Data Provenance Support in Entity Resolution

Xu Wang

Supervisor: Dr. Qing Wang

COMP4560: Advanced Computing Project

Australian National University Semester 2, 2016

October 28, 2016
I wish to express my sincere thanks to Dr. Qing Wang, my supervisor, for patiently providing me with all the necessary materials, answering my questions and guiding me through the process of this project over the past four months.

My thanks also go to Prof. Weifa Liang, the course convener, for valuable assistance in academic writing and in planning and creating presentations. I would like to take this opportunity to express gratitude to all of the faculty members in the department for their help and support. I also thank my family members for their unceasing encouragement, and financial and spiritual support and attention. I am also grateful to my friends who supported me throughout this venture.
Abstract

Entity resolution, also known as record linkage, refers to the process of deciding which records from one or more databases refer to the same entities. Entity resolution has many significant real-world applications, such as censuses and criminal detection, although this technique contains several drawbacks. It is difficult to establish reliability and repair inconsistencies in entity resolution results without the use of other techniques from traditional approaches. Data provenance refers to the history of data and the way in which the data has been processed. Adopting provenance support in entity resolution process provides solutions for above issues. This research proposes and implements a framework of integrating data provenance techniques with entity resolution process in order to enhance the reliability and capability of the original system. Firstly, three levels of data structures are generated for records, pairs, clusters for the storage of provenance data. Secondly, to enable querying and analysis of entity resolution results, query methods are also proposed, based on the provenance data of each level. In addition, a graphical user interface is provided to aid in graphical queries and data visualisation. The proposed framework was tested using two datasets, CORA and NCVR, and the efficiency of the system in terms of space and time were evaluated for data storage structures and query methods respectively.

Keywords: Entity Resolution, Data Provenance, Storage Structure, Query Methods, Data visualisation
Contents

Acknowledgements ................................................. 1

Abstract ................................................................... 2

1 Introduction ......................................................... 5
   1.1 ER Applications .............................................. 5
   1.2 Limitations .................................................. 6
   1.3 Project Objectives .......................................... 7
   1.4 Contributions ................................................ 7

2 Related Work ....................................................... 8
   2.1 Entity Resolution ............................................. 8
   2.2 Data Provenance ............................................ 9

3 Framework ........................................................ 10
   3.1 Overview ...................................................... 10
   3.2 Classification Model ....................................... 11
   3.3 Clustering Algorithm ....................................... 12

4 Provenance Storage .............................................. 15
   4.1 Individual Record Level .................................... 15
      4.1.1 Data Sources ........................................... 15
      4.1.2 Blocking Provenance Data ....................... 15
   4.2 Record Pair Level .......................................... 16
   4.3 Record Cluster Level ....................................... 18
      4.3.1 Cluster Dictionary ................................... 19
      4.3.2 Edge Dictionary ...................................... 19
## 5 Provenance Queries

5.1 Individual Record Level ............................................... 21  
5.1.1 FindRecord .......................................................... 21  
5.1.2 FindBlock ............................................................ 23  
5.2 Record Pair Level ...................................................... 24  
5.3 Record Cluster Level ................................................... 25  
5.3.1 FindCluster ........................................................... 25  
5.3.2 Reconstruct ............................................................ 25  

## 6 Experiments

6.1 Experimental Setup .................................................... 28  
6.1.1 Datasets ............................................................... 28  
6.1.2 ER Rules ............................................................... 28  
6.1.3 Experimental Results ............................................... 29  

## 7 Conclusion


## Appendices

A Independent Study Contract ............................................. 33  
B Project Description ....................................................... 36  
C Software Description ..................................................... 39  
D Readme ........................................................................... 41
Chapter 1

Introduction

Entity resolution (ER), also named record linkage, refers to the process of deciding which records from one or more databases refer to the same entities in the real world [25]. For example, for records of citizen information in a database as depicted in Table 1.1, it is possible to apply ER techniques to determine which records may refer to a single person. In this case, three people are identified: Xu Wang ($r_1, r_2, r_3$), Lu Yu ($r_4, r_5$) and Yang Wang ($r_6, r_7, r_8$).

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>ADDRESS</th>
<th>POSTCODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>Xu Wang</td>
<td>5 Dunn Place</td>
<td>2602</td>
</tr>
<tr>
<td>$r_2$</td>
<td>Xu Wang</td>
<td>5 Dunn Pl.</td>
<td>2602</td>
</tr>
<tr>
<td>$r_3$</td>
<td>X. Wang</td>
<td>5 Dunn Pl.</td>
<td>2602</td>
</tr>
<tr>
<td>$r_4$</td>
<td>Lu Yu</td>
<td>1 Elliot Street</td>
<td>2612</td>
</tr>
<tr>
<td>$r_5$</td>
<td>L. Yu</td>
<td>1 Elliot St.</td>
<td>2612</td>
</tr>
<tr>
<td>$r_6$</td>
<td>Yang Wang</td>
<td>5 Dunn Place</td>
<td>2602</td>
</tr>
<tr>
<td>$r_7$</td>
<td>Y. Wang</td>
<td>Dunn Pl.</td>
<td>2602</td>
</tr>
<tr>
<td>$r_8$</td>
<td>Wang Yang</td>
<td>Dunn Pl.</td>
<td>2602</td>
</tr>
</tbody>
</table>

Table 1.1: Sample records

1.1 ER Applications

ER has been extensively studied for more than 50 years [26], and is currently used in numerous real-world fields.

- A major application for ER is the national census [13]. The advantages of applying ER techniques to the analysis of census data include improvements in data quality and reduction of workloads when dealing with large-scale data[29].
CHAPTER 1. INTRODUCTION

The US Census Bureau has adopted record linkage techniques in a test census of Tampa, Florida in 1985, and the quality of the results were significantly enhanced [16].

