Linking Temporal Records

Pei Li
University of Milan-Bicocca
pei.li@disco.unimib.it

Xin Luna Dong
AT&T Labs-Research
lunadong@research.att.com

Andrea Maurino
University of Milan-Bicocca
maurino@disco.unimib.it

Divesh Srivastava
AT&T Labs-Research
divesh@research.att.com

ABSTRACT

Many data sets 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 (*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 enable interesting longitudinal data analysis. However, existing record linkage techniques ignore the temporal information and can fall short for temporal data.

This paper 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.

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 [7, 12] for recent surveys). Existing techniques typically proceed in two steps: the first step compares similarity between each pair of records, deciding if they match or do not match; 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 contain customer information over years, and so on; identifying records that refer to the same entity enables interesting longitudinal data analysis over such data [17].

Table 1: Records from DBLP.

ID	name	affiliation	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

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

EXAMPLE 1.1. *Consider records that describe paper authors in Table 1; each record is derived from a publication record at DBLP (we may skip some co-authors for space reason). These records describe 3 real-world persons:* 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* affiliation, *we may split entities* E_2 *and* E_3*, as records for each of them can have different values for* affiliation. *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 of the three entities.* □

Despite the challenges, temporal information does present 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

[1] http://www.informatik.uni-trier.de/∼ley/db/.

Proceedings of the VLDB Endowment, Vol. 4, No. 11
Copyright 2011 VLDB Endowment 2150-8097/11/08... $ 10.00.

years. Second, record values seldom change *erratically*. In our example, $r_2, r_3, r_7, r_8, r_{10}$ 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 paper 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 applied 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 already improve linkage results, and applying our clustering methods can obtain results with high precision and recall (Section 5).

This paper focuses on improving quality (precision and recall) of linking temporal records. We can improve efficiency of linkage by applying previous techniques such as canopy [13] to create small blocks of records that are candidates for temporal linkage.

2. OVERVIEW

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.*, person affiliation, business addresses). We distinguish *single-valued* and *multi-valued* attributes, where the difference is whether for an attribute an entity can have a single or multiple values at any time. Consider a set $\mathbf{R}$ of records, each 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 2.1 (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 2.2. *Consider the records in Table 1, where each record describes an author by her* name*,* affiliation*, 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:* $\{r_1\}, \{r_2, \ldots, r_6\}, \{r_7, \ldots, r_{12}\}$. □

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 1.1). 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 penalty for value disagreement and reduce reward for value agreement over a long period. Our experimental results show that applying decay in similarity computation can already improve over traditional linkage techniques.

Second, when clustering records according to record similarity, traditional techniques do not consider time order of the records. However, time order can often provide important clues. In Example 1.1, 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 1.1 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 strong indicator of referring to different entities. We define *disagreement decay* to capture this intuition.

DEFINITION 3.1 (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 1.1 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 3.2 (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. □

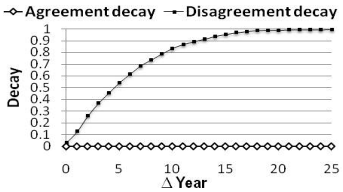


Figure 1: Address **decay curves.**

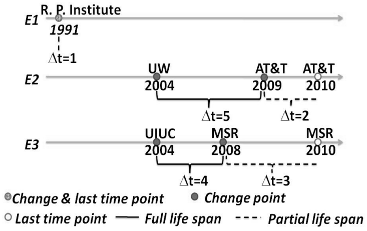


Figure 2: Learning $d^{\neq}(\mathsf{aff}, \Delta t)$**.**

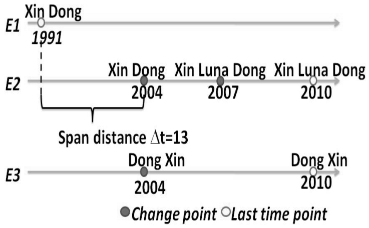


Figure 3: Learning $d^{=}(\mathsf{name}, \Delta t)$**.**

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. Second, decay observes *monotonicity*; that is, for any $\Delta t < \Delta t'$ and any attribute A, $d^{\neq}(A, \Delta t) \leq d^{\neq}(A, \Delta t')$ and $d^{=}(A, \Delta t) \leq d^{=}(A, \Delta t')$. 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 3.3. *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).*

We observe that (1) the disagreement decay increases from 0 to 1 when 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 address match does not correspond to record match and vice versa.* □

3.2 Learning decay

Decay can be specified by domain experts or learnt 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 remove records that violate the first assumption. 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.

Consider attribute A and time period Δt. We next describe how we calculate the decay for A and Δt according to the labels. Appendix A describes alternative ways of learning decay, whose results lead to similar linkage results in our experiments.

