period of Δt. We call $\max\{0, t_3 - t_2 + \delta\}$ the *span distance* for v between E_1 and E_2.[3)]

Algorithm 2 LearnAgreeDecay($\bar{C}, A$)

Input: $\bar{C}$ Clusters of records in the sample data set, where records in the same cluster refer to the same entity and records in different clusters refer to different entities.
A An attribute for decay learning.
Eneure: Agreement decay $d^{=}(\Delta t, A)$.

```
1: //Find life spans
2: for each C ∈ C̄ do
3:    sort records in C in increasing time order to r_1, ..., r_|C|;
4:    start = 1;
5:    while start ⩽ |C| do
6:       end = start + 1;
7:       while r_start.A = r_end.A and end ⩽ |C| do
8:          end + +;
9:       end while
10:      if end > |C| then
11:         r_start.t_next = r_|C|−1.t + δ; // partial life span
12:      else
13:         r_start.t_next = r_end.t; // full life span
14:      end if
15:      start = end;
16:   end while
17: end for
18: //learn agreement decay
19: L̄ = φ
20: for each C, C′ ∈ C̄ do
21:    same = false;
22:    for each r ∈ C s.t. r.t_next ≠ null do
23:       for each r′ ∈ C′ s.t. r.t_next ≠ null do
24:          if r.A = r′.A then
25:             same = true;
26:             if r.t ⩽ r′.t then
27:                insert max{0, r′.t − r.t_next + 1} into L̄;
28:             else
29:                insert max{0, r.t − r′.t_next + 1} into L̄;
30:             end if
31:          end if
32:       end for
33:    end for
34:    if !same then
35:       insert ∞ into L̄;
36:    end if
37: end for
38: for Δt = 1, ..., max_{l∈L̄}{l} do
39:    d^=(A, Δt) = |{l∈L̄ | l⩽Δt}| / |L̄|
40: end for
```

[3)] We can easily extend to the case where v has multiple life spans for the same entity.

For any pair of entities, we find the shared values and compute the corresponding span distance for each of them. If two entities never share any value, we use ∞ as the span distance between them. We denote by $\bar{L}$ the bag of span distances and compute the agreement decay as

$$d^{=}(A, \Delta t) = \frac{|\{l \in \bar{L} | l \leqslant \Delta t\}|}{|\bar{L}|}. \quad (3)$$

Algorithm LearnAgreeDecay (Algorithm 2) describes the details and we next show monotonicity of its results.

Proposition 4 Let A be an attribute. For any $\Delta t < \Delta t'$, the decay learned by Algorithm LearnAgreeDecay satisfies $d^{=}(A, \Delta t) \leqslant d^{=}(A, \Delta t')$.

Proof In LearnAgreeDecay, as Δt increases, $|\{l \in \bar{L} | l \leqslant \Delta t\}|$ is non-decreasing; thus, $\frac{|\{l \in \bar{L} | l \leqslant \Delta t\}|}{|\bar{L}|}$ is non-decreasing. □

Example 7 Consider learning agreement decay for name from data in Example 1. As shown in Fig. 3, entities E_1 and E_2 share value *Xin Dong*, for which the life span for E_1 is $[1991, 1992)$ and that for E_2 is $[2004, 2009)$. Thus, the span distance between E_1 and E_2 is $2004 - 1992 + 1 = 13$. No other pair of entities shares the same value; thus, $\bar{L} = \{13, \infty, \infty\}$. Accordingly, $d^{=}(\text{name}, \Delta t \in [0, 12]) = \frac{0}{3} = 0$, and $d^{=}(\text{name}, \Delta t \geqslant 13) = \frac{1}{3} = 0.33$.

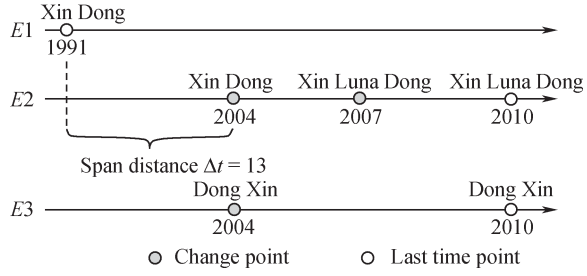

Fig. 3 Learning $d^{=}(\text{name}, \Delta t)$

Similarly, we can apply *single-count decay* or *probabilistic decay* to learn agreement decay. We omit the details here for brevity.

3.3 Applying decay

Here we describe how we apply decay in record-similarity computation. We first focus on single-valued attributes and then extend our method for multi-valued attributes.

3.3.1 Single-valued attributes

When computing similarity between two records with a big time gap, we often wish to reduce the penalty if they have different values and reduce the reward if they share the same value. Thus, we assign weights to the attributes according to the decay; the lower the weight, the less important an attribute is in the record-similarity computation, so there is a lower penalty for value disagreement or lower reward for value agreement. This weight is decided both by the time gap and by the similarity between the values (to decide whether to apply agreement or disagreement decay). We denote by $w_A(s, \Delta t)$ the weight of attribute A with value similarity s and time difference Δt. Given records r and r', we compute their similarity as

$$sim(r, r') = \frac{\sum_{A \in \mathbf{A}} w_A(s(r.A, r'.A), |r.t - r'.t|) \cdot s(r.A, r'.A)}{\sum_{A \in \mathbf{A}} w_A(s(r.A, r'.A), |r.t - r'.t|)}. \quad (4)$$

Next we describe how we set $w_A(s, \Delta t)$. With probability s, the two values are the same and we shall use the complement of the agreement decay; with probability $1 - s$, they are different and we shall use the complement of the disagreement decay. Thus, we set $w_A(s, \Delta t) = 1 - s \cdot d^{=}(A, \Delta t) - (1 - s) \cdot d^{\neq}(A, \Delta t)$. In practice, we use thresholds θ_h and θ_l to indicate high similarity and low similarity respectively, and set $w_A(s, \Delta t) = 1 - d^{=}(A, \Delta t)$ if $s > \theta_h$ and $w_A(s, \Delta t) = 1 - d^{\neq}(A, \Delta t)$ if $s < \theta_l$. Our experiments show robustness of our techniques with respect to different settings of the thresholds.

Example 8 Consider records r_2 and r_5 in Example 1 and we focus on single-valued attributes name and a liation. Assume the name similarity between r_2 and r_5 is 0.9 and the affiliation similarity is 0. Suppose $d^{=}(\text{name}, \Delta t = 5) = 0.05$, $d^{\neq}(\text{a}\ \ , \Delta t = 5) = 0.9$, and $\theta_l = 0.8$. Then, the weight for name is $1 - 0.05 = 0.95$ and that for a liation is $1 - 0.9 = 0.1$. So the similarity is $sim(r_1, r_2) = \frac{0.95 \times 0.9 + 0.1 \times 0}{0.95 + 0.1} = 0.81$. Note that if we do not apply decay and assign the same weight to each attribute, the similarity would become $\frac{0.5 \times 0.9 + 0.5 \times 0}{0.5 + 0.5} = 0.45$.

Thus, by applying decay, we are able to merge $r_2 - r_6$, despite the affiliation change of the entity. Note however that we will also incorrectly merge all records together because each record has a high decayed similarity to r_1.

3.3.2 Multi-valued attributes

In this subsection we consider multi-valued attributes such as co-authors. We start by describing record-similarity computation with such attributes, and then describe how we learn and apply decay for such attributes.

Multi-valued attributes differ from single-valued attributes in that the same entity can have multiple values for such attributes, even at the same time; therefore, (1) having different values for such attributes does not indicate record *mismatch*;

and (2) sharing the same value for such attributes is additional evidence for record *match*.

Consider a multi-valued attribute A. Consider records r and r'; $r.A$ and $r'.A$ each is a set of values. Then, the similarity between $r.A$ and $r'.A$, denoted by $s(r.A, r'.A)$, is computed by a variant of Jaccard distance between the two sets.

$$s(r.A, r'.A) = \frac{\sum_{v \in r.A, v' \in r'.A, s(v,v') > \theta_h} s(v, v')}{\min\{|r.A|, |r'.A|\}}. \quad (5)$$

If the relationship between the entities and the A-values is one-to-many, we add the attribute similarity (with a certain weight) to the record similarity between r and r'. In particular, let $sim'(r, r')$ be the similarity between r and r' when we consider *all* attributes and w_A be the weight for attribute A, then,

$$sim'(r, r') = min\{1, sim(r, r') + \sum_{multi-valued\ A} w_A \cdot s(r.A, r'.A)\}. \quad (6)$$

On the other hand, if the relationship between the entities and the A-values are many-to-many, we apply Eq. (6) only when $sim(r, r') > \theta_s$, where θ_s is a threshold for high similarity on values of single-valued attributes.

