University of Alberta

Entity Resolution for Large Relational Datasets

by

Zhaochen Guo

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Spring 2010
Edmonton, Alberta

Examining Committee

Denilson Barbosa, Computing Science

Mario Nascimento, Computing Science

Marek Reformat, Electrical and Computer Engineering

Abstract

As the volume of data on the Web or in databases increases, data integration is becoming more expensive and challenging than ever before. One of the challenges is entity resolution when integrating data from different sources. References with different representations but referring to the same underlying entity need to be resolved. And, references with similar descriptions but referring to different entities need to be distinguished from one another. Correctly de-duplicating and disambiguating these entities is an essential task in preparing high quality data. Traditional approaches mainly focus on the attribute similarity of references, but they do not always work for datasets with insufficient information. However, in relational datasets like social networks, references are always associated with one or more relationships and these relationships can provide additional information for identifying duplicates.

In this thesis, we solve the entity resolution problem by using relationships in the relational datasets. We implement a relational entity resolution algorithm to resolve entities based on an existing algorithm, greatly improving its efficiency and performance. Also, we generalize the single-type entity resolution algorithm to a multi-type entity resolution algorithm for applications that require to resolve multiple types of reference simultaneously and demonstrate its advantage over the single-type entity resolution algorithm. To improve the efficiency of the entity resolution process, we implement two blocking approaches to reduce the number of redundant comparisons performed by other methods. In addition, we implement a disk-based clustering algorithm that addresses the scalability problem, and apply it on a large academic social network dataset.

Acknowledgements

First and foremost, I would like to sincerely thank my advisor Denilson Barbosa for his advice and support. This thesis would not have been possible without him and without the encouragement and patience he has given me over the past two years.

My thanks also go the professors in my thesis committee, Mario Nascimento and Marek Reformat for reading my thesis and providing valuable advice and comments. I would also like to thank Eleni Stroulia, for the collaboration during my graduate studies and everything that followed.

Of course, life would not be the same without the fellows DBers, officemates and friends Filipe Mesquita, Jagoda Walny, Guohua Liu, Xiaomin Zhang, Jichuan Shi, Zhijie Wang, Baochun Bai, Jean Cao and Jia Zeng. I will never forget the happy time with you for discussing projects, homeworks, life or any kinds of activities. I also acknowledge Database Group and the Computing Science department for making it a stimulating and interactive environment. Thanks also go to Edith Drummond and Karen Berg for patiently answering all my questions with the general affairs.

Last, but not least, I would like to thank my wife Man Xu for her understanding and love during the past few years. Her support and encouragement were in the end what made this thesis possible. Also, I owe my deepest gratitude and love to our parents for their dedication and support all the way.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we first give an overview of the data integration and challenges during the integration process in Section 1.1. Section 1.2 introduces the entity resolution problem, various challenges, and approaches that use attributes and relationships for entity resolution. In Section 1.3, we describe the applications that motivate us to propose algorithms for entity resolution. Section 1.4 lists the contributions we have achieved in this thesis, and Section 1.5 defines the terms that are used frequently in the thesis. Finally Section 1.6 gives the organization of this thesis.

1.1 Data Integration

As the volume of data increases, there is an urgent need to intelligently analyze the data and transform the information into useful knowledge. Data mining is one such process that employs intelligent methods to extract hidden knowledge from various datasets. Companies use data mining to analyze the shopping behavior of customers and arrange promotion activities, governments use it to predict the epidemiological trends and make precautions accordingly, and sociologists use it to analyze the behavior of social communities and find social problems and solutions [2, 4, 13].

While data mining is essential and useful in various domains, we have to be aware that the quality of the knowledge mined from datasets is highly dependent on the quality of these datasets. A dataset that is incomplete, noisy and inconsistent may result in conclusions that do not truly reflect the underlying truth and lead to false decisions. So we need to pre-process datasets to create a complete, clean and consistent dataset prior to any data mining tasks.

Very often, the datasets to be analyzed come from multiple heterogeneous sources such as databases, flat files or the Web, and need to be integrated and cleaned first. Data integration is one such process to integrate data from these sources into a clean and coherent dataset. Though the process seems straightforward, a number of issues need to be considered during

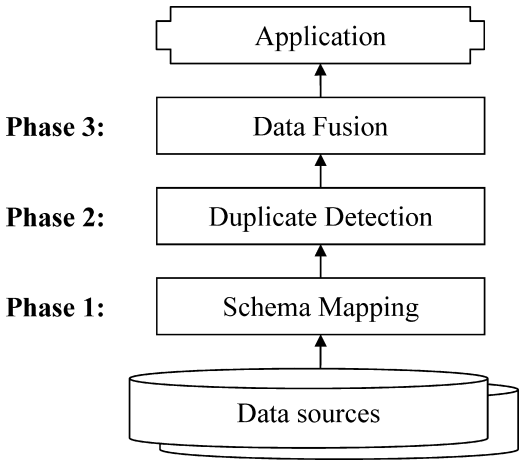

Figure 1.1: A data integration process [11].

the integration process. Bleiholder and Naumann [11] categorize these issues into three phases as shown in Figure 1.1. The first phase is the *schema mapping* which resolves the *schema level inconsistency*. In different datasets, the schemata that describe the same domain may have different names and result in *schema level inconsistency*. For example, *surname* may be used to represent *last name* in one dataset, while *family name* may be used to represent *last name* in another dataset. Also, *state* and *province* may be used to describe addresses in different datasets. The second phase is the *duplicate detection* which aims to resolve the *tuple level inconsistency*. Sometimes, one real object may have several different descriptions in the dataset and this may cause *tuple level inconsistency*. For example, in the bibliographic dataset, an author called *Richard R. Muntz* may have several different representations such as *R. R. Muntz*, *R. Muntz*, or *Richard Muntz*; Simply integrating them without eliminating these duplicates will result in data with redundancy. The third phase is the *data fusion* which aims to solve the *attribute level inconsistency*. When the duplicates of an object are merged, the value of their attributes may be incomplete or conflict with each other.

In this thesis, we mainly focus on issues raised during the *duplicate detection* phase (which we refer to as *entity resolution*). For every dataset, we assume that the schema level inconsistency has already been resolved and all the datasets are formatted using a common format. Also, we are not concerned about the data fusion phase after the entity resolution.

1.2 Entity Resolution

Entity resolution is a process to resolve duplicated representations (references) of real objects (entities). It has received considerable attention and has been addressed as record linkage [52, 68], duplicate detection [51, 59], hardening soft databases [19], the merge/purge

problem [37] and reference matching [49]. The duplication problem arises when different references referring to the same entity exist in the same dataset and are described by similar or different attributes. For example, in personal information management (PIM) systems, a person may have a tuple in one dataset with name, age, home address and other personal information, and a tuple in another dataset with name, company address, and education background. Given these datasets, the task of entity resolution is to map the references to their underlying real entities, and to group them together so that all the references referring to the same entity are put in the same group.

In principle, entity resolution is easy. Any pair of references that have similar attributes may be potential duplicates. The more similar their attributes are, the more likely they are duplicates. Given a similarity measure and a threshold, we compare each pair of references and compute their similarity score using the similarity measure. If the similarity score of this pair is greater than the given threshold, then the two references are considered to be duplicates. Though the basic process is simple, there are several aspects that make entity resolution challenging [31].

Accuracy. The accuracy of entity resolution depends on several factors. First, unique identifiers like Social Security Number (SSN) may not be provided to link references in one dataset with references in other datasets, and references referring to the same entity may have different attributes. Furthermore, references with the same attributes may represent different entities. For example, according to DBLP [1] at the time of writing, there are 16 authors called *Wei Wang* from different institutions. In addition, other factors like typographical errors, use of abbreviations and misuse of data cleaning tools can also make entity resolution more complicated and difficult.

Efficiency. The efficiency of the entity resolution for large datasets is always a concern for applications that require high performance. As discussed, the entity resolution process is a process of pair-wise comparisons of potential duplicates; Since every reference in the dataset may be a potential duplicate of one another, this will result in $O(n^2)$ computational cost and a naive approach that compares every pair is impractical.

Privacy. The privacy of information is strongly connected with the approaches of entity resolution. Through entity resolution, we can collect information that cannot be disclosed from only one dataset [70]. Take the PIM system for example, if an approach can connect a person's personal information with their work-related information, then more information about this person that could not be found before, can be uncovered. So through entity resolution, policies or rules can be proposed to protect the privacy of information by analyzing the approaches of entity resolution.

In this thesis, we only address the issues of accuracy and efficiency, and leave the privacy

[1] `http://www.informatik.uni-trier.de/~ley/db/index.html`

issue for future work.

1.2.1 Entity Resolution Using Attributes

An intuitive way to measure the similarity of two references, is to measure the similarity of their attributes such as names, addresses and so on. Most traditional approaches adapt this measurement and they are referred to as attribute-based approach. The accuracy of these attribute-based approaches is greatly dependent on the similarity measures employed. In this thesis we are only concerned about the generic string similarity measures since the similarity of most non-string attributes like numbers is easy to compute. To measure the string similarity, a number of generic string similarity measures have been proposed for various cases, and many approaches using these measures for entity resolution are developed. We will cover most of them in the related work.

Although attribute-based approaches are commonly used for entity resolution, there are still several problems with them. First, it is difficult to determine an appropriate merging threshold for the similarity measures. Second, for the references with the same attributes but referring to different entities, the attribute-based methods cannot distinguish them from one another.

1.2.2 Entity Resolution Using Relationships

For datasets on which attribute-based approaches can not achieve high accuracy, relationships can be used to further improve the resolution results. In practice, there are many datasets with rich relationships between references. For example, in bibliographic datasets, we have *co-authorship* relationship between collaborators, and *publishedIn* relationship between papers and venues. Also in social networks, each individual may be involved in several different relationships such as *friendship*, *marriage* relationship or *membership*. These relationships can be viewed as additional properties of references in the datasets and can help with entity resolution. Since the relationships are specific to entities, the references referring to the same entity should exhibit some similarity in the relationships they involve. For example, researchers tend to collaborate with researchers they are familiar with, and individuals prefer to join the same groups their friends have joined in. The similarity can be used to help identify and distinguish references.

Given the additional properties (relationships) of references, the relational entity resolution is to seek evidence in relationships to resolve duplicates. In this thesis, we refer to the evidence as relational similarity. Usually, the relational similarity is measured by the number of common entities connected to a pair of potential duplicates. For example, we can compute the relational similarity of two author references by counting the number of coauthors they share. One challenge of measuring this relational similarity is to find out the

shared entities. For example, it is difficult to compare the coauthors just using their names since some authors cannot be uniquely identified by name. We need to know if the shared coauthors are really the same authors rather than different authors with the same name. So the resolution of one pair of references may affect the resolution of other related references. To solve this problem, we need to do the entity resolution collectively and iteratively. By collectively, references can be resolved jointly instead of independently, and the resolving results can be propagated to related references. By iteratively, the propagated information in one iteration can be used to further improve the accuracy of entity resolution in following iterations.

A requirement of many data integration tasks is to resolve multiple types of references in the dataset. Also, references in the dataset are often involved in several relationships with other types of references such as the *publishedIn* relationship between paper references and venue references, and *writtenBy* relationship between paper references and author references.

The result of resolving one type of reference may provide additional information for the resolution of another type of reference. For example, if two paper references are considered to be duplicates, then the venues in which the papers are published can also be considered as the same venue. In this thesis, we generalize the idea of single type relational entity resolution to multi-type relational entity resolution, in which the types of references and the relationships between references are expanded from one type to multiple types. Each resolution result is propagated to other references through the relationships it involves and every pair of references can exploit all their relationships to measure their similarity.

1.3 Motivation

Our motivation for the work in this thesis mainly comes from the ReaSoN (REseArch SOcial Networks) project. Before introducing the motivation, we first give a description of the ReaSoN project.

ReaSoN

ReaSoN is a project that develops an environment to support the analysis of the researchers' social network and provides information to research communities. The ReaSoN social networks consist of two basic networks: the co-authorship network in which nodes are authors and edges are the co-authorships formed through the collaboration on scientific papers, and the citation network in which nodes are papers and edges are the citations between papers. In addition, several other graphs can be derived from the basic networks such as the venue citation network, in which nodes correspond to venues and a link $v_1 \rightarrow v_2$ indicates that venue v_1 has published papers that cite the papers in v_2. Similarly, we can build a graph that reflects the collaboration between research institutions.

P1: "Mapping a Common Geoscientific Object Model to Heterogeneous Spatial Data Repositories", "Silvia Nittel, Jiong Yang(UCLA), Richard R. Muntz", "ACM-GIS", "1996"

P2: "STING: A Statistical Information Grid Approach to Spatial Data Mining", "W. Wang, J. Yang(University of California, Los Angeles), Richard Muntz", "VLDB", "1997"

P3: "AGILE: A General Approach to Detect Transitions in Evolving Data Streams", "Jiong Yang(Case Western Reserve University), Wei Wang(UNC)", "ICDM", "2004"

P4: "A framework for ontology-driven subspace clustering", "Jinze Liu, Wei Wang, Jiong Yang", "KDD", "2004"

P5: "Adaptive contact probing mechanisms for delay tolerant applications", "W. Wang, V. Srinivasan, M. Motani", "Conference on Mobile Computing and Networking", "2007"

P6: "Adaptive contact probing mechanisms for delay tolerant applications", "Wei Wang, Vikram Srinivasan, Mehul Motani", "MOBICOM", "2007"

Figure 1.2: An example from the ReaSoN dataset in the format: <title, authors(affiliation if available), venue, year>.

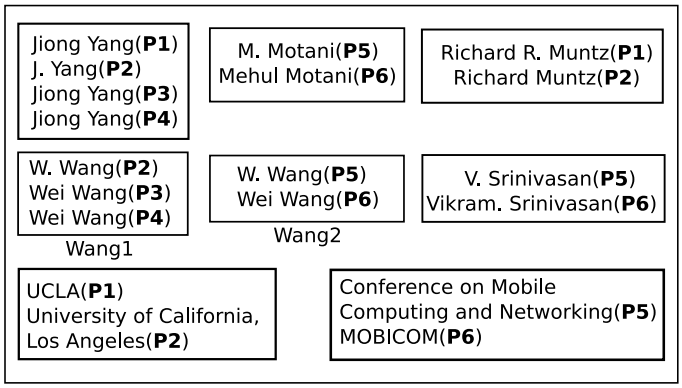

Figure 1.3: The ground truth of the example dataset. All the references referring to the same entity are grouped together.

Building these graphs from ReaSoN dataset seems straightforward, what we need to do is to extract relationships between references, and to use the relationships and the references to construct the graph. However, the task is not as easy as expected for the following reasons.

First, the ReaSoN dataset contains a number of duplicates and these duplicates are difficult to resolve using attribute-based approaches. The ReaSoN dataset is consisted of data from several different sources: DBLP, ACM Digital Library and Google Scholar, and these datasets have a large portion of overlaps on publications in the database community. Thus there will be a number of duplicates for authors, papers and venues.

Consider the example in Figure 1.2 from the ReaSoN dataset. The example dataset contains six paper records from the ReaSoN dataset. Each paper record contains four types of references: paper, author, venue and affiliation, of which the affiliation information is not available for all authors. The ground truth in Figure 1.3 shows that there are duplicates for authors, papers, and venues. In addition, some of the duplicates are difficult to detect. For example, *W. Wang* in *P2* refers to the same author as *Wei Wang* in *P3*; However, from

the name of these two author references, we cannot conclude that they are the same author. And similarly for *W. Wang* in *P5* and *Wei Wang* in *P6*. Moreover, *Wei Wang* in *P3, P4* has the same name as *Wei Wang* in *P6*, but the ground truth tells us that they are different authors. In addition, it is also difficult to determine if *MOBICOM* is the same venue as *Conference on Mobile Computing and Networking* or if *UCLA* refers to the same affiliation as *University of California, Los Angeles*.

Second, we need to resolve entities of multiple types. As introduced, the academic social graphs contain several types of references: author, paper, venue and affiliations, each of which may have duplicates in the dataset. To build a clean graph with high quality data, we need to propose an entity resolution approach for multiple types of entities.

Third, the efficiency of entity resolution may be a challenge for the ReaSoN dataset which contains 35,878,709 references that are too large to be handled by current entity resolution approaches. A more efficient and scalable approach is needed to resolve the entities in the ReaSoN dataset.

1.4 Contribution

Given these requirements and challenges, we build our own entity resolution system for ReaSoN based on an approach proposed by Bhattacharya and Getoor [8], and we make several improvements over their approach. Below are the contributions we make in this thesis.

First, we generalize Bhattacharya and Getoor's approach so that the algorithm can be applied to multiple types of references. Their approach improves the accuracy of entity resolution over attributed-based approaches by combining relationships and attributes of references. However, it resolves only one type of reference (which is the author reference). Also, for each resolution decision, only one relationship is used even though there may be other relationships available between the pair of references. To adapt it to our requirements, we generalize the single-type entity resolution approach to a multi-type resolution approach, and for each pair of references, we consider all the relationships the pair of references is involved in.

Second, we implement two blocking methods to improve the efficiency of the approach using the idea proposed by McCallum et al. [49]. The methods split the whole dataset into small overlapping partitions, each of which contains only references that are similar to each other. Using the partition strategy, the number of redundant comparisons will be greatly reduced and thus the efficiency of the entity resolution process can be improved.

Third, we change the entity resolution process of Bhattacharya and Getoor and get rid of some unnecessary data structures to further improve the space efficiency of their approach. The original approach uses a priority queue to speedup the merging process; However, we

find that the entity resolution can be performed in an iterative way so that the additional information is not needed for the resolution process.

Finally, we propose a record level blocking method to split the whole dataset into small partitions that are both attribute and relational independent, and we convert our multi-type entity resolution algorithm to a disk-based algorithm so that our algorithm is scalable for very large datasets. In addition, we apply our algorithm on the ReaSoN dataset and produce a clean dataset for ReaSoN.

1.5 Terminology

Before moving to the details, we first give definitions to some terminologies that will be used in the rest of the thesis. Unless stated otherwise, all the terms mentioned in this thesis conform to the following definitions.

- **Entity.** An entity is a real-world object that could be observed or perceived in different forms. In the ReaSoN dataset, all the entities we have are the author, paper, venue and organization.
- **Reference.** Reference is a mention or representation of an entity such as the name of a person or organization. An entity may be observed in several representations and each representation is a reference of the entity. For example, in the ground truth of the example dataset, *Wang1* and *Wang2* are the entities, and the names in the rectangles are references of *Wang1* and *Wang2*.
- **Cluster.** A cluster is a logical container or set in which references are labeled with the same class according to some rules. In this thesis, a cluster is a set of references that refer to the same entity. The task of clustering is to group the references of each entity into a separate cluster.
- **Record.** A record is a bibliography entry which contains all the information of a paper such as title, year, authors, venue and others.

1.6 Organization

The remainder of this thesis is organized as follows; We first review most important literature about entity resolution in Chapter 2. Chapter 3 formalizes the entity resolution problem and describes how we generalize the current approach to the multi-type entity resolution approach. In Chapter 4, we describe two blocking methods for improving the efficiency, and an optimized clustering process for improving the space efficiency. Chapter 5 describes datasets for the experimental evaluation and presents the experiment results. Then in Chapter 6, we introduce the record level blocking technique and describe the approach for

large-scale entity resolution. Finally, we conclude in Chapter 7 by summarizing the thesis and presenting the future works that arise from our research.

Chapter 2

Related Work

Entity resolution, also known as record linkage [52, 68] and object reconciliation [25], is a topic that has received considerable attention in both academia and industry recently. A large number of approaches have been proposed and implemented in the statistics and computer science community [68, 69, 56, 38, 8, 5, 25]. In this chapter, we review some of the important studies related to our work which include approximate similarity measures for attribute comparison, various entity resolution algorithms, techniques addressing efficiency, and frameworks or models that integrate different measures to gain better accuracy and so on. A more thorough analysis of literature on entity resolution problem is given by Winkler [69] and Elmagarmid et al. [27]. Also Getoor and Diehl [32] review more studies that explore link relationships in the relational datasets for entity resolution.

2.1 Overview

In 1950s, the problem of record linkage was originated in the "follow-up" statistics of families by Newcombe et al. [52] as the problem of grouping together references of a particular individual or family from one or more data sources based on their attribute similarity. Then Fellegi and Sunter [29] formally developed a mathematical model to formalize the record linkage problem. Thereafter, a large number of approaches have been proposed based on the Fellegi-Sunter model [9, 37, 49, 64].

The goal of entity resolution is to identify references that refer to the same real-world entity from one or more data sources. In principle, the process of entity resolution is simple; It compares potential duplicates using a similarity measure and merges the pairs if their similarity score is greater than a given threshold. In reality, however, the process involves more than matching and merging. A complete entity resolution system consists of at least three components: 1) a standardization component, 2) an entity identification component, and 3) an evaluation component. The standardization component employs some pre-defined rules or domain-dependent knowledge to transform the attributes of references

into an unified format for later comparison. Then the entity identification component uses similarity measures to detect duplicates. Finally, the evaluation component evaluates the performance and accuracy of the resolution step. For each component, a number of problems may arise during the process and a number of approaches have been proposed to address them. We will discuss most of the problems and the corresponding approaches in the following sections.