- ER methodologies have also been widely applied in medical studies, including the construction of links between morbidity and mortality data, the linkage of existing cases to reduce redundancy surveys, and the tracking of longitudinal separation of cases [17]. In recent years, Australia has been placed at the forefront of applying record linkage in medical studies as researchers have conducted comprehensive studies in this area [18].

- A further domain in which ER is important is the crime and fraud detection; since criminals often provide false or misleading personal information when questioned by the police, it is generally difficult to identify a single offender across various databases or systems [7]. By integrating ER techniques with administrative databases, officers can determine the identity of a suspect efficiently and with high accuracy.

1.2 Limitations

Although ER techniques have been applied within a wide range of fields, there are certain disadvantages of ER approaches.

Assessing the reliability of ER results is the primary limitation of ER techniques. An identifier is an attribute which makes a record distinguishable from other records. An example of identifier is the ID attribute in Table 1.1. Lacking of identifiers may therefore reduce the accuracy of the ER results. Even if identifiers are provided, the quality of data may also affect the ER results. For instance, in a register database for a website, ER results can be unreliable if users have registered with aliases or fake personal profiles. In both case, it is hard for the database owners to assess the reliability of the ER results without other techniques.

In addition, the majority of ER methods do not offer a way of repairing or refining inconsistent results. For instance, in the same register database described above, suppose that the database owners know the ER results contain inconsistencies and want to repair them. However, the inconsistencies may caused by using low-quality data or configuring the system with inefficient algorithms and improper parameters, and current ER techniques do not provide mechanisms for retrieving these information. Thus, the detection and correction of inconsistencies is challenging with current ER techniques.
1.3 Project Objectives

Provenance is the chronology of the ownership, custody or location of a historical object [19]. In the context of databases, this refers to the description of the origins of the data and the process by which it arrived at the database [5].

Data provenance techniques are helpful in facilitating ER limitations. When incorrect results are detected, data provenance mechanism helps us review the history and eventually resolve errors. Thus, integrating data provenance techniques with ER process is a valuable task.

The goal of this project is to incorporate data provenance techniques into the ER process in order to enhance the reliability of the original system. The objectives of this project are as follows:

1. To develop storage structures that can be used to efficiently capture provenance information of ER.
2. To develop query methods which provide detailed overviews on the history of an entity.
3. To evaluate the effectiveness and efficiency of the proposed data storage structures and query methods using two real-world data sets, CORA and NCVR.

1.4 Contributions

The current research extends the existing knowledge in this area in the following ways:

1. A rule-based classification algorithm and a cycle-based clustering algorithm are implemented for ER processes.
2. Provenance storage structures are designed at the individual record level, the record pair level, and the record cluster level for the storage of provenance data.
3. Query methods are proposed for the efficient querying of provenance data; in addition, a visual querying tool is developed for graphical querying and data visualisation.
4. The effectiveness and efficiency of the proposed data storage structures and query methods are evaluated using two real-world data sets, CORA and NCVR.
Chapter 2

Related Work

This chapter provides an overview of two areas that are related to this project, entity resolution and data provenance.

2.1 Entity Resolution

The initial idea of ER was raised in 1946 [10]. Numerous studies in this area have been conducted over the last 70 years. Traditional ER techniques are deterministic, and records are linked if and only if all corresponding pairs of attributes have identical value [24]. However, this method is data-sensitive [20], time-consuming and lacks accuracy and scalability [22]. Thus, probabilistic ER was introduced, in which more identifiers were taken into account to calculate the weight and eventually the probability of whether two records belong to the same real-world entity. Also, researchers conducted many studies to improve the quality of ER [2, 21]. Although, their work enhanced the accuracy of ER results but did not address solutions for some ER limitations. For instance, no efficient mechanism for backdating ER history or assessing result reliability has been integrated with ER yet.

The Trio is a database management system developed by Stanford University researchers [28]. To obtain ER support, the same research team also developed a variant of Trio named Trio-ER. Trio-ER is a platform for ER process and has built-in tracking features [1]. However, as this system is based on relational database system, its query methods all conform to SQL style. This makes the system lack scalability and flexibility when dealing with complicated data structures.
2.2 Data Provenance

Data provenance makes the history of the results available to users [6]. It has been applied in many areas such as e-science [23], web data [14] and scientific workflow systems [3]. However, from my knowledge, though data provenance techniques have been applied in various fields, there are few studies that integrates data provenance techniques with ER progress to enhance the performance and expand the functionality. However, the increasingly rapid development of database and data processing techniques address some critical technical issue of data provenance. One outstanding issue is querying data provenance [11]. It is hard to design efficient query methods for querying large-scale complex provenance information. Also, archiving and maintaining provenance data have not been perfectly solved in complicated databases [4].

Researchers from University of Pennsylvania have demonstrated the importance of data provenance, and proposed a framework for relational databases and hierarchical data such as XML [5]. They discussed two types of data provenance: why and where, namely the evidences of the appearance, and the original place and trace of some data. However, the integration of data provenance techniques and ER processes are not included in their scope.
Chapter 3

Framework

This chapter presents an overview of general ER workflow, and describes the algorithms on which the proposed framework is based.

3.1 Overview

Figure 3.1: General ER workflow diagram

Figure 3.1 illustrates a general ER workflow, which includes four phases: indexing or blocking, comparison, classification, and clustering [8].

- The indexing or blocking phase divides records into different blocks. Records in the same block are considered as possibly belonging to the same real-world entity. Accordingly, pairwise comparison is only performed on records within
blocks, in order to reduce the time complexity \[12\]. Suppose that there are \(n\) records in a data set, the time complexity without the use of blocking, is \(O(n^2)\). However, if the whole data set is evenly divided into \(k\) non-overlapping blocks, the time complexity is reduced to \(O(n^2/k)\) which is significantly lower than \(O(n^2)\) for large \(n\).