Now consider decay on such multi-valued attributes. First, we do not learn disagreement decay on multi-valued attributes but we learn agreement decay in the same way as for single-valued attributes. Second, we apply agreement decay when we compute the similarity between values of a multi-valued attribute, so if the time gap between two similar values is large, we discount the similarity. In particular, we revise Eq. (5) as follows:

$$s(r.A, r'.A) = \frac{\sum_{v \in r.A, v' \in r'.A, s(v,v') > \theta_h} (1 - d^{=}(A, |r.t - r'.t|)) s(v, v')}{\min\{|r.A|, |r'.A|\}}. \quad (7)$$

Example 9 Consider records r_2 and r_5 and multi-valued attribute co-authors (many-to-many relationship) in Example 1. Let $\theta_h = 0.8$ and $w_{\mathsf{co}} = 0.3$. Record r_2 and r_5 share one co-author with string similarity 1. Suppose $d^{=}(\mathsf{co}, \Delta t = 5) = 0.05$. Then, $s(r_2.\mathsf{co}, r_5.\mathsf{co}) = \frac{(1-0.05)*1}{\min\{2,2\}} = 0.475$. Recall from Example 8 that $sim(r_2, r_5) = 0.81 > \theta_h$; therefore, the overall similarity is $sim'(r_2, r_5) = \min\{1, 0.81 + 0.475 * 0.3\} = 0.95$.

4 Temporal clustering

As shown in Example 8, even when we apply decay in similarity computation, traditional clustering methods do not necessarily lead to good results as they ignore the time order of the records. This section proposes three clustering methods, all processing the records in increasing time order. *Early binding* (Section 4.1) makes eager decisions and merges a record with an already created cluster once it computes a high similarity between them. *Late binding* (Section 4.2) compares a record with each already created cluster and keeps the probability, and makes clustering decision at the end. *Adjusted binding* (Section 4.3) is applied after early binding or late binding, and improves over them by comparing a record also with clusters created later and adjusting the clustering results. Our experimental results show that adjusted binding significantly outperforms traditional clustering methods on temporal data.

4.1 Early binding

Algorithm: Early binding considers the records in time order; for each record it eagerly creates its own cluster or merges it with an already created cluster. In particular, consider record r and already created clusters $C_1, \ldots, C_n$. Early proceeds in three steps.

1. Compute the similarity between r and each $C_i, i \in [1, n]$.
2. Choose the cluster C with the highest similarity. Merge r with C if $sim(r, C) > \theta$, where θ is a threshold indicating high similarity; create a new cluster C_{n+1} for r otherwise.
3. Update signature for the cluster with r accordingly.

Cluster signature When we merge record r with cluster C, we need to update the signature of C accordingly (step 3). As we consider r as the latest record of C, we take r's values as the latest values of C. For the purpose of similarity computation, which we describe shortly, for each latest value v we wish to keep 1) its various representations, denoted by $\bar{R}(v)$, and 2) its earliest and latest time stamps in the current period of occurrence, denoted by $t_e(v)$ and $t_l(v)$ respectively. The latest occurrence of v is clearly $r.t$. We maintain the earliest time stamp and various representations recursively as follows. Let v' be the previous value of C, and let s_{max} be the highest similarity between v and the values in $\bar{R}(v')$. (1) If $s_{max} > \theta_h$, we consider the two values as the same and set $t_e(v) = t_e(v')$ and $\bar{R}(v) = \bar{R}(v') \cup \{v\}$. (2) If $s_{max} < \theta_l$, we consider the two values as different and set $t_e(v) = r.t$ and $\bar{R}(v) = \{v\}$. (3) Otherwise, we consider that with probability s_{max} the two values are the same, so we set $t_e(v) = sim(v, v')t_e(v') + (1 - sim(v, v'))r.t$ and $\bar{R}(v) = \bar{R}(v') \cup \{v\}$.

Similarity computation When we compare r with a clus-

ter C (step 1), for each attribute A, we compare r's A-value $r.A$ with the A-value in C's signature, denoted by $C.A$. We make two changes in this process: first, we compare $r.A$ with each value in $\bar{R}(C.A)$ and take the maximum similarity; second, when we compute the weight for A, we use $t_e(C.A)$ for disagreement decay as $C.A$ starts from time $t_e(C.A)$, and use $t_l(C.A)$ for agreement decay as $t_l(C.A)$ is the last time we see $C.A$.

We describe Algorithm EARLY in Algorithm 3. EARLY runs in time $O(|\mathbf{R}|^2)$; the quadratic time is in the number of records in each block after preprocessing.

Algorithm 3 EARLY($\mathbf{R}$)

Input: $\mathbf{R}$ records in increasing time order
Output: $\bar{C}$ clustering of records in $\mathbf{R}$

```
1: for each record r ∈ R do
2:    for each C ∈ C̄ do
3:       compute record-cluster similarity sim(r, C);
4:    end for
5:    if max_{C∈C̄} sim(r, C) ⩾ θ then
6:       C = Argmax_{C∈C̄} sim(r, C);
7:       insert r into C;
8:       update signature of C;
9:    else
10:      insert cluster {r} into C̄;
11:   end if
12: end for
13: return C̄;
```

Example 10 Consider applying early binding to records in Table 1. Table 2 shows the signature of a liation for each cluster after we process each record. The change in each step is in bold.

Table 2 Example 10: cluster signature in early binding

	C_1	C_2	C_3
r_1	**R.Poly, 1991-1991**	-	-
r_2	**UW, 2004-2004**	-	-
r_7	UW, 2004-2004	**UI, 2004-2004**	-
r_3	**UW, 2004-2005**	UI, 2004-2004	-
r_8	UW, 2004-2005	**UI, 2004-2007**	-
r_4	**UW, 2004-2007**	UI, 2004-2007	-
r_9	UW, 2004-2007	**MSR, 2008-2008**	-
r_{10}	**UI, 2009-2009**	MSR, 2008-2008	-
r_{11}	UI, 2009-2009	**MSR, 2008-2009**	-
r_5	UI, 2009-2009	MSR, 2008-2009	**AT&T, 2009-2009**
r_{12}	UI, 2009-2009	**MSR, 2008-2010**	AT&T, 2009-2009
r_6	UI, 2009-2009	MSR, 2008-2010	**AT&T, 2009-2010**

We start with creating C_1 for r_1. Then we merge r_2 with C_1 because of the high record similarity (same name and high disagreement decay on a liation with time difference 2004-1991=13). The new signature of C_1 contains address UW from 2004 to 2004. We then create a new cluster C_2 for r_7, as r_7 differs significantly from C_1. Next, we merge r_3 and r_4 with C_1 and merge r_8 and r_9 with C_2. The signature of C_1 then contains address UW from 2004 to 2007, and the signature of C_2 contains address MSR from 2008 to 2008.

Now consider r_{10}. It has a low similarity to C_2 (r_{10} and r_9 has a short time distance but different affiliations), but a high similarity to C_1 (fairly similar name and high disagreement decay on a liation with time difference $2009 - 2004 = 5$). We thus wrongly merge r_{10} with C_1. This eager decision further prevents merging r_5 and r_6 with C_1 and we create C_3 for them separately.

4.2 Late binding

Instead of making eager decisions and comparing a record with a cluster based on such eager decisions, late binding keeps all evidence, considers them in record-cluster comparison, and makes a global decision at the end.

Late binding is facilitated by a bipartite graph (N_R, N_C, E), where each node in N_R represents a record, each node in N_C represents a cluster, and each edge $(n_r, n_C) \in E$ is marked with the probability that record r belongs to cluster C (see Fig. 4 for an example). Late binding clusters the records in two stages: first, *evidence collection* creates the bi-partite graph and computes the weight for each edge; then, *decision making* removes edges such that each record belongs to a single cluster.

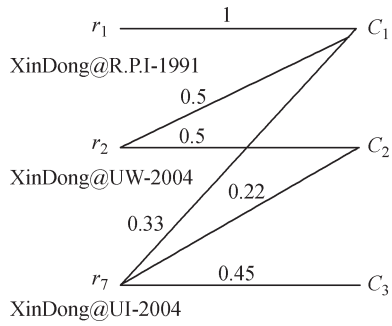

Fig. 4 Example 11: A part of the bi-partite graph

4.2.1 Evidence collection

Late binding behaves similarly to early binding at the evidence collection stage, except that it keeps all possibilities rather than making eager decisions. For each record r and already created clusters $C_1, \ldots, C_n$, it proceeds in three steps.