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 $r_1, \ldots, r_n, n \geq 1$. We call a time point t a *change point* if r_1 is at time t or if at time t there exists a record $r_i, i \in [2, n]$, whose A-value is different from r_{i-1}. For each change point t (associated with a new value), we can compute a *life span*: if t is not the last change point of E, we call the life span 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 last time stamp for this value and δ denotes one time unit. 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. We denote by $\bar{L}_f$ the bag of lengths of full life spans, and by $\bar{L}_p$ that for partial life spans.

To learn $d^{\neq}(A, \Delta t)$, we consider all full life spans and the partial life spans with at least length Δt (we do not know for others if the value will change in time Δt). We compute the decay as

$$d^{\neq}(A, \Delta t) = \frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|}. \quad (1)$$

We give details in Algorithm LEARNDISAGREEDECAY (see Appendix A). We can prove that the decay it learns satisfies the monotonicity property (proofs of all results are in the appendix).

PROPOSITION 3.4. *Let* A *be an attribute. For any* $\Delta t < \Delta t'$*, the decay learned by Algorithm* LEARNDISAGREEDECAY *satisfies* $d^{\neq}(A, \Delta t) \leq d^{\neq}(A, \Delta t')$. □

EXAMPLE 3.5. *Consider learning disagreement decay for* affiliation *from the data in Example 1.1. For illustrative purpose, we remove record* r_{10} *as its affiliation information is incorrect. Take* E_2 *as an example. As shown in Figure 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.*

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

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 $t_1 \leq t_3$. Then, for any $\Delta t \geq \max\{0, t_3 - t_2 + \delta\}$, 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]

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

$$d^{=}(A, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}. \quad (2)$$

Algorithm LEARNAGREEDECAY (Appendix A) describes the details and we next show monotonicity of its results.

[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.

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

PROPOSITION 3.6. *Let A be an attribute. For any $\Delta t < \Delta t'$, the decay learned by Algorithm* LEARNAGREEDECAY *satisfies* $d^{=}(A, \Delta t) \leq d^{=}(A, \Delta t')$. □

EXAMPLE 3.7. *Consider learning agreement decay for* name *from data in Example 1.1. As shown in Figure 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* $[2004, 2007)$. *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,* $d^{=}(\mathsf{name}, \Delta t \in [0, 12]) = \frac{0}{3} = 0$, *and* $d^{=}(\mathsf{name}, \Delta t \geq 13) = \frac{1}{3} = 0.33$. □

3.3 Applying decay

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

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 is an attribute in record-similarity computation, so the less penalty for value disagreement or the less 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

$$sim(r, r') = \frac{\sum_{A \in \mathbf{A}} w_A(s(r.A, r'.A), |r.t - r'.t|) \cdot s(r.A, r'.A)}{\sum_{A \in \mathbf{A}} w_A(s(r.A, r'.A), |r.t - r'.t|)}. \quad (3)$$

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 3.8. *Consider records r_2 and r_5 in Example 1.1 and we focus on single-valued attributes* name *and* affiliation. *Assume the name similarity between r_2 and r_5 is .9 and the affiliation similarity is 0. Suppose* $d^{=}(\mathsf{name}, \Delta t = 5) = .05, d^{\neq}(\mathsf{aff}, \Delta t = 5) = .9$, *and* $\theta_h = .8$. *Then, the weight for* name *is* $1 - .05 = .95$ *and that for* affiliation *is* $1 - .9 = .1$. *So the similarity is* $sim(r_1, r_2) = \frac{.95 * .9 + .1 * 0}{.95 + .1} = .81$. *Note that if we do not apply decay and assign the same weight to each attribute, the similarity would become* $\frac{.5 * .9 + .5 * 0}{.5 + .5} = .45$.

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 with r_1. □

4. TEMPORAL CLUSTERING

As shown in Example 3.8, 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 (details in Appendix C) 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 obviously $r.t$. 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 cluster 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 $C.A$.

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

EXAMPLE 4.1. *Consider applying early binding to records in Table 1. 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* affiliation *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 with C_2 (r_{10} and r_9 have a short time distance but different affiliations), but a high similarity with C_1 (fairly similar name and high disagreement decay *on* affiliation *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 bi-partite 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 Figure 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.

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 $C_1, \dots, C_n$, it proceeds in three steps.

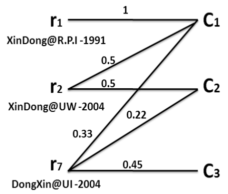
1. Compute the similarity between r and each $C_i, i \in [1, n]$.
2. Create a new cluster C_{n+1} and assign similarity as follows. (1) If for each $i \in [1, n]$, $sim(r, C_i) \leq \theta$, we consider that r is unlikely to belong to any C_i and set $sim(r, C_{n+1}) = \theta$. (2) If there exists $i \in [1, n]$, 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 last 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. Note also that in practice, we may set low similarities to 0 to improve efficiency; we discuss the details in Appendix D.

