# Linking Records with Value Diversity

Pei Li

Department of Informatics and Communications

University of Milan - Bicocca

A thesis submitted for the degree of

*Doctor of Philosophy*

Yet to be decided

I would like to dedicate this thesis to my loving parents.

# Acknowledgements

And I would like to thank all colleagues in the lab and My Ph.D advisors whose long-term supervision and fruitful discussions on the topics covered in this dissertation. Special mention should be given to my external supervisor at AT&T Labs - Research, Xin Luna Dong, who taught me everything about research from scratch and also is a role-model, a life-time friend.

# Abstract

Most record linkage techniques assume that information of the underlying entities do not change and is provided in different representations and sometimes with errors. For example, mailing lists may contain multiple entries representing the same physical address, but each record may be slightly different, e.g., containing different spellings or missing some information. As a second example, consider a company that has different customer databases (e.g., one for each subsidiary). A given customer may appear in different ways in each database, and there is a fair amount of guesswork in determining which customers match.

However, in real-world, we often observe value diversity in real-world data sets for linkage. For example, many data sets contains *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. Value diversity also exists *group linkage*: linking records that refer to entities in the same group. Applications for group linkage includes finding businesses in the same chain, finding conference attendants from the same affiliation, finding players from the same team, etc. In such cases, although different members in the same group can share some similar *global* values, they represent different entities so can also have distinct *local* values, requiring a high *tolerance* for value diversity. However, most existing record linkage techniques assume that records describing the same real-world entities are fairly consistent and often focus on different representations of the same value, such as ”IBM” and ”International Business Machines”. Thus, they can fall short

when values may vary for the same entity. This dissertation studies *how to improve linkage quality of integrated data with tolerance to fairly high diversity, including temporal linkage, and group linkage*.

We solve the problem of temporal record linkage in two ways. 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.

For group linkage, we present a two-stage algorithm: the first stage identifies *cores* containing records that are very likely to belong to the same group; the second stage collects strong evidence from the cores and leverages it for merging more records in the same group, while being tolerant to differences in other values. Our algorithm is designed to ensure efficiency and scalability. An experiment shows that it finished in 2.4 hours on a real-world data set containing 6.8 million records, and obtained both a precision and a recall of above .95.

Finally, we build the CHRONOS system which offers users the useful tool for finding real-world entities over time and understanding history of entities in the bibliography domain. The core of CHRONOS is a temporal record-linkage algorithm, which is tolerant to value evolution over time. Our algorithm can obtain an F-measure of over 0.9 in linking author records and fix errors made by *DBLP*. We show how CHRONOS allows users to explore the history of authors, and how it helps users understand our linkage results by comparing our results with those of existing systems, highlighting differences in the results, explaining our decisions to users, and answering "what-if" questions.

# Contents

# List of Figures

# Chapter 1

# Introduction

*Record linkage*, which 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 Elmagarmid et al. [2007]; Koudas et al. [2006] fro recent surveys). We often observe value diversity in real-world data sets for linkage. For example, many data sets contains *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. Value diversity also exists *group linkage*: linking records that refer to entities in the same group. Applications for group linkage includes finding businesses in the same chain, finding conference attendants from the same affiliation, finding players from the same team, etc. In such cases, although different members in the same group can share some similar *global* values, they represent different entities so can also have distinct *local* values, requiring a high *tolerance* for value diversity. However, most existing record linkage techniques assume that records describing the same real-world entities are fairly consistent and often focus on different representations of the same value, such as ”IBM” and ”International Business Machines”. Thus, they can fall short when values may vary for the same entity. This dissertation studies *how to improve linkage quality of integrated data with tolerance to fairly high diversity, including temporal linkage, and group linkage*.

## 1.1 Existing Record Linkage Techniques

We begin this chapter by reviewing the state of the art for record linkage techniques. Typical record linkage solutions are based on independent pairwise comparisons employing two similarity functions. A similarity function computes attribute-level similarities by comparing values in the same attributes of two records (a typical example is the edit distance). The second similarity function combines the attribute level similarity measures to compute an overall similarity of two records. Groundwork of record linkage problems was done by Fellegi and Sunter Fellegi and Sunter [1969b], where a rigorous statistical framework is introduced that can estimate probabilities for a possible assignment decision procedure, given certain attribute similarity functions. Algorithms investigated in the following texts are considered as relatively more sophisticated solutions beyond independent pairwise comparisons.

**Fuzzy deduplication**: Most domain-independent approaches for record linkage rely on textual similarity functions (*e.g.*, edit distance or cosine metric), predicting that two tuples whose textual similarity is greater than a pre-specified similarity threshold are duplicates. However, using these functions to detect duplicates due to equivalence errors requires that the threshold be dropped low enough, resulting in a large number of false positives. For instance, tuple pairs with values ”USSR” and ”United State” are likely to be declared duplicates if we were to detect ”US” and ”United States” as duplicates using textual similarity. To solve such problems, *fuzzy deduplication approach* proposed by Ananthakrishna et al. rely on hierarchies to detect equivalence errors in each relation, and to reduce the number of false positives.

**Iterative record linkage**: Bhattacharya and Getoor propose an iterative record linkage approach that exploits both attribute similarity and the similarity of *linked objects*. They consider cases where the link are among entities of the same type, so that when two records are detected to refer to the same real-world entity, this may in turn allow additional inferences and make the deduplication process iterative.

**Record linkage in complex information spaces**: Instead of focusing on resolving references of the same classes, the work of Dong et al. studies the problem of resolving different references in complex information spaces, where references belong to *multiple related classes*. The author use a prime example of Personal Information

Management (PIM), where the goal is to provide a coherent view of all the information on one's desktop. The proposed algorithm has three principal features: (1) it exploits the associations between references for similarity comparisons; (2) it propagates information between linkage decisions to accumulate positive and negative evidences; (3) it gradually enrich references by merging attribute values.

**Relationship-based graphical approaches**: Beyond attribute similarities, the quality of record linkage can also be improved by exploiting additional *contextual information*. For instance, "D. White" might be used to refer to an author in the context of a particular publication. The publication might also refer to different authors, which can be linked to their affiliated organizations etc., forming chains of relationships among entities. Such knowledge can be exploited alongside attribute-based similarity resulting in improved accuracy of record linkage. Relationship-based graphical approach ( Kalashnikov et al. [2005]) systematically exploits not only features but also relationships among entities for the purpose of disambiguation. It views the database as a graph of entities that are linked to each other view relationships. It first utilizes a feature-based method to identify a set of candidate entities. Graph theoretic techniques are then used to discover and analyze relationships that exist among candidates.

**Web based approach**: World Wide Web (WWW) search engines are commonly used for learning about real-world entities, such as people. In such cases, users search the name of the target entity in search engines to obtain a set of Web pages that contains that name. However, ambiguity in names typically causes the search results to contain Web pages of several different entities. For example, if we want to know about a "George Bush" other than the former U.S. president, many pages about the former president are returned in the search results, which may be problematic. Existing work Yoshida et al. [2010] studies the problem of disambiguation on people's names by considering both *strong evidence*, such as named entities (NEs) (*e.g.*, Bill Gates), Compound key words (CKWs) (*e.g.*, chief software architect) and URLs, and reliable *weak evidence*.

## 1.2 Value Diversity in Record Linkage

Most existing record linkage techniques assume that records describing the same real-world entities are fairly consistent and are provided in different representations and sometimes with errors. For example, mailing lists may contain multiple entries representing the same physical address, but each record may be slightly different, *e.g.*, containing different spellings or missing some information. As a second example, consider a company that has different customer databases (*e.g.*, one for each subsidiary). A given customer may appear in different ways in each database, and there is a fair amount of guesswork in determining which customer match Benjelloun et al. [2009]. However, we do observe value diversity in real-world entities. In this dissertation, we often on two types of value diversity: (1) linking temporal records where we need to be tolerant to value diversity over time, (2) *linking records of the same group* where we need to be tolerant to value diversity within the same group.

### 1.2.1 Value Diversity in Temporal Dimension

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 Weikum et al. [2011].

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

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

Table 1.1: 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 |

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 1.2.1** *Consider records that describe paper authors in Table 1.1; 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 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.

### 1.2.2 Value Diversity within Groups

Group linkage is slightly different from traditional record linkage. One major motivation for our work comes from identifying *business chains*–connected business entities that share a brand name and provide similar products and services (*e.g.*, *Walmart, McDonald's*). With the advent of the Web and mobile devices, we are observing a boom in *local search*; that is, searching local businesses under geographical constraints. Local search engines include *Google Maps, Yahoo! Local, YellowPages, yelp, ezlocal,* etc. The knowledge of business chains can have big economic values to local search engines, as it allows users to search by business chain, allows search engines to render the returned results by chains, allows data collectors to clean and enrich information within the same chain, allows the associated review system to connect reviews on branches of the same chain, and allows sales people to target potential customers. Business listings are rarely associated with specific chains explicitly, so we need to identify the chains. Sharing the same name, phone number, or URL domain name can all serve as evidence of belonging to the same chain. However, only for US businesses there are tens of thousands of chains and we cannot easily develop any rule set that applies to all chains.

We are also motivated by applications where we need to find people from the same organization, such as counting conference attendants from the same affiliation, counting papers by authors from the same institution, and finding players of the same team.

Table 1.2: Identified US Business chains. For each chain, we show the number of stores, distinct business names, distinct phone numbers, distinct URL domain names, and distinct categories.

| Name | #Store | #Name | #Phn | #URL | #Cat |
|---|---|---|---|---|---|
| *USPS-United States Post Office* | 12,776 | | | | |
| *SUBWAY* | 11,278 | | | | |
| *State Farm Insurance* | 8,711 | | | | |
| *McDonald's* | 7,450 | | | | |
| *Edward Jones* | 6,781 | | | | |

The organization information is often missing, incomplete, or simply too heterogeneous to be recognized as the same (*e.g.*, "International Business Machines Corporation", "IBM Corp.", "IBM", "IBM Research Labs", "IBM-Almaden", etc., all refer to the same organization). Contact phones, email addresses, and mailing addresses of people all provide extra evidence for linkage, but they can also vary for different people even in the same organization.

Group linkage faces challenges not present for traditional record linkage. First, although different members in the same group can share some similar *global* values, they represent different entities so can also have distinct *local* values. For example, different branches in the same business chain can provide different local phone numbers, different addresses, etc. It is not easy to distinguish such difference from various representations for the same value and sometimes erroneous values in the data. Second, there are often millions of records for group linkage, and a group can contain tens of thousands of members. Even a simple pairwise record comparison within a group can already be very expensive, so *scalability* is a big challenge. We use the example of identifying business chains throughout the paper for illustration.

**Example 1.2.2** *We consider a set of 18 million real-world business listings in the US*[1]*, each describing a business by its name, phone number, URL domain name, location, and category. Our algorithm automatically finds 80,000 business chains and there are 1M listings that belong to some chain. Table 1.2 lists the largest five chains we found. We observe that (1) each chain contains up to 13K different branch stores, and (2) different branch listings from the same chain can have a large variety of names, phone numbers, and URL domain names.*

[1]We omit name of the data source for double-blind reviewing.

Table 1.3: Real-world business listings. We show only state for location and simplify names of category. There is a wrong value in italic font.

| RID | name | phone | URL (domain) | location | category |
|---|---|---|---|---|---|
| $r_1$ | Home Depot, The | 808 | | NJ | furniture |
| $r_2$ | Home Depot, The | 808 | | NY | furniture |
| $r_3$ | Home Depot, The | 808 | homedepot | MD | furniture |
| $r_4$ | Home Depot, The | 808 | homedepot | AK | furniture |
| $r_5$ | Home Depot, The | 808 | homedepot | MI | furniture |
| $r_6$ | Home Depot, The | 101 | homedepot | IN | furniture |
| $r_7$ | Home Depot, The | 102 | homedepot | NY | furniture |
| $r_8$ | Home Depot, USA | 103 | homedepot | WV | furniture |
| $r_9$ | Home Depot USA | 808 | | SD | furniture |
| $r_{10}$ | Home Depot - Tools | 808 | | FL | furniture |
| $r_{11}$ | Taco Casa | | tacocasa | AL | restaurant |
| $r_{12}$ | Taco Casa | 900 | tacocasa | AL | restaurant |
| $r_{13}$ | Taco Casa | 900 | tacocasa, *tacocasatexas* | AL | restaurant |
| $r_{14}$ | Taco Casa | 900 | | AL | restaurant |
| $r_{15}$ | Taco Casa | 900 | | AL | restaurant |
| $r_{16}$ | Taco Casa | 701 | tacocasatexas | TX | restaurant |
| $r_{17}$ | Taco Casa | 702 | tacocasatexas | TX | restaurant |
| $r_{18}$ | Taco Casa | 703 | tacocasatexas | TX | restaurant |
| $r_{19}$ | Taco Casa | 704 | | NY | food store |
| $r_{20}$ | Taco Casa | | tacodelmar | AK | restaurant |

*Table 1.3 shows a set of 20 business listings in this data set. After investigating their webpages, we find that $r_1 - r_{18}$ belong to three business chains: $Ch_1 = \{r_1 - r_{10}\}$, $Ch_2 = \{r_{11} - r_{15}\}$, and $Ch_3 = \{r_{16} - r_{18}\}$; $r_{19}$ and $r_{20}$ do not belong to any chain. Note the slightly different names for businesses in chain $Ch_1$; also note that $r_{13}$ is integrated from different sources and contains two URLs, one (*tacocasatexas*) being wrong.*

*Simple linkage rules do not work well on this data set. For example, if we require only high similarity on* name *for chain identification, we may wrongly decide that $r_{11} - r_{20}$ all belong to the same chain as they share a popular restaurant name* Taco Casa. *Traditional linkage strategies do not work well either. If we apply* Swoosh*-style linkage Benjelloun et al. [2009] and iteratively merge records with high similarity on* name *and shared* phone *or* URL*, we can wrongly merge $Ch_2$ and $Ch_3$ because of the*

*wrong URL from $r_{13}$. If we require high similarity between listings on* name, phone, URL, category*, we may either split $r_6 - r_8$ out of chain $Ch_1$ because of their different local phone numbers, or learn a low weight for* phone *but split $r_9 - r_{10}$ out of chain $Ch_1$ since sharing the same phone number, the major evidence, is downweighted.* □

## 1.3 Contributions of the Dissertation

This dissertation makes the following contributions.

**Temporal record linkage**

Chapter 2 studies linking temporal records, where we need to be tolerant to value diversity over time, and makes three contributions. First, we apply *time decay*, which aims to capture the effect of time elapse on entity value evolution. 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. 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.

**Group linkage**

Chapter 3 studies the problem of group linkage. Group linkage is different from any existing record linkage problem, where the goal is to link records that refer to the

same real-world entity. Group linkage, on the other hand, is essentially to link records that refer to entities in the same group.

The key idea in our solution is to find strong evidence that can glue group members together, while being tolerant to differences in values specific for individual group member. For example, we wish to reward sharing of primary values, such as *primary phone numbers* or *URL domain names* for chain identification, but would not penalize differences from local values, such as *locations*, local phone numbers, and even categories. For this purpose, our algorithm proceeds in two stages. First, we identify *cores* containing records that are very likely to belong to the same group. Second, we collect strong evidence from the resulting cores, such as primary phone numbers and URL domain names in business chains, based on which we cluster the cores and remaining records into groups. The use of cores and strong evidence distinguishes our clustering algorithm from traditional clustering techniques for record linkage. In this process, it is crucial that core generation makes few false positives even in presence of erroneous values, such that we can avoid ripple effect on clustering later. Our algorithm is designed to ensure efficiency and scalability.

To the best of our knowledge, we are the first one studying the group-linkage problem. We make three contributions. First, we study core generation in presence of erroneous data. Our core is robust in the sense that even if we remove a few possibly erroneous records, we still have strong evidence that the rest of the records must belong to the same group. We propose efficient algorithm for core generation. Second, we reduce the group linkage problem into clustering cores and remaining records. Our clustering algorithm leverages strong evidence collected from cores and meanwhile is tolerant to value variety of records in the same group. Finally, we conducted experiments on two real-world data sets in different domains, showing high efficiency and effectiveness of our algorithms.

## 1.4 An Application: Facilitating History Discovery by Linking Temporal Records

Whereas our technical contributions apply to record linkage applications in general, we have grounded them into a particular system, CHRONOS. To further motivate the tech-

nical problems we solve in the dissertation, we next describe the goals of CHRONOS and the challenges we face to achieve these goals. We describe CHRONOS system in details in Chaper 4.

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. From such data, users often wish to search for entities in a particular period, and understand the history of one entity or all entities in the data set. For example, *DBLP*[1] lists research papers over many decades; *DBLP* users may wish to find authors by name and year, find the publication history and affiliation history of an author, find the number of her co-authors in each year over time, find her research topics over time, and so on.

A major challenge for enabling such search and exploration is to identify records that describe the same real-world entity over a long period of time; only with such an integrated view, we will be able to trace the history of that entity and collect statistics over time. However, linking temporal records is by no means easy. First, we need to be able to link together records for the same real-world entity but at different times. This is hard because entities can evolve over time; for example, a researcher can move from one affiliation to another, change her research topic, and collaborate with different co-authors over time. Thus, records that describe the same real-world entity at different times can contain different values; blindly requiring value consistency of the linked records may cause false negatives. Second, we need to be able to distinguish records that share common attribute values but refer to different real-world entities. This is especially hard for temporal records because it is more likely to find highly similar entities over a long time period than at the same time; for example, having two persons with highly similar names in the same university over the past 30 years is more likely than at the same time. Thus, records that describe different entities at different times can share common values; blindly matching records that have similar attribute values can cause false positives.

We build the CHRONOS system[2], which offers users a useful tool for finding real-world entities over time and understanding history of entities in the bibliography do-

[1] http://www.dblp.org/.

[2] *Chronos* is a Greek God for *time*; he has three heads, a man, a bull, and a lion, showing the importance of "linkage".

main. The core of CHRONOS is a temporal record-linkage algorithm, which is tolerant to value evolution over time Li et al. [2011]. Our algorithm can obtain an F-measure of over 0.9 in linking author records and can fix errors made by *DBLP*. There are two key ideas for the linkage techniques: first, we apply *time decay* that captures the effect of time elapse on entity value evolution; second, we apply *temporal clustering* that considers records in time order and accumulates evidence over time to enable decision making with a global view.

## 1.5 Outline

The following chapters - Chapter 2 and Chapter 3 describe temporal linkage and group linkage. They elaborate on the ideas outlined in Section 1.3. Chapter 4 describes how we incorporate these technical solutions in the CHRONOS system. Finally, Chapter 5 concludes the dissertation and discusses directions for future research.

Parts of this dissertation have been published or under submission in conferences. In particular, the temporal linkage algorithm (Chapter 2) is described in Li et al. [2011], the group linkage algorithm is under submission for SIGMOD 2013. Finally, the CHRONOS system is described in Li et al. [2011] and demonstrated in Li et al. [2012].

# Chapter 2

# Linking Temporal Records

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 case, 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 technique ignore the temporal information and can fall short for temporal data.

This chapter studies linking temporal records. We begin by defining the temporal record linkage problem and giving an overview of our approach. We introduce the concept of agreement/disagreement decay and propose a statistic model of obtaining decay weights from sample data in Section 2.2. Section 2.3 describes three temporal record linkage techniques - *Early Binding* (Section 2.3.1), *Late Binding* (Section 2.3.2) and *Adjusted Binding* (Section 2.3.3). Among them, given decayed record/object similarities, greedy linkage decisions are made locally in *Early Binding*. Instead of making local binary decisions, a more conservative, probabilistic decision-making model is proposed in *Late Binding*. And in *Adjusted Binding*, to better leverage later evidence, we allow symmetric record/object comparisons from a global point of view. We experimented the aforementioned decay weight over existing record linkage techniques and our proposed techniques, on two real-world data sets to show the effectiveness and efficiency of temporal record linkage in Section 2.4. Finally, Section 2.6 summarizes this chapter.

## 2.1 Problem Definition and Overview of Our Approach

This section formally defines the temporal record linkage problem (Section 2.1) and provides an overview of our approach (Section 2.2).

### 2.1.1 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 2.1.1 (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 2.1.2** *Consider the records in Table 1.1, where each record describes an author by* 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:* $\{r_1\}, \{r_2, \ldots, r_6\}, \{r_7, \ldots, r_{12}\}$.

### 2.1.2 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 1.2.2). 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 2.2), 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 1.2.2, 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 2.3) that consider time order of records and can further improve linkage results.

## 2.2 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.

### 2.2.1 Definition

As time goes by, the value of an entity may evolve; for example, entity $E_2$ in Example 1.2.2 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.

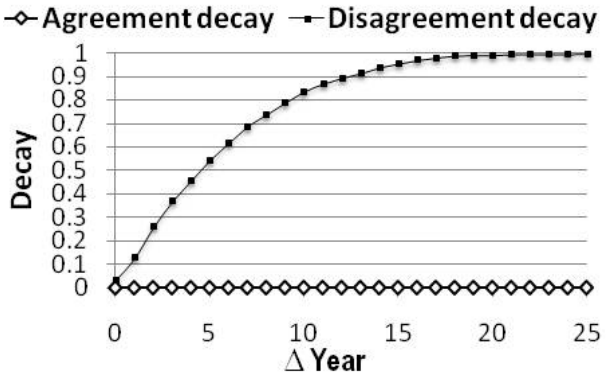


Figure 2.1: Decay curves for Address

**Definition 2.2.1 (Disagreement decay)** *Let $\Delta t$ be a time distance and $A \in \mathbf{A}$ be a single-valued attribute.* Disagreement decay *of $A$ over time $\Delta t$, denoted by $d^{\neq}(A, \Delta t)$, is the probability that an entity changes its $A$-value within time $\Delta t$.* □

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 1.2.2 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 2.2.2 (Agreement decay)** *Let $\Delta t$ be a time distance and $A \in \mathbf{A}$ be an attribute. The* agreement decay *of $A$ over time $\Delta t$, denoted by $d^{=}(A, \Delta t)$, is the probability that two different entities share the same $A$-value within time $\Delta t$.* □

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, 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 $A$, $d^{\neq}(A, \Delta t) \leq d^{\neq}(A, \Delta t')$ 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 2.2.3** *Figure 2.1 shows the curves of disagreement decay and agreement decay on attribute* address *learned from a European patent data set (described in detail in Section 2.4).*

*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.*

### 2.2.2 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.[1] 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 $\Delta t$. We next describe three ways to calculate decay for $A$ and $\Delta t$ according to the labels, namely, *deterministic decay*, *single-count decay* and *probabilistic decay*.

