# CLUSTERING AND ENTITY RESOLUTION FOR SEMI-STRUCTURED DATA

by

CHUAN ZHAO

A dissertation submitted in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

WASHINGTON STATE UNIVERSITY
Department of Electrical Engineering and Computer Science

DECEMBER 2011

To the Faculty of Washington State University:

The members of the Committee appointed to examine the dissertation of

CHUAN ZHAO find it satisfactory and recommend that it be accepted.

Krishnamoorthy Sivakumar,
Ph.D., Chair

Lawrence B. Holder, Ph.D.

Ananth Kalyanaraman, Ph.D.

## ACKNOWLEDGMENTS

I wish to thank all those who have assisted and supported me during this research undertaking. I owe my deepest gratitude to my major adviser Dr. Krishnamoorthy Sivakumar, who gave me precious opportunities to do research on a variety of projects with him, sparked my interest in my research area, encouraged and guided me to complete my dissertation at each stage, and took on the arduous task of checking, improving and polishing my work. It is one of the biggest academic challenges in my life to write the dissertation for my Ph.D. degree. Dr. Sivakumar has gone far and above what was expected of him as a major professor, by checking every word of my draft and pointing out every mistake I made. It would be impossible for me to complete this dissertation without Dr. Sivakumar's patient advice and support. I will always be grateful to him and regard him as the most important person in my academic life.

I would like to gratefully and genuinely thank Dr. Lawrence Holder and Dr. Ananth Kalyanaraman, my committee members, for generously providing comments and suggestions which enabled me to gain different perspectives to the issues of my research. Dr. Holder and Dr. Ananth provided many constructive guidance for

my dissertation proposal which were very important to my research work after the preliminary exam.

I am also grateful to the Information Services Group staff and all other staff members in the department. Your kind help and support are always appreciated. I also wish to thank all of my friends and colleagues. I will never forget the memories that we have shared, studying together in classrooms, working side by side in the office and lab. These are the greatest memories I could ask for.

Most importantly, I want to express my heartfelt gratitude to my dearest parents for their unwavering support of my studies and for their deep love and trust in me. I thank my family for a lifetime of encouragement which has helped me realize my potential. I feel fortunate to have been raised in an environment filled with such endless and harmonious love. Although my father already left me for more than six years, his affable smile has always been my biggest impetus when I was faced with difficulties. I would like to deeply thank my mother for her painstaking care all the time. Her maternal love is always the most selfless and greatest love in the world.

CLUSTERING AND ENTITY RESOLUTION FOR SEMI-STRUCTURED DATA

Abstract

by Chuan Zhao, Ph.D.
Washington State University
December 2011

Chair: Krishnamoorthy Sivakumar

Using a graph representation of the data, a graph-based similarity measure to assess the similarity between data records is proposed. Both direct and indirect similarity are considered, which comprehensively capture the relationship between data records. Different data mining techniques and applications, including clustering and entity resolution are explored.

First, the problem of clustering is considered for a dataset consisting of non-numeric attributes. The $K$-medoid clustering algorithm is used; some postprocessing steps are introduced to improve the quality of clustering. A set of validity indices are proposed to assess the quality of the clustering results. To reduce computational complexity, a sampling strategy is introduced. Effect of sampling on the values of validity indices and clustering result is discussed. Influence of different similarity

measures, postprocessing steps, and cluster numbers on the quality of clustering is discussed, both analytically and experimentally. Similar enhancements to the fuzzy $K$-medoid algorithm are provided. The clusters resulting from the proposed algorithm can sometimes be interpreted as grouping objects sharing a common attribute that was not used in the clustering algorithm. A multi-medoid $K$-medoid algorithm is proposed by introducing multiple medoids in each cluster to enhance the performance of the $K$-medoid algorithm. Finally, an optional node move step is introduced to produce better clustering results based on edge-oriented evaluation measures.

The entity resolution problem, which is the process of determining whether multiple records refer to the same real world entity, is studied next. It is an important step during data cleaning and integration. A general entity resolution framework called ERUDITE, which includes data preprocessing (filtering), record matching, and postprocessing (inconsistency elimination, record updating, and equivalent record elimination), is presented. Different record matching models are explored for both supervised and unsupervised learning methods. Two record updating algorithms are proposed to significantly improve the entity resolution result. The entity resolution result generally contains inconsistent decisions. New inconsistency elimination methods are proposed and their performances are compared with that of existing methods. Experiments for both unsupervised and supervised learning on two public datasets show the good performance of the proposed framework.

# TABLE OF CONTENTS

ACKNOWLEDGMENTS ..... iii

ABSTRACT ..... v

LIST OF TABLES ..... ix

LIST OF FIGURES ..... xi

1. Introduction ..... 1

    1.1 Data Similarity ..... 2
    1.2 Clustering ..... 15
    1.3 Entity Resolution ..... 22

2. Definitions ..... 36

3. Similarity Measure ..... 41

    3.1 Data representation ..... 41
    3.2 Direct similarity metric ..... 44
    3.3 Domain-optimized direct similarity calculation ..... 48
    3.4 Indirect similarity measure ..... 49
    3.5 Modified Version for Clustering ..... 52

4. Clustering ..... 55

    4.1 $K$-medoid Clustering ..... 56
    4.2 Multi-Medoid $K$-medoid Clustering ..... 73
    4.3 Edge-Oriented Node Move Algorithm ..... 76
    4.4 Interpretation of Clustering Result with Domain Independent Similarity Measure ..... 80

4.5 Fuzzy $K$-medoid Clustering ..... 80
4.6 Visualization Tool ..... 84

5. Entity Resolution ..... 94

5.1 Problem definition ..... 94
5.2 Framework ..... 95
5.3 Preprocessing ..... 98
5.4 Record matching ..... 105
5.5 Postprocessing ..... 112

6. Prediction of Movie Rating ..... 129

6.1 Rounding ..... 129
6.2 $K$-Nearest-Neighbor Method ..... 130

7. Experimental Results ..... 132

7.1 Clustering ..... 132
7.2 Entity Resolution ..... 159
7.3 Prediction of Movie Rating ..... 173

8. Conclusion ..... 176

# LIST OF TABLES


Table Page

4.1 Notations used in validity indices ..... 56

4.2 Notations used in evaluation measures for fuzzy $K$-medoid ..... 82

6.1 IMDB movie rating rounding method ..... 129

7.1 Average $CoS$ values ..... 138

7.2 Clustering result compared with "perfect" clustering for movies from year 1987 to 2006 ..... 150

7.3 Clustering result compared with "perfect" clustering for movies fromo year 1991 to 1995 ..... 151

7.4 Running time (second) of our $K$-medoid algorithm vs. number of medoid in each cluster ..... 153

7.5 Clustering result compared with SA-Cluster in Zhou et al. [2009] for political blogs ..... 155

7.6 Clustering result compared with MLR-MCL in Satuluri and Parthasarathy [2009] for Cora paper citation network ..... 156

7.7 Clustering result compared with MLR-MCL in Satuluri and Parthasarathy [2009] for Epinions social network ..... 157

7.8 Clustering result compared with Newman and Girvan [2004] for dolphin social network ..... 157

7.9 Clustering result compared with Newman and Girvan [2004] for condensed matter collaboration network ..... 158

7.10 Running time (second) of our $K$-medoid algorithm and node move algorithm on political blogs dataset ..... 158

7.11 Running time (second) of our node move algorithm on different datasets ..... 158

7.12 Unsupervised learning: Using record updating to get best threshold . 165

7.13 Unsupervised learning results with best matching threshold ..... 165

7.14 Unsupervised learning results without TC ..... 167

7.15 Unsupervised learning: running time ..... 167

7.16 Supervised learning: 5-fold cross validation with/without domain-optimized direct similarity calculation, using direct similarity only .. 168

7.17 Supervised learning: 5-fold cross validation with/without domain-optimized direct similarity calculation, using both direct and indirect similarity ..... 169

7.18 Supervised learning: 3-fold cross validation with/without domain-optimized direct similarity calculation, using direct similarity only .. 169

7.19 Supervised learning: 3-fold cross validation with/without domain-optimized direct similarity calculation, using both direct and indirect similarity ..... 170

7.20 Supervised learning: inconsistency elimination ..... 171

7.21 Supervised learning: inconsistency elimination with different methods 172

7.22 Supervised learning: Cora, inconsistency elimination vs. group size . 173

7.23 Supervised learning: Cora, compare with other methods ..... 173

7.24 Supervised learning: running time (second) ..... 174

# LIST OF FIGURES


| Figure | | Page |
|---|---|---|
| 3.1 | Data representations | 42 |
| 3.2 | Graph explanation of similarity measures | 45 |
| 3.3 | Sets $x$, $y$, and $z$ | 47 |
| 4.1 | Flow chart for $K$-medoid clustering algorithm | 64 |
| 4.2 | Flow chart for fuzzy $K$-medoid clustering algorithm | 83 |
| 4.3 | Cluster visualization tool: cluster | 87 |
| 4.4 | Cluster visualization tool: node | 88 |
| 4.5 | Cluster visualization tool: edge | 89 |
| 4.6 | Cluster visualization tool: search | 90 |
| 4.7 | Cluster visualization tool: dominant node | 91 |
| 4.8 | Cluster visualization tool: zoom | 92 |
| 4.9 | Cluster visualization tool: statistics | 93 |
| 5.1 | Framework of ERUDITE | 96 |
| 5.2 | Illustration of linear interpolation method | 109 |
| 7.1 | Initial cluster number vs. similarity measure vs. $CS$ | 136 |
| 7.2 | Initial cluster number vs. similarity measure vs. $RoM$ | 136 |
| 7.3 | Initial cluster number vs. similarity measure vs. $CoS$ | 137 |
| 7.4 | Cluster numbers vs. Iteration for different similarity measures and initial cluster numbers: $K$-medoid clustering | 139 |
| 7.5 | Illustration of Theorem 4.1.2 | 140 |

7.6 Effect of Actor-Role pairs on cluster validity indices ..... 140

7.7 Clustering result for Enron dataset ..... 141

7.8 Sampling percentage vs. sampling strategy vs. sampling error ..... 142

7.9 Sampling percentage vs. sampling strategy vs. clustering similarity . 142

7.10 Entropy/Purity vs. similarity measures vs. number of medoid in each cluster ..... 149

7.11 Validity indices vs. similarity measures vs. number of medoid in each cluster ..... 152

7.12 Unsupervised learning: Influence of matching thresholds on F1 (left: Cora, right: CDDB) ..... 163

7.13 Unsupervised learning: Influence of matching thresholds on number of unique records with equivalent(s) (left: Cora, right: CDDB) ..... 164

7.14 Nonrounding classification result ..... 174

7.15 Rounding classification result ..... 175

**Dedication**

*To my parents*

# CHAPTER 1. INTRODUCTION

The data to be mined in data mining could be in various formats: unstructured, semi-structured and structured. Un-structured data can be of different types, like text, video, image, etc. Semi-structured data are often in XML or other markup language format. Structured data have the same defined format and are described by a data model such as a database model or an entity-relationship model. Semi-structured data can be naturally represented by a graph with nodes representing entities and edge/link representing relationship between entities. Mining data in a graph format can provide a number of benefits from the natural characteristics of a graph.

In our research, the data in the datasets are semi-structured, which contain tags or other markers to separate semantic elements and enforce hierarchies of records and fields within them. For example, we used the IMDB movie XML data, the ENRON E-mail data, and the Cora citation XML data.

In this chapter we will introduce our data similarity measures, clustering methods, entity resolution framework, and discuss their related work.

## 1.1 Data Similarity

It is easy for a human being to check if two data records look similar. But if you ask how similar they are, the answer is generally one of the following: "exactly the same," "very similar," "kind of similar," "not very similar," "not similar at all," etc. These qualitative descriptions are not enough — we need quantitative values to accurately measure the similarity. Similarity measure is a quantitative measure of "closeness" between data records and plays an important role in our data mining research. Different similarity measures can lead to very different content and quality of the data mining result. We propose a graph-based similarity measures in which both similarity between simple attributes (direct similarity) and similarity between data objects (indirect similarity) are considered.

Data records are often described by their attribute values, which typically are strings and numbers, so similarity measures usually rely on string comparison techniques. Many techniques have been applied for matching attributes (fields) with string data in the equivalent record detection context. Elmagarmid et al. [2007] concluded the field matching techniques as follows:

- Character-Based Similarity Metrics: The character-based similarity metrics are designed to handle typographical errors well. They include edit distance, affine gap distance, Smith-Waterman distance, Jaro distance metric, and Q-gram dis-

tance, etc.

- Token-Based Similarity Metrics: It is often the case that typographical conventions lead to rearrangement of words (e.g., "John Smith" versus "Smith, John"). In such cases, character-level metrics fail to capture the similarity of the entities. Token-based metrics try to compensate for this problem. Atomic Strings, WHIRL, and Q-Grams with tf.idf are examples of this technique.

- Phonetic Similarity Metrics: Character-level and token-based similarity metrics focus on the string-based representation of the database records. However, strings may be phonetically similar even if they are not similar at a character or token level. The phonetic similarity metrics try to address such issues and match such strings.

While multiple methods exist for detecting similarities of string-based data, the methods for capturing similarities in numeric data are rather primitive. Typically, the numbers are treated as strings (and compared using the metrics described above) or simple range queries, which locate numbers with similar values (Elmagarmid et al. [2007]).

Field (attribute) matching is the basis for measuring the similarity between data records. The next step is to measure similarity between records with multiple attributes. Commonly used methods include weighted combination of attribute

similarities, probabilistic matching model and supervised learning. We use weighted combination of attribute similarities for direct similarity, and use supervised probabilistic matching model on both direct and indirect similarities.

We know that distance measures are known as metrics if they satisfy positivity, symmetry and triangle inequality. But for similarities, the triangle inequality typically does not hold; while symmetry and positivity typically do. Apparently, the word "similar" satisfies reflectiveness and symmetry, too. But, it does not necessarily satisfy transitivity. In other words, $a$ is similar to $b$ and $b$ is similar to $c$ do not mean that $a$ must be similar to $c$, because the common part of $a$ and $b$ could be different or only share a little with the common part of $b$ and $c$.

Existing similarity measures are based on two different aspects: one is topology (graph) structure, such as data node, edge, node degree, and neighbor; the other is data node content, for example node attribute vector. Some methods combine both aspects, for example our method introduced in Chapter 3.

In the real world, data similarity could be a complicated thing. Let's think about movie similarity:

- The movie similarity could be based on users' interest/taste and may or may not make use of graph structure. For example: 1) Based on users ratings given movies' attributes: year, genre, actor, director, etc.; 2) Based on users taste: given a user's favorite movies, find this user's potential other favorite movies,

or given movies a user dislikes, find other potential movies that this user dislike either; 3) User likes some movie may be just because he/she likes some very unimportant actor/actress in it.

- Only based on movies' attribute values, calculate their similarity.
- Only based on node/link structure, for example centrality.
- Based on movies' both attribute values and graph structure.
- For each attribute of some movie, find movies similar to it, then combine these movies to get the similar movies.
- For some given movie combine the attribute values into a single value, then find movies with similar value.
- For two similar movies, they may have some common actors, which are common attribute node neighbors of these two movie nodes. The weight contribution to similarity for each common actor neighbor node should be based on two things: one is the node degree of this actor node: the bigger the degree is, the smaller the weight is; the other is the role this actor played in these two movies: the more important the role is, the bigger the weight is.
- Many methods use only 2-step intermediate common neighbor nodes. It is easy to see that $n$ steps intermediate nodes can be used. For each step, node type

may or may not be specified.

The existing methods for data similarity generally calculate the similarity in following ways:

- Compare the contents of the data nodes.
- Make use of the nodes/edges connected to them: 1) the number of nodes/edges they share; 2) adjacency matrix or incidence matrix; 3) make use of bipartite graph, like hub/authority nodes in citation relationship: similar to pageRank method of Google search engine; or customer/product nodes in purchase relationship: customer are similar if they purchase similar products and products are similar if they are purchased by similar customers.

In our method, we consider attribute values for direct similarity, and graph structure for indirect similarity. For direct similarity, we use the weighted combination of attribute similarities, and omit the weight contribution factor for different nodes of each attribute. For indirect similarity, we consider the weight contribution of different neighbor nodes, and we use both 2-step intermediate common neighbor nodes and 3-step intermediate neighbor nodes on two different kinds of paths.

### *1.1.1 Related Work*

Cook and Holder [2007] described the similarity measure between two entity clusters as a weighted combination of the attribute similarity and graph-based similarity between them. This similarity appears similar to our similarity measures, but they have following differences: 1) our similarity measures consider node similarity, not cluster similarity; 2) our data representation includes attribute nodes, not just data nodes; 3) our indirect similarity through common neighbor data nodes (see section 3.1 and 3.2 for the definition of attribute node, data node, and neighbor data node) does not consider the data nodes which are derived by the common attribute nodes since they are already used for the direct similarity; the metric in Cook and Holder [2007] can not or did not do this with the reference graph.

Blondel et al. [2004] calculated the similarity matrix for each node pair in a graph with adjacency matrix for synonym extraction and web searching. They generalized "hubs and authorities" method of Kleinberg [1999]. Their method is as follows: Let $G_A$ and $G_B$ be two directed graphs with respectively $n_A$ and $n_B$ vertices. They defined a $n_B \times n_A$ similarity matrix $S$ whose real entry $S_{ij}$ expresses how similar vertex $j$ (in $G_A$) is to vertex $i$ (in $G_B$): we say that $S_{ij}$ is their similarity score. The similarity matrix can be obtained as the limit of the normalized even iterates of $S(k+1) = B \times S(k) \times A^T + B^T \times S(k) \times A$ where $A$ and $B$ are adjacency matrices

of the graphs and $S(0)$ is a matrix whose entries are all equal to one. In the special case where $G_A = G_B = G$, the matrix $S$ is square and the score $s_{ij}$ is the similarity score between the vertices $i$ and $j$ of $G$.

Park and Seo [2005] developed a search system that can find XML documents that matches the structure and content of a given XML document. They used XML tag name, tag value, and tag structures in their similarity measure. $NodeSim(e1, e2) = w_1 \times TagSim(e1, e2) + w_2 \times ValueSim(e1, e2)$, where $w_1 + w_2 = 1$. When a tag or a value is a substring of the other, the returned value of $TagSim$ or $ValueSim$ is 1. Node similarity was used to calculate the document similarity used for XML document search.

Huang and Lai [2006] proposed a node structural metric to measure the similarity between nodes which made use of the number of shared edges. They used this metric to cluster graph to simplify the graph representation. The similarity degree between two nodes is partly determined by the number of edges between them. In particular, the greater the number of the edges the two nodes share, the more similar they are. At the same time, the greater the number of edges they do not share, the less similar they are. Jaccard coefficient is able to measure the degree of overlap, which is defined as: $sim(a, b) = \frac{\#(a_i=b_i=1)}{\#(a_i=1)+\#(b_i=1)-\#(a_i=b_i=1)}$, where $a$ and $b$ are binary vectors. For instance, the numerator in the equation denotes the number of an attribute $i$ (i.e., edge) occurring in both $a$ and $b$. An edge-by-node matrix $R$ (also called

an incidence matrix) of $G$ is defined as $R = (r_{ij})_{|E|\times|V|}$, each entry of which is constructed by $r_{ij} = 1$ if node $v_j$ is incident with edge $e_i$) or $r_{ij} = 0$ otherwise. If $r_i$ and $r_j$ are $i$th and $j$th column vector of $R$ ($r_i$ and $r_j$ are node vectors of node $i$ and $j$.), then $sim(r_i, r_j) = \frac{r_i^T r_j}{r_i^T r_i + r_j^T r_j - r_i^T r_j}$. Note that the more similar two nodes are, the less links that connect them. The degree of similarity of two nodes connecting only one edge, for example, will reach the maximum, i.e., 1. In the context of graph clustering, their intention was to group two nodes with many linking edges. This means that they attempted to find those nodes with minimal similarities among them.