- The comparison phase carries out pairwise comparisons between records within the same block. In this phase, the similarity between two records is calculated and stored in the form of weight vector. A weight vector is an array storing the similarity values between the attributes of the pair [27]. For example, for record pair \((r_1, r_2)\) and attributes name and address in Table 1.1, assume that we have \(sim(name) = 1\) and \(sim(address) = 0.9\) where \(sim\) is a comparison function and \(sim(A)\) indicates the similarity value of attribute \(A\) between two records, the weight vector of pair \((r_1, r_2)\) is: \([1, 0.9]\).

- The classification phase classifies weight vectors generated in the comparison phase into two categories: match and non-match. A match refers to a record pair whose members belong to the same entity, while a non-match refers to a record pair whose members cannot belong to the same entity [8].

- The clustering phase generates a set of clusters from the matches and non-matches obtained from the classification phase [15]. When this phase is complete, the output consists of a set of clusters, each of which represents a different real-world entity.

### 3.2 Classification Model

This section presents a rule-based classification algorithm proposed for this project [9].

A rule has the form \(P_1 \land P_2 \land \ldots \land P_n\), where \(P_i\) \((i = 1, 2, \ldots, n)\) is a comparison predicate \(sim(A) \ opr \ \tau\), where \(sim(A)\) indicates the similarity value of attribute \(A\), \(opr\) indicates the operator and \(\tau \in [0, 1]\) indicates the threshold. Each rule \(R\) is associated with a weight \(W \in [-1, 1]\). If a weight vector \(w\) satisfies all comparison predicates in a rule \(R\), in which the associated weight is \(W\), then we say \(w\) has weight \(W\). If the weight of a rule is positive, it is known as a match rule; if negative, it is known as a non-match rule. A rule is also classified as hard if the associated weight \(W\) is +1 or -1, or soft if \(W\) has any other values. Thus, there are four types of rules: hard match, hard non-match, soft match and soft non-match.

It is possible that a weight vector is classified by multiple rules. If a weight vector \(w\) is classified by \(R_1, R_2, \ldots, R_n\), and the associated weights are \(W_1, W_2, \ldots, W_n\) respectively, the final weight of \(w\) is \(W = max\{abs(W_1), abs(W_2), \ldots, abs(W_n)\}\), where
abs(W_i) indicates the absolute value of W_i.

**Example 3.1:** Assume that we have a rule \( R_1 = (sim(name) > 0.8) \land (sim(age) > 0.5) \) with weight \( W_1 = 0.8 \), and a weight vector \( w = [sim(name) = 1.0, sim(age) = 0.8] \). Since \( w \) satisfies all comparison predicates in rule \( R_1 \), and the weight of \( R_1 \) is both positive and less than 1, we can say that \( w \) is a soft match classified by \( R_1 \).

**Example 3.2:** Assume that we have a weight vector \( w \) which is classified by \( R_1 \), \( R_2 \) and \( R_3 \), and the associated weights are \( W_1 = 0.8 \), \( W_2 = -0.7 \) and \( W_3 = -0.9 \) respectively. The final weight of \( w \) is -0.9 as 0.9 is the largest absolute value of all rule weights.

The rule-based classification algorithm consists of one or more rules. For each weight vector, the algorithm checks all rules. If the algorithm found a rule which is satisfied by this weight vector, this weight vector is then added into the hard match set, hard non-match set, and soft (match/non-match) set according to the rule weight. For simplicity, it is assumed that there are no inconsistencies between hard matches and hard non-matches, since this type of inconsistency can be resolved by domain experts.

### 3.3 Clustering Algorithm

A cycle-based clustering algorithm is used in this project[25]. This algorithm examines hard matches to produce the initial set of clusters, then examines the soft matches/non-matches in order to expand clusters and eliminate inconsistencies.

Algorithm 1 gives the details of the cycle-based clustering algorithm. Lines 1 to 4 add all record pairs in the soft set into a queue \( Q \), and then sort all pairs in \( Q \) in ascending order according to rule weights. Lines 5 to 7 generate initial clusters from hard match set \( M \). Lines 8 to 13 harden soft pairs. The function \( makeCluster \) attempts to convert a soft match to a hard match or a soft non-match to a hard non-match. This conversion is carried out unless an inconsistency is found, in which case the inconsistency is repaired by harden the pair to the opposite type. Following this stage, the clusters are modified, either by creating a new cluster, expanding an existing cluster or merging two existing clusters. After both hard pairs and soft pairs have been examined, the final cluster set \( C \) is generated.

Two types of inconsistencies may occur in the cycle-based clustering algorithm, and these are depicted in Figure 3.2, as follows:

- As shown in Figure 3.2a, a soft match \((r_1, r_2)\) is inconsistent if in any cluster there exist two records \( r'_1 \) and \( r'_2 \) such that \((r_1, r'_1)\) is a hard match and \((r_2, r'_2)\)...
is a hard non-match.

- As shown in Figure 3.2b, a soft non-match \((r_1, r_2)\) is inconsistent if in any cluster, there exist two records \(r_1'\) and \(r_2'\) such that both \((r_1, r_1')\) and \((r_2, r_2')\) are hard matches.

Figure 3.2: Inconsistencies in the clustering phase
Algorithm 1 Cycle-based clustering algorithm

**Input:** $M$ is the hard match set

- $N$ is the hard non-match set
- $S$ is the soft set
- $Q$ is a queue

**Output:** $C$ is a dictionary. Keys: cluster ids, values: lists of cluster members.

1. for record $r$ in $S$ do
2.   $Q \leftarrow r$
3. end for
4. quickSort($Q$)
5. for pair $p$ in $M$ do
6.   $C \leftarrow initializeCluster(p)$
7. end for
8. while $Q \neq$ empty do
9.   $temp \leftarrow Q.get()$
10. if $temp \notin N$ then
11.   $C \leftarrow makeCluster(temp,temp[type],isConsistent(temp))$
12. end if
13. end while
14. return $C$
4.1 Individual Record Level

Individual record level provenance stores the information of individual records. There are two kinds of provenance data that are recorded: data sources and blocking provenance data.