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

#### 2.2.2.1 Disagreement decay

By definition, disagreement decay for $\Delta t$ is the probability that an entity changes its $A$-value within time $\Delta t$. 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 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 $\delta$ denotes one time unit (in Example 1.2.2, 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 $\Delta t$ (we cannot know for others if the value will change within $\Delta t$). We compute the decay as

$$d^{\neq}(A, \Delta t) = \frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|}. \tag{2.1}$$

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

**Proposition 2.2.4** *Let $A$ be an attribute. For any $\Delta t < \Delta t'$, the decay learned by Algorithm* LEARNDISAGREEDECAY *satisfies $d^{\neq}(A, \Delta t) \leq d^{\neq}(A, \Delta t')$.* □

**Proof 2.2.5** *In* LEARNDISAGREEDECAY*, as $\Delta t$ increases, $|\{l \in \bar{L}_f | l \leq \Delta t\}|$ is non-decreasing while $|\{l \in \bar{L}_p | l \geq \Delta t\}|$ is non-increasing; thus, $\frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|}$ is non-decreasing.*

**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 $d^{\neq}(\Delta t, A)$.

```
L̄_f = φ; L̄_p = φ;
for each C ∈ C̄ do
    sort records in C in increasing time order to r_1, ..., r_|C|;
    // Find life spans
    start = 1;
    while start ≤ |C| do
        end = start + 1;
        while r_start.A = r_end.A and end ≤ |C| do
            end++;
        end while
        if end > |C| then
            insert r_|C|.t − r_start.t + δ into L̄_p; // partial life span
        else
            insert r_end.t − r_start.t into L̄_f; // full life span
        end if
        start = end;
    end while
end for
// learn decay
for Δt = 1, ..., max_{l∈L̄_f∪L̄_p}{l} do
    d^≠(A, Δt) = |{l∈L̄_f | l≤Δt}| / (|L̄_f| + |{l∈L̄_p | l≥Δt}|)
end for
```

**Example 2.2.6** *Consider learning disagreement decay for* affiliation *from the data in Example 1.2.2. For illustrative purposes, we remove record $r_{10}$ as its affiliation information is incorrect. Take $E_2$ as an example. As shown in Fig. 2.2, 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 $d^{\neq}(\mathsf{aff}, \Delta t \geq 5) = \frac{2}{2} = 1$.*

**Single-count decay:** We consider an entity at most once and learn the disagreement

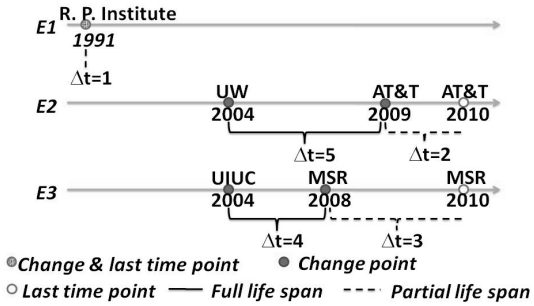


Figure 2.2: Learning $d^{\neq}(\mathsf{aff}, \Delta t)$

decay $d^{\neq}(A, \Delta t)$ as the fraction of entities that have changed their $A$-value within time $\Delta t$. 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.(2.1).

**Proposition 2.2.7** *Let $A$ be an attribute. For any $\Delta t < \Delta t'$, the decay learned using* Single-count decay *satisfies $d^{\neq}(A, \Delta t) \leq d^{\neq}(A, \Delta t')$.* □

**Proof 2.2.8** *In* single-count decay*, as $\Delta t$ increases, $|\{l \in \bar{L}_f | l \leq \Delta t\}|$ is non-decreasing while $|\{l \in \bar{L}_p | l \geq \Delta t\}|$ is non-increasing; thus, $\frac{|\{l \in \bar{L}_f | l \leq \Delta t\}|}{|\bar{L}_f| + |\{l \in \bar{L}_p | l \geq \Delta t\}|}$ is non-decreasing.*

**Example 2.2.9** *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, $d^{\neq}(\mathsf{aff}, \Delta t \in [0, 3]) = \frac{0}{1+1} = 0$, $d^{\neq}(\mathsf{aff}, \Delta t \in [4, 7]) = \frac{1}{1+1} = .5$, $d^{\neq}(\mathsf{aff}, \Delta t \geq 8) = \frac{1}{1+0} = 1$.*

*For comparison, on the same data set* LEARNDISAGREEDECAY *learns the following disagreement decay: $d^{\neq}(\mathsf{aff}, \Delta t \in [0, 3]) = 0$, $d^{\neq}(\mathsf{aff}, \Delta t = 4) = \frac{1}{2+1} = 0.33$, $d^{\neq}(\mathsf{aff}, \Delta t \in [5, 7]) = \frac{2}{2+1} = .67$, $d^{\neq}(\mathsf{aff}, \Delta t \geq 8) = \frac{2}{2+0} = 1$. 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 $t', t \leq t' \leq t_{next}$. 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

$$d^{\neq}(A, \Delta t) = \frac{\sum_{l \in \bar{L}_f, l \leq \Delta t} p(l)}{\sum_{l \in \bar{L}_f} p(l) + |\{l \in \bar{L}_p | l \geq \Delta t\}|}. \tag{2.2}$$

**Proposition 2.2.10** *Let $A$ be an attribute. For any $\Delta t < \Delta t'$, the decay learned using* Probabilistic decay *satisfies $d^{\neq}(A, \Delta t) \leq d^{\neq}(A, \Delta t')$.* □

**Proof 2.2.11** *In* probabilistic decay*, as $\Delta t$ increases, $\sum_{l \in \bar{L}_f, l \leq \Delta t} p(l)$ is non-decreasing while $|\{l \in \bar{L}_p | l \geq \Delta t\}|$ is non-increasing; thus, $\frac{\sum_{l \in \bar{L}_f, l \leq \Delta t} p(l)}{\sum_{l \in \bar{L}_f} p(l) + |\{l \in \bar{L}_p | l \geq \Delta t\}|}$ is non-decreasing.*

**Example 2.2.12** *Consider learning disagreement decay for* affiliation *from the data set in Example 1.2.2. 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, $d^{\neq}(\mathsf{aff}, \Delta t \in [0, 1]) = \frac{0}{2+3} = 0$, $d^{\neq}(\mathsf{aff}, \Delta t = 2) = \frac{0}{2+2} = 0$, $d^{\neq}(\mathsf{aff}, \Delta t = 3) = \frac{0.5+0.33}{2+1} = 0.28$, $d^{\neq}(\mathsf{aff}, \Delta t = 4) = \frac{0.33+0.33+0.5+0.5}{2} = 0.84$, and $d^{\neq}(\mathsf{aff}, \Delta t \geq 5) = \frac{0.33+0.33+0.33+0.5+0.5}{2} = 1$.*

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

Experimental results (Section 2.4) show that these three methods learn similar (but more or less smooth) decay curves, and their results lead to similar linkage results.

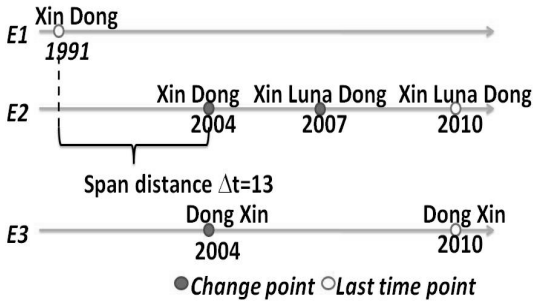


Figure 2.3: Learning $d^{=}(\mathsf{name}, \Delta t)$

#### 2.2.2.2 Agreement decay

By definition, agreement decay for $\Delta t$ is the probability that two different entities share the same value within time period $\Delta t$. 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 $\Delta t$. We call $\max\{0, t_3 - t_2 + \delta\}$ the *span distance* for $v$ between $E_1$ and $E_2$.[1]

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 $\infty$ as the span distance between them. We denote by $\bar{L}$ the bag of span distances and compute the agreement decay as

$$d^{=}(A, \Delta t) = \frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}. \tag{2.3}$$

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

**Proposition 2.2.13** *Let $A$ be an attribute. For any $\Delta t < \Delta t'$, the decay learned by Algorithm* LEARNAGREEDECAY *satisfies $d^{=}(A, \Delta t) \leq d^{=}(A, \Delta t')$.* □

**Proof 2.2.14** *In* LEARNAGREEDECAY, *as $\Delta t$ increases, $|\{l \in \bar{L} | l \leq \Delta t\}|$ is non-decreasing; thus, $\frac{|\{l \in \bar{L} | l \leq \Delta t\}|}{|\bar{L}|}$ is non-decreasing.*

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

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

**Input:** $\bar{C}$ Clusters of records in the sample data set.
$A$ An attribute for decay learning.
**Output:** Agreement decay $d^{=}(\Delta t, A)$.

```
//Find life spans
for each C ∈ C̄ do
  sort records in C in increasing time order to r_1, ..., r_|C|;
  start = 1;
  while start ≤ |C| do
    end = start + 1;
    while r_start.A = r_end.A and end ≤ |C| do
      end++;
    end while
    if end > |C| then
      r_start.t_next = r_|C|-1.t + δ; // 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
L̄ = φ
for each C, C' ∈ C̄ do
  same = false;
  for each r ∈ C s.t. r.t_next ≠ null do
    for each r' ∈ C' s.t. r.t_next ≠ null do
      if r.A = r'.A then
        same = true;
        if r.t ≤ r'.t then
          insert max{0, r'.t − r.t_next + 1} into L̄;
        else
          insert max{0, r.t − r'.t_next + 1} into L̄;
        end if
      end if
    end for
  end for
  if !same then
    insert ∞ into L̄;
  end if
end for
for Δt = 1, ..., max_{l∈L̄}{l} do
  d^=(A, Δt) = |{l∈L̄ | l≤Δt}| / |L̄|
end for
```

**Example 2.2.15** *Consider learning agreement decay for* name *from data in Example 1.2.2. As shown in Fig. 2.3, 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,* $d^{=}(\mathsf{name}, \Delta t \in [0, 12]) = \frac{0}{3} = 0$*, and* $d^{=}(\mathsf{name}, \Delta t \geq 13) = \frac{1}{3} = 0.33$*.*

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

### 2.2.3 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.

#### 2.2.3.1 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 $\Delta t$. Given records $r$ and $r'$, we compute their similarity as

$$sim(r, r') = \frac{\sum_{A \in \mathbf{A}} w_A(s(r.A, r'.A), |r.t - r'.t|) \cdot s(r.A, r'.A)}{\sum_{A \in \mathbf{A}} w_A(s(r.A, r'.A), |r.t - r'.t|)}. \tag{2.4}$$

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 $\theta_h$ and $\theta_l$ 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 2.2.16** *Consider records $r_2$ and $r_5$ in Example 1.2.2 and we focus on single-valued attributes* name *and* affiliation. *Assume the name similarity between $r_2$ and $r_5$ is 0.9 and the affiliation similarity is 0. Suppose $d^{=}(\mathsf{name}, \Delta t = 5) = 0.05$, $d^{\neq}(\mathsf{aff}, \Delta t = 5) = 0.9$, and $\theta_h = 0.8$. Then, the weight for* name *is $1 - 0.05 = 0.95$ and that for* affiliation *is $1 - 0.9 = 0.1$. So the similarity is $sim(r_1, r_2) = \frac{0.95*0.9+0.1*0}{0.95+0.1} = 0.81$. Note that if we do not apply decay and assign the same weight to each attribute, the similarity would become $\frac{0.5*0.9+0.5*0}{0.5+0.5} = 0.45$.*

*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 $r_1$.*

#### 2.2.3.2 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.

$$s(r.A, r'.A) = \frac{\sum_{v \in r.A, v' \in r'.A, s(v,v') > \theta_h} s(v, v')}{\min\{|r.A|, |r'.A|\}}. \tag{2.5}$$

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,

$$sim'(r, r') = min\{1, sim(r, r') + \sum_{multi-valued\ A} w_A \cdot s(r.A, r'.A)\}. \quad (2.6)$$

On the other hand, if the relationship between the entities and the $A$-values are many-to-many, we apply Eq.(2.6) only when $sim(r, r') > \theta_s$, where $\theta_s$ 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.(2.5) as follows.

$$s(r.A, r'.A) = \frac{\sum_{v \in r.A, v' \in r'.A, s(v,v') > \theta_h} (1 - d^{=}(A, |r.t - r'.t|)) s(v, v')}{\min\{|r.A|, |r'.A|\}}. \quad (2.7)$$

**Example 2.2.17** *Consider records $r_2$ and $r_5$ and multi-valued attribute* co-authors *(many-to-many relationship) in Example 1.2.2. 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 2.2.16 that $sim(r_2, r_5) = 0.81 > \theta_h$; therefore, the overall similarity is $sim'(r_2, r_5) = \min\{1, 0.81 + 0.475 * 0.3\} = 0.95$.*

## 2.3 Temporal Clustering

As shown in Example 2.2.16, 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.

### 2.3.1 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 $\theta$ 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 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

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

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

```
for each record r ∈ R do
  for each C ∈ C̄ do
    compute record-cluster similarity sim(r, C);
  end for
  if max_{C∈C̄} sim(r, C) ≥ θ then
    C = Argmax_{C∈C̄} sim(r, C);
    insert r into C;
    update signature of C;
  else
    insert cluster {r} into C̄;
  end if
end for
return C̄;
```

$C.A$. 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.

**Example 2.3.1** *Consider applying early binding to records in Table 1.1. Table 2.1 shows the signature of* affiliation *for each cluster after we process each record. The change in each step is in bold.*

*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 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*

Table 2.1: Example 2.3.1: 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** |

*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.*

### 2.3.2 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 Fig. 2.4 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.

#### 2.3.2.1 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 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 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 2.4) 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 $\theta$.
- *Top-K* keeps the top-$k$ edges whose associated similarity scores are above threshold $\theta$.
- *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 $\theta_{gap}$, or (2) the score for $e_{i+1}$ is less than threshold $\theta$.

**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

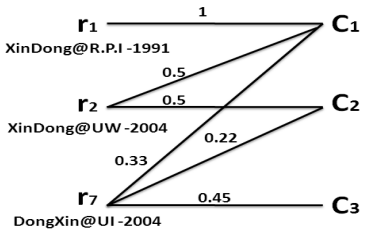


Figure 2.4: Example 2.3.2: A part of the bi-partite graph

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

$$sim(r, C) = \sum_{i=1}^{m} sim(r, r_i)P(r_i)\Pi_{j=i+1}^{m}(1 - P(r_i)). \tag{2.8}$$

**Example 2.3.2** *Consider applying late binding to the records in Table 1.1 and let* $\theta = 0.8$. *Fig. 2.4 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 (*$0.89 > \theta$*) 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,* $sim(r_7, C_1) = 0.8 * 0.5 + 0.4 * 0.5 = 0.6 < \theta$. *For* $C_2$*, we shall compare* $r_7$ *only with* $r_2$ *and the similarity is* $0.4 < \theta$. *Because of the low sim-*

*ilarities, 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.*

#### 2.3.2.2 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.

**Proposition 2.3.3** LATE *algorithm runs in time $O(|\mathbf{R}|^2)$ and chooses the clustering with the highest probability from all possible valid clusterings.* □

**Proof 2.3.4** *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.*

**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 = ∅;
//Evidence collection
for each record r ∈ R do
  insert node n_r into N_R;
  for each n_C ∈ N_C do
    compute decayed record-cluster similarity sim(r, C);
  end for
  if max_{n_C ∈ N_C} sim(r, C) ≤ θ 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 ∈ N_C, where sim(r, C) > θ do
      compute no decayed similarity sim'(r, C);
      if sim'(r, C) > θ 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 ∈ N_C, sim'(r,C)>θ}(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' ≠ C;
  if cluster C is created for record r' ≠ r then
    select edge (n_{r'}, n_C);
    remove edges (n_{r'}, n_{C'}) for all C' ≠ C;
  end if
end while
return C̄;
```

Table 2.2: Example 2.3.5: 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 | **0.4** | 0.25 | **0.4** | 0.16 | 0.12 | 0.17 | **0.27** | 0.1 | **0.24** |
| $C_3$ | 0 | 0 | **0.45** | 0.23 | **0.48** | 0.22 | 0.24 | 0.26 | 0.2 | 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 | **0.3** | 0.16 | 0.16 | 0.22 | 0.18 |

**Example 2.3.5** *Continuing from Example 2.3.2. After evidence collection, we created 5 clusters and the weight of each record-cluster pair is shown in Table 2.2. 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 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 5 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.*

### 2.3.3 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 1.2.2, 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.

#### 2.3.3.1 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 $r_k.t < r.t < r_{k+1}.t$.[1] 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 $cons(r, C)$:

$$cons(r, C) = \min(sim(r, C_1), sim(r_{k+1}, C_2)). \tag{2.9}$$

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

$$freq(C) = \frac{C_{late} - C_{early}}{|C|}. \tag{2.10}$$

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

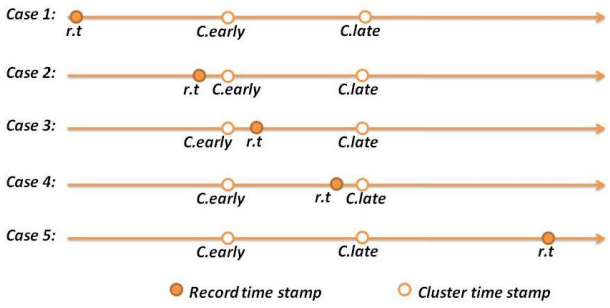


Figure 2.5: Continuity between record $r$ and cluster $C$

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

$$cons_f(r, C) = 1 - \frac{|freq(C) - freq(C')|}{\max\{freq(C), freq(C')\}}. \tag{2.11}$$

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. 2.5 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.

$$cont(r, C) = e^{-\lambda y}; \tag{2.12}$$

$$y = \frac{|r.t - C.early| + \alpha}{C.late - C.early + \alpha}. \tag{2.13}$$

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[1]. 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. 2.5, $cont(r, C)$ is close to 0 in cases 1, 5, close to 1 in

[1]In practice, we set $\alpha$ 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 $\lambda$ 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 $cons(r_1, C_2) > cons$.

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

$$sim(r, C) = cons(r, C) \cdot cont(r, C). \tag{2.14}$$

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.

**Example 2.3.6** *Consider $r_{10}$ and $C_4 = \{r_9, r_{11}, r_{12}\}$ in the results of Example 2.3.5. For value consistency, inserting $r_{10}$ into $C_4$ results in $\{r_9, \ldots, r_{12}\}$. Assume $sim(r_{10}, \{r_9\}) = sim(r_{11}, \{r_9,$
$r_{10}\}) = .6$. 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.*

#### 2.3.3.2 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 simi-

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

**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 $\mathbf{R}$.