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 each 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 $r_1, \dots, r_m$ 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

$$sim(r, C) = \sum_{i=1}^{m} sim(r, r_i) P(r_i) \Pi_{j=i+1}^{m} (1 - P(r_i)). \qquad (4)$$



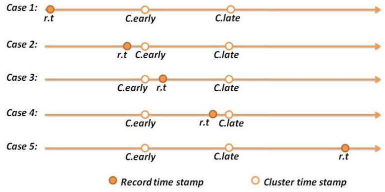


Figure 4: Ex. 4.2. A part of the bi-partite graph.

Figure 5: Continuity between record r and cluster C.

EXAMPLE 4.2. *Consider applying late binding to the records in Table 1 and let $\theta = .8$. Figure 4 shows a part of the bi-partite 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 ($.89 > \theta$) is high but that without decay ($.5 < \theta$) is low. We thus create a new cluster C_2 and set $sim(r_2, C_2) = .89$. After normalization, each edge from r_2 has a weight of .5.*

Now consider r_7. For C_1, with probability .5 we shall compare r_7 with r_2 (suppose $sim(r_7, r_2) = .4$) and with probability 1-.5=.5 we shall compare r_7 with r_1 (suppose $sim(r_7, r_1) = .8$). Thus, $sim(r_7, C_1) = .8 * .5 + .4 * .5 = .6 < \theta$. For C_2, we shall compare r_7 only with r_2 and the similarity is $.4 < \theta$. Because of the low similarities, we create a new cluster C_3 and set $sim(r_7, C_3) = .8$. After normalization, the probabilities from r_7 to C_1, C_2 and C_3 are .33, .22 and .45 respectively. □

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 $\mathcal{C}$ be a clustering and we denote by $\mathcal{C}(r)$ the cluster to which r belongs in $\mathcal{C}$. We can compute the probability of $\mathcal{C}$ as $\Pi_{r \in \mathbf{R}} P(r \in \mathcal{C}(r))$, where $P(r \in \mathcal{C}(r))$ denotes the probability that r belongs to $\mathcal{C}(r)$. 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 LATE algorithm in Appendix D and next state the optimality of the decision-making stage.

PROPOSITION 4.3. LATE *algorithm runs in time $O(|\mathbf{R}|^2)$ and chooses the clustering with the highest probability among all possible valid clusterings.* □

EXAMPLE 4.4. *Continue with Example 4.2. After evidence collection, we created 5 clusters and the weight of each record-cluster pair is shown in Table 2. Weights of selected edges are in bold.*

We first select edge (n_{r_1}, n_{C_1}) with weight 1. We then choose (n_{r_2}, n_{C_2}) with weight .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 .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 5 clusters: $\{r_1\}$, $\{r_2, \dots, r_6\}$, $\{r_7, r_8\}$, $\{r_9, r_{11}, r_{12}\}$, $\{r_{10}\}$.

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. □

Table 2: Example 4.4. 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**	.5	.33	.37	.27	.38	.16	.13	.18	.24	.12	.22
C_2	0	**.5**	.22	**.4**	.25	**.4**	.16	.12	.17	**.27**	.1	**.24**
C_3	0	0	**.45**	.23	**.48**	.22	.24	.26	.2	.17	.23	.18
C_4	0	0	0	0	0	0	**.44**	.19	**.29**	.16	**.33**	.18
C_5	0	0	0	0	0	0	0	**.3**	.16	.16	.22	.18

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 1.1, 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 deterministic algorithm and Appendix E describes the probabilistic one.

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 describe how we compute value consistency next and discuss occurrence frequency in Appendix E. Consider the value consistency between r and $C = \{r_1, \dots, r_m\}$ (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 $C_1 = \{r_1, \dots, r_k\}$ is low; 2) r_{k+1} may be inconsistent with r, so the similarity between r_{k+1} and the sub-cluster $C_2 = \{r_1, \dots, r_k, r\}$ is low. We take the minimum as $cons(r, C)$:

$$cons(r, C) = \min(sim(r, C_1), sim(r_{k+1}, C_2)). \tag{5}$$

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 Figure 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.

$$cont(r, C) = e^{-\lambda y}; \tag{6}$$

$$y = \frac{|r.t - C.early| + \alpha}{C.late - C.early + \alpha}. \tag{7}$$

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 Figure 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 shall 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

$$sim(r, C) = cons(r, C) \cdot cont(r, C). \tag{8}$$

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 long time gap.