1. Compute the similarity between r and each $C_i, i \in [1, n]$.
2. Create a new cluster C_{n+1} and assign similarity as fol-

lows. (1) If for each $i \in [1,n]$, $sim(r,C_i) \leqslant \theta$, we consider that r is unlikely to belong to any C_i and set $sim(r,C_{n+1}) = \theta$. (2) If there exists $i \in [1,n]$, such that not only $sim(r,C_i) > \theta$, but also $sim'(r,C_i) > \theta$, where $sim'(r,C_i)$ is computed by ignoring decay, we consider that r is very likely to belong to C_i and set $sim(r,C_{n+1}) = 0$. (3) Otherwise, we set $sim(r,C_{n+1}) = \max_{i\in[1,n]} sim(r,C_i)$.

3. Normalize the similarities such that they sum up to 1 and use the results as probabilities of r belonging to each cluster.
   Update the signature of each cluster accordingly.

In the final step, we normalize the similarities such that the higher the similarity, the higher the result probability. Note that in contrast to early binding, late binding is conservative when the record similarity without decay is low (Step 2(3)); this may lead to splitting records that have different values but refer to the same entity, and we show later how adjusted binding can benefit from the conservativeness.

Edge deletion In practice, we may set low similarities to 0 to improve performance; we next describe several edge-deletion strategies. Our experimental results (Section 5) show that they obtain similar results, while they all improve over not deleting edges in both efficiency and accuracy of the results.

- *Thresholding* removes all edges whose associated similarity scores are less or equal to a threshold θ.
- *Top-K* keeps the top-k edges whose associated similarity scores are above threshold θ.
- *Gap* orders the edges in descending order of the associated similarity scores to $e_1, e_2, \ldots, e_n$, and selects the edges in decreasing order until reaching an edge e_i where (1) the scores for e_i and e_{i+1} have a gap larger than a given threshold θ_{gap}, or (2) the score for e_{i+1} is less than threshold θ.

Cluster signature For each cluster, the signature consists of all records that may belong to the cluster along with the probabilities. For each value of every record, we maintain the earliest time stamp, the latest time stamp, and similar values, as we do in early binding.

Similarity computation When we compare r with a cluster C, we need to consider the probability that a record in C's signature belongs to C. Let $r_1, \ldots, r_m$ be the records of C in increasing time order, and let $P(r_i), i \in [1,m]$, be the probability that r_i belongs to C. Then, with probability $P(r_m)$, record r_m is the latest record of C and we should compare r with it; with probability $(1-P(r_m))P(r_{m-1})$, record r_{m-1} is the latest record of C and we should compare r with it; and so on. Note that the cluster is valid only when r_1, for which we create the cluster, belongs to the cluster, so we use $P(r_1) = 1$ in the computation (the original $P(r_1)$ is used in the decision-making stage). Formally, the similarity is computed as

$$sim(r,C) = \sum_{i=1}^{m} sim(r,r_i)P(r_i)\Pi_{j=i+1}^{m}(1-P(r_i)). \quad (8)$$

Example 11 Consider applying late binding to the records in Table 1 and let $\theta = 0.8$. Figure 4 shows a part of the bipartite graph. At the beginning, we create an edge between r_1 and C_1 with weight 1. We then compare r_2 with C_1: the similarity with decay ($0.89 > \theta$) is high but that without decay ($0.5 < \theta$) is low. We thus create a new cluster C_2 and set $sim(r_2,C_2) = 0.89$. After normalization, each edge from r_2 has a weight of 0.5.

Now consider r_7. For C_1, with probability 0.5 we shall compare r_7 with r_2 (suppose $sim(r_7,r_2) = 0.4$) and with probability $1 - 0.5 = 0.5$ we shall compare r_7 with r_1 (suppose $sim(r_7,r_1) = 0.8$). Thus, $sim(r_7,C_1) = 0.8\times0.5 + 0.4\times0.5 = 0.6 < \theta$. For C_2, we shall compare r_7 only with r_2 and the similarity is $0.4 < \theta$. Because of the low similarities, we create a new cluster C_3 and set $sim(r_7,C_3) = 0.8$. After normalization, the probabilities from r_7 to C_1, C_2 and C_3 are 0.33, 0.22 and 0.45 respectively.

4.2.2 Decision making

The second stage makes clustering decisions according to the evidence we have collected. We consider only *valid* clusterings, where each non-empty cluster contains the record for which we create the cluster. Let $\bar{C}$ be a clustering and we denote by $\bar{C}(r)$ the cluster to which r belongs in $\bar{C}$. We can compute the probability of $\bar{C}$ as $\Pi_{r\in\mathbf{R}}P(r \in \bar{C}(r))$, where $P(r \in \bar{C}(r))$ denotes the probability that r belongs to $\bar{C}(r)$. We wish to choose the valid clustering with the highest probability. Enumerating all clusterings and computing the probability for each of them can take exponential time. We next propose an algorithm that takes only polynomial time and is guaranteed to find the optimal solution.

1. Select the edge (n_r, n_C) with the highest weight.
2. Remove other edges connected to n_r.
3. If n_r is the first selected edge to n_C but C is created for record $r' \neq r$, select the edge $(n_{r'}, n_C)$ and remove all other edges connected to $n_{r'}$ (so the result clustering is

valid).

4. Go to Step 1 until all edges are either selected or removed.

We describe algorithm LATE in Algorithm 4 and next state the optimality of the decision-making stage.

Algorithm 4 LATE($\mathbf{R}$)

Input: $\mathbf{R}$ records in increasing time order
Output: $\bar{C}$ clustering of records in $\mathbf{R}$

Initialize a bi-partite graph (N_R, N_C, E) where $N_R = N_C = E = \emptyset$;
//Evidence collection
for each record $r \in \mathbf{R}$ **do**
  insert node n_r into N_R;
  for each $n_C \in N_C$ **do**
    compute decayed record-cluster similarity $sim(r, C)$;
  end for
  if $\max_{n_C \in N_C} sim(r, C) \leqslant \theta$ **then**
    insert node n_{C_r} into N_C;
    insert edge (n_r, n_{C_r}) with weight θ into E;
  else
    $newCluster = true$;
    for each $n_C \in N_C$, where $sim(r, C) > \theta$ **do**
      compute no decayed similarity $sim'(r, C)$;
      if $sim'(r, C) > \theta$ **then**
        $newCluster = false$; **break**;
      end if
    end for
    if *newCluster* **then**
      insert node n_{C_r} into N_C;
      insert edge (n_r, n_{C_r}) with weight $\max_{n_C \in N_C, sim'(r,C) > \theta}(sim(r, C))$ into E;
    end if
  end if
  delete edges with low weights;
  normalize weights for all edges from n_r;
end for
// Decision making
while $|E| > |N_R|$ **do**
  select edge (n_r, n_C) with maximal edge weight;
  remove edges $(n_r, n_{C'})$ for all $C' \neq C$;
  if cluster C is created for record $r' \neq r$ **then**
    select edge $(n_{r'}, n_C)$;
    remove edges $(n_{r'}, n_{C'})$ for all $C' \neq C$;
  end if
end while
return $\bar{C}$;

Proposition 5 LATE algorithm runs in time $O(|\mathbf{R}|^2)$ and chooses the clustering with the highest probability from all possible valid clusterings. □

PROOF Our *evidence collection* step guarantees that if C_r is created for record r, then the edge (N_r, N_{C_r}) has the highest weight among edges from N_r. Thus, the *decision making* step chooses the edge with the highest weight for each record and obtains the optimal solution. □

Example 12 Continuing from Example 11. After evidence collection, we created 5 clusters and the weight of each record-cluster pair is shown in Table 3. Weights of selected edges are in bold.

Table 3 Example 12: weights on the bipartite graph

	r_1	r_2	r_7	r_3	r_8	r_4	r_9	r_{10}	r_{11}	r_5	r_{12}	r_6
C_1	**1**	0.5	0.33	0.37	0.27	0.38	0.16	0.13	0.18	0.24	0.12	0.22
C_2	0	**0.5**	0.22	**0.40**	0.25	**0.40**	0.16	0.12	0.17	**0.27**	0.10	**0.24**
C_3	0	0	**0.45**	0.23	**0.48**	0.22	0.24	0.26	0.20	0.17	0.23	0.18
C_4	0	0	0	0	0	0	**0.44**	0.19	**0.29**	0.16	**0.33**	0.18
C_5	0	0	0	0	0	0	0	**0.30**	0.16	0.16	0.22	0.18

We first select edge (n_{r_1}, n_{C_1}) with weight 1. We then choose (n_{r_2}, n_{C_2}) with weight 0.5 (there is a tie between C_1 and C_2; even if we choose C_1 at the beginning, we will change back to C_2 when we select edge (n_{r_3}, n_{C_2})), and (n_{r_8}, n_{C_3}) with weight 0.48. As C_3 is created for record r_7, we also select edge (n_{r_7}, n_{C_3}) and remove other edges from r_7. We choose edges for the rest of the records similarly and the final result contains five clusters: $\{r_1\}, \{r_2, \ldots, r_6\}$, $\{r_7, r_8\}, \{r_9, r_{11}, r_{12}\}, \{r_{10}\}$.