```
repeat
  //E-step
  for each record r ∈ R do
    for each cluster C ∈ C̄ 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 C̄ is not changed
return C̄;
```

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

$$C_{early} = \sum_{i=1}^{m} r_i.t \cdot (p_i \Pi_{k=1}^{i-1}(1 - p_k)); \tag{2.15}$$

$$C_{late} = \sum_{i=1}^{m} r_i.t \cdot (p_i \Pi_{k=i+1}^{m}(1 - p_k)). \tag{2.16}$$

Our experiments (Section 2.4) shows that probabilistic adjusting takes much longer than deterministic adjusting, but does not have an obvious performance gain.

## 2.4 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.

### 2.4.1 Experiment settings

**Data and golden standard:** We experimented on two real-world data sets: a benchmark of European patent data set[1] 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 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 3.4 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 Fig. 2.1 (Section 2.2); for name, both agreement

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

Table 2.3: 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 |

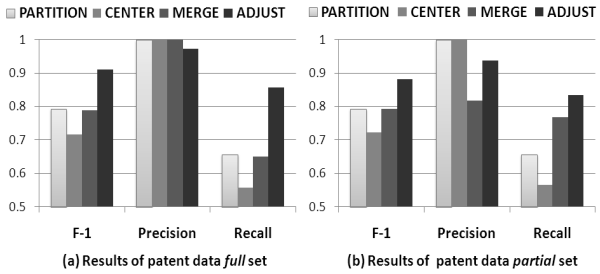


Figure 2.6: Results on the patent data set

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 Hassanzadeh et al. [2009a]. 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.

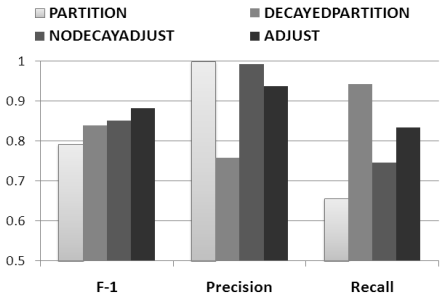


Figure 2.7: Different components on patent *partial* data

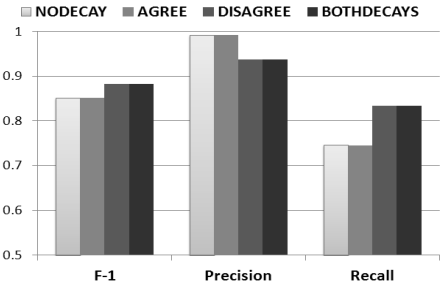


Figure 2.8: Different decays on patent *partial* data

*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

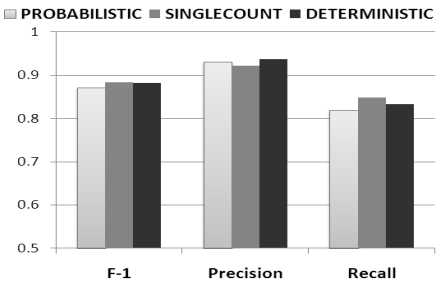


Figure 2.9: Comparing different decay learning methods

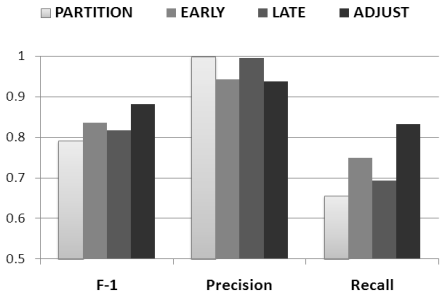


Figure 2.10: Different clustering methods on patent *partial* data

Soundex similarity is 1, we set similarity to 1.

- co-author: We use a variant of Jaccard metric (see Eq.(2.7)), 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 affiliation). 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

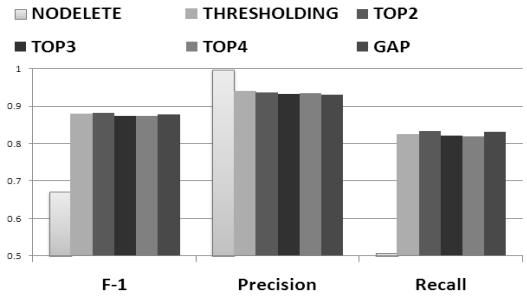


Figure 2.11: Comparing different edge deletion strategies

the set of true positive pairs by $TP$. Then, $P = \frac{|TP|}{|TP|+|FP|}$, $R = \frac{|TP|}{|TP|+|FN|}$, $F = \frac{2PR}{P+R}$.

### 2.4.2 Results on patent data

Fig. 3.6 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.

Figure 3.7 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.

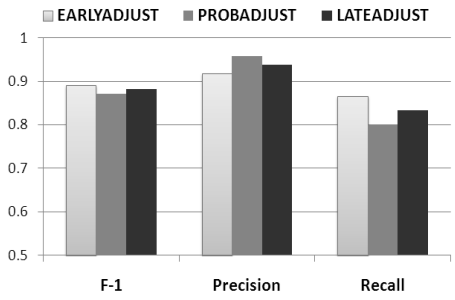


Figure 2.12: Different adjusted binding methods on patent *partial* data

#### 2.4.2.1 Applying decay

**Disagreement vs agreement decay:** Figure 2.8 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 2.2. We observe that (1) these three methods learn similar curves, and (2) as shown in Fig. 2.9, applying the three different curves lead to very similar results for ADJUST, while DETERMINISTIC obtains slightly higher F-measure than the other two.

#### 2.4.2.2 Temporal clustering

**Different clustering methods:** Figure 2.10 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 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;

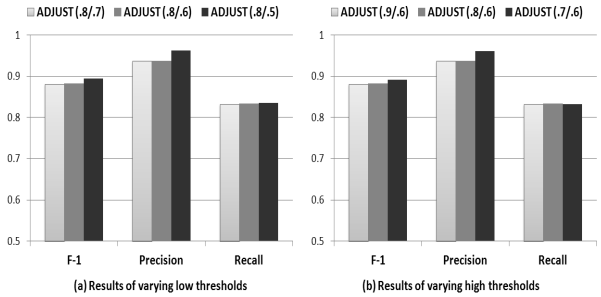


Figure 2.13: Different thresholds on patent *partial* data

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 . Fig. 2.11 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.

**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. Fig. 2.12 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. 2.17(c), LATEADJUST improves over EARLYADJUST by 26% on another data set, the DBLP *WW* data set.

#### 2.4.2.3 Robustness

We ran two experiments to test robustness against parameter settings. We first changed thresholds $\theta_h$ and $\theta_l$ 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. 2.13).

Second, we applied different attribute weights ($w_{\mathsf{name}} \in [0.4, 1], w_{\mathsf{addr.}} = 1 - w_{\mathsf{name}}$) to compute no-decayed similarity. Fig. 2.14 shows that (1) ADJUST is robust against

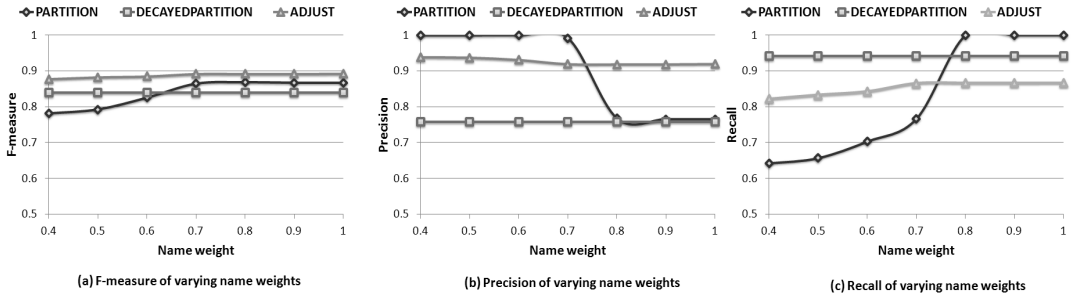


Figure 2.14: Comparison of applying different attribute weights

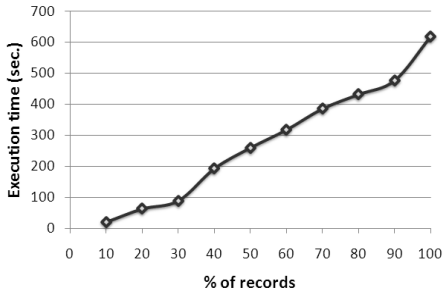


Figure 2.15: Scalability of ADJUST

attribute weights; and (2) ADJUST always outperforms PARTITION in F-measure.

#### 2.4.2.4 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. 2.15. 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 pre-processing), showing scalability of our techniques.

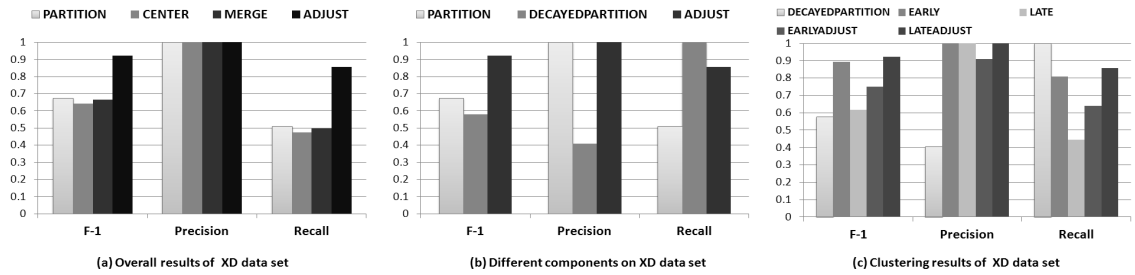


Figure 2.16: Results of XD data set

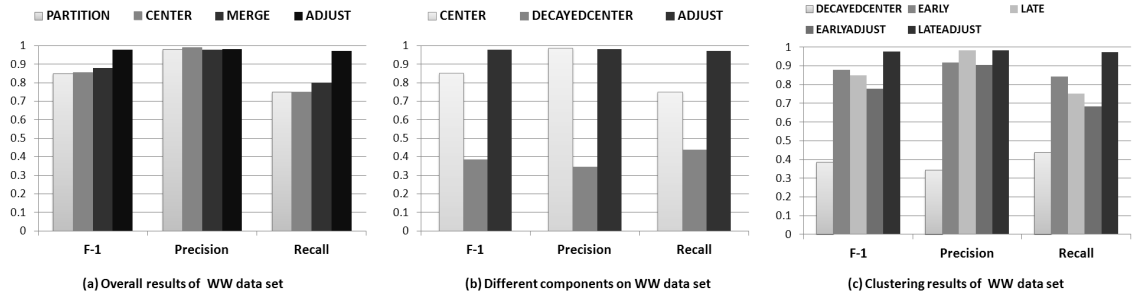


Figure 2.17: Results of WW data set

### 2.4.3 Results on DBLP data

#### 2.4.3.1 *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 each have 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 0.8).

Figure 2.16(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. 2.16(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.

#### 2.4.3.2 *WW* data set

We first report results on the 302 records for which DBLP has identified 18 clusters, of 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 contains 1 record, and 6 clusters contain more than 10 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 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 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 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 4 transitions because there are very few records for one of the affiliations and so not enough evidence for merging. Finally, Fig. 2.17 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 affiliation; and 35 are wrongly merged mainly because of the 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 0.94 and a recall of 0.94.

## 2.5 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 Elmagarmid et al. [2007]; Koudas et al. [2006]. 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 Fellegi and Sunter [1969a], classify a pair of records as *match, unmatch* and *maybe*; *distance-based* approaches Dey [2008], apply distance metrics to compute similarity of each attribute, and take the weighted sum as the record similarity; *rule-based* approaches Hernandez and Stolfo [1998], 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 Domingos [2004]; Fellegi and Sunter [1969a]; Winkler [2002]. 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 Ananthakrishna et al. [2002]; Chen et al. [2005]; On et al. [2007]. 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 Hassanzadeh et al. [2009a]. 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* Hernandez and Stolfo [1998]), may iteratively specify seeds of clusters and assign vertexes to the seeds (*e.g.*, *Ricochet algorithm* Wijaya and Bressan [2009]), and may perform clustering by solving an optimization problem (*e.g.*, *Cut clustering* Flake et al. [2004]). 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 Yakout et al. [2010] and Burdick et al. [2011]. In Yakout et al. [2010] 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. Burdick et al. [2011] 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 Ozsoyoglu and Snodgrass [1995], temporal knowledge discovery paradigms Roddick and Spiliopoulou [2002] and data currency models Fan et al. [2011] 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 Cohen and Strauss [2003]; Cormode et al. [2009]. They use decay to reduce the effect of older tuples on data analysis. Of them, *backward* decay Cohen and Strauss [2003] measures time difference backward from the latest time and *forward* decay Cormode

et al. [2009] 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.

## 2.6 Summary

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 Fan et al. [2011] for linkage.

# Chapter 3

# Linking Records in the Same Group

In this chapter, we study the problem of *group linkage*: linking records that refer to entities in the same group. Applications for group linkage include finding businesses in the same chain, finding conference attendants from the same affiliation, finding players from the same team, etc. Group linkage faces challenges not present for traditional record linkage. First, although different members in the group can share some similar global values, they represent different entities so can also have distinct *local* values, requiring a high *tolerance* for value diversity. Second, there are often millions of records for group linkage, and a group can contain tens of thousands of members, requiring a high *scalability*.

This chapter begins by formally defining the problem of group linkage and introducing the outline of our solution in Section 3.1. Then, we describe a two-stage algorithm in the following sections: the first stage identifies *cores* containing records that are very likely to belong to the same group (Section 3.2); the second stage collects strong evidence from the cores and leverages it for merging more records in the same group, while being tolerant to differences in other values (Section 3.3). Section 3.4 presents experimental results and Section 3.5 discusses related work. Finally Section 3.6 summarizes this chapter.

## 3.1 Problem Definition and Overview of Our Approach

This section formally defines the group linkage problem and provides an overview of our solution.

### 3.1.1 Problem definition

Let $\mathbf{R}$ be a set of records that describe real-world entities by a set of attributes $\mathbf{A}$. For each record $r \in \mathbf{R}$, we denote by $r.A$ its value on attribute $A \in \mathbf{A}$. Because of duplicates and heterogeneity of the data, a real-world entity may be represented by a few records in $\mathbf{R}$, with different representations of the same attribute value and sometimes even erroneous values.

We consider the *group linkage* problem; that is, finding records that represent real-world entities belonging to the same group. As an example application, we wish to find *business chains*–a set of business entities with the same or highly similar names at different locations, either under shared corporate ownership (i.e., sharing a brand and central management, such as *Walmart* and *Home Depot*), or under franchising agreements (i.e., operating with the right or license granted by a company for marketing its products in a specific territory, such as *Subway* and *McDonald's*).[1] We focus on non-overlapping groups, which often hold in applications, and leave overlapping groups for future work.

**Definition 3.1.1 (Group linkage)** *Given a set* $\mathbf{R}$ *of records,* group linkage *identifies a set of clusters* $\mathbf{CH}$ *of records in* $\mathbf{R}$*, such that (1) records that represent real-world entities in the same group belong to one and only one cluster in* $\mathbf{CH}$*, and (2) records that represent real-world entities not in any group do not belong to any cluster in* $\mathbf{CH}$*.* □

We assume we have already applied record-linkage techniques (*e.g.*, Guo et al. [2010]) to merge different records that represent the same entity. Our experiments show that minor mistakes for record linkage typically do not significantly affect the results of group linkage, and records that describe the same entity but fail to be merged are often put into the same group. We leave a better combination of record linkage and group linkage for future work.

[1] http://en.wikipedia.org/wiki/Chain_store.

**Example 3.1.2** *Consider records in Example 1.2.2, where each record describes a business store (at a distinct location) by attributes* name*,* phone*,* URL*,* location*, and* category*.*

*The ideal solution to the group linkage problem contains 3 clusters:* $Ch_1 = \{r_1 - r_{10}\}$*,* $Ch_2 = \{r_{11} - r_{15}\}$*, and* $Ch_3 = \{r_{16} - r_{18}\}$*. Among them,* $Ch_2$ *and* $Ch_3$ *represent two different chains with the same name. Listings* $r_{19}$ *and* $r_{20}$ *do not belong to any chain, so not to any cluster in the solution either.* □

### 3.1.2 Overview of our solution

Group linkage is slightly different from traditional record linkage because it essentially looks for records that represent entities in the same group, rather than records that represent exactly the same entity. Different members in the same group often share a certain amount of commonality (*e.g.*, common name, primary phone and URL domain of chain stores), but meanwhile can also have a lot of differences (*e.g.*, different addresses, local phone numbers of chain stores); thus, we need to allow much higher variety in some attribute values to avoid false negatives. On the other hand, as we have shown in Section 1.2.2, simply lowering our requirement on similarity of records or similarity of a few attributes in clustering can lead to a lot of false positives, which we also wish to avoid.

The key idea of our solution is to distinguish between *strong* evidence and *weak* evidence. For example, different branches in the same business chain often share the same URL domain name and those in North America often share the same 1-800 phone number. Thus, a URL domain or phone number shared among many business listings with highly similar names can serve as strong evidence for chain identification. In contrast, a phone number alone shared by only a couple of business entities is much weaker evidence, since one might be an erroneous or out-of-date value.

To facilitate leveraging strong evidence, our solution contains two stages. The first stage collects records that are highly likely to belong to the same group; for example, a set of business listings with the same name and phone number are very likely to be in the same chain. We call the results *cores* of the groups; from them we can collect strong evidence such as name, primary phone number, and primary URL domain of chains. The key of this stage is to be robust against erroneous values and make as few

false positives as possible, so we can avoid identifying strong evidence wrongly and causing ripple effect later; however, being too strict and missing the cores of many groups can miss important strong evidence.

The second stage clusters cores and remaining records into groups according to the discovered strong evidence. It decides whether several cores belong to the same group, and whether a record that does not belong to any core actually belongs to some group. It also employs weak evidence, but treats it differently from strong evidence. The key of this stage is to leverage the strong evidence and meanwhile be tolerant to diversity of values in the same group, so we can remove false negatives made in the first stage.

We next illustrate our approach for business-chain identification.

**Example 3.1.3** *Continue with the motivating example. In the first stage we generate three cores:* $Cr_1 = \{r_1 - r_7\}$, $Cr_2 = \{r_{14}, r_{15}\}$, $Cr_3 = \{r_{16} - r_{18}\}$. *Records* $r_1 - r_7$ *are in the same core because they have the same name, five of them* ($r_1 - r_5$) *share the same phone number* 808 *and five of them* ($r_3 - r_7$) *share the same URL* homedepot. *Similar for the other two cores. Note that* $r_{13}$ *does not belong to any core, because one of its URLs is the same as that of* $r_{11} - r_{12}$, *and one is the same as that of* $r_{16} - r_{18}$, *but except name, there is no other common information between these two groups of records. To avoid mistakes, we defer the decision on* $r_{13}$. *Indeed, recall that* tacocasatexas *is a wrong value for* $r_{13}$. *For a similar reason, we defer the decision on* $r_{12}$.

*In the second stage, we generate groups - business chains. We merge* $r_8 - r_{10}$ *with core* $Cr_1$, *because they have similar names and share either the primary phone number or the primary URL. We also merge* $r_{11} - r_{13}$ *with core* $Cr_2$, *because (1)* $r_{12} - r_{13}$ *share the primary phone* 900 *with* $Cr_2$, *and (2)* $r_{11}$ *shares the primary URL* tacocasa *with* $r_{12} - r_{15}$. *We do not merge* $Cr_2$ *and* $Cr_3$ *though, because they share neither the primary phone nor the primary URL. We do not merge* $r_{19}$ *or* $r_{20}$ *to any chain, because there is again not much strong evidence. We thus obtain the ideal result.* □

To facilitate this two-stage solution, we find attributes that provide evidence for group identification and classify them into three categories. Since there are typically only a few such attributes, the classification can be performed by domain experts (for wrongly classified attributes, we may learn low weights to partially fix the problem).

- *Common-value attribute:* We call an attribute $A$ a common-value attribute if all entities in the same group have the same or highly similar $A$-values. Such

attributes include business-name for chain identification and organization for organization linkage.

- *Dominant-value attribute:* We call an attribute $A$ a dominant-value attribute if entities in the same group often share one or a few primary $A$-values (but there can also exist other less-common values), and these values are seldom used by entities outside the group. Such attributes include phone and URL-domain for chain identification, and office-address, phone-prefix, and email-server for organization linkage.
- *Multi-value attribute:* We call the rest of the attributes mutli-value attributes as there is often a many-to-many relationship between groups and values of these attributes. Such attributes include category for chain identification.

In the rest of the chapter, we describe core identification in Section 3.2 and group linkage in Section 3.3. Our algorithm requires common-value and dominant-value attributes, which often exist for groups in practice.

## 3.2 Core Identification

The first stage of our solution creates cores consisting of records that are very likely to belong to the same business chain. The key in core identification is to be robust to possible erroneous values. This section starts with presenting the criteria we wish the cores to meet (Section 3.2.1), then describes how we efficiently construct similarity graphs to facilitate core finding (Section 3.2.2), and finally gives the algorithm for core identification (Section 3.2.3).

### 3.2.1 Criteria for a core

At the first stage we wish to make only decisions that are highly likely to be correct; thus, we require that each core contains only highly similar records, and different cores are fairly different and easily distinguishable from each other. In addition, we wish that our results are robust even in presence of a few erroneous values in the data. In the motivating example, $r_1 - r_7$ form a good core, because *808* and *homedepot* are very popular values among these records. In contrast, $r_{13} - r_{18}$ does not form a good

core, because records $r_{14} - r_{15}$ and $r_{16} - r_{18}$ do not share any phone number or URL domain; the only "connector" between them is $r_{13}$, so they can be wrongly merged if $r_{13}$ contains erroneous values. Also, considering $r_{13} - r_{15}$ and $r_{16} - r_{18}$ as two different cores is risky, because (1) it is not very clear whether $r_{13}$ is in the same chain as $r_{14} - r_{15}$ or as $r_{16} - r_{18}$, and (2) these two cores share one URL domain name so are not fully distinguishable.

We capture this intuition with *connectivity of a similarity graph*. We define the *similarity graph* of a set $\mathbf{R}$ of business listings as an undirected graph, where each node represents a listing in $\mathbf{R}$, and an edge indicates high similarity between the connected two listings (we describe in which cases we consider two listings as highly similar later). Figure 3.1 shows the similarity graph for the motivating example.

Each core would correspond to a connected sub-graph of the similarity graph. We wish such a sub-graph to be *robust* such that even if we remove a few nodes the sub-graph is still connected; in other words, even if there are some erroneous records, without them we still have enough evidence showing that the rest of the records must belong to the same chain. The formal definition goes as follows.

**Definition 3.2.1 ($k$-robustness)** *A graph $G$ is* $k$-robust *if after removing arbitrary $k$ nodes and edges to these nodes, $G$ is still connected. A clique or a single node is $k$-robust for any $k$.* □

In Figure 3.1, the subgraph with nodes $r_1 - r_7$ is 2-robust, but that with $r_{13} - r_{18}$ is not 1-robust as removing $r_{13}$ can disconnect it.

According to the definition, we can partition the similarity graph into a set of $k$-robust subgraphs. As we do not wish to split any core unnecessarily, we require the *maximal $k$-robust partitioning*; that is, merging any two subgraphs should not obtain a graph that is also $k$-robust.

**Definition 3.2.2 (Maximal $k$-robust partitioning)** *Let $G$ be a similarity graph. A partitioning of $G$ is a* maximal $k$-robust partitioning *if it satisfies the following properties.*

1. *Each node belongs to one and only one subgraph.*
2. *Each subgraph is $k$-robust.*

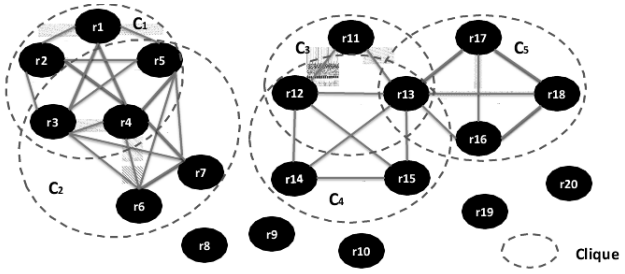


Figure 3.1: Similarity graph for records in Table 1.3.

3. *The result of merging any two subgraphs is not $k$-robust.* □

Note that a data set can have more than one maximal $k$-robust partitioning. Consider $r_{11}-r_{18}$ in Figure 3.1. There are three maximal 1-robust partitionings: $\{\{r_{11}\}, \{r_{12}, r_{14}-r_{15}\}, \{r_{13}, r_{16}-r_{18}\}\}$; $\{\{r_{11}-r_{12}\}, \{r_{14}-r_{15}\}, \{r_{13}, r_{16}-r_{18}\}\}$; and $\{\{r_{11}-r_{15}\}, \{r_{16}-r_{18}\}\}$. If we treat each partitioning as a possible world, records that belong to the same partition in all possible worlds have high probability to belong to the same chain and so form a core. Accordingly, we define a core as follows.

**Definition 3.2.3 (Core)** *Let* $\mathbf{R}$ *be a set of business listings and* $G$ *be the similarity graph of* $\mathbf{R}$*. The records that belong to the same subgraph in every maximal* $k$*-robust partitioning of* $G$ *form a* core *of* $\mathbf{R}$*. A core contains at least 2 records.* □

**Example 3.2.4** *Consider Figure 3.1 and assume* $k = 1$*. There are two connected subgraphs. For records* $r_1 - r_7$*, the subgraph is 1-robust, so they form a core. For records* $r_{11} - r_{18}$*, there are three maximal 1-robust partitionings for their subgraph, as we have shown. Two subsets of records belong to the same subgraph in each partitioning:* $\{r_{14} - r_{15}\}$ *and* $\{r_{16} - r_{18}\}$*; they form 2 cores.* □

### 3.2.2 Constructing similarity graphs

Generating the cores requires analysis on the similarity graph. Real-world data can often contain millions of records; it is unscalable to compare every pair of records and create edges accordingly. We next describe how we construct and represent the similarity graph in a scalable way.

Table 3.1: (a) Inverted index for the similarity graph in Figure 3.1. (b) Simplified inverted index for (a).

| Record | V-Cliques |
|---|---|
| $r_1$ | $C_1$ |
| $r_2$ | $C_1$ |
| $r_3$ | $C_1, C_2$ |
| $r_4$ | $C_1, C_2$ |
| $r_5$ | $C_1, C_2$ |
| $r_6$ | $C_2$ |
| $r_7$ | $C_2$ |
| $r_{11}$ | $C_3$ |
| $r_{12}$ | $C_3, C_4$ |
| $r_{13}$ | $C_3, C_4, C_5$ |
| $r_{14}$ | $C_4$ |
| $r_{15}$ | $C_4$ |
| $r_{16}$ | $C_5$ |
| $r_{17}$ | $C_5$ |
| $r_{18}$ | $C_5$ |

(a)

| Record | V-Cliques | Represent |
|---|---|---|
| $r_{1/2}$ | $C_1$ | $r_1 - r_2$ |
| $r_3$ | $C_1, C_2$ | $r_3$ |
| $r_4$ | $C_1, C_2$ | $r_4$ |