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

EXAMPLE 4.5. *Consider r_{10} and C_4 in the results of Example 4.4. For value consistency, inserting r_{10} into C_4 results in $\{r_9, \dots, r_{12}\}$. Assume $sim(r_{10}, \{r_9\}) = sim(r_{11}, \{r_9, r_{10}\}) = .6$. Then, $cons(r_{10}, C_4) = .6$. 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 $.26 * .6 = .16$. On the other hand, the similarity between r_{10} and C_5 is $1 \cdot e^{-2\cdot\frac{1}{1}} = .14$. 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. □

5. EXPERIMENTAL EVALUATION

This section describes experimental results on two real data sets. We show that (1) our technique significantly improves over 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 (see Appendix F).

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.

[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 shall 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 $cons(r_1, C_2) > cons$.

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

Table 3: 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

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 potpourri). For each subset we extracted Author records with attributes name, affiliation, 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 3 shows statistics of the data sets.

Implementation: We learned decay from both Patent data sets. The decay we learned for the address attribute is shown in Figure 1; 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 on 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 [10]. They all compute pairwise record similarity but apply different clustering strategies. Appendix F.1 gives the details.
- *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.

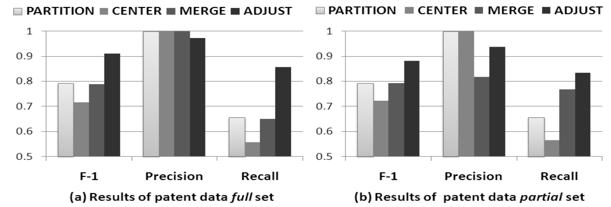
We present details of similarity computation in Appendix F.1. By default, when we computed the record similarity without applying decay, we used weight .5 for both name and address (or affiliation). No matter whether we applied decay, we used weight .3 for co-author. We applied threshold .8 for deciding if a similarity is high in various contexts. In addition, we set $\theta_h = .8, \theta_l = .6, \lambda = .5, \alpha = 1$ in our methods. We vary these parameters and present some results in Appendix F.2 to demonstrate robustness.

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

Measure: We compared pairwise linking decisions with the golden standard and measured 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 on the *partial* data set, which is harder. Among 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.



(a) Results of patent data *full* set (b) Results of patent data *partial* set

Figure 6: Results on the patent data set.

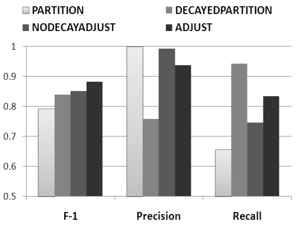


Figure 7: Contribution of different components.

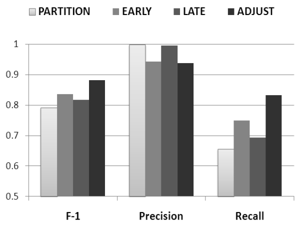


Figure 8: Different clustering methods on patent *partial* data.

Different components: 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 increases the recall a lot, 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 increases the recall quite a bit without reducing the precision much. Finally, we observe that applying agreement decay does not change the results much, since the agreement decays of both attributes are close to 0.

Different clustering methods: Figure 8 compares early, late, and adjusted binding. We observe that they all 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 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 the precision much.

5.3 Results on DBLP data

***XD* data set:** The golden standard contains 8 clusters: the *Xin* cluster has 36 records in years 2003-2010, including name *Dong Xin* and 2 affiliations (*UIUC, MSR*); the *Dong* cluster has 29 records in years 2003-2010, including 3 names (*Xin Dong, Luna Dong, Xin Luna Dong*) and 3 affiliations (*UW, Google, AT&T*); the rest of them each has 1 or 2 records, including 1 name *Xin Dong* and 1 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 .8).

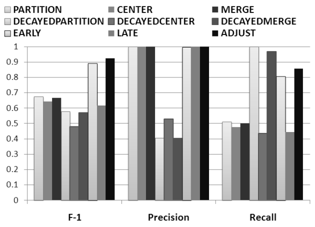


Figure 9: Results on *XD* set.

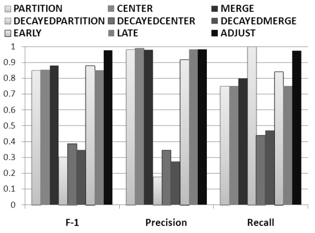


Figure 10: Results on *WW* set.

Figure 9 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, except that applying decay to some baseline methods (PARTITION and MERGE) can considerably reduce the precision and result in a low F-measure, as this data set is small and extremely difficult.