4.1.1 Data Sources

Data sources include the source information of the original data sets and the blocking phase. For the original data sets, name and owners of the data sets are recorded. For the blocking phase, source data includes the number of blocks, the selected attributes, encoding methods for each attribute and the execution time. Dictionary is an abstract data type for the storage of items, which consists of key-value pairs in which the value can be accessed by the key. Data sources are stored in a data source dictionary, in which keys are types of provenance data and values are corresponding provenance values. Figure 4.1 gives an example of data source dictionary.

4.1.2 Blocking Provenance Data

While performing blocking, records are divided into different blocks according to the blocking methods applied on chosen attributes.

In this case, blocking information was stored in a blocking dictionary. Every time a new block is created, a key-value pair is added to the blocking dictionary; in this case, the key is the block ID and the value is the member list of that block. When a record is added to an existing block, it will be appended to the member list of that block.
However, querying the blocking dictionary is inefficient when the number of blocks is large. For the case where there are \( m \) records in \( n \) blocks, a query to find which block contains a particular record can take a time of \( O(mn) \). To solve this problem, a supplementary data structure was developed in the form of a block indexing dictionary. The block indexing dictionary is derived from the blocking dictionary. To generate the block indexing dictionary, search is carried out for every single record \( r \) in the blocking dictionary to identify the corresponding block name. Following this, a key-value pair is added to the block indexing dictionary, in which the key is the record ID and the value is the block name. Thus, block indexing dictionary does not contain more information than the blocking dictionary; however, it enables efficient searching, since the time complexity of querying in the block indexing dictionary is reduced to \( O(m) \). Figure 4.2 shows an example of blocking dictionary and the corresponding block indexing dictionary.

4.2 Record Pair Level

Provenance at the record pair level includes information about the comparison and classification phase. The attributes and comparison functions applied in the com-
parison phase and all details about the applied algorithm in the classification phase will be recorded. In this case, as a rule-based classification algorithm was used, details of rules need to be included. All above provenance data were stored in a pair level metadata dictionary. Figure 4.3 shows an example of pair level metadata dictionary.

```
pair level metadata dictionary=
{
    Comparison attributes: [1, 3, 5]
    Comparison functions: [q-gram, q-gram, q-gram]
    No. of weight vectors: 149012
    Classification algorithm: Rule-based
    Rules:
        R_1: P_1 \land P_2 \land \ldots \land P_n
        R_2: P_1 \land P_2 \land \ldots \land P_n
        ...
}
```

Figure 4.3: Example of pair level metadata dictionary

Record pair level provenance includes information about record pairs. The provenance of the comparison and classification phase was captured individually and then combined to construct the final pair level provenance. In the comparison phase, the similarity between records was calculated and weight vectors were generated accordingly.

In the classification phase, weight vectors obtained from the comparison phase are classified into three different sets: hard matches set, hard non-matches set and soft set. The classification phase provenance depends on the classification algorithms applied. As described in Chapter 3, a rule-based algorithm was used in the classification phase. Thus, when a record pair has been classified by a rule, the detail of that rule including rule content, rule weight, and rule ID need to be recorded. Thus, when classifying a weight vector \( w \), a dictionary was created in which keys are provenance name and values are corresponding values. This dictionary was then added to weight vector dictionary as the value of key \( (r_1, r_2) \). Figure 4.4 shows an example of weight vector dictionary.

However, computational issues arise for weight vector dictionary. For \( r \) records and \( f(r) \) weight vectors, since weight vectors are generated pairwise, the complexity of \( f(n) \) is \( O(n^2) \). In order to find a certain pair \( (r_1, r_2) \), it is necessary to check each pair, and the time required for this is therefore \( O(r^2) \) in the worst case. To solve this issue, a pair indexing dictionary was constructed, in which keys are record IDs and values are record pairs containing the record ID in the corresponding key. Then, if
queries are carried out in the pair indexing dictionary, for a specific record \((r_1, r_2)\), a time of \(O(r)\) time is required to find one record and a time of \(O(g(n))\) is required to get information of \((r_1, r_2)\) in the related list, where \(g(n)\) indicates the size of the largest cluster. The new time complexity is \(O(rg(n))\), and \(O(rg(n)) \leq O(n^2)\) always holds since the size of a cluster can never exceed the number of total records. Thus, using pair indexing dictionary can significantly enhance query performance. Figure 4.5 shows an example of pair indexing dictionary.

```json
weight vector dictionary =
{
  "(1,2)" = {"w_vec" = [1,1,0,8,1],
              "rule_id" = "1",
              "rule_type" = "hard+x",
              "rule_weight" = "1"},
  "(1,3)" = {
              ......},
  "(2,3)" = {
              ......}
}
```

Figure 4.4: Example of weight vector dictionary

```json
Pair indexing dictionary =
{
  "1" = ["(1,2)" : {{"w_vec"=[],...},
                 "(1,3)" : {{"w_vec"=[],...}}]
  "2" = ["(1,2)" : {{"w_vec"=[],...},
                 "(2,3)" : {}},
                 .....]
}
```

Figure 4.5: Example of pair indexing dictionary

### 4.3 Record Cluster Level

Record cluster level provenance stores information about the final clusters, and about how these clusters were generated. In this project, cluster dictionary and edge dictionary will be generated to store above provenance information.
4.3.1 Cluster Dictionary

The cluster dictionary holds information about the final clusters. As described in Chapter 3, since clusters are generated incrementally, the cluster dictionary will be modified accordingly. There are three cases:

- Creation of a new cluster $c_1$ with initial members $r_1$ and $r_2$. In this case, a key-value pair is created in which the key is cluster ID $c_1$ and the value is a list containing initial members $[r_1, r_2]$.