Note that despite the error made for r_{10}, we are still able to correctly merge r_5 and r_6 with C_2 because we make the decision at the end. Note however that we did not merge r_9, r_{11} and r_{12} with C_3, because of the conservativeness of late binding.

4.3 Adjusted binding

Neither early binding nor late binding compares a record with a cluster created later. However, evidence from later records may fix early errors; in Example 1, after observing r_{11} and r_{12}, we are more confident that $r_7 - r_{12}$ refer to the same entity but record r_{10} has out-of-date affiliation information. Adjusted binding allows comparison between a record and clusters that are created later.

Adjusted binding can start with the results from either early or late binding and iteratively adjust the clustering (*deterministic adjusting*), or start with the bi-partite graph created from evidence collection of late binding, and iteratively adjust the probabilities (*probabilistic adjusting*). We next describe the two algorithms.

4.3.1 Deterministic algorithm

Deterministic adjusting proceeds in EM-style.

1. *Initialization*: Set the initial assignment as the result of early or late binding.
2. *Estimation* (E-step): Compute the similarity of each record-cluster pair and normalize the similarities as in late binding.
3. *Maximization* (M-step): Choose the clustering with the maximum probability, as in late binding.
4. *Termination*: Repeat E-step and M-step until the results converge or oscillate.

Similarity computation The E-step compares a record r with a cluster C, whose signature may contain records that occur later than r. Our similarity computation takes advantage of this complete view of value evolution as follows.

First, we consider *consistency* of the records, including consistency in evolution of the values, in occurrence frequency, and so on. We next describe how we compute value consistency and occurrence frequency.

Consider the value consistency between r and $C = \{r_1, \ldots, r_m\}$ (if $r \in C$, we remove r from C), denoted by $cons(r, C) \in [0, 1]$. Assume the records of C are in time order and $r_k.t < r.t < r_{k+1}.t$.[4)] Inserting r into C can affect the consistency of C in two ways: 1) r may be inconsistent with r_k, so the similarity between r and the sub-cluster $C_1 = \{r_1, \ldots, r_k\}$ is low; 2) r_{k+1} may be inconsistent with r, so the similarity between r_{k+1} and the sub-cluster $C_2 = \{r_1, \ldots, r_k, r\}$ is low. We take the minimum as $cons(r, C)$,

$$cons(r, C) = \min(sim(r, C_1), sim(r_{k+1}, C_2)). \tag{9}$$

Occurrence consistency considers a cluster C. The *occurrence frequency* of C, denoted by $freq(C)$, is computed by

$$freq(C) = \frac{C_{late} - C_{early}}{|C|}. \tag{10}$$

Let C' be the cluster after inserting record r into C. The *occurrence consistency* between r and C, denoted by $cons_f(r, C)$ is computed by

$$cons_f(r, C) = 1 - \frac{|freq(C) - freq(C')|}{\max\{freq(C), freq(C')\}}. \tag{11}$$

Second, we consider *continuity* of r and C's other records in time, denoted by $cont(r, C) \in [0, 1]$. Consider the five cases in Fig. 5 and assume the same consistency between r and C. Record r is farther away in time from C's records in cases 1 and 5 than in cases 2—4, so it is less likely to belong to C in cases 1 and 5. Let $C.early$ denote the earliest time stamp of records in C and $C.late$ denote the latest one. We compute the continuity as follows.

$$cont(r, C) = e^{-\lambda y}; \tag{12}$$

$$y = \frac{|r.t - C.early| + \alpha}{C.late - C.early + \alpha}. \tag{13}$$

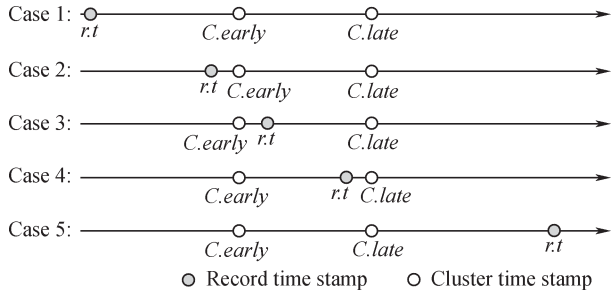

Fig. 5 Continuity between record r and cluster C

Here, $\lambda > 0$ is a parameter that controls the level of continuity we require; $\alpha > 0$ is a small number such that when the denominator (resp. numerator) is 0, the numerator (resp., denominator) can still affect the result[5)]. Under this definition, the higher the time difference between r and the earliest record in C compared with the length of C, the lower the continuity. In Fig. 5, $cont(r, C)$ is close to 0 in cases 1, 5, close to 1 in cases 2, 3, and close to $e^{-\lambda}$ in case 4. Note that we favor time points close to $C.early$ more than those close to $C.late$; thus, when we merge two clusters that are close in time, we will gradually move the latest record of the early cluster into the late cluster, as it has a higher continuity with the late cluster.

Finally, the similarity of r and C considers both consistency and continuity, and is computed by

$$sim(r, C) = cons(r, C) \cdot cont(r, C). \tag{14}$$

Recall that late binding is conservative for records whose similarity without decay is low and may split them. Adjusted binding re-examines them and merges them only when they have both high consistency and high continuity, and thus avoids aggressive merging of records with big time gap.

We describe the detailed algorithm, Adjust, in Algorithm 5. Our experiments show that Adjust does not necessarily converge, but the quality measures of the results at the oscillating rounds are very similar.

[4)] We can extend our techniques to the case when r has the same time stamp as some record in C.

[5)] In practice, we set α to one time unit, and set $\lambda = -\ln cons$ where $cons$ is the minimum consistency we require for merging a record with a cluster. When we merge two clusters C_1 and C_2 where $C_1.late = C_2.early$, with such λ the latest record r_1 of C_1 has continuity $cons$ with C_1 and continuity 1 with C_2, so can be merged with C_2 if $cons(r_1, C_2) > cons$.

Example 13 Consider r_{10} and $C_4 = \{r_9, r_{11}, r_{12}\}$ in the results of Example 12. For value consistency, inserting r_{10} into C_4 results in $\{r_9, \ldots, r_{12}\}$. Assume $sim(r_{10}, \{r_9\}) = sim(r_{11}, \{r_9,$
$r_{10}\}) = 0.6$. Then, $cons(r_{10}, C_4) = 0.6$. For continuity, if we set $\lambda = 2$ and $\alpha = 1$, we obtain $l(r_{10}, C_4) = e^{-2 \cdot \frac{1+1}{2+1}} = 0.26$. Thus, the similarity is $0.26 \times 0.6 = 0.16$. On the other hand, the similarity between r_{10} and C_5 is $1 \cdot e^{-2 \cdot \frac{1}{1}} = 0.14$. We thus merge r_{10} with C_4.

Similarly, we then merge r_8 with C_4 and in turn r_7 with C_4, leading to the correct result. Note that we do not merge r_1 with C_2, because of the long time gap and thus a low continuity.

4.3.2 Probabilistic adjusting

Probabilistic adjusted binding proceeds in three steps.

- The algorithm starts with the bi-partite graph created from evidence collection in late binding.
- It iteratively adjusts the weight of each edge and keeps all edges, until the weights converge or oscillate.
  In each iteration, it (1) re-computes the similarity between each record and each cluster, (2) normalizes the weights of edges from the same record node, and (3) re-computes the signature of each cluster.
- It selects the possible world with the highest probability as in late binding.