2.2 Data Standardization

When integrating data from different sources, many issues may cause the inconsistency problem in the dataset. Datasets from different sources are usually collected by different organizations using different methods and tools. The format used in one dataset may not be the format used in other datasets, and these data sources may follow different conventions. For example, the name *John Smith* in one paper may be spelt as *Smith, John* in another paper. These inconsistencies can greatly affect the accuracy of the resolution results. Thus, a data standardization process prior to the resolving process is needed. Here we introduce generic approaches for standardizing names, addresses and so on.

Herzog et al. [38] separate the parsing process from the standardization process and introduce various methods to address the problem of data standardization and parsing. Mainly, they introduce the following methods: the standardization of spellings which replaces spelling variations of words with a common consistent spelling, consistency of coding which converts different representations of attributes into an uniform unit or coding scheme, and integrity checks on attribute values such as checking the month of a year or days of a month. Borkar et al. [12] develop a tool to automatically split unformatted texts into structured segments. Their approach builds a probabilistic model based on Hidden Markov Models (HMM) that incorporates various information of the dataset including the sequence of elements, their length distribution, and other external data dictionaries. Churches et al. [17] and Christen et al. [16] use a combination of lexicon-based tokenization and HMM to standardize the names in medical records and show an improvement of accuracy over the rule-based systems.

In the domain of digital library, several approaches are also proposed to extract citation metadata like author, title, venue and other information from the citation strings. Day et al. [24] adopt an ontological knowledge representation framework for automatic citation extraction which can represent complicated structures and perform matchings like hierarchical matching or regular expressions. Cortez et al. [66] also propose a method using a knowledge base. Their methods differ from others in that they build the knowledge base automatically and extract the metadata without supervision. Also, they exploit the context within the citation string and the context between citations within a paper for better extractions.

2.3 Attribute Similarity Measures

Given a pair of references, most traditional entity resolution approaches employ generic string similarity measures or domain specific measures to compute the attribute similarity and make decisions based on the similarity score. A great number of similarity measures have been proposed for computing string similarity. They are basically divided into three categories [27]: character-level similarity measures, token-level similarity measures, and phoneme-based similarity measures.

2.3.1 Character-Level Similarity Measures

Character-level similarity measures match strings character by character. The *Levenshtein distance* [46] (also known as *Edit Distance*) is one of the most commonly used measures for string similarity. It models the transformation from one string to the other in three operations: insertion, deletion, and substitution. The edit distance between two strings is then given by the minimum number of operations required to transform one string to the other. Waterman et al. [67] generalize the edit distance by allowing multiple deletions and insertions to handle strings that are either truncated or shortened. Furthermore, for strings with mismatches at the beginning and at the end, Smith and Waterman [61] extend the edit distance by assigning lower costs to the characters at the beginning and the end of a string than in the middle because they observe that it is less likely to make mistakes in the middle of a string than at the beginning and the end.

The edit distance and its variants work well for most strings; However, they may not be suitable for strings like names of persons or companies. For example *Martha* and *Marhta* refer to the same person though one name has a spelling error. Using the edit distance measure, their string similarity is only 0.67 because two operations are needed to transform one name to the other and the total length of the name is six. Thus this pair of references is unlikely to be considered as duplicates. To solve this problem, The Jaro distance [39] and its variant the Jaro-Winkler [56] distance were proposed. According to these measures, two characters from string s_1 and s_2 respectively, are considered matching if their distance is within $\lfloor \frac{\max(|s_1|,|s_2|)}{2} \rfloor - 1$. Then *Martha* and *Marhta* are exactly matched. To show the difference between names, the transposition is introduced to measure the transformations between strings. Transposition is defined as half the number of characters that match but in different sequence order. For *Martha* and *Marhta*, the transportation is one because only one switch of "t" and "h" is needed to transform one string to the other, and the final Jaro-Winkler distance is 0.94, which can much better reflect the true similarity.

For strings with typographical errors, the N-gram [65] is another technique from the natural language processing area. The strings to be compared are first converted into a set of tokens (N-grams), each of which has N characters; Then the similarity can be measured

by comparing the two set of N-grams.

2.3.2 Token-Level Similarity Measures

Token-level similarity measures focus on the common terms shared by strings instead of characters. In the information retrieval area, token-level similarity measures are heavily used by search engines for comparing documents or retrieving documents for given queries. One measure is the *cosine similarity* with the Vector Space Model [58]. The basic idea is to represent strings using vectors in which the components are the term frequency-inverse document frequency (tf-idf) weight [58] of each token in strings, and then the similarity is computed as the product of these vectors. Cohen [18] describes a system named WHIRL which employs this similarity measure for entity resolution. One problem with this measure is that it ignores the similarity between tokens and does not work for strings with spelling errors. Bilenko [10] addresses this problem by proposing the *SoftTF-IDF* measure, which considers not only the shared tokens but also the similarity of tokens. Gravano et al. [35] introduce an approach that combines the N-gram technique and the tf-idf weighting scheme to solve the spelling error problem. For a thorough analysis of these similarity measures, Cohen et al. [20] give a detailed comparison based on results from several real datasets.

2.3.3 Phoneme-Based Similarity Measures

While the above measures focus on the spelling aspect of strings, some strings may be phonetically similar but not similar in spelling such as *Smyth* and *Smis*. For this problem, several phonetic similarity measures [15] are proposed based on phonetic codings such as Soundex [57], NYSIIS [62], ONCA [33], Metaphone [54], and Double Metaphone [55]. These measures first convert strings into codes using a phonetic coding scheme based on their pronunciations, and then compare them by matching their phonetic codes. Only strings with the same phonetic code are considered to be duplicates.

2.4 Standard Entity Resolution Approaches

After standardizing the datasets and choosing the similarity measures, the next step is to match every pair of potential duplicates and resolve the duplicates. To compare references, the entity resolution process is modeled as a probabilistic model in which the probability is the normalized similarity score of two references between 0 and 1. Depending on the similarity score and the given threshold, pairs of references are labeled as match, possible match and non-match. Newcombe et al. [52] were the first to recognize duplicate detection as a Bayesian inference problem. Fellegi and Sunter [29] formally give a theory for the problem, which combines linkage rules with the probability to resolve entities. In the next section we introduce various approaches based on the probabilistic model and its variants.

2.4.1 Similarity-Based Approaches

Using the probabilistic model, a similarity-based approach identifies duplicates by measuring the probability that two references are duplicates through similarity measures. Several approaches have employed similarity-based measures to detect duplicates. Monge and Elkan [50, 51] propose several string matching algorithms for detecting duplicated records in the database. They first introduce a basic field matching algorithm which treats fields as a sequence of atomic strings (sorted) and computes the matching score by counting the number of matched strings. To address the problem of abbreviations and ordering of strings, they propose a recursive field matching algorithm which compares every pair of strings and chooses the maximum score as the similarity score of two strings. Cohen [18] implements the WHIRL system that uses the cosine similarity measure, together with the vector space model, to measure the similarity of records.

Chaudhuri et al. [14] propose a fuzzy similarity measure to match tuples in databases. They define three transformation operations: token replacement, token insertion, and token deletion, each of which is associated with a cost. The similarity of two tuples is then measured by computing the cost required to transform one tuple into the other using these operations. Note that this method is different from the edit distance in that the fuzzy matching function focuses on token level operations while the edit distance operates at the character level.

2.4.2 Rule-Based Approaches

The probabilistic model enforces pre-defined rules on references. For example, a rule can be defined like this: *if the similarity of two references is greater than threshold θ_1, then they are considered to be duplicates; Otherwise if the similarity is less than threshold θ_2, then they are not duplicates.* The rule-based approaches aim to categorize references using these pre-defined rules or rules learned from training datasets.

Fellegi and Sunter [29] propose to use linkage rules for entity resolution and describe a theory of properties of the optimal linkage rule and the way to construct the rule. Hernández and Stolfo [37] suggest the use of an equational theory to model the logic of domain equivalence. In their system, the rules for their test data are first described and evaluated using a declarative rule language and then converted into a more efficient C implementation. The logic of these rules is expressed using well chosen similarity measures and thresholds like those introduced above. Figure 2.1 shows an example of rules defined in their method.

Lee et al. [45] describe a knowledge-based cleaning framework for duplicate identification. They build their knowledge base using a set of rules such as duplicate identification rules, merge/purge rules, update rules and alert rules. Each rule is in the form of **if** <**condition**> **then** <**action**> in which the *action* is fired when the *condition* is met. Jin et al. [40]

```
Given two records, r1 and r2.
IF the last name of r1 equals the last name of r2,
   AND the first names differ slightly,
   AND the address of r1 equals the address of r2
THEN
   r1 is equivalent to r2.
```

Figure 2.1: A simplified rule in English to illustrate the equational theory [37].

also use the rule-based decision model to detect duplicates. Galhardas et al. [30] describe another data cleaning framework which aims to separate the logical expression of the data transformation from their physical implementation. They define a declarative language based on logical data transformation operators and use this language to describe the data cleaning problem at the logical level. One advantage of this method is that it can simplify the testing and deployment of rules.

2.4.3 Supervised-Learning-Based Approaches

A problem with similarity-based and rule-based approaches is that the weights of different attributes and the threshold for optimal resolution results are difficult to decide. Also, some references may require different measures for different attributes. Manually choosing the measures or weights for each attribute is not only tedious but apt to produce errors. In the presence of some labeled samples, a number of supervised learning approaches that incorporate the properties of datasets into the similarity measures are proposed for entity resolution.

Bilenko et al. [10] compare several similarity measures on different benchmark datasets, they found out that no single similarity measure can perform the best in all cases. They propose to use similarity vectors of two references to train a binary support vector machine (SVM) classifier and use the classifier's confidence in the *match* class as a new similarity measure for references. Tejada et al. [64] describe an object identification system consisting of two learner components: a mapping rule learner that learns to find the most important attributes and the best combination of attributes for mapping objects, and a transformation weight learner that learns the weights for a set of predefined string transformations for each attribute. Other approaches [21, 59, 71] also employ supervised learning methods to train a probabilistic model like decision tree, Bayesian network or SVM, and later use the model to identify duplicates. Zhu et al. [71] use a genetic algorithm to adaptively learn the costs for each string edit operation of the edit distance.

2.4.4 Active-Learning-Based Approaches

One problem with supervised learning techniques is that a large training dataset is required to train a classifier and preparing such a training dataset that contains enough negative or positive representative cases is difficult. To reduce the size of training datasets and still achieve high accuracy, some methods have been proposed using active learning techniques [22].

An active learner starts with a small labeled training dataset and a large pool of unlabeled data items. It then actively picks out the important records that when labeled will improve the performance of the learner during the learning process. Each time new labeled records are added into the training dataset, the learner will be retrained. ALIAS [59] is a learning-based duplicate detection system which aims to minimize the size of labeled datasets. For ambiguous instances that are difficult to resolve, they introduce an uncertainty score for each instance and propose a classifier independent way to measure the uncertainty score of instances. Using the uncertainty score, the system picks out the most uncertain instances to be manually labeled and retrains the learner based on the users' feedback. Tejada et al. [64, 63] use a similar strategy for learning mapping rules. A mapping-rule learner in their system actively chooses the most informative candidate mappings and thus reduces the number of training examples.

2.5 Relational Entity Resolution Approaches

Most traditional entity resolution approaches rely heavily on the attributes; However, when the attributes are not enough to uniquely identify the references, these approaches will not work properly. Recently, with the increasing number of relational datasets, many approaches focus on exploring the links or relationships between entities for entity resolution.

In relational datasets like databases with key-foreign key constraints, or datasets of social networks, references are often involved in several relationships and these relationships can provide additional information to help with entity resolution. Ananthakrishna et al. [3] use the dimensional hierarchy derived from the key-foreign key relationship in data warehouses as extra information to identify duplicates. Also they propose to propagate the merging results from level to level through a top-down traversal approach so that the process can benefit from prior results. Bhattacharya and Getoor [6, 8] propose an iterative entity resolution approach which combines the attribute and linkage similarity to resolve entities iteratively. Furthermore, they explore several relational similarity measures like Common Neighbors, Jaccard Coefficient, Adamic/Adar Similarity, Adar Similary with Ambiguity Estimate, and Higher-Order Neighborhoods. Parag et al. [60] use a collective model to solve the entity resolution problem; They build undirected graphical models for the datasets based

on conditional random fields (CRF) in which nodes are record nodes (binary nodes of pairs of records) and attribute nodes (pairs of attributes and their similarity score), and edges are the connections between record nodes and attribute nodes. Using the model, the optimal inference can be performed in polynomial time by computing the min-cut of the graph. However, the granularity of their graph (nodes on attribute level) makes their approach difficult to scale for large datasets.

Dong et al. [25] propose a reference reconciliation algorithm that exploits the relationships among different types of references. They model the relationships as a dependency graph in which the nodes are similarity between references, and edges are the dependency relationship between the resolution result of reference pairs. Then entities are iteratively resolved and the resolving results are propagated along the dependency graph. One problem with their approach is that building the dependency graph is expensive and may not scale for large datasets. Kalashnikov et al. [43] also use the relationships between references to construct a graph in which the nodes are entities and edges are the relationships between them. They measure the relational similarity of potential duplicates by computing the confidence weight for each possible path between nodes.

More recently, Kalashnikov et al. [44, 41] apply the relational entity resolution algorithm on people disambiguation using the relationships extracted from the web such as the co-occurrence of references and the hyper-links between them. Furthermore, they develop the WEST (Web Entity Search Technologies) system [42] to improve the searches of people over the Internet. The system employs several approaches: a GraphER approach which disambiguates people by analyzing the social network extracted off the web pages, an EnsembleER approach which uses supervised learning to combine results of multiple base ER systems, and a WebER approach which further improves the GraphER approach by collecting from the Web a large amount of additional information about the entities in the system.

Bhattacharya and Getoor [7] formulate the multi-type entity resolution using the bibliography dataset and explore some preliminary work on using relationships between multiple types of references. However, their results mostly focus on how relational clustering outperforms attribute-based clustering, and no experiments are carried out to evaluate the accuracy and efficiency of the multi-type entity resolution approach, especially no results show that it is the multi-type relationship or the single-type relationship that improves the clustering results.

2.6 Techniques For Improving Efficiency

In addition to the accuracy of entity resolution algorithms, efficiency is another challenge for entity resolution, especially when the datasets are large. In real datasets, most reference pairs are not duplicates, and comparing them will waste plenty of time and resources. A

number of approaches have been proposed to reduce the number of unnecessary comparisons and improve the efficiency.

2.6.1 Sorted Neighborhood Method (SNM)

Sorted Neighborhood Method (SNM) is a method proposed by Hernández and Stolfo [37] to effectively reduce the number of comparisons by limiting the similarity measures on a small portion of the datasets. The method can be summarized in three steps. The first step is to choose representative attributes of references and compute a key for each reference based on these attributes. The second step is to sort the dataset over the key. And the last step is to move a fixed size sliding window over the sorted references and comparing references within the same window. For example, if the size of the sliding window is w, then every reference is compared with its previous $w-1$ references. As the window moves forward, the first reference in the window will slide out.

As we see, the accuracy of the SNM method depends on the quality of the keys chosen. A poorly-chosen key will filter out a lot of true duplicates. Also, the size of the window is very important. For references with a large number of duplicates, a small window size will result in the same result as that of a poorly-chosen key, and a large window size will bring in too much unnecessary comparisons. To solve this problem, they implemented a multi-pass sorted neighborhood method which runs the SNM method independently several times and each time the references are sorted using a different key and a small window size. Finally the final result is computed by merging the results from the multi-pass.

2.6.2 Canopies

McCallum et al. [49] propose the use of *Canopies* for partitioning datasets. The basic idea is to use an inexpensive approximate distance measure to roughly partition the whole dataset into a number of overlapping subsets which are referred to as *canopies* and then employ standard similarity measures to compare references within the same canopy. For attributes with text values, they propose to use an inverted index to efficiently construct canopies. This technique is also used in [21, 8] for reducing the computational cost. Cohen and Richman [21] propose to use the cosine similarity together with the tf-idf weighting scheme as the approximate distance measure. Gravano et al. [34] implement the approximate string joins in database framework using the positional N-gram which can find more potential duplicates that cannot be found using tf-idf. Chaudhuri et al. [14] also employ N-grams to build an error tolerant index for quickly retrieval of potential candidates.

Nin et al. [53] introduce a semantic blocking technique for building *canopies*. In their method, each canopy is initialized with a reference and then iteratively expanded by merging relational connected references until no more references can be added. This technique can

avoid typographical errors since the canopies are built using semantic context (relationships) rather than string similarity measures. However, for datasets that are not highly connected, this method may not work well.

2.7 Tools for Entity Resolution

Over the past few years, a number of tools and products for entity resolution have been developed in the research community and the commercial market.

Febrl [1] (Freely Extensible Biomedical Record Linkage) [15] is an open-source tool that is used for entity resolution and provides a platform for researchers to evaluate various entity resolution techniques or conduct their own experiments. As an entity resolution system, Febrl implements several components: a component for data cleaning and standardization, a component for Hidden Markov Model training, and a component that consists of several attribute similarity measures like phonetic name encoding and approximate string comparison. In addition, researchers can add their own measures and resolution algorithms in the tool.

TAILOR [26] (**R**ec**o**rd **Li**nk**a**ge **T**oolbox) is a record linkage toolbox that implements most of the state-of-the-art tools and models in the literature for entity resolution. The system adapts a layered design and consists of four layers: a searching methods layer for efficiency which implements several blocking methods including sorting blocking, hashing bocking, and sorted neighborhood approaches, a comparison functions layer which contains several similarity measures like Hamming distance, Edit distance, Jaro's distance, N-grams and Soundex code, a measurement tools layer which provides several performance metrics to assess the accuracy and performance of different entity resolution algorithms, and a supporting tools layer that provides additional functionality. In addition, the toolbox implements several machine learning models for entity resolution such as a induction model, a clustering model and a hybrid model.

GRLS [28] (Generalized Record Linkage System) is an entity resolution system based on Fellegi-Sunter probabilistic linkage theory. It implements the entity resolution in three phases: a searching phase which generates potential pairs using criteria provided by users, a decision phase which applies linkage rules to potential pairs and a grouping phase to group records that are considered to refer to the same entity in the decision phase. In addition, GRLS provides a framework for users to test linkage parameters and perform queries.

On the commercial market, IBM has developed several products [2] for information integration. They provide solutions to standardize data files, validate and enrich data elements using trusted data like postal records and address information; Also they match records

[1] `http://sourceforge.net/projects/febrl/`
[2] `http://www-01.ibm.com/software/data/integration/`

across different data sources and identify duplicates. Trillium Software [3] provides solutions for data quality including data cleaning and standardization, entity de-duplications and identification, and address validation. Google Fusion Tables [4] is an experimental system that focuses on data management and collaboration like merging multiple data sources, querying and visualizing the datasets and Web publishing.

[3] http://www.trilliumsoftware.com/
[4] http://tables.googlelabs.com/

Chapter 3

Multi-Type Relational Entity Resolution

In this chapter, we introduce the entity resolution approach that uses multiple relationships to resolve multiple types of references. Section 3.1 gives a formal definition of the entity resolution problem. Section 3.2 introduces the graph model adopted for modeling the relational dataset. In Section 3.3, we describe the details of the multi-type relational entity resolution approach including the clustering algorithm, the bootstrapping process, and the iterative relational clustering process. Section 3.4 describes the similarity measures used in our approach. Finally Section 3.5 discusses how the constraints enforcement can help with the entity resolution.

3.1 Problem Definition

Notation

We first give the notation for formalizing the entity resolution problem. The input of the approach is a relational dataset consisting of a set of records. Let $\mathcal{D} = \{rec_1, rec_2, \ldots, rec_n\}$ be the dataset, in which rec_i is a record. Each record $rec_i = \{r_{i1}, r_{i2}, \ldots, r_{ik}\}$ is a bibliographic entry corresponding to a paper, and each entry contains several references r_{ij} such as author, paper, venue and so on. Associated with each reference are its type $r_{ij}.T$ and a set of attributes $\{r_{ij}.A_1, r_{ij}.A_2, \ldots\}$. Examples of type are author, paper, venue, etc. Each reference has only one type, and at least one attribute; and the number of their attributes varies according to their types and data sources.

Besides the input dataset, we also have a *ground truth* dataset for evaluating the accuracy of our approach. Let $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ be the set of entities in $\mathcal{D}$. Each entity $e_j = \{r_{j1}, r_{j2}, \ldots\}$ has a set of references referring to it. The same as references, each entity has a type $e_j.T$ and a set of attributes $\{e_j.A_1, e_j.A_2, \ldots\}$.