- Addition of a new member $r_1$ to an existing cluster $c_1$. In this case, $r_1$ is simply appended to the value of $c_1$.

- Merging of two existing clusters $c_1$ and $c_2$. In this case, the value of $c_2$ is appended to $c_1$. Following this, redundant records are eliminated and $c_2$ is deleted.

4.3.2 Edge Dictionary

The edge dictionary holds information about how clusters were generated. In the cycle-based clustering algorithm described above, an edge is defined as either a hard match or a converted hard match, on which creation, expansion or merging of clusters can be based. Thus, the recording of the edges of a cluster provides information about how that cluster was generated, and enables further operations such as reconstruction.

Edge dictionary is generated at clustering phase. Following this, the edge dictionary is modified at the same time as the corresponding cluster dictionary. Thus, there are also three cases:

- Creation of a new edge set $e_1$ with edge $(r_1, r_2)$. In this case, a key-value pair is created in which the key is the edge set ID $e_1$ and the value is edge $(r_1, r_2)$.

- Addition of a new edge $(r_1, r_2)$ to an existing edge set $e_1$. In this case, $(r_1, r_2)$ is appended to the value of $e_1$.

- Merging of two edge sets $e_1$, $e_2$ according to edge $(r_1, r_2)$. In this case, both the values of $e_2$ and the new edge $(r_1, r_2)$ are appended to the value of $e_1$, and $e_2$ is then deleted.

For a particular cluster, the same ID is used for both the cluster dictionary and the edge dictionary, in order to facilitate searching. The main idea is, on the creation of a cluster $c$, the edge set is also created with an ID of $c$. Then if modifications are carried out on any cluster $c$, the edge set with the same ID id is also modified. This guarantees that there is always exactly one cluster and one edge set for each cluster. Figure 4.6 gives examples of cluster dictionary and edge dictionary.
Figure 4.6: Example of cluster dictionary and edge dictionary
Provenance Queries

This chapter presents an explanation of the methods developed for querying the provenance storage structures discussed in Chapter 4. ER provenance query tool (ERPQT) was designed for graphical queries and data visualisation.

Query methods provide users with an easy way to access provenance data. Traditional query methods using fixed syntax such as SQL show good performance in relational databases. However, these fixed query languages are less efficient in certain applications. For instance, when data is managed by NoSQL, SQL is no longer suitable, and other query languages need to be applied. In addition, the use of SQL has limitations in terms of scalability, expandability and extensibility, thus restricting its use in queries.

To resolve above concerns, in this project, query methods for querying provenance data are proposed. Interfaces were created which allowed efficient access to provenance data, and enabled queries with flexible parameters. Three levels of queries were designed at individual record level, record pair level, and record cluster level.

5.1 Individual Record Level

Individual record level provenance includes data sources and blocking information. Two methods are proposed at this level: FindRecord and FindBlock.

5.1.1 FindRecord

In some cases, users may want to retrieve the raw data of one or more records. A simple approach is to open and search the original data set manually. However, this can be time-consuming when the scale of the data set is large. Thus, the method
FindRecord is proposed for users to efficiently access original data and obtain relevant provenance information in one query.

FindRecord extracts the required individual records from the original data set, and obtains the provenance information from data sources dictionary. This function accepts user input parameters in order to query the provenance data storage structure. If there is an ID attribute in the data set, the parameters can be a single record ID or multiple record IDs separated by commas. This function can also be used to query with particular attributes. If the ID attribute is missing from the dataset or the user has no information on the ID, either complete or partial information of the attributes and values of the records can be used as query parameters. In this case, the function accepts key-value pairs in the form ‘key : value’, in which key is the attribute names and value is the corresponding attribute value. Querying with multiple key-value pairs separated by commas is also supported. If the query succeeds, the function returns a list in which the elements are the information of individual records.

An example of this is as follows. For the data in Table 1.1, FindRecord(r1,r2) can be used to obtain the full records of r1 and r2. However, if IDs of records r1 and r2 are unknown, but the user has information on a person named Xu Wang with postcode 2602, a query can be carried out for this information, using FindRecord(name: Xu Wang, postcode: 2602). In both cases, FindRecord returns the first two records in Table 1.1. An example of executing FindRecord in the ERPQT is given in Figure 5.1.

![Figure 5.1: FindRecord](image-url)
CHAPTER 5. PROVENANCE QUERIES

5.1.2 FindBlock

As discussed in Chapter 3, blocking phase divides records into blocks, and is an important technique for reducing the time complexity of ER process. However, the use of unsuitable blocking methods and attributes may raise errors at this phase. A method of retrieving blocking provenance is therefore required in order to track these errors.

FindBlock uses the block dictionary and the block indexing dictionary to identify either which records are contained in a particular block or which block a particular record belongs to. It accepts unique identifiers of records as parameters. There are two usages of this method:

1. If the input is a single record ID or multiple record IDs separated by commas, FindBlock identifies the blocks which contain these input records, and returns a list of block IDs. For instance, for the data in Figure 4.2, FindBlock(1,2) returns \([\text{block}_1, \text{block}_2]\), indicating that record 1 is contained in \(\text{block}_1\) and record 2 is contained in \(\text{block}_2\).

2. If the input has the form \(\text{blk : } B\), FindBlock identifies all the members of block \(B\) and returns a list of members of that block. For instance, for the data in Figure 4.2, FindBlock(\(\text{blk : } \text{block}_1\)) returns \([1, 2, 3]\), indicating that \(\text{block}_1\) contains records 1, 2 and 3.

Figure 5.2 gives an example of FindBlock in ERPQT.

```
Figure 5.2: FindBlock
```
5.2 Record Pair Level

Pair level provenance contains provenance information about comparison and classification phase. *FindPair* retrieves the provenance data of these two phases.