Narayanan and Karp [2007] compared the protein interaction networks of two species to detect functionally similar (conserved) protein modules between them. In their similarity measure, sequence similarity defined node similarity. Given as input two graphs and a node similarity function $sim()$, the function $sim(u, v)$ is true whenever node $u$ is similar to node $v$ (e.g., based on sequence similarity of proteins) and false otherwise. (a) $sim(u, v)$ is true whenever the BLAST E-value of proteins $u$ and $v$ is at most $10^{-7}$ and each protein is among the 10 best BLAST matches of the other; (b) $sim(u, v)$ is true whenever the BLAST E-value of $u$ and $v$ is less than that of 60% of ortholog pairs in some ortholog database.

Loos et al. [2003] investigated whether a computer can recognize disease-specific facial patterns in unrelated individuals. The graph nodes are labeled with a set of feature vectors, the so-called jets. A jet contains local texture information and is

calculated with a set of Gabor wavelets of different spatial sizes and orientations. The similarity between two jets is calculated by their normalized scalar product. The node similarity is the maximum of all jet similarities at the node.

Berg and Lässig [2006] used a simple binary approximation of node similarity: two genes are counted as orthologous (node similarity $\theta_{ij} = 1$) if they appear as putative orthologs in the Ensemble database, and otherwise not ($\theta_{ij} = 0$). Each node may have several such putative orthologs.

Simsek and Jensen [2005] studied how to find a target node in a network whose topology is known only locally. They proposed that larger the similarity between two nodes, higher the probability of having a direct link between them. In similarity-based navigation, nodes forward the message to the target node, given a number of attributes on nodes and a similarity metric. Statistical relationship between node similarity and probability of a link allow them to compute the probability that a given neighbor links to the target node. They treated the citation graph as an undirected network, defining node similarity using paper titles and abstracts. The title and abstract of each paper were represented as weighted-term vectors using TFIDF (Term Frequency - Inverse Document Frequency) weighting. Paper similarity was computed using a standard cosine correlation measure.

Smeaton and Morrissey [1995] computed node-node similarity values using standard information retrieval techniques. Their method was based on global vector

matching between texts but combined with locally matching structures like sentences. In calculating the similarity between pairs of Software Product Descriptions (SPDs) where each SPD was composed of a series of sub-nodes, the popular TFIDF weighting formula was used. Each index term was weighted by its inverse document frequency weight and the score between pairs of SPDs was computed by aggregating the scores between sub-nodes in each of the SPDs being matched.

Kleinberg [1999] proposed a link-based model for the conferral of authority, and showed how it leads to a method that consistently identifies relevant, authoritative www pages for broad search topics. Their model was based on the relationship that exists between the authorities for a topic and those pages that link to many related authorities — they referred to pages of this latter type as hubs. Hubs and authorities exhibit what could be called a mutually reinforcing relationship: a good hub is a page that points to many good authorities; a good authority is a page that is pointed to by many good hubs. The graph of Hubs and authorities is bipartite: one partite set is hub nodes, the other is authority nodes. If $p$ is highly referenced page, in the local region of the link structure near $p$, the strongest authorities can potentially serve as a broad-topic summary of the pages related to $p$.

Lu et al. [2001] explored the definition of similarity based on connectivity only, and proposed several algorithms for this purpose. Their metrics took advantage of the local neighborhoods of the nodes in the networked information space. Two variations

of similarity estimation between two nodes were described: one was based on the separate local neighborhoods of the nodes, and the other was based on the joint local neighborhood expanded from both nodes at the same time. They designed two graph-based similarity metrics. One metric was based on how strongly two papers were connected in the citation graph. The other metric measured how similar the two local citation communities are. The first metric was computed using a maximum flow / minimum cut value, using an efficient graph-theoretic algorithm, to find out the amount of flow that can be pushed from one paper to the other. Obviously the more flow which can be pushed from source to sink, the more paths there are between them, which means the stronger they are connected. In the second metric, they used vectors to compute similarity. In each local citation graph they assigned an authority weight to every node. Then they constructed two vectors of the nodes in both local citation graphs, the elements of which are the authority weights. The original idea behind this scheme was that two papers are similar if their individual local citation graphs share similar authority papers (compute the cosine distance of the two corresponding vectors).

Penner et al. [2008] defined twinness as a measure of node similarity within local structures in networks: two nodes are twins in subgraph $H$ if they have identical neighbors in $H$. In Schubert et al. [2005], a node $v \in V$ from the first document $D = (V, E)$ is called similar (denoted $v \sim v'$) to a node $v' \in V'$ from the second

document $D' = (V', E')$ if we allow that $v$ can be mapped to $v'$ in a node set correspondence. Hence, the similarity relation $\sim$ is just the union of all acceptable node set correspondences, and each one-to-on mapping that is a subset of $\sim$ is a possible solution. Leicht et al. [2006] assumed that the edges in a network themselves indicate a similarity between the vertices they connect. In their method, vertices $i$ and $j$ are similar if either of them has a neighbor $v$ that is similar to the other.

Jeh and Widom [2002] measured the structural-context similarity using bipartite similarity rank. Their basic idea was that two objects are similar if they are related to similar objects. For a node $v$ in a graph, they denoted by $I(v)$ and $O(v)$ the set of in-neighbors and out-neighbors of $v$, respectively. Individual in-neighbors are denoted as $I_i(v)$, for $1 \leq i \leq |I(v)|$, and individual out-neighbors are denoted as $O_i(v)$, for $1 \leq i \leq |O(v)|$. They defined basic SimRank as $s(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$, where $C$ is a constant between 0 and 1. If $a = b$ then $s(a,b)$ is defined to be 1. If either $a$ or $b$ does not have any in-neighbors then $s(a,b) = 0$. In the definition of bipartite SimRank such as for customers and products, similarity of products and similarity of customers are mutually-reinforcing notions: customers are similar if they purchase similar products, and products are similar if they are purchased by similar customers. Let $s(A,B)$ denote the similarity between customer $A$ and $B$, and let $s(c,d)$ denote the similarity between items $c$ and $d$. They have: $s(A,B) = \frac{C_1}{|O(A)||O(B)|} \sum_{i=1}^{|O(A)|} \sum_{j=1}^{|O(B)|} s(O_i(A), O_j(B))$, and

$s(c,d) = \frac{C_2}{|I(c)||I(d)|} \sum_{i=1}^{|I(c)|} \sum_{j=1}^{|I(d)|} s(I_i(c), I_j(d))$, where $C_1$ and $C_2$ are constants. Then they defined bipartite SimRank in homogeneous domains such as web pages and scientific papers as: $s_1(a,b) = \frac{C_1}{|O(a)||O(b)|} \sum_{i=1}^{|O(a)|} \sum_{j=1}^{|O(b)|} s_2(O_i(a), O_j(b))$, and $s_2(a,b) = \frac{C_2}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s_1(I_i(a), I_j(b))$. Depending on the domain and application, either score or a combination may be used.

Lifshits [2007] discussed similarity search in bipartite graphs. For a bipartite graph with people and movies in each partite, the person-person similarity is the number of 2-step (person to movie then to person) chains in the graph. Person-movie similarity is the number of 3-step chains. So to find the most similar person or movie to a person $q$ is to find person or movie with maximal number of 2-step or 3-step chains to $q$.

Li et al. [2006] proposed a new method based on semantic pathway covering for semantic similarity between gene ontology terms. An algorithm, COMBINE, was presented, which considered information contents of two given nodes and those of all nodes included in the two nodes' pathways. The information contents of the intersection and union of the two pathways were calculated respectively and then the ratio of these two information contents was used as the similarity value of the two nodes.

SNN (shared nearest neighbor) is the number of shared neighbors as long as the two objects are on each other's nearest neighbor lists. The underlying proximity

measure can be any meaningful simiarity or dissimilarity measure (Tan et al. [2006]). The Jarvis-Patrick clustering algorithm uses this SNN similarity.

## 1.2 Clustering

Data mining and knowledge discovery in databases often requires dividing data into groups where similar objects are assigned to the same group. This task is often executed by clustering. Clustering groups objects with high similarity into a single cluster and separates objects with low similarity into different clusters. Data that requires clustering is often unstructured or semi-structured, consisting of non-numeric attributes (e.g., movie data, E-mail data, news articles). A movie contains a variety of different attributes including *Director*, *Actor*, *Role*, etc. An E-mail has attributes like *Sender*, *Receiver*, *Subject*, *Date*, etc. All these attributes are non-numeric (with the exception of *Date*). Clustering is usually performed based on distance between objects in the data set. When the attributes are non-numeric, there is no obvious notion of distance between two objects. Instead, we measure the degree of similarity between objects and consider an effective way to calculate this similarity value.

In this dissertation, we consider a graphical representationof objects and relationships between them in a non-numeric dataset. Also, we use our graph-based similarity measure to evaluate similarity between objects. We implement our clus-

tering algorithm using different variations of the proposed similarity measure. Our graph representation scheme, similarity measure, and clustering method are generally applicable to a variety of datasets.

Our motivations for clustering research are formulated as follows:

- Use our new similarity measure with a graphical data representation for clustering.
- Enhance the K-medoid method based on postprocessing steps with new validity indices.
- Improve K-medoid method with multiple medoids in one cluster.
- Cluster movies using *Director* and other important attributes of a movie. Moreover, we wanted to check the clustering results without using *Director* attribute in the similarity measure for the interpretation of *Director* attribute.
- Develop an independent graph-based post-clustering node move method.

We propose measures to evaluate the clustering results. We present two validity indices — Cohesion-Separation or $CS$ for short and Representativeness of Medoid or $RoM$ for short — which are based on the cohesion and separation of clusters. Furthermore, we present another validity index called Concentration of Similarity ($CoS$ for short) which measures how well the nodes similar to a given node are concentrated in the same cluster.

To reduce computations involved in calculating the validity indices, we sample the nodes in each cluster and calculate the validity indices based on the sampled nodes. We compare different sampling strategies based on their sampling errors and time complexities. Inspired by our experimental results, we present two theorems relating the validity index and cluster number which are theoretical results about the influence of postprocessing on the value of validity indices. Based on our experimental results, the influence of different similarity measure variations, clustering postprocessing steps, and cluster number on the quality of clustering is discussed.

We propose an improved K-medoid clustering method in which each cluster has more than one medoid. Experimental results show that this method is effective in some situations.

We propose an independent edge-oriented node move algorithm which can be applied to a clustering result to improve the clustering quality for edge-oriented evaluation measures.

Furthermore, we improve the fuzzy K-medoid clustering method in the same way we improve K-medoid method. Finally, we developed a cluster visualization tool to visually display the clustering result in different forms.

Our initial results on clustering have been presented in Zhao and Sivakumar [2009]. See Zhao and Sivakumar [2011a] for a more comprehensive presentation, including recent results.

### *1.2.1 Related Work*

Clustering has been widely studied and used in many applications, especially since the late 1960s. Classical and recent approaches include prototype-based approaches such as K-mean, FCM, and EM; hierarchical approaches such as single-link or complete-link agglomerative clustering; graph-based approaches such as OPOSSUM; density-based approaches such as DBSCAN, CLIQUE, and DENCLUE; and scalable approach such as BIRTH and CURE (Tan et al. [2006]). Most of the classical techniques are based on distances between the data objects in the original feature space, while some techniques like OPOSSUM (Strehl and Ghosh [2000]) use similarity measures. There is no "best" clustering algorithm that is suitable for all types of data objects.

$K$-means (MacQueen [1967]) and $K$-medoid (Kaufman and Rousseeuw [1990]) are two well-known clustering algorithms which are the most prominent in prototype-based clustering techniques. $K$-means is simple and efficient, but it has trouble clustering data that contains outliers. Also $K$-means is restricted to data for which there is a notion of a center (centroid). $K$-medoid is robust to outliers, and it can be applied to a wider ranger of data since it requires only a proximity measures for a pair of objects (Tan et al. [2006]).

Chu et al. [2002] proposed an incremental multi-centroid, multi-run sampling

scheme (IMCMRS) for $k$-medoid-based clustering algorithms, to overcome the inefficiency of $k$-medoid algorithms which is one of the main limitations of $K$-medoid clustering algorithm. Their algorithm is based on an observation that there is a higher probability of better medoids being selected close to the centroid of the clusters, and based on the efficiency of the centroid-based clustering (such as $K$-means).

Strehl and Ghosh [2000] proposed a relationship-based technique for graph-partitioning-based clustering of market baskets data that tries to sidestep the "curse-of-dimensionality" issue by working in a suitable similarity space instead of the original high-dimensional feature space.

Zhou et al. [2009] proposed a graph clustering algorithm, SA-Cluster, based on both structural and attribute similarities through a unified distance measure. One main difference between their method and our method is that their structural similarity is led by neighbor nodes through edges in the original graph, while in our method it is led by common/uncommon neighbor nodes through attribute nodes. We compare out experimental results with their results on the same dataset.

Yin et al. [2006] measured similarity between two objects based on the similarities between the objects linked with them. Although our indirect similarity measure (presented in Section 3.4) has a similar flavor, we consider common neighbors and uncommon neighbors separately and for common neighbors we divide them into two categories, so that the relation between objects are more clear and comprehensive.

Jedidi [1998] analyzed weekly box office revenues for approximately 100 successful motion pictures using a finite mixture regression technique to determine regularity in sales patterns. Based on an exponential decay model applied to market share data, four clusters of movies, varying in opening strength and decay rate, were found.

Cohesion and separation are often used for unsupervised cluster evaluation (Tan et al. [2006]). For Euclidean space, the traditional measure of cohesion is the sum of the squared error (SSE), and the measure of separation is the between group sum of squares (SSB) (Tan et al. [2006]). Some postprocessing techniques can be applied to reduce the SSE (Tan et al. [2006]). Because the similarity measure (e.g., for movie) is often not an Euclidean space, we use four (two for cohesion and two for separation) related but different measures to evaluate our clustering result.

When we have externally derived class labels for the data objects, supervised measures of cluster validity could be used. Entropy, purity, precision, recall, and F-measure are supervised and classification-oriented measures of cluster validity. Statistic and Jaccard coefficient are two of the most frequently used similarity-oriented cluster validity measures.

See Halkidi et al. [2002a,b] for a comprehensive review of clustering validity and methods.

Gibson et al. [2005] presented an algorithm for finding large, dense subgraphs in massive graphs. Their algorithm was based on a recursive application of fin-

gerprinting via shingles. Wu and Kalyanaraman [2008] presented the design and implementation of a parallel approach to identify protein families from large-scale metagenomic data. Given a set of peptide sequences they reduced the problem to one of detecting arbitrarily-sized dense subgraphs from bipartite graphs. Their approach parallelized this task on a distributed memory machine through a combination of divide-and-conquer and combinatorial pattern matching heuristic techniques.

Newman and Girvan [2004] proposed and studied a set of algorithms for discovering community structure in networks — natural divisions of network nodes into densely connected subgroups. Their algorithms involved iterative removal of edges from the network to split it into communities, the edges removed being identified using one of a number of possible "betweenness" measures which are recalculated after each removal. They also proposed a measure called modularity for the strength of the community structure found by their algorithms, which gave them an objective metric for choosing the number of communities into which a network should be divided. We compare out experimental results with their results on the same dataset.

Satuluri and Parthasarathy [2009] presented a multi-level algorithm for graph clustering using flows simulation. The graph is first successively coarsened to a manageable size, and a small number of iterations of flow simulation is performed on the coarse graph. The graph is then successively refined, with flows from the previous graph used as initializations for brief flow simulations on each of the intermediate

graphs. When they reach the final refined graph, the algorithm is run to convergence and the high-flow regions are clustered together, with regions without any flow forming the natural boundaries of the clusters. We compare our experimental results with their results on the same dataset.

## 1.3 Entity Resolution

Entity resolution is the process of determining whether multiple records refer to the same real world entity. This problem is also known as deduplication, identity resolution, object identification, coreference resolution, record linkage, merge-purge, data association, etc.

Entities are described by their attributes. The values of the attributes constitute the information of some specific entity. For example, a person has attributes like name, data of birth, fingerprint, etc. Because the entity is a real world entity, so the collection of its true attribute values are unique and truly existing. We may or may not know the true attribute values of an entity. We only have the reference(s) of the entity, which is a collection of its attribute values. We may have multiple references for some specific real world entity with different collections of attributes values. Because they look different, we need to determine whether they refer to the same real world entity or not, which is what entity resolution needs to do, in the

narrow sense.

In a broader sense, entity resolution involves more work. We may only have unstructured data without explicit entity references yet. Or the entity references may not have consistent format. Talburt [2010] listed five major activities for entity resolution in a larger context:

- Entity reference extraction: Locating and collecting entity references from unstructured information.
- Entity reference preparation: The application of profiling, standardization, data cleaning, and other data quality techniques to structured entity references prior to the start of the resolution process.
- Entity reference resolution: Resolving (deciding) whether two references are to the same or different entities.
- Entity identity management: Building and maintaining a persistent record of entity identity information over time.
- Entity relationship analysis: Exploring the network of associations among different but related entities.

In our research, our work mainly involves entity reference preparation and resolution. Our data source is in text format generally with explicitly designated entity

attributes, so we do not need to do entity reference extraction, which often involves feature extraction or named entity recognition work. We did some entity reference preparation work to improve the data quality to make them more comparable.

In its simplest form, the techniques for performing entity resolution need to establish good match criteria for any pair of records, where entity resolution can be viewed as a classification problem: given a vector of similarity values between the attributes of two records, classify it as "match (duplication)" or "non-match (non-duplication)." This match criteria may involve a metric and a user-defined threshold, which determines the point dividing matching records from non-matching ones. Classically, as defined by Fellegi and Sunter [1969] and reported by Winkler [2003], this criteria returns one of three responses: matches, does not match, or needs more review. Brizan and Tansel [2006] provide a brief description of various matching techniques reported in the literature. Elmagarmid et al. [2007] presented a comprehensive survey of the existing techniques used for detecting nonidentical equivalent entries in database records. Winkler also gave an overview of this problem in Winkler [2006].

Entity resolution systems generally use four basic techniques to determine whether two references are equivalent. They are direct matching, transitive equivalence, association analysis, and asserted equivalence (Talburt [2010]). We use all four techniques in our entity resolution framework.

Traditional entity resolution methods often exploit textual similarity between data attributes. Some more recent methods consider relational similarities derived from the context of data as additional information. There are some other methods that are based on probabilistic models. Methods that are used for matching records with multiple fields can be broadly divided into two categories (Elmagarmid et al. [2007]): one is approaches that rely on training data to "learn" how to match the records. This category includes some probabilistic approaches and supervised machine learning techniques; the other is approaches that rely on domain knowledge or on generic distance metrics to match records. This category includes approaches that use declarative languages for matching and approaches that devise distance metrics appropriate for the equivalent detection task.

In our research, we developed a method that combined the useful aspects of both categories. Also, current methods seldom deal with or describe details about inconsistency elimination for the final classification result. Furthermore, current methods often lack modularity or are hard to be adapted. For these reasons, we present a simple but effective method, which is based on graph-based similarity measures and a simple probabilistic model. We focus on both the identification part and the merging part of the entity resolution problem. We also propose a filtering strategy (pre-processing step) and several inconsistency elimination methods and two effective record updating algorithms (post-processing steps) to improve the computational efficiency and

classification results. Our record updating algorithms have two other functions: one is to find the records with "true" attributes (exemplar or canonical records); the other is to find a good matching threshold used for unsupervised learning when determining whether a record pair is equivalent or not. Sometimes we use the word "duplicate" for "equivalent" in the context of entity resolution, although equivalent might be a better term.