Figure 3.1(a) shows the references in the example dataset using the notation defined

Reference Schema: reference = {Attributes..., Type}

r_{11} = {Jiong Yang, author} r_{12} = {Richard R. Muntz, author} r_{13} = {UCLA, affiliation}
r_{14} = {ACM-GIS, venue} r_{15} = {Mapping..., 1996, paper}

r_{21} = {W. Wang, author} r_{22} = {J. Yang, author} r_{23} = {Richard Muntz, author} r_{25} = {VLDB, venue}
r_{24} = {University of California, Los Angeles, affiliation} r_{26} = {STING..., 1997, paper}

r_{31} = {Jiong Yang, author} r_{32} = {Wei Wang, author} r_{33} = {Case Western Reserve University, affiliation}
r_{34} = {UNC, affiliation} r_{35} = {ICDM, venue} r_{36} = {AGILE..., 2004, paper}

r_{41} = {Wei Wang, author} r_{42} = {Jiong Yang, author} r_{43} = {KDD, venue}
r_{44} = {A framework..., 2004, paper}

r_{51} = {W. Wang, author} r_{52} = {V. Srinivasan, author} r_{53} = {M. Motani, author}
r_{54} = {Conference on Mobile Computing and Networking, venue} r_{55} = {Adaptive..., 2007, paper}

r_{61} = {Wei. Wang, author} r_{62} = {Vikram Srinivasan, author} r_{63} = {Mehul Motani, author}
r_{64} = {MOBICOM, venue} r_{65} = {Adaptive..., 2007, paper}

(**a**) Example dataset formalized using the notations.

e_1 = {r_{11}, r_{22}, r_{31}, r_{42}} e_2 = {r_{12}, r_{23}} e_3 = {r_{21}, r_{32}, r_{41}} e_4 = {r_{13}, r_{24}}
e_5 = {r_{51}, r_{61}} e_6 = {r_{52}, r_{62}} e_7 = {r_{53}, r_{63}} e_8 = {r_{54}, r_{64}} e_9 = {r_{55}, r_{65}}

(**b**) Ground truth of the dataset.

Figure 3.1: References in the example dataset and the ground truth.

above. The dataset $\mathcal{D}_s = \{P1, P2, \ldots, P6\}$ contains six records, each of which consists of several references. As shown in the figure, all the references follow the same schema: a reference type and a set of attributes. Note that relationships between references are not presented there, and we will show them in the graph model later.

The Objective of Entity Resolution

Given the dataset $\mathcal{D}$, the goal is to find, for each entity, all the references referring to it and then group those references together. Formally, it is to partition the dataset $\mathcal{D}$ into a set of clusters $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$, in which each cluster $\mathcal{C}_i = \{r_{i,1}, r_{i,2}, \ldots, r_{i,k}\}$ consists of a set of references referring to the same entity e_i and no references in other cluster $\mathcal{C}_j$ refer to e_i. For the example dataset, the ideal partition is the ground truth given in Figure 3.1(b).

3.2 Graph Model for Relational Entity Resolution

To measure the relational similarity of references, we first need to model the relationships between references. The intuitive way is to build a graph of the dataset in which nodes are references and edges are relationships between references. Note that the nodes may be different types of references instead of single type, and they could be different references such as author and paper. Also, depending on the types of references, the relationships between references can be different such as *co-authorship* and *writtenBy*.

Figure 3.2 shows such a graph for record *P1*. This graph model is commonly used by different approaches [25, 43]. The advantage of this graph model is that different types of

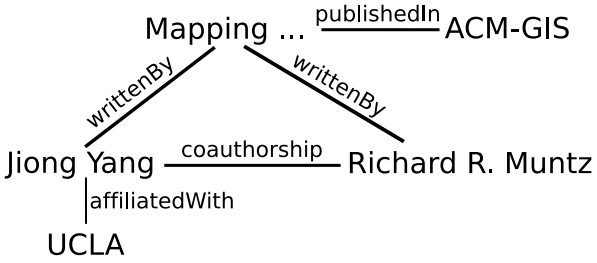

Figure 3.2: Graph model used by other approaches [25, 43].

references and all the relationships are modeled and can be accessed directly. However, the problem is that too much information needs to be maintained. For example, for record *P1*, we need to store 5 nodes and 5 edges in the graph. Considering datasets with millions of references, the approaches using this graph model will not work properly in that case.

To simplify the problem, we use the hyper-edge model proposed by Bhattacharya and Getoor [8] instead of the graph model shown above. A hyper-edge model is a model using hyper-edges to model the relationships in records, in which the references are the nodes associated with the hyper-edge. One problem with Bhattacharya and Getoor's approach is that they only consider one type of reference (author) in their approach. To adapt it for the multi-type entity resolution, we generalize the model to allow multiple types of reference. So for each hyper-edge (record), the nodes (references) associated with it are not just one type, but all the types of references in the record.

Here we illustrate how we build the graph model and how this graph model helps with the entity resolution, For each record $rec_i = \{r_{i1}, r_{i2}, \ldots\}$ in the dataset, we create a hyper-edge $\mathcal{H}_i = \{r_{i1}, r_{i2}, \ldots\}$, on which are the references in record rec_i. To facilitate accessing neighbors on the hyper-edge, each reference r_{ij} has an attribute $r_{ij}.\mathcal{H} = \mathcal{H}_i$ which is the hyper-edge r_{ij} is associated with. Thus instead of storing all the neighbors in a set for access, we access neighbors through the hyper-edge. Also the relationships between references can be inferred from the types of the connected references.

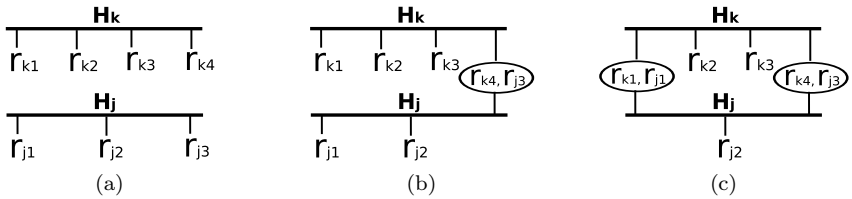

Figure 3.3: Process of the relational entity resolution using hyper-edge graph model.

Figure 3.3a shows the initial hype-edge-based graphs $\mathcal{H}_k$ and $\mathcal{H}_j$ for two records respectively. From the figure we can see that no graph is connected with one another at

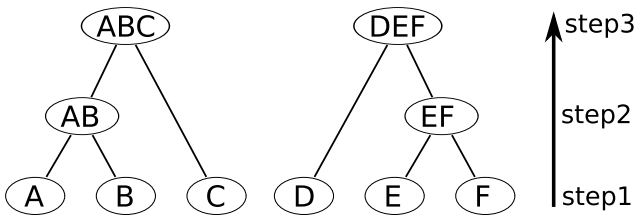

Figure 3.4: Agglomerative hierarchical clustering process.

the initial stage, this is because no references are matched and merged so far. As duplicates are found and merged, the graphs (hyper-edges) are then connected by these merged references as shown in Figure 3.3b. Then we can use the connected graphs to seek more relational information (common neighbors) for references to further improve the resolution results (Figure 3.3c). By iteratively repeating this process, we can resolve the duplicates and ultimately reach our goal of entity resolution.

Given the graph model for relational entity resolution, we then describe how the model can be used for measuring relational similarity and resolving duplicates.

3.3 Multi-Type Relational Entity Resolution

In this section, we describe the approach for resolving multiple types of entities using relationships. We begin by introducing the agglomerative hierarchical clustering algorithm. Then we introduce the bootstrapping process which uses the attributed-based method to initialize some connections between graphs. After the bootstrapping, we illustrate how we use relationships for entity resolution. At last we describe the iterative clustering process for further improving accuracy of entity resolution. The ideas of using agglomerative hierarchical clustering, bootstrapping and relationships were proposed by Bhattacharya and Getoor [8]. However, they only explore these ideas on single type of reference (author), and we, in this thesis, extend them for multiple types of references.

3.3.1 Agglomerative Hierarchical Clustering (AHC)

Entity resolution is a process of matching potential duplicates and merging true duplicates. According to the definition of *clustering* [36]: "the process of grouping a set of physical or abstract objects into classes of *similar* objects is called *clustering*", and "a *cluster* is a collection of data objects that are *similar* to one another within the same cluster and are *dissimilar* to the objects in other clusters", the entity resolution process is a natural *clustering* process in which each group of results generated by the clustering process is a *cluster*.

For clustering, we employ the agglomerative hierarchical clustering algorithm [36] to

```
   input  : R = {r_1, r_2, ..., r_n}
   output: C = {c_1, c_2, ...} AND c_i = {r_{i,1}, r_{i,2}, ...}
 1 //Cluster initialization. Initialize each cluster with a reference.
 2 C = {c_1, c_2, ... c_n} and c_i = {r_i};
 3 //Graph model initialization.
 4 initializeGraphModel(C);
 5 //blocking.
 6 foreach c_i ∈ C do
 7 |  S_i = findPotentialDuplicates(c_i);
 8 end
 9 //Bootstrapping.
10 foreach c_i ∈ C do
11 |  bootstrapping(S_i, c_i);
12 end
13 //Relational clustering.
14 while there are new clusters merged do
15 |  foreach c_i ∈ C do
16 |  |  foreach s_i ∈ S_i do
17 |  |  |  simScore = similarity(c_i, s_i);
18 |  |  |  if simScore ≥ merge-threshold then
19 |  |  |  |  newCluster = merge(c_i, s_i);
20 |  |  |  |  add newCluster to C;
21 |  |  |  |  notify neighbors and similar clusters of c_i and s_i;
22 |  |  |  |  remove c_i and s_i from C;
23 |  |  |  end
24 |  |  end
25 |  end
26 end
27 return C;
```

Algorithm 1: Algorithm for the memory-based entity resolution

resolve the references. The basic process of AHC is shown in Figure 3.4. Given a set of references and similarity measures, the algorithm starts by placing each reference in a separate cluster (step 1). During the clustering process (step 2), the algorithm uses similarity measures to match pairs of potential duplicates and merge the pairs that are considered to be true duplicates (pairs with similarity score greater than a given threshold). This process repeats iteratively until no more clusters can be merged or a termination condition is satisfied (step 3). The final sets of clusters are the results of the clustering algorithm.

Algorithm 1 gives a high level description of our approach based on the AHC algorithm. The input is a set of references in the dataset and the output is the final clustering results of the algorithm. The first step is to initialize a set of clusters with one reference per cluster (Line 2). Also we initialize the graph model for the records in the dataset (Line 4). Lines 6-8 show the blocking process which aims to improve the efficiency by reducing unnecessary comparisons, the details of blocking will be introduced in Chapter 4. After the

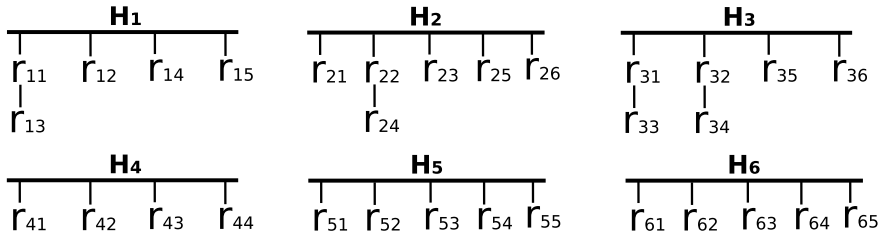

Figure 3.5: The graph model of the example dataset in the initial state.

blocking, we execute the bootstrapping process (Lines 10-12) and the relational clustering process (lines 14-26) which will be introduced in the following sections.

3.3.2 Graph Model Initialization

The first step of our entity resolution approach is to initialize the graph model. For each record, we build a simple graph using a hyper-edge. The nodes on the edge can be paper references, author references and venue references. Note that the affiliation is associated with the author reference rather than the hyper-edge. The initial graphs for the example dataset are shown in Figure 3.5 in which all references are initially placed in a separated cluster and separated from each other. The r_{13}, r_{24}, r_{33} and r_{34} are affiliations of the corresponding author references.

3.3.3 Bootstrapping

At the initial stage, each reference is placed in a separated cluster, and these graphs are not connected to each other; However, some initial connections between graphs are needed to measure the relational similarity of references. Bootstrapping is one such process aiming to boost some initial connections by merging clusters with high attribute similarity.

The bootstrapping uses the attribute-based approach for duplicates resolution, which matches and merges references according to their attribute similarity. Given a pair of references, an attribute similarity measures and a merging threshold, the attribute similarity score is first computed for the pair of references; If the similarity score is greater than the given threshold, then this pair of references is considered to be duplicates and thus is merged together; Otherwise, we continue to process the next pair. Considering the facts that a decision cannot be adjusted once it is made during the clustering process (which is one drawback of the AHC algorithm), and that the merging decisions made during the bootstrapping will affect the results of the relational clustering, a high accuracy of the bootstrapping results needs to be maintained so that the following clustering process will not be affected much by the false positives.

Algorithm 2 describes the details of the bootstrapping process. The input is a set

input : $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$
output: $\mathcal{C}$

foreach $c_i \in \mathcal{C}$ **do**
  $mergeSet.add(c_i)$;
  //$\mathcal{S}_i$ is a set of potential duplicates of c_i obtained via blocking.
  foreach $s_{ij} : \mathcal{S}_i$ **do**
    if *merged*(c_i, s_{ij}) **then**
      continue;
    end
    $score \leftarrow sim_A(c_i, s_{ij})$;
    if $score \geq mergeThreshold$ **then**
      $mergeSet.add(s_{ij})$;
    end
    if $score < removeThreshold$ **then**
      $remove(\mathcal{S}_i, s_{ij})$;
      $remove(\mathcal{S}(s_{ij}), c_i)$;
    end
  end
  if *size of* $mergeSet \leq 1$ **then**
    continue;
  end
  $c_{new} \leftarrow merge(mergeSet)$;
  $add(\mathcal{C}, c_{new})$;
  for *mergedCluster : mergeSet* **do**
    $remove(\mathcal{C}, mergedCluster)$;
  end
end

Algorithm 2: The Bootstrapping Step.

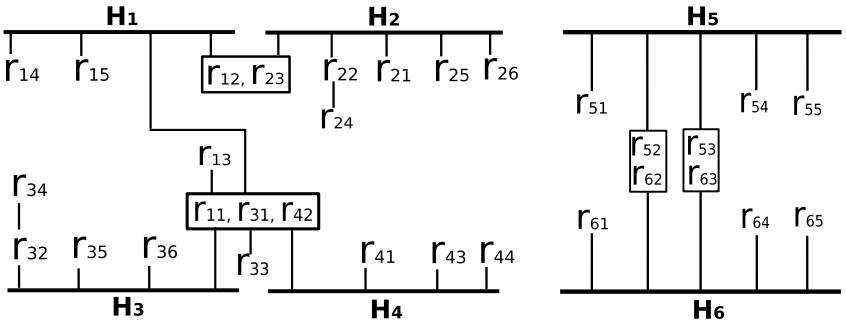

Figure 3.6: The bootstrapping result of the example dataset.

of clusters initialized with one reference per cluster, and a set of potential duplicates for each cluster retrieved through the blocking. The bootstrapping goes through the set of clusters *once* (not iteratively). For each cluster c_i, we compare it with its potential duplicate s_{ij} (Lines 4-16). Lines 5-7 check if c_i and s_{ij} have been compared before, if so we ignore them and continue to process the next potential duplicate; Otherwise, we compute their attribute similarity *score* (Line 8). If the score is greater than the given merging threshold *mergeThreshold*, we add the reference into the merging set *mergeSet* (Lines 9-11). To improve the efficiency, we define a removing threshold *removeThreshold* which is used to filter out pairs of references with similarity score less than *removeThreshold* (Lines 12-15). After comparing c_i with all of its potential duplicates in $\mathcal{S}_i$, we then check if any clusters should be merged (Line 17). If not, we continue to handle the next cluster; Otherwise, we merge the clusters in *mergeSet* into a new cluster and remove the merged clusters from cluster set $\mathcal{C}$.

In the motivating example dataset, we can infer that *Richard R. Muntz* (r_{12}) in *P1* and *Richard Muntz* (r_{23}) in *P2* refer to the same author using their name attributes, and the same for *Jiong Yang* (r_{31}) in *P3* and *Jiong Yang* (r_{42}) in *P4* since they share exactly the same first name and last name. For author references with abbreviated first name like *W. Wang*, *J. Yang*, and *V. Srinivasan*, it is difficult to tell if they refer to *Wei Wang*, *Jiong Yang*, and *Vikram Srinivasan* respectively because the abbreviated first name may be used by other authors with the same last name. However, compared to *W. Wang* and *Y. Wang*, we are more confident that *V. Srinivasan* and *Vikram Srinivasan* refer to the same author since *Srinivasan* is an uncommon last name in the dataset. Also we believe *M. Motani* and *Mehul Motani* are the same author for the same reason. So through the bootstrapping, we can identify some duplicated references. Figure 3.6 gives the bootstrapping result of the example dataset.

One may argue that *Wei Wang* (r_{41}) in *P4* and *Wei Wang* (r_{61}) in *P6* have the same name and should be merged as that of *Jiong Yang* (r_{31}) in *P3* and *Jiong Yang* (r_{42}) in

P4; However, they are not the same author according to the ground truth (Figure 3.1(b)). In fact, *W. Wang* (r_{21}) in *P2*, *Wei Wang* (r_{32}) in *P3*, and *Wei Wang* (r_{41}) in *P4* refer to the *Wei Wang* in the University of North Carolina, while *W. Wang* (r_{51}) in *P5* and *Wei Wang* (r_{61}) in *P6* refer to the *Wei Wang* in National University of Singapore.

Apparently, these ambiguous names cannot be distinguished during the bootstrapping process using attribute-based approaches. In next section, we will introduce the approach using relationships to further improve the results of entity resolution.

3.3.4 Iteratively Relational Entity Resolution

As introduced before, the relationships between references can provide additional information for entity resolution. In this section, we introduce how we use relationships to resolve duplicates and how we propagate the merging results through the iteratively clustering process.

Relational Clustering

As Figure 3.6 shows, the separated local graphs are connected somehow after the bootstrapping. For example, *P1* and *P2* are connected through r_{12} and r_{23}, *P1, P3*, and *P4* are connected by author *Jiong Yang* (r_{11}, r_{31}, r_{42}). Also some references that were not related before, may have new evidence to show they are true duplicates, such as the *co-authorship* between author references, the *publishedIn* between paper and venue references, and the *writtenBy* between paper references and author references.

For the example dataset, through the *co-authorship*, we can infer that r_{11} (*Jiong Yang*) and r_{22} (*J. Yang*) are the same person since they both colloborate with the same author *Richard Muntz* (r_{12} and r_{23}). Also r_{32} (*Wei Wang*) and r_{41} (*Wei Wang*) refer to the same person because they collaborate with *Jiong Yang* (r_{11}, r_{31}, and r_{42}). Similarly, it is the same for r_{51} (*W. Wang*) and r_{61} (*Wei Wang*). Figure 3.7 shows the result of the first round relational clustering. The highlighted nodes and edges are the merged references and the relationships used for clustering.

Iterative Clustering

From the illustration above, we can see that the context of references is not static. When two references are merged, their neighbors and similar references get new information and need be re-evaluated. For example, when r_{12} (*Richard R. Muntz*) and r_{23} (*Richard Muntz*) are merged, their neighbors r_{11} (*Jiong Yang*) and r_{22}(*J. Yang*) which were irrelevant before, now get connected through the merged references and can be merged because of the shared neighbor (Figure 3.8). So the intermediate resolution results can be used to further improve the clustering results. To propagate the merging results, we cluster the references

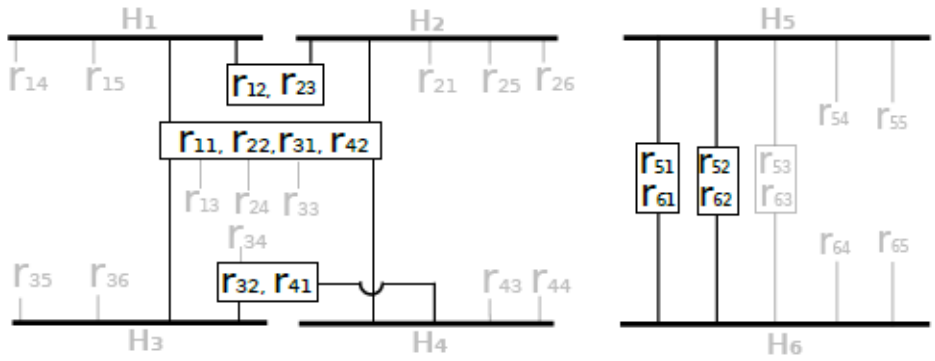

Figure 3.7: The first round clustering result of the example dataset.