There are numerous possible methods for implementing the comparison and classification phases in ER. Errors and inconsistencies may arise in these two phases. For instance, in this research, a rule-based algorithm is implemented in the classification phase. If an error is identified in the final result, a method is required for obtaining details of the weight vectors and rules used, in order to ascertain whether the rules have been set correctly.

*FindPair* is designed to handle this case. This function takes record identifiers as inputs, and the number of input identifiers specifies the usage of this function, as follows:

1. If the input contains only one record identifier $r_1$, *FindPair* returns a list of all pairs containing $r_1$ and their corresponding provenance information.

2. If the input contains exactly two record identifiers separated by a comma, for example $r_1, r_2$, *FindPair* returns the pair $(r_1, r_2)$ and its relevant provenance information if $(r_1, r_2)$ exists. Otherwise, *FindPair* returns an empty result.

If *FindPair* detects more than two parameters, it rejects this query and raises an exception, since the use of more than two parameters is inappropriate for record pair level query. An example of the use of *FindPair* for the record pair (12,14) in ERPQT is illustrated in Figure 5.3.

Figure 5.3: FindPair
CHAPTER 5. PROVENANCE QUERIES

5.3 Record Cluster Level

Provenance information at the record cluster level involves the cluster dictionary and the edge dictionary. Two methods are proposed for querying at this level: FindCluster and Reconstruct.

5.3.1 FindCluster

At cluster level, provenance data may need to be queried when a user requires the entity to which a particular record belongs. Alternatively, the members of a particular entity may also be required.

FindCluster is designed to identify the entity to which a record belongs, and all the records which belong to an entity. It accepts the unique identifiers of records as parameters. This function can be used in two ways, as follows:

1. If the input is a single record ID or multiple record IDs separated by commas, FindCluster returns the corresponding clusters for each record ID in the form of a list. For instance, for the data in Figure 4.6, FindCluster(1, 4, 6) returns [cluster_1, cluster_2, cluster_3], indicating that records 1, 4 and 6 belong to cluster_1, cluster_2 and cluster_3 respectively.

2. If the input has form 'clus : C', FindCluster returns a list of the members of cluster C. For instance, for the data in Figure 4.6, FindCluster(clus : cluster_2) returns [4, 5], indicating that the members of cluster_2 are record 4 and 5.

Figure 5.4 shows an example of FindCluster in ERPQT.

5.3.2 Reconstruct

In this project, the clustering phase is based on the cluster dictionary and edge dictionary generated at the classification phase. If an error arises in the cycle-based clustering algorithm, a method is required to determine the point at which this error occurred. For instance, in the ER result generated from the data set in Table 1.1, users find that record r_3 and r_5 are in the same cluster. However, it can be seen that r_3 and r_5 do not refer to the same person. In order to determine whether this error arose at the clustering phase, it will be necessary to inspect the details of how the clustering was carried out.

Reconstruct method allows users to reconstruct a cluster. Reconstruct accepts a cluster id c_1, and returns the sequence of generation of cluster c_1. Each element of the sequence consists of a record pair with its timestamp. For instance, for the data in Figure 4.6, Reconstruct(cluster_1) returns [[rec_tuple: (1, 2), timestamp: 1],
Figure 5.4: FindCluster

\[\text{rec\_tuple: (2, 3), timestamp: 1.2}\], indicating that at time 1, records 1 and 2 were added to cluster \emph{cluster\_1} according to edge (1, 2), and at time 1.2, record 3 was added to cluster \emph{cluster\_1} according to edge (2, 3).

The ERPQT allows users to generate both static figures and animations of reconstructions. For larger clusters, a visual presentation is more effective and convenient for users. Figure 5.5 illustrates an example of the reconstruction of a complex cluster in ERPQT.
Figure 5.5: Reconstruction
Chapter 6

Experiments

This chapter presents the results of experimental evaluations using two datasets, CORA and NCVR, to assess the performance of the proposed system in terms of both space and time.

6.1 Experimental Setup

The implementation was written in Python, with interpreter version 2.7.11. All programs were tested using a Mac OS X Yosemite machine with 2.5 GHz Intel Core i7 CPU and 16 Gbytes RAM.

6.1.1 Datasets

Two real-world datasets, CORA and NVCR, were used to evaluate the performance of the implementation. CORA contains the records of 1878 published articles on machine learning, and contains six attributes including name, authors, title, name, volume, and date. CORA is public available on the Internet. NCVR contains the profiles of voters in North Carolina, for this research, a subset containing 296,433 records of this dataset was used. Table 6.1 gives more details of both of these datasets. Table 6.1 gives detail of CORA and NCVR.

6.1.2 ER Rules

The rules used in the classification phase are given in Table 6.2. The comparison function \( \text{sim}(\) is used to calculate the similarity value of an attribute in the comparison phase. The q-gram function was used for all comparisons in this project.
CHAPTER 6. EXPERIMENTS

<table>
<thead>
<tr>
<th>Data set</th>
<th>Data set size(Mb)</th>
<th>Number of records</th>
<th>Number of blocks</th>
<th>Number of weight vectors</th>
<th>Number of clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>0.256</td>
<td>1,878</td>
<td>189</td>
<td>146,122</td>
<td>134</td>
</tr>
<tr>
<td>NCVR</td>
<td>53.5</td>
<td>296,433</td>
<td>168,531</td>
<td>1,487,913</td>
<td>117,511</td>
</tr>
</tbody>
</table>