The ideal entity resolution result is that the matching relationship we found among the data records is exactly the same to the true equivalence relationship, but in practice, for any non-trivial entity resolution work, it is not realistic. Talburt [2010] illustrated the relation between these two relationships. Let $S$ denote the Cartesian product of the dataset itself, which is the set of all ordered pairs of references in the dataset. Let $M$ stand for the set of matching pairs of references, and $E$ stand for the the set of true equivalent pairs of references. Then $M \bigcap E$ is the set of true positive results, $M - E$ is the set of false positive results, $E - M$ is the set of false negative results, and $S - (M \bigcup U)$ is the set of true negative results. Our aim is to make $M - E$ and $E - M$ as small as possible.

Our method is evaluated using F1 (most commonly used form of F-measure (Rijsbergen [1979])) and "area under precision-recall curve" (AUC). Experimental results show our method can produce high quality entity resolution result for both unsupervised and supervised learning. The proposed method encompasses the entire

entity resolution process (in the narrow sense) and includes some steps that are quite general and applicable to other entity resolution methods based on pairwise record matching.

Our experiments thus far have been conducted on a single set of records; i.e., we have not yet considered entity resolution on multiple data sources. However, our current method can be easily adapted for the multiple data sources scenario.

For large datasets and/or computationally intensive similarity measures, the running time of an entity resolution process could be an issue. To solve this problem, blocking or canopy (Elmagarmid et al. [2007]) methods are generally used to efficiently select a subset of record pairs for subsequent similarity computation while other "dissimilar" pairs are simply ignored. In our research, we present a filtering method on the test datasets which has similar effect of canopy.

Our method differs from the classic Fellegi-Sunter Model (FSM) mainly in two ways. First, we use different probabilistic matching method. FSM focuses on attribute value pattern, while our method uses similarity value pattern. Second, FSM only uses direct matching, while our method uses direct matching, transitive equivalence, and asserted equivalence. Moreover, our indirect similarity measure is context-based, which gives the benefit of association analysis, although we do not make collective matching decision directly.

The main contributions of our work on entity resolution include:

- We propose a general framework to address the entity resolution problem.
- We present a simple probabilistic classification model for supervised learning, in which similarity pattern is used instead of attribute value pattern. Methods to estimate the probabilities required by the classifier are also given.
- We propose two record updating algorithms to: 1) improve classification result; 2) find records with "true" attributes; 3) find good matching threshold used for unsupervised learning.
- We present several inconsistency elimination methods to eliminate the inconsistencies in the matching result as well as improve the classification accuracy.
- We compare the performance of our method with that of several recent entity resolution methods (Singla and Domingos [2006], Wick et al. [2009] and S. Rendle [2006]) using F1 and AUC. For the same dataset, our method generally outperforms others.

Our initial results on entity resolution have been presented in Zhao and Sivakumar [2010]. See Zhao and Sivakumar [2011b] for a more comprehensive presentation, including recent results.

### *1.3.1 Related Work*

Minton and Nanjo [2005] described the relationship between two field values by a set of heterogeneous transformations. They estimated the probability that some pair is a match given the types of transformations and used it as a measure of the "distance" between two attribute values. Some of our ideas were inspired by this work.

Chen et al. [2007] presented a graphical approach for entity resolution. It used the analysis of the entity-relationship graph constructed for the dataset being analyzed. They measured the degree of interconnectedness between various pairs of nodes in the graph.

Dong et al. [2005] considered complex information spaces for reference reconciliation problem. Their references belong to multiple related classes and each reference may have very few attribute values. Their algorithm exploited context information, and once they decide to merge two references, they have two mechanisms for leveraging this information: reconciliation propagation and reference enrichment. Their exploration of context information and reconciliation propagation are a bit similar to our indirect similarity concept, and the idea of their reference enrichment is a little similar to our record updating algorithm. However, we exploit context information in a different way and update the record (instead of joining), with some restriction.

Whang et al. [2009] proposed an iterative blocking framework where the entity resolution results of blocks are reflected to subsequently processed blocks. Blocks are iteratively processed until no block contains any more matching records. In our method, we do re-filtering after record updating but it is different from their iterative blocking idea.

Singla and Domingos [2006] proposed an integrated solution to the entity resolution problem based on Markov logic. Markov logic combines first-order logic and probabilistic graphical models by attaching weights to first-order formulas, and viewing them as templates for features of Markov networks. They showed how a small number of axioms in Markov logic capture the essential features of many different approaches to this problem, in particular non-i.i.d. (independent and identically distributed) ones, as well as the original Fellegi-Sunter model. We compared out experimental results with their results on the same dataset.

Bhattacharya and Getoor [2006] extended the Latent Dirichlet Allocation model for unsupervised entity resolution and proposed a probabilistic model for collective entity resolution for relational domains where references are connected to each other. They did not introduce a decision variable for each potential equivalent pair of references, but instead had an entity label for each reference. To model collaborative relations between entities, they introduced a group label for each reference, so that entities coming from the same collaborative group are more likely to be observed in a

relation. They also proposed an unsupervised Gibbs sampling algorithm for collective entity resolution.

Bhattacharya and Getoor [2007] proposed a relational clustering algorithm that used both attribute and relational information for determining the underlying domain entities. They investigated the impact that different relational similarity measures have on entity resolution quality.

S. Rendle [2006] proposed a model that uses structural information given as pairwise constraints to guide collective decisions about object identification in addition to a learned similarity measure. We compared out experimental results with their results on the same dataset.

Bohannon et al. [2005] introduced a cost framework for the constraint repair problem that allows for the application of techniques from record-linkage to the search for good repairs. They proved that finding minimal-cost repairs in this model is NP-complete in the size of the database, and introduced an approach to heuristic repair-construction based on equivalence classes of attribute values. Following this approach, they defined two greedy algorithms.

Bohannon et al. [2007] proposed a class of constraints, referred to as conditional functional dependencies (CFDs), and studied their applications in data cleaning. In contrast to traditional functional dependencies (FDs) that were developed mainly for schema design, CFDs aim at capturing the consistency of data by incorporating

bindings of semantically related values. They developed techniques for detecting CFD violations in SQL as well as techniques for checking multiple constraints in a single query.

Fuxman et al. [2005] presented ConQuer, a system for efficient and scalable answering of SQL queries on databases that may violate a set of constraints. ConQuer permitted users to postulate a set of key constraints together with their queries. The system rewrote the queries to retrieve all (and only) data that is consistent with respect to the constraints.

Cong et al. [2007] studied effective methods for improving both data consistency and accuracy. They employed a class of conditional functional dependencies (CFDs) proposed in Bohannon et al. [2007] to specify the consistency of the data, which are able to capture inconsistencies and errors beyond what their traditional counterparts can catch. To improve the consistency of the data, they proposed two algorithms: one for automatically computing a repair D' that satisfies a given set of CFDs, and the other for incrementally finding a repair in response to updates to a clean database.

Chaudhuri et al. [2007] observed that there are scenarios where additional constraints on the data are available that can be used to evaluate the quality of deduplication. They formalized the aggregate constraints and illustrated it through various examples. They integrated the use of constraints in deduplication by using the textual similarity between tuples to restrict the search space of partitions.

Christen [2008] described the Febrl (Freely Extensible Biomedical Record Linkage) system, which is available under an open source software licence. It contains many recently developed advanced techniques for data cleaning and standardization, indexing (blocking), field comparison, and record pair classification, and encapsulates them into a graphical user interface. Bilgic et al. [2005] introduced D-Dupe which is an interactive tool that combines data mining algorithms for entity resolution with a task-specific network visualization which displays the collaboration context for potential equivalents. Raman and Hellerstein [2001] presented Potter's Wheel, an interactive data cleaning system that tightly integrates transformation and discrepancy detection. Koudas et al. [2005] presented a prototype system called SPIDER, which supports flexible string attribute value matching in large databases. Elfeky et al. [2002] developed an interactive record linkage toolbox named TAILOR. Users of TAILOR can build their own record linkage models by tuning system parameters and by plugging in in-house developed and public domain tools.

Arasu et al. [2009] did collective deduplication of entity references in the presence of constraints. Their framework was based on a simple declarative Datalog-style language with precise semantics. Galhardas et al. [2001] presented a language, and execution model and algorithms that enable users to express data cleaning efficiently.

In relational databases, accurate deduplication for records of one type is often dependent on the decisions made for records of other types. Culotta and McCallum

Culotta and McCallum [2005] modeled the inter-dependencies explicitly to collectively deduplicate records of multiple types. In our method, we use a simple but effective probabilistic model. Moreover, although we do record matching pairwise, our record updating and equivalent elimination are executed on records collectively.

Wick et al. [2009] proposed a discriminatively-trained model that jointly performs co-reference resolution and canonicalization (the process of generating a standardized representation of the referent entity), enabling features over hypothesized entities. We compared out experimental results with their results on the same dataset.

Chaudhuri et al. [2005] observed that the distance thresholds for detecting real equivalent entries are different for each database tuple. They proposed an efficient algorithm for computing the required threshold for each object in the database. In our method, we use record updating to efficiently and effectively find a good matching threshold for unsupervised learning.

Sarawagi and Bhamidipaty [2002] presented a method which provides a covering and challenging set of training pairs that bring out the subtlety of the deduplication function. Our supervised learning method also does "active" learning in the sense that some newly labeled pairs of records are sometimes added into training set after postprocessing (see algorithm 8 in section 5.5).

Swoosh (Benjelloun et al. [2008]) developed by the Stanford SERF project is a generic approach to entity resolution. They identified four properties which they

thought enable much more efficient entity resolution algorithms if they are satisfied by the match and merge functions. They developed different algorithms for different situations about these properties. They did not study the internal details of the functions used to compare and merge records. Rather, they view these functions as "black-boxes" to be invoked by the entity resolution engine. In our method, we take care of the details of comparing and merging of records, as well as focus on modularizing the whole entity resolution process to make it general and adaptable. In SERF, the transitive equivalence is made by merging the attribute values of equivalent references, while in our method, we explicitly eliminate inconsistency first, then update attribute values of some records. In particular, we resolve the inconsistencies with edges in both training and test set.

For more comprehensive surveys of entity resolution see Elmagarmid et al. [2007] and Winkler [2006].

The correlation clustering problem was introduced by Bansal et al. [2002], which shows a constant factor approximation algorithm for minimizing disagreements, based on the principle of counting erroneous triangles (triangles with two positive labeled edges and one negative labeled edge). Ailon et al. [2005] proposed a randomized 3-approximation algorithm for this problem. In our method, we implement the ideas of these two correlation clustering algorithms for inconsistency elimination and compare the results with those of our methods.

# CHAPTER 2. DEFINITIONS

This chapter gives definition of the terms used in this dissertation. Some of them come from Tan et al. [2006] and the Glossary part of Talburt [2010].

Attribute: A characteristic associated with an entity that can take on a defined set of values.

Attribute Node: Each attribute value is represented as one attribute node in our graph representation of data.

Blocking: A technique for match prospecting (finding the references that are most likely to match a given reference when there are many references to choose from) based on selecting all (a block of) records that share a certain attribute value.

Canopy: An clustering algorithm intended to speed up clustering operations on large data sets. The algorithm uses a computationally simple, approximate distance measure to efficiently divide the data into overlapping subsets (called "canopies"). Then clustering is performed by measuring exact distances only between points that occur in a common canopy (McCallum et al. [2000]).

Clustering: The process of cluster analysis or an entire collection of clusters.

Clusters: Useful groups of data objects.

Cluster Cohesion: An cluster validity index which is used to determine how

closely related the objects in a cluster are, also called cluster compactness, tightness.

Cluster Separation: An cluster validity index which is used to determine how distinct or well-separated a cluster is from other clusters, also called cluster isolation.

Data Mining: A collection of methods and techniques for finding implicit relationships in a collection of data.

Data Node: Each entity reference is represented as one data node in our graph representation of data.

Data Quality: The degree of fitness for use of data in particular application.

Edge: In our graph representation of data, edge is used to connect data node and attribute node. Edge includes directed edge (also called link, e.g. web page links) and undirected edge (e.g. edge between age of person and the person).

Entity: A real world person, place, or other object that has a unique identity that distinguishes it from all other entities of the same type.

Entity Reference: A collection of identity attribute values that describe a particular entity.

Entity Resolution: A body of knowledge and practice related to the activities supporting a process to decide whether two entity references are equivalent or not.

Equivalent References/Records: Two entity references/records are said to be equivalent if, and only if, they refer to the same real world entity.

F-measure: A measure of a test's accuracy which considers both the precision

and the recall of the test to compute the score.

F1: Most commonly used F-measure, which equals 2 times precision then times recall divided by the sum of precision and recall.

False Negative: A term used to describe the situation where a decision process provides a negative answer when it should have provided a positive answer.

False Positive: A term used to describe the situation where a decision process provides a positive answer when it should have provided a negative answer.

Fellegi-Sunter Record Linkage Model: A model for determining a set of agreement patterns for a direct matching entity resolution process that will keep the false positive and false negative rates for automated equivalence decisions within predefined limits, and at the same time, minimize the number of equivalence decisions that must be made by inspection.

Filtering: A step used in our entity resolution framework which takes as input the record pairs and outputs candidate record pairs which will be processed in the subsequent record matching step. It efficiently selects a subset of record pairs for subsequent similarity computation.

Inconsistency: Inconsistent decision in entity resolution.

Link: Direct edge in our graph representation of data.

Matching: An operation between two values that gives a true or false result where true indicates that the values are identical or closely related according to some

rule or algorithm.

Medoid: The most representative data object for a group of data objects.

Merge-purge: An entity resolution architecture in which direct matching and transitive equivalence are successively used to resolve the references into clusters of equivalent references.

Node: In our graph representation of data, node is used to represent entity reference or attribute values.

Precision: The fraction of retrieved instances that are relevant, which equals true positive divided by the sum of true positive and false positive.

Recall: The fraction of relevant instances that are retrieved, which equals true positive divided by the sum of true positive and false negative.

Record: An entity reference.

Record Linkage/Linking: A term originally used to designate a specific problem of determining direct matching equivalences between two lists of references assumed to have no internal equivalences, it is now commonly used to describe any entity resolution process based on the merge-purge architecture.

Record Update: A step used in our entity resolution framework which involves updating of some entities' attribute values.

Semi-Structured Data: Structured data that does not conform with the formal structure of tables and data models associated with relational databases but

nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data (Buneman [1997]).

Similarity Measure: A numerical measure of the degree to which two data objects are alike.

Similarity Metric: A similarity measure which could be converted to a distance metric which satisfies positivity, symmetry, and triangle inequality.

Structured Data: Data that is organized in such a way that all of the attribute values describing a particular entity are presented in a consistent and predictable pattern that can be programmed into a computer, e.g. database tables.

Transitive Closure: A method used in our entity resolution framework making use of transitive relations to eliminate inconsistent decisions.

True Negative: A term used to describe the situation where a decision process provides a correct negative answer.

True Positive: A term used to describe the situation where a decision process provides a correct positive answer.

Unstructured Data: Data that is not structured, e.g. free-form text, images, audio streams.

# CHAPTER 3. SIMILARITY MEASURE

Similarity measure is an assessment of "closeness" between two records and plays an important role in our clustering and entity resolution work. Different similarity measures can lead to very different content and quality of the data mining result. We propose graph-based similarity measures in which both direct similarity led by simple attributes and indirect similarity led by data records are considered.

## 3.1 Data representation

Data that requires mining often consists of non-numeric attributes (e.g., movie data, Email data, news articles). Naturally, we use a graph to represent these data objects and their relationships. Each value of each attribute is represented by a node. An edge connects two nodes which have a direct relationship. For example, in a database of movies, each movie record has attributes like title, name (for people in the movie such as actor), role, etc. Thus we have a title node, name nodes, role nodes, etc. Names and roles in one movie are connected to each other and to that movie title node.

Unfortunately, this kind of graph representation can lead to some problems. Consider the following scenario: Two movies $m_1$ and $m_2$ have a same role called $r_1$,

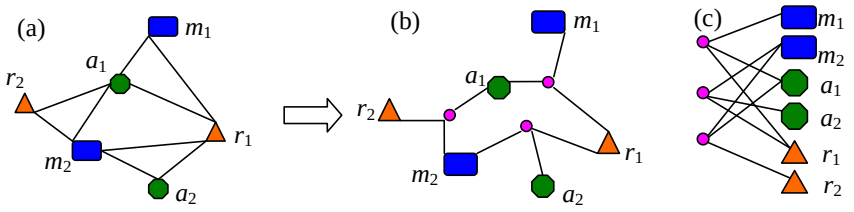


**Figure 3.1:** Data representations

an actor $a_1$ played role $r_1$ in $m_1$ and role $r_2$ in $m_2$, and another actor $a_2$ played role $r_1$ in $m_2$; then the graph depicting this scenario would be as shown in Figure 3.1 (a). In Figure 3.1 (a), ambiguity is introduced as it seems to imply that $a_1$ also played (role) $r_1$ in (movie) $m_2$. This problem could be solved in different ways, e.g., introducing duplicate nodes. We use The RElationship Generating Graph Analysis Engine (REGGAE) (Larson [2008]) developed by Applied Technical Systems (ATS) to solve this problem by storing context information without introducing duplicate nodes for the same data element. REGGAE is a bipartite graph structure consisting of a data layer and a context layer. Data layer contains only data (e.g., "Name: $a$", "Role: $r_1$", "Title: $m_1$") and the context layer stores connections between data nodes. Nodes in the context layer may only connect to nodes in the data layer, and nodes in the data layer may only connect to nodes in the context layer. The formal definitions of data layer and context layer are as follows (Larson [2008]):

*Data Layer*: Populated with entities (cells), with each entity represented by a basic

Type-Value construct.

*Context Layer*: Populated with context nodes (links), which provide contextual relationships between entities on the graph.

Figure 3.1 (b) is the REGGAE bipartite graph depicting the previous scenario. The small circles (colored purple) are nodes in the context layer and the rest (colored other than purple) are nodes in the data layer. A node in the data layer is called an "entity node" and a node in the context layer is called a "link node." We can see that the ambiguity problem is solved through the introduction of link nodes. Figure 3.1 (c) is a bipartite isomorphic version of figure 3.1 (b).

For entity nodes of REGGAE graph, besides the nodes for attributes' values, we should also include one other kind of node called *Data node* to represent each data object consisting of attributes. (Note that this *Data node* is different with the data node we mentioned in previous page.) The attribute value's node will be called *attribute node*, which takes the form of "AttributeName: AttributeValue", e.g., "Role: Super man". For Data nodes, we can use the unique ID of this data object in the dataset as its value, such as "Movie: MovieID" or "Email: MsgID". Note that if the attribute's value is a set, we should use one node for each value in the set. An edge between an attribute node and a data (record) node (through link node) indicates that the record has the said attribute.

Given data and their representation, one important thing is to investigate the

similarity between data entities. Data similarity is used by many data mining techniques and tasks, such as clustering and entity resolution. Similarity between data objects involve similarity between simple attributes and similarity between data records. We now present the similarity measures, which are based on a graph structure, used in our data mining research.