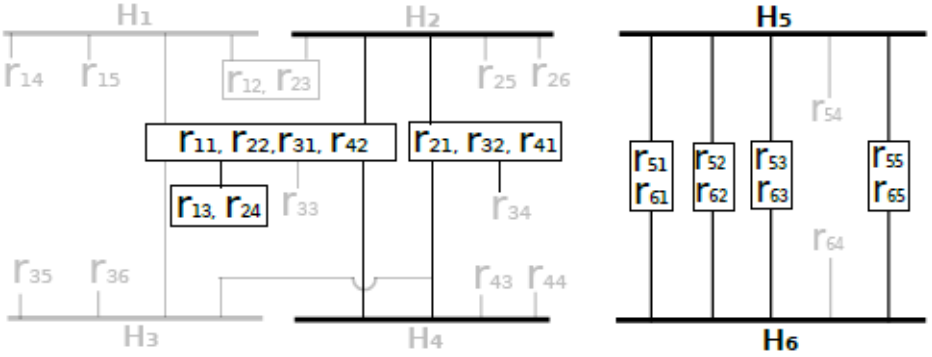

Figure 3.8: The second round clustering result of the example dataset.

iteratively instead of separately. Each time when a pair of references is merged, we notify their neighbors and related references and re-evaluate them in the next iteration.

In the example dataset, we apply the propagation strategy on the result of the first round clustering. When references r_{11} (*Jiong Yang*) and r_{22} (*J. Yang*) are merged, the merging information is propagated to their affiliations r_{13} (*UCLA*) and r_{24} (*University of California, Los Angeles*) respectively, and results in the merging of r_{13} and r_{24}. Also, after the first round clustering, author reference r_{21} (*W. Wang*) now shares a common neighbor *Jiong Yang* with r_{32} and r_{41} (*Wei Wang*), and can be merged for their relational similarity. Similarly, paper references r_{55} and r_{65} are merged because of the three shared author references and the paper reference.

This clustering process repeats until no duplicates can be found any more. Figure 3.9 shows the final results of the iterative clustering. So through various relationships, information propagation and the iterative process, we can resolve the duplicates for multi-type entities simultaneously.

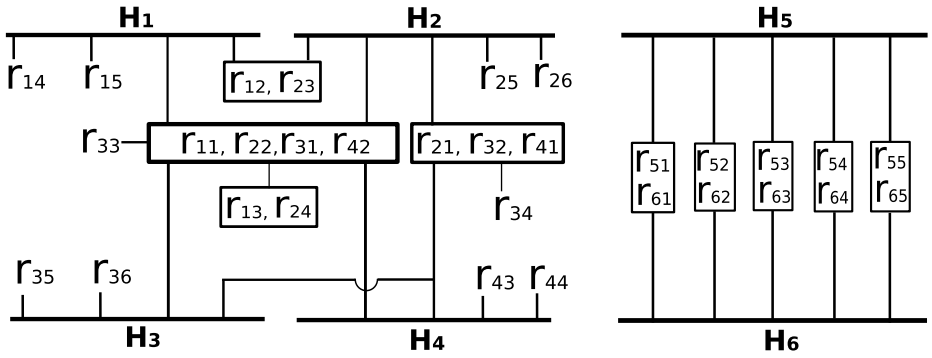

Figure 3.9: The final clustering result of the example dataset.

Implementation

Algorithm 1 (Lines 13-26) gives the process of the iteratively relational clustering. In every iteration, we compute the similarity score (which combines attribute similarity and relational similarity) of each cluster c_i and its potential duplicate s_i. For the pair of references with similarity score greater than the merging threshold, we merge them into a new cluster, and then remove them from cluster set $\mathcal{C}$. The different between the relational clustering process and other method is that for each merged pair of clusters, we notify their neighbors and related potential duplicates. By doing so, these affected clusters will be re-compared with their potential duplicates. Also, those unaffected clusters will not be considered again so that unnecessary comparisons can be avoided and the process can converge finally.

3.4 Similarity measures

The accuracy and efficiency of the entity resolution algorithm greatly depend on the methods employed for comparing and merging references. In this section, we introduce various similarity measures used in our approach including attribute similarity measures, relational similarity measures and their hybrid.

3.4.1 Name Ambiguity

Before introducing the similarity measures, we first describe the ambiguity of author names. Recall that in the motivating example, we merge *V. Srinivasan* in *P5* and *Vikram Srinivasan* in *P6* because their names are very similar and not *ambiguous*. But the similar names *W. Wang* in *P5* and *Wei Wang* in *P6* are not merged because we think they are common names and may represent different entities. So a natural question is how to determine if a given name is *ambiguous*.

We use the name ambiguity measure introduced by Bhattacharya and Getoor [8] in which the name ambiguity is defined as the probability that a name is shared by different entities.

In our datasets, the author names are mainly in two forms: *F. Last* with initial first name and full last name, and *First Last* with the full name. For names in the form *F. Last*, the name ambiguity is indicated by the number of distinct names (same last name and different first name) in the dataset. For example, *W. Wang* is considered as ambiguous because it is shared by three distinct names *Weining Wang*, *Wenqiang Wang* and *Wei Wang*. In our approach, we define a name as ambiguous if it is shared by at least two distinct authors. For names in form *First Last*, we do not define their ambiguity as a whole; Instead, we measure the ambiguity of its first name and last name which is computed as the number of distinct names using them divided by the number of total names using them. Then when we compare a pair of author references, we incorporate this ambiguity into their name similarity like multiplying the similarity by a coefficient less than 1.0.

3.4.2 Attribute Similarity Measures

Attribute similarity measures are employed to compute attribute similarity for references. The similarity score is then used for reference resolution decisions. In our datasets, the attributes of references are strings such as the paper title, author names and others, so we are only concerned about the string similarity measures.

Given a pair of references r_i and r_j with a set of attributes $r_i.A_k$ and $r_j.A_k$ respectively [1], let $sim_A(r_i, r_j)$ be the attribute similarity of r_i and r_j, and $sim_{str}(r_i.A_k, r_j.A_k)$ be the string similarity of r_i and r_j's attribute A_k. The attribute similarity of r_i and r_j is computed as the summation of the string similarity of each attribute as follow.

$$sim_A(r_i, r_j) = w_1 * sim_{str}(r_i.A_1, r_j.A_1) + w_2 * sim_{str}(r_i.A_2, r_j.A_2) + \dots \tag{3.1}$$

in which $w_1, w_2, \dots$ are the weights assigned to each attribute according to their importance. Usually, the weights can be tuned manually by the domain experts or learned from a labeled dataset. In our approach, we manually choose the weights that can be used to achieve best accuracy.

Although we use the same measure to compute the string similarity, the way to compute the similarity of papers and authors is still different. For papers, the string similarity measure can be applied on the title directly and the formula to compute the string similarity is as follow:

$$sim_{str}(str1, str2) = \frac{EditDistance(str1, str2)}{max(length(str1), length(str2))} \tag{3.2}$$

For authors, the situation is slightly different. First, the author names are usually very short. Second, a lot of names are provided with initial names like *J. Yang*. These features make the edit distance not work well directly on them. Thus we split the name into different

[1] Here we assume the pair of references has the same number of attributes. For the reference that does not have the attribute of the other reference, we add the attribute with a null value.

parts, then measure the similarity score for each of them, and combine these scores together as the final attribute similarity score of the author references. The formula is defined as follow:

$$\begin{aligned} sim(name1, name2) &= \alpha * sim_{str}(firstname1, firstname2) \\ &+ \beta * sim_{str}(middlname1, middlname2) \\ &+ (1 - \alpha - \beta) * sim_{str}(lastname1, lastname2) \end{aligned}$$

in which the α, β, and $(1-\alpha-\beta)$ are the weights for each part of the name. Note that if the *firstname* or *lastname* of the two references is exactly the same and is ambiguous according to our measurement of name ambiguity , we will decrease their similarity score by a value between 0.0 and 1.0.

3.4.3 Relational Similarity Measures

Relational similarity measures are used to assess the relationship similarity between references, and can provide additional information for entity resolution. Several measures have been evaluated in [8] such as *Common Neighbors, Jaccard Coefficient, Adamic/Adar Similarity, AdarName*, and *Higher-Order Neighborhoods*, of which we choose *Jaccard coefficient* as our relational similarity measure for the following reasons. First, *Jaccard coefficient* has been shown to outperform other measures except *AdarName*; Second, the *AdarName* which performs best can only be used for author entity resolution and we need to resolve other references such as papers.

Jaccard coefficient is a measure for computing the similarity between two sets. For any reference pair (r_i, r_j), as the graph model shows, each reference has a set of neighbors that can be collected through the hyper-edges. We let $Nbr(r_i) = \{r_{i1}, r_{i2}, \ldots\}$ be the neighbors of r_i, and $Nbr(r_j) = \{r_{j1}, r_{j2}, \ldots\}$ be the neighbors of r_j; Then r_i and r_j are considered as relational similar if they share at least one common neighbor, and we measure the relational similarity by the *Jaccard coefficient* which is defined as the size of the intersection divided by the size of union of the two sets of neighbors. The formula is given as follow:

$$sim_R(r_i, r_j) = Jacc(r_i, r_j) = \frac{\mid Nbr(r_i) \cap Nbr(r_j) \mid}{\mid Nbr(r_i) \cup Nbr(r_j) \mid} \tag{3.3}$$

Depending on the requirements of the entity resolution task and the relationships available, the relational similarity measures are divided into single-type relational similarity measures which measure the Jaccard Coefficient of only one type of neighbors such as the co-authors, and multi-type relational similarity measures which measure the Jaccard Coefficient of several type of neighbors of the given reference such as co-authors, papers or affiliations.

Single-type Relational Similarity Measures

Single-type entity resolution is required in many situations such as personal information management or demographic census where the main task is to reconcile duplicated person references. In the co-authorship social network, only the author references need to be resolved and the relationship commonly used is the co-authorship [8].

We can use Function 3.3 to compute the single-type relational similarity. However, there are some restrictions on the references and their neighbors which are listed as follow:

- $r_i.T = r_j.T$.
- $r_{i1}.T = r_{i2}.T = \ldots = r_{j1}.T = r_{j2}.T = \ldots$

First, only the same type of references can be compared. Second, the type of the references' neighbors should also be the same. For instance, the neighbors of *W. Wang* in paper record **P2** are $Nbr(W.Wang(P2)) = \{J.Yang(P2), RichardMuntz(P2)\}$ rather than the set like $\{VLDB(P2), J.Yang(P2)\}$.

Multi-type Relational Similarity Measures

The multi-type relational similarity is computed by combining the similarity scores of all the relationships the pair of references is involved as follow:

$$sim_R(r_i, r_j) = w_1 * sim_{R_1}(r_i, r_j) + w_2 * sim_{R_2}(r_i, r_j) + \ldots \tag{3.4}$$

in which the $R_1, R_2, \ldots$ refer to the relationships between r_i and r_j. $w_1, w_2, \ldots$ are the weights for each type of relationships. The same as the attribute similarity measure, the weights for each type can either be learned from training data, or manually set by domain experts. Some relationships have higher weight than others, such as *writtenBy* relationship which is more important than the co-authorship since sharing a paper is a stronger proof than sharing a co-author.

3.4.4 Hybrid Measures

In some cases, neither of the above similarity measures works well separately. To achieve higher accuracy, we use the hybrid measure proposed in [8] which combines the attribute similarity measure and relational similarity measure. Let $sim(r_i, r_j)$ be the similarity of reference r_i and r_j, it is formulated as follow:

$$sim(r_i, r_j) = \mu * sim_A(r_i, r_j) + (1 - \mu) * sim_R(r_i, r_j), \qquad 0 \leq \mu \leq 1 \tag{3.5}$$

in which the attribute similarity $sim_A(r_i, r_j)$ is computed through Function 3.1, and relational similarity $sim_R(r_i, r_j)$ is computed by Function 3.3 or Function 3.4. μ is the weight of attribute similarity and $(1 - \mu)$ is the weight for relational similarity. We can see that the similarity is attribute-based if μ is set to 1.0, and fully relation-based if μ is set to 0.0;

3.5 Constraints Enforcement

Up to this point, we always seek for positive evidence either from attributes or relationships for entity resolution. Sometimes, negative evidence can also help with the resolution. Constraints enforcement [25] is one part of the entity resolution system that uses these negative evidence (which is referred to as constraint) under certain circumstances to prevent from making false decisions. In bibliographic domain, a common constraint is that authors in one paper are distinct from one another. So if two references share a hyper-edge, they should not be merged even their similarity score is high. Also, in the citation network, one paper should not cite a paper twice and papers published *later* than this one. The constraints vary in different domains. In our approach, we check the constraints on authors during the clustering process and set clusters that violate the constraints to be non-mergeable by either removing them from each other's set of potential duplicates or set their similarity score to 0.0.

3.6 Summary

For multiple types of references, relationships between different types of references can help resolve these references simultaneously. In this chapter, we formalize the multi-type entity resolution problem, and introduce the hyper-edge graph model for the relational clustering algorithm. Also we describe the agglomerative hierarchical clustering algorithm and the clustering process including the bootstrapping process for boosting connections and relational clustering process that uses relationships to improve the entity resolution.

For the pair-wise decisions during the clustering process, the correctness of the decision is greatly dependent on the similarity measures employed. The attribute similarity measures and relational similarity measures used in our approach, and a hybrid measure that combines these two similarity measure are introduced in this chapter. In addition, we discuss the constraints enforcement involved during the clustering process.

In the next chapter, we will introduce various methods for improving the efficiency of our entity resolution approach.

Chapter 4

Efficiency of Relational Entity Resolution

The previous chapter describes the multi-type entity resolution algorithm which mainly focuses on improving the accuracy. While high accuracy is crucial for entity resolution, the efficiency is also an important aspect many applications are concerned about, especially when handling large datasets. In this chapter, we introduce approaches aiming to improve the efficiency of the algorithm. Section 4.1 introduces two blocking methods that help reduce unnecessary comparisons, and Section 4.2 describes the improvement over a relational clustering algorithm.

4.1 Blocking

As introduced before, the entity resolution is an iterative process of matching and merging references. For each reference, its potential duplicates can vary greatly depending on different standards. In principle, every reference in a dataset can be considered as a potential duplicate of others. For a dataset with n references, it will result in $O(n^2)$ comparisons which are too high and impractical for large datasets with millions of references such as the ReaSoN dataset, and even worse if the similarity measures are expensive and the dimensions of data attributes are high. In reality, the average number of duplicates per entity is far less than the total number of references in most datasets. This means that most of the comparisons are redundant. In this section, we use a technique named *blocking* to reduce these redundant comparisons.

Blocking, also referred as *canopies* [49], is a technique that uses inexpensive approximate similarity measures to split a large dataset into small overlapping partitions. References that are strongly similar according to the similarity measures are grouped together in the same partition, and references that are less similar will be distributed into different partitions. We consider all the references in the same partition as potential duplicates of one another and references in different partitions as distinct. Also, we only compare the references within the

same partition so that the number of comparisons can be greatly reduced and the efficiency of the algorithm gets improved. Note that the overlapping partitions allow one reference to appear in several partitions, by which we can achieve higher recall rate. For example, suppose *Jiong Yang* is a potential duplicate of *Jiong Yan*, and *Jiong Yan* is a potential duplicate of *Jiang Yan*; However, *Jiong Yang* and *Jiang Yan* may not be considered as potential duplicates by the blocking filter since both their first names and last names are different. To handle this case, we put *Jiong Yan* in both partitions so that we can still group *Jiong Yang* and *Jiang Yan* via *Jiong Yan* if later on we consider them as true duplicates.

A good blocking approach requires a good approximate similarity measure which can efficiently group the potential duplicates and separate unrelated references. For references with string attributes, most inexpensive similarity measures are based on the inverted index, which is a data structure commonly used in the information retrieval area. The inverted index stores the mapping from tokens to the documents containing them, and through the mapping, we can retrieve the documents containing the given terms efficiently. In the following part, we describe two inverted-index-based approximate string similarity measures, a simplified one for author references, and a complicated one for both paper references and author references.

4.1.1 Naive Inverted-Index-Based Blocking

Considering that most author names have at most three parts: first name, middle name, and last name, two names sharing the same initial first name and the full last name are very likely to refer to the same author. Using this observation, we can define our blocking filter for author references as follow: author reference r_i is similar to author reference r_j if and only if $r_i.A$ and $r_j.A$ have the same initial first name and full last name.

To filter author references using an inverted index, we format each name as *F-LAST* in which F is the initial first name and $LAST$ is the full last name. The *terms* in the index are names in format *F-LAST* and the *documents* are the references with the same *F-LAST* name. Here the document id is the reference id and the content of the *document* is the value of their attributes. The advantage of this method is that we can retrieve the potential duplicates for one reference efficiently and do not need to compute any similarity (the similarity measure is built into the inverted index). And one drawback is that we have to sacrifice some accuracy for datasets with spelling errors.

4.1.2 Vector Space Model (VSM) Based Blocking

Although the naive inverted-index-based blocking approach works well for author names, it is not suitable for longer strings like paper titles. In this section we introduce the VSM-based blocking technique. Vector Space Model [58] is an algebraic model that represents

documents by vectors. Each vector consists of weights of terms in the document which are computed using the tf-idf weighting scheme [58]. Given documents d_1 and d_2, let $\mathcal{V}(d_1)$ and $\mathcal{V}(d_2)$ be the vector representations of documents d_1 and d_2 respectively. The similarity of d_1 and d_2 is computed as the cosine similarity of the two vectors [48]:

$$sim(d_1, d_2) = \frac{\mathcal{V}(d_1) \cdot \mathcal{V}(d_2)}{\mid \mathcal{V}(d_1) \mid\mid \mathcal{V}(d_2) \mid} \tag{4.1}$$

And the weights of terms in the vectors are computed using the tf-idf [48] as follow:

$$tf_{t,d} = \frac{\mathcal{N}_{t,d}}{\sum_{k=0}^{n} \mathcal{N}_{t_k,d}} \tag{4.2}$$

$$idf_t = \log \frac{\mid D \mid}{df_t} \tag{4.3}$$

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t \tag{4.4}$$

in which t is the term, d is the document, D is the document set, n is the total number of terms in d, $\mathcal{N}_{t,d}$ is the frequency of t appeared in d, and df_t is the number of documents that contain term t.

For paper references or author references, their attributes are considered as the *documents*, then the potential duplicates of a reference r_i can be retrieved using the VSM-based approach as the following process. First, we retrieve from the inverted index the *documents* that share common terms with r_i. Then we compute the similarity score of reference r_i and the retrieved *documents* using the above VSM-based similarity measure and rank these *documents* by the similarity scores. The final potential duplicates of reference r_i are the *documents* with similarity score greater than a given threshold.

The advantage of this approach is that it ranks the retrieved references according to their relevance to reference r_i so that the potential duplicates retrieved are relevant, while the drawback is that the approach is more expensive than the naive inverted-index-based approach because it needs more computations such as the tf-idf weights and the cosine similarity.

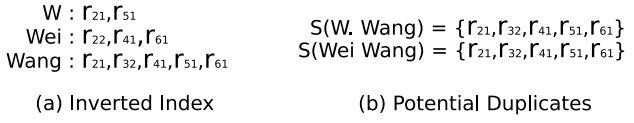

Figure 4.1: The indexing and blocking results for the example dataset.

Figure 4.1 shows a blocking example in which (a) shows the inverted index built for the author references and (b) shows the potential duplicates of two author references.

The blocking methods improve the efficiency by reducing redundant comparisons prior to the phase of entity resolution. To achieve higher improvement, we then focus on the entity resolution phase and optimize the clustering process. In next section, we introduce an optimized iterative clustering process over the approach of Bhattacharya and Getoor.

```
1  Find similar references using blocking
2  Initialize clusters using bootstrapping
3  for cluster c_i, c_j such that similar(c_i, c_j) do
4      Insert < sim(c_i, c_j), c_i, c_j > into priority queue
5  end
6  while priority queue not empty do
7      Extract < sim(c_i, c_j), c_i, c_j > from queue
8      if sim(c_i, c_j) less than threshold then stop
9      Merge c_i and c_j to new cluster c_ij
10     Remove entries for c_i and c_j from queue
11     for each cluster c_k such that similar (c_ij, c_k) do
12         Insert < sim(c_ij, c_k), c_ij, c_k > into queue
13     end
14     for each cluster c_n neighbor of c_ij do
15         for c_k such that similar (c_k, c_n) do
16             Update sim(c_k, c_n) in queue
17         end
18     end
19 end
```

Algorithm 3: The relational clustering algorithm proposed by Bhattacharya and Getoor [8].

4.2 Improvement of Iterative Clustering Process

As introduced in Chapter 3, a crucial step during the clustering process is to notify clusters that are related to the merged clusters and to re-compute the similarity between these affected clusters and their potential duplicates. Since the computation of similarity takes most of the time during the clustering process, effectively and accurately choosing pairs to be re-computed is very important to the entity resolution process. In this section, we describe the improvement over the clustering process proposed by Bhattacharya and Getoor. We start by briefly introducing their algorithm and the data structure used in their approach.

4.2.1 Clustering Algorithm of Bhattacharya and Getoor

To effectively update the similarity of references, a max priority queue [23] is used as their underlying data structure. Max priority queue is a data structure used to organize a set of elements in a binary tree structure. Each element in the priority queue consists of two clusters and their similarity score, and the element with the highest similarity score is always kept at the top of the queue so that the pair of most similar references can be obtained directly. Figure 4.2(a) lists a priority queue for our example dataset. We can see that the entry with the highest similarity score is always at the top of the queue.