| $r_5$ | $C_1, C_2$ | $r_5$ |
| $r_{6/7}$ | $C_2$ | $r_6 - r_7$ |
| $r_{11}$ | $C_3$ | $r_{11}$ |
| $r_{12}$ | $C_3, C_4$ | $r_{12}$ |
| $r_{13}$ | $C_3, C_4, C_5$ | $r_{13}$ |
| $r_{14/15}$ | $C_4$ | $r_{14} - r_{15}$ |
| $r_{16/17/18}$ | $C_5$ | $r_{16} - r_{18}$ |

(b)

We add an edge between two records if they have the same value for each common-value attribute and share at least one value on a dominant-value attribute.[1] All records that share values on the common-value attributes and share the same value on a dominant-value attribute form a clique, which we call a *v-clique*. We can thus represent the graph with a set of v-cliques, denoted by $\mathbf{C}$; for example, the graph in Figure 3.1 can be represented by 5 v-cliques. In addition, we maintain an *inverted index*, where each entry corresponds to a record $r$ and contains the v-cliques that $r$ belongs to (see Table 3.1(a) as an example). We denote the index by $\bar{L}$ and the v-cliques that record $r$ belongs to by $\bar{L}(r)$. Whereas the size of the similarity graph is quadratic in the number of the nodes, the size of the inverted index is only linear in that number. The inverted index also makes it easy to find *adjacent v-cliques*, v-cliques that share nodes, as they appear in the same entry.

Graph construction is then reduced to v-clique finding, which can be done by scan-

[1] In practice, for common-value attributes we require only highly similar values. We can also adapt our method for other edge-adding strategies; we compare different strategies in experiments (Section 2.4).

ning values of dominant-value attributes. In this process, we wish to prune a v-clique if it is a sub-clique of another one. Pruning by checking every pair of v-cliques can be very expensive since the number of v-cliques is also huge. Instead, we do it together with v-clique finding. Specifically, our algorithm GRAPHCONSTRUCTION takes $\mathbf{R}$ as input and outputs $\mathbf{C}$ and $\bar{L}$. We start with $\mathbf{C} = \bar{L} = \emptyset$. For each value $v$ of a dominant-value attribute, we denote the set of records with $v$ by $\bar{R}_v$ and do the following.

1. Initialize the v-cliques for $v$ as $\mathbf{C_v} = \emptyset$. Add a single-record cluster for each record $r \in \bar{R}_v$ to a working set $\bar{T}$. Mark each cluster as "unchanged".

2. For each $r \in \bar{R}_v$, scan $\bar{L}$ and consider each v-clique $C \in \bar{L}(r)$ that has not been considered yet. For all records in $C \cap \bar{R}_v$, merge their clusters. Mark the merged cluster as "changed" if the result is not a proper sub-clique of $C$. If $C \subseteq \bar{R}_v$, remove $C$ from $\mathbf{C}$. This step removes the v-cliques that must be sub-cliques of those we will form next.

3. For each cluster $C \in \bar{T}$, if there exists $C' \in \mathbf{C_v}$ such that $C$ and $C'$ share the same value for each common-value attribute, remove $C$ and $C'$ from $\bar{T}$ and $\mathbf{C_v}$ respectively, add $C \cup C'$ to $\bar{T}$ and mark it as "changed"; otherwise, move $C$ to $\mathbf{C_v}$. This step merges clusters that share values on common-value attributes. At the end, $\mathbf{C_v}$ contains the v-cliques with value $v$.

4. Add each v-clique with mark "changed" in $\mathbf{C_v}$ to $\mathbf{C}$ and update $\bar{L}$ accordingly. The marking prunes size-1 v-cliques and the sub-cliques of those already in $\mathbf{C}$.

**Proposition 3.2.5** *Let $\mathbf{R}$ be a set of business listings. Denote by $n(r)$ the number of values on dominant-value attributes from $r \in \mathbf{R}$. Let $n = \sum_{r \in \mathbf{R}} n(r)$ and $m = \max_{r \in \mathbf{R}} n(r)$. Let $s$ be the maximum v-clique size. Algorithm* GRAPHCONSTRUCTION *(1) runs in time $O(ns(m+s))$, (2) requires space $O(n)$, and (3) its result is independent of the order in which we consider the records.* □

This proposition (all proofs in Appendix .1) shows that GRAPHCONSTRUCTION reduces both time and space complexity. We observed from experiments that we can save space by 2 orders of magnitude.

**Example 3.2.6** *Consider graph construction for records in Table 1.3. Figure3.1 shows the similarity graph and Table 3.1(a) shows the inverted list. We focus on records* $r_1 - r_8$ *for illustration.*

*First,* $r_1 - r_5$ *share the same name and phone number* 808*, so we add v-clique* $C_1 = \{r_1 - r_5\}$ *to* $\mathbf{C}$*. Now consider URL* homedepot *where* $\bar{R}_v = \{r_3 - r_8\}$*. Step 1 generates 6 clusters, each marked "unchanged", and* $\bar{T} = \{\{r_3\}, \ldots, \{r_8\}\}$*. Step 2 looks up* $\bar{L}$ *for each record in* $\bar{R}_v$*. Among them,* $r_3 - r_5$ *belong to v-clique* $C_1$*, so it merges their clusters and marks the result* $\{r_3 - r_5\}$ *"unchanged" (*$\{r_3 - r_5\} \subset C_1$*); then,* $\bar{T} = \{\{r_3 - r_5\}, \{r_6\}, \{r_7\}, \{r_8\}\}$*. Step 3 compares these clusters and merges the first three as they share the same name, marking the result as "changed". At the end,* $\mathbf{C}_v = \{\{r_3 - r_7\}, \{r_8\}\}$*. Finally, Step 4 adds* $\{r_3 - r_7\}$ *to* $\mathbf{C}$ *and discards* $\{r_8\}$ *since it is marked "unchanged".* □

Given the sheer number of records in $\mathbf{R}$, the inverted index can still be huge. The following proposition shows that records in the same v-clique but not any other v-clique must belong to the same core. As we show later, such records cannot affect robustness judgment, so we do not need to distinguish them.

**Proposition 3.2.7** *Let* $G$ *be a similarity graph. Let* $r$ *and* $r'$ *be two nodes in* $G$ *that belong to the same v-clique* $C$ *and not any other v-clique. Then,* $r$ *and* $r'$ *must belong to the same partition in any maximal* $k$*-robust partitioning.* □

Thus, we simplify the inverted index such that for each v-clique we keep only a *representative* for nodes belonging only to this v-clique. Table 3.1(b) shows the simplified index for Table 3.1(a).

*Case study:* On a data set with 6.8M records (described in Section 2.4), our graph-construction algorithm finished in 2.2 hours. The original similarity graph contains 6.8M nodes and 569M edges. The inverted index contains 0.8M entries, each associated with at most 6 v-cliques; in total there are 94K v-cliques. The simplified inverted index further reduces the size of the index to 0.15M entries, where an entry can represent up to 11K records. Therefore, the simplified inverted index reduces the size of the similarity graph by 3 orders of magnitude.

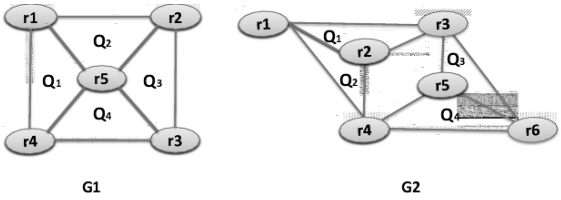


Figure 3.2: Two example graphs.

### 3.2.3 Identifying cores

We solve the core-identification problem by reducing it to a Max-flow/Min-cut Problem. However, computing the max flow for a given graph $G$ and a source-destination pair takes time $O(|G|^{2.5})$, where $|G|$ denotes the number of nodes in $G$; even the simplified inverted index can still contain hundreds of thousands of entries, so it can be very expensive. We thus first merge certain v-cliques according to a sufficient (but not necessary) condition for $k$-robustness and consider them as a whole in core identification; we then split the graph into subgraphs according to a necessary (but not sufficient) condition for $k$-robustness. We apply reduction only on the resulting subgraphs, which are substantially smaller as we show at the end of this section. Section 3.2.3.1 describes screening before reduction, Section 3.2.3.2 describes the reduction, and Section 3.2.3.3 gives the full algorithm, which iteratively applies screening and the reduction. Note that the notations in this section are with respect to v-cliques, thus can be slightly different from those in Graph Theory.

#### 3.2.3.1 Screening

A graph can be considered as a union of v-cliques, so essentially we need to decide if a union of v-cliques is $k$-robust. First, we give the following sufficient condition for $k$-robustness (proof in Appendix 5).

**Theorem 3.2.8 ($(K+1)$-connected condition)** *Let $G$ be a graph consisting of a union $Q$ of v-cliques. If for every pair of v-cliques $C, C' \in Q$, there is a path of v-cliques between $C$ and $C'$ and every pair of adjacent v-cliques on the path share at least $k+1$ nodes, graph $G$ is $k$-robust.* □

We call a single v-clique or a union of v-cliques that satisfy the $(k+1)$-connected condition a $(k+1)$-*connected v-union*. A $(k+1)$-connected v-union must be $k$-robust but not vice versa. In Figure 3.1, subgraph $\{r_1 - r_7\}$ is a 3-connected v-union, because the only two v-cliques, $C_1$ and $C_2$, share 3 nodes. Indeed, it is 2-robust. On the other hand, graph $G_1$ in Figure 3.2 is 2-robust but not 3-connected (there are 4 v-cliques, where each pair of adjacent v-cliques share only 1 or 2 nodes). Accordingly, we can consider a v-union as a whole in core identification.

Next, we present a necessary condition for $k$-robustness (proof in Appendix 5).

**Theorem 3.2.9 ($(K+1)$-overlap condition)** *Graph $G$ is $k$-robust only if for every $(k+1)$-connected v-union $Q \in G$, $Q$ shares at least $k+1$ common nodes with the subgraph consisting of the rest of the v-unions.* □

We call a graph $G$ that satisfies the $(k+1)$-overlap condition a $(k+1)$-*overlap graph*. A $k$-robust graph must be a $(k+1)$-overlap graph but not vice versa. In Figure 3.1, subgraph $\{r_{11} - r_{18}\}$ is not a 2-overlap graph, because there are two 2-connected v-unions, $\{r_{11} - r_{15}\}$ and $\{r_{13}, r_{16} - r_{18}\}$, but they share only one node; indeed, the subgraph is not 1-robust. On the other hand, graph $G_2$ in Figure 3.2 satisfies the 3-overlap condition, as it contains four 3-connected v-unions (actually four v-cliques), $Q_1 - Q_4$, and each v-union shares 3 nodes in total with the others; however, it is not 2-robust (removing $r_3$ and $r_4$ disconnects it). Accordingly, for $(k+1)$-overlap graphs we still need to check $k$-robustness by reduction to a Max-flow Problem.

Now the problem is to find $(k+1)$-overlap subgraphs. Let $G$ be a graph where a $(k+1)$-connected v-union overlaps with the rest of the v-unions on no more than $k$ nodes. We split $G$ by removing these overlapping nodes. For subgraph $\{r_{11} - r_{18}\}$ in Figure 3.1, we remove $r_{13}$ and result with two subgraphs $\{r_{11} - r_{12}, r_{14} - r_{15}\}$ and $\{r_6 - r_8\}$ (recall from Example 3.2.4 that $r_{13}$ cannot belong to any core). Note that the result subgraphs may not be $(k+1)$-overlap graphs (*e.g.*, $\{r_1 - r_2, r_4 - r_5\}$ contains two v-unions that share only one node), so we need to further screen them.

We now describe our screening algorithm, SCREEN, (details in Algorithm 6), which takes a graph $G$, represented by $\mathbf{C}$ and $\bar{L}$, as input, finds $(k+1)$-connected v-unions in $G$ and meanwhile decides if $G$ is a $(k+1)$-overlap graph. If not, it splits $G$ into subgraphs for further examination.

1. If $G$ contains a single node, output it as a core if the node represents multiple records that belong only to one v-clique.
2. For each v-clique $C \in \mathbf{C}$, initialize a v-union. We denote the set of v-unions by $\bar{Q}$, the v-union that $C$ belongs to by $Q(C)$, and the overlap of v-cliques $C$ and $C'$ by $\bar{B}(C, C')$.
3. For each v-clique $C \in \mathbf{C}$, we merge v-unions as follows.

   (a) For each record $r \in C$ that has not been considered, for every pair of v-cliques $C_1$ and $C_2$ in $r$'s index entry, if they belong to different v-unions, add $r$ to overlap $\bar{B}(C_1, C_2)$.

   (b) For each v-union $Q$ where there exist $C_1 \in Q$ and $C_2 \in Q(C)$ such that $|\bar{B}(C_1, C_2)| \geq k + 1$, merge $Q$ and $Q(C)$.

   At the end, $\bar{Q}$ contains all $(k+1)$-connected v-unions.
4. For each v-union $Q \in \bar{Q}$, find its border nodes as $\bar{B}(Q) = \cup_{C \in Q, C' \notin Q} \bar{B}(C, C')$. If $|\bar{B}(Q)| \leq k$, split the subgraph it belongs to, denoted by $G(Q)$, into two subgraphs $Q \setminus \bar{B}(Q)$ and $G(Q) \setminus Q$.
5. Return the remaining subgraphs.

**Proposition 3.2.10** *Denote by $|\bar{L}|$ the number of entries in input $\bar{L}$. Let $m$ be the maximum number of values from dominant-value attributes of a record, and $a$ be the maximum number of adjacent v-unions that a v-union has. Algorithm* SCREEN *takes time $O((m^2 + a) \cdot |\bar{L}|)$ and the result is independent of the order in which we examine the v-cliques.* □

Note that $m$ and $a$ are typically very small, so SCREEN is basically linear in the size of the inverted index. Finally, we have results similar to Proposition .1.3 for v-unions, so we can further simplify the graph by keeping for each v-union a single representative for all nodes that only belong to it. Each result $k$-overlap subgraph is typically very small.

**Example 3.2.11** *Consider Table 3.1(b) as input and $k = 1$. Step 2 creates five v-unions $Q_1 - Q_5$ for the five v-cliques in the input.*

**Algorithm 6** SCREEN($G, \bar{C}, \bar{L}, k$)

**Input:** $G$: Simplified similarity graph.
$\bar{C}$: Set of cores.
$\bar{L}$: Inverted list of the similarity graph.
$k$: Robustness requirement.
**Output:** $\bar{G}$ Set of subgraphs in $G$.

**if** $G$ contains a single node $r$ **then**
  **if** $r$ represent multiple records **then**
    add $r$ to $\bar{C}$.
  **end if**
  return $\bar{G} = \phi$.
**else**
  initialize v-union $Q(C)$ for each v-clique $C$ and add $Q(C)$ to $\bar{Q}$.
  // find v-union
  **for each** v-clique $C \in G$ **do**
    **for each** record $r \in C$ that is not proceeded **do**
      **for each** v-clique pair $C_1, C_2 \in \bar{L}(r)$ **do**
        **if** $C_1, C_2$ are in different v-unions **then**
          add $r$ to overlap $\bar{B}(Q(C_1), Q_(C_2))$.
        **end if**
      **end for**
    **end for**
    **for each** v-union $Q$ where $\bar{B}(Q, Q(C)) \geq k$ **do**
      merge $Q$ and $Q(C)$ as $Q_m$.
      **for each** v-union $Q' \neq Q, Q' \neq Q(C)$ **do**
        set $\bar{B}(Q', Q_m) = \bar{B}(Q', Q) \cup \bar{B}(Q', Q(C))$
      **end for**
    **end for**
  **end for**
  // screening
  **for each** v-union $Q \in \bar{Q}$ **do**
    compute $\bar{B}(Q) = \cup_{Q' \in \bar{Q}} \bar{B}(Q, Q')$.
    **if** $|\bar{B}(Q)| < k$ **then**
      add subgraphs $Q \setminus \bar{B}(Q)$ and $G(Q) \setminus Q$ into $\bar{G}$
    **end if**
  **end for**
**end if**
**return** $\bar{G}$;

*Step 3 starts with v-clique $C_1$. It has 4 nodes (in the simplified inverted index), among which 3 are shared with $C_2$. Thus, $\bar{B}(C_1, C_2) = \{r_3 - r_5\}$ and $|\bar{B}(C_1, C_2)| \geq 2$, so we merge $Q_1$ and $Q_2$ into $Q_{1/2}$. Examining $C_2$ reveals no other shared node.*

*Step 3 then considers v-clique $C_3$. It has three nodes, among which $r_{12} - r_{13}$ are shared with $C_4$ and $r_{13}$ is shared with $C_5$. Thus, $\bar{B}(C_3, C_4) = \{r_{12} - r_{13}\}$ and $\bar{B}(C_3, C_5) = \{r_{13}\}$. We merge $Q_3$ and $Q_4$ into $Q_{3/4}$. Examining $C_4$ and $C_5$ reveals no other shared node. We thus result with three 2-connected v-unions: $\bar{Q} = \{Q_{1/2}, Q_{3/4}, Q_5\}$.*

*Step 4 then considers each v-union. For $Q_{1/2}$, $\bar{B}(Q_{1/2}) = \emptyset$ and we thus split subgraph $Q_{1/2}$ out and merge all of its nodes to one $r_{1/.../7}$. For $Q_{3/4}$, $\bar{B}(Q_{3/4}) = \{r_{13}\}$ so $|\bar{B}(Q_{3/4})| < 2$. We split $Q_{3/4}$ out and obtain $\{r_{11} - r_{12}, r_{14/15}\}$ ($r_{13}$ is excluded). Similar for $Q_5$ and we obtain $\{r_{16/17/18}\}$. Therefore, we return three subgraphs.* □

#### 3.2.3.2 Reduction

Intuitively, a graph $G(V, E)$ is $k$-robust if and only if between any two nodes $a, b \in V$, there are more than $k$ paths that do not share any node except $a$ and $b$. We denote the number of non-overlapping paths between nodes $a$ and $b$ by $\kappa(a, b)$. We can reduce the problem of computing $\kappa(a, b)$ into a Max-flow Problem.

For each input $G(V, E)$ and nodes $a, b$, we construct the (directed) *flow network* $G'(V', E')$ as follows.

1. Node $a$ is the source and $b$ is the sink (there is no particular order between $a$ and $b$).
2. For each $v \in V, v \neq a, v \neq b$, add two nodes $v', v''$ to $V'$, and two directed edges $(v', v''), (v'', v')$ to $E'$. If $v'$ represents $n$ nodes, the edge $(v', v'')$ has weight $n$, and the edge $(v'', v')$ has weight $\infty$.
3. For each edge $(a, v) \in E$, add edge $(a, v')$ to $E'$; for each edge $(u, b) \in E$, add edge $(u'', b)$ to $E'$; for each other edge $(u, v) \in E$, add two edges $(u'', v')$ and $(v'', u')$ to $E'$. Each edge has capacity $\infty$.

**Lemma 3.2.12** *The max flow from source $a$ to sink $b$ in $G'(V', E')$ is equivalent to $\kappa(a, b)$ in $G(V, E)$.* □

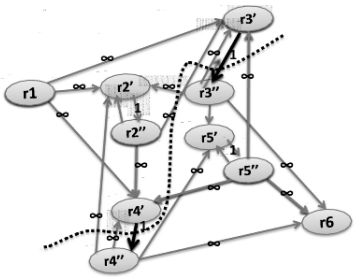

Figure 3.3: Flow network for $G_2$ in Figure 3.2.

**Example 3.2.13** *Consider nodes $r_1$ and $r_6$ of graph $G_2$ in Figure 3.2. Figure 3.3 shows the corresponding flow network, where the dash line (across edges $(r'_3, r''_3), (r'_4, r''_4)$) in the figure cuts the flow from $r_1$ to $r_6$ with a minimum cost of 2. The max flow/min cut has value 2. Indeed, $\kappa(r_1, r_6) = 2$.* □

Recall that in a $(k+1)$-connected v-union, between each pair of nodes there are at least $k+1$ paths. Thus, if $\kappa(a, b) = k+1$, $a$ and $b$ belong to different v-unions, while $a$ and $a'$ belong to the same v-union, we must have $\kappa(a', b) \geq k+1$. We thus have the following sufficient and necessary condition for $k$-robustness (proof in Appendix .3).

**Theorem 3.2.14 (Max-flow condition)** *Let $G(V, E)$ be an input similarity graph. Graph $G$ is $k$-robust if and only if for every pair of adjacent $(k+1)$-connected v-unions $Q$ and $Q'$, there exist two nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$ such that the max flow from $a$ to $b$ in the corresponding flow network is at least $k+1$.* □

If a graph $G$ is not $k$-robust, we shall split it into subgraphs for further core finding. In the corresponding flow network, each edge in the minimum cut must be between a pair of nodes derived from the same node in $G$ (other edges have capacity $\infty$). These nodes cannot belong to any core and we use them as separator nodes, denoted by $\bar{S}$. Suppose the separator separates $G$ into $\bar{X}$ and $\bar{Y}$ (there can be more subgraphs). We then split $G$ into $\bar{X} \cup \bar{S}$ and $\bar{Y} \cup \bar{S}$.

Note that we need to include $\bar{S}$ in both sub-graphs to maintain the integrity of each v-union. To understand why, consider $G_2$ in Figure 3.2 where $\bar{S} = \{r_3, r_4\}$. According to the definition, there is no 2-robust core. If we split $G_2$ into $\{r_1 - r_2\}$ and $\{r_5 - r_6\}$ (without including $\bar{S}$), both subgraphs are 2-robust and we would return them as cores. The problem happens because v-cliques $Q_1 - Q_4$ "disappear" after we remove the separators $r_3$ and $r_4$. Thus, we should split $G_2$ into $\{r_1 - r_4\}$ and $\{r_3 - r_6\}$ instead and that would further trigger splitting on both subgraphs. Eventually we wish to exclude

**Algorithm 7** SPLIT$(G, \bar{C}, k)$

**Input:** $G$: Simplified similarity graph.
$\bar{C}$: Set of cores.
$k$: Robustness requirement.
**Output:** $\bar{G}$ Set of subgraphs in $G$.

**for each** adjacent $(k+1)$-connected v-unions $Q, Q'$ **do**
  find a pair of nodes $a \in Q \setminus Q', b \in Q' \setminus Q$.
  construct flow-network $G'$ and compute $\kappa(a, b)$ by Ford & Fulkerson Algorithm.
  **if** $\kappa(a, b) \leq k$ **then**
    get separator $\bar{S}$ from $G'$ and remove $\bar{S}$ from $G$ to obtain disconnected subgraphs; mark $\bar{S}$ as "separator" and add it to each subgraph in $G$.
    return the set $\bar{G}$ of subgraphs.
  **end if**
**end for**
**if** $\bar{G} = \phi$ **then**
  add $G$ to $\bar{C}$.
**end if**
**return** $\bar{G}$;

the separator nodes from any core, so we mark them as "separators" and exclude them from the returned cores.

Algorithm SPLIT (details in Algorithm 7) takes a $(k+1)$-overlap subgraph $G$ as input and decides if $G$ is $k$-robust. If not, it splits $G$ into subgraphs on which we will then re-apply screening.

1. For each pair of adjacent $(k+1)$-connected v-unions $Q, Q' \in G$, find $a \in Q \setminus Q', b \in Q' \setminus Q$. Construct flow network $G'(V', E')$ and apply Ford & Fulkerson Algorithm Ford and Fulkerson [1962] to compute the max flow.
2. Once we find nodes $a, b$ where $\kappa(a, b) \leq k$, use the min cut of the corresponding flow network as separator $\bar{S}$. Remove $\bar{S}$ and obtain several subgraphs. Add $\bar{S}$ back to each subgraph and mark $\bar{S}$ as "separator". Return the subgraphs.
3. Otherwise, $G$ passes $k$-robustness test and output it as a core.

**Example 3.2.15** *Continue with Example 3.2.13 and $k = 2$. There are four 3-connected v-unions. When we check $Q_1$ and $Q_3$, we find $\bar{S} = \{r_3, r_4\}$. We then split $G_2$ into two subgraphs $\{r_1 - r_4\}$ and $\{r_3 - r_6\}$, while marking $r_3$ and $r_4$ as "separators".*

*Now consider graph $G_1$ in Figure 3.2 and $k = 2$. There are four 3-connected v-unions (actually four v-cliques) and six pairs of adjacent v-unions. For $Q_1$ and $Q_2$, we check nodes $r_2$ and $r_4$ and find $\kappa(r_2, r_4) = 3$. Similarly we check for every other pair of adjacent v-unions and decide that the graph is 2-robust.* □

**Proposition 3.2.16** *Let $p$ be the total number of pairs of adjacent v-unions, and $g$ be the number of nodes in the input graph. Algorithm* SPLIT *runs in time $O(pg^{2.5})$.* □

Recall that if we solve the Max-Flow Problem directly for each pair of sources in the original graph, the complexity is $O(|\bar{L}|^{4.5})$, which would be dramatically higher.

#### 3.2.3.3 Full algorithm

We are now ready to present the full algorithm, CORE (Algorithm 8). Initially, it initializes the working queue $\mathbf{Q}$ with only input $G$ (Line 1). Each time it pops a subgraph $G'$ from $\mathbf{Q}$ and invokes SCREEN (Lines 3-4). If the output of SCREEN is still $G'$ (so $G'$ is a $(k+1)$-overlap subgraph) (Line 5), it removes any node with mark "separator" in $G'$ and puts the new subgraph into the working queue (Line 7), or invokes SPLIT on $G'$ if there is no separator (Line 9). Subgraphs output by SCREEN and SPLIT are added to the queue for further examination (Lines 10, 13) and identified cores are added to $\bar{C}$, the core set. It terminates when $\mathbf{Q} = \emptyset$.

The correctness of algorithm CORE is guaranteed by the following Lemmas (all proofs in Appendix .4).

**Lemma 3.2.17** *For each pair of adjacent nodes $r, r'$ in graph $G$, there exists a maximal $k$-robust partitioning such that $r, r'$ are in the same subgraph.* □

**Lemma 3.2.18** *The set of nodes in a separator $\bar{S}$ of graph $G$ does not belong to any core in $G$, where $|\bar{S}| \leq k$.* □

**Proposition 3.2.19** *Let $G$ be the input graph and $q$ be the number of $(k+1)$-connected v-unions in $G$. Define $a, p, g, m$, and $|\bar{L}|$ as in Proposition .2.5 and .3.5. Algorithm* CORE *finds correct cores of $G$ in time $O(q((m^2 + a)|\bar{L}| + pg^{2.5}))$ and is order independent.* □

**Algorithm 8** CORE($G, \bar{L}, k$)

**Input:** $G$: Simplified similarity graph.
$\bar{L}$: Inverted list of the similarity graph.
$k$: Robustness requirement.
**Output:** $\bar{C}$ Set of cores in $G$.