## 3.2 Direct similarity metric

Two (or more) data nodes $a$ and $b$ could share some common attribute node(s). In Figure 3.2, data nodes (square nodes) $x$, $y$, and $cc$ share a common attribute node (oval node). For simplicity, link nodes are omitted. $X$ denotes data node $x$ and its context including attribute nodes and neighbor data nodes (similarly for $Y$). A node's neighbor data node is the data node which shares attribute node(s) with it. We name similarity led by common attribute nodes as *Direct similarity*. We have the following similarity measure for direct similarity between $a$ and $b$:

$$S_d = \sum_{i=1}^{m} w_i S_{d_i}, \tag{3.1}$$

where $S_{d_i}$ is similarity value associated with $i$th attribute of $a$ and $b$, and $w_i$ is the corresponding weight for $i$th attribute. Typically, $0 \leq S_{d_i} \leq 1$, $w_i \geq 0$ and $\sum_{i=1}^{m} w_i = 1$, $m$ being the number of attributes considered. Calculation of $S_{d_i}$ depends on the data type of the attribute. It compares the similarity between attribute $i$'s

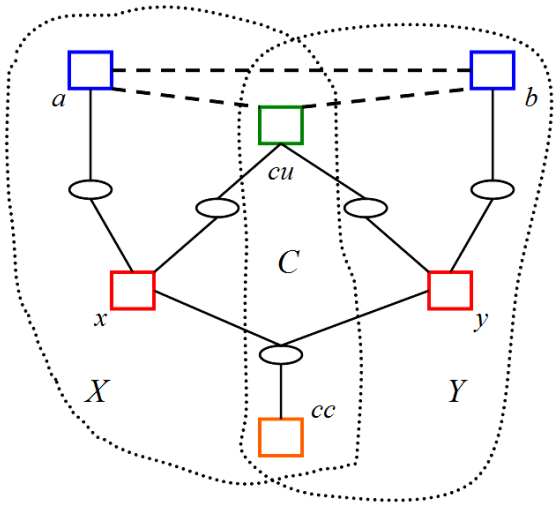


**Figure 3.2:** Graph explanation of similarity measures

values for two data objects. In our research, we use Jaccard coefficient (Tan et al. [2006]) to calculate $S_{d_i}$:

$$S_{d_i} = \frac{|a_i \bigcap b_i|}{|a_i| + |b_i| - |a_i \bigcap b_i|}, \tag{3.2}$$

where $a_i$ and $b_i$ is the $i$th attribute of $a$ and $b$. For the attribute whose value is a set, $S_{d_i}$ is between 0 and 1; for the attribute whose value is a single string (we can treat it as a set with only one item for equation (3.2)), $S_{d_i}$ is either 0 or 1.

In the following, we shall show that our direct similarity measure defined in Equations (3.1) and (3.2) can be easily converted to a distance metric. We have $S_d = \sum_{i=1}^{m} w_i S_{d_i}$, where $i$ is $i$th attribute, so it suffices to show that each $S_{d_i}$ can be converted to a metric. For simplicity, we use $s$ to denote $S_{d_i}$. We will show that

$d = 1 - s$ satisfies the properties of *Positivity*, *Symmetry*, and *Triangle Inequality* and hence is a metric. In the following, we use $x$ to denote both attribute $x$ and its value set.

**Positivity**

(a) $d(x, y) \geq 0$ for all $x$ and $y$.

*Proof.* We know $|x \bigcap y| \leq |x|$ and $|x \bigcap y| \leq |y|$.

$d(x, y) = 1 - s(x, y) = 1 - \frac{|x \bigcap y|}{(|x|+|y|-|x \bigcap y|)} = \frac{|x|+|y|-2|x \bigcap y|}{|x|+|y|-|x \bigcap y|)} \Rightarrow d(x, y) \geq 0.$

(b) $d(x, y) = 0$ if and only if $x = y$.

*Proof.* i. $x = y \Rightarrow |x \bigcap y| = |x| = |y| \Rightarrow d(x, y) = \frac{|x|+|y|-2|x \bigcap y|}{|x|+|y|-|x \bigcap y|)} = 0$;

ii. $d(x, y) = 0 \Rightarrow |x| + |y| - 2|x \bigcap y| = 0 \Rightarrow |x| = |x \bigcap y| = |y| \Rightarrow x = y.$

**Symmetry**: $d(x, y) = d(y, x)$ for all $x$ and $y$.

*Proof.* $d(x, y) = 1 - \frac{|x \bigcap y|}{(|x|+|y|-|x \bigcap y|)} = 1 - \frac{|y \bigcap x|}{(|y|+|x|-|y \bigcap x|)} = d(y, x).$

**Triangle Inequality**: $d(y, z) \leq d(x, y) + d(x, z)$ for all $x$, $y$, and $z$.

*Proof.* Figure 3.3 illustrates sets $x$, $y$, and $z$, as well as the other variables used in our proof. We then have:

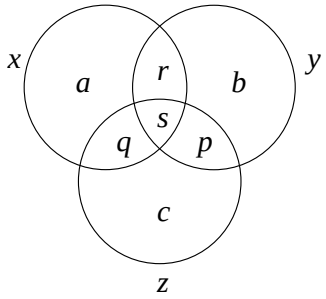


**Figure 3.3:** Sets $x$, $y$, and $z$

$$s(x,y) = \frac{r+s}{a+q+b+p+r+s} \leq \frac{p+r+s}{b+p+r+s+q}, \tag{3.3}$$

$$s(x,z) = \frac{q+s}{a+r+c+p+q+s} \leq \frac{p+q+s}{c+q+s+p+r}, \tag{3.4}$$

$$\text{and } s(y,z) = \frac{p+s}{b+r+c+q+p+s}. \tag{3.5}$$

Note that $d(y,z) \leq d(x,y) + d(x,z) \Leftrightarrow 1 - s(y,z) \leq 1 - s(x,y) + 1 - s(x,z)$, so it suffices to show that

$$s(y,z) - s(x,y) - s(x,z) + 1 \geq 0. \tag{3.6}$$

From equations (3.3)-(3.5) we get

$$\begin{aligned} s(y,z) - s(x,y) - s(x,z) + 1 &\geq \frac{p+s}{b+r+c+q+p+s} \\ &- \frac{p+r+s}{b+p+r+s+q} - \frac{p+q+s}{c+q+s+p+r} + 1 = \frac{A}{B}, \end{aligned}$$

where

$$B = (b+c+p+q+r+s)(c+p+q+r+s)(b+p+q+r+s) \geq 0,$$

and

$$\begin{aligned} A &= (p+s)(c+p+q+r+s)(b+p+q+r+s) \\ &\quad -(p+q+s)(b+c+p+q+r+s)(b+p+q+r+s) \\ &\quad -(p+r+s)(b+c+p+q+r+s)(c+p+q+r+s) \\ &\quad +B \\ &= b^2(c+r)+c^2(b+q)+(2bc+br+cq)(p+q+r+s) \\ &\geq 0. \end{aligned}$$

This shows (3.6), which completes the proof.

## 3.3 Domain-optimized direct similarity calculation

As described in section 3.2, we do field (attribute) matching for direct similarity between two records. The direct similarity metric used in our entity resolution framework is able to produce good results. However, for a particular dataset, some domain-oriented optimizations can be made to get better field matching performance and hence better entity resolution results.

Our experiments were conducted on two datasets: Cora and CDDB (Weis et al. [2009]). Details about the datasets as well as the specific domain-oriented optimizations are described in chapter 7. Our domain-optimized direct similarity calculation can get better similarity values which leads to better filtering (see section 5.3.1) and

consequently a better entity resolution result (see Tables 7.16-7.19 and Tables 7.23).

## 3.4 Indirect similarity measure

Two data nodes $x$ and $y$ could share some common neighbor data node(s). Nodes $x$ and $y$ are called neighbors if they share some attribute node(s). Two nodes, that do not share any common attributes (non-neighbors) can still be somewhat similar if they:

- Share some common neighbor nodes. In other words, nodes $x$ and $y$ may not share any common attributes but both $x$ and $y$ may have common attributes with a third node $z$. We consider only those common attributes between $x$ and $z$ that are not common between $x$ and $y$ (since the direct similarity has already accounted for them) and similarly consider only those common attributes between $y$ and $z$ that are not common between $y$ and $x$. In short, we refer to this as two data nodes $x$ and $y$ having a common neighbor data node(s) $z$, through *different* attribute node(s).

- Have uncommon neighbor nodes. This is based on the neighbor nodes of $x$ exclusive or $y$. In short, we refer to this as two data nodes $x$ and $y$ having *uncommon neighbor* data node(s).

We call similarity led by common or uncommon neighbor data node as *Indirect similarity.*

Indirect similarity is illustrated in Figure 3.2. Here, $cc$ is a common neighbor data node of $x$ and $y$ through the same attribute node (we will not count this in the calculation of indirect similarity between $x$ and $y$); $cu$ is a common neighbor data node of $x$ and $y$ through different attribute nodes — it will be considered in the calculation of indirect similarity between $x$ and $y$; $a$ and $b$ are uncommon neighbor data node of $x$ and $y$, respectively. Note that there are no direct edges connecting two record (square) nodes or between two attribute (oval) nodes even after we omit the link nodes.

We propose three different methods to calculate the indirect similarity between $x$ and $y$:

- General: consider weighted similarities on paths $(x, cu, y)$ and $(x, a, b, y)$ for all $cu$, $a$, and $b$;
- Relaxed: consider weighted similarities on paths $(x, cu, y)$, $(x, a, b, y)$, $(x, a, cu, y)$, and $(x, cu, b, y)$ for all $cu$, $a$, and $b$;
- Simple general and simple relaxed: unweighted; i.e., set all weights to 1.

Now let us see how to calculate the indirect similarity using the general version. Suppose record $x$ has $m$ neighbor records (call set $A$) and record $y$ has $n$ neighbor

records (call set $B$) with $k$ common neighbor records through different properties (set $CU = A \cap B \setminus CC$ (set difference), where $CC$ is the set of common neighbor records, through same properties and $|CC| = h$); then the indirect similarity between records $x$ and $y$ is formulated in equation (3.7), where $A' = A \setminus C$, $B' = B \setminus C$, and $w_{xCU_j}$ is the weight for neighbor $CU_j$ of $x$, and so on.

$$S_i = \frac{\sum_{j=1}^{k} w_{xCU_j} w_{yCU_j} S_{xCU_j} S_{yCU_j} + \sum_{j=1}^{m-k-h} \sum_{l=1}^{n-k-h} w_{xA'_j} w_{A'_j B'_l} w_{yB'_l} S_{xA'_j} S_{A'_j B'_l} S_{yB'_l}}{\sum_{j=1}^{k} w_{xCU_j} w_{yCU_j} + \sum_{j=1}^{m-k-h} \sum_{l=1}^{n-k-h} w_{xA'_j} w_{A'_j B'_l} w_{yB'_l}} \tag{3.7}$$

We define weight $w_{xy}$ as the number of common attributes of $x$ and $y$ divided by the size of the union set of attribute of $x$ and $y$. Note that the similarities $S$ in the right-hand side of equation (3.7) are the direct similarities between the appropriate nodes.

For simplicity, we can also set all the weights in equation (3.7) to one so that we have equation (3.8) which is the simple version to calculate $S_i$:

$$S_i = \frac{\sum_{j=1}^{k} S_{xCU_j} S_{yCU_j} + \sum_{j=1}^{m-k-h} \sum_{l=1}^{n-k-h} S_{xA'_j} S_{A'_j B'_l} S_{yB'_l}}{k + (m-k-h)(n-k-h)}. \tag{3.8}$$

Note that in equations (3.7) and (3.8), if $m = k + h$ exclusive or $n = k + h$, then $|A'| = 0$ or $|B'| = 0$, which removes part of indirect similarity through non-common neighbor records. So we present a relaxed definition of $S_i$. Equation (3.9) is the relaxed version for equation (3.8) (similarly for equation (3.7)), where we set

$p = k + h$.

$$S_i = \frac{\sum_{j=1}^{k} S_{xCU_j} S_{yCU_j} + \sum_{j=1}^{m-p} \sum_{l=1}^{n-p} S_{xA'_j} S_{A'_j B'_l} S_{yB'_l} + \sum_{j=1}^{m-p} \sum_{l=1}^{k} S_{xA'_j} S_{A'_j CU_l} S_{yCU_l} + \sum_{j=1}^{k} \sum_{l=1}^{n-p} S_{xCU_j} S_{CU_j B'_l} S_{yB'_l}}{k + (m-p)(n-p) + (m-p)k + k(n-p)} \tag{3.9}$$

Note that in equations (3.7), (3.8), and (3.9), we exclude $x$ from $B'$ and $y$ from $A'$. So for the situation where $x$ and $y$ do not have a common record neighbor and $x$ and $y$ is the only neighbor for each other, both the numerator and the denominator will be zero. In this case, we simply set $S_i = 1$.

## 3.5 Modified Version for Clustering

Compared with entity resolution which needs to find equivalent records, clustering only needs to group similar records together, so we only use a simplified and modified version of our similarity measure for clustering which is expressed in equation (3.10):

$$S = w_d S_d + w_{si} S_{si}, \tag{3.10}$$

where $S_d$ is direct similarity, $S_{si}$ is simple indirect similarity where we only consider common neighbor data nodes through different attribute nodes, and $w_d$, $w_{si}$ are their corresponding (non-negative) weights with $w_d + w_{si} = 1$. Typically $0 \leq S \leq 1$.

Combine equation (3.1) and (3.10), we have:

$$S = w_d(\sum_{i=1}^{m} w_i S_{d_i}) + w_{si} S_{si}. \tag{3.11}$$

Equation (3.11) means that we can consider the set of neighbor data nodes as an attribute for the similarity calculation. The only difference for this attribute is that the common neighbor data nodes for two data nodes is through different attribute nodes, rather than simply taking the intersection. For this reason, if all the attributes are single-value attribute, then $S_{si} = 0$, which means we only need to consider direct similarity.

So far we only consider nodes' own attributes for direct similarity, while in the real life, we have many dataset in which one node have edge(s) to other nodes. For this situation, we have two options to modify our similarity measure: One is simply letting the two nodes' similarity equal to 1 or some value if they have a edge between them; the other option is for each data node treating the set of nodes which has edge to this node as an attribute. In our research, we choose the second option and call this set of neighbor nodes *direct neighbors*. For example, if two students both have their web home page link to IEEE's home page, then we prefer to consider similarity between these two students, rather than consider it between a student and IEEE. Finally, we have our similarity measure for clustering as follows:

$$S = w_d(\sum_{i=1}^{m} w_i S_{d_i}) + w_{on} S_{on} + w_{si} S_{si}, \tag{3.12}$$

where "on" represents for original neighbors led by edge between nodes, and $w_d + w_{on} + w_{si} = 1$.

As defined in chapter 2, edge could be undirected or directed (i.e., link). For

each kind of edge in the dataset, we treat the neighbor data nodes led by it as one attribute. While for link, we treat in-neighbors and out-neighbors as two separate attributes. A node $x$ 's in-neighbors are the data nodes which has link to $x$. A node $x$'s out-neighbors are the nodes which has link from $x$.

# CHAPTER 4. CLUSTERING

$K$-medoid is a simple prototype-based clustering algorithm that uses the medoid (the most representative one) of the objects in a cluster as the prototype of the cluster (Kaufmann and Rousseeuw [1990]). We use $K$-medoid because it requires only a proximity measure for a pair of objects, which we already have through the similarity measure. Also, $K$-medoid tends to produce globular clusters in which each object is sufficiently similar to the cluster's medoid or to other objects in the cluster. We enhance the basic $K$-medoid algorithm in several respects which will be discussed below. We introduce a set of validity indices to direct the clustering process, and evaluate and improve the clustering quality.

We also propose a modified $K$-medoid clustering method called multi-medoid $K$-medoid clustering, in which each cluster has more than one medoid. It could improve the quality of clustering result, at the price of worse time efficiency.

For the dataset with original edge, we designed a node move algorithm to effectively improve the quality of clustering result for edge-oriented evaluation measures.

We also enhance the traditional fuzzy $K$-medoid method and propose the corresponding validity indices.

## 4.1 $K$-medoid Clustering

### *4.1.1 Validity Indices*

Cluster evaluation (validation) is an important and necessary step for any clustering algorithm. Cohesion and separation are often used for unsupervised cluster evaluation (Tan et al. [2006]). Cohesion indicates how closely nodes in one cluster are grouped together; separation indicates how well separated clusters are between each other. We propose three different validity indices and two of them are based on the idea of cohesion and separation. Each index measures a different aspect of clustering quality. Some notations used in our validity indices are formalized in Table 4.1.

**Table 4.1:** Notations used in validity indices

| | |
|---|---|
| $C_i$ | $i$th cluster |
| $\|C_i\|$ | Size of the $i$th cluster |
| $c_i$ | medoid of $i$th cluster |
| $K$ | Number of clusters |
| $s(x_i, x_j)$ | Similarity between nodes $x_i$ and $x_j$ |

Before we introduce our validity indices let us look at four definitions first. $Cohesion_{medoid}$ or $C_m$ for short is the normalized nonmedoid-to-medoid similarity

within all the clusters:

$$C_m = \frac{\sum_{i=1}^{K}(\sum_{j=1}^{|C_i|} s(c_i, x_j) - 1)}{\sum_{i=1}^{K}(|C_i| - 1)}. \tag{4.1}$$

Here $s(c_i, x_j)$ is the similarity between medoid $c_i$ and node $x_j$. It tells us how similar are nodes to their cluster medoid; it could be regarded as an indication of the average "size" of each cluster in similarity space (analogous to "radius" in Euclidean space). In general, for a fixed dataset, we want to maximize the value of $C_m$. Note that we subtract 1 from the numerator and denominator of equation (4.1) for removing the similarity between medoid and itself for each cluster (according to our similarity measure, a node's similarity to itself is 1).

$Cohesion_{average}$ or $C_a$ for short is the normalized node-to-node similarity within all the clusters:

$$C_a = \frac{\sum_{i=1}^{K}(\sum_{j=1}^{|C_i|}\sum_{l=j+1}^{|C_i|} s(x_j, x_l))}{0.5\sum_{i=1}^{K}(|C_i|(|C_i| - 1))}. \tag{4.2}$$

It indicates how "dense" the clusters are. Again, we want to maximize the value of $C_a$ for a given dataset.

$Separation_{medoid}$ or $S_m$ for short is the average medoid-to-medoid similarity between all clusters:

$$S_m = \frac{\sum_{i=1}^{K}\sum_{j=i+1}^{K} s(c_i, c_j)}{0.5K(K-1)}. \tag{4.3}$$

A small value for $S_m$ is desirable for a good clustering.

$Separation_{average}$ or $S_a$ for short is the normalized node-to-node similarity between all clusters:

$$S_a = \frac{\sum_{i=1}^{K} \sum_{j=i+1}^{K} \sum_{m=1}^{|C_i|} \sum_{n=1}^{|C_j|} s(x_m, x_n)}{\sum_{i=1}^{K} \sum_{j=i+1}^{K} (|C_i||C_j|)}. \tag{4.4}$$

This value should also be small for a good clustering result.

The first validity index is called Cohesion-Separation or $CS$ for short because it is based on $C_a$ and $S_a$:

$$CS = \frac{C_a}{S_a}. \tag{4.5}$$

For a good clustering result, we expect a large value for $C_a$ and a small value for $S_a$ and hence a large value for $CS$.

The second validity index is called Representativeness of Medoids ($RoM$ for short) because it measures to what degree the medoids are representative of the cluster nodes:

$$RoM = |\log \frac{C_a}{C_m}| + |\log \frac{S_m}{S_a}|. \tag{4.6}$$

If the medoid is representative of the cluster, we expect $C_a$ and $C_m$ to be close and hence the ratio would be close to one. Similarly, we also expect the ratio of $S_m$ to $S_a$ to be close to one. Therefore, a small value for $RoM$ is preferred for good representativeness of medoids.

The time complexity for calculating $CS$ or $RoM$ is $O(n^2)$, where $n$ is the size of the dataset. Clearly, $C_a$ and $S_a$ computationally intensive. To reduce complexity, we

can sample the nodes in the clusters and then only use the sampled nodes to calculate $C_a$ and $S_a$. We calculate the relative error (RMSE and STDEV) due to sampling, and the similarity of clustering results with and without sampling, which means we compared the contents of the clusters of non-sampling version and sampling version.