Algorithm 3 lists the algorithm proposed by Bhattacharya and Getoor [8]. After the blocking and bootstrapping steps (Lines 1-2), a priority queue is created and initialized with all pairs of potential duplicates and their similarity score (Lines 3-5). Then the clus-

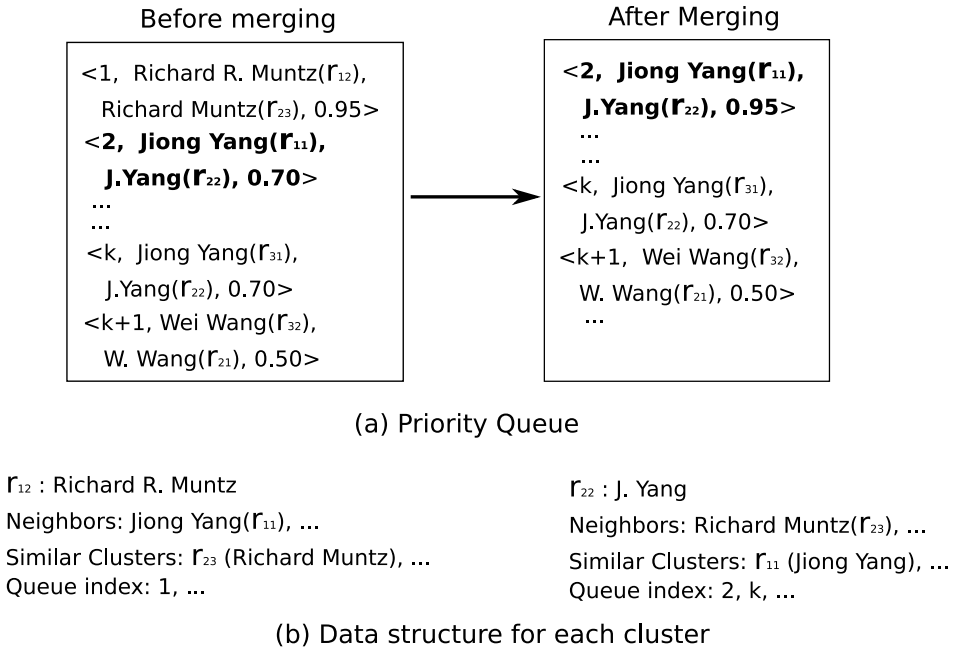

Figure 4.2: The priority queue and data structure used in BG's approach. Each entry in the priority queue contains an entry index, a pair of clusters and their similarity.

tering process is repeated until the priority queue becomes empty (Line 6) or the maximum similarity in the queue is less than the given threshold (Line 8). Lines 11-18 give the steps for notifying similar clusters and updating the similarity of clusters in the queue.

To propagate the merging results and update the similarity efficiently, three additional lists are maintained for each cluster c_i: a list of similar clusters that contain potential duplicates of references in c_i, a list of neighboring clusters that share hyper-edges with references in c_i, and a list of indexes to their entries in the priority queue for cluster c_i. The first two lists are used to access the similar clusters and the neighboring clusters of the merged clusters; And the third list is used to update the similarity score of the affected clusters. Figure 4.2 lists all the data structures used in their approach, Figure 4.2(a) is the priority queue, in which each entry contains an entry index, two clusters, and their similarity. Figure 4.2(b) gives two examples of the data structure maintained for each cluster, which include the cluster id, attributes, a list of neighbors, a list of similar clusters and a list of indexes to the entries in the priority queue.

4.2.2 Drawbacks of Bhattacharya and Getoor's Algorithm

Apparently, Bhattacharya and Getoor's method is time efficient for retrieving clusters and updating similarity. This is a good strategy as long as the dataset is small and can be loaded into the main memory for processing. However, for large datasets and systems with limited memory, this strategy may not work properly because too much information needs to be

Figure 4.3: Two examples for the data structure used in our approach.

maintained for each cluster. To reduce the memory consumption of the clustering algorithm, we optimize their clustering process by eliminating some unnecessary data structures.

In Algorithm 3, when two clusters are merged, the similar clusters and neighboring clusters of the merged clusters are re-evaluated immediately (Line 11 and Line 14 in Algorithm 3), and the similarity between them and their potential duplicates is re-computed and then updated in the priority queue (Line 11-18). However, the similarity of a pair is not used until its entry is extracted from the top of the priority queue. Furthermore, the similarity score becomes useless after it is used for making decisions. This means that the computation of the similarity score can be postponed until the moment we need them, and also the similarity score does not need to be maintained.

Based on the above observation, we replace the priority queue with a status for each cluster that indicates the availability of new information, and postpone the computation of similarity to the time it is needed (the next iteration). In addition, the list of neighboring clusters is not maintained since all the neighbors can be obtained quickly through the hyper-edges of cluster c_i. Figure 4.3 lists the data structure used in our optimized clustering process, from which we can see that the priority queue, entry indexes and the list of neighbors of each cluster are eliminated. Instead, we use just a status flag to indicate the availability of new information.

4.2.3 Implementation of Our Iterative Clustering Algorithm

Algorithm 4 presents a detailed implementation of our clustering algorithm. In the implementation, each cluster c_i has a status $c_i.hasHint$ to mark if there is any new information (which is referred to as *hint*) available. For example, when *Richard R. Muntz* in *P1* and *Richard Muntz* in *P2* are merged, the merging result (*hint*) is propagated to their neighbors, and the status of *Jiong Yang* in *P1* and *J. Yang* in *P2* is updated with *hasHint* set to *true*.

The input of the clustering algorithm is the set of cluster generated from the bootstrapping and the output is a final cluster set with all the true duplicates resolved. The algorithm iterates the set of cluster repeatedly until no more clusters can be merged. In each iteration, we go through the set of clusters once. For each cluster c_i, Lines 3-5 check if there is any new *hint* for cluster c_i available. If not, the algorithm continues to handle the next cluster; Otherwise, the cluster c_i is compared with its potential duplicates s_i (Line 8). For the pairs

```
input  : C : Output of the bootstrapping
output: C : Final cluster set
while there is new clusters merged do
    foreach c_i ∈ C do
        if c_i.hasHint = false then
            continue;
        end
        //S_i is the set of potential duplicates of c_i.
        foreach s_i ∈ S_i do
            simScore ← similarity(c_i, s_i) ;
            if simScore ≥ threshold then
                newCluster ← merge(c_i, s_i);
                add(C, newCluster);
                remove(C, c_i);
                remove(C, s_i);
                //Nbr(newCluster) is a set of neighboring clusters of newCluster.
                foreach n_i ∈ Nbr(newCluster) do
                    n_i.hasHint = true;
                    foreach s_ni ∈ S(n_i) do
                        s_ni.hasHint = true;
                    end
                end
                foreach s_j ∈ S(newCluster) do
                    s_j.hasHint = true;
                end
                //Here we replace c_i with newCluster.
                c_i ← newCluster;
            end
        end
        //The new hint has been used, so set the hasHint to false.
        c_i.hasHint = false;
    end
end
```

Algorithm 4: Iterative Relational Clustering

with similarity score greater than the threshold, we first merge them into a new cluster and then remove them from the cluster set $\mathcal{C}$ (Lines 10-13).

The key difference between our clustering process and that of Bhattacharya and Getoor is shown in Lines 15-23. When traversing the neighbor sets (Lines 13-18), we just mark the *hasHint* of affected clusters as *true* rather than computing the similarity between them and their potential duplicates immediately. For each of the neighboring clusters, we further update the status of their similar clusters (Lines 17-19) so that these clusters are re-considered in the next iteration. Note that we stop the resolution propagation at the similar clusters of the neighbors. On one hand, the neighbors and their similar clusters may not be merged and then the results do not need to be propagated further; On the other hand, if some neighbors and their potential duplicates are merged, it should be their responsibility to propagate their resolution results. The same as the neighboring clusters, we only update the status of similar clusters of c_i (Lines 21-23) of the merged clusters.

4.3 Summary

Efficiency is another important indicator of a good entity resolution approach besides the accuracy. For the pairwise comparison based clustering approach, reducing the redundant comparisons and optimizing the clustering process are two efforts we can make to improve the efficiency of our approach. In this chapter, we first describe two blocking methods to reduce the redundant comparisons, one for author references, and one for both paper references and author references. Then we analyze the clustering algorithm introduced by Bhattacharya and Getoor, and propose an iterative clustering algorithm to improve the efficiency by reducing the usage of additional data structures. In the next chapter, we will give the experimental evaluation about the accuracy and efficiency of our entity resolution approach.

Chapter 5

Experimental Evaluation

In this chapter, we describe experiments carried out for evaluating the accuracy, efficiency and scalability of our entity resolution algorithm, and discuss the results. Section 5.1 introduces the datasets used for evaluation. Section 5.2 describes the dataset preprocessing. Section 5.3 introduces the metrics used for evaluating the algorithms. And Section 5.4 presents and analyzes the results of these experiments.

5.1 Datasets

The datasets used in our experiments can be divided into two categories: datasets for benchmarking, and datasets for scalability measurement. The datasets for benchmarking contain references to be resolved and manually labeled answers. The datasets for scalability measurement contain references but no ground truth is provided.

Table 5.1 lists the basic information of these datasets. Columns *papers* and *authors* list the number of paper references and author references, and columns *unique papers* and *unique authors* give the number of entities in the ground truth for paper and author respectively. Columns *venues* and *affiliations* show the number of references for venue and affiliation. N/A indicates that the number for the current column is unknown in the given dataset. The first three datasets: CiteSeer, arXiv and DB, are used for benchmarking. The datasets DBLP, ACM DL, ACM Citation and Google Scholar constitute the ReaSoN dataset which is used for measuring the large-scale entity resolution approach. More details about these datasets are given below.

CiteSeer Dataset

CiteSeer is a scientific digital library that focuses primarily on the literature in computer and information science [1]. It was developed at the NEC Research Institute, Princeton, New Jersey and now hosted and maintained at Pennsylvania State University. The dataset

[1] `http://citeseer.ist.psu.edu`

	papers	unique papers	authors	unique authors	venues	affiliations
CiteSeer	1,504	875	2,892	1,165	N/A	N/A
arXiv	29,555	N/A	58,515	9,200	N/A	N/A
DB	116,455	N/A	291,006	N/A	96,706	N/A
DBLP	1,221,236	N/A	3,082,861	N/A	1,221,138	N/A
ACM DL	1,217,844	N/A	2,467,773	N/A	945,388	1,158,426
ACM Citation	5,886,251	N/A	12,823,152	N/A	5,574,378	N/A
Google Scholar	68,674	N/A	162,663	N/A	48,925	N/A
ReaSoN	8,394,005	N/A	18,536,449	N/A	7,789,829	1,158,426

Table 5.1: Datasets used for evaluation. Datasets CiteSeer, arXiv and DB are used for benchmarking. ReaSoN is consisted of datasets DBLP, ACM DL, ACM Citation and Google Scholar. Columns *papers, authors, venues* and *affiliations* represent references, and columns *unique papers* and *unique authors* represent entities.

we use is a small portion of the entire library with 1504 paper references and 2892 author references. All the papers and authors have been manually labeled with a cluster id that uniquely identifies the true entities of the references. The attributes available in the CiteSeer dataset include the title for papers and the name for authors. Other information like year, venue and affiliation are not provided. Table 5.2 lists a sample of the dataset.

aid	a_c_id	norm_name	full_name	a_no	pid	p_c_id	title
6	10	aamodt_a	Aamodt, A	0	673	4701	Knowledge...
10	10	aamodt_agnar	Agnar Aamodt	0	1018	6602	Case-Based...
11	10	aamodt_agnar	Agnar Aamodt	0	671	4701	Knowledge...

Table 5.2: A sample record in the CiteSeer dataset.

In the sample, *aid* is the *author id*, *a_c_id* is the *author cluster id*, *norm_name* is the *normalized name* with the last name followed by the first name and middle name. *full_name* is the *author name* in the paper, *a_no* is the order of the author in the paper, *pid* is the *paper id* and *p_c_id* is the *paper cluster id*. Due to space limitation, we did not list the full title. The CiteSeer dataset was originally created by Giles et al., and then hand-clustered by Aron Culotta and Andrew McCallum in University of Massachusetts, Amherst, and further cleaned by Indrajit Bhattacharya and Lise Getoor in University of Maryland. It can be download from their website [2].

arXiv dataset

arXiv is an e-print service [3] which provides access to papers in Physics, Mathematics, Non-linear Sciences, Compute Science, Quantitative Biology, and other fields. It is owned and operated by Cornell University. The dataset we use is a subset of publications in high energy physics. It was originally used in KDD Cup 2003 [1]. David Jensen at University of

[2] `http://www.cs.umd.edu/projects/linqs/projects/er/DATA/citeseer.dat`
[3] `http://arxiv.org`

Massachusetts, Amherst labeled the ground truth of authors on part of the dataset which was further cleaned by Indrajit Bhattacharya and Lise Getoor in University of Maryland. The arXiv dataset contains 29,555 paper references and 58,515 name references referring to 9,200 authors. The format of the dataset is the same as that of CiteSeer dataset. The only difference is that arXiv dataset does not have paper titles available. It can be download from Lise Getoor's website [4].

DBLP

DBLP [5] provides bibliographic information on major computer science journals and proceedings. Initially it focused on papers in *DataBase systems and Logic Programming* and now gradually expands to other fields of computer science [6]. DBLP is maintained by Michael Ley at University of Trier. As of June 2009, the dataset contains more than 1.2 million bibliographic records. The version we use contains the records up to June 2009 with 1,221,236 paper references, 3,082,861 author references and 1,221,138 venue references. Each record contains attributes of papers like title, author name, year, venue and so on. A sample record of DBLP is given in Figure 5.1. A snapshot of DBLP can be downloaded from their website [7].

```
<dblp>
  <inproceedings key="conf/sigmod/Ramaswamy08" mdate="2008-06-10">
  <author>Sridhar Ramaswamy</author>
  <title>Extreme data mining.</title>
  <pages>1-2</pages>
  <year>2008</year>
  <booktitle>SIGMOD Conference</booktitle>
  <ee>http://doi.acm.org/10.1145/1376616.1376617</ee>
  <crossref>conf/sigmod/2008</crossref>
  <url>db/conf/sigmod/sigmod2008.html#Ramaswamy08</url>
  </inproceedings>
</dblp>
```

Figure 5.1: A sample record in the DBLP dataset.

The DBLP dataset consists of a set of records with one paper per record. Each paper has a key that uniquely identifies it in the dataset. All the author names in DBLP come with the full first name and last name. Initial or full middle names are also given for some authors. *Booktitle* is the venue where the paper is published, and the *crossref* field refers to the bibliography of the conference. More details about DBLP is introduced by Michael Ley [47].

[4] http://www.cs.umd.edu/projects/linqs/projects/er/DATA/arxiv.dat
[5] http://www.informatik.uni-trier.de/~ley/db/
[6] http://www.informatik.uni-trier.de/~ley/db/about/faqdblp.html
[7] http://dblp.uni-trier.de/xml/

ACM Digital Library

ACM Digital Library (ACM DL) [8] is a collection of publications from ACM journals, newsletter articles and conference proceedings on computer science. The dataset we use is compiled of bibliography information of most publications in the digital library, and contains 1,217,844 paper references, 2,467,773 author references, 945,388 venue references, and 1,158,426 affiliation references. For each bibliography record, ACM DL provides information like paper title, author name, year, venue, DOI and so on. Appendix 7.2 lists a sample record in ACM DL and gives detail description of the record. In addition to the basic information that other datasets provide, ACM DL also contains affiliations for some authors and citations of papers. The ACM DL dataset is a proprietary dataset which we had a license to use for 2 years.

ACM Citation

ACM citation is a dataset consisted of the papers cited by papers in ACM DL. As described above, the citations of a paper are a list of citation string rather than a formatted bibliography record. To build the dataset, we first extract from the citation strings the metadata information of papers like titles, author names and venues using the Flux-Cim algorithm[66]. After the conversion, the ACM Citation dataset obtained contains 5,886,251 paper references, 12,823,152 author references and 5,574,378 venue references. Figure 5.2 lists a sample record.

```
<node>
  <cite_id>252</cite_id>
  <string>Blandford A., Butterworthb R., Curzonb P., Models of interactive systems: a case study on
programmable user modelling, International Journal of Human-Computer Studies, vol. 60 (2004), 149--200,</
string>
  <author>
    <name>A Blandford</name>
  </author>
  <author>
    <name>R Butterworthb</name>
  </author>
  <author>
    <name>P Curzonb</name>
  </author>
  <title>Models of interactive systems: a case study on programmable user modelling</title>
  <venue_fullname>International Journal of Human-Computer Studies</venue_fullname>
  <year>2004</year>
  <pages>149--200</pages>
  <volume>vol.60</volume>
</node>
```

Figure 5.2: A sample record in ACM citation dataset.

The *string* is the original citation string in the ACM DL, and the *author name*, *paper title*, *venue_fullname*, *year*, *pages* and *volume* are the attributes extracted from the citation

[8] `http://portal.acm.org/dl.cfm`

string. The author name in ACM Citation datasets are provided with the full last name and initial or full first name, middle names are available for some of them.

Google Scholar dataset

Google Scholar is a freely-accessible scientific paper search engine developed by Google. It provides scholars with a vast volume of information about scientific literature across several fields. Given the title of a paper, Google Scholar can retrieve a list of related publications ordered by a relevance descending ranking. The snippet of each result contains the number of citations the paper received and hyper-links to those citations. Following the link, we can further retrieve information about those citations which include title, year, author names and probably venues. Due to length limitation, Google Scholar may omit or shorten some information like venue so that the information of papers may not be complete. In this thesis, we construct a dataset through Google Scholar. We first compile a collection of papers in the Database area from DBLP. Then for each paper P_i, we retrieve the papers that cite P_i, and extract their information. A sample of extracted records is shown in Figure 5.3. This is also a proprietary dataset obtained through semi-automated crawling which we have no right to release.

```
<record>
   <title>FLASH: A Language-Independent, Portable File Access System.</title>
   <year>1980</year>
   <id>"conf/sigmod/AllchinKW80"</id>
   <citedby>
      <cite>
         <title>The Starburst Long Field Manager</title>
         <other>T Lehman, B Lindsay - Proc. 1989 VLDB Conference, Amsterdam, Netherlands, Sept,
1989 - books.google.com</other>
      </cite>
      <cite>
         <title>Efficient database access from Prolog</title>
         <other>S Ceri, G Gottlob, G Wiederhold - IEEE Transactions on Software Engineering, 1989 -
doi.ieeecomputersociety.org</other>
      </cite>
      <cite>
         <title>Research in knowledge base management systems</title>
         <other>G Wiederhold, SJ Kaplan, D Sagalowicz - ACM SIGMOD Record, 1981 -
portal.acm.org</other>
      </cite>
      <cite>
         <title>DATA DEFINITION FACILITY OF CRITIAS</title>
         <other>X Qian, G Wiederhold - The 4th International Conference on Entity-Relationship  , 1985
- Order from IEEE Computer Society</other>
      </cite>
      <cite>
         <title>An Efficient Implementation of Search Trees on 0 (log N) Processors</title>
         <other>MJ Carey, CD Thompson - 1982 - eecs.berkeley.edu</other>
      </cite>
   </citedby>
</record>
```

Figure 5.3: A sample record in Google Scholar dataset.

In the Google Scholar dataset, the *id* is the *key* in the DBLP record. The *cite*s in the *citedby* part refer to the papers that cite the given paper. The author name and venue

name need to be further split and processed. After the cleaning, we construct a dataset that contains 68,674 paper references, 162,663 author references and 48,925 venue references.

ReaSoN dataset

The ReaSoN dataset is composed of the DBLP, ACM DL, ACM Citation and Google Scholar datasets. The whole dataset contains 8,394,005 paper references, 18,536,449 author references, 7,789,829 venue references and 1,158,426 affiliation references.

DB dataset

With no ground truth labels available, measuring the accuracy of the clustering algorithm on the ReaSoN dataset is not practical, especially for such a large dataset. Instead of accurately evaluating the algorithm, we adopt an approximate measurement by measuring an approximate accuracy of the algorithm on a small sample dataset of the ReaSoN dataset. The papers in the sample dataset are chosen from a number of database venues (Appendix 7.2). The correctness of the clustering on this sample dataset is judged manually.

The process to construct the sample dataset is straightforward; We collect a list of venues in the database community, and then add papers in these venues from different sources to the sample dataset. The final dataset (DB dataset) contains 116,455 paper references, 291,006 author references, and 96,706 venue references. To measure the accuracy, we randomly choose several last names, and pick out the author references with these last names. Then, we label these author references manually. The partial ground truth for DB dataset contains 140 entities and 1518 references.

Other properties of these datasets

The basic information of these datasets like the domain, source, and format and so on is given above. For entity resolution, there are some other properties that may help determine parameters for clustering. We list some of them for the three datasets in Table 5.3 and give a detailed explanation for each property. The ReaSoN dataset is not listed because it is not used for the benchmark.