```
1: Let Q = {G}, C̄ = ∅;
2: while Q ≠ φ do
3:    Pop G′ from Q;
4:    Let P̄ = SCREEN(G′, L̄, k, C̄);
5:    if P̄ = {G′} then
6:       if G′ contains "separator" nodes then
7:          Remove separators from G′ and add the result to Q if it is not empty;
8:       else
9:          Let S̄ = SPLIT(G′, k, C̄);
10:         add graphs in S̄ to Q;
11:      end if
12:   else
13:      add graphs in P̄ to Q;
14:   end if
15: end while
16: return C̄;
```

**Example 3.2.20** *First, consider graph $G_2$ in Figure 3.2 and $k = 2$. Table 3.2 shows the step-by-step core identification process. It passes screening and is the input for* SPLIT. SPLIT *then splits it into $G_2^1$ and $G_2^2$, where $r_3$ and $r_4$ are marked as "separators".* SCREEN *further splits each of them into $\{r_3\}$ and $\{r_4\}$, both discarded as each represents a single node (and is a separator). So* CORE *does not output any core.*

*Next, consider the motivating example, with the input shown in Table 3.1(b) and $k = 1$. Originally,* Q$= \{G\}$. *After invoking* SCREEN *on $G$, we obtain three subgraphs $G^1, G^2$, and $G^3$.* SCREEN *outputs $G^1$ and $G^3$ as cores since each contains a single node that represents multiple records. It further splits $G^2$ into two single-node graphs $G^4$ and $G^5$, and outputs the latter as a core. Note that if we remove the 1-robustness requirement, we would merge $r_{11} - r_{18}$ to the same core and get false positives.* □

*Case study:* On the data set with 6.8M records, our core-identification algorithm finished in 1.4 minutes. SCREEN is invoked 102K times in total; except the original graph, the size of the input is at most 10.5K and on average 2.2. SPLIT is invoked only 384

Table 3.2: Step-by-step core identification in Example 3.2.20.

| Input | Method | Output |
|---|---|---|
| $G_2$ | SCREEN | $G_2$ |
| $G_2$ | SPLIT | $G_2^1 = \{r_1 - r_4\}, G_2^2 = \{r_3 - r_6\}$ |
| $G_2^1$ | SCREEN | $G_2^3 = \{r_3\}, G_2^4 = \{r_4\}$ |
| $G_2^2$ | SCREEN | $G_2^3 = \{r_3\}, G_2^4 = \{r_4\}$ |
| $G_2^3$ | SCREEN | - |
| $G_2^4$ | SCREEN | - |
| $G$ | SCREEN | $G^1 = \{r_{1/\ldots/7}\}, G^2 = \{r_{11}, r_{12}, r_{14/15}\}$, $G^3 = \{r_{16/17/18}\}$ |
| $G^1$ | SCREEN | Core $\{r_1 - r_7\}$ |
| $G^2$ | SCREEN | $G^4 = \{r_{11}\}, G^5 = \{r_{14/15}\}$ |
| $G^3$ | SCREEN | Core $\{r_{16} - r_{18}\}$ |
| $G^4$ | SCREEN | - |
| $G^5$ | SCREEN | Core $\{r_{14} - r_{15}\}$ |

times; the size of the input is at most 175 and on average 10. Recall that the simplified inverted index contains .15M entries, so SCREEN reduces the size of the input to SPLIT by three orders of magnitude.

## 3.3 Group Linkage

The second stage clusters the cores and the remaining records, which we call *satellites*, into chains. To avoid merging records based only on weak evidence, we require that *a cluster cannot contain more than one satellite but no core*. The key in clustering is two fold: first, we wish to leverage the strong evidence that we collect from the cores generated in the first stage; second, we need to be tolerant to variety of values within the same chain. This section first describes the objective function for clustering (Section 3.3.1) and then proposes a greedy algorithm for clustering (Section 3.3.2).

### 3.3.1 Objective function

Ideally, we wish that each cluster is *cohesive* (each element, being a core or a satellite, is close to other elements in the same cluster) and different clusters are *distinct* (each element is fairly different from those in other clusters). Since we expect that busi-

nesses in the same chain may have fairly different attribute values, we adopt *Silhouette Validation Index (SV-index)* Guo et al. [2010] as the objective function as it is more tolerant to diversity within a cluster. Given a clustering $\mathcal{C}$ of elements $\mathbf{E}$, the SV-index of $\mathcal{C}$ is defined as follows.

$$S(\mathcal{C}) = Avg_{e \in \mathbf{E}} S(e); \tag{3.1}$$

$$S(e) = \frac{a(e) - b(e) + \alpha}{\max\{a(e), b(e)\} + \beta}. \tag{3.2}$$

Here, $a(e) \in [0, 1]$ denotes the similarity between element $e$ and its own cluster, $b(e) \in [0, 1]$ denotes the maximum similarity between $e$ and another cluster, $\beta > \alpha > 0$ are small numbers to keep $S(e)$ finite and non-zero (we discuss in Section 2.4 how we set the parameters in this section). A nice property of $S(e)$ is that it falls in $[-1, 1]$, where a value close to 1 indicates that $e$ is in an appropriate cluster, a value close to $-1$ indicates that $e$ is mis-classified, and a value close to 0 while $a(e)$ is not too small indicates that $e$ is equally similar to two clusters that should possibly be merged. Accordingly, we wish to obtain a clustering with the maximum SV-index. We next describe how we maintain set signatures and compare an element with a cluster.

**Set signature:** When we maintain the signature for a core or a cluster, we keep all values of an attribute and assign a high *weight* to a popular value. Specifically, let $\bar{R}$ be a set of records. Consider value $v$ and let $\bar{R}(v) \subseteq \bar{R}$ denote the records in $\bar{R}$ that contain $v$. The weight of $v$ is computed by $w(v) = \frac{|\bar{R}(v)|}{|\bar{R}|}$.

**Example 3.3.1** *Consider* phone *for core* $Cr_1 = \{r_1 - r_7\}$ *in Table 1.3. There are* 7 *business listings in* $Cr_1$, 5 *providing* 808 ($r_1 - r_5$), *one providing* 101 ($r_6$), *and one providing* 102 ($r_7$). *Thus, the weight of* 808 *is* $\frac{5}{7} = .71$ *and the weight for* 101 *and* 102 *is* $\frac{1}{7} = .14$, *showing that* 808 *is the primary phone for* $Cr_1$. □

Note that when we compare an element $e$ with its own cluster $Ch$, we generate the signature of $Ch$ using its elements excluding $e$.

**Similarity computation:** We consider that an element $e$ is similar to a chain $Ch$ if they have highly similar values on common-value attributes (*e.g.*, name), share at least one *primary* value (we explain "primary" later) on dominant-value attributes (*e.g.*, phone, domain-name); in addition, our confidence is higher if they also share values

on multi-value attributes (*e.g.*, category). Note that although we assume different branches in a business chain should have different values on distinct-value attributes, for some coarse-granularity values, such as state or region of the location, we still often observe sharing of values (*e.g.*, a business chain in one state, or in a few neighboring states). We can treat such attributes (*e.g.*, state) as a multi-value attribute. Formally, we compute the similarity $sim(e, Ch)$ as follows.

$$sim(e, Ch) = \min\{1, sim_s(e, Ch) + \tau w_m sim_{multi}(e, Ch)\}; \tag{3.3}$$

$$sim_s(e, Ch) = \frac{w_c sim_{com}(e, Ch) + w_o sim_{dom}(e, Ch)}{w_c + w_o}; \tag{3.4}$$

$$\tau = \begin{cases} 0 \, if sim_s(e, Ch) < \theta_{th}, \\ 1 \, otherwise. \end{cases} \tag{3.5}$$

Here, $sim_{com}, sim_{dom},$ and $sim_{multi}$ denote the similarity for common-, dominant-, and multi- attributes respectively. We take the weighted sum of $sim_{com}$ and $sim_{dom}$ as strong indicator of $e$ belonging to $Ch$ (measured by $sim_s(e, Ch)$), and only reward weak indicator $sim_{multi}$ if $sim_s(e, Ch)$ is above a pre-defined threshold $\theta_{th}$. Weights $0 < w_c, w_o, w_m < 1$ indicate how much we reward value similarity or penalize value difference; we learn the weights from sampled data.

*Common-Value attribute:* Similarity $sim_{com}$ is computed as the average of similarities on each common-value attribute $A$. For each $A$, $e$ and $Ch$ may each contain a set of values. We penalize values with low similarity and apply cosine similarity (in practice, we take into consideration various representations of the same value):

$$sim_{com}(e.A, Ch.A) = \frac{\sum_{v \in e.A \cap ch.A} w(v)^2}{\sqrt{\sum_{v \in e.A} w(v)^2} \sqrt{\sum_{v' \in ch.A} w(v')^2}}. \tag{3.6}$$

*Dominant-value attribute:* Similarity $sim_{dom}$ rewards similarity on primary values (values with high weights) but meanwhile is tolerant to other different values. If the primary value of an element is the same as that of a cluster on a dominant-value attribute, we consider them having probability $p$ to be in the same chain. Then, if they share $n$ such values, the probability becomes $1 - (1 - p)^n$. Since we use weight to measure whether the value is primary and allow slight difference on values, with a value $v$ from $e$ and $v'$ from $Ch$, we consider the probability that $e$ and $Ch$ belong to the same chain as $p \cdot w_e(v) \cdot w_{Ch}(v') \cdot s(v, v')$, where $w_e(v)$ measures the weight of $v$ in $e$,

$w_{Ch}(v')$ measures the weight of $v'$ in $Ch$, and $s(v, v')$ measures the similarity between $v$ and $v'$. Therefore, we compute $sim_{dom}(r.A, Ch.A)$ as follows.

$$sim_{dom}(e.A, Ch.A) = 1 - \prod_{v \in e.A, v' \in ch.A} (1 - p \cdot w_e(v) \cdot w_{Ch}(v') \cdot s(v, v')). \tag{3.7}$$

*Multi-Value attribute:* We allow diversity in such attributes and use a variant of Jaccard distance for similarity computation. Formally, $sim_{multi}(e, Ch)$ is computed as the sum of the similarities for each multi-value attribute $A$. For each $A$, without losing generality, assume $|e.A| \leq |Ch.A|$ and we treat two values as the same if their similarity is deemed high (above a threshold); we have

$$\begin{aligned} &sim_{multi}(e.A, Ch.A) \\ &= \frac{\sum_{v \in e.A} \max_{v' \in Ch.A, s(v,v') > \theta} s(v, v') \max\{w_e(v), w_{Ch}(v')\}}{\sum_{v \in e.A} \max_{v' \in Ch.A, s(v,v') > \theta} \max\{w_e(v), w_{Ch}(v')\}}. \end{aligned} \tag{3.8}$$

**Example 3.3.2** *Consider element $e = r_{13}$ and cluster $Ch_2 = \{r_{14} - r_{15}\}$ in Example 1.2.2. Assume $w_c = w_o = .5, w_m = .05, \theta_{th} = .8, p = .8$. Since $e$ and $Ch_2$ share exactly the same value on* name, category *and* state*, we have $sim_{\mathsf{name}}(e, Ch_2) = sim_{\mathsf{state}}(e, Ch_2) = sim_{\mathsf{cat}}(e, Ch_2) = 1$. For dominant-value attributes, both $e$ and $Ch_2$ provide* 900 *with weight* 1*, and they do not share URL, so $sim_{dom}(e, Ch_2) = 1 - (1 - .8 \cdot 1 \cdot 1 \cdot 1) = .8$. Thus, we have $sim_s(e, Ch_2) = \frac{.5 \cdot 1 + .5 \cdot .8}{.5 + .5} = .9 > \theta_{th}$, $sim_w(e, Ch_2) = .05 \cdot (1 + 1) = .1$, so $sim(e, Ch_2) = \min\{1, .87 + .1\} = .97$.* □

*Attribute weights:* We apply attribute weights in order to reward attribute value consistency and penalize value difference. Accordingly, for attribute $A$, we define two types of weights: *agreement weight* $w^{agr}$ and *disagreement weight* $w^{dis}$. Disagreement weight is defined as the probability of two records belonging to different chains given that they disagree on $A$-values. Agreement weight is defined as the probability of two records belonging to the same chain given that they agree on $A$-values. We distinguish between ambiguous and unambiguous $A$-values, i.e., values shared by multiple chains (*e.g.*, name) are considered ambiguous and not strong indicator of two records belonging to the same chain, thus have a lower weight; on the other hand, values owned by

a single chain are considered unambiguous and are associated with a high weight. We learn both agreement and disagreement weights from labeled data.

As aforementioned, we consider both agreement and disagreement weights for common-value and dominant-value attributes. For multi-value attributes, we only apply agreement weights to reward weak evidence, and down-weight the weights to make sure the overall similarity is within $[0, 1]$. How we apply agreement and disagreement weights to compute overall similarity can be found in previous work Li et al. [2011].

### 3.3.2 Clustering algorithm

In most cases, clustering is intractable Gonzalez [1982]; Sima and Schaeffer. [2006]. We next propose a greedy algorithm that approximates the optimal clustering. Our algorithm starts with an initial clustering and then iteratively examines if we can improve the current clustering (increase SV-index) by adjusting subsets of the clusters. According to the definition of SV-index, in both initialization and adjusting, we always assign an element to the cluster with which it has the highest similarity.

**Initialization**: Initially, we (1) assign each core to its own cluster and (2) assign a satellite $r$ to the cluster with the highest similarity if the similarity is above threshold $\theta_{ini}$ and create a new cluster for $r$ otherwise. We update the signature of each core along the way. Note that initialization is sensitive in the order we consider the records. Although designing an algorithm independent of the ordering is possible, such an algorithm is more expensive and our experiments show that the iterative adjusting can smooth out the difference.

**Example 3.3.3** *Continue with the motivating example in Table 1.3 and assume* $\theta_{th} = .8$. *First, consider records* $r_1 - r_{10}$, *where* $Cr_1 = \{r_1 - r_7\}$ *is a core. We first create a cluster* $Ch_1$ *for* $Cr_1$. *We then merge records* $r_8 - r_{10}$ *to* $Ch_1$ *one by one, as they share similar names, and either primary phone number or primary URL.*

*Now consider records* $r_{11} - r_{20}$; *recall that there are 2 cores and 5 satellites after core identification. Figure 3.4 shows the initialization result* $\mathcal{C}_a$. *Initially we create two clusters* $Ch_2, Ch_3$ *for cores* $Cr_2, Cr_3$. *Records* $r_{11}, r_{19} - r_{20}$ *do not share any primary value on dominant-value attributes with* $Ch_2$ *or* $Ch_3$, *so have a low similarity with them; we create a new cluster for each of them. Records* $r_{12}$ *and* $r_{13}$ *share the primary phone with* $Cr_2$ *so have a high similarity; we link them to* $Ch_2$. □

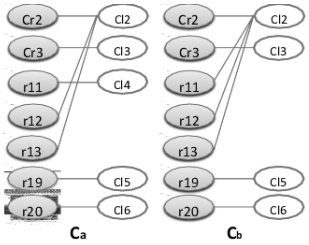


Figure 3.4: Clustering of $r_{11} - r_{20}$ in Table 1.3.

**Cluster adjusting:** Although we always assign an element $e$ to the cluster with the highest similarity so $S(e) > 0$, the result clustering may still be improved by merging some clusters or moving a *subset* of elements from one cluster to another. Recall that when $S(e)$ is close to 0 and $a(e)$ is not too small, it indicates that a pair of clusters might be similar and is a candidate for merging. Thus, in cluster adjusting, we find such candidate pairs, iteratively adjust them by merging them or moving a subset of elements between them, and choose the new clustering if it increases the SV-index.

We first describe how we find candidate pairs. Consider element $e$ and assume it is closest to clusters $Ch$ and $Ch'$. If $S(e) \leq \theta_s$, where $\theta_s$ is a threshold for considering merging, we call it a *border* element of $Ch$ and $Ch'$ and consider $(Ch, Ch')$ as a candidate pair. We rank the candidates according to (1) how many border elements they have and (2) for each border element $e$, how close $S(e)$ is to 0. Accordingly, we define the *benefit* of merging $Ch$ and $Ch'$ as $b(Ch, Ch') = \sum_{e \text{ is a border of } Ch \text{ and } Ch'}(1 - S(e))$, and rank the candidate pairs in decreasing order of the benefit.

We next describe how we re-cluster elements in a candidate pair $(Ch, Ch')$. We adjust by merging the two clusters, or moving the border elements between the clusters, or moving out the border elements and merging them. Figure 3.5 shows the four re-clustering plans for a candidate pair. Among them, we consider those that are valid (*i.e.*, a cluster cannot contain more than one satellite but no core) and choose the one with the highest SV-index. When we compute SV-index, we consider only elements in $Ch, Ch'$ and those that are second-to-closest to $Ch$ or $Ch'$ (their $a(e)$ or $b(e)$ can be changed) such that we can reduce the computation cost. After the adjusting, we need to re-compute $S(e)$ for these elements and update the candidate-pair list accordingly.

**Example 3.3.4** *Consider adjusting cluster $\mathcal{C}_a$ in Figure 3.4. Table 3.3(a) shows similarity of each element-cluster pair and SV-index of each element. Thus, the SV-index*

Table 3.3: Element-cluster similarity and SV-index for clusterings in Figure 3.4. Similarity between an element and its own cluster is in bold and the second-to-highest similarity is in italic. Score $S(e)$ for a border element is in italic.

| | $\text{Ch}_2$ | $\text{Ch}_3$ | $\text{Ch}_4$ | $\text{Ch}_5$ | $\text{Ch}_6$ | $S(e)$ |
|---|---|---|---|---|---|---|
| $\text{Cr}_2$ | **.9** | *.5* | *.5* | *.5* | *.5* | .44 |
| $\text{Cr}_3$ | *.6* | **1** | .5 | .5 | .5 | .4 |
| $r_{11}$ | *.7* | .5 | **1** | .5 | .5 | *.3* |
| $r_{12}$ | **.99** | .5 | *.95* | .5 | .5 | *.05* |
| $r_{13}$ | **1** | .9 | *.95* | .5 | .5 | *.05* |
| $r_{19}$ | *.5* | *.5* | *.5* | **1** | *.5* | .5 |
| $r_{20}$ | *.5* | *.5* | *.5* | *.5* | **1** | .5 |

(a) Cluster $\mathcal{C}_a$.

| | $\text{Ch}_2$ | $\text{Ch}_3$ | $\text{Ch}_5$ | $\text{Ch}_6$ | $S(r)$ |
|---|---|---|---|---|---|
| $r_{11}$ | **.79** | *.5* | *.5* | *.5* | .37 |
| $r_{12}$ | **.96** | *.5* | *.5* | *.5* | .48 |
| $r_{13}$ | **.97** | *.9* | .5 | .5 | *.07* |
| $\text{Cr}_2$ | **.87** | *.5* | *.5* | *.5* | .43 |
| $\text{Cr}_3$ | *.58* | **1** | .5 | .5 | .42 |
| $r_{19}$ | *.5* | *.5* | **1** | *.5* | .5 |
| $r_{20}$ | *.5* | *.5* | *.5* | **1** | .5 |

(b) Cluster $\mathcal{C}_b$.

*is .32.*

*Suppose* $\theta_s = .3$. *Then,* $r_{11} - r_{13}$ *are border elements of* $Ch_2$ *and* $Ch_4$*, where* $b(Ch_2, Ch_4) = .7 + .95 + .95 = 2.6$ *(there is a single candidate so we do not need to compare the benefit). For the candidate, we have two re-clustering plans,* $\{\{r_{11} - r_{13}, Cr_2\}\}$, $\{\{r_{11} - r_{13}\}, \{Cr_2\}\}$, *while the latter is invalid. For the former (*$\mathcal{C}_b$ *in Figure 3.4), we need to update* $S(e)$ *for every element and the new SV-index is* $.4$ *(Table 3.3(b)), higher than the original one.* □

The full clustering algorithm CLUSTER (details in Algorithm 9) goes as follows.

1. Initialize a clustering $\mathcal{C}$ and a list $Que$ of candidate pairs ranked in decreasing order of merging benefit. (Lines 1-2).
2. For each candidate pair $(Ch, Ch')$ in $Que$ do the following.

   (a) Examine each valid adjusting plan and compute SV-index for it, and choose the one with the highest SV-index. (Line 4).

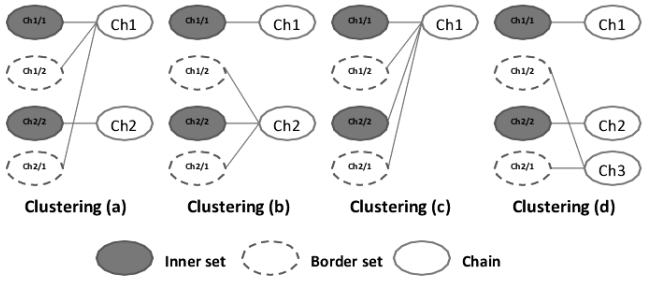


Figure 3.5: Reclustering plans for $Ch_1$ and $Ch_2$.

(b) Change the clustering if the new plan has a higher SV-index than the original clustering. Recompute $S(e)$ for each relevant element $e$ and move $e$ to a new cluster if appropriate. Update $Que$ accordingly. (Lines 6-16).

3. Repeat Step 2 until $Que = \emptyset$.

**Proposition 3.3.5** *Let $l$ be the number of distinct candidate pairs ever in $Que$ and $|\mathbf{E}|$ be the number of input elements. Algorithm* CLUSTER *takes time $O(l \cdot |\mathbf{E}|^2)$.* □

Note that we first block records according to name similarity and take each block as an input, so typically $|\mathbf{E}|$ is quite small. Also, in practice we need to consider only a few candidate pairs for adjusting in each input, so $l$ is also small.

**Example 3.3.6** *Continue with Example 3.3.4 and consider adjusting $\mathcal{C}_b$. Now there is one candidate pair $(Ch_2, Ch_3)$, with border $r_{13}$. We consider clusterings $\mathcal{C}_c$ and $\mathcal{C}_d$. Since $S(\mathcal{C}_c) = .37 < .40$ and $S(\mathcal{C}_d) = .32 < .40$, we keep $\mathcal{C}_b$ and return it as the result. We do not merge records $Ch_2 = \{r_{11} - r_{15}\}$ with $Ch_3 = \{r_{16} - r_{18}\}$, because they share neither phone nor the primary URL.* CLUSTER *returns the correct chains.*□

## 3.4 Experimental Evaluation

This section describes experimental results on a real-world business-listing data set. Experimental results show high accuracy and scalability of our techniques.

**Algorithm 9** CLUSTER($\mathbf{E}, \theta_s$)

**Input:** $\mathbf{E}$: A set of cores and satellites for clustering.
$\theta_s$: Pre-defined threshold for considering merging.
**Output:** $\mathcal{C}$: A clustering of elements in $\mathbf{E}$.