***WW* data set:** We first report results on the 302 records for which DBLP has identified 18 clusters, among which (1) 3 involve 2 affiliations, 2 involve 3 affiliations, and 1 involves 4 affiliations, so in total 10 affiliation transitions; (2) two authors share the same affiliation *Fudan*; (3) the largest cluster contains 92 records, the smallest one contains 1 record, and 6 clusters contain more than 10 records. ADJUST obtains both high precision (.98) and high recall (.97). We highlight that (1) ADJUST is able to distinguish the different authors in most cases; (2) among the 10 transitions, ADJUST identifies 5 of them. ADJUST makes four types of mistakes: (1) it merges the two *Fudan* clusters, as one of them contains a single record with 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 year before the earliest record in the big *Fudan* cluster, so makes a hard 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 4 transitions because there are very few records for one of the affiliations and so not enough evidence for merging. Finally, Figure 10 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 affiliation; and 35 are wrongly merged mainly because of high similarity of affiliations (*e.g.*, many records with *"technology"* in affiliation 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 .94 and a recall of .94.

6. RELATED WORK AND CONCLUSIONS

Related work: We have compared our techniques with traditional record-linkage techniques in Section 2 and in experiments; Appendix G gives more details. We next compare with related works regarding temporal information.

A suite of temporal data models [15] and temporal knowledge discovery paradigms [16] have been proposed in the past; however, we are not aware of any work focusing on linking temporal records. Behavior based linkage [20] leverages 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 apply decay in a global fashion (rather than just between pairs of records) such that we can handle value evolution over time.

The notion of decay has been proposed in the context of data warehouses and streaming data recently [3, 4]. They use decay to reduce the effect of older tuples on data analysis. Among them, *backward* decay [3] measures time difference backward from the latest time and *forward* decay [4] 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.

Conclusions and future work: This paper 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, and considering erroneous data especially erroneous time stamps.

7. REFERENCES

[1] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

[2] Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Exploiting relationships for object consolidation. In *IQIS*, pages 47–58, 2005.

[3] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *PODS*, pages 223–233, 2003.

[4] G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu. Forward decay: A practical time decay model for streaming systems. *ICDE*, pages 138–149, 2009.

[5] D. Dey. Entity matching in heterogeneous databases: A logistic regression approach. *Decis. Support Syst.*, 44:740–747, 2008.

[6] P. Domingos. Multi-relational record linkage. In *Proceedings of the KDD Workshop on Multi-Relational Data Mining*, pages 31–48.

[7] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[8] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[9] G. Flake, R. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1:385–408, 2004.

[10] O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, pages 1282–1293, 2009.

[11] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2:9–37, 1998.

[12] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.

[13] A. K. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proc. of SIGKDD*, pages 169–178, 2000.

[14] B. W. On, N. Koudas, D. Lee, and D. Srivastava. Group linkage. In *ICDE*, pages 496–505, 2007.

[15] G. Ozsoyoglu and R. Snodgrass. Temporal and real-time databases: a survey. *TKDE*, 7:513–532, 1995.

[16] J. F. Roddick and M. Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *TKDE*, 14:750–767, 2002.

[17] G. Weikum, N. Ntarmos, M. Spaniol, P. Triantafillou, A. Benczúr, S. Kirkpatrick, P. Rigaux, and M. Williamson. Longitudinal analytics on web archive data: It's about time! In *CIDR*, pages 199–202, 2011.

[18] D. T. Wijaya, S. Bressan, J. Joxan, and D. T. Wijaya. Ricochet: A family of unconstrained algorithms for graph clustering. In *DASFAA*, pages 153–167, 2009.

[19] W. E. Winkler. Methods for record linkage and bayesian networks. Technical report, Series RRS2002/05, U.S. Bureau of the Census, 2002.

[20] M. Yakout, A. K. Elmagarmid, H. Elmeleegy, M. Ouzzani, and A. Qi. Behavior based record linkage. *PVLDB*, 3:439–448, 2010.

APPENDIX

A. DETAILS OF LEARNING DECAY

We describe Algorithms LearnDisagreeDecay and LearnAgreeDecay and show the monotonicity of their results.

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

```
Input: $\mathcal{C}$ 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 $d^{\neq}(\Delta t, A)$.
$\bar{L}_f = \phi$; $\bar{L}_p = \phi$;
for each $C \in \mathcal{C}$ do
    sort records in $C$ in increasing time order to $r_1, \ldots, r_{|C|}$;
    // Find life spans
    $start = 1$;
    while $start \leq |C|$ do
        $end = start + 1$;
        while $r_{start}.A = r_{end}.A$ and $end \leq |C|$ do
            $end++$;
        end while
        if $end > |C|$ then
            insert $r_{|C|}.t - r_{start}.t + \delta$ into $\bar{L}_p$; // partial life span
        else
            insert $r_{end}.t - r_{start}.t$ into $\bar{L}_f$; // full life span
        end if
        $start = end$;
    end while
end for
// Learn decay
for $\Delta t = 1, \ldots, \max_{l \in \bar{L}_f \cup \bar{L}_p}\{l\}$ do
    $d^{\neq}(A, \Delta t) = \frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|}$
end for
```

Proof Monotonicity. In LearnDisagreeDecay, as Δt increases, $|\{l \in \bar{L}_f | l \leq \Delta t\}|$ increases while $|\{l \in \bar{L}_p | l \geq \Delta t\}|$ decreases; thus, $\frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|}$ increases.