	CiteSeer	arXiv	DB
average_authors/paper	1.92	1.98	2.50
percent_single_author_ref	22.39%	19.62%	10.10%
percent_amb_author_ref	15.81%	9.94%	10.10%
percent_amb_single_author_ref	3.18%	0.0%	0.98%
percent_amb_author_entity	0.26%	4.11%	13.57%

Table 5.3: Some statistics of the datasets.

Average number of author references per paper (average_authors/paper).
For relational clustering, authors are connected through the co-authorship. Each paper

acts as a link and the number of author references in the paper determines the number of relationships. Thus the larger the average number of author references in each paper, the more relationships an author reference will be involved on average, and the more we can gain from using these relationships.

Percentage of author references with no collaborators (percent_single_author_ref). As just stated, more author references in a paper mean more relationships to use. In other words, the author references with no coauthors can not use the co-authorship for duplicate resolution. If there are no other types of relationships that can help in this case, the only fact we can rely on is the attributes which are the names of the author references. So counting the percentage of single author references can predict how much we need to rely on attributes.

Percentage of author references with ambiguous names (percent_amb_author_ref). A name (last name and first initial) is considered *ambiguous* if it is shared by at least two author references or entities [8]. In these datasets, many first names are provided with only the initial letter instead of the full name and this ambiguity makes the author references difficult to be distinguished from one another. So the percentage of author references with ambiguous name is an indicator of the ambiguity of datasets.

Percentage of single author references with ambiguous names (percent_amb_single_author_ref).

For author references with ambiguous name, some of them could be correctly resolved using relationships. However, for those author references with no collaborators, it will be very difficult to identify them. A high percentage of this kind of references will decrease the clustering accuracy a lot.

Percentage of author entities with ambiguous names (percent_amb_author_entity). This percentage is different from that of author references. The percentage of author entities with ambiguous name is the number of *entities* with ambiguous name divided by the total number of *entities* while the percentage of author references with ambiguous name is the number of *references* with ambiguous name divided by the total number of *references*. And these two numbers should be considered when we decide the threshold for the attribute similarity. For example, if the percentage of author entities with ambiguous name is very low, then we can lower the threshold since most of the author references refer to the same entity. Otherwise, an appropriate threshold should be chosen for similarity measures.

5.2 Data Preprocessing

As introduced above, these datasets from different sources are in different formats, some attributes in the record may have different names. For example, the ACM DL dataset uses *venue_fullname* and *venue_name* to represent venue while the DBLP dataset uses *journals*

and even *booktitle* to refer to venues. So prior to the entity resolution, we need a schema mapping to transform the different schemas into an uniform format to eliminate the inconsistency at the scheme level. Since the format of datasets is fixed, and all the mappings can be enumerated, we simply pre-process the datasets using hard-coded mappings. Figure 5.4 shows a sample record in the uniform format.

```
<node>
  <key>...</key>
  <paper_id>...</paper_id>
  <title>Simulating HCI for special needs</title>
  <type>JOURNAL ARTICLE</type>
  <year>2007</year>
  <keyword>...</keyword>
  <venue_id>241</venue_id>
  <venue_fullname>ACM SIGACCESS Accessibility and Computing</venue_fullname>
  <volume>...</volume>
  <issue>89</issue>
  <pages>...</pages>
  <author>
    <author_id>221</author_id>
    <name>Pradipta Biswas</name>
    <affi_id>231</affi_id>
    <affiliation>University of Cambridge, England</affiliation>
  </author>
  <cite>...</cite>
  <cite>...</cite>
  <abstract>...</abstract>
</node>
```

Figure 5.4: A example of a record in the common format.

In addition to the schema mapping, we assign a unique id for each reference in the dataset. As shown in Figure 5.4, there are *paper_id*, *venue_id*, *author_id* and *affi_id*, each of which has encoded its data source and reference type in the id. Take the venue id *241* for example, the *2* indicates that the reference is from the ACM DL dataset, and *4* indicates it is a venue reference, and the left *1* is the id assigned for this venue reference. Through this encoded unique id, we can not only identify references globally, but also obtain their data source and reference type easily. In the future work, we will eliminate the *type* member of each reference and use the *id* instead to get the reference type.

5.3 Evaluation Metrics

5.3.1 Accuracy Metrics

As many other entity resolution approaches [8, 25, 43] do, we assess the accuracy of our entity resolution algorithms with *precision*, *recall*, and *F-measure*, in which the *F-measure* is defined as the harmonic mean of precision and recall. To be more specific, we use the same metric as that used by Bhattacharya and Getoor [8] since we compare the accuracy of our approach with that of their approach.

As a pairwise comparison approach, we measure the accuracy of algorithms by counting

the number of pairs of true duplicates found. Recall that the ground truth set and the answer set are actually partitions of references in the dataset. To evaluate the accuracy, we first get the set of distinct pairs of references in these clusters. For example, if a result set $\mathcal{R} = \{< r_1, r_2 >, < r_3, r_4, r_5 >\}$ contains two clusters, then the set of distinct pairs we get is $\mathcal{P} = \{(r_1, r_2), (r_3, r_4), (r_3, r_5), (r_4, r_5)\}$. We let $\mathcal{T}$ be the set of pairs of references in the ground truth, and $\mathcal{A}$ be the set of pairs of references in the answer set. Then the precision is computed as follow:

$$precision = \frac{\mid \mathcal{T} \cap \mathcal{A} \mid}{\mid \mathcal{A} \mid}$$

and Recall is

$$recall = \frac{\mid \mathcal{T} \cap \mathcal{A} \mid}{\mid \mathcal{T} \mid}$$

and F1 is

$$F1 = 2 * \frac{(precision * recall)}{(precision + recall)}$$

Note that we are only concerned about the pair-wise decisions; For example, the author entity *Jiong Yang* in the example dataset has 4 references, then there are 6 different reference pairs in set $\mathcal{T}$ by pairing each reference with others in the cluster.

One problem with this metric is that those references without any duplicate will not be counted. Thus the metric will not reflect the true precision for clusters with cardinality 1. For example, suppose we have a dataset $\mathcal{D} = \{< r_1 >, < r_2 >, < r_3 >, < r_4, r_5 >\}$, in which only r_4 and r_5 are duplicates, and the clustering algorithm obtains a result with each reference in a separate cluster $\{< r_1 >, < r_2 >, < r_3 >, < r_4 >, < r_5 >\}$. Using our pairwise metric, both precision and recall are zero; However, the truth is that we only made one mistake and the precision and recall of the algorithm should be higher.

Though with this problem, we still use the pairwise accuracy metric for the following reasons; First we can compare our result with that of Bhattacharya and Getoor; Second, the accuracy is distorted only when a large portion of references do not have duplicates in the datasets. According to the statistics, those references with no duplicates in the CiteSeer and arXiv datasets are 24.4% and 5.2% respectively which we believe do not affect the accuracy very much.

5.3.2 Performance Metrics

The other aspect we are concerned about is the performance of the entity resolution approach on various datasets. To assess the performance, we measure the execution time and the memory consumption of these algorithms. For execution time, we use the CPU time for the time spent on a certain process or task. For memory usage, we measure the maximum memory consumed during an execution process. The unit is *ms* (milliseconds) in CPU time and *Mb* (Megabytes) for memory respectively. The algorithms are written in Java and run

on Sun's JVM 6.0. All execution time and memory are reported on a Dell server with two 1.9GHz Quad-Core AMD Opteron Processors and 32G of memory. Also, unless stated otherwise, all the programs are single-threaded.

5.3.3 Similarity Functions and Thresholds

For our algorithms, different similarity functions and thresholds are employed for blocking, matching and merging respectively. Appropriate parameters for the similarity functions and thresholds are very important to the accuracy and efficiency of the algorithms. For attribute similarity functions, we employ the same set of measures for the same type of references in all datasets. For author references, different weights are assigned to different parts of a name. Specifically, we set weights $w_{first} = 0.35$, $w_{middle} = 0.10$ and $w_{last} = 0.55$ when measuring the similarity of author names. For paper references, their attribute similarity is measured by the string similarity of their titles, however, we decrease their title similarity by 0.8 if the year of papers is different. For the relational similarity, we only measure the *writtenBy* relationship similarity for papers, and for authors, we measure both *writtenBy* relationship and *co-authorship* relationship. As introduced in Chapter 3, we combine the attribute similarity and relationship similarity together to form the final similarity score of two references. The weights for attribute and relationship similarity are different for different datasets and we will specify them in each of the experiments.

Besides the weights for similarity measurement, the threshold is also another key parameter for blocking and entity resolution. For blocking, the blocking threshold is set to 0.2 for the CiteSeer dataset and the arXiv dataset, and to 0.15 for the DB dataset. The detailed methods for determining the blocking thresholds are given in the experiment section. As for entity resolution, the threshold varies depending on different datasets since the ambiguity of these datasets is different, and we will list these thresholds in each experiment.

For these parameters mentioned above, we manually tune each of them and choose the value that can help obtain the best accuracy and efficiency. In addition, we compare our results with that of Bhattacharya and Getoor. For accuracy, we use the results reported by them. For efficiency, to be fair, we re-execute their code on these datasets and use the results reported by our machine.

5.4 Experiments and Results

In this section, we describe the experiments for evaluating the accuracy and efficiency of algorithms proposed in our approach and analyze their results.

5.4.1 Accuracy of Bootstrapping

The main goal of bootstrapping is to initialize connections for the initial graph models and provide relational evidence for relational clustering. Since the result of bootstrapping will be propagated to the clustering step, the precision of the bootstrapping step is very important and should be maintained at a high value.

Datasets \ Accuracy	Precision	Recall	F1
CiteSeer	0.9996	0.9244	0.9605
arXiv	0.9936	0.9120	0.9510
DB	0.9944	0.7403	0.8487

Table 5.4: Accuracy of the bootstrapping step on CiteSeer, arXiv and DB dataset.

Figure 5.4 shows the accuracy of the bootstrapping step on different datasets. The merging thresholds of bootstrapping for datasets CiteSeer, arXiv and DB are 0.50, 0.59 and 0.66 respectively. For datasets CiteSeer and arXiv, the precision of bootstrapping is maintained above 99%, and the highest recall obtained is above 90%, of which the recall on CiteSeer is slightly higher. We can see that attribute-based approaches can achieve high accuracy for some datasets. The reason is that the *ambiguity* of these two datasets is very low. Recall that in CiteSeer dataset, only 3 of 1165 entity names are ambiguous and in arXiv dataset 4.11% names are ambiguous. Thus the threshold for the attribute similarity can be set to a low value to tolerate some false positives. However, on DB dataset the recall is only 74.03% which is much lower than that on CiteSeer and arXiv. In Table 5.3, we can find that 13.57% of the names in DB dataset are ambiguous, which means that setting a lower threshold will bring in many ambiguous references that do not refer to the same entities, and thus result in lower precision.

Though we have obtained high accuracy for some datasets during the bootstrapping step, a higher accuracy should be achieved using relationships according to our hypothesis. In the next section, we will measure the accuracy of the multi-type relational clustering algorithm and verify if the relationships can further improve the result of entity resolution.

5.4.2 Accuracy of Multi-Type Relational Clustering

In this experiment, we compare three clustering algorithms: attribute-based clustering, single-type relational clustering and multi-type relational clustering. The single-type relational clustering is applied only on author references, and the multi-type relational clustering is applied on both the paper references and author references. The accuracy of these algorithms on the three benchmarking datasets is reported below.

The following tables show various results on the three datasets. *AC* means *Attribute-based Clustering* and *RC* means *Relational Clustering*. Single-type RC is the relational

clustering on single type of reference and Multi-type RC is the relational clustering on multiple types of references. *BG's RC* is the relational clustering algorithm proposed by Bhattacharya and Getoor.

Accuracy / Datasets	Precision	Recall	F1
AC	0.9963	0.9840	0.9901
Single-type RC	0.9996	0.9912	0.9955
Multi-type RC	0.9988	0.9940	0.9964
BG's RC	N/A	N/A	0.9950

Table 5.5: Accuracy of different clustering algorithms on CiteSeer.

Table 5.5 shows the accuracy results of different algorithms on CiteSeer dataset. The weights for the relationship similarity and attribute similarity are 0.6 and 0.4 respectively, and the merging threshold is 0.20. The first three rows list the results of our algorithms, and the last row shows the best accuracy reported by Bhattacharya and Getoor [8].

As the results show, the attribute-based clustering can achieve over 99% accuracy (F1 measure). However, the single-type RC can achieve higher accuracy (99.55%) which indicates that more true duplicates can be found using relationships. For example, in the CiteSeer dataset, references *23. K. Marriott* and *K. Marriott* refer to the same author entity, but cannot be identified using just the name since their first names are different (23 vs K). However, the common neighbor *P. Stuckey* they share (co-authorship) can provide evidence for their merging.

We have already seen that references with coauthors can use the co-authorship to help resolve duplicates. Then a natural question is how to resolve the pairs of references that do not share any coauthors. Results in multi-type RC tell us that using multiple types of relationships can further improve the accuracy. In some cases, the potential duplicates that do not share coauthors may share papers. In other words, the merging of two paper references can infer that all the authors in these two papers are identical respectively. Finally, we compare our results with that reported by Bhattacharya and Getoor [8]. From the results, we can see the F1 measure of our multi-type RC algorithm outperforms BG's RC though not too much. Note that all algorithms show very high accuracy on the CiteSeer dataset.

Accuracy / Datasets	Precision	Recall	F1
AC	0.9868	0.9636	0.9750
Single-type RC	0.9904	0.9748	0.9826
BG's RC	N/A	N/A	0.9850

Table 5.6: Accuracy of different clustering algorithms on arXiv.

Table 5.6 lists the accuracy of algorithms on the arXiv dataset. The weights for the relationship similarity and attribute similarity are 0.75 and 0.25 respectively, and the merg-

Datasets \ Accuracy	Precision	Recall	F1
AC	0.9944	0.7403	0.8487
Single-type RC	0.9433	0.9274	0.9353
Multi-type RC	0.9426	0.9366	0.9396

Table 5.7: Accuracy of different clustering algorithms on DB.

ing threshold is 0.235. Since the paper title is not provided, no result of multi-type RC is reported. The same as that on CiteSeer, the single-type relational clustering outperforms the attribute-based clustering in both precision and recall.

Table 5.7 lists the accuracy of different algorithms on the DB dataset. The weights for the relationship similarity and attribute similarity are 1.00 and 0.00 respectively which means we use only the relationship similarity to obtain the best accuracy, and the merging threshold is 0.012. We can see that the accuracy is increased from 84.87% to 93.53% through the relational clustering. Especially the recall is greatly improved. To explain why the result on the DB dataset is so different from that of the CiteSeer and arXiv datasets, we revisit some properties shown in Table 5.3. The average number of authors per paper in DB dataset is 2.50 which is higher than the 1.92 and 1.98 of the other two datasets, also the percentage of author references without collaborators is the lowest of the three datasets (10.10% vs 22.39% and 19.2%). This two numbers indicate that the references in DB dataset are more connected and less isolated. So more relationships are available and the accuracy of relational clustering (Single-type RC and multi-type RC) is much higher than that of the AC. Notice that the multi-type RC does not outperform single-type RC very much. We gain the recall while sacrifice the precision. This means that the merging of paper references does not contribute much to the resolution of author references, or put it another way, most of the papers with duplicates have two or more author references and the author references can use the co-authorship relationship instead of the *writtenBy* relationship to resolve duplicates.

In the above experiments, we measure the accuracy of the bootstrapping step and our relational clustering algorithms. In the following sections, we will evaluate the efficiency of the blocking methods and the modified clustering process and check if they can improve the entity resolution process.

5.4.3 Blocking Methods

In the experiments for evaluating blocking methods, we first explore how the blocking threshold affects the results of blocking and the accuracy of clustering algorithms. Then we give a detailed comparison about the two blocking methods.

We choose Lucene, a high-performance text search engine[9], to implement the *VSM-*

[9] `http://lucene.apache.org/`

Based Blocking, and HashMap to implement the inverted index for *Naive Inverted-Index-Based Blocking*. As discussed in Section 4.1, an appropriate threshold for the approximate similarity measure is important for retrieving the set of potential duplicates of each reference. A high threshold will filter out references that should exist in the set and reduce the recall of the entity resolution, while a low threshold will bring in many unrelated references and increase the number of unnecessary comparisons.

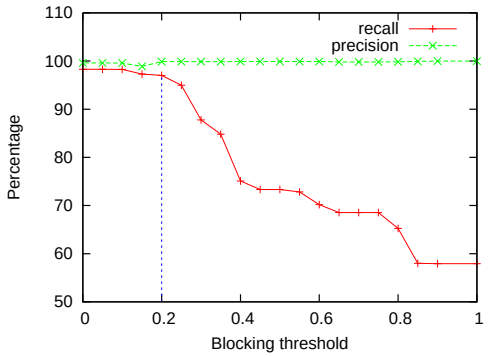

Figure 5.5: Precision and Recall of the clustering algorithm on CiteSeer dataset given different blocking thresholds.

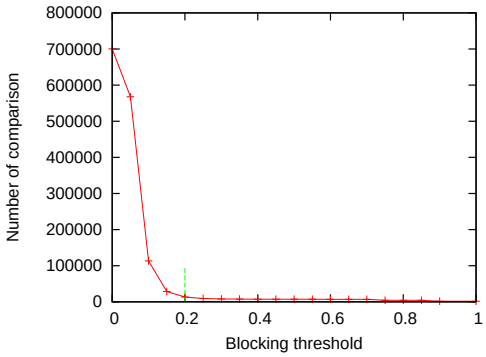

Figure 5.6: The number of comparisons during the clustering process on CiteSeer dataset given different blocking thresholds.

Figure 5.5 shows the relationship between the clustering accuracy and the blocking threshold on the CiteSeer dataset. From the figure, we can see that the precision is kept at a very high level (above 99%) for most values of the threshold and almost not affected by the varying threshold, this is because no matter how the blocking threshold changes, the clustering algorithm is always applied on the same dataset, and the only difference is that different thresholds produce different subsets of the whole dataset. Furthermore, the higher the threshold is, the less unrelated references we have in each reference's set of potential duplicates and the higher of the precision. This can be observed from the precision curve in the figure. On the other hand, the recall is decreasing greatly when the threshold becomes greater than 0.2. The reason is that higher threshold filters out more related references that should be in the set of potential duplicates. To achieve high accuracy, a threshold value around 0.2 will be a good choice for author clustering on the CiteSeer dataset.

In addition to the precision and recall, we also count the number of comparisons made during the clustering process for different blocking thresholds. Figure 5.6 reveals their relationship. The number of comparisons increases rapidly as the threshold decreases from 0.2 to 0.0, and does not change much when the threshold is greater than 0.2. This conforms to the analysis above. Thus, to balance the accuracy and efficiency of the blocking methods, we need to choose an appropriate threshold so that we can gain high accuracy with little

loss of efficiency or vice versa.

One problem with determining the blocking threshold is that the best value is always unknown until we finish running the algorithm on the dataset for all the possible thresholds. This is not an issue for those small datasets; However, for large dataset like ReaSoN, it will take several days for just one iteration on each threshold, and a precise measuring of the best threshold will not be possible. To solve this problem, we adopt an approximate method. We randomly choose a sample dataset from the large dataset and determine the threshold using the experiments above on the sample dataset.

As analyzed in Section 4.1, the *VSM-Based Blocking* does not perform efficiently on references with short attribute strings, and we propose the *Naive Inverted-Index-Based Blocking* which may outperform the VSM-Based Blocking for the author reference resolution. To compare the accuracy and efficiency of the two blocking methods and measure which one is better than the other, we apply both blocking approaches on the three datasets. Table 5.8 gives the results. Note that for each dataset, the bootstrapping and clustering algorithms and the parameters are the same for both blocking strategies.

	Precision	Recall	F1	time(s)	memory(MB)
Lucene(CiteSeer)	0.9996	0.9914	0.9955	2.47	56.74
HashMap(CiteSeer)	0.9997	0.9649	0.9820	0.68	25.63
Lucene(arXiv)	0.9904	0.9748	0.9826	50.21	619.82
HashMap(arXiv)	0.9927	0.9660	0.9792	8.35	187.60
Lucene(DB)	0.9173	0.9462	0.9318	564.70	4542.42
HashMap(DB)	0.9433	0.9274	0.9353	24.02	1185.94

Table 5.8: Accuracy and efficiency of different blocking approaches on CiteSeer, arXiv and DB dataset.

From the results in the table, we can see that the accuracy of *Naive Inverted-Index-Based Blocking* is lower than that of *VSM-Based Blocking* on CiteSeer dataset and arXiv dataset. This is because some last names with spelling errors can not be indexed using the Naive Inverted-Index-Based Blocking and results in the low recall. Although the recall is slightly lower, the precision is higher than that of the VSM-Based Blocking. This result is as expected since the Naive Inverted-Index-Based Blocking can locate the similar references more accurately. For the DB dataset, we can see that the accuracy of Naive Inverted-Index-Based Blocking is slightly higher than that of the VSM-based blocking. this is because the DB dataset is more ambiguous than the other two datasets and the potential duplicates retrieved by the VSM-Based Blocking contain many ambiguous references; This results in the much lower accuracy of the VSM-based blocking.