```
Initialize C according to E;
Compute S(C) and generate a list Que of candidate pairs;
for each candidate pair (Ch, Ch') ∈ Que do
    compute SV-index for its valid re-clustering plans and choose the clustering C_max with the highest SV-index;
    if S(C) < S(C_max) then
        let C = C_max, change = true;
        while change do
            change = false;
            for each relevant element e do
                recompute S(e);
                When appropriate, move e to a new cluster and set change = true;
                if S(e) < θ_s in the previous or current C then
                    update the merging benefit of the related candidate pair and add it to Que or remove it from Que when appropriate;
                end if
            end for
        end while
    end if
end for
return C;
```

### 3.4.1 Experiment settings

**Data and golden standard:** We experimented on a set of business listings in the US obtained from *YellowPages.com*. There are 6.8 million business listings, each with attributes name, phone, URL, location and category. We experimented on the whole data set to study scalability of our techniques.

To evaluate accuracy of our techniques, we considered four subsets. First, we considered a *Random data set* with 2062 records, where 1559 belong to 30 randomly selected business chains, and 503 do not belong to any chain; among the 503 records, 86 are highly similar in name to records in the business chains and the rest are randomly selected. We also considered three hard cases. (1) *AI data set* contains 2446 records for the same business chain *Allstate Insurance*. These records have the same name, but

Table 3.4: Statistics of the business-listing subsets.

| | #Records | #Chains | Chain sizes | #Single-business records |
|---|---|---|---|---|
| *Random* | 2062 | 30 | [2, 308] | 503 |
| *AI* | 2446 | 1 | 2446 | 0 |
| *UB* | 322 | 7 | [2, 275] | 5 |
| *FBIns* | 1149 | 14 | [33, 269] | 0 |

1499 provide URL *"allstate.com"*, 854 provide another URL *"allstateagencies.com"*, while 130 provide both, and 227 records do not provide any value for phone or URL. (2) *UB data set* contains 322 records with exactly the same name *Union Bank* and highly similar category values; 317 of them belong to 9 different chains while 5 do not belong to any chain. (3) *FBIns data set* contains 1149 records with similar names and highly similar category values; they belong to 14 different chains. Among the records, 708 provide the same wrong name *Texas Farm Bureau Insurance* and meanwhile provide a wrong *URL farmbureauinsurance-mi.com*. For each data set, we manually verified all the chains by checking store locations provided by the business-chain websites and used it as the golden standard. Table 3.4 shows statistics of the four subsets.

**Measure:** We considered each business chain as a cluster and compared pairwise linking decisions with the golden standard. We measured the quality of the results by *precision* ($P$), *recall* ($R$), and *F-measure* ($F$). If we denote the set of true-positive pairs by $TP$, the set of false-positive pairs by $FP$, and the set of false-negative pairs by $FN$, then, $P = \frac{|TP|}{|TP|+|FP|}$, $R = \frac{|TP|}{|TP|+|FN|}$, $F = \frac{2PR}{P+R}$.

**Implementation:** We implemented the technique we proposed in this paper, and call it CORECLUSTER. In core generation, we considered two records to be similar if (1) their name similarity is above .95; and (2) they share at least one phone or URL domain name. We required 1-robustness for cores. In clustering, (1) for blocking, we put records whose name similarity is above .8 in the same block; (2) for similarity computation, we computed string similarity by Jaro-Winkler distance Cohen et al. [2003], we set $\alpha = .01, \beta = .02, \theta_{th} = .6, p = .8$, we learned other weights from 1000 records randomly selected from *Random* data and we used the learnt weights on all data sets; (3) for clustering, we set $\theta_{ins} = .8$ for initialization and $\theta_s = .2$ for adjusting. We discuss later how these choices may affect our results.

For comparison, we also implemented the following baselines:

- SAMENAME links records with highly similar names (similarity above .95);
- CONNECTEDGRAPH generates the similarity graph as CORECLUSTER but con-

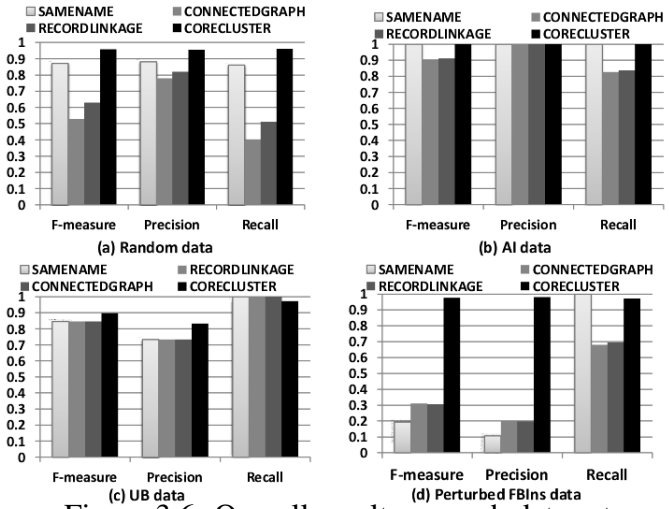

Figure 3.6: Overall results on each data set.

siders each connected subgraph as a chain;

- RECORDLINKAGE computes record similarity by Eq.(3) with the same weights as in CORECLUSTER and then applies one baseline clustering technique PARTITION Hassanzadeh et al. [2009a], which links all records that are transitively similar. (We experimented on two other clustering methods and obtained similar results.)

We implemented the algorithms in Java. We used a Linux machine with Intel Xeon X5550 processor (2.66GHz, cache 8MB, 6.4GT/s QPI). We used MySQL to store the business listings.

### 3.4.2 Evaluating effectiveness

We first evaluate effectiveness of our algorithms. Figure 3.6 compares CORECLUSTER with the three baseline methods on the data sets. We have the following observations on data sets *Random, AI,* and *UB*. (1) CORECLUSTER obtains the highest F-measure (above .9) on each data set. It has the highest precision on each subset as it applies core identification and leverages the strong evidence collected from resulting cores. It also has a very high recall (above .95) on each subset because the clustering phase is tolerant to diversity of values within chains. (2) SAMENAME can have false positives when listings of highly similar names belong to different chains and can also have false negatives when some listings in a chain have fairly different names from other listings. It only performs well in *AI*, where it happens that all listings have the same name

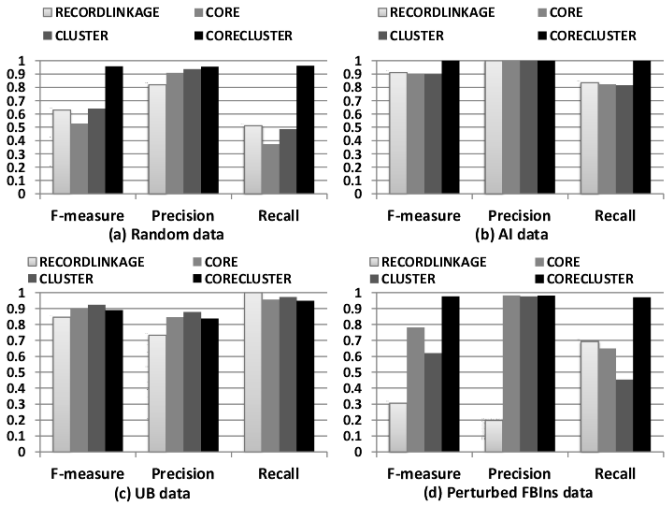

Figure 3.7: Contribution of different components.

and belong to the same chain. (3) CONNECTEDGRAPH requires in addition sharing at least one phone or URL domain. As a result, it has a lower recall than SAMENAME; it has less false positives than SAMENAME, but because it has less true positives, its precision can appear to be lower too. (4) RECORDLINKAGE requires high weighted similarity between the records, which is a weaker requirement for merging records than CONNECTEDGRAPH. On various data sets it has very similar numbers of false positives to but much more true positives than CONNECTEDGRAPH; as a result, it has a higher recall and also a higher precision.

On the *FBIns* data set, because a large number of listings (708) have both a wrong name and a wrong URL, each method wrongly puts all records in the same chain. We manually perturbed the data as follows: (1) among the 708 listings with wrong URLs, 408 provide a single (wrong) URL and we fixed it; (2) for all records we set name to *"Farm Bureau Insurance"*, so removed hints from business names. Even after perturbing, this data set remains the hardest data set and Figure 3.6(d) shows the results. We observe that all baseline methods have very low F-measure (below .3) while CORECLUSTER still obtains a F-measure as high as .98. We used the perturbed *FBIns* data set hereafter instead of the original one for other experiments.

**Contribution of different components:** We compared CORECLUSTER with (1) CORE, which applies Algorithm COREIDENTIFICATION but does not apply clustering, and (2) CLUSTER, which considers each individual record as a core and applies Algorithm CLUSTER. Figure 3.7 shows the results on each data set. First, we observe that CORE improves over the baseline method RECORDLINKAGE on precision but has a lower re-

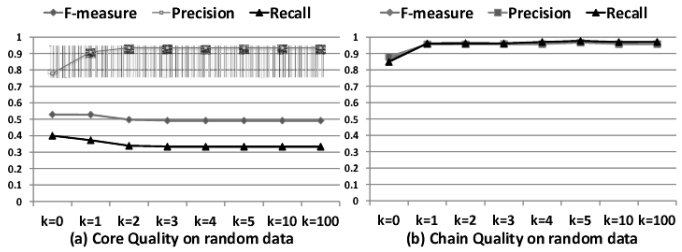

Figure 3.8: Effect of robustness requirement on *Random* data.

call, because it sets a high requirement for merging records into chains. Note however that its goal is indeed to obtain a high precision such that the strong evidence collected from the cores are trustworthy for the clustering phase. Second, CLUSTER often performs better than RECORDLINKAGE. On some data sets (*Random, UB*) it can obtain an even higher precision than CORE, because CORE can make mistakes when too many records have erroneous values, but CLUSTER may avoid some of these mistakes by considering also similarity on state and category. However, applying clustering on the results of CLUSTER would not change the results, but applying clustering on the results of CORE can obtain a much higher F-measure, especially a higher recall (98% higher than CLUSTER on *Random*). This is because the result of CLUSTER lacks the strong evidence collected from high-quality cores so the final results would be less tolerant to diversity of values, showing the importance of core identification. Finally, we observe that CORECLUSTER obtains the best results in most of the data sets. The only exception is *UB*, where its F-measure is 2% lower than that of CLUSTER; this data set contains less diverse values for the same chain so CLUSTER has a high recall. We note also that although CORECLUSTER has more false positives than CORE, it can have a higher precision as it has much more true positives.

We next evaluate various choices in the two stages. Unless specified otherwise, we observed similar patterns on each data set and report the results on *Random*; in some cases we report the results on perturbed *FBIns* since results on other data sets are not very distinguishable.

#### 3.4.2.1 Core identification

**Robustness requirement:** We first show how the robustness requirement can affect the results. Figure 3.8 shows the results when we vary $k$. We have three observations. (1) When $k = 0$, we essentially take every connected subgraph as a core, so the gen-

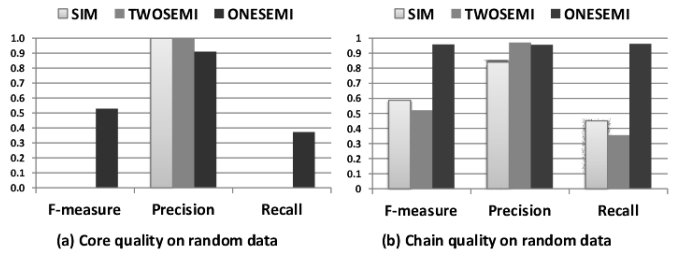

Figure 3.9: Effect of graph generation on *Random* data.

erated cores can have a much lower precision; those false positives cause both a low precision and a low recall for the resulting chains because we may collect some wrong strong evidence while miss some other such evidence. (2) When we vary $k$ from 1 to 4, the number of false positives decreases while that of false negatives increases for the cores, and the F-measure of the chains increases but only very slightly. (3) When we continue increasing k, the results of cores and clusters remain stable. This is because setting k=4 already splits the graph into subgraphs, each containing a single v-clique, so further increasing k would not change the cores. This shows that considering $k$-robustness is important, but $k$ does not need to be too high.

**Graph generation:** We compared three edge-adding strategies for similarity graphs: SIM takes weighted similarity on name, phone, URL, category and requires a similarity of over .8; TWOSEMI requires sharing name and at least two values on dominant-value attributes; ONESEMI requires sharing name and one value on dominant-value attributes. Recall that by default we applied ONESEMI. Figure 3.9 compares these three strategies. We observe that (1) SIM requires similar values on each attribute except location and so has a high precision, with a big sacrifice on recall for the cores; as a result, the F-measure of the chains is very low (.59); (2) TWOSEMI has the highest requirements and so even lower recall than SIM for the cores, and in turn it has the lowest F-measure for the chains (.52). This shows that only requiring high precision for cores with big sacrifice on recall can also lead to low F-measure for the chains.

We also varied the similarity requirement for names and observed very similar results (varying by .04%) when we varied the threshold from .8 to .95.

**Core identification:** We compared two core-generation strategies: COREIDENTIFICATION iteratively invokes SCREEN and SPLIT,
ONLYSCREEN only iteratively invokes SCREEN. Recall that by default we apply COREIDENTIFICATION. We observe the same results on all data sets, showing that the in-

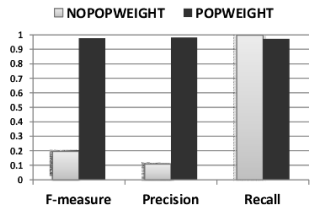


Figure 3.10: Value weights on perturbed *FBIns* data.

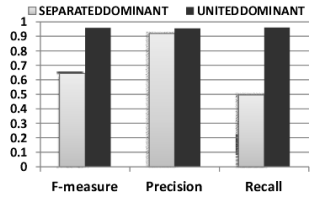


Figure 3.11: Dominant-value attributes on *Random*.

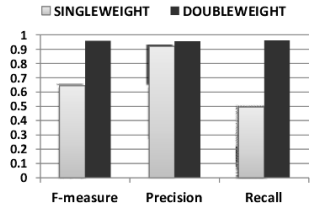


Figure 3.12: Distinct values on *Random* data.

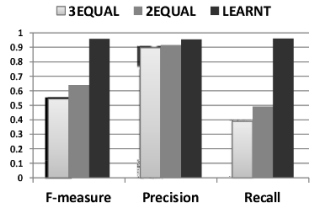


Figure 3.13: Attribute weights on *Random* data.

puts to SPLIT all pass the $k$-robustness test. This shows that although SCREEN in itself cannot guarantee soundness of the resulting cores (k-robustness), it already does well in obtaining $k$-robust cores. The cases in which SPLIT can make a difference appear to be rare in practice.

#### 3.4.2.2 Clustering

**Value weight:** We first show importance of setting popularity weights in set signature. Figure 3.10 compares the results with and without setting popularity weights on perturbed *FBIns* data. We observe that setting the popularity weight helps distinguish primary values from unpopular values, thus can significantly improve the precision and so improve F-measure. We also observe an improvement of 1% on *Random* and 5% on *UB* for F-measure.

**Attribute weight:** We next considered our weight learning strategy. We first compared SEPARATEDDOMINANT, which learns separated weights for different dominant-value attributes, and UNITEDDOMINANT, which considers all such attributes as a whole and learns one single weight for them. By default we applied UNITEDDOMINANT. Figure 3.11 shows that the latter improves over the former by 95.4% on recall and obtains slightly higher precision, because it penalizes only if neither phone nor URL

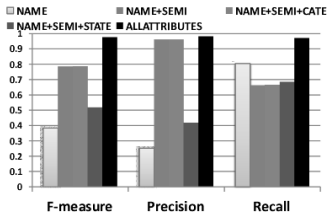


Figure 3.14: Attribute contribution on perturbed *FBIns*.

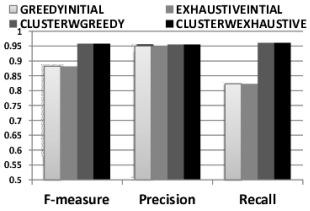


Figure 3.15: Clustering strategies on *Random* data.

is shared and so is more tolerant to different values for dominant-value attributes.

Next, we compared SINGLEWEIGHT, which learns a single weight for each attribute, and DOUBLEWEIGHT, which learns different weights for distinct values and non-distinct values for each attribute. By default we applied DOUBLEWEIGHT. Figure 3.12 shows that DOUBLEWEIGHT significantly improves the recall (by 94%) since it rewards sharing of distinct values, and so can link some satellite records with null values on dominant-value attributes to the chains they should belong to.

We also compared three weight-setting strategies: (1) 3EQUAL considers common-value attributes, dominant-value attributes, and multi-value attributes, and sets the same weight for each of them; (2) 2EQUAL sets equal weight of .5 for common-value attributes and dominant-value attributes, and weight of .1 for each multi-value attribute; (3) LEARNED applies weights learned from labeled data. Recall that by default we applied LEARNED. Figure 3.13 compares their results. We observe that (1) 2EQUAL obtains higher F-measure than 3EQUAL, since it distinguishes between strong and weak indicators for record similarity; (2) LEARNED significantly outperforms the other two strategies, showing effectiveness of weight learning.

**Attribute contributions:** We then consider the contribution of each attribute for chain classification. Figure 3.14 shows the results when we consider only a subset of the attributes on the perturbed *FBIns* data. We have four observations. (1) Considering only name but not any other attribute obtains a high recall but a very low precision, since all listings on this data set have the same name. (2) Considering dominant-value attributes in addition to name can improve the precision significantly and improve the F-measure by 104%. (3) Considering category in addition does not further improve the results while considering state in addition even drops the precision significantly, since three chains in this data set contain the same wrong value on state. (4) Considering both category and state improves the recall by 46% and obtains the highest

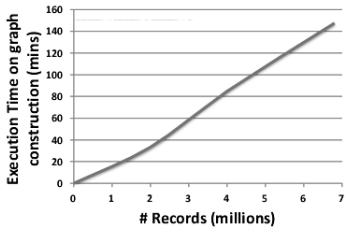


Figure 3.16: Scalability of our algorithm.

F-measure.

**Clustering strategy**: We compared four clustering algorithms:
GREEDYINITIAL performs only initialization as we described in Section 3.3; EXHAUSTIVEINITIAL also performs only initialization, but by iteratively conducting matching and merging until no record can be merged to any core; CLUSTERWGREEDY applies cluster adjusting on the results of GREEDYINITIAL, and CLUSTERWEXHAUSTIVE applies cluster adjusting on the results of EXHAUSTIVEINITIAL. Recall that by default we apply CLUSTERWGREEDY. Figure 3.15 compares their results. We observe that (1) applying cluster adjusting can improve the F-measure a lot (by 8.6%), and (2) exhaustive initialization does not significantly improve over greedy initialization, if at all. This shows effectiveness of the current algorithm CLUSTER.

**Robustness w.r.t. parameters:** We also ran experiments to test robustness against parameter setting. We observed very similar results when we ranged $p$ from .8 to 1, $\theta_{th}$ from .5 to .7, $\theta_{ini}$ from .6 to .9, and $\theta_s$ from .1 to .4.

### 3.4.3 Evaluating efficiency

Our algorithm finished in 2.4 hours on the whole data set, which contains 6.8 million listings. It spent 2.2 hours for graph construction, 1.4 minutes for core generation, and 15 minutes for clustering. This is reasonable given that it is an offline process.

We next focus on graph construction since it costs the longest time. We randomly divided the whole data set into 3 subsets of the same size. We started with one subset and gradually added more. Figure 3.16 shows that the execution time grows linearly in the size of the data.

### 3.4.4 Summary and recommendations

We summarize our observations as follows.

1. Identifying cores and leveraging evidence learned from the cores is crucial in business-chain identification.
2. There are often erroneous values in real data and it is important to be robust against them; applying OneSemi and requiring $k \in [1, 5]$ already performs well on most data sets that have reasonable number of errors.
3. Learning different weights for different attributes, distinguishing the weights for distinct and non-distinct values, and setting weights of values according to their popularity are critical for obtaining good clustering results.
4. Our algorithm is robust on reasonable parameters setting.
5. Our algorithm is efficient and scalable.

## 3.5 Related Work

Record linkage has been extensively studied in the past (surveyed in Elmagarmid et al. [2007]; Koudas et al. [2006]). Traditional linkage techniques aim at linking records that refer to the same real-world entity, so implicitly assume value consistency between records that should be linked. Group linkage is different in that it aims at linking records that refer to entities in the same group. The variety of individual entities requires better use of strong evidence and tolerance on different values even within the same group. These two features differ our work from any previous linkage technique.

For record clustering in linkage, existing work may apply transitive rule Hernandez and Stolfo [1998], or do match-and-merge Benjelloun et al. [2009], or reduce it to an optimization problem Flake et al. [2004]. Our work is different in that our core-identification algorithm aims at being robust to a few erroneous records; and our clustering algorithm emphasizes leveraging the strong evidence collected from the cores.

For record-similarity computation, existing work can be rule based Hernandez and Stolfo [1998], classification based Fellegi and Sunter [1969b], or distance based Dey [2008]. There has also been work on weight (or parameter) learning from labeled data Fellegi and Sunter [1969b]; Winkler [2002]. Our work is different in that (1) we weight the values according to their popularity within a group such that

similarity on primary values (strong evidence) is rewarded more, (2) we are tolerant to difference on individual values from different entities in the same group, and (3) we distinguish weights for distinct values and non-distinct values such that similarity on distinct values is rewarded more. Note that some previous works are also tolerant to different values but leverage evidence that may not be available in our contexts: Guo et al. [2010] is tolerant to possibly false values by considering agreement between different data providers, Fan et al. [2009] is tolerant to dynamic semantics of different relations, and Li et al. [2011] is tolerant to out-of-date values by considering time stamps; we are tolerant to diversity within the same group.

Two-stage clustering has been proposed in the IR and machine learning community Bansal et al. [2007]; Larsen and Aone [1999]; Liu et al. [2002]; Wijaya and Bressan [2009]; Yoshida et al. [2010]; however, they identify cores in different ways. Techniques in Larsen and Aone [1999]; Wijaya and Bressan [2009] consider a core as a single record, either randomly selected or selected according to the weighted degrees of nodes in the graph. Techniques in Yoshida et al. [2010] generate cores using agglomerative clustering. These two methods are not robust to erroneous values. Techniques in Bansal et al. [2007] identify cores as *bi-connected components*, where removing any node would not disconnect the graph. Although this corresponds to the *1-robustness* requirement in our solution (defined in Section 3.2), they generate overlapping clusters; it is not obvious how to derive non-overlapping clusters in applications such as business-chain identification and how to extend their techniques to guarantee $k$-robustness. Finally, techniques in Larsen and Aone [1999]; Liu et al. [2002] require knowledge of the number of clusters for one of the stages, so do not directly apply in our context. We compare with these methods whenever applicable in experiments (Section 2.4), showing that our algorithm is robust in presence of erroneous values and consistently generate high-accuracy results on sets of records with different features.

We also distinguish our work from the so-called *group linkage* in Huang [2010]; On et al. [2007], which has different goals from our work. On et al. On et al. [2007] decided similarity between groups of records. Huang Huang [2010] essentially solved the record-linkage problem (each entity is a *group* of records) by analysis of social network. Our goal is to find records that belong to the same group.

## 3.6 Summary

In this chapter we studied how to link records to identify groups. We proposed a two-stage algorithm that is shown empirically scalable and accurate over two real-world data sets. Future work includes studying the best way to combine record linkage and group linkage, extending our work for finding overlapping groups, and applying the two-stage framework in other contexts where tolerance to value diversity is critical.

# Chapter 4

# An Application: Chronos: Facilitating History Discovery by Linking Temporal Records

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. From such data, users often wish to search for entities in a particular period, and understand the history of one entity or all entities in the data set. For example, *DBLP*[1] lists research papers over many decades; *DBLP* users may wish to find authors by name and year, find the publication history and affiliation history of an author, find the number of her co-authors in each year over time, find her research topics over time, and so on.