In LearnAgreeDecay, as Δt increases, $|\{l \in \bar{L} | l \leq \Delta t\}|$ increases; thus, $\frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}$ increases. □

We next describe two alternatives of decay learning, namely, *single-count* decay and *probabilistic* decay for disagreement decay (similar for agreement decay). Experimental results show that they learn similar (but more or less smooth) decay curves as LearnDisagreeDecay and LearnAgreeDecay.

Single-count decay considers an entity at most once and learns 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 one 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 in any longer time. We learn disagreement decay using Eq.(1).

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

$$d^{\neq}(A, \Delta t) = \frac{\sum_{l \in \bar{L}_f, l \leq \Delta t} p(l)}{\sum_{l \in \bar{L}_f} p(l) + |\{l \in \bar{L}_p | l \geq \Delta t\}|}. \quad (9)$$

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

```
Input: $\mathcal{C}$ 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.
Output: Agreement decay $d^{=}(\Delta t, A)$.
//Find life spans
for each $C \in \mathcal{C}$ do
    sort records in $C$ in increasing time order to $r_1, \ldots, r_{|C|}$;
    $start = 1$;
    while $start \leq |C|$ do
        $end = start + 1$;
        while $r_{start}.A = r_{end}.A$ and $end \leq |C|$ do
            $end++$;
        end while
        if $end > |C|$ then
            $r_{start}.t_{next} = r_{|C|-1}.t + \delta$; // partial life span
        else
            $r_{start}.t_{next} = r_{end}.t$; // full life span
        end if
        $start = end$;
    end while
end for
//Learn agreement decay
$\bar{L} = \phi$
for each $C, C' \in \mathcal{C}$ do
    $same = false$;
    for each $r \in C$ s.t. $r.t_{next} \neq \mathsf{null}$ do
        for each $r' \in C'$ s.t. $r.t_{next} \neq \mathsf{null}$ do
            if $r.A = r'.A$ then
                $same = true$;
                if $r.t \leq r'.t$ then
                    insert $\max\{0, r'.t - r.t_{next} + 1\}$ into $\bar{L}$;
                else
                    insert $\max\{0, r.t - r'.t_{next} + 1\}$ into $\bar{L}$;
                end if
            end if
        end for
    end for
    if $!same$ then
        insert $\infty$ into $\bar{L}$;
    end if
end for
for $\Delta t = 1, \ldots, \max_{l \in \bar{L}}\{l\}$ do
    $d^{=}(A, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}$
end for
```

B. LEARNING AND APPLYING DECAY FOR MULTI-VALUED ATTRIBUTES

In this section we consider multi-valued attributes such as co-authors. We start with 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 *un-match*; 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 of $r.A$ and $r'.A$, denoted by $s(r.A, r'.A)$, is computed by a variant of Jaccard distance between the two sets.

$$s(r.A, r'.A) = \frac{\sum_{v \in r.A, v' \in r'.A, s(v,v') > \theta_h} s(v, v')}{\min\{|r.A|, |r'.A|\}}. \quad (10)$$

If the relationship between the entities and the A-values are 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,

$$sim'(r, r') = \min\{1, sim(r, r') + \sum_{multi-valued\ A} w_A s(r.A, r'.A)\}. \quad (11)$$

On the other hand, if the relationship between the entities and the A-values are many-to-many, we apply Eq.(11) 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 and 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.(10) as follows.

$$\begin{aligned} & s(r.A, r'.A) \\ = & \frac{\sum_{v \in r.A, v' \in r'.A, s(v,v') > \theta_h} (1 - d^{=}(A, |r.t - r'.t|)) s(v, v')}{\min\{|r.A|, |r'.A|\}} \end{aligned} \quad (12)$$

EXAMPLE B.1. *Consider records r_2 and r_5 and multi-valued attribute* co-authors *(many-to-many relationship) in Example 1.1. Let $\theta_h = .8$ and $w_{\mathsf{co}} = .3$. Record r_2 and r_5 share one co-author with string similarity 1. Suppose $d^{=}(\mathsf{co}, \Delta t = 5) = .05$. Then, $s(r_2.\mathsf{co}, r_5.\mathsf{co}) = \frac{(1-.05)*1}{\min\{2,2\}} = .475$. Recall from Example 3.8 that $sim(r_2, r_5) = .81 > \theta_h$; therefore, the overall similarity is $sim'(r_2, r_5) = \min\{1, .81 + .475 * .3\} = .95$.* □

C. DETAILS OF EARLY BINDING

We describe algorithm EARLY in Algorithm 3.