Though the VSM-Based Blocking outperforms the Inverted Index-Based Blocking slightly on the accuracy, their efficiency is exactly the opposite. The blocking time using Lucene on CiteSeer dataset is 2.47 seconds which is almost 4 times more than the 0.68 seconds of

HashMap. The difference on arXiv and DB datasets is much more than that on CiteSeer dataset. In addition, the memory usage of HashMap is far less than that of Lucene. In short, the VSM-Based Blocking can achieve a little higher accuracy than the Naive Inverted-Index-Based Blocking, but it requires much more time and memory during the blocking process. In applications that efficiency is more important than accuracy, we can sacrifice a little accuracy on the clustering by adopting the Naive Inverted-Index-Based Blocking approach.

5.4.4 Efficiency of Relational Entity Resolution

In addition to the blocking methods, we also measure the time efficiency and the space efficiency of the relational clustering algorithms. Here we report the execution time of the clustering algorithm on all the benchmarking datasets. The time we count is the sum of the time spent on the bootstrapping and the relational clustering.

	CiteSeer	arXiv	DB
AC	0.26	3.77	14.54
Single-type RC	0.49	7.12	70.84
BG's AC	0.60	92.30	2908.73
BG'S RC	2.70	378.37	6849.95

Table 5.9: Execution time of different algorithms for author references on datasets CiteSeer, arXiv and DB (in CPU Time).

Table 5.9 lists the clustering time of AC, Single-type RC and BG's results [8]. The execution time is reported with parameters setting for the best accuracy. For CiteSeer, not only our attribute-based clustering algorithm outperforms that of BG, but our relational clustering algorithm improves the execution time more than 5 times as that of BG and achieves better accuracy. As the number of references increase in the arXiv and DB datasets, our algorithm reduces the clustering time even more than that of BG without any loss in accuracy.

	CiteSeer	arXiv	DB
AC	103.42	996.20	1692.00
Single-type RC	119.14	998.31	2212.30
BG's AC	27.86	879.00	Out of Memory
BG's RC	40.09	1653.00	Out of Memory

Table 5.10: Memory usage of different algorithms on datasets CiteSeer, arXiv and DB (in Megabytes).

Although no memory usage is reported in Bhattacharya and Getoor's experiment [8], we re-execute their code on these three datasets on our machine and report the memory consuption in Table 5.10. The memory usage reported here is the peak memory consumed during the execution process. We can see from Table 5.10 that though our algorithms require more memory on the CiteSeer datasets, as the number of references increases, the

memory usage of BG's algorithm increases much faster than ours, especially for the relational clustering. This result can verify the improvement of the optimization of the clustering process.

	CiteSeer	DB
Single-type RC(Paper)	0.14	3.89
Single-type RC(Author)	0.49	70.84
Single-type RC(Sum)	0.63	74.62
Multi-type RC	0.56	73.16

Table 5.11: Comparison of execution time of single type clustering and multi-type clustering algorithm (in CPU time).

We have already shown that the relational clustering can further improve the accuracy over the attribute-based approaches, here we compare the efficiency of the multi-type RC and the single-type RC. Table 5.11 lists the results on CiteSeer and DB datasets. The arXiv dataset is not included here because no paper titles are provided. Rows 1 and 2 are the clustering time of single type RC for paper references and author references in CiteSeer and DB datasets respectively. Row 3 is the total execution time of the relational clustering algorithm spent on different types of references. The last row lists the time of multi-type clustering on the two datasets.

Intuitively, the multi-type RC should take more time than that of single-type RC. In addition to the time spent on each single type of references, multi-type RC needs to compute relational similarity of different relationships. However, the results show that the multi-type RC spends even less time than that of the single-type RC. The reason is that using different relationships for entity resolution sometimes can reduce the expensive comparisons of relationships between the same type of references. For example, if two papers are considered as the same paper, then all the authors of these two papers are the same and there is no need to use co-authorships several times (each time for one author) to resolve the authors. To sum up, the multi-type relational clustering algorithm can improve the accuracy of the clustering without any loss of time efficiency or even with improvement.

5.4.5 Scalability of Relational Entity Resolution

Although we have improved the efficiency and memory usage of our algorithms, we are not sure if these algorithms are scalable when time or memory resources are limited and the datasets to be resolved are very large. To evaluate the scalability, we measure the execution time and the maximum memory usage of the algorithms on datasets with different sizes. Figure 5.7 shows the execution time of the algorithm on subsets of the arXiv and DB datasets. From the figure, we can see that the time of clustering increases almost linearly with the size of datasets. Also, in Figure 5.8, the maximum memory consumed by the

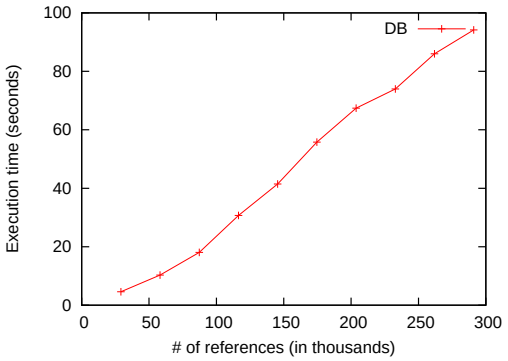

Figure 5.7: Execution time of the clustering with different number of author references.

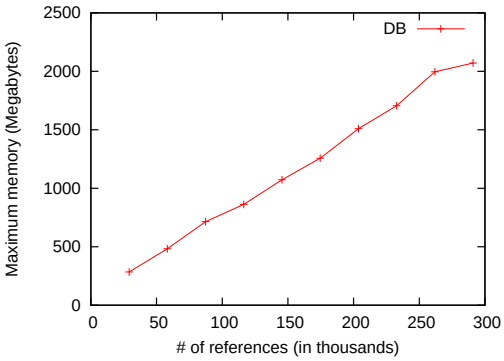

Figure 5.8: Maximum memory of the clustering with different number of author references.

algorithm shows the same trend as that of the execution time. As shown, the memory-based clustering algorithm is not scalable for large datasets with millions of references.

5.5 Summary

For entity resolution, accuracy and efficiency are two important factors of good algorithms. In the experiments, we evaluated the accuracy of our multi-type relational clustering algorithm and showed that the relationships can be used to greatly improve the accuracy of entity resolution, especially for datasets with rich relationships. For efficiency, we compared two blocking methods for reducing the number of redundant comparisons, and showed that the *Naive Inverted-Index-Based Blocking* can achieve higher accuracy than the *VSM-Based Blocking*. Also we showed that the efficiency of our algorithms outperforms that of Bhattacharya and Getoor by several times. Finally, we measured the scalability of our clustering algorithm and found that the execution time and memory consumption are linear proportional to the size of the datasets. In the next chapter, we will describe a record level blocking algorithm to address the scalability problem of our current entity resolution approach.

Chapter 6

Large-Scale Entity Resolution

The experiments have shown that our approach is not scalable for large datasets. In this chapter, we introduce a record-level blocking technique to address the scalability problem for large datasets. Section 6.1 gives an introduction of approaches for the large-scale entity resolution and Section 6.2 describes the record level blocking method. Section 6.3 briefly describes how we resolve the entities for each small partition, and Section 6.4 introduces a disk-based approach for merging results of those partitions. Finally Section 6.5 ends with a summary.

6.1 Introduction

Given the increasing volume of data, a large number of datasets have grown to the size that cannot be processed using the current approaches (which we refer to as memory-based approaches in the following sections). For example, the ReaSoN dataset has 35,878,709 references, and in the experiment, our improved clustering algorithm requires about 2G of memory for 300,000 references. Apparently, the memory-based approaches are not scalable for large datasets, and we need a solution to handle these large datasets.

Most approaches addressing the scalability problem of entity resolution use the *Sorted Neighborhood Method* [37], *Canopies* [49] or their variants which we have introduced in the related work. These approaches work well for large datasets when the attribute-based approaches are employed for entity resolution, since attribute-based approaches rely on attributes of references and the small partitions obtained through blocking are attribute independent so that they can be handled one by one without referring to references in other partitions.

However, for approaches that use relationships for entity resolution, the above blocking methods do not work for large datasets because the partitions are not *relationally* independent of one another, and references in other partitions may be needed when we resolve references in one partition. For instance, in the example dataset, *Jiong Yang* in *P1* and *J.*

Yang in *P2* are put in the same partition for their similar attributes, but the names of their coauthors *Richard R. Muntz* and *Richard Muntz* are dissimilar from either of them and are put in another partition. Thus we need to load all the relational dependent partitions into memory when we process one partition. Furthermore, these relational dependent partitions cannot be accessed sequentially because they may be randomly distributed in the disk.

In this section, we introduce a record level blocking method which incorporates the relationships of references into partitions. The blocking techniques we discussed above focus on references, and we refer to them as *reference level blocking*, while this new blocking method focuses on records, and we refer to it as *record level blocking*. Recall that when we initialize the graph model using records, all the relationships are formed within each record. Thus by splitting the dataset by records, we can get partitions that are not only attribute independent but also relational independent.

Our approach for the large-scale entity resolution can be summarized in three steps; We first use the record level blocking method to split the whole dataset into small overlapping partitions, then we apply the memory-based clustering approach on each of the small partitions. And finally we integrate the results of all the partitions into a single result set. In following sections we will describe each of the steps in details.

6.2 Record Level Blocking

Blocking is a technique to split the whole dataset into small partitions according to some approximate similarity measures. Record level blocking is one such blocking technique that applies the measures on records. Figure 6.1 gives an example of how record level blocking works.

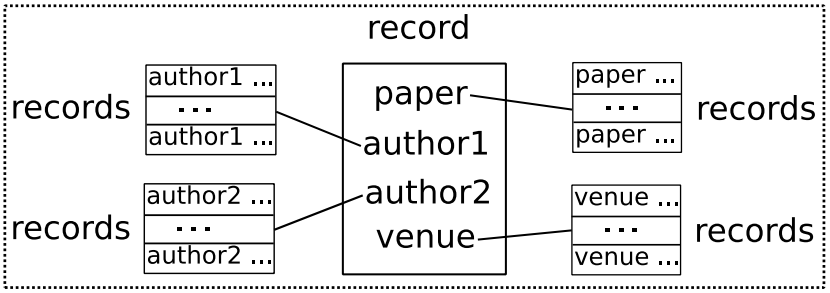

Figure 6.1: A sample partition initialized from *record*. The *records* are a set of records that contain similar references to references in *record* like paper, author1, author2 and venue.

To construct a partition, we start from a record that contains several references. For each reference r_i in the record, we retrieve all the records that contain similar references to r_i. As shown in Figure 6.1, we retrieve the records containing similar references to paper, author1, author2 and venue, and then put them in the same partition. In other words, we

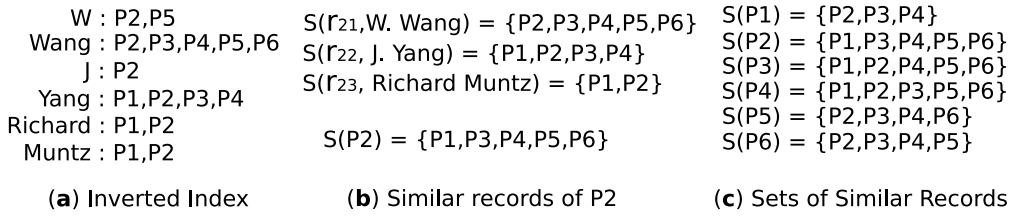

Figure 6.2: Examples for illustrating the process of record level blocking.

put records that are similar to each other in the same partitions. Here we define two records $rec_i = \{r_{i,1}, r_{i,2}, \ldots\}$ and $rec_j = \{r_{j,1}, r_{j,2}, \ldots\}$ to be similar if record rec_i has at least one reference $r_{i,k}$ that is similar to a reference $r_{j,k}$ in record rec_j.

Using the record level blocking, we can easily build partitions for records based on the two reference level blocking techniques we developed. Though the basic process is the same as that of the reference level blocking, there are still several differences. In the following sections, we will describe the details of the record level blocking.

6.2.1 Indexing Datasets

Similar to the reference level blocking, the record level blocking uses inverted index as the data structure to index the datasets and builds different indexes for different types of references in the datasets. One may argue that it is much easier to build just one index for the entire dataset. However, we make our choice for the following reason. Suppose that if we build only one index for the whole dataset, then each record instead of reference is considered as a document. This makes it difficult to judge if two records are similar because the similarity threshold is hard to choose. A high threshold will filter out records that are similar in short author names, while a low threshold will bring in records that share only several common words in title but are not similar at all. Thus using one index will reduce the recall of the result. Instead, building indexes for different types of reference can help retrieve more relevant records.

To build the index, we treat the attributes of every reference r_i as a document, and the id of the *record* containing reference r_i instead of the reference id as the document id. Figure 6.2(a) shows an example of the inverted index for authors in *P2* which lists the mappings from terms in the attributes of references to the record containing these references. For example, the document list of *W* consists of records *P2* and *P5* which contain the name *W. Wang*.

6.2.2 Building Blocks

After building indexes of the dataset, the next step is to build blocks, a set of overlapping partitions of the dataset that are both relational and attribute independent of each other.

input : $Rec = \{rec_1, rec_2, \ldots, rec_n\}$, in which $rec_i = \{r_{i1}, r_{i2}, \ldots, r_{ik}\}$
output: A set of blocks.

```
1  blockId = 0;
2  foreach rec_i ∈ Rec do
3      recordSet = ∅;
4      //Retrieve similar records for each of the references in rec_i.
5      foreach r_ij ∈ rec_i do
6          S(s_ij) = getSimilarRecord(r_ij);
7      end
8      //Merge all the set of similar records.
9      S(rec_i) = ∪_j S(s_ij);
10     foreach s_ij ∈ S(rec_i) do
11         //Check if these two records have been compared before.
12         if hasCompared(rec_i, s_ij) = true then
13             continue;
14         end
15         recordSet.add(s_ij);
16     end
17     if recordSet.isEmpty() = true then
18         continue;
19     end
20     recordSet.add(rec_i);
21     blockId++;
22     //Output blocks to the disk.
23     output(blockId, recordSet);
24     //Update the global map.
25     foreach rec ∈ recordSet do
26         globalMap.add(rec, blockId);
27     end
28 end
```

Algorithm 5: The process of building blocks.

To build blocks, we first need to retrieve and group the similar records. Figure 6.1 have already illustrated the general idea of retrieving similar records for a given record rec_i. Here we describe more details of the implementation. Algorithm 5 gives the process of building blocks. We traverse all the records in the dataset once. For each record rec_i, we first create a record set to store the records that we will put in a block. Then we retrieve similar records $\mathcal{S}(s_{ij})$ for each reference r_{ij} in rec_i (Lines 5-7) using the reference level blocking techniques. After that, we merge the similar records of record rec_i to construct the set of similar records $\mathcal{S}(s_{ij})$ of s_{ij}. Figure 6.2(b) gives an example of similar records for references r_{21}, r_{22} and r_{23} in record *P2*, and the similar record set of record *P2*. Also Figure 6.2(c) lists all sets of similar records for records in the example dataset.

Now we have a set of similar records $\mathcal{S}_(rec_i)$ for each record rec_i; However, the process is not just putting all the similar records together to form a block since a lot of record pairs will be compared several times. For example, in Figure 6.2(c), if we put the similar records of each record together, then the record pairs <*P2,P3*>, <*P2,P4*>, <*P3,P4*> in the first block are also in the other blocks, which means that there will be five more redundant comparisons on these pairs of records.

In order to reduce as many unnecessary comparisons as possible and further clean the initial blocks, we employ a global map to keep the history of built blocks and a cleaning rule to help with the construction of blocks. The global map maps each record rec_i to the list of blocks containing rec_i. Using the global map, the cleaning rule is defined as follow: if the lists of blocks of two records rec_i and rec_j share a common block, then rec_i and rec_j are already in the same block and should not be added into a new block again.

Lines 10-16 show the building process for each block. We first check if rec_i has been compared with its similar record s_{ij} before (Line 12). If yes, we continue to process the next similar record; Otherwise, we add s_{ij} into the record set *recordSet* of rec_i. After checking each similar record of rec_i, we then examine the *recordSet*, if *recordSet* is an empty set which means that rec_i has already been in the same block with its similar records, then there is no need to create a block (Lines 17-19); Otherwise, we create a new block, and add rec_i to it (Line 20) and serialize the block to disk (Lines 21-23). At the same time, we update the global map with the new block.

The final blocks built using the global map and cleaning rule for the example dataset are shown in Figure 6.3. We may notice that the pair *P5* and *P6* appear in three different blocks. This is because *P3* and *P4* have never been compared with *P5* and *P6* before. This problem can not be avoided using our current strategy. One method is to use another global map for the comparison history in the memory-based clustering to further avoid the redundant comparisons. However, we have not implemented this method in this thesis.

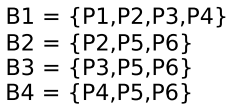

Figure 6.3: Blocks for the example dataset.

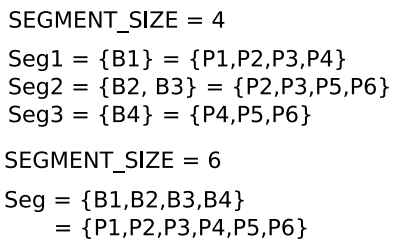

Figure 6.4: Segments for the example dataset with different segment size.

6.2.3 Building Segments

Recall that the purpose of record level blocking is to split the dataset into partitions so that each partition can be handled independently by the memory-based entity resolution algorithm. The blocks constructed above are one such partition and can be used for clustering. However, we notice that the total number of blocks is four which is too large given that the total number of records in the dataset is only six. So for very large dataset like ReaSoN, the number of blocks will be too large and the number of references in each block iiss too small that clustering them one by one cannot make full use of the memory and CPU resources.

To make full use of the available resources, we further group blocks into segments and let the segments be the processing unit of the memory-based algorithm instead of blocks. The process to build segments is straightforward. We first give the size of each segment as *SEGMENT_SIZE*. Here the size of a segment is defined as the number of records in the segment. Next we go over the blocks obtained above, and continuously add the blocks into a segment until the number of records in the segment gets closest to the defined *SEGMENT_SIZE*, then we create a new segment and repeat the segment building process. Figure 6.4 gives two examples of segment with different *SEGMENT_SIZE*, from which we can see that some of the repeated comparisons can be further eliminated.

Deciding the segment size is not easy and depends on several factors. In our implementation, we define the segment size according to the available memory and number of CPU cores. For example, with a server of 32G memory and 8 CPU cores, we allocate 4G memory for each segment and schedule 8 clustering threads to run concurrently for each segment. Though this configuration of assigning resources seems to work well intuitively, the CPUs are still not fully used in our experiments. One possible reason is that there is plenty of time spent on I/O operations. Also, the thread's context switching may be an overload. In addition, whether 4G x 8, 8G x 4 or other configurations are better or worse than our current configuration has not been verified and needs to be explored in future works.

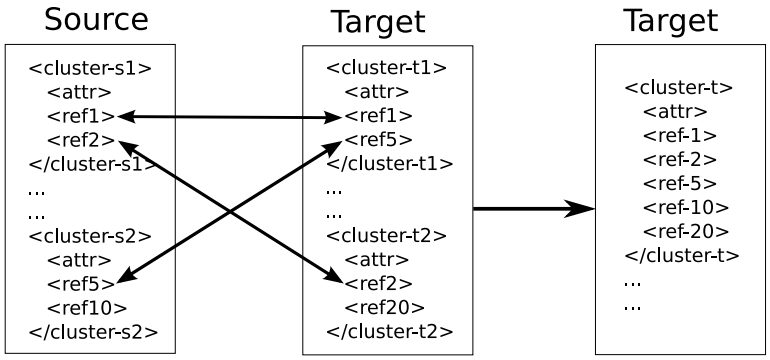

Figure 6.5: Illustration of the process of merging clusters in *Source* to *Target*.

6.3 Relational Clustering

After we split the large datasets into overlapping partitions that are both attribute and relational independent, the next step is to apply the memory-based entity resolution algorithm on each of the partitions (segments), and to produce a corresponding clustering result. As discussed in the segment building process, the segments here are both relational and attribute independent of each other and can be scheduled to be resolved concurrently.

6.4 Merging Results

Given the clustering result of each segment, the last step is to merge these results into a final result set. Ideally when all the duplicates are resolved correctly, the merging process is simply computing the transitive closure over the results of segments. In fact, any false decision made during the clustering process may bring more mistakes during the merging process. For example, if author reference *W. Wang* is correctly merged with *Wei Wang* in one cluster but falsely merged with *Weining Wang* in another cluster, then grouping *Wei Wang* with *Weining Wang* through *W. Wang* will be a false decision. Thus before merging two clusters that share any common references, we first need to compare their attribute similarity. In addition, the result sets are too large (same size as the whole dataset) to be loaded in memory for an one-time merging. To solve this problem, we propose to use a disk-based approach to merge the result sets.