A major challenge for enabling such search and exploration is to identify records that describe the same real-world entity over a long period of time; only with such an integrated view, we will be able to trace the history of that entity and collect statistics over time. However, linking temporal records is by no means easy. First, we need to be able to link together records for the same real-world entity but at different times. This is hard because entities can evolve over time; for example, a researcher can move from one affiliation to another, change her research topic, and collaborate with different co-authors over time. Thus, records that describe the same real-world entity at different

[1] http://www.dblp.org/.

times can contain different values; blindly requiring value consistency of the linked records may cause false negatives. Second, we need to be able to distinguish records that share common attribute values but refer to different real-world entities. This is especially hard for temporal records because it is more likely to find highly similar entities over a long time period than at the same time; for example, having two persons with highly similar names in the same university over the past 30 years is more likely than at the same time. Thus, records that describe different entities at different times can share common values; blindly matching records that have similar attribute values can cause false positives.

We describe the CHRONOS system[1], which offers users a useful tool for finding real-world entities over time and understanding history of entities in the bibliography domain. The core of CHRONOS is a temporal record-linkage algorithm, which is tolerant to value evolution over time Li et al. [2011]. Our algorithm can obtain an F-measure of over 0.9 in linking author records and can fix errors made by *DBLP*. There are two key ideas for the linkage techniques: first, we apply *time decay* that captures the effect of time elapse on entity value evolution; second, we apply *temporal clustering* that considers records in time order and accumulates evidence over time to enable decision making with a global view.

This chapter presents the novel features of CHRONOS and focuses on the following two aspects. First, we show how CHRONOS allows users to explore the history, including searching authors in a particular time period or in a particular affiliation, tracing the history of publications, co-authors, and affiliations of a particular author, and understanding the statistics of authors, publications, etc., over time. Second, we further show how CHRONOS helps users understand our linkage results (linking citation records for the same real-world author) by comparing our results with those of existing systems, such as the manual linkage results from *DBLP*, highlighting differences in the results, explaining to users our decisions, and answering "what-if" questions such as "What would the results look like if we had not applied time decay?" and "What if we had removed these three records?"

In the rest of the chapter, we first describe the features of the CHRONOS system in Section 4.1, and then describe the system architecture and underlying temporal linkage

[1] *Chronos* is a Greek God for *time*; he has three heads, a man, a bull, and a lion, showing the importance of "linkage".

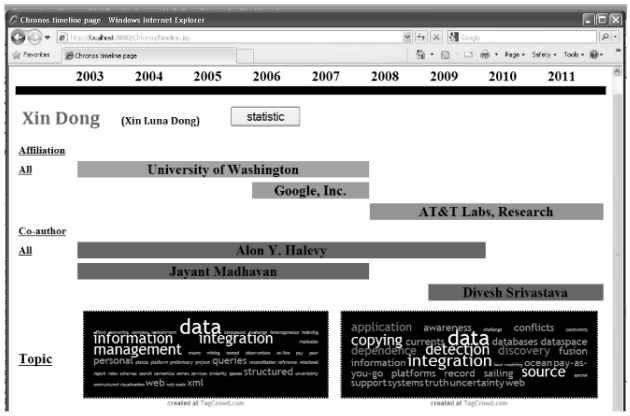


Figure 4.1: Research history of author "Xin Dong".

algorithm in Section 4.2. We discuss related work in Section 4.3, and conclude in Section 4.4.

## 4.1 System Features

We start by describing the features of CHRONOS through user scenarios. CHRONOS includes all papers collected by *DBLP* till June 1st, 2012. For each paper, we extract a record for each author of that paper, with information for name, paper title, co-authors, conference, and year. In addition, we enrich each author record with information on email and affiliation for the associated time stamp whenever possible and collect such information from digital libraries such as *ACM*[1], *IEEE*[2], *Scopus*[3], journal websites, the PDFs of the papers, and so on.

First, CHRONOS allows users to search for authors over time and find the history of particular authors.

**Example 4.1.1** *Consider a user who would like to find an author named "Xin Dong". She searches "Xin Dong" and* CHRONOS *returns 6 "Xin Dong" entities and 1 "Dong Xin" entity, each one showing the publication period and current affiliation. The user can select one of them, or refine the query by searching "Xin Dong 2011" (the authors named "Xin Dong" and published in 2011) or searching "Xin Dong AT&T" (the*

[1]http://dl.acm.org/
[2]http://ieeexplore.ieee.org/
[3]http://www.scopus.com/

*authors named "Xin Dong" and was at AT&T at some time").*

*Suppose the user has selected one "Xin Dong" entity. She can click the "History" button to trace the history of various aspects of this author, such as her affiliations, co-authors, research topics, and so on. Figure 4.1 shows a screenshot for this. It shows that this author stayed at "University of Washington" in 2003-2007, at "Google, Inc" in 2006-2007, and at "AT&T Labs" from 2008 till now. Note that this history is purely derived from the author's publications, so may not be precise (*e.g.*, the author may have joined AT&T Labs in 2007 but started publishing with that affiliation only since 2008). The topic is generated for every five years as the tag cloud*[1] *of publication titles and available abstracts.*

*The user can also click the "Statistics" button to see statistics about the author, including graphs of the number of publications by that author over years, the number of co-authors over years, and so on. The user can even see statistics of all authors over years, such as the number of authors and the number of publications.* □

Second, in case the user is interested in the author-linkage results, CHRONOS compares its own temporal-linkage results with (1) the manual linkage results by *DBLP*, and (2) the linkage results by BASIC, a traditional record-linkage technique that compares each pair of author records and applies transitivity in clustering the records into author entities Hassanzadeh et al. [2009b].

**Example 4.1.2** *Suppose the user is interested in comparing the listed papers by* CHRONOS *and by* DBLP*, she can click the "Comparison" button.* CHRONOS *will show side-by-side the list of papers according to the linkage results by* CHRONOS*, by* DBLP*, and by* BASIC *(see Figure 4.2).* CHRONOS *also highlights differences between the lists: for each list from* DBLP *and* BASIC*, it highlights the publications not included in its own list; for its own list, it highlights the publications not included in the list from* DBLP *or from* BASIC *(using different colors).*

*If the user would like to understand the different decisions, she can click on one highlighted publication and* CHRONOS *would explain the reason. For example, if she wonders why publication #22 from* DBLP *is excluded from the list by* CHRONOS*, she can click on #22 and* CHRONOS *would explain "The author 'Xin Dong' of that paper is from 'University of Nebraska-Lincoln', it is unlikely that she moved from 'AT&T*

[1] http://www.tagcrowd.com/.


Figure 4.2: Comparison on publications by "Xin Dong". Only a subset of papers are shown to fit the differences in one screen.

*Labs' to 'University of Nebraska-Lincoln' in 2010 and moved back to 'AT&T Labs' in 2011". As another example, if she wonders why publication #15 (excluded by* BASIC*) is included in the list by* CHRONOS*, she can click on publication #15 and* CHRONOS *would explain "The author 'Xin Dong' of that paper is from 'University of Washington'; later she moved to 'AT&T Labs' in 2008".*

*If the user is curious and would like to understand more, such as why it is considered unlikely for the author to "move from 'AT&T Labs' to 'University of Nebraska-Lincoln' in 2010 and move back to 'AT&T Labs' in 2011" but likely for the author to "move from 'University of Washington' to 'AT&T Labs' in 2008", she can ask for more details. For the previous explanation for publication #22, if the user clicks the "Details" button, the explanation will be extended as "The author was at 'AT&T Labs' in 2008-2010; the probability that she moved to another affiliation in 2010 is .26 and the probability that she moved again in 2011 is .13". Similarly, extended explanation for publication #15 can be "The author was at 'University of Washington' in 2003-2007; the probability that she moved to another affiliation in 2008 is .55".* □

Third, for advanced users, CHRONOS answers "what-if" questions and help users compare different results when revising some of the data or applying different linkage methods. Specifically, CHRONOS allows the user to (1) select a subset of records by searching or by selecting on record basis, (2) change the time stamp of some of the

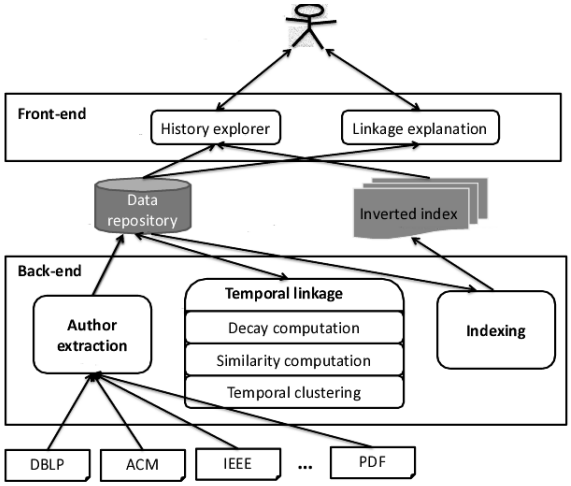


Figure 4.3: Architecture of the CHRONOS system.

selected records, (3) choose to consider decay or not consider decay, and choose to apply different clustering methods, and then compare the results.

**Example 4.1.3** *Consider an advanced user who would like to understand the linkage results further. In particular, she wonders what if the two publications with* Google Inc. *affiliation were published in 1986-1987 instead of 2006-2007. She could choose all publications by the selected "Xin Dong" entity, and then change the time stamp of the two publications.* CHRONOS *then applies linkage at runtime, shows the publication list from the new results and from the original results side-by-side, and highlights the differences. The difference might be that the two publications are considered to belong to another "Xin Dong" because of the big time gap between the two revised records and the rest of the records.* □

## 4.2 System Architecture

We next describe the architecture of the system and also the key techniques for linking temporal records.

### 4.2.1 Architecture

Figure 4.3 depicts the architecture of CHRONOS. At the back end CHRONOS contains three components in charge of data collection and cleaning: **Author extraction, Tem-**

**poral linkage** and **Indexing**. At the front end CHRONOS contains two components in charge of interaction with users, search, and decision explanation: **History explorer**, and **Linkage explanation**. We next describe each component in more detail.

**Author extraction:** This component takes the *DBLP* data as input. For each paper, it extracts records about authors, including author name, paper title, conference, co-author, publication year, and so on. It then follows the links provided by *DBLP* to external sources (*e.g.*, *ACM, IEEE, Scopus*, journal websites, and PDF paper files) and enriches the records by extracting information on affiliation and email of the author at the time of publication. It stores the results in the **Data repository**, which hosts a database using *MySQL*[1].

**Temporal linkage:** This component identifies author records that refer to the same real-world person. We describe this component in more detail shortly. Note that linkage on the full data set can take hours, but this is performed offline and the results are also stored in **Data repository**, ready for online search.

**Indexing:** This component builds an **Inverted index** for each identified real-world author. To facilitate search by name, affiliation, and time period, each author is indexed by her names and affiliations over time, and also the years of her publications. We used *Lucene*[2] for indexing.

**History explorer:** This component is the interface through which the user interacts with the system. It offers (1) author search by name, time period, and affiliation, (2) history tracing for each author, and (3) statistics view of the data. Upon receiving an author query, it finds relevant authors through the **Inverted index**, and then retrieves details about the author from the **Data repository**.

**Linkage explanation:** This component explains linkage decisions and is in charge of three tasks. First, it shows the comparison of results from CHRONOS, from *DBLP*, and from BASIC. For each selected author, it chooses the cluster from *DBLP* or BASIC with the largest number of publications in the author publication list generated by CHRONOS. Second, it explains the decision of a particular paper included in or excluded from the list of papers for a particular author. Explanations are generated mainly according to the decay, as we define shortly. Third, it performs online tem-

[1]http://www.mysql.com.
[2]http://lucene.apache.org.

poral linkage and answers "what-if" questions. Note that since online linkage will be performed only on a small subset of records, it is quite efficient and takes only a few seconds.

### 4.2.2 Temporal Linkage

The core of the system is the **Temporal linkage** component, which links author records that refer to the same real-world entity. It contains three sub-components: **Decay computation**, **Similarity computation**, and **Temporal clustering**. We next briefly describe the techniques applied in each sub-component, and refer the interested readers to Li et al. [2011] for details.

**Decay computation:** One key idea of our temporal linkage algorithm is to apply time decay, which aims to capture the effect of time elapse on entity value evolution. Specifically, we define *disagreement decay* as the probability that an entity changes its value of a particular attribute within a particular period of time. Symmetrically, we define *agreement decay* as the probability that two entities share a common value of a particular attribute within a particular period of time. For example, a disagreement decay of .6 for affiliation and 5 years means that the probability that an author changes her affiliation within 5 years is .6; an agreement of .1 for affiliation and 5 years means that the probability that two different authors share the same affiliation within 5 years is only .1. With the use of decay, we do not penalize variety of values over a long time too much, and meanwhile do not reward similarity of values over a long time too much. We learn decay for each attribute from a set of labeled data, for which we know if two records refer to the same entity and if two strings represent the same value.

**Similarity computation:** We compare a record with a cluster of records considering the following two aspects. First, we consider *value consistency*. We compare the record with the cluster on each attribute, and then take a linear combination of the similarities. In this computation we apply decay, so we are more tolerant to value variety over time. For example, the record of "Xin Dong" from "AT&T" in 2008 has high value consistency with the cluster of "Xin Dong" records from "University of Washington" in 2003-2007, despite the affiliation difference. Second, we consider *continuity*. We compare the time stamp of the record with the time period of the cluster. The higher the continuity, the more likely that the record belongs to the cluster. Our

previous example also observes a high continuity, but another record of "Xin Dong" from "RPI" in 1991 has a low continuity with that cluster. The final similarity combines value consistency and continuity.

**Temporal clustering:** Another key idea of our temporal linkage algorithm is record clustering with a global view of the data. We consider author records in time order and accumulate evidence over time to enable global decision making. Our clustering algorithm proceeds iteratively. In each round, it computes the probability that a record belongs to each cluster according to the record-cluster similarity, and chooses the clustering with the highest probability. It then refines the results iteratively until the results converge.

## 4.3 Related Work

There have been several applications for exploring temporal information. BIBNETMINER Sun et al. [2008] is the closest to CHRONOS: it collects data from *DBLP* and allows users to explore the history of authors on a time line. However, it takes the linkage results from *DBLP* directly while we focus on enriching and improving DBLP data by applying temporal linkage. INZEIT Setty et al. [2010] collects data from New York Times Annotated Corpus and focuses on determining insightful time points as milestones for user queries. PRIMA Moon et al. [2009] considers historical data with evolving schemas. None of the systems emphasizes linkage of temporal records.

There have been other systems related to bibliography data.
DBLIFE DeRose et al. [2007] is able to track entities over time, but it applies handcrafted information extraction rules and the entity resolution methods rely heavily on domain knowledge. WINACS Weninger et al. [2011] extracts data from Web-based information network to perform entity resolution and disambiguation. Again, none of them applies temporal linkage techniques.

Finally, there have been a few demonstrations on record linkage. LINKDB Ioannou et al. [2011] demonstrates the performance and visualization of probabilistic record linkage techniques. SEMEX Cai et al. [2005] performs reference reconciliation on personal data. We differ in that we consider linkage of *temporal* records.

## 4.4 Summary

This chapter aims to exhibit the strength of temporal information in information search and exploration. CHRONOS allows users to search for authors by their name, affiliation, and time period; it also allows users to trace the history and statistics of various aspects of an author, a conference, all publications, and so on. The core of the system is a temporal linkage algorithm that is tolerant to value variety over time when identifying records that refer to the same real-world entity. CHRONOS helps users understand its linkage results by comparison with results of other methods, explanation of the differences, and answering "what-if" questions.

# Chapter 5

# Conclusions

Due to heterogeneous schemas, possible errors in data sets and faulty update processes, traditional record linkage techniques may fall short for many cases. Tolerance to value diversity in the data thus needs to be considered to improve the results for linkage. In this dissertation we have described two problems where value diversity are taken into consideration: temporal linkage, where we need to be tolerant to value diversity over time, and group linkage, where we need to be tolerant to different local values within groups.

This dissertation makes the following contributions for temporal linkage.

- We apply *time decay*, which aims to capture the effect of time elapse on entity value evolution. 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.

- We describe three temporal clustering methods that consider records in time order and accumulate evidence over time to enable global decision making. 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.

- 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.

We make the following contributions for group linkage:

- We study core generation in presence of erroneous data. Our core is robust in the sense that even if we remove a few possibly erroneous records, we still have strong evidence that the rest of the records must belong to the same group. We propose efficient algorithm for core generation.
- We then reduce the group linkage problem into clustering cores and remaining records. Our clustering algorithm leverages strong evidence collected from cores and meanwhile is tolerant to value variety of records in the same group.
- We conducted experiments on two real-world data sets in different domains, showing high efficiency and effectiveness of our algorithms.

There are still continuing challenges in linking records with value diversity, including combining information of multiple dimensions for linkage, such as combining temporal information with spatial information to achieve better results, and to enhance record linkage with information in more complicated frameworks, such as information from crowdsourcing.

# Proofs from Chapter 3

## .1 Proofs in Section 3.2.2

**Proposition .1.1** *Let* $\mathbf{R}$ *be a set of business listings. Denote by* $n(r)$ *the number of values on dominant-value attributes from* $r \in \mathbf{R}$*. Let* $n = \sum_{r \in \mathbf{R}} n(r)$ *and* $m = \max_{r \in \mathbf{R}} n(r)$*. Let* $s$ *be the maximum v-clique size. Algorithm* GRAPHCONSTRUCTION *(1) runs in time* $O(ns(m+s))$*, (2) requires space* $O(n)$*, and (3) its result is independent of the order in which we consider the records.* □

**Proof .1.2** *We first prove that* GRAPHCONSTRUCTION *runs in time* $O(ns(m+s))$*. Step 2 of the algorithm takes in time* $O(nsm)$*, where it takes in time* $O(ns)$ *to scan all records for a dominant-value attribute, and a record can be scanned maximally* $m$ *times. Step 3 takes in time* $O(ns^2)$*. Thus, the algorithm runs in time* $O(ns(m+s))$*.*

*We next prove that* GRAPHCONSTRUCTION *requires space* $O(n)$*. For each value* $v$ *of a dominate-value attribute, the algorithm keeps three data sets:* $\bar{L}$ *that takes in space* $O(n)$*,* $\mathbf{C_v}$ *and* $\bar{T}$ *that require space in total no greater than* $O(|\mathbf{R}|)$*. Since* $O(n) \geq O(|\mathbf{R}|)$*, the algorithm requires space* $O(n)$*.*

*We now prove that the result of* GRAPHCONSTRUCTION *is order independent. Given* $\bar{L}$ *and* $\bar{R}_v$*, Step 2 scan* $\bar{L}$ *and apply transitive rule to merge clusters of records in* $C \cap \bar{R}_v$*, for each v-clique* $C \in \bar{L}$*. The process is independent from the order in*

*which we consider the records in $\bar{R}_v$. The order independence of the result in Step 3 is proven in Benjelloun et al. [2009]. Therefore, the final result is independent from the order in which we consider the records.*

**Proposition .1.3** *Let $G$ be a similarity graph. Let $r$ and $r'$ be two nodes in $G$ that belong to the same v-clique $C$ and not any other v-clique. Then, $r$ and $r'$ must belong to the same partition in any maximal $k$-robust partitioning.* □

**Proof .1.4** *We prove that there does not exist such a maximal $k$-robust partitioning where $r$ and $r'$ are in different partitions. Suppose $r$ and $r'$ belong to subgraphs $P$ and $P'$ respectively in a maximal $k$-robust partitioning, where $r, r'$ are connected to $n, n'$ nodes in $P, P'$ respectively. $P$ and $P'$ are connected by at least $m$ nodes , $m = \min\{n+1, n'+1\}$. We next prove that $P \cup P'$ is $k$-robust.*

*If $P, P'$ are both v-cliques, $P \cup P' \subseteq C$ is also a v-clique and $k$-robust.*

*We next consider the case where one of the partitions is not a v-clique. Suppose $P$ is not a v-clique, each node in $P$ is connected to more than $k$ nodes in $P$, where $k < n$. If $P'$ is a v-clique, each node in $P' \subseteq C$ is connected to at least $n+1$ nodes in $P$, thus $P \cup P'$ is $k$-robust. If $P'$ is not a v-clique, we have $k < \min\{n, n'\} < m$, thus $P \cup P'$ is also $k$-robust.*

*Since the fact that $P \cup P'$ is $k$-robust violates Condition 3 in Definition 3.2.2, it proves that $r, r'$ must belong to the same partition in any maximal $k$-robust partitioning.*