Table 6.1: Characteristics of data sets used in my experiments

<table>
<thead>
<tr>
<th>Data set</th>
<th>Rule ID</th>
<th>Rule</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORA</td>
<td>u_1</td>
<td>sim(title) &gt; 0.6 ∧ sim(author) &gt; 0.3 ∧ sim(date) &gt; 0.3 ∧ sim(name) &gt; 0.1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>u_2</td>
<td>sim(author) &gt; 0.5 ∧ sim(name) &gt; 0.5</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>u_3</td>
<td>sim(title) &gt; 0.1 ∧ sim(name) &gt; 0.9 ∧ sim(vol) &gt; 0.7</td>
<td>0.9</td>
</tr>
<tr>
<td></td>
<td>u_4</td>
<td>sim(title) &lt; 0.4 ∧ sim(author) &lt; 0.3</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>u_5</td>
<td>sim(gender) = 1 ∧ sim(first_name) ≥ 0.8 ∧ sim(last_name) ≥ 0.8 ∧ sim(age) ≥ 0.7 ∧ sim(phone_number) = 1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>u_6</td>
<td>sim(first_name) ≥ 0.6 ∧ sim(last_name) ≥ 0.6 ∧ sim(age) ≥ 0.5</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>u_7</td>
<td>sim(first_name) ≥ 0.7 ∧ sim(zip_code) ≥ 0.7 ∧ sim(phone_number) ≥ 0.7</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>u_8</td>
<td>sim(first_name) ≥ 0.5 ∧ sim(last_name) ≥ 0.5 ∧ sim(zip_code) ≥ 0.8</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>u_9</td>
<td>sim(gender) &lt; 1 ∧ sim(phone_number) &lt; 1 ∧ sim(first_name) &lt; 0.2 ∧ sim(last_name) ≤ 0.2</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table 6.2: ER rules used in my experiments

6.1.3 Experimental Results

Experimental evaluations were carried out of the space and time efficiency of the proposed provenance storage structures and query methods respectively. For each data set, the program was run three times, and the average size of storage structures and the average loading and execution time of query methods were recorded. Each storage structure was stored on the machine described above with the file size recorded. Loading and execution time for each query method was recorded using the built-in time library in Python. The experimental results of CORA and NCVR are presented in Table 6.3 and Table 6.4 respectively.

The results in Tables 6.3 and 6.4 show that linear memory space is required for the storage of all provenance structures, with the exception of the pair indexing dictionary. A fixed memory size is required for the data source dictionary and pair level metadata dictionary, since these have pre-set structures with the same number of key-value pairs for each dataset. From Table 6.3 it can be seen that CORA pair indexing dictionary requires 42.6 Mbytes of memory space while the size of CORA data set is 0.256 Mbytes; the space complexity is $O(n^2)$. However, it can also be seen from Table 6.4 that the NCVR pair indexing dictionary requires only 118.5 Mbytes memory space while the size of the NCVR data set is 53.5 Mbytes; the space complexity in this case is approximately $O(n)$.

The reason for this difference is that the size of the pair indexing dictionary is determined by the number of weight vectors. Since comparisons are only performed within blocks, the number of weight vectors is determined by the size and number of the blocks. Table 6.1 shows that CORA contains only 189 blocks with 1,878 records.
in total, while the 296,433 records within the NVCR dataset are divided into 168,531 blocks. Thus, the average size of the blocks in the CORA dataset is nearly ten times larger than in NVCR, and the space complexity of the pair indexing dictionary for this dataset is much higher than for NVCR.

Furthermore, the number and size of blocks is determined by the blocking attributes and encoding methods. Thus, the space performance of the pair indexing dictionary may significantly differ between datasets or when varying ER settings for the same data set.

Tables 6.3 and 6.4 show that query time is less than 7 ms for the CORA dataset, and less than 1.2 s for NCVR. Loading and execution times are related to the total size of the storage structure used. For instance, FindRecord visits the original dataset and the data source dictionary, and both the total loading and execution times are directly proportional to the total size of original dataset and the data source dictionary. Also, FindPair only needs to visit the pair indexing dictionary, and the loading and execution times are therefore in direct proportion to the size of the pair indexing dictionary.
Chapter 7

Conclusion

This work presents a framework which allows the integration of data provenance techniques with ER process. A cycle-based algorithm is implemented at the clustering phase of the ER process, and data provenance structures are constructed at three different levels: individual record, record pair and record cluster. Query methods are also presented for retrieving provenance information at each level. In addition, a graphical interface is designed for the visualisation of queries and retrieved data.

The results of experimental evaluations of the proposed framework using two real-world data sets CORA and NCVR are described. The results demonstrate that the proposed storage structures and query methods are efficient in terms of both space and time.
Appendices
Appendix A

Independent Study Contract
INDEPENDENT STUDY CONTRACT

Note: Enrolment is subject to approval by the projects co-ordinator

SECTION A (Students and Supervisors)

UnitID: U 5935 088
Surname: Wang
First Names: Xu
Project Supervisor (may be external): Qian Wang
Course Supervisor (a RSCS academic): Weifa Liang
Course Code, Title and Unit: COMP 4560 12 units

Semester: [ ] S1  [X] S2 Year: 2016

Project Title:
see attached

Learning Objectives:
see attached

Project Description:
see attached
APPENDIX A. INDEPENDENT STUDY CONTRACT

ASSESSMENT (as per course's project rules web page, with the differences noted below):

<table>
<thead>
<tr>
<th>Assessed project components:</th>
<th>% of mark</th>
<th>Due date</th>
<th>Evaluated by:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report: name style: report</td>
<td>45%</td>
<td></td>
<td>kerry Taylor</td>
</tr>
<tr>
<td>(e.g. research report, software description...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Artifact: name kind: software</td>
<td>45%</td>
<td></td>
<td>Greg Wang</td>
</tr>
<tr>
<td>(e.g. software, user interface, robot...)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Presentation:</td>
<td>10%</td>
<td></td>
<td>Wenfa Liang</td>
</tr>
</tbody>
</table>

MEETING DATES (IF KNOWN):

STUDENT DECLARATION: I agree to fulfil the above defined contract:

_________________________  22/07/2016
Signature               Date

SECTION B (Supervisor):

I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project.