Algorithm 5 ADJUST($\mathbf{R}, C$)

Input: $\mathbf{R}$ records in increasing time order.
$\bar{C}$ pre-clustering of records in $\mathbf{R}$.
Output: $\bar{C}$ new clustering of records in $\mathbf{R}$.

```
1: repeat
2:   //E-step
3:   for each record r ∈ R do
4:     for each cluster C ∈ C̄ do
5:       compute sim(r, C) = cons(r, C) ∗ cont(r, C);
6:     end for
7:   end for
8:   //M-step
9:   Choose the possible world with the highest probability as in Ln.28–
     25 of LATE;
10: until C̄ is not changed
11:   return C̄;
```

In the second step, when we compute the similarity between a record and a cluster, we compute consistency $cons(r, C)$ and continuity $cont(r, C)$ similarly as in deterministic adjusted binding, except that we also need to consider the probability of a record belonging to a cluster. For value consistency, we consider probability in the same way as in late binding. For continuity, we compute the probabilistic earliest and latest time stamps of a cluster as follows. Suppose cluster C is connected to m records $r_1, \ldots, r_m$ where $r_1.t \leqslant r_2.t \leqslant \cdots \leqslant r_m.t$, each with probability $p_i, i \in [1, m]$. We compute C_{early} and C_{late} as follows:

$$C_{early} = \sum_{i=1}^{m} r_i.t \cdot (p_i \Pi_{k=1}^{i-1}(1 - p_k)). \quad (15)$$

$$C_{late} = \sum_{i=1}^{m} r_i.t \cdot (p_i \Pi_{k=i+1}^{m}(1 - p_k)). \quad (16)$$

Our experiments (Section 5) shows that probabilistic adjusting takes much longer than deterministic adjusting, but does not have an obvious performance gain.

5 Experimental evaluation

This section describes the results of experiments on two real data sets. We show that (1) our technique significantly improves on traditional methods on various data sets; (2) the two key components of our strategy, namely, decay and temporal clustering, are both important for obtaining good results; (3) our technique is robust with respect to various data sets and reasonable parameter settings; (4) our techniques are efficient and scalable.

5.1 Experiment settings

Data and golden standard We experimented on two real-world data sets: a benchmark of European patent data set[6] and the DBLP data set. From the patent data 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 from French patents, where different inventors rarely share similar names; we thus increased the hardness by deriving a data set with only first name and last name initial for each inventor. We call the original data set the *full* set and the derived one the *partial* set.

From the DBLP data we considered two subsets: the *XD* data set contains 72 records for authors named *Xin Dong*, *Luna Dong, Xin Luna Dong*, or *Dong Xin*, for which we manually identified 8 authors; the *WW* data set contains 738 records for authors named *Wei Wang*, for 302 of which DBLP has manually identified 18 authors (the rest is left in a pot-

[6] http://www.esf-ape-inv.eu/

pourri). For each subset we extracted Author records with attributes name, a liation, and co-author (we extracted affiliation information from the papers) on 2/1/2011; the time stamp of each record is the paper publication year. Table 4 shows statistics of the data sets.

Table 4 Statistics of the experimental data sets

	#Records	#Entities	Years
Patent (full or partial)	1871	359	1978—2003
DBLP-*XD*	72	8	1991—2010
DBLP-*WW*	738	18+potpourri	1992—-2011

Implementation We learned decay from both patent data sets. The decay we learned for the address attribute is shown in Fig. 1 (Section 3); for name, both agreement and disagreement decay are close to 0 on both data sets. We observed similar linkage results when we learned the decays from half of the data and applied them to the other half. We also applied the decay learned from the *partial* data set on linking DBLP records.

We pre-partitioned the records by the initial of the last name, and implemented the following methods on each partition.

- *Baseline methods* include PARTITION, CENTER, and MERGE [7]. They all compute pairwise record similarity but apply different clustering strategies. We give the details as follows.
  - PARTITION starts with single-record clusters and merges two clusters if they contain similar records (i.e., applying the transitive rule).
  - CENTER scans the records, merging a record r with a cluster if it is similar to the *center* of the cluster; otherwise, creates a new cluster with r as its center.
  - MERGE starts from the result of CENTER and merges two clusters if a record from one cluster is similar to the center of the other cluster.

  Since CENTER and MERGE are order sensitive, we run each of them 5 times and report the best results.
  Decayed baseline methods include DECAYEDPARTITION, DECAYEDCENTER, and DECAYEDMERGE, each modifying the corresponding baseline method by applying decays in record-similarity computation.
- *Temporal clustering methods* include NODECAYADJUST, applying ADJUST without using decay.
- *Full methods* include EARLY, LATE, and ADJUST, each applying both decay and the corresponding clustering algorithm.

Similarity computation We compute similarity between a pair of attribute values as follows.

- name: We used Levenshtein metric except that if the Levenshtein similarity is above 0.5 and the Soundex similarity is 1, we set similarity to 1.
- address: We used TF/IDF metric, where token similarity is measured by Jaro-Winker distance with threshold 0.9.
  If the TF/IDF similarity is above 0.5 and the Soundex similarity is 1, we set similarity to 1.
- co-author: We use a variant of Jaccard metric (see Eq. (7)), where name similarity is measured by Levenshtein distance and $\theta_h = 0.8$. We apply this similarity only when the record similarity w.r.t single-valued attributes is above $\theta_s = 0.5$.

By default, when we compute the record similarity without applying decay, we use weight 0.5 for both name and address (or a liation). Whether or not we apply decay, we use weight 0.3 for co-author. We apply threshold 0.8 for deciding if a similarity is high in various contexts. In addition, we set $\theta_h = 0.8, \theta_l = 0.6, \lambda = 0.5, \alpha = 1$ in our methods. We vary these parameters to demonstrate robustness.

We implemented the algorithms in Java, using a WindowsXP machine with 2.66 GHz Intel CPU and 1 GB of RAM.

Measure We compare pairwise linking decisions with the golden standard and measure the quality of the results by *precision* (P), *recall* (R), and *F-measure* (F). We denote the set of false positive pairs by FP, the set of false negative pairs by FN, and the set of true positive pairs by TP. Then, $P = \frac{|TP|}{|TP|+|FP|}$, $R = \frac{|TP|}{|TP|+|FN|}$, $F = \frac{2PR}{P+R}$.

5.2 Results on patent data

Figure 6 compares ADJUST with the baseline methods. ADJUST obtains slightly lower precision (but still above .9) but much higher recall (above .8) on both data sets; it improves the F-measure over baseline methods by 15%—27% on the full data set, and by 11%—22% on the partial data set. The *full* data set is simpler as very few inventors share similar full names; as a result, ADJUST obtains higher precision and recall on this data set. The slightly lower recall on the *partial* data set is because early false matching can prevent correct later matching. We next give a detailed comparison of the *partial*

data set, which is harder. Of the baseline methods, PARTITION obtains the best results on the patent data set and we next show results only on it. Results for the other two baseline methods follow the same pattern.

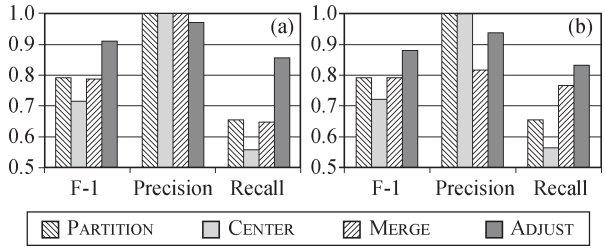

Fig. 6 Results on the patent data set

Figure 7 shows the contribution of applying decay and applying temporal clustering. We observe that DECAYEDPARTITION and NODECAYADJUST both improve over PARTITION, and ADJUST obtains the best result. Applying decay on baseline methods greatly increases the recall, but it is at the price of a big drop in precision. Temporal clustering, on the other hand, considers the time information in clustering and in continuity computation, so it significantly increases the recall without much reduction in precision.

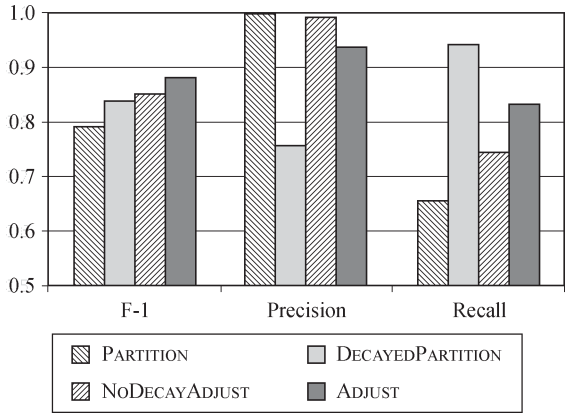

Fig. 7 Different components on patent *partial* data

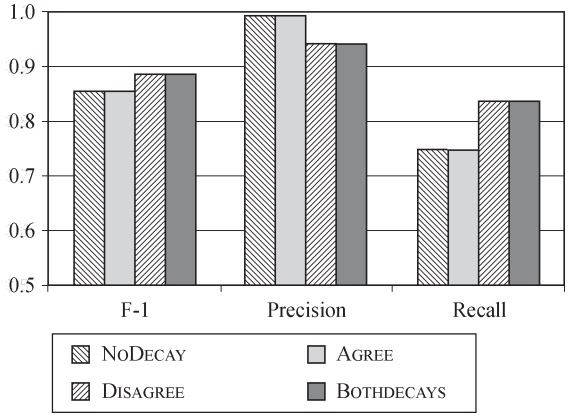

Fig. 8 Different decays on patent *partial* data

5.2.1 Applying decay

Disagreement vs agreement decay Figure 8 compares the results of applying no decay, applying only agreement decay, applying only disagreement decay, and applying both decays. We observe that while applying disagreement significantly improve the results, applying agreement decay does not change the results much, since the agreement decays of both attributes are close to 0.