In addition to the above two validity indices, we also propose another measure called *Concentration of Similarity* ($CoS$) which measures how well the nodes similar to a given node are concentrated in the same cluster. For each node, we calculate its top $m$ most similar nodes (using our similarity measure) within the same cluster — call this set $A$. We compare this with set $B$ consisting of the top $m$ similar nodes among all the nodes in the dataset. The average size of $A \cap B$ (over all the nodes) expressed as a fraction of $m$ is defined as $CoS$. Ideally, we want $CoS$ to be as close to 1 as possible.

Note that $CoS$ and $C_m$ can be used to evaluate not only the whole clustering result but also a single cluster, while $CS$ and $RoM$ generally are only used for the whole clustering result. The value of $CS$, $RoM$, or $CoS$ is independent of the dataset or similarity value, so they can be used as absolute validity indices to compare any two clustering results. Furthermore, unlike $RoM$, $CS$ and $CoS$ are independent of the clustering method.

Some commonly used validity indices can identify only the well separated hyper sphere shaped clusters, since these indices measure the variance of the clusters around

some representative points. However, some clusters, especially the arbitrary shaped clusters, do not have a representative center point (Legány et al. [2006]). But for our validity indices, introduction of $CoS$ can be used to measure arbitrarily shaped clusters.

The traditional $K$-medoid target function/validity index is to minimize the total sum of the distance between each data object and its medoid (Saha and Mukhopadhyay [2008]). It is easy to see that as the value of $K$ increases, each data object generally will be closer to its medoid, which would decrease the value of this target function. So this validity index is not a good measure when $K$ is allowed to vary. In general, our validity index $CS$ does not have this problem, because the denominator $S_a$ may also increase with $K$ (except in some special cases discussed in Theorem 4.1.2).

### *4.1.2 Clustering Algorithm*

Our $K$-medoid clustering method has four steps: initialization, association, re-initialization, and postprocessing.

**Initialization**: The initialization step is to find $K$ initial medoids. Our initialization method consists of three basic steps: sampling, determining first medoid, and successively determining rest of the $K-1$ medoids. First, we select a small sample of

the nodes (usually 5-10% nodes); this can be done either randomly or by picking say one node every 20 or 10 nodes. Then, we take the median of the sample as the first medoid. We define the median as a node with the largest sum of similarities to other nodes in the sample. Finally, for each successive medoid, we select the node that is most dissimilar to any of the previously selected medoids.

**Association**: During Association, we associate each nonmedoid node to its most similar medoid. We break ties randomly or associate the nonmedoid node to the smallest cluster, unless the node is most similar to the medoid of its current cluster, in which case we do not change its cluster membership. For the dataset with original edge, the other option to break ties is to choose the cluster with most records having edge with it.

**Re-initialization**: In this step, we compute the median node (node with largest sum of similarities to other nodes in the same cluster) for each cluster (after Association) and assign it as the new medoid of the cluster. If the new medoid for a cluster is a node that was used as a medoid in a previous iteration, we keep the current medoid. This is done to avoid oscillatory behavior where the medoid of a cluster repeatedly switches between two nodes.

**Postprocessing**: When no new medoids are generated, the Association – Re-initialization loop should be terminated. To improve the quality of clustering, we perform additional postprocessing steps. In our $K$-medoid clustering method, we use

three different postprocessing strategies: *split*, *merge*, and *move*.

The general idea of a split is to split the largest or sparsest cluster. We split the cluster which (1) is too big, say, has more than 50% of all nodes; (2) has the smallest total/normalized nonmedoid-medoid/node-node (intra-cluster) similarity. In our experiments, we tried different split metrics in (2) and we observed that normalized node-node similarity was a good choice. After split, the medoid of the split cluster is replaced by two randomly selected nodes from the respective clusters (locally), or replaced by two nodes from the respective clusters or whole dataset (globally) which are farthest from current medoids.

The general idea of a merge is to merge the smallest clusters or closest clusters. We merge (1) the clusters which are too small, say, each has less than 1% of all nodes, or just has one node; (2) the two clusters which have the largest total/normalized medoid-medoid/node-node(inter-cluster) similarity. Because a merge often results in a decrease in cohesion, we impose some conditions on merge. We merge the two clusters which are: 1) the most similar (has biggest inter-cluster similarity); 2) densest (has biggest intra-cluster similarity); 3) particularly close (the similarity between them is bigger than some factor (e.g., twice) of the second biggest inter-cluster similarity. After merge, the medoids of the merged clusters are replaced by a randomly selected new node from the merged cluster (locally), or replaced by the node from the merged cluster or the whole dataset (globally) which is farthest from current medoids.

We alternate between the split and merge steps to control the number of the clusters. A sequence of split/merge steps can be used to fine tune the total number of clusters, since this is not always known or given. Finally, note that split/merge postprocessing is performed after and followed by several iterations of Association – Re-initialization steps.

The third kind of postprocessing we perform is *move*. We call a node *lonely* if it shares no common Property node with any other nodes in its cluster. During a *move* we move lonely nodes to a different cluster where it shares the most (at least one) common Property node with other nodes in the cluster. This helps increase the quality of the clustering by increasing the cohesion and decrease the separation further. Note that this postprocessing step is executed only at the end of the whole clustering process.

**Termination**: Different termination conditions can be used to stop the execution of the clustering process. For example, when an validity index (like $CS$, $RoM$, or $CoS$) is bigger or smaller than some threshold. Alternatively, after split and merge are executed a given number of times, the clustering process can be stopped. In our experiments, we stop the clustering process after five split/merge steps (splitting of big cluster and the merging of small clusters are not counted).

Since our clustering algorithm is based on $K$-medoid, the computational complexity for finding the medoid for each cluster is $O(n^2)$ where $n$ is the size of the

cluster.

The clustering algorithm is summarized in Figure 4.1 and formalized in Algorithm 1.

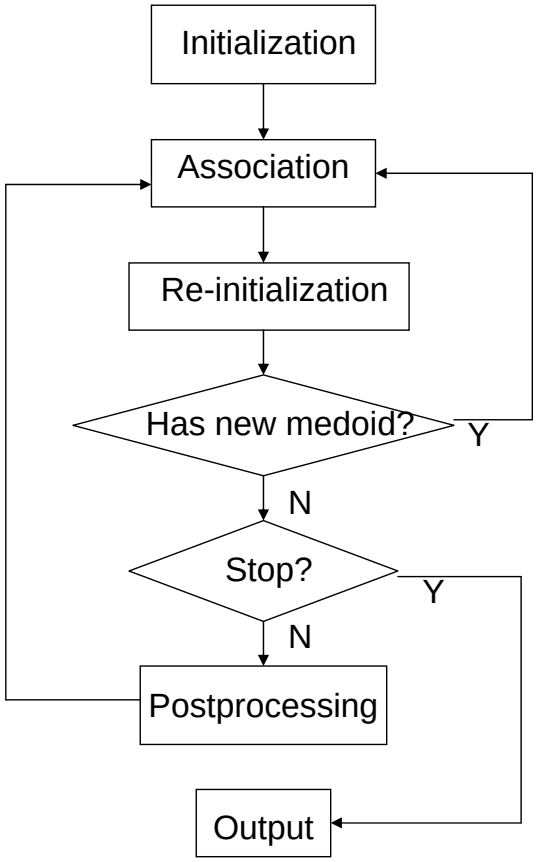


**Figure 4.1:** Flow chart for $K$-medoid clustering algorithm

**Algorithm 1** $K$-medoid clustering algorithm

**Input:** Dataset with size of $n$, and initial cluster number $K$
**Output:** $K'$ clusters

1) Initialization: generate $K$ initial medoids
1.1) take a small subset of all nodes as sample
1.2) take the median node of the sample as the first medoid
1.3) Remaining $K$-1 medoids are selected as the node farthest from previous medoids
1.4) set variable fixup to 1
2) Association: associate each non-medoid to its most similar medoid
3) Re-initialization: select median node of each cluster as medoid
4) Check if there is any cluster having new medoid generated
5) If the answer to 4) is yes, then go back to 2), else go to 6)
6) Check if stop condition is satisfied
7) If the answer to 6) is yes, then go to 9), else go to 8)
8) Postprocessing: split/merge
**if** fixup == $x$ **then**
    go to 8.$x$)
**end if**
8.1) Split the cluster with the smallest intra-cluster average node-node similarity, then set fixup to 2 and go to 2)
8.2)
**if** there is a cluster which has more than 50% of n nodes **then**
    split it and go to 2)
**else**
    set fixup to 3 and go to 8.3)
**end if**
8.3) Merge the two clusters which are the most similar to each other, the densest, and particularly close to each other, then set fixup to 4 and go to 2);
**if** there are no two clusters which satisfy these three conditions **then**
    set fixup to 4 and go to 8.4)
**end if**
8.4) Merge the clusters which have just one node;
**if** the new merged cluster still only has only one node **then**
    merge it to the second smallest cluster, then go to 2);

    **if** there is no small cluster **then**
        reset fixup to 1 and go to 6)
    **end if**
**end if**
9) Postprocessing: move. Move each lonely node to some cluster where it has most links to other nodes
10) Print out the clustering result

### 4.1.3 Influence of Postprocessing on Validity Indices

We now present some theoretical results about the influence of postprocessing on validity indices.

LEMMA 4.1.1. *If the dataset size is fixed, then the number of total intra-cluster pairs is maximal when one cluster's size is maximal and others are minimal.*

LEMMA 4.1.2. *If the dataset size is fixed, then the number of total intra-cluster pairs is minimal when all clusters have the same size or the nodes have the most equitable distribution among clusters.*

Here is the proof for these two lemmas:

*Proof.* Let $x_i$ be the size of cluster $C_i$, let $K$ be the number of clusters and $C$ be the size of the dataset. We have $\sum_{i=1}^{K} x_i = C$. The number of total intra-cluster pairs is: $\sum_{i=1}^{K} \frac{x_i(x_i-1)}{2} = \frac{\sum_{i=1}^{K}(x_i^2-x_i)}{2} = \frac{\sum_{i=1}^{K} x_i^2 - \sum_{i=1}^{K} x_i}{2} = \frac{\sum_{i=1}^{K} x_i^2 - C}{2}$,
so the number of total intra-cluster pairs is maximal or minimal when $\sum_{i=1}^{K} x_i^2$ is maximal or minimal.
1) $\sum_{i=1}^{K} x_i^2 = (\sum_{i=1}^{K} x_i)^2 - \psi = C^2 - \psi$, so when $\psi = 0$, $\sum_{i=1}^{K} x_i^2$ has maximal value of $C^2$, and $\psi = 0$ means that one cluster's size is $C$, and others' is 0. Note that for actual situation, this means that one cluster's size is $C + 1 - K$, and others' is 1.
2) Let $f(x) = \sum_{i=1}^{K} x_i^2 = \sum_{i=1}^{K} x_i^2 + \lambda(\sum_{i=1}^{K} x_i - C)$

Take gradient both sides: $\bigtriangledown f(x) = (2x_i) + \lambda(1)$;

$(2x_i) + \lambda(1) = 0 \Leftrightarrow x_i = -\frac{\lambda}{2}, i = 1, 2, ..n$

$\Leftrightarrow -\frac{n\lambda}{2} = C \Leftrightarrow \lambda = -\frac{2C}{n} \Leftrightarrow x_i = \frac{C}{n}$;

So when $x_i = \frac{C}{n}, f(x)$ has maximum or minimum value. In our case, it is a minimum value. If $\frac{C}{n}$ is a fraction between integers $c$ and $c+1$, then the most equitable distribution of nodes among clusters will have size either $c$ or $c+1$, and that would be the one with smallest sum of $x_i^2$. The reason is as follows: consider the two possible changes (perturbations) that could be made: one is to move one node from a cluster with size $n$ to a cluster with size $n+1$; the other is to move one node from a cluster with size $n+1$ to a cluster with size $n$. For the first situation:

$(n-1)^2 + (n+2)^2 = 2n^2 + 2n + 5 > n^2 + (n+1)^2 = 2n^2 + 2n + 1.$

For the second situation, the nodes distribution is not changed.

■

LEMMA 4.1.3. *Given a clustering of a dataset of size $n$, let $k+1 \geq 2$ be the number of clusters, and $x$ be the size of the biggest cluster. After a postprocessing (followed by re-association) of the given clustering result, let $x'$ be the size of the biggest cluster (again, assuming we have at least two clusters). If $x' < \frac{1+\sqrt{1+2\left(\frac{(n-x)^2}{k} - n + x^2\right)}}{2}$, then the total number of pairs of nodes, both belonging to the same cluster, after postprocessing is smaller than that before postprocessing.*

We make use of lemma 4.1.1 and 4.1.2 for the proof for lemma 4.1.3:

*Proof.* Let $m$ be the size of the second biggest cluster after postprocessing, and $t$ be average size of the clusters other than the biggest one before postprocessing (note that $kt = n - x$), then according to the previous two lemmas, we have:

the minimal number of total intra-cluster pairs before postprocessing is: $\frac{t(t-1)}{2}k + \frac{x(x-1)}{2}$;

the maximal number of total intra-cluster pairs after postprocessing is: $\frac{m(m-1)}{2} + \frac{x'(x'-1)}{2}$.

$\frac{m(m-1)}{2} + \frac{x'(x'-1)}{2} < \frac{t(t-1)}{2}k + \frac{x(x-1)}{2}$

$\Leftrightarrow m(m-1) + x'(x'-1) < t(t-1)k + x(x-1)$

$\Leftarrow 2x'(x'-1) < t(t-1)k + x(x-1)$

$\Leftrightarrow 2x'^2 - 2x' - (t^2k - tk + x^2 - x) < 0$

$\Leftrightarrow x' < \frac{1+\sqrt{1+2(t^2k-tk+x^2-x)}}{2}$

$\Leftrightarrow x' < \frac{1+\sqrt{1+2(\frac{(n-x)^2}{k}-n+x^2)}}{2}$

■

For example, with $n = 500$, $k = 10$, and $x = 100$, the condition in lemma 4.1.3 is satisfied if the size of the biggest cluster after postprocessing is $x' \leq 113$.

Following is a simple sufficient condition for the condition in lemma 4.1.3: $x < \frac{\sqrt{kn^2-k^2n+kn}-n}{k-1}, k > 1$ and $x' < x$.

*Proof.* $x' < \frac{1+\sqrt{1+2(t^2k-tk+x^2-x)}}{2}$

$\Leftarrow x' < x$ and $x < \frac{1+\sqrt{1+2(t^2k-tk+x^2-x)}}{2}$;

$x < \frac{1+\sqrt{1+2(t^2k-tk+x^2-x)}}{2} \Leftrightarrow (2x-1)^2 < 1+2(t^2k-tk+x^2-x)$

$\Leftrightarrow x^2 - x - (t^2k-tk) < 0 \Leftrightarrow x < \frac{1+\sqrt{1+4(t^2k-tk)}}{2} = \frac{1+\sqrt{1+4(\frac{(n-x)^2}{k}-n+x)}}{2}$

$\Leftrightarrow (2x-1)^2 < 1+4(\frac{(n-x)^2}{k} - n + x) \Leftrightarrow (k-1)x^2 + 2nx - (n^2-nk) < 0$

$\Leftrightarrow x < \frac{\sqrt{kn^2-nk^2+nk}-n}{k-1}$

■

For example, with $n = 500$ and $k = 10$, this condition is satisfied if the size of the biggest cluster before postprocessing is $x \leq 118$. Note that after a split, we often have $x' < x$.

The following Theorem (which uses Lemma 4.1.3) explains the effect of post-processing on $C_a$.

THEOREM 4.1.1. *For a given clustering of a dataset, let $n_1$ be the total number (over all clusters) of pairs of nodes, both belonging to the same clusters, $n_2$ be the total number of pairs of nodes, each belonging to a different cluster. After a postprocessing followed by re-association of the given clustering result, let $n_{11}$ (out of the original $n_1$) be the number of pairs of nodes still in same cluster, $n_{12} = n_1 - n_{11}$ be the pairs of nodes now separated into two different clusters. Furthermore, let $n_{22}$ (out of the original $n_2$) be the number of pairs of nodes now assigned to same cluster. Let $s_{11}$, $s_{12}$, $s_{22}$ respectively be the average similarity value of the $n_{11}$, $n_{12}$, $n_{22}$ pairs of nodes. Let $C_a$ be the $C_a$ value of the clustering result before split and $C'_a$ be the $C_a$ value after postprocessing (followed by re-association). We then have:*

*If the assumptions of Lemma 4.1.3 are satisfied,* $\frac{s_{11}}{s_{22}} \triangleq \theta > 1$, *and* $\frac{s_{12}}{s_{22}} < \delta$, *then* $C'_a > C_a$, *where* $\delta \triangleq \frac{n_{11}n_{22}+n_{12}n_{22}+\theta(n_{11}n_{12}-n_{11}n_{22})}{n_{11}n_{12}+n_{12}n_{22}} > 1$.

*Proof.* Because of split, $n_{12} > n_{22}$, so $n_{11}n_{12} - n_{11}n_{22} > 0 \Rightarrow \theta(n_{11}n_{12} - n_{11}n_{22}) > n_{11}n_{12} - n_{11}n_{22}$

$\Rightarrow n_{11}n_{22} + \theta(n_{11}n_{12} - n_{11}n_{22}) > n_{11}n_{12}$

$\Rightarrow n_{11}n_{22} + n_{12}n_{22} + \theta(n_{11}n_{12} - n_{11}n_{22}) > n_{11}n_{12} + n_{12}n_{22}$

$\Rightarrow \delta > 1.$

$\frac{s_{12}}{s_{22}} < \delta \Rightarrow \frac{s_{12}}{s_{22}}(n_{11}n_{12} + n_{12}n_{22}) < n_{11}n_{22} + n_{12}n_{22} + \theta(n_{11}n_{12} - n_{11}n_{22})$

$\Rightarrow \frac{s_{12}}{s_{22}}(n_{11}n_{12} + n_{12}n_{22}) < n_{11}n_{22} + n_{12}n_{22} + \frac{s_{11}}{s_{22}}(n_{11}n_{12} - n_{11}n_{22})$

$\Rightarrow s_{12}(n_{11}n_{12} + n_{12}n_{22}) + s_{11}n_{11}n_{22} < s_{22}n_{11}n_{22} + s_{22}n_{12}n_{22} + s_{11}n_{11}n_{12}$

$\Rightarrow s_{11}n_{11}n_{11} + s_{12}(n_{11}n_{12} + n_{12}n_{22}) + s_{11}n_{11}n_{22} < s_{11}n_{11}n_{11} + s_{22}n_{11}n_{22} + s_{22}n_{12}n_{22} + s_{11}n_{11}n_{12}$

$\Rightarrow (s_{11}n_{11} + s_{12}n_{12})(n_{11} + n_{22}) < (s_{11}n_{11} + s_{22}n_{22})(n_{11} + n_{12})$

$\Rightarrow \frac{s_{11}n_{11}+s_{12}n_{12}}{n_{11}+n_{12}} < \frac{s_{11}n_{11}+s_{22}n_{22}}{n_{11}+n_{22}}$

$\Rightarrow C_a < C'_a.$

■

This theorem shows that if $s_{11} > s_{22}$ (which is usually true for a postprocessing step) and the condition of Lemma 4.1.3 is satisfied, then postprocessing (split/merge/move) usually results in an increase in $C_a$, as long as $\frac{s_{12}}{s_{22}} < \delta$. Note that $C_a$ would increase, even if $s_{12} > s_{22}$ (which indicates a bad reassociation), since

$\delta > 1$. Moreover, the sufficient condition given by the sufficient condition of lemma 4.1.3 is often satisfied after a split; this explains why after a split, $C_a$ often increases.