## .2 Proofs in Section 3.2.3

**Theorem .2.1 ($(K+1)$-connected condition)** *Let $G$ be a graph consisting of a union $Q$ of v-cliques. If for every pair of v-cliques $C, C' \in Q$, there is a path of v-cliques between $C$ and $C'$ and every pair of adjacent v-cliques on the path share at least $k+1$ nodes, graph $G$ is $k$-robust.* □

**Proof .2.2** *Given Menger's Theorem Bruhn et al. [2005], graph $G$ is $k$-robust if for any pair of nodes $r, r'$ in $G$, there exists at least $k+1$ independent paths that do not share any nodes other than $r, r'$ in $G$. We now prove that for any pair of nodes $r, r'$ in graph $G$ that satisfies $(k+1)$-connected condition, there exists at least $k+1$ independent paths between $r, r'$. We consider two cases, 1) $r, r'$ are adjacent such that there exists a v-clique in $G$ that contains $r, r'$; 2) $r, r'$ are not adjacent such that there exists no v-clique in $G$ that contains $r, r'$.*

*We first consider Case 1 where there exists a v-clique $C$ containing $r, r'$. Since each v-clique in $G$ has more than $k+1$ nodes, there exist at least $k$ 2-length paths and one 1-length path between $r, r' \in C$. It proves that there exists at least $k+1$ independent paths between $r$ and $r'$.*

*We next consider Case 2 where there exists no v-clique containing $r, r'$ in $G$. Suppose $r \in C, r' \in C'$, where $C, C'$ are different v-cliques in $G$. Since there exists a path of v-cliques between $C$ and $C'$ where every pair of adjacent v-cliques in the path share at least $k+1$ nodes, there exists at least $k+1$ independent paths between $r$ and $r'$.*

*Given the above two cases, we have that there exist at least $k+1$ independent paths between every pair of nodes in $G$, therefore $G$ is $k$-robust.*

**Theorem .2.3 ($(K+1)$-overlap condition)** *Graph $G$ is $k$-robust only if for every $(k+$*

$1$)-*connected v-union* $Q \in G$, $Q$ *shares at least* $k+1$ *common nodes with the subgraph consisting of the rest of the v-unions.* □

**Proof .2.4** *We prove that if graph* $G$ *contains a* $(k+1)$*-connected v-union* $Q$ *that shares at most* $k$ *common nodes with the rest of the graph,* $G$ *is not* $k$*-robust. Since* $Q$ *shares at most* $k$ *common nodes with the subgraph consisting of the rest of the v-unions, removing the common nodes will disconnect* $Q$ *from* $G$*, it proves that* $G$ *is not* $k$*-robust. Thus,* $(k+1)$*-overlap condition holds.*

**Proposition .2.5** *Denote by* $|\bar{L}|$ *the number of entries in input* $\bar{L}$*. Let* $m$ *be the maximum number of values from dominant-value attributes of a record, and* $a$ *be the maximum number of adjacent v-unions that a v-union has. Algorithm* SCREEN *takes time* $O((m^2+a)\cdot|\bar{L}|)$ *and the result is independent of the order in which we examine the v-cliques.* □

**Proof .2.6** *We first prove the time complexity of* SCREEN*. It takes in time* $O(m^2|\bar{L}|)$ *to scan all entries in* $\bar{L}$ *and find common nodes between each pair of adjacent v-cliques (Step 3(a)). It takes in time* $O(a|\mathbf{C}|)$ *to merge v-unions, where* $|\mathbf{C}|$ *is the number of v-cliques in* $G$ *(Step 3(b)). Since* $|\mathbf{C}| < |\bar{L}|$*, the algorithm runs in time* $O(m^2+a)\cdot|\bar{L}|$*.*

*We next prove that the result of* $Screen$ *is independent of the order in which we examine the v-cliques, that is, 1) finding all maximal* $(k+1)$*-connected v-unions in* $G$ *is order independent; 2) removing all nodes in* $\bar{B}(Q)$ *from* $G$ *where* $|barB(Q)| \leq k$ *is order independent.*

*Consider order independency of finding all v-unions in* $G$*. To find all v-unions in* $G$ *is conceptually equivalent to find all connected components in an abstract graph* $G_A$*, where each node in* $G_A$ *is a v-clique in* $G$ *and two nodes in* $G_A$ *are connected if the two corresponding v-cliques share more than* $k$ *nodes.* SCREEN *checks whether each node*

*in $G$ is a common node between two v-cliques (Step 3(a)), and if two cliques share more than $k$ nodes, merges their v-unions (Step 3(b)), which is equivalent to connect two nodes in $G_A$. Once all nodes in $G$ is scanned, all edges in $G_A$ are added, and the order in which we examine nodes in $G$ is independent from the structure of $G_A$ and the connected components in $G_A$. Therefore, finding all v-unions in $G$ is order independent.*

*Consider order independency of removing nodes in $G$. Suppose $Q_1, Q_2, ..., Q_m, m > 0$ are all v-unions in $G$ with $|\bar{B}(Q_i)| \leq k, i \in [1, m]$. Since $G$ is finite, $Q_i$ is finite and unique; thus, removing all nodes in $\bar{B}(Q))$ from $G$ where $|\bar{B}(Q)| \leq k$ is order independent.*

## .3 Proofs in Section 3.2.3.2

**Lemma .3.1** *The max flow from source $a$ to sink $b$ in $G'(V', E')$ is equivalent to $\kappa(a, b)$ in $G(V, E)$.* □

**Proof .3.2** *According to Menger's Theorem Bruhn et al. [2005], the minimum number of nodes whose removal disconnects $a$ and $b$, that is $\kappa(a, b)$, is equal to the maximum number of independent paths between $a$ and $b$. The authors in Even and Tarjan [1975] proves that the maximum number of independent paths between $a$ and $b$ in an undirected graph $G(V, E)$ is equivalent to the maximal value of flow from $a$ to $b$ or the minimal capacity of an $a - b$ cut, the set of nodes such that any path from $a$ to $b$ contains a member of the cut, in $G'(V', E')$.*

**Theorem .3.3 (Max-flow condition)** *Let $G(V, E)$ be an input similarity graph. Graph $G$ is $k$-robust if and only if for every pair of adjacent $(k + 1)$-connected v-unions $Q$*

*and $Q'$, there exist two nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$ such that the max flow from $a$ to $b$ in the corresponding flow network is at least $k+1$.* □

**Proof .3.4** *According to Menger's Theorem Bruhn et al. [2005], $\kappa(a,b)$ in $G$ is equivalent to the max-flow from $a$ to $b$ in the corresponding flow network. We need to prove that graph $G$ is $k$-robust if and only if for each pair of adjacent $(k+1)$-connected v-unions $Q$ and $Q'$, there exists two nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$ such that $\kappa(a,b) \geq k+1$.*

*We first prove that if $G$ is $k$-robust, for each pair of adjacent $(k+1)$-connected v-unions $Q$ and $Q'$, there exists two nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$ such that $\kappa(a,b) \geq k+1$. Since $G$ is $k$-robust, for each pair of nodes $a$ and $b$ in $G$, we have $\kappa(a,b) \geq k+1$.*

*We next prove that if $G$ is not $k$-robust, there exists a pair of adjacent $(k+1)$-connected v-unions $Q$ and $Q'$ such that for each pair of nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$, we have $\kappa(a,b) < k+1$. Since $G$ is not $k$-robust, there exists a separator $\bar{S}$, a set of nodes in $G$ with size no greater than $k$ whose removal disconnects $G$ into two subgraphs $\bar{X}$ and $\bar{Y}$. Suppose $Q$ and $Q'$ are two v-unions in $G$ such that $Q \subseteq \bar{X}, Q' \subseteq \bar{Y}$ and $Q \cap Q' \neq \emptyset$. For each pair of nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$, we have $a \in \bar{X}$ and $b \in \bar{Y}$, and removing the set of nodes in $\bar{S}$ disconnects $a$ and $b$; thus $\kappa(a,b) < k+1$.*

*The above two cases proves that graph $G$ is $k$-robust if and only if for every pair of adjacent $(k+1)$-connected v-unions $Q$ and $Q'$, there exist two nodes $a \in Q \setminus Q'$ and $b \in Q' \setminus Q$ such that $\kappa(a,b) \geq k+1$, i.e. the max flow from $a$ to $b$ in the corresponding flow network is at least $k+1$.*

**Proposition .3.5** *Let $p$ be the total number of pairs of adjacent v-unions, and $g$ be the number of nodes in the input graph. Algorithm* SPLIT *runs in time $O(pg^{2.5})$.* □

**Proof .3.6** *Authors in Even and Tarjan [1975] proves that it takes in time $O(g^{2.5})$ to compute $\kappa(a, b)$ for a pair of nodes $a$ and $b$ in $G$. In the worst case* SPLIT *needs to compute $\kappa(a, b)$ for $p$ pairs of adjacent v-unions. Thus,* SPLIT *runs in time $O(pg^{2.5})$.*

# .4 Proofs in Section 3.2.3.3

**Lemma .4.1** *For each pair of adjacent nodes $r, r'$ in graph $G$, there exists a maximal $k$-robust partitioning such that $r, r'$ are in the same subgraph.* □

**Proof .4.2** *For each pair of adjacent nodes $r, r'$ in $G$, we prove the existence of such a maximal $k$-robust partitioning by constructing it.*

*By definition, adjacent node $r, r'$ form a v-clique $C$. Therefore, there exists a maximal v-clique $C'$ in $G$ that contains $r, r'$, i.e., $C \subseteq C'$. V-clique $C'$ can be obtained by keep adding nodes in $G$ to $C$ so that each newly-added node is adjacent to each node in current clique until no nodes in $G$ can be added to $C'$. By definition, any v-clique is $k$-robust, therefore there exists a maximal $k$-robust sub-graph $G'$ in $G$ such that $C' \subseteq G'$. Graph $G'$ can be obtained by keep adding nodes in $G$ to $C'$ so that each newly-added node is adjacent to at least $k+1$ nodes in current graph $G'$ until no nodes in $G$ can be added to $G'$. We remove $G'$ from $G$ and take $G'$ as a subgraph in the desired partitioning.*

*We repeat the above process to a randomly-selected pair of adjacent nodes in the remaining graph $G \setminus G'$ until it is empty. The desired partitioning satisfies Condition 1 and 2 of Definition 3.2.2 because the above process makes sure each subgraph is exclusive and $k$-robust; it satisfies Condition 3 of Definition 3.2.2 because the above process makes sure each subgraph is maximal, which means merging arbitrary number of subgraphs in the partitioning would violate Condition 2.*

*In summary, the desired partitioning is a maximal $k$-robust partitioning. It proves that for each pair of adjacent nodes $r$ and $r'$ in graph $G$, there exists a maximal $k$-robust partitioning such that $r$ and $r'$ are in the same subgraph.*

**Lemma .4.3** *The set of nodes in a separator $\bar{S}$ of graph $G$ does not belong to any core in $G$, where $|\bar{S}| \leq k$.* □

**Proof .4.4 (Lemma .4.3)** *Suppose the set $\bar{S}$ of nodes separate $G$ into $m$ disconnected sets $\bar{X}_i, i \in [1, m], m > 0$. To prove that each node $r \in \bar{S}$ does not belong to any core in $G$, we prove that for a node $r' \in G, r' \neq r$, there exists a maximal $k$-robust partitioning such that $r$ and $r'$ are separated. Node $r'$ falls into the following cases: 1) $r' \in \bar{X}_i, i \in [1, m]$ ; 2) $r' \in \bar{S}$.*

*Consider Case 1) where $r' \in \bar{X}_i, i \in [1, m]$. We construct a maximal $k$-robust partitioning of $G$ where $r$ and $r'$ are in different subgraphs. We start with a maximal $k$-robust subgraph $G'$ in $G$ that contains $r$ and $r''$ where $r''$ is adjacent to $r$ and in $\bar{X}_j, j \neq i, j \in [1, m]$, and find other maximal $k$-robust subgraphs as in Lemma .4.1. Since $\bar{S}$ separates $\bar{X}_i$ and $\bar{X}_j$, maximal $k$-robust subgraph $G'$ that contains $r$ and $r''$ does not contain any node in $\bar{X}_i$. It proves that there exists a maximal $k$-robust partitioning of $G$ where $r$ and $r'$ are not in the same subgraph.*

*Consider Case 2) where $r' \in \bar{S}$. We construct a maximal $k$-robust partitioning of $G$ such that $r$ and $r'$ are in different subgraphs. We create two maximal $k$-robust subgraphs $G'$ and $G''$, where $G'$ contains $r$ and an adjacent node $r_i \in \bar{X}_i, i \in [1, m]$, $G''$ contains $r'$ and an adjacent node $r_j \in \bar{X}_j, j \neq i, j \in [1, m]$. We create other subgraphs as in Lemma .4.1. Since each path between $r_i \in \bar{X}_i$ and $r_j \in \bar{X}_j$ contains at least one node in $\bar{S}$ and $|\bar{S}| \leq k$, graph $G' \cup G''$ is not $k$-robust. Therefore, the created partitioning is a maximal $k$-robust partitioning. It proves that there exists a*

*maximal $k$-robust partitioning of $G$ where $r$ and $r'$ are not in the same subgraph.*

*Given the above two cases, we have that any node in separator $\bar{S}$ of $G$ does not belong to any core in $G$, where $|\bar{S}| \leq k$.*

**Proposition .4.5** *Let $G$ be the input graph and $q$ be the number of $(k+1)$-connected v-unions in $G$. Define $a, p, g, m$, and $|\bar{L}|$ as in Proposition .2.5 and .3.5. Algorithm* CORE *finds correct cores of $G$ in time $O(q((m^2+a)|\bar{L}|+pg^{2.5}))$ and is order independent.*□

**Proof .4.6** *We first prove that* CORE *correctly finds cores in $G$, that is 1) nodes not returned by* CORE *do not belong to any core; 2) each subgraph returned by* CORE *forms a core.*

*We prove that nodes not returned by* CORE *do not belong to any core in $G$. Nodes not returned by* CORE *belong to separators of subgraphs in $G$. Suppose $\bar{S}$ is a separator of graph $G_n \in \mathbf{Q}$ found in either* SCREEN *or* SPLIT *phase, where $G_n \subseteq G, n \geq 0, G_0 = G$, and $\bar{S}$ separates $G_n$ into $m$ sub-graphs $\bar{X}_n^i, i \in [1,m], m > 1$. Graph $G_n^i \in \mathbf{Q}$ is a subgraph of $G_n$ such that any node $r \in \bar{X}_n^j, j \in [1,m], j \neq i$ does not belong to $G_n^i$. Nodes removed in $G_n^i$ by* CORE *belong to separator $\bar{S}$ in $G_n$. Given Lema .4.3, such nodes do not belong to any core in $G_n$ and thus does not belong to any core in $G$.*

*We next prove that each subgraph returned by* CORE *forms a core in $G$. We prove two cases: 1) subgraph $G'$ in $G$ forms a core if there exists a separator $\bar{S}$ that disconnects $G'$ from $G$, where $|\bar{S}| \leq k$ and $G' \cup \bar{S}$ and $G'$ are both $k$-robust; 2) if a subgraph is a core in $G_n^i$, it is a core in graph $G_n$.*

*We consider Case 1) that subgraph $G'$ in $G$ forms a core if there exists a separator $\bar{S}$ that disconnects $G'$ from $G$, where $|\bar{S}| \leq k$ and $G' \cup \bar{S}$ and $G'$ are both $k$-robust. For a pair of nodes $r_1, r_2$ in $G'$, we prove that there exists no maximal $k$-robust partitioning*

*where $r_1$ and $r_2$ are in different subgraphs. Suppose such a partitioning exists, and $G_1, G_2$ are subgraphs containing $r_1, r_2$ respectively. Since $G_1, G_2 \subseteq G' \cup \bar{S}$, we have that $G_1 \cup G_2$ is $k$-robust, it violates the fact that the result of merging any two subgraphs in a maximal $k$-robust partitioning is not $k$-robust. Therefore, there exists no maximal $k$-robust partitioning where $r_1$ and $r_2$ are in different subgraphs. It proves that $G'$ is a core in $G$.*

*We next consider Case 2) that if a subgraph $G'$ is a core in $G_n^i$, it is a core in graph $G_n$. We prove that a pair of nodes $r_1, r_2 \in G'$ belong to the same subgraph of all maximal $k$-robust partitioning in $G_n$. Suppose there exists such a partitioning of $G_n$ where $r_1 \in G_1, r_2 \in G_2$. Since $G_n^i \subseteq \bar{X}_n^i \cup \bar{S}$, we have $G_1, G_2 \subseteq G_n^i$, otherwise $G_1, G_2$ are not $k$-robust. Since $r_1, r_2$ belong to the same core in $G_n^i$, we have $G_1 = G_2$. It proves that if $G'$ is a core in $G_n^i$, it is a core in $G_n$.*

*The above two cases prove that each subgraph returned by* CORE *forms a core in $G$. In summary, nodes not returned by* CORE *do not belong to any core, and each subgraph returned by* CORE *forms a core in $G$. Thus,* CORE *correctly finds all cores in $G$. It further proves that the result of* CORE *is independent from the order in which we find and remove separators of graphs in* $\mathbf{Q}$.

*We now analyze the time complexity of* CORE. *For each $(k+1)$-connected v-unions in $G$, it takes in time $O(m^2 + a)|\bar{L}|$ to proceed* SCREEN *phase and in time $O(pg^{2.5})$ to proceed* SPLIT *phase. In total there are $q$ v-unions in $G$, thus the algorithm takes in time $O(q((m^2 + a)|\bar{L}| + pg^{2.5}))$.*

## .5 Proof in Section 3.3

**Proposition .5.1** *Let $l$ be the number of distinct candidate pairs ever in $Que$ and $|\mathbf{E}|$ be the number of input elements. Algorithm* CLUSTER *takes time $O(l \cdot |\mathbf{E}|^2)$.* □

**Proof .5.2** *It takes time $O(|\mathbf{E}|^2)$ to initialize clustering $\mathcal{C}$ and list $Que$. It takes $|\mathbf{E}|^2$ to check each distinct candidate pair in $Que$, where it takes $O(|\mathbf{E}|)$ to examine all valid clustering plans and select the one with highest SV-index (Step 2(a)), and it takes $O(|\mathbf{E}|^2)$ to recompute SV-index for all relevant elements and update $Que$ (Step 2(b)). In total there are $l$ distinct candidate pairs ever in $Que$, thus* CLUSTER *takes time $O(l \cdot |\mathbf{E}|^2)$.*

# References


Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB '02*, pages 586–597. 2

Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002. 49

Nilesh Bansal, Fei Chiang, Nick Koudas, and Frank Wm. Tompa. Seeking stable clusters in the blogosphere. In *VLDB*, pages 806–817, 2007. 89

Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1):255–276, 2009. 4, 8, 88, 104

Indrajit Bhattacharya and Lise Getoor. Iterative record linkage for cleaning and integration. In *DMKD '04*, pages 11–18. 2

Henning Bruhn, Reinhard Diestel, and Maya Stein. Menger's theorem for infinite graphs with ends. *J. Graph Theory*, 50(3):199–211, 2005. 105, 107, 108

Douglas Burdick, Mauricio A. Hernández, Howard Ho, Georgia Koutrika, Rajasekar Krishnamurthy, Lucian Popa, Ioana Stanoi, Shivakumar Vaithyanathan, and Sanjiv R. Das. Extracting, linking and integrating data from public sources: A financial case study. *IEEE Data Engineering*, 34(3):60–67, 2011. 50

Yuhan Cai, Xin Luna Dong, Alon Halevy, Jing Michelle Liu, and Jayant Madhavan. Personal information management with SEMEX. In *SIGMOD*, pages 921–923, 2005. 99

Zhaoqi Chen, Dmitri V. Kalashnikov, and Sharad Mehrotra. Exploiting relationships for object consolidation. In *IQIS*, pages 47–58, 2005. 49

E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. In *PODS*, pages 223–233, 2003. 50

William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. of IIWEB*, pages 73–78, 2003. 80

G. Cormode, V. Shkapenyuk, D. Srivastava, and B. Xu. Forward decay: A practical time decay model for streaming systems. *ICDE*, pages 138–149, 2009. 50

Pedro DeRose, Warren Shen, Fei Chen, Yoonkyong Lee, Douglas Burdick, AnHai Doan, and Raghu Ramakrishnan. DBLife: A community information management platform for the database research community. In *CIDR*, pages 169–172, 2007. 99

Debabrata Dey. Entity matching in heterogeneous databases: A logistic regression approach. *Decis. Support Syst.*, 44:740–747, 2008. 49, 88

Pedro Domingos. Multi-relational record linkage. In *In Proceedings of the KDD Workshop on Multi-Relational Data Mining*, 2004. 49

Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD '05*, pages 85–96. 2

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007. 1, 49, 88

Shimon Even and Robert Endre Tarjan. Network flow and testing graph connectivity. *SIAM J. Comput.*, 4(4):507–518, 1975. 107, 109

W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. In *Proceedings of the 30th Symposium on Principles of Database Systems of Data*, pages 71–82. ACM, 2011. 50, 51

Wenfei Fan, Xibei Jia, Jianzhong Li, and Shuai Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009. 89

Ivan P. Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969a. 49

Ivan P. Fellegi and Alan B. Sunter. A theory for record linkage. *Journal of the Americal Statistical Association*, 64(328):1183–1210, 1969b. 2, 88

G. Flake, R. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1:385–408, 2004. 50, 88

L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962. 68

T.F. Gonzalez. On the computational complexity of clustering and related problems. *Lecture Notes in Control and Information Sciences*, page 174182, 1982. 75

Songtao Guo, Xin Dong, Divesh Srivastava, and Remi Zajac. Record linkage with uniqueness constraints and erroneous values. *PVLDB*, 3(1):417–428, 2010. 53, 72, 89

Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Rene J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, pages 1282–1293, 2009a. 40, 50, 81

Oktie Hassanzadeh, Fei Chiang, Hyun Chul Lee, and Renée J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009b. 94

Mauricio A. Hernandez and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2:9–37, 1998. 49, 50, 88

Shu Huang. Mixed group discovery: Incorporating group linkage with alternatively consistent social network analysis. *International Conference on Semantic Computing*, 0:369–376, 2010. 89

Ekaterini Ioannou, Wolfgang Nejdl, Claudia Niederée, and Yannis Velegrakis. LinkDB: a probabilistic linkage database system. In *SIGMOD*, pages 1307–1310, 2011. 99

Dmitri V. Kalashnikov, Sharad Mehrotra, and Zhaoqi Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM Data Mining (SDM)*, 2005. 3

Nick Koudas, Sunita Sarawagi, and Divesh Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, 2006. 1, 49, 88

Bjornar Larsen and Chinatsu Aone. Fast and effective text mining using linear-time document clustering. In *KDD*, pages 16–22, 1999. 89

Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011. 12, 75, 89, 92, 98

Pei Li, Christina Tziviskou, Haidong Wang, Xin Luna Dong, Xiaoguang Liu, Andrea Maurino, and Divesh Srivastava. Chronos: facilitating history discovery by linking temporal records. *PVLDB*, 5(12):2006–2009, 2012. 12

Xin Liu, Yihong Gong, Wei Xu, and Shenghuo Zhu. Document clustering with cluster refinement and model selection capabilities. In *SIGIR*, pages 191–198, 2002. 89

Hyun Jin Moon, Carlo Curino, MyungWon Ham, and Carlo Zaniolo. PRIMA: archiving and querying historical data with evolving schemas. In *SIGMOD*, pages 1019–1022, 2009. 99

Byung Won On, N. Koudas, Dongwon Lee, and D. Srivastava. Group linkage. In *ICDE*, pages 496–505, 2007. 49, 89

G. Ozsoyoglu and R.T. Snodgrass. Temporal and real-time databases: a survey. *TKDE*, 7:513–532, 1995. 50

John F. Roddick and Myra Spiliopoulou. A survey of temporal knowledge discovery paradigms and methods. *TKDE*, 14:750–767, 2002. 50

Vinay Setty, Srikanta Bedathur, Klaus Berberich, and Gerhard Weikum. InZeit: efficiently identifying insightful time points. *PVLDB*, 3(2):1605–1608, 2010. 99

J. Sima and S. E. Schaeffer. On the np-completeness of some graph cluster measures. *Lecture Notes in Computer Science*, pages 530–537, 2006. 75

Yizhou Sun, Tianyi Wu, Zhijun Yin, Hong Cheng, Jiawei Han, Xiaoxin Yin, and Peixiang Zhao. BibNetMiner: mining bibliographic information networks. In *SIGMOD*, pages 1341–1344, 2008. 99

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. 4

Tim Weninger, Marina Danilevsky, Fabio Fumarola, Joshua Hailpern, Jiawei Han, Thomas J. Johnston, Surya Kallumadi, Hyungsul Kim, Zhijin Li, David McCloskey, Yizhou Sun, Nathan E. TeGrotenhuis, Chi Wang, and Xiao Yu. WINACS: construction and analysis of web-based computer science information networks. In *SIGMOD*, pages 1255–1258, 2011. 99

Derry Tanti Wijaya and Stephane Bressan. Ricochet: A family of unconstrained algorithms for graph clustering. In *DASFAA*, pages 153–167, 2009. 50, 89

William E. Winkler. Methods for record linkage and bayesian networks. Technical report, U.S. Bureau of the Census, 2002. 49, 88

Mohamed Yakout, Ahmed K. Elmagarmid, Hazem Elmeleegy, Mourad Ouzzani, and Alan Qi. Behavior based record linkage. *PVLDB*, 3:439–448, 2010. 50

Minoru Yoshida, Masaki Ikeda, Shingo Ono, Issei Sato, and Hiroshi Nakagawa. Person name disambiguation by bootstrapping. In *SIGMIR*, pages 10–17, 2010. 3, 89