Decay learning methods We learned the decay in three ways: DETERMINISTIC, SINGLECOUNT, and PROBABILISTIC, as described in Section 3. We observe that (1) these three methods learn similar curves, and (2) as shown in Fig. 9, applying the three different curves lead to very similar results for ADJUST, while DETERMINISTIC obtains slightly higher F-measure than the other two.

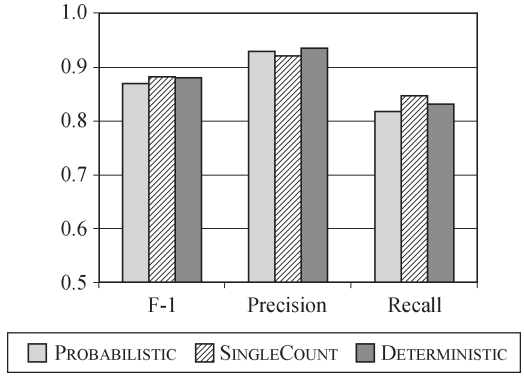

Fig. 9 Comparing different decay learning methods

5.3.2 Temporal clustering

Different clustering methods Figure 10 compares early, late, and adjusted binding. We observe that all bindings improve the recall over PARTITION, and reduce the precision only slightly. Between EARLY and LATE, EARLY has a lower precision as it makes local decisions, while LATE has a lower

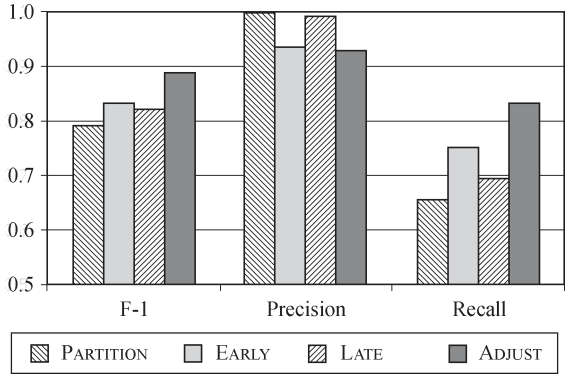

Fig. 10 Different clustering methods on patent *partial* data

recall as it is conservative in merging records with similar names but different addresses (high decayed similarity but low non-decayed similarity). ADJUST significantly improves the recall over both methods by comparing early records with clusters formed later, without sacrificing much precision.

Edge deletion strategies We tried various edge-deletion strategies for late binding: NODELETE keeps all edges; THRESHOLDING keeps edges with similarity over 0.8; TOPK keeps only the top-k edges with similarity over 0.8; GAP keeps the top edges with weights above 0.8 and gap within 0.1. Figure 11 shows that (1) NODELETE keeps all edges, which often have low weights after normalization, and can thus split many clusters and obtain a very low recall; and (2) different edge-deletion strategies lead to very similar F-measures and improve both efficiency and result quality over NODELETE.

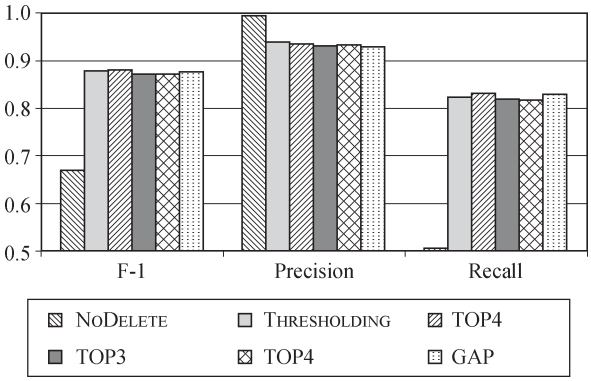

Fig. 11 Comparing different edge deletion strategies

Cluster adjusting strategies We implemented three versions of adjusted binding: LATEADJUST applies deterministic binding on the results of late binding; EARLYADJUST applies deterministic binding on the results of early binding; and PROBADJUST applies probabilistic binding on the bi-partite graph created in late binding. Figure 12 shows their results. First, we observe that PROBADJUST obtains similar results to LATEADJUST while the running time is 50% longer (not shown in the figure); showing that it does not have obvious advantage. Second, we observe that EARLYADJUST and LATEADJUST obtain similar results on the patent data set; however, as shown in Fig. 17(c), LATEADJUST improves over EARLYADJUST by 26% on another data set, the DBLP *WW* data set.

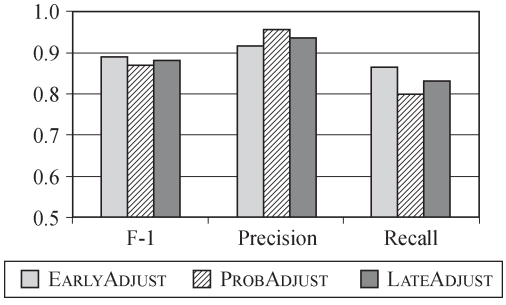

Fig. 12 Different adjusted binding methods on patent *partial* data

5.2.3 Robustness

We ran two experiments to test robustness against parameter settings. We first changed thresholds θ_h and θ_l for string similarity and observed very similar results (varying within 0.4%) when $\theta_h \in [0.7, 0.9]$ and $\theta_l \in [0.5, 0.7]$ (see Fig. 13).

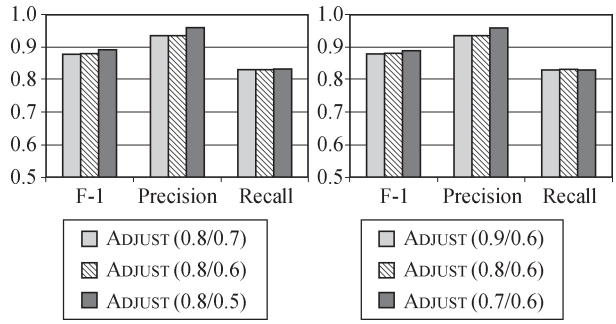

Fig. 13 Different thresholds on patent *partial* data

Second, we applied different attribute weights ($w_{\text{name}} \in [0.4, 1]$, $w_{\text{addr.}} = 1 - w_{\text{name}}$) to compute no-decayed similarity. Figure 14 shows that (1) ADJUST is robust against attribute weights; and (2) ADJUST always outperforms PARTITION in F-measure.

5.2.4 Scalability

To test scalability of our techniques, we randomly divided the partial patent data set into 10 subsets with similar sizes without splitting entities. We started with one subset and gradually added more, and reported the execution time in Fig. 15. We observe that (1) ADJUST terminated in 10.3 minutes on all 1871 records and is reasonably fast given that this is an offline process; and (2) the execution time grows nearly linearly in the size of the data (though can be quadratic in the size of a partition after pre-processing), showing scalability of our techniques.