We now present a result about the relation between the number of clusters $K$, $C_a$, and $S_a$.

THEOREM 4.1.2. *Given $C_a$ and $S_a$ for some nontrivial ($1 < K < n$) clustering result with $K = k_1$ on a dataset of size $n$, and for any $1 \leq i \leq k_1$, $|C_i| = \frac{n}{k_1}$; i.e., all the clusters are of the same size. Furthermore, given $C'_a$ and $S'_a$ for a different nontrivial clustering result with $K = k_2$ (caused by postprocessing on the same dataset), and for any $1 \leq i \leq k_2$, $|C_i| = \frac{n}{k_2}$; i.e., all the clusters are again of the same size. Then $\frac{S'_a}{S_a} < \frac{k_2(k_1-1)}{k_1(k_2-1)}$ if and only if $\frac{C'_a}{C_a} > \frac{k_2(n-k_1)}{k_1(n-k_2)}$.*

*Proof.* Note that the sum of similarities between all pairs of nodes is a constant for a given dataset. For a given clustering, this sum can be decomposed into two components: (a) sum of similarities between pairs of nodes, both belonging to the same cluster; by definition of $C_a$, this is $C_a$ times the number of pairs of nodes, both in same cluster. (b) sum of similarities between pairs of nodes, each belonging to a different cluster; by definition of $S_a$, this is $S_a$ times the number of pairs of nodes, each in a different cluster. Apparently the sum of $C_a$ * (number of pairs of movies in same clusters) and $S_a$ * (number of pairs of movies in different clusters) is a constant,

Therefore,

$$\frac{n}{2k_1}(\frac{n}{k_1}-1)k_1C_a+\frac{n^2}{k_1^2}\frac{k_1}{2}(k_1-1)S_a=$$
$$\frac{n}{2k_2}(\frac{n}{k_2}-1)k_2C'_a+\frac{n^2}{k_2^2}\frac{k_2}{2}(k_2-1)S'_a.$$

Rearranging, we get

$$k_2nC_a-k_1nC'_a-k_1k_2(C_a-C'_a)=$$
$$nk_2S_a-nk_1S'_a-nk_1k_2(S_a-S'_a). \tag{4.7}$$

Now observe that

$$\frac{S_a}{S'_a}>\frac{k_1(k_2-1)}{k_2(k_1-1)}$$
$$\Leftrightarrow S_ak_2(k_1-1)>S'_ak_1(k_2-1)$$
$$\Leftrightarrow k_2S_a-k_1S'_a-k_1k_2(S_a-S'_a)<0$$
$$\Leftrightarrow nk_2S_a-nk_1S'_a-nk_1k_2(S_a-S'_a)<0$$
$$\Leftrightarrow k_2nC_a-k_1nC'_a-k_1k_2(C_a-C'_a)<0$$
$$\Leftrightarrow (k_2n-k_1k_2)C_a<(k_1n-k_1k_2)C'_a$$
$$\Leftrightarrow \frac{C_a}{C'_a}<\frac{k_1(n-k_2)}{k_2(n-k_1)},$$

where we have used equation (4.7) in the third step. This completes the proof.

■

In particular, if $k_2>k_1$; i.e., the number of clusters increases, then $\frac{k_2(k_1-1)}{k_1(k_2-1)}<1$, and $\frac{k_2(n-k_1)}{k_1(n-k_2)}>1$. Therefore, if $\frac{S'_a}{S_a}<\frac{k_2(k_1-1)}{k_1(k_2-1)}$ or equivalently $\frac{C'_a}{C_a}>\frac{k_2(n-k_1)}{k_1(n-k_2)}$, then

$S_a > S'_a$ and $C_a < C'_a$. In other words, an increase in $C_a$ to some degree is accompanied by a decrease in $S_a$ (which ultimately increases $CS$), when the number of clusters increases. This is often the case for a postprocessing split step. Some clustering methods like OPPOSUM partition the data into roughly equal-sized clusters (Tan et al. [2006]), and for some applications like clustering market baskets, each cluster should contain roughly the same number of samples (Pal and Jain [2005]). Theorem 4.1.2 can be directly applied to these methods and applications.

## 4.2 Multi-Medoid $K$-medoid Clustering

$K$-medoid clustering uses the most central object in a cluster as the representative object in the cluster, and associate the other objects to this cluster which is most similar to this representative object. When we use distance (e.g., in Euclidian space) to represent the dissimilarity between two objects, $K$-medoid would perform well because of the transitive property, which means if two objects are close to some medoid, then they must be close to each other. However, when we use other similarity measures, we can not assure this transitive property. For this reason, we proposed an modified $K$-medoid clustering method: multi-medoid $K$-medoid clustering. In this method, each cluster has more than one medoid to represent this cluster, which would increase the probability of being similar for the objects associated together.

For simplicity, we let all clusters have the same number of medoids.

The validity indices and clustering algorithm of multi-medoid $K$-medoid clustering is similar to those of the $K$-medoid clustering described in previous sections. We will focus on the difference in the following sections.

### *4.2.1 Validity Indices*

For $C_m$, for each nonmedoid node, we use its average similarity to medoids as its nonmedoid-to-medoid similarity:

$$C_m = \frac{\sum_{i=1}^{K} \sum_{j=1}^{|C_i|-n} \sum_{l=1}^{n} s(c_{i_l}, x_j)}{n \sum_{i=1}^{K} (|C_i| - n)}, \tag{4.8}$$

where $n$ is the number of medoids in each cluster and $c_{i_1}, \ldots, c_{i_n}$ are the $n$ medoids of the $i$th cluster.

For $S_m$, we have the similar modification:

$$S_m = \frac{\sum_{i=1}^{K} \sum_{j=i+1}^{K} \sum_{h=1}^{n} \sum_{l=1}^{n} s(c_{i_h}, c_{j_l})}{0.5n^2K(K-1)}. \tag{4.9}$$

Note that no matter how many medoids each cluster has, the calculation of $C_a$ and $S_a$ remains the same.

### *4.2.2 Clustering Algorithm*

We focus on the difference compared with the algorithm specified in section 4.1.2.

**Initialization**: Using the method for $K$-medoid clustering, we get the first medoid for each cluster. Then for each cluster, we choose some node as the next medoid of it whose existing medoids have biggest average similarity with this node. We are done when each cluster has $n$ medoids.

**Association**: During Association, we associate each nonmedoid node to its most similar medoid. Similar to the modification we did for $C_m$, we use average similarity as nonmedoid-to-medoid similarity.

**Re-initialization**: In this step, we compute the top $n$ median nodes for each cluster (after Association) and assign them as the new medoids of the cluster.

**Postprocessing and Termination**: Only difference is caused by the modified calculation method for the validity indices.

We now discuss the computational complexity of the multi-medoid $K$-medoid algorithm compared to the single medoid $K$-medoid algorithm. For Initialization step, for each cluster, the computations needed to calculate the medoids other than the first one is much less than that needed to calculate the first medoid. Therefore, the computational complexity of the initialization step for the multi-medoid $K$-medoid

algorithm is similar to that of the single medoid $K$-medoid algorithm. For Association step, the computational complexity is $n$ times that of single medoid $K$-medoid algorithm, where $n$ is the number of medoids in each cluster. For Re-initialization step, the computations are similar to that of the single medoid $K$-medoid algorithm, because for both methods, to get the top one median, we need to calculate the average similarity between each node and all the other nodes. For Postprocessing step, the multi-medoid $K$-medoid algorithm requires more computations than single medoid $K$-medoid algorithm, due to the calculation of $C_m$ and $S_m$. A precise expression for the increased complexity is difficult to obtain. In chapter 7, we shall present experimental results about the running time for both single medoid and multi-medoid algorithms.

## 4.3 Edge-Oriented Node Move Algorithm

For the dataset with original edge, sometimes we prefer to group the data nodes with edge between them into the same cluster, with a reasonable balance of the size and number of the clusters. For this situation, we propose an edge-oriented node move algorithm which can be applied after the termination step of our clustering algorithm. Indeed, this node move algorithm can be applied to any clustering result for this kind of dataset and preference. Our edge-oriented node move algorithm is

described in algorithm 2.

The basic idea of this algorithm is to move records between clusters. A cluster is a record's moving destination if it has the most number of this records' direct neighbors. In each iteration, we move some records to their destination cluster which is the destination for the most number of records, repeatedly, until there is no record that needs to be moved — usually because some restrictions are reached. These restrictions are used to limit the the scale of move in each iteration so as to control the balance of the size of the clusters (avoid too big cluster). We reset the restrictions at the beginning of each iteration.

Note that after node move, some cluster(s) could be empty, so the number of clusters $K'$ after applying this algorithm could be smaller than $K$.

Now let us discuss why algorithm 2 generally converges. First, there is only one destination cluster for each iteration for the inner loop (lines 16 through 60); therefore, after a node is moved to the destination cluster, this cluster will still have the most direct neighbors of this node. Second, for the inner loop, each node is moved at most one time (lines 27 through 29). Third, we use some variables (e.g. $canbeMovedIn$, $numMoveOut$, $numMoves$) to control the move behavior. Fourth, in each subsequent iteration, the number of nodes that need to be moved decreases in general because more and more nodes have their most direct neighbors in the same cluster. Lastly, we use variable $numIteration$ to control the number of outer loops.

**Algorithm 2** Edge-Oriented Node Move Algorithm

**Input:** $K$ clusters for dataset with original edge, number of iterations $numIteration$
**Output:** $K'$ clusters

1: **for** each record $i$ **do**
2: calculate how many direct neighbors it has in each cluster $j$ as $numNeighborInCluster_{i_j}$
3: **end for**
4: set $x = 0$
5: **loop**
6: $x = x + 1$
7: **for** each cluster $i$ **do**
8: set $canbeMovedIn_i = true$
9: set $previousIsMoveIn_i = false$
10: set $numMoveIn_i = numMoveOut_i = 0$
11: **end for**
12: **for** each record $i$ **do**
13: set $numMoves_i = 0$
14: **end for**
15: set $numMoves = -1$
16: **loop**
17: **if** numMoves==0 **then**
18: jump out loop
19: **end if**
20: calculate the average size of all non-empty clusters as $avgClusterSize$
21: **for** each cluster $i$ **do**
22: set $numAsDestination_i = 0$
23: **end for**
24: set $numMoves = 0$
25: **for** each cluster $i$ **do**
26: **for** each record $p$ in cluster $i$ **do**
27: **if** $numMoves_p == 1$ **then**
28: skip record $p$
29: **end if**
30: set $numAsDestination_j = numAsDestination_j + 1$ where $j = \text{argmax}_j\, numNeighborInCluster_{p_j}$ and $j \neq i$ and $canBeMovedIn_j == true$, add $< p, j >$ to $destHashTable_i$
31: **end for**
32: **end for**
33: set $dest = \text{argmax}_i\, numAsDestination_i$

**Algorithm 2** Edge-Oriented Node Move Algorithm (continued)

```
34:         for each cluster i do
35:             if i == dest then
36:                 skip cluster i
37:             end if
38:             add all the records whose hash value in destHashTable_i is dest to a
    set records
39:             if |records| > avgClusterSize then
40:                 skip cluster i
41:             end if
42:             for each record p in records do
43:                 remove p from cluster i, add p to cluster dest, update
    corresponding values in numNeighborInCluster
44:                 set numMoves = numMoves + 1, numMoves_p = numMoves_p + 1
45:             end for
46:             set numMoveOut_i = numMoveOut_i + |records|
47:             if |records| > 0 then
48:                 set previousIsMoveIn_i = false
49:             end if
50:             if canBeMovedIn_i == false and numMoveIn_i > 0 and
    (numMoveIn_i − numMoveOut_i) < avgClusterSize then
51:                 set canBeMovedIn_i = true
52:             end if
53:         end for
54:         if canBeMovedIn_dest == true and
    (numMoves − numMoveOut_dest) > avgClusterSize then
55:             set canBeMovedIn_dest = false
56:         end if
57:         if numMoves > 0 then
58:             set numMoveIn_dest = numMoves, previousIsMoveIn_dest = true,
    numMoveOut_dest = 0
59:         end if
60:     end loop
61:     if x == numIteration then
62:         jump out loop
63:     end if
64: end loop
```

Experimental results show that the number of nodes moved in each iteration of inner loop has an oscillating characteristic with respect to an overall decreasing trend.

## 4.4 Interpretation of Clustering Result with Domain Independent Similarity Measure

In our similarity measure for clustering, generally we use all the available (related) attributes, but sometimes we do not have some attribute, and we want to see if we can interpret the clustering result for the missed attribute. With this so-called domain independent similarity measure, in our experiments, we tested our clustering results for movies for the interpretation of *Director* attribute.

## 4.5 Fuzzy $K$-medoid Clustering

If data objects are distributed into well-separated groups, then a crisp classification of the objects into disjoint clusters would be desirable. However, in many cases, the objects in a dataset cannot be partitioned into well-separated clusters. Based on this consideration, we also implemented a fuzzy clustering method on the movie dataset. We implemented the fuzzy version of $K$-medoid (Tan et al. [2006], Stein and Eissen [2002]) so that we can compare the two clustering results.

### *4.5.1 Validity Indices*

We used the same similarity measures for fuzzy $K$-medoid clustering. Also, we used the same evaluation measures as we used in $K$-medoid method. However, the calculation of cohesion and separation evaluation measures are different because of the introduction of membership weights (degrees) for each node. Correspondingly, they are modified as below:

$$C_m = \frac{\sum_{i=1}^{K}\sum_{j=1}^{n} w_{ii}^p w_{ji}^p s(m_i, x_j)}{\sum_{i=1}^{K}\sum_{j=1}^{n} w_{ii}^p w_{ji}^p} \tag{4.10}$$

$$C_a = \frac{\sum_{i=1}^{K}\sum_{j=1}^{n}\sum_{l=1}^{n} w_{ji}^p w_{li}^p s(x_j, x_l)}{\sum_{i=1}^{K}\sum_{j=1}^{n}\sum_{l=1}^{n} w_{ji}^p w_{li}^p} \tag{4.11}$$

$$S_m = \frac{\sum_{i=1}^{K}\sum_{j=i+1}^{K} w_{ii}^p w_{ji}^p s(m_i, m_j)}{\sum_{i=1}^{K}\sum_{j=i+1}^{K} w_{ii}^p w_{ji}^p} \tag{4.12}$$

$$S_a = \frac{\sum_{i=1}^{K}\sum_{j=i+1}^{K}\sum_{m=1}^{n}\sum_{n=1}^{n} w_{mi}^p w_{nj}^p s(x_m, x_n)}{\sum_{i=1}^{K}\sum_{j=i+1}^{K}\sum_{m=1}^{n}\sum_{n=1}^{n} w_{mi}^p w_{nj}^p} \tag{4.13}$$

Some notations not listed in Table 4.1 are formalized in Table 4.2.

**Table 4.2:** Notations used in evaluation measures for fuzzy $K$-medoid

| | |
|---|---|
| $n$ | size of dataset |
| $w_{ij}(i \neq j)$ | weight with which $i$th node belongs to cluster $C_j$ |
| $w_{ii}$ | weight with which $i$th medoid belongs to cluster $C_i$ |
| $p$ | influence of the weight, $0 < p < \infty$ |

### *4.5.2 Clustering Algorithm*

The fuzzy $K$-medoid clustering method we use is similar to the $K$-medoid method described in Section 4.1. It includes four steps: initialization, computation of the fuzzy pseudo-partition (weights), re-initialization, and postprocessing.

**Initialization**: Similar to $K$-medoid method. Choose $K$ initial medoids, as dissimilar as possible to each other.

**Computing fuzzy pseudo-partition**: In this step, we calculate the weight of each node for each cluster as follows:

$$w_{ij} = \frac{s(x_i, m_j)}{\sum_{l=1}^{K} s(x_i, m_l)}. \tag{4.14}$$

In particular, for medoid, we have:

$$w_{ii} = \frac{s(m_i, m_i)}{\sum_{l=1}^{K} s(m_i, m_l)} = \frac{1}{\sum_{l=1}^{K} s(m_i, m_l)}. \tag{4.15}$$

**Re-initialization**: Re-compute the medoid of each cluster using the fuzzy pseudo-partition. The new medoid of $i$th cluster is the one which maximizes (4.16), where $x_j$ is any node in the dataset.

$$\sum_{j=1}^{n} w_{ji}^{p} w_{ii}^{p} s(x_j, m_i) \tag{4.16}$$

**Postprocessing**: Same as in the $K$-medoid clustering and includes *split*, *merge*, and *move*.

The fuzzy clustering algorithm is illustrated in Figure 4.2. Because the algorithm is very similar to that of $K$-medoid, we will not list it here.

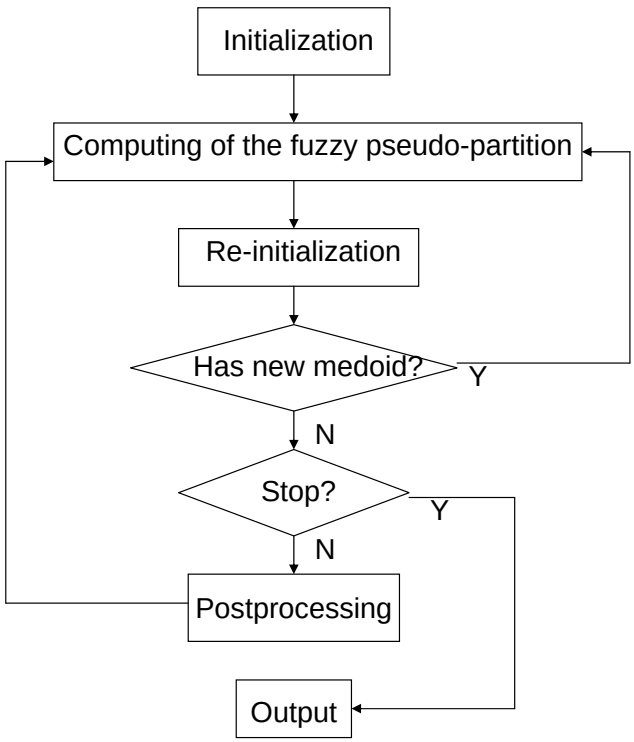


**Figure 4.2:** Flow chart for fuzzy $K$-medoid clustering algorithm

## 4.6 Visualization Tool

We have designed a cluster visualization tool to visualize the clustering content in a two-dimension space. This tool has following features:

1. Display a particular cluster or all clusters: it can display the content of any number of clusters on the canvas.

2. Display cluster(s) of any iteration step: it can display the clustering result at the end of any iteration during the clustering process.

3. Node search and highlight / de-highlight: user can use node title (like movie's name for movie dataset) to search from the nodes in the displayed cluster(s). Partial match is allowed. The search result node(s) will be highlighted and can be de-highlighted.

4. Two modes of display: it can display cluster content in two modes: nonmedoid-medoid and node-node. In nonmedoid-medoid mode, nonmedoid nodes are connected to medoid node; in node-node mode, two nodes are connected if their similarity value is bigger than 0.

5. Pause/resume/stop drawing of cluster: when cluster content is being drawn, user can pause/resume or stop the drawing process.

6. Indicate processing percentage when drawing: when cluster content is being drawn, the processing percentage is indicated.

7. Highlight / de-highlight dominant node: for movie dataset, dominant movie is the one which shares the most Names and Roles with other movies in the same cluster.

8. Scatter cluster nodes on the canvas: normally the nodes are arranged on the canvas in a regular style. User can scatter the nodes which will be displayed on the canvas in a random way.

9. Filter: for movie dataset, it can be selected to only display the nodes whose numbers of Name and Role are within some ranges. Note that medoid is always displayed.

10. Zoom in/out with two different ratios: user can zoom in or out the canvas with two different ratios: 1.1 or 2. Zooming times may be limited due to the resource limit (like memory).