The disk-based approach introduces a target file for storing the merged clusters to help with the merging process. For each clustering result of a segment (which is stored in a source file), we first load the source file into memory, and then we sequentially compare the clusters in the target file with that in the source file and merge clusters that meet our merging rules. At last we write back the new merged clusters into the target file. Note that clusters in the target file that are merged in this round, are also removed from the target file. Once all the

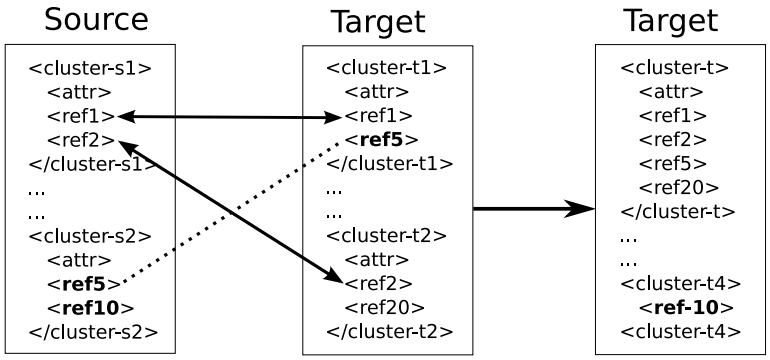

Figure 6.6: Illustration of the process of merging clusters with conflicts in *Source* to *Target*.

results are processed, the clusters in the target file will be the final clustering results for the dataset.

Figure 6.5 shows the typical scenario for integrating the results in a source file to a final target file. From the figure, we can see that $ref1$ is merged with $ref2$ in the source file and with $ref5$ in the target file. Meanwhile, $ref2$ and $ref20$ are in the same cluster in the target file and so do $ref5$ and $ref10$ in the source file. If no false decisions are made in the clustering process, then by iteratively expanding a cluster $\mathcal{C}_i$ with references that are duplicates of the references in $\mathcal{C}_i$, we can obtain the final clusters for all the references as shown in the target file on the right in Figure 6.5. However, for results with false positives, we need to further check the attributes of clusters. Thus we define two merging conditions as follow: 1) clusters should have at least one reference in common, 2) The similarity score of representative attributes[1] of clusters should be higher than a given threshold. For the cluster that meets the first condition but fail the second one, we simply ignore the cluster in the source file but remove the references shared by them from this cluster. For example, in Figure 6.6, clusters *cluster-s2* and *cluster-t1* share a common reference $ref5$, and we assume the attributes of *cluster-s2* are not similar to that of *cluster-t1*. To resolve this conflict, we remove $ref5$ from *cluster-s2* and create a new cluster *cluster-t4* for other references in *cluster-s2*.

6.5 Summary

Entity resolution for very large datasets is an important task for many applications especially for web applications. Resolving entities using both attributes and relationships can improve the accuracy of entity resolution. In this chapter, we propose a record level blocking method that splits the whole dataset into small partitions that are both attribute and

[1]Choosing the representative attribute of a cluster is domain dependent. In our case, we choose the representative attribute of the cluster from references in DBLP first, then ACM DL, and then the attribute shared by most references in the cluster, and then the one with longest string.

relational independent. Then we apply the memory-based entity resolution algorithm on each of partitions, and integrate results of those partitions into the final clustering result. In addition, we explore methods to optimize the size of partitions while maintaining accuracy, and use disk-based method for merging results that cannot be accomplished in memory.

Chapter 7

Conclusions and Future Work

This chapter summarizes the contributions we have achieved in this thesis and discusses possible future works that can further extend or improve my current work.

7.1 Conclusions

Entity resolution, which aims to resolve the tuple level inconsistency during the data integration process, is a crucial step for preparing data of high quality. Most approaches adopt pairwise comparisons for entity resolution. Traditional approaches use attributes to resolve duplicates and can achieve good results on some datasets. Recently, relationships are used to further improve the accuracy of entity resolution. In this thesis, we generalize the single type relational entity resolution algorithm proposed by Bhattacharya and Getoor [8] to a multi-type algorithm that can exploit multiple relationships of references and resolve multiple types of references simultaneously, and show that the multi-type relational algorithm can achieve the same high accuracy of single type relational algorithm.

As the volume of datasets increases, the efficiency and scalability of these algorithms are also important for entity resolution systems, especially when the datasets are large. Blocking is a commonly used technique to reduce the unnecessary pairwise comparisons and thus improve efficiency by partitioning large datasets into independent small subsets. We implemented two blocking methods, a *Naive Inverted-Index-Based Blocking* method that exploits the advantage of inverted index for efficiently retrieving similar references, and a *Vector Space Model Based Blocking* which combines the cosine similarity measure and the tf-idf weighting scheme to further rank and filter the potential duplicates. The former method is specific for author references and performs much more efficiently with little loss of accuracy, and the later one works for both author references and paper references and achieves higher accuracy while with much loss of efficiency.

In addition to reducing the redundant comparisons, we also work on optimizing the relational clustering process proposed by Bhattacharya and Getoor [8]. We propose to use

an iterative clustering process which postpones the propagation of merging results to the time they are needed. By this iterative clustering process, the similarity scores of references that are affected by new merged references do not need to be updated immediately and thus can reduce the memory space for these intermediate results and avoid some unnecessary computations.

For datasets that are too large to be loaded into memory for resolution, scalability is a concern for entity resolution process. Entity resolution algorithms that work well in memory can not be easily ported into disk-based algorithms when relationships are required for relational clustering. We propose a record-level blocking technique to adapt our in-memory clustering algorithm to a disk-based algorithm. The record-level blocking partitions the large datasets into small subsets which are both attribute and relational independent of each other. Using the disk-based entity resolution approach, we resolve the duplicates in the ReaSoN dataset and build an integrated dataset for the ReaSoN project.

To conclude, in this thesis, we use multiple relationships among references to improve the accuracy of entity resolution and explore various methods to improve the efficiency of the process. Though we only apply these approaches on the bibliographic datasets, the idea and methods can be extended and used in other fields like Natural Language Process (NLP) and Personal Information Management (PIM). Take applications in NLP for example, our relational clustering methods can use relationships like co-occurrence to help with the named entity recognitions and relationships identification between entities.

7.2 Future Work

The exploration of relationships for entity resolution and various methods for improving efficiency have accomplished some results that are satisfactory, they also lay groundwork for future work. There are a number of challenges that are worth to address in the near future.

First, the parameters and thresholds in the clustering algorithm are all tuned manually. In order to obtain the best accuracy, we have to try a number of possible parameters which is tedious and time-consuming. In the experiments, we found that these parameters are related to several properties of the datasets. So building a model to map the properties to the parameters or using machine learning techniques to learn the best parameters will be worth a try.

Second, we can explore more graph properties to help with the clustering. Currently, the only graph property used is the neighborhood relationships (the relationships within a record), we can explore with nodes that are not just one degree away, but two or more. For example, two similar references that refer to the same entity may be connected through a path with more than two nodes.

Third, a system that can tolerate and correct mistakes is needed. In some circumstances,

two merged references may be found later to refer to different entities; However, due to the problem with the agglomerative hierarchical clustering algorithm, two references can neither be divided nor swapped once they are merged. This problem not only decreases the accuracy of our algorithm, but propagates the mistakes to other related references. Thus a model that can undo or auto-correct decisions that we mistakenly made will be beneficial for the entity resolution process.

Fourth, the current algorithm cannot resolve datasets incrementally. Whenever a new dataset is added, the clustering process needs to be restarted from scratch with all datasets. The reason is that when the clustering is finished, all the relationships between references will be lost so that these relationships can not be used when the new dataset is added. Thus we need to an incremental approach to preserve relationships during the clustering process and reload them when needed.

Finally, the relational datasets are becoming more common either from the social networks or graphs built from the Web. We can apply our relational algorithm on other domains like Natural Language Processing to de-duplicate entities through the co-occurrence relationship, and disambiguate person names on the web based on the web structure of the web pages.

Appendix

Appendix A

```
<STARTREC>
   <BIBNO>100000</BIBNO>
   <WBIBNO>99977</WBIBNO>
   <PUBTYPE>JOURNAL ARTICLE</PUBTYPE>
   <TITLE><![CDATA[Inside risks: the clock grows at midnight]]></TITLE>
   <AUTHEDIT>
      <person_id>PP15039075</person_id>
      <lname><![CDATA[Neumann]]></lname>
      <fname><![CDATA[Peter]]></fname>
      <mname><![CDATA[G.]]></mname>
   </AUTHEDIT>
   <PUBYEAR>1991</PUBYEAR>
   <LANG>ENGLISH</LANG>
   <ISSN>0001-0782</ISSN>
   <JRLNAME>Communications of the ACM</JRLNAME>
   <JRLABBREV>Commun. ACM</JRLABBREV>
   <JRLVOL>34</JRLVOL>
   <JRLDATE>Jan. 1991</JRLDATE>
   <JRLISS>1</JRLISS>
   <SPAGE>170</SPAGE>
   <REFERENCES>
      <REF>
         <REF_SEQNO>1</REF_SEQNO>
         <REF_ID>1</REF_ID>
         <REF_STR><![CDATA[Lamport, L., and Melliar-Smith, P.M. Synchronizing clocks in the presence 52-78.]]></
REF_STR>
         <REF_BIBNO>2455.2457</REF_BIBNO>
      </REF>
      <REF>
         <REF_SEQNO>2</REF_SEQNO>
         <REF_ID>2</REF_ID>
         <REF_STR><![CDATA[Rushby, J.M., and yon Henke, F. formal verification of the interactire Convergence Clock
Synchronization Algorithm using EHDM. SRI- ark CA, february 1989]]></REF_STR>
         <REF_BIBNO></REF_BIBNO>
      </REF>
      <REF>
         <REF_SEQNO>3</REF_SEQNO>
         <REF_ID>3</REF_ID>
         <REF_STR><![CDATA[Garman, J.R. The bug heard 'round the world. ACM Softw. eng. not,6,5 (October 1981),
3-10.]]></REF_STR>
         <REF_BIBNO></REF_BIBNO>
      </REF>
      <REF>
         <REF_SEQNO>4</REF_SEQNO>
         <REF_ID>4</REF_ID>
         <REF_STR><![CDATA[Other cases noted above were discussed in "'RISKS" and in .SEN 11, 2; 13, 2; 14, 2; 14, 6"
and 15, 1.]]></REF_STR>
         <REF_BIBNO></REF_BIBNO>
      </REF>
   </REFERENCES>
   <ABSTRACT><![CDATA[]]></ABSTRACT>
</STARTREC>
```

Figure 1: A sample record of ACM Digital Library.

Figure 1 lists a sample record in ACM Digital Library in XML format. Each record contains the type of the paper(**PUBTYPE**), title(**TITLE**), author name(**AUTHEDIT**) in three parts, last name(**lname**), first name(**fname**), and middle name(**mname**). Other information is also given, such as year(**PUBYEAR**), venue name(**JRLNAME**, **JRLABBREV**), volume(**JRLVOL**), issue(**JRLISS**), pages(**SPAGE**) and citations in the reference part(**REFERENCE**).

Appendix B

Venue
SIGIR Forum (ACM SIGIR Forum)
SIGKDD Explor. Newsl. (ACM SIGKDD Explorations Newsletter)
SIGMOD Rec. (ACM SIGMOD Record)
SIGWEB Newsl. (ACM SIGWEB Newsletter)
TODS (ACM Transactions on Database Systems)
TOIS (ACM Transactions on Information Systems)
TWEB (ACM Transactions on the Web)
CIKM (Conference on Information and Knowledge Management)
DaMoN (Data Management On New Hardware)
DMKD (Data Mining And Knowledge Discovery)
DOLAP (Data Warehousing and OLAP)
DPDS (International Symposium on Databases for Parallel and Distributed Systems)
GIR (Workshop On Geographic Information Retrieval)
GIS (Geographic Information Systems)
IHIS (Interoperability Of Heterogeneous Information Systems)
IPSN (Information Processing In Sensor Networks)
ISIS (Information Quality in Informational Systems)
IRP2PN (Information Retrieval In Peer-To-Peer Networks)
JACM (Journal of the ACM)
KDD (Conference on Knowledge Discovery in Data)
MDM (International Conference On Mobile Data Management)
MMDB (ACM International Workshop On Multimedia Databases)
MobiDE (International Workshop on Data Engineering for Wireless and Mobile Access)
OODS (International Workshop on Object-Oriented Database Systems)
PODS (Symposium on Principles of Database Systems)
SAC (Symposium on Applied Computing)
SIGIR (Annual ACM Conference on Research and Development in Information Retrieval)
SIGMOD (International Conference on Management of Data)
VLDB Journal (The International Journal on Very Large Data Bases)
UIMC (Conference On Ubiquitous Information Management And Communication)
VLDB (Very Large Data Bases)
WIDM (Workshop On Web Information And Data Management)
WSDM (Web Search and Web Data Mining)
WWW (International World Wide Web Conference)
ER (International Conference on Conceptual Modeling / the Entity Relationship Approach)
ICDE (International Conference on Data Engineering)
EDBT (Extending Database Technology)
ICDT (International Conference on Database Theory)
RIDE (Research Issues in Data Engineering)
DOOD (Deductive and Object-Oriented Databases)
TKDE (Transactions on Knowledge and Data Engineering)
DKE (Data & Knowledge Engineering)
WWW Journal (World Wide Web Journal)

Table 1: List of venues in DB dataset.

Bibliography

[1] Kdd cup 2003 arxiv hep-th dataset. http://www.cs.cornell.edu/projects/kddcup/datasets.html.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[3] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *ACM International Conference on Very Large Data Bases*, pages 586–597, 2002.

[4] Y. Atilgan and F. Dogan. Data mining on distributed medical databases: Recent trends and future directions. In *IT Revolutions*, pages 216–224, 2008.

[5] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[6] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 11–18, 2004.

[7] I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *KDD workshop on Multi-relational data mining*, pages 3–12. ACM, 2005.

[8] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

[9] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *International Conference on Knowledge Discovery and Data Mining*, pages 39–48, 2003.

[10] M. Bilenko, R. J. Mooney, W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

[11] J. Bleiholder and F. Naumann. Data fusion. *ACM Computing Surveys*, 41(1):1–41, 2008.

[12] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *ACM SIGMOD International Conference on Management of Data*, pages 175–186, 2001.

[13] R. Brelger, K. Carley, and P. Pattison. Data mining in social networks. *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*, pages 287–302, 2004.

[14] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *ACM SIGMOD International Conference on Management of Data*, pages 313–324, 2003.

[15] P. Christen. Febrl: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *International Conference on Knowledge Discovery and Data Mining*, pages 1065–1068, 2008.

[16] P. Christen, T. Churches, and J. X. Zhu. Probabilistic name and address cleaning and standardisation. In *The Australasian Data Mining Workshop*, pages 99–108.

[17] T. Churches, P. Christen, K. Lim, and J. Zhu. Preparation of name and address data for record linkage using hidden markov models. *BMC Medical Informatics and Decision Making*, 2(1):9, 2002.

[18] W. W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transaction on Information System*, 18(3):288–321, 2000.

[19] W. W. Cohen, H. A. Kautz, and D. A. McAllester. Hardening soft information sources. In *International Conference on Knowledge Discovery and Data Mining*, pages 255–259, 2000.

[20] W. W. Cohen, P. D. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI Workshop on Information Integration on the Web*, pages 73–78, 2003.

[21] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *International Conference on Knowledge Discovery and Data Mining*, pages 475–480, 2002.

[22] D. A. Cohn, L. E. Atlas, and R. E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[23] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[24] M.-Y. Day, T.-H. Tsai, C.-L. Sung, C.-W. Lee, S.-H. Wu, C.-S. Ong, and W.-L. Hsu. A knowledge-based approach to citation extraction. In *Proceedings of the IEEE International Conference on Information Reuse and Integration*, pages 50–55, 2005.

[25] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *ACM SIGMOD International Conference on Management of Data*, pages 85–96, 2005.

[26] M. G. Elfeky, A. K. Elmagarmid, and V. S. Verykios. Tailor: A record linkage tool box. In *International Conference on Data Engineering*, pages 17–28, 2002.

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

[28] M. Fair. Generalized record linkage system - statistics canada's record linkage software. *Austrian Journal of Statistics*, 33(1-2):37–53, 2004.

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

[30] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *ACM International Conference on Very Large Data Bases*, pages 371–380, 2001.

[31] H. Garcia-Molina. Entity resolution: Overview and challenges. In *International Conference on Conceptual Modeling*, pages 1–2. 2004.

[32] L. Getoor and C. P. Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.

[33] L. E. Gill. Ox-link: The oxford medical record linkage system. In *Proceeding of International Record Linkage Workshop and Exposition*, pages 15–33, 1997.

[34] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *ACM International Conference on Very Large Data Bases*, pages 491–500, 2001.

[35] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an rdbms for web data integration. In *International World Wide Web Conference*, pages 90–101, 2003.

[36] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[37] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 127–138, 1995.

[38] T. Herzog, F. Scheuren, and W. Winkler. Standardization and parsing. *Data Quality and Record Linkage Techniques*, pages 107–114.

[39] M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Society*, 84:414–420, 1989.

[40] L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *International Conference on Database Systems for Advanced Applications*, page 137, 2003.

[41] D. V. Kalashnikov, Z. Chen, S. Mehrotra, and R. Nuray-Turan. Web people search via connection analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20(11):1550–1565, 2008.

[42] D. V. Kalashnikov, Z. Chen, R. Nuray-Turan, S. Mehrotra, and Z. Zhang. West: Modern technologies for web people search. In *International Conference on Data Engineering*, pages 1487–1490, 2009.

[43] D. V. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transaction on Database System*, 31(2):716–767, 2006.

[44] D. V. Kalashnikov, S. Mehrotra, Z. Chen, R. Nuray-Turan, and N. Ashish. Disambiguation algorithm for people search on the web. In *International Conference on Data Engineering*, pages 1258–1260, 2007.

[45] M.-L. Lee, T. W. Ling, and W. L. Low. Intelliclean: a knowledge-based intelligent data cleaner. In *International Conference on Knowledge Discovery and Data Mining*, pages 290–294, 2000.

[46] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[47] M. Ley. Dblp - some lessons learned. In *Proceedings of the VLDB Endowment*, volume 2, pages 1493–1500, 2009.

[48] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[49] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *International Conference on Knowledge Discovery and Data Mining*, pages 169–178, 2000.

[50] A. E. Monge and C. Elkan. The field matching problem: Algorithms and applications. In *International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[51] A. E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 21–27, 1997.

[52] H. B. Newcombe, J. M. Kennedy, S. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, Oct 1959.

[53] J. Nin, V. Muntés-Mulero, N. Martínez-Bazan, and J.-L. Larriba-Pey. On the use of semantic blocking techniques for data cleansing and integration. In *International Database Engineering and Applications Symposium*, pages 190–198. IEEE Computer Society, 2007.

[54] L. Philips. Hanging on the metaphone. *Computer Language Magazine*, 7(12):39–44, 1990.

[55] L. Philips. The double metaphone search algorithm. *C/C++ Users Journal*, 18(5), 2000.

[56] E. H. Porter, W. E. Winkler, B. O. T. Census, and B. O. T. Census. Approximate string comparison and its effect on an advanced record linkage system. In *Advanced Record Linkage System. U.S. Bureau of the Census, Research Report*, pages 190–199, 1997.

[57] R. C. Russell. Soundex. U.S. Patent 1,435,663, November 1922.

[58] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA, 1986.

[59] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *International Conference on Knowledge Discovery and Data Mining*, pages 269–278, 2002.

[60] P. Singla and P. Domingos. Multi-relational record linkage. In *KDD Workshop on Multi-Relational Data Mining*, pages 31–48, 2004.

[61] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[62] R. L. Taft. *Name search techniques*. Number 1. Albany, Bureau of Systems Development, New York State Identification and Intelligence System, 1970.

[63] S. Tejada, C. A. Knoblock, and S. Minton. Learning object identification rules for information integration. *Information System*, 26(8):607–633, 2001.

[64] S. Tejada, C. A. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *International Conference on Knowledge Discovery and Data Mining*, pages 350–359, 2002.

[65] J. R. Ullmann. A binary n-gram technique for automatic correction of substitution, deletion, insertion and reversal errors in words. *Computer Journal*, 20(2):141–147, 1977.

[66] E. C. C. Vilarinho, A. S. da Silva, M. A. Gonçalves, F. de Sá Mesquita, and E. S. de Moura. Flux-cim: flexible unsupervised extraction of citation metadata. In *ACM/IEEE Joint Conference on Digital Libraries*, pages 215–224, 2007.

[67] M. S. Waterman, T. F. Smith, and W. A. Beyer. Some biological sequence metrics. *Advances in Mathematics*, 20(3), 1976.

[68] W. E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

[69] W. E. Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006.

[70] E. Zheleva and L. Getoor. To join or not to join: the illusion of privacy in social networks with mixed public and private user profiles. In *International World Wide Web Conference*, pages 531–540, 2009.

[71] J. J. Zhu and L. H. Ungar. String edit analysis for merging databases. In *KDD 2000 Workshop on Text Mining*, pages 117–118, 2000.