5.3 Results on DBLP data

5.3.1 *XD* data set

The golden standard contains eighi clusters: the *Xin* cluster has 36 records in years 2003—2010, including name *Dong Xin* and two affiliations (*UIUC, MSR*); the *Dong* cluster has 29 records in years 2003—2010, including three names (*Xin*

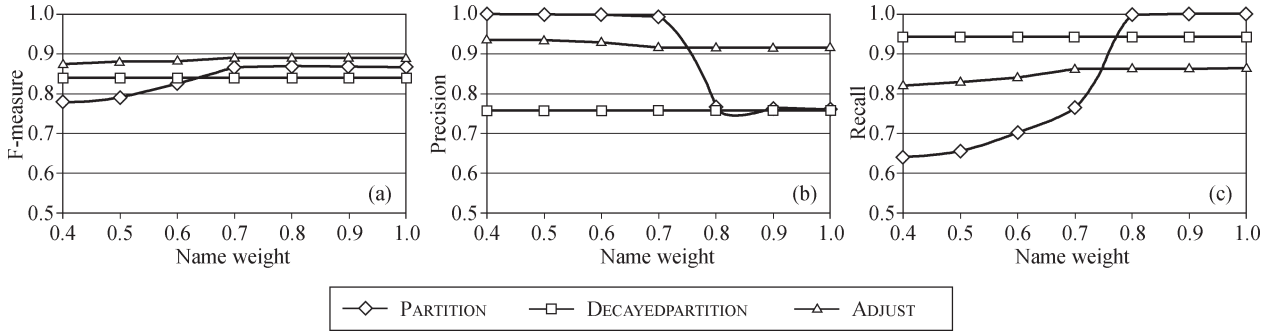

Fig. 14 Comparison of applying different attribute weights. (a) F-measure of varying name weights, (b) Precision of varying name weights, (c) Recall of varying name weights

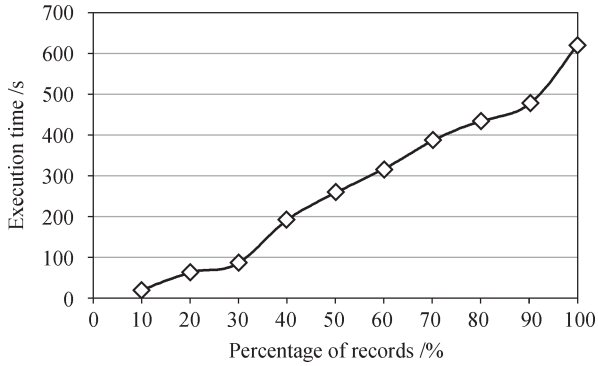

Fig. 15 Scalability of ADJUST

Dong, Luna Dong, Xin Luna Dong) and three affiliations (*UW, Google, AT&T*); the rest each have one or two records, including one name *Xin Dong* and one affiliation.

ADJUST results in 9 clusters and makes only one mistake: it splits the *Xin* records in 2009 with affiliation *UIUC* from the rest of *Xin* records. This is because Xin moved to *MSR* in 2008, so ADJUST considers the two affiliations as conflicting. We highlight that (1) ADJUST fixes an error in DBLP: it (correctly) separates the records with affiliation *UNL* from the *Dong* cluster; and (2) ADJUST is able to distinguish the various people, even though their names are exactly the same or very similar (the similarity between *Xin Dong* and *Dong Xin* is set to 0.8).

Figure 16(a) shows the results of various methods on this data set. ADJUST improves over baseline methods by 37%—43%. Other observations are similar to those on the Patent data set (see Fig. 16(b)—(c)), except that applying decay to some baseline methods (PARTITION and CENTER) can considerably reduce the precision and result in a low F-measure, as this data set is small and extremely difficult.

5.3.2 *WW* data set

We first report results on the 302 records for which DBLP has identified 18 clusters, of which (1) three involve two affiliations, two involve three affiliations, and one involves tour affiliations, so in total 10 affiliation transitions; (2) two authors share the same affiliation *Fudan*; (3) the largest cluster contains 92 records, the smallest contains one record, and six clusters contain more than ten records.

ADJUST obtains both high precision (0.98) and high recall (0.97). We highlight that (1) ADJUST is able to distinguish the different authors in most cases; (2) of the ten transitions, ADJUST identifies tive of them. ADJUST makes four types of mistakes: (1) it merges the two *Fudan* clusters, as one of them contains a single record with the year in the middle of the time period of the other cluster; (2) it merges the big *Fudan* cluster with another record, whose affiliation appears different from the rest in its own cluster, and time stamp is one

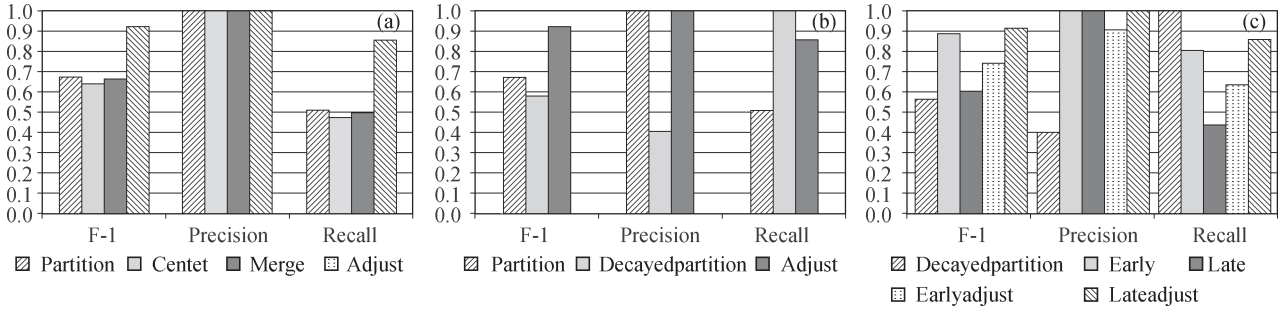

Fig. 16 Results of XD data set. (a) Overall results of XD data set, (b) Different components on XD data set, (c) Clustering results of XD data set

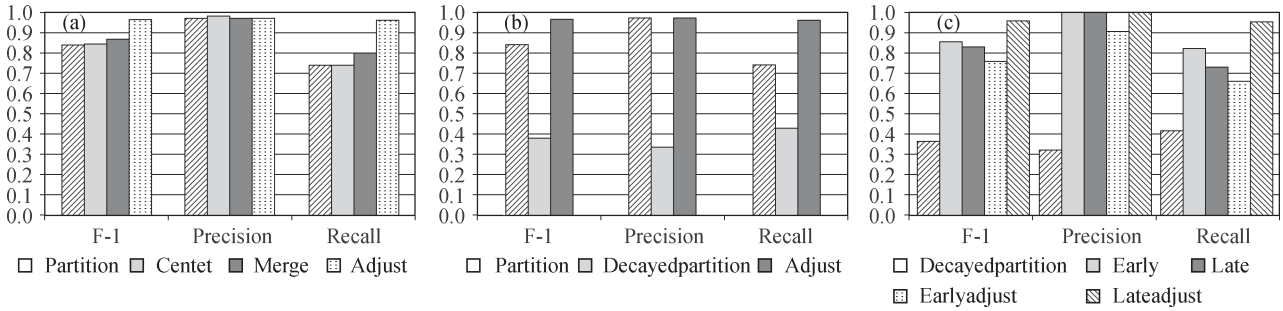

Fig. 17 Results of WW data set. (a) Overall results of WW data set, (b) Different components on WW data set, (c) Clustering results of WW data set

year before the earliest record in the big *Fudan* cluster, and so makes a strong case for the adjusting step; (3) it does not identify one of the transitions for the same reason as in the *XD* data set; and (4) it does not identify the other tour transitions because there are very few records for one of the affiliations and so not enough evidence for merging. Finally, Fig. 17 shows that ADJUST is significantly better than all other methods.

In the complete DBLP WW data set, 124 other WW records are merged with these 18 clusters and we manually verified the correctness. Among them, 63 are correctly merged, fixing errors from DBLP; 26 are wrongly merged but can be correctly separated if we have department information for a liation; and 35 are wrongly merged mainly because of the high similarity of affiliations (e.g., many records with *"technology"* in a liation are wrongly merged because the IDF of *"technology"* is not so low on this small data set). If we count these additional records, we are still able to obtain a precision of 0.94 and a recall of 0.94.

6 Related work

There are two bodies of work similar to ours: linkage techniques, and works regarding temporal information.

Record linkage Record linkage has been extensively studied in recent years [1,2]. To the best of our knowledge, existing techniques do not consider evolution of entities over time and treat the data as snapshot data. Our techniques differ from them in two aspects: the way we compute record similarity and the way we cluster records.

For record-similarity computation, existing works can be divided into three categories: *classification-based* approaches [8], classify a pair of records as *match, unmatch* and *maybe*; *distance-based* approaches [9], apply distance metrics to compute similarity of each attribute, and take the weighted sum as the record similarity; *rule-based* approaches [10], apply domain knowledge to match records. Our work falls in the *distance-based* category; however, we apply decay such that the weights we use for combining attribute similarities are functions of the time difference between the records, so we are tolerant of value evolution over time.

Many record linkage techniques, especially classification-based approaches, require learning parameters or classification models from learning data [8,11,12]. Their learning techniques all assume that record values do not change over time and value differences are due to different representations of the same value (e.g., "Google" and "Google, Inc."). We also learn parameters from learning data, but we are different in that we take into account possible value change over time; the decay curves we learn can be considered as consisting of parameters learned for different time gaps.

Relational entity resolution techniques take entity relationships (e.g., co-author, co-citation) into account when computing record similarity [13–15]. Our techniques also consider such multi-valued attributes, but we apply agreement decay and give less reward to similar values of such attributes in case of a big time gap.