D. DETAILS OF LATE BINDING

We describe algorithm LATE in Algorithm 4.

In Line 24 of the algorithm, we remove edges with low similarity scores for each record before we normalize the edge weights. We next describe several edge-deletion strategies. Our experimental results (Appendix F.2) 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 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.

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

```
Input: $\mathbf{R}$ records in increasing time order
Output: $\mathcal{C}$ clustering of records in $\mathbf{R}$
for each record $r \in \mathbf{R}$ do
    for each $C \in \mathcal{C}$ do
        compute record-cluster similarity $sim(r, C)$;
    end for
    if $\max_{C \in \mathcal{C}} sim(r, C) \geq \theta$ then
        $C = Argmax_{C \in \mathcal{C}} sim(r, C)$;
        insert $r$ into $C$;
        update signature of $C$;
    else
        insert cluster $\{r\}$ into $\mathcal{C}$;
    end if
end for
return $\mathcal{C}$;
```

PROOF PROPOSITION 4.3. 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. □

E. DETAILS OF ADJUST BINDING

We describe Algorithm ADJUST in Algorithm 5. Next, we describe frequency consistency and probabilistic adjusting.

Frequency consistency: Consider a cluster C. The *occurrence frequency* of C, denoted by $freq(C)$, is computed by

$$freq(C) = \frac{C_{late} - C_{early}}{|C|}. \quad (13)$$

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

$$cons_f(r, C) = 1 - \frac{|freq(C) - freq(C')|}{\max\{freq(C), freq(C')\}}. \quad (14)$$

In practise, we can also apply a window of size $n \leq |C|$, so that occurrence frequency is only considered within n records.

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.

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,

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

```
Input: R records in increasing time order
Output: $\mathcal{C}$ clustering of records in R
 1: Initialize a bi-partite graph $(N_R, N_C, E)$ where $N_R = N_C = E = \emptyset$;
 2: //Evidence collection
 3: for each record $r \in \mathbf{R}$ do
 4:     insert node $n_r$ into $N_R$;
 5:     for each $n_C \in N_C$ do
 6:         compute decayed record-cluster similarity $sim(r, C)$;
 7:     end for
 8:     if $\max_{n_C \in N_C} sim(r, C) \le \theta$ then
 9:         insert node $n_{C_r}$ into $N_C$;
10:         insert edge $(n_r, n_{C_r})$ with weight $\theta$ into $E$;
11:     else
12:         $newCluster = true$;
13:         for each $n_C \in N_C$, where $sim(r, C) > \theta$ do
14:             compute no decayed similarity $sim'(r, C)$;
15:             if $sim'(r, C) > \theta$ then
16:                 $newCluster = false$; break;
17:             end if
18:         end for
19:         if $newCluster$ then
20:             insert node $n_{C_r}$ into $N_C$;
21:             insert edge $(n_r, n_{C_r})$ with weight $\max_{n_C \in N_C, sim'(r,C) > \theta}(sim(r, C))$ into $E$;
22:         end if
23:     end if
24:     delete edges with low weights;
25:     normalize weights for all edges from $n_r$;
26: end for
27: // Decision making
28: while $|E| > |N_R|$ do
29:     select edge $(n_r, n_C)$ with maximal edge weight;
30:     remove edges $(n_r, n_{C'})$ for all $C' \neq C$;
31:     if cluster $C$ is created for record $r' \neq r$ then
32:         select edge $(n_{r'}, n_C)$;
33:         remove edges $(n_{r'}, n_{C'})$ for all $C' \neq C$;
34:     end if
35: end while
36: return $\mathcal{C}$;
```

except that we 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 $r_1.t \le r_2.t \le \cdots \le r_m.t$, each with probability $p_i, i \in [1, m]$. We compute C_{early} and C_{late} as follows.

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

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

F. DETAILS OF EXPERIMENTS

F.1 Details of implementations

Traditional methods: We implemented three traditional methods: PARTITION, CENTER, and MERGE [10]. All methods compute the similarity between a pair of records as the weighted sum of the attribute similarities. They differ in clustering methods:

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

```
Input: R records in increasing time order.
    $\mathcal{C}$ pre-clustering of records in R.
Output: $\mathcal{C}$ new clustering of records in R.
repeat
    //E-step
    for each record $r \in \mathbf{R}$ do
        for each cluster $C \in \mathcal{C}$ do
            compute $sim(r, C) = cons(r, C) * cont(r, C)$;
        end for
    end for
    //M-step
    Choose the possible world with the highest probability as in Ln.28-35 of LATE;
until $\mathcal{C}$ is not changed
return $\mathcal{C}$;
```

- 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.