11. Navigation: the canvas provides horizontal and vertical navigation bars when visible area is smaller than the actual size of the canvas.

12. Small canvas (thumbnail): their is a small canvas right beside the big one with the same content as big canvas but better global view. It has navigation function

which means when you click on the small canvas the corresponding place will be moved into the view of the big canvas.

13. Node/edge selection: node or edge on the canvas can be selected and highlighted and corresponding information will be displayed to the user.

14. Statistics: it can show statistics of clusters:

    - Size of each cluster before/after filtering
    - Max, min, mean, stddev of number of shared Names and Roles for all pairs of two movies in each cluster (for movie dataset)
    - Dominant node in each cluster
    - Dominant Name and Role in each cluster (for movie dataset, dominant Name/Role is the Name/Role which is shared the most times in that cluster)

This tool helped us better check and understand the clustering results (for movies). For example, it can visually display the medoid movie and its connections (relationship) with other movies in the same cluster.

Figure 4.3 is the content of one cluster displayed on the canvas. The nonmedoid-medoid mode is used and the nodes are arranged on the canvas in a regular way.

Figure 4.4 shows that one node is selected. The node-node mode is used and the nodes are scattered on the canvas in a random way.

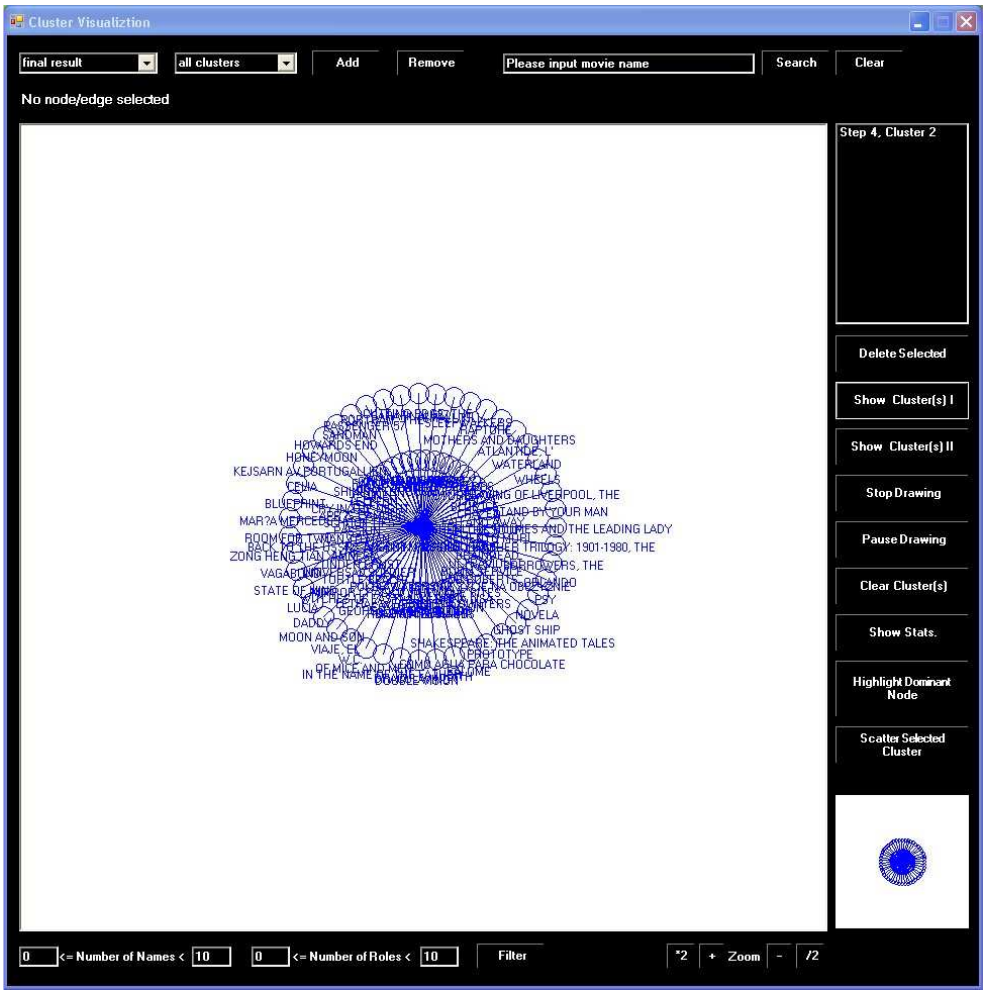


**Figure 4.3:** Cluster visualization tool: cluster

Figure 4.5 shows that one edge is selected. The node-node mode is used and the nodes are scattered on the canvas in a random way.

Figure 4.6 shows that user searches the node which contains string "the." We can see that 3 nodes are found and highlighted with black color.

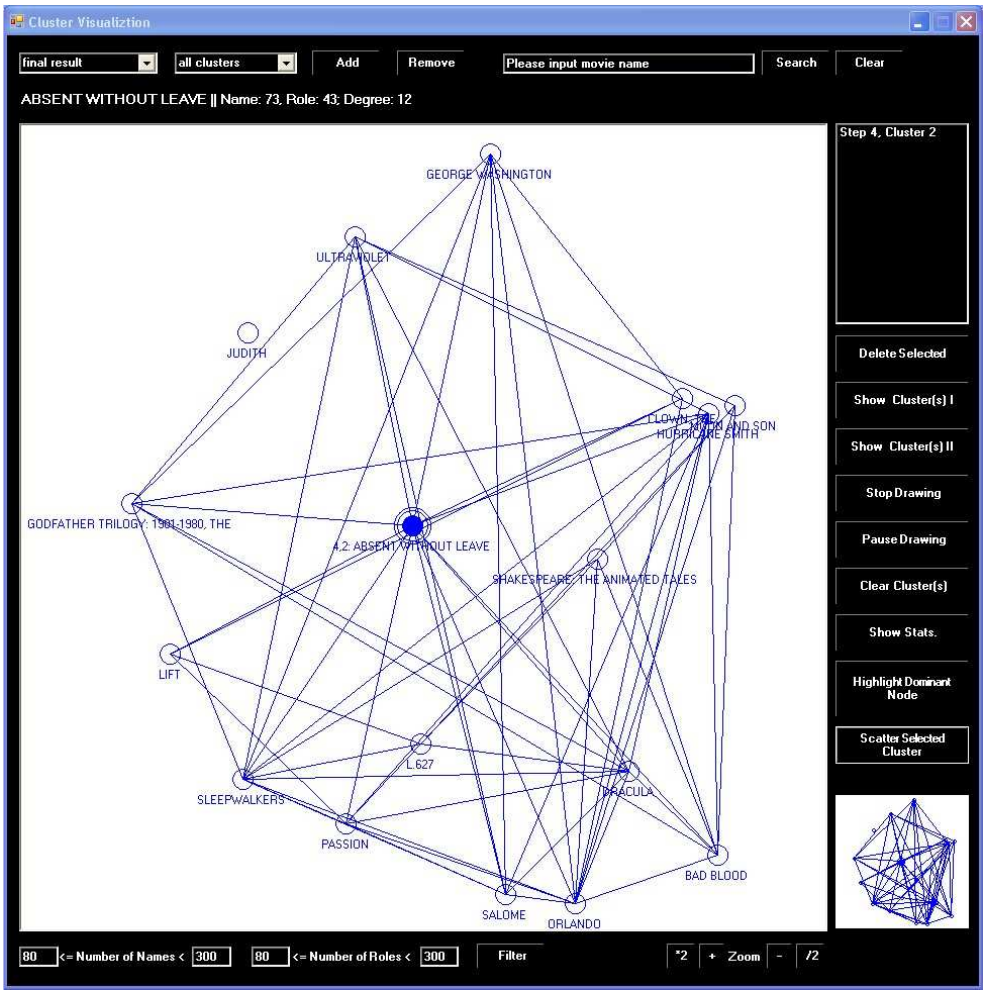


**Figure 4.4:** Cluster visualization tool: node

Figure 4.7 shows the dominant node in a cluster. It is highlighted with red color.

Figure 4.8 shows the visible area of the canvas after zoom in. Apparently the canvas was enlarged.

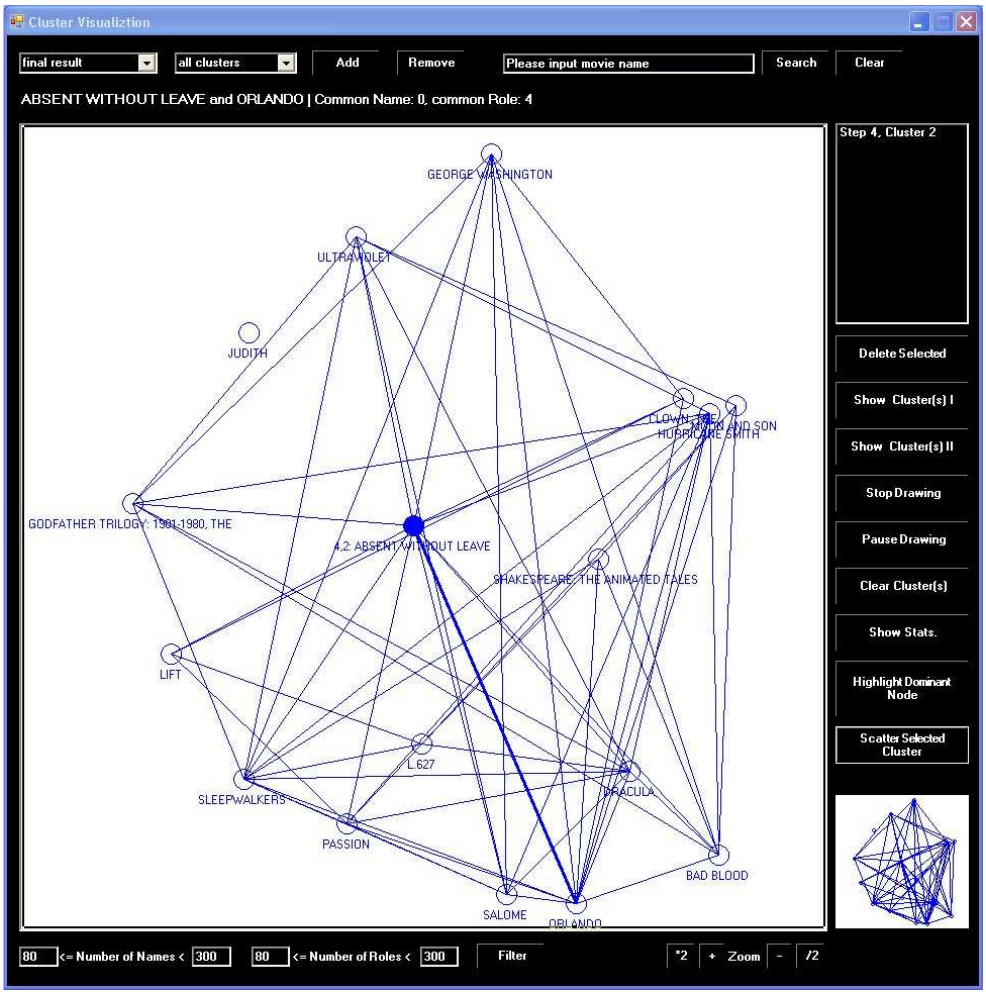


**Figure 4.5:** Cluster visualization tool: edge

Figure 4.9 shows the statistics window which displays corresponding information of cluster 2 after 4th iteration.