For record clustering, there exists a wealth of literature on clustering algorithms for record linkage [7]. Among them, unconstrained and unsupervised algorithms that result in disjoint clusters are closest to ours. These algorithms may apply the transitive rule and efficiently perform clustering by a single scan of record pairs (e.g., *Partition algorithm* [10]), may iteratively specify seeds of clusters and assign vertexes to the seeds (e.g., *Ricochet algorithm* [16]), and may perform clustering by solving an optimization problem (e.g., *Cut clustering* [17]). These methods typically consider the records in decreasing order of record similarity while we consider the records in time order and collect evidence globally. Thus, our techniques do not necessarily merge records with high value similarity if the resulting entity shows erratic changes in a time period, and do not necessarily split records with low value similarity if value evolution over time is likely.

The techniques closest to ours can be found in [18] and

[19]. In [18] the authors study *behavior based linkage* where it leverages the periodical *behavior patterns* of each entity in linking pairs of records and learns such patterns from transaction logs. Their behavior pattern is different from the decay in our techniques in that decay learns the probability of value changes over time for *all* entities. In addition, we do not require a fixed and repeated value change pattern of particular entities, and we apply decay in a global fashion (rather than just between pairs of records) such that we can handle value evolution over time. Burdick et al. [19] applies domain-dependent rules to leverage temporal information in linking records, while we are the first to present a theoretical model that can be applied generally.

Temporal data A suite of temporal data models [20], temporal knowledge discovery paradigms [21] and data currency models [6] have been proposed in the past; however, we are not aware of any work focusing on linking temporal records. The notion of decay has recently been proposed in the context of data warehouses and streaming data [22,23]. They use decay to reduce the effect of older tuples on data analysis. Of them, *backward* decay [22] measures time difference backward from the latest time and *forward* decay [23] measures time difference forward from a fixed landmark. Their decay function is either binary or a fixed (exponential or polynomial) function. We differ in that 1) we consider time difference between two records rather than from a fixed point, and 2) we learn the decay curves purely from the data rather than using a fixed function.

7 Conclusions and future work

This article studied linking records with temporal information. We apply decay in record-similarity computation and consider the time order of records in clustering; thus, our linkage technique is tolerant of entity evolution over time and can glean evidence globally for decision making. Future work includes combining temporal information with other dimensions of information such as spatial information to achieve better results, considering erroneous data especially erroneous time stamps, and combining our work with recent work on inferring temporal ordering of records [6] for linkage.

References

1. Elmagarmid A, Ipeirotis P, Verykios V. Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(1): 1–16
2. Koudas N, Sarawagi S, Srivastava D. Record linkage: similarity measures and algorithms. In: Proceedings of the 25th ACM SIGMOD International Conference on Management of Data. 2006, 802–803
3. Weikum G, Ntarmos N, Spaniol M, Triantafillou P, Bencz ú r A, Kirkpatrick S, Rigaux P, Williamson M. Longitudinal analytics on web archive data: It ' s about time! In: Proceedings of the Biennial Conference on Innovative Data Systems Research. 2011, 199–202
4. McCallum A, Nigam K, Ungar L. Efficient clustering of highdimensional data sets with application to reference matching. In: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2000, 169–178
5. Li P, Dong X, Maurino A, Srivastava D. Linking temporal records. Proceedings of the VLDB Endowment, 2011, 4(7): 956–967
6. Fan W, Geerts F, Wijsen J. Determining the currency of data. In: Proceedings of the 30th Symposium on Principles of Database Systems of Data. 2011, 71–82
7. Hassanzadeh O, Chiang F, Lee H, Miller R. Framework for evaluating clustering algorithms in duplicate detection. Proceedings of the VLDB Endowment, 2009, 2(1): 1282–1293
8. Fellegi I P, Sunter A B. A theory for record linkage. Journal of the American Statistical Association, 1969, 64(328): 1183–1210
9. Dey D. Entity matching in heterogeneous databases: A logistic regression approach. Decision Support Systems, 2008, 44(3): 740–747
10. Hern á ndez M, Stolfo S. Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery, 1998, 2(1): 9–37
11. Domingos P. Multi-relational record linkage. In: In Proceedings of the KDD-2004 Workshop on Multi-Relational Data Mining. 2004, 31–48
12. Winkler W. Methods for record linkage and bayesian networks. Technical report, Statistical Research Division, US Census Bureau, Washington, DC, 2002
13. Ananthakrishna R, Chaudhuri S, Ganti V. Eliminating fuzzy duplicates in data warehouses. In: Proceedings of the 28th International Conference on Very Large Data Bases. 2002, 586–597
14. Chen Z, Kalashnikov D, Mehrotra S. Exploiting relationships for object consolidation. In: Proceedings of the 2nd International Workshop on Information Quality in Information Systems. 2005, 47–58
15. On B, Koudas N, Lee D, Srivastava D. Group linkage. In: Proceedings of IEEE 23rd International Conference on the Data Engineering. 2007, 496–505
16. Wijaya D, Bressan S. Ricochet: A family of unconstrained algorithms for graph clustering. In: Database Systems for Advanced Applications. 2009, 153–167
17. Flake G, Tarjan R, Tsioutsiouliklis K. Graph clustering and minimum cut trees. Internet Mathematics, 2004, 1(4): 385–408
18. Yakout M, Elmagarmid A, Elmeleegy H, Ouzzani M, Qi A. Behavior based record linkage. Proceedings of the VLDB Endowment, 2010, 3(1-2): 439–448
19. Burdick D, Hern á ndez M A, Ho H, Koutrika G, Krishnamurthy R, Popa L, Stanoi I, Vaithyanathan S, Das S R. Extracting, linking and integrating data from public sources: A financial case study. IEEE Data Engineering, 2011, 34(3): 60–67
20. Ozsoyoglu G, Snodgrass R. Temporal and real-time databases: A survey. IEEE Transactions on Knowledge and Data Engineering, 1995,

7(4): 513–532

21. Roddick J, Spiliopoulou M. A survey of temporal knowledge discovery paradigms and methods. IEEE Transactions on Knowledge and Data Engineering, 2002, 14(4): 750–767
22. Cohen E, Strauss M. Maintaining time-decaying stream aggregates. Journal of Algorithms, 2006, 59(1): 19–36
23. Cormode G, Shkapenyuk V, Srivastava D, Xu B. Forward decay: A practical time decay model for streaming systems. In: Proceedings of the IEEE 25th International Conference on Data Engineering. 2009, 138–149

Pei Li is a Ph.D student at Università di Milano Bicocca. Currently she is nearing the completion of her doctoral thesis in Computer Science. Previously, she studied electronic engineering at Beijing University of Posts and Telecommunications, where she received her BS and MS degrees. Her research interests are data integration and record linkage, with special focus on entity resolution with value inconsistency.

Xin Luna Dong is a researcher at AT&T Labs-Research. She received her PhD from University of Washington in 2007, received her Master's Degree from Peking University in China in 2001, and her Bachelor's Degree from Nankai University in China in 1998. Her research interests include databases, information retrieval, and machine learning, with an emphasis on data integration, data cleaning, personal information management, and Web search. She is co-chairing Sigmod/PODS PhD Symposium 2012, Sigmod New Researcher Symposium 2012, and QDB (Quality of DataBases) 2012, has co-chaired WebDB'10, was a co-editor of the IEEE Data Engineering special issue on Towards quality data with fusion and cleaning, and has served in the program committees of VLDB'12, Sigmod'12, VLDB'11, Sigmod'11, VLDB'10, WWW'10, ICDE'10, and VLDB'09.

Andrea Maurino is an assistant professor at Università di Milano Bicocca, his research interest covers many areas in the field of database systems and service science. In the field of data quality, his research interests are: record linkage, and cooperative information systems, assessment techniques of data intensive web applications. In the field of service science he focuses on the analysis of quality of services and non functional properties. He is author of more 40 papers including international journals and conferences; he is also the author of 4 book chapters. He was program co-chair of QDB'09 workshop, guest editor of IEEE Internet Computing in 2010. He is a reviewer for several journals including Information Systems, Knowledge Data and Engineering.

Divesh Srivastava is the head of the Database Research Department at AT&T Labs-Research. He received his PhD. from the University of Wisconsin, Madison, and his BTech from the Indian Institute of Technology, Bombay. He is a Fellow of the ACM, on the board of trustees of the VLDB Endowment, and an associate editor of the ACM Transactions on Database Systems. He has served as the associate Editor-in-Chief of the IEEE Transactions on Knowledge and Data Engineering, and the program committee co-chair of many conferences, including VLDB 2007. He has presented keynote talks at several conferences, including VLDB 2010. His research interests span a variety of topics in data management.