_________________________  22/07/2016
Signature               Date

REQUIRED DEPARTMENT RESOURCES:

SECTION C (Course coordinator approval)  21-7-16

Signature            Date

SECTION D (Projects coordinator approval)

Signature            Date

Research School of Computer Science  Form updated Jun-12
Appendix B

Project Description
1 Project Title

Entity Provenance: Data Provenance Support in Entity Resolution

2 Project Description

Provenance has been identified as a major requirement for data analytics applications, but is still largely unexplored [3]. The goal of the project is to develop techniques for capturing, storing, and querying the provenance information of entity resolution, which help us keep track of when and how an entity was resolved. This would enable transparency and reproducibility of entity resolution results and allow a user to understand, identify and improve imprecise entity resolution results. The specific tasks are:

1. Conduct a literature review on entity resolution and data provenance techniques.
2. Develop storage structures that can be used to capture provenance information of entity resolution.
3. Develop methods to query provenance information of entity resolution which can provide a detailed overview on the history of an entity.
4. Evaluate the effectiveness and efficiency of the developed methods over two real-world data sets CORA and NCVR:
   - The CORA data set contains 1,878 machine learning publications and is publicly available together with its ground truth,
   - The NCVR data set is a public voter registration data set from North Carolina.
5. Write up a project report.

This project will be based on the following articles:


3 Learning Objectives

On the completion of the project, the following learning objectives are expected to achieve:

- Have a good understanding for the literature of entity resolution and data provenance techniques;
- Develop methods to store and query the provenance information of entity resolution;
- Be able to conduct experiments and analyze their results;
- Be able to effectively communicate about data and project results understandably, using adequate indicators, tables, and graphs.
Appendix C

Software Description

The software structure are described as follows:

- Directly used modules
  1. ./WeightVectors
  2. ./febrl

- Modified other’s modules
  1. ./rule_based_classification.py
  2. ./rules.py
  3. ./ER.py

- Original modules
  1. ./test.py
  2. ./dataprovenance.py
  3. ./cir_clustering_funs.py
  4. ./pyqtGUI.py
  5. ./query_handler.py
  6. ./visualizer.py

To test the correctness of ER process, the code was run for three times on both CORA and NVCR data set. To test the correctness of Entity Resolution Provenance Query Tool, all buttons, comboboxes and text input area were manually tested three times. All of the tests executed successfully, and all the designed functionalities were fully achieved. The procedure of testing the software can be found in Appendix D.
All experiments were conducted in a Mac OS X Yosemite machine with 2.5GHz CPU and 16GB RAM. Two data sets provided by Dr. Qing Wang, CORA and NCVR, were used for experimental evaluation. The implementation was written in Python programming language, with interpreter version 2.7.11.
Appendix D

Readme
Entity Resolution Provenance Query Tool (ERPQT)

Project Description:

This software is a framework of provenance support in entity resolution. It provides users with a general ER process and corresponding provenance data structures and query methods. Also, a graphical user interface (GUI) was also included to support data visualisation and graphical query.

Setup

1. This software was tested to run successfully in Mac OS X Yosemite. To run the code, set up following dependencies first:

   1. Python 2.x interpreter. Can be found in https://www.python.org/downloads/. The version should be no earlier than 2.7.11

   2. PyQt4. https://www.riverbankcomputing.com/software/pyqt/download. Alternatively, this package can be download through homebrew or macports


   For users who want to use the GUI, dependency 1, 2, 3 are all required. Other users only need to set up dependency 1 correctly.

2. Check whether the two data sets cora-publication.csv and ncvoter.csv are in the ./datasets folder.

Run

1. To run the software in the terminal, go to the ERPQT directory and run:
   
   ~ERPQT$ python test.py [dataset]
   
   [dataset] specifies in which data set do you want to run a ER progress. Currently supported options are:

   cora     - Run ER progress in CORA data set
   ncvr     - Run ER progress in NCVR data set

2. To run the GUI, go to the ERPQT directory and run:
   
   ~ERPQT$ python pyqtGUI.py
If the interpreter gives error "unknown locale: UTF-8", add following lines to your "/.bash_profile:
   export LC_ALL=en_US.UTF-8
   export LANG=en_US.UTF-8

More detailed instruction of the GUI can be found in the attached user manual.

Project Structure

1. ./WeightVectors
   This is a third-party library for blocking phase.

2. ./febrl
   This is a third-party library for comparison phase.

3. ./rules
   The folder stores the rules used in classification phase.
   rules.csv The file contains rules.

4. ./datasets
   The folder stores the data sets.
   cora-publication.csv CORA data set
   ncvoter.csv NCVR data set

5. ./result
   The folder stores the provenance data generated in a ER process, including
   record/pair/cluster level provenance data, metadata and classified
   weight vectors.
   output.csv Raw weight vectors
   classified.pkl Classified weight vectors
   record_level.pkl Record level provenance
   pair_level.pkl Pair level provenance
   cluster_level.pkl Cluster level provenance
   metadata_level.pkl metadata

6. ./tmp
   The folder stores the intermediate results generated in a ER process, including
   hard match set, hard non-match set and soft set.
   hardmatchset.pkl The file stores hard matches
APPENDIX D. README

hardnonmatchset.pkl  The file stores hard non-matches
softset.pkl          The file stores soft matches/non-matches

7. ./rule_based_classification.py & ./rules.py
   The module of modified rule-based classification algorithm.

8. ./test.py
   The entry of running a ER progress.

9. ./ER.py
   The module of ER progress, contains basic functions for ER process.

10. ./dataprovancenace.py
    The module of running a ER process, performs ER process.

11. ./cir_clustering_funs.py
    The module of circle-based clustering

12. ./pyqtGUI.py
    The main body of the GUI

13. ./query_handler.py
    The module provides query methods.

14. ./visualizer.py
    The module of data visualization.

Known Issues:

1. The program may face a infinite loop when plotting data, possible solution is abort the program and restart.

2. When dealing with large scale data sets (e.g. NCVR), the execution time could be long.
Bibliography