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 .5 and the Soundex similarity is 1, we set similarity 1.
- address: We used TF/IDF metric, where token similarity is measured by Jaro-Winker distance with threshold .9. If the TF/IDF similarity is above .5 and the Soundex similarity is 1, we set similarity 1.
- co-author: We used a variant of Jaccard metric (see Eq.(12)), where name similarity is measured by Levenshtein distance and $\theta_h = .8$. We apply this similarity only when the record similarity w.r.t single-valued attributes is above $\theta_s = .5$.

F.2 Additional experimental results

We compared several implementation details and reported efficiency on the partial Patent data. We used the default setting unless mentioned otherwise and reported results for ADJUST.

Decay learning methods: We learned the decay in three ways: DETERMINISTIC learns the decay as described in Section 3; SINGLECOUNT learns single-count decay; and PROBABILISTIC learns the probabilistic decay (Appendix A). We observed that (1) these three methods learn similar curves, and (2) applying the three different curves lead to very similar results (Figure 11).

Edge deletion strategies: We tried various edge-deletion strategies for late binding: NODELETE keeps all edges; THRESHOLDING keeps edges with similarity over .8; TOPK keeps only the top-k edges with similarity over .8; GAP keeps the top edges with weights above .8 and gap within .1 (Appendix D). Figure 12 shows that (1) NODELETE keeps all edges, which often have low weights after normalization, and can thus split many clusters and obtain a very

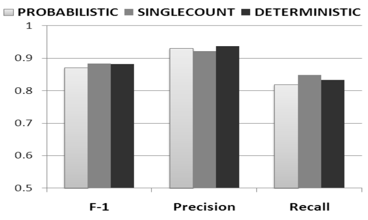


Figure 11: Comparing different decay learning methods.

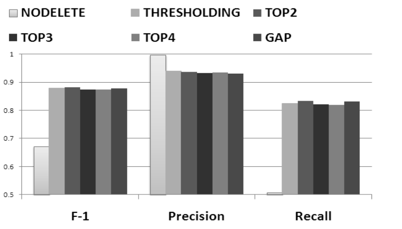


Figure 12: Comparing different edge deletion strategies.

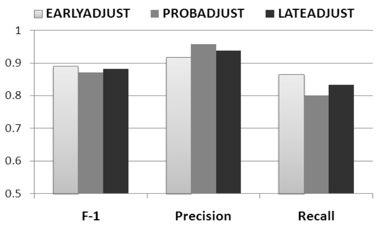


Figure 13: Comparing different adjusted binding methods.

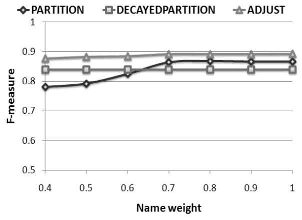


Figure 14: Results of varying attribute weights.

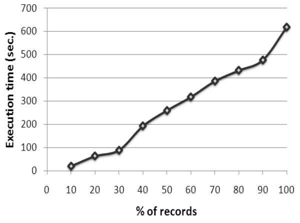


Figure 15: Scalability of ADJUST.

low recall; and (2) different edge-deletion strategies lead to very similar F-measures and improve both efficiency and result quality over NODELETE.

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 13 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, LATEADJUST improves over EARLYADJUST by 26% on the DBLP *WW* data set.

Different parameters: 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 $.4\%$) when $\theta_h \in [.7, .9]$ and $\theta_l \in [.5, .7]$. Second, we applied different attribute weights ($w_{\mathsf{name}} \in [0.4, 1], w_{\mathsf{aff}} = 1 - w_{\mathsf{name}}$) to compute no-decayed similarity. Figure 14 shows that (1) ADJUST is robust against attribute weights; and (2) ADJUST always outperforms PARTITION.

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 Figure 15. We observe that (1) ADJUST terminated in 10.3 minutes on all 1871 records and is reasonably fast given that this is an off-line 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 preprocessing), showing scalability of our techniques.

G. COMPARING WITH EXISTING RECORD LINKAGE TECHNIQUES

Record linkage has been extensively studied in recent years [7, 12]. 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 [5] apply distance metrics to compute similarity of each attribute, and take the weighted sum as the record similarity; *rule-based* approaches [11] 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 training data [6, 19, 8]. 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 learn parameters from training data as well, 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 [1, 2, 14]. 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 [10]. 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 can of record pairs (*e.g.*, *Partition algorithm* [11]), may iteratively specify seeds of clusters and assign vertexes to the seeds (*e.g.*, *Ricochet algorithm* [18]), and may perform clustering by solving an optimization problem (*e.g.*, *Cut clustering* [9]). 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.