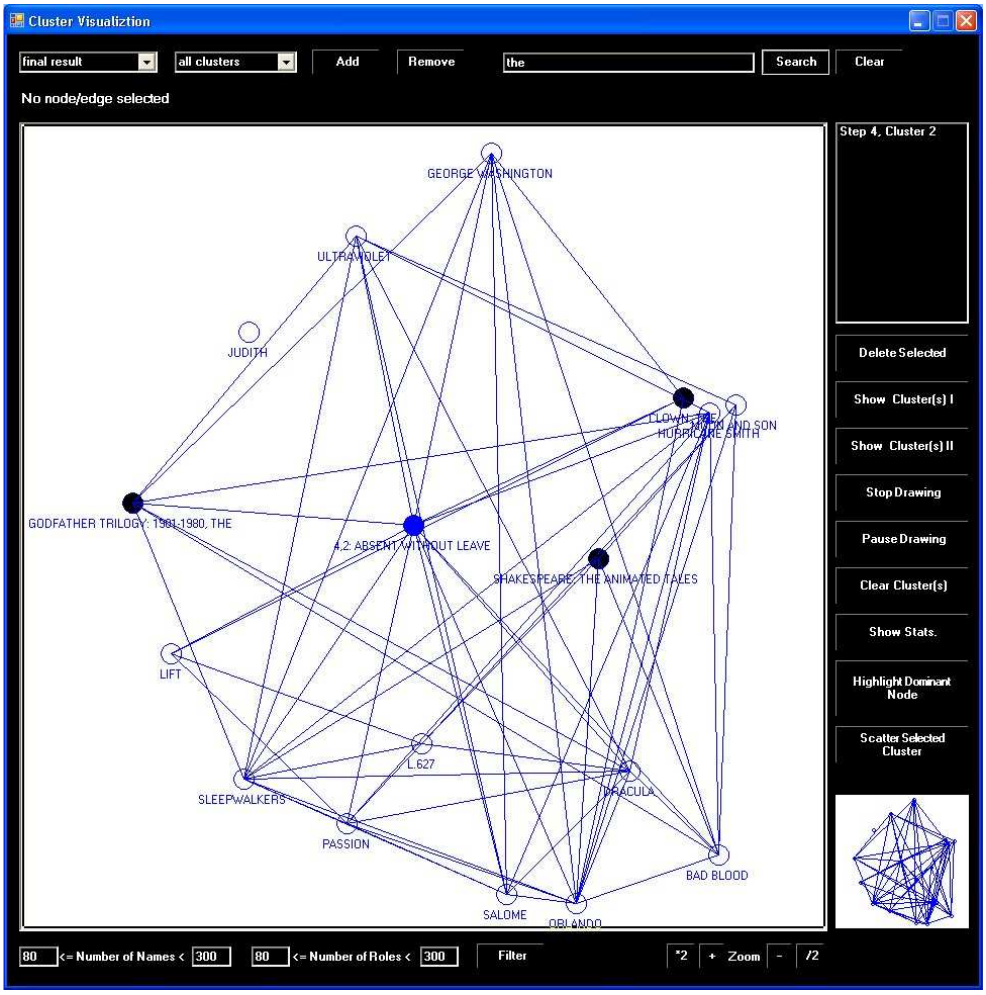


**Figure 4.6:** Cluster visualization tool: search

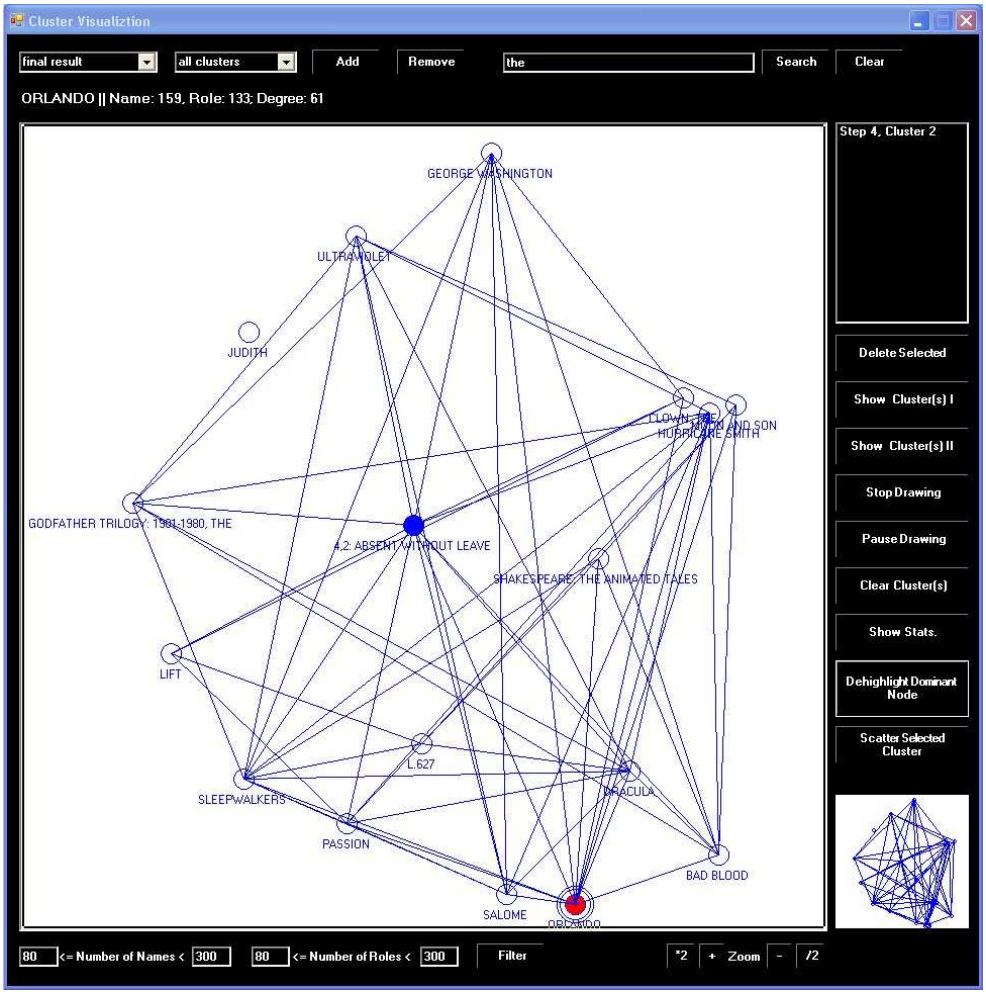


**Figure 4.7:** Cluster visualization tool: dominant node

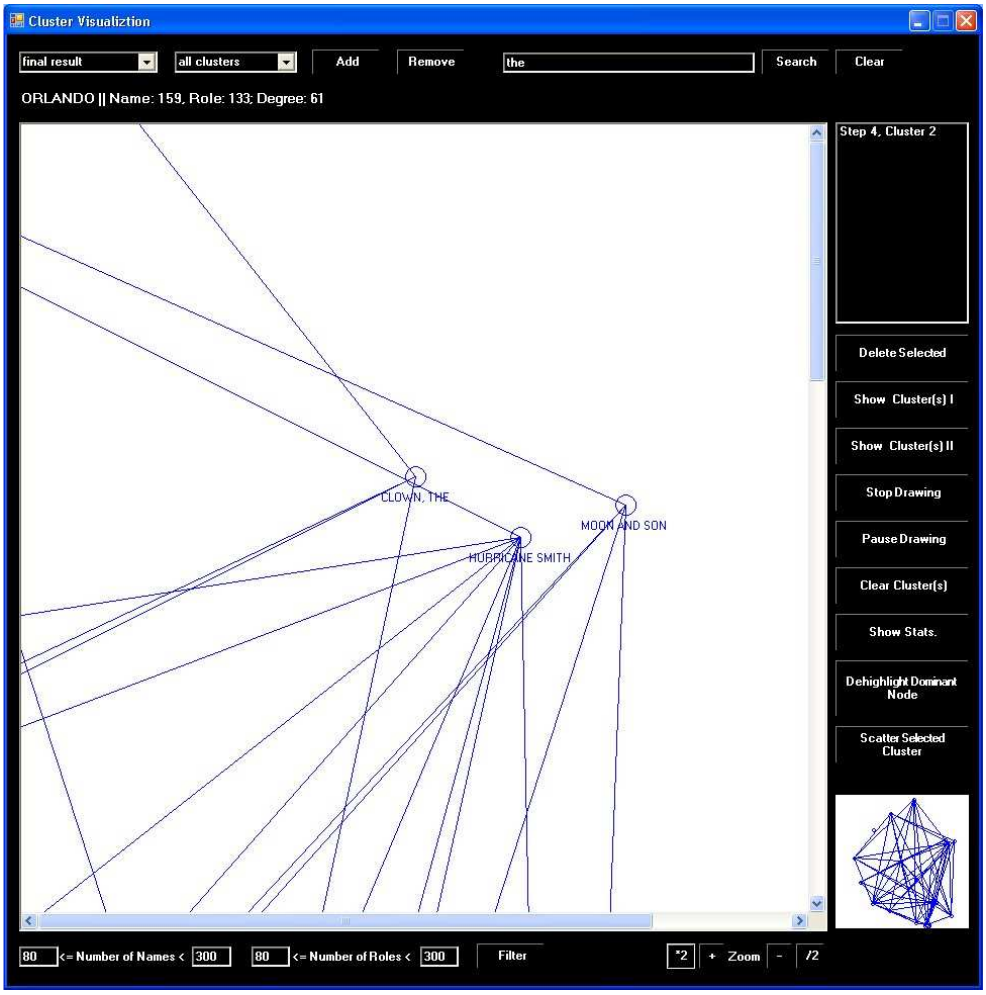


**Figure 4.8:** Cluster visualization tool: zoom

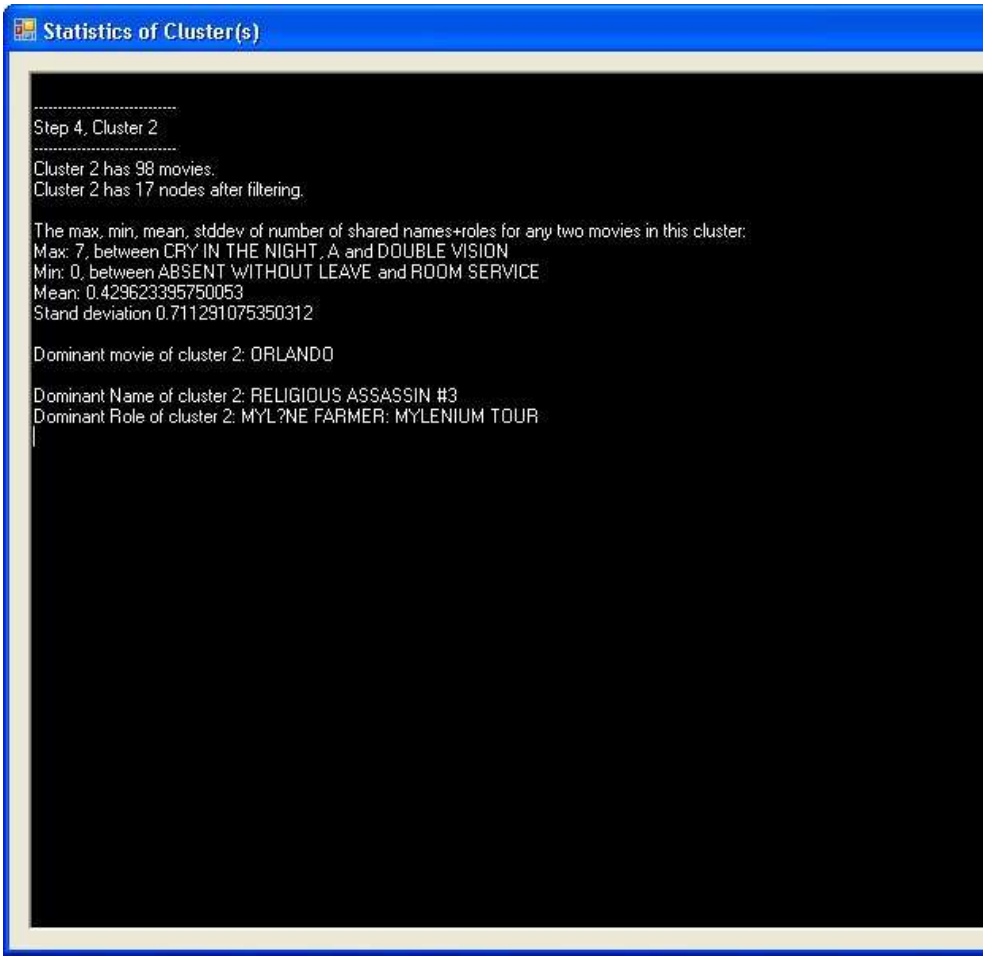


**Figure 4.9:** Cluster visualization tool: statistics

# CHAPTER 5. ENTITY RESOLUTION

## 5.1 Problem definition

The Fellegi-Sunter model Fellegi and Sunter [1969] formalizing the approach of Newcombe *et al.* Newcombe et al. [1959] is commonly used in the literature. We use a notation similar to that of the Fellegi-Sunter model with some differences. Our definition of entity resolution problem is as follows:

Definition 5.1.1. Entity resolution *is the process which takes one or more sets of records $D$ as input and outputs the set of records $T$ with "true" attributes and the equivalent/nonequivalent relation set $R$ for any pair of records in $D$.*

In $D$, some records are equivalent records of other records. The purpose of entity resolution is to find the relation between any two records in $D$ where the relation is either "equivalent" or "nonequivalent". Also, entity resolution needs to find the set of records $T$ for $D$ with "true" attributes, which means that for a records set $S$ (a subset of $D$) in which all the records are equivalent of one another, it needs to find a record $r$ which is the "true" record for all the records in $S$. $r$ could be some record in $S$, or a record different from any record in $S$. In our entity resolution method, we

always find a record in $S$ as $r$.

In next section we will present our framework to solve the entity resolution problem.

## 5.2 Framework

We call our method ERUDITE which stands for Entity Resolution with record Updating and Duplicate & InconsisTency Elimination. (When we named our framework we used term "duplicate" instead of "equivalent" which we use now.) As illustrated in Figure 5.1, ERUDITE takes $D$ as input and outputs $T$ and $R$.

In the data preprocessing step, data preparation can be used which often precedes the entity resolution process. The data preparation stage includes a parsing, a data transformation, and a standardization step (Elmagarmid et al. [2007]). We will discuss our data preparation steps in section 5.3.

To improve the efficiency of entity resolution, we present a method called filtering which significantly reduces the number of record comparisons. By improving the efficiency of record comparison, filtering can be executed rapidly. We treat filtering as part of the data preprocessing step because the essential entity resolution process is executed on the output of the filtering step. Filtering takes all record pairs as input and outputs the candidate record pairs.

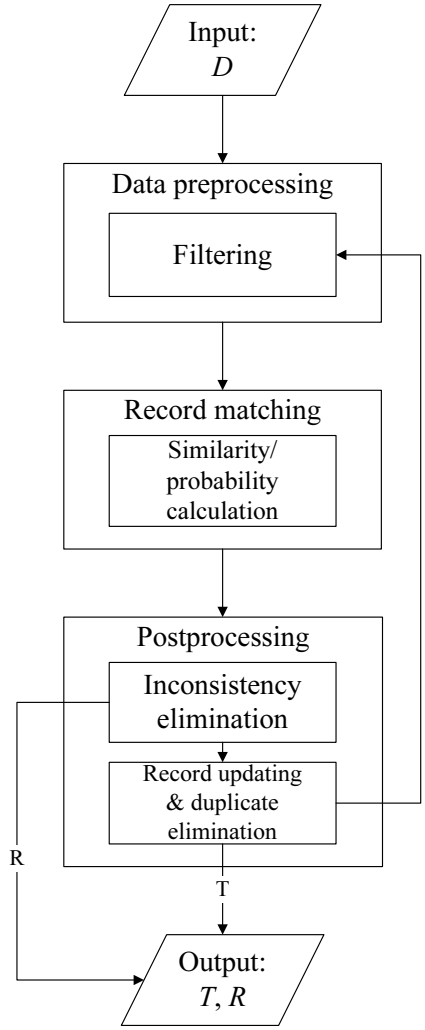


**Figure 5.1:** Framework of ERUDITE

We then perform pairwise record comparisons for all candidate record pairs. In this step, we need to calculate the similarity (for unsupervised learning) or similarity and probability (for supervised learning) for each candidate record pair and then determine whether it is a equivalent record pair or not.

The classification result often has inconsistent decisions. We propose an inconsistency elimination step to make the inconsistent decisions consistent. After this step, we can output the relation set $R$. We also present a record updating step to update some records' attributes to improve the entity resolution accuracy. This can also find the records with "true" attributes which we call equivalent elimination. Furthermore, record updating for unsupervised learning can find a good matching threshold. Inconsistency elimination, record updating, and equivalent elimination constitute the postprocessing steps.

After record updating and equivalent elimination step, we can either output the record set $T$ or go back to the filtering step and repeat the whole precess. Generally one repetition is enough.

To calculate the similarity for each record pair, we present similarity measures which are based on a graph explanation.

## 5.3 Preprocessing

In general, data preprocessing (preparation) improves the quality of the input data and makes the data records more comparable. In chapter 7, we describe several data preprocessing steps for the Cora dataset. In the following subsection we describe Filtering, which we consider as one preprocessing step in ERUDITE. Filtering, an important step in our entity resolution method, takes as input the record pairs and outputs candidate record pairs, which will be processed in the subsequent record matching step.

### *5.3.1 Filtering*

In our method, the similarity matrix has $O(N^2)$ entries where $N$ is the number of records in the dataset. Moreover, our indirect similarity calculation involves common and uncommon neighbor records for each pair of records. Therefore, the complexity of our algorithm grows rapidly with the size of the dataset. To address this problem, we introduce a "filtering" procedure to our entity resolution method. Filtering efficiently selects a subset of record pairs for subsequent similarity computation, ignoring the remaining pairs as highly dissimilar and therefore irrelevant.

**Simple Filtering**

We use two simple filtering strategies: (1) using only part of the attributes that we use for the normal calculation of direct similarity, and (2) only calculating direct similarity and use it to compare with some filtering threshold to decide if a record pair is a candidate record pair for further consideration. The second strategy greatly decreases the calculation time because indirect similarity computation is much more time consuming than direct similarity computation. By improving the efficiency of record comparison, we can rapidly filter out the record pairs which are not likely to be equivalent, so as to reduce the number of record comparisons.

Since the objective of filtering is to filter out most of the dissimilar pairs efficiently, we should choose at most one or two characterizing attributes for the filtering algorithm. However, we must take into account the specific characteristics of the dataset. If a particular attribute is able to characterize records well, then we can use this attribute individually; otherwise, we can combine two attributes. Calculating the similarity based on a single attribute may itself take some nontrivial time. For this situation, it would be preferable to use this attribute individually. Based on these considerations, we propose two filtering algorithms described in algorithm 3 and algorithm 4.

In algorithms 3 and 4, $a_{p_1}$ is record $a$'s $p_1$ attribute, $s_{p_1}$ is the $p_1$ attribute similarity between records $a$ and $b$, and so on. Parameters $t$, $t_1$, and $t_2$ are filter-

**Algorithm 3** Filtering algorithm: use combined attributes

**Input:** All record pairs, two most characterizing attributes $p_1$ and $p_2$
**Output:** Candidate record pairs

```
1: for each record pair (a, b) do
2:     let s = 0
3:     if both a_{p1} and b_{p1} are not empty and at least one of a_{p2} and b_{p2} is empty then
4:         s = s_{p1}
5:     else
6:         if at least one of a_{p1} and b_{p1} is empty and both a_{p2} and b_{p2} are not empty then
7:             s = s_{p2}
8:         else
9:             if all of a_{p1}, b_{p1}, a_{p2}, and b_{p2} are not empty then
10:                s = 0.5 × s_{p1} + 0.5 × s_{p2}
11:            end if
12:        end if
13:    end if
14:    if s ≥ t then
15:        put (a, b) into candidate record pairs set
16:    else
17:        ignore (a, b)
18:    end if
19: end for
```

**Algorithm 4** Filtering algorithm: use attributes individually

**Input:** All record pairs, two most characterizing attributes $p_1$ and $p_2$

**Output:** Candidate record pairs

```
1: for each record pair (a, b) do
2:     if both a_{p1} and b_{p1} are not empty then
3:         if s_{p1} ≥ t_1 then
4:             put (a, b) into candidate record pairs set
5:         else
6:             ignore (a, b)
7:         end if
8:     else
9:         if both a_{p2} and b_{p2} are not empty then
10:            if s_{p2} ≥ t_2 then
11:                put (a, b) into candidate record pairs set
12:            else
13:                ignore (a, b)
14:            end if
15:        end if
16:    end if
17: end for
```

ing thresholds. For unsupervised learning, we generally set $t$, $t_1$, and $t_2$ to 0.5; for supervised learning, we set them to 0.4.

For Cora, for example, by setting $t = 0.5$, without domain-optimized direct similarity calculation (see section 7.2.2), before any record updating, the above algorithm reduces the total candidate record pairs from 1,762,503 to 88,064 (about 95% of the pairs are filtered out), and the number of real equivalent pairs after filtering is 67,781 (out of a total of 72,125), which means that about 94% of the real equivalent pairs are kept. By introducing domain-optimized direct similarity calculation, before any record updating, the above algorithm reduces the total candidate record pairs for Cora dataset to 104,819 (about 94% of the pairs are filtered out), and the number of real equivalent pairs after filtering is 71,548, which means that about 99% of the real equivalent pairs are kept.

For CDDB, by setting $t = 0.5$, without domain-optimized direct similarity calculation, before any record updating, the above algorithm reduces the candidate record pairs from 47653203 to 5941 (nearly 99.99% pairs are filtered out), and the number of real equivalent pairs after filtering is 276 (out of a total of 302), which means that about 91.4% of the real equivalent pairs are kept. Note that we do not use domain-optimized direct similarity calculation for filtering for CDDB because for a big dataset, filtering requires much more time with domain-optimized direct similarity calculation, which is contradictory to the spirit of filtering.

One common method to reduce the number of record comparisons is blocking. Blocking typically refers to the procedure of subdividing records into a set of mutually exclusive subsets (blocks) under the assumption that no matches occur across different blocks. In contrast to blocking that requires hard, non-overlapping partitions, another method uses a simple comparison metric to group records into overlapping clusters called canopies (Elmagarmid et al. [2007]). Our simple filtering method is more similar to the canopy method. After we filter the record pairs, it is possible that for three records $a$, $b$, and $c$, pairs $(a, b)$ and $(a, c)$ are kept as candidate record pairs, while pair $(b, c)$ is filtered out. This amounts to saying that $a$ and $b$ are in the same cluster, $a$ and $c$ are also in the same cluster, while $b$ and $c$ are not; these two clusters overlap and $a$ is in the overlapping part. The difference between our method and canopy is that we do not generate any clusters explicitly for filtering.

Besides the simple filtering method, we also designed another filtering method using the idea of blocking. We call it graph-based filtering.

**Graph-based Filtering**

We now propose a graph-based filtering procedure that uses the inherent graph representation of the dataset and produces a set of (non-overlapping) clusters. Each cluster consists of a collection of record nodes; only node pairs where both nodes belong to the same cluster are considered as candidate equivalent pairs. In other words, inter-cluster pair of nodes are filtered out. Compared to the previous filtering

method, where the candidate pairs is one big set, the candidate pairs after our graph-based filtering is separated into different clusters so that our entity resolution method will be only applied to each cluster locally.

We use the REGGAE graph representation to generate the clusters. Our cluster generation algorithm is described in algorithm 5:

**Algorithm 5** Blocks generation algorithm

1: Randomly pick up a node from the non-allocated nodes as the first node of a new cluster;

2: Add all the non-allocated neighbor nodes of the node into this cluster;

3: For all the new nodes in the cluster, add all their non-allocated neighbor nodes as well into this cluster;

4: Repeat step 3 until there is no new node; mark all nodes in this cluster as allocated;

5: Repeat step 1, 2, 3, 4 until there is no non-allocated node.

In our implementation of this algorithm, we added some constraints and post-processing to generate the clusters of reasonable size. First, for steps 2) and 3), we need the neighbor node to have at least $n$ common property nodes with the current node; Second, we do step 4) only $t$ times; Third, we merge the cluster of size not bigger than $s$ with another bigger cluster. Through experiments, we found that $n = 4$ and $t = 1$ is a good choice for both Cora and CDDB dataset. For Cora, $s$ could be

some value between 4 and 10. Our merging algorithm is given in algorithm 6:

**Algorithm 6** Blocks merging algorithm

1: **for** each cluster $G_i$ which has no more than $s$ nodes **do**

2: **for** each cluster $G_j$ which has more than $s$ nodes **do**

3: **for** all the nodes $n_i$ in $G_i$ **do**

4: calculate $SM_j = \sum_{i=1}^{m} max(cp(n_i, n_k))$, where $m$ is the size of cluster $G_i$, $n_k$ is the $k$th node in $G_j$, and $cp(x, y)$ is the number of common property nodes between node $x$ and $y$

5: **end for**

6: **end for**

7: Choose cluster $G_j$ which has the biggest $SM_j$ as the merging target for cluster $G_i$

8: **end for**

In both filtering methods, the candidate record pairs output by the filtering step will be determined to be equivalent or not in the record matching step which is discussed in next section.

## 5.4 Record matching

We do pair-wise record comparison for all candidate record pairs. We have different record matching methods for unsupervised learning and supervised learning.

Indirect similarity is only used for supervised learning.

### *5.4.1 Unsupervised learning*

In unsupervised learning, we simply compare direct similarity value of each candidate record pair with a matching threshold. If the similarity value is greater than or equal to the threshold, then we classify it as a equivalent pair; else we classify it as a nonequivalent pair. Therefore, for unsupervised learning, using a good matching threshold is important. How to get a good threshold? We can always use a simple threshold say 0.5. However, this may not give the best result (almost impossible), or even a good result. In our framework, our record updating algorithm used for unsupervised learning can find a good matching threshold quickly with the help of only a very small set of training data. When we use 1%, 10%, 50%, or 100% of candidate record pairs as training set, our record updating algorithm always gets almost the same threshold as determined experimentally (to maximize some criterion like F1 measure).

The record matching method for unsupervised learning is as follows:

If there is no training data available, we use 0.5 as matching threshold. Then we compare the similarity of each candidate record pair with matching threshold to determine whether it is equivalent or not.

If training data is available, even if it is very small (say 1% of all candidate record pairs), we first use this training data to experimentally determine a threshold which maximizes the F1 score (over the training set). We then perform record matching followed by record updating. Using the same training data, possibly with some of the records updated, we experimentally determine the new threshold which maximizes F1 score. We use this new threshold as matching threshold to do record matching in all subsequent steps (as if there is no training data). Our experimental results show that this threshold would be the best or nearly the best threshold. Detailed experimental results are presented in chapter 7.

### *5.4.2 Supervised learning*

Instead of using similarity between two records to determine whether they are equivalent (as done in unsupervised learning), in supervised learning we compute the probability that the two records are equivalent. This probability is then used to determine possible duplication (based on a threshold on the probability). Given both direct and indirect similarity values, we use Bayes' theorem to calculate the probability of being equivalent as follows:

$$P(D|S_d, S_i) = \frac{f(S_d, S_i|D) \times p(D)}{f(S_d, S_i)}, \tag{5.1}$$

where $P(D)$ is the probability of being equivalent in training set. To use this equation, we have to assume that the distribution of similarities for training set is similar to that for test set. In equation (5.1), $f(S_d, S_i|D)$ and $f(S_d, S_i)$ are the probability density functions (for all the equivalent pairs from training set and for all the record pairs in training set, respectively) for direct similarity $S_d$ and indirect similarity $S_i$. Note that both $S_d$ and $S_i$ take continuous values between 0 and 1.

To obtain the probability density functions $f(S_d, S_i|D)$ and $f(S_d, S_i)$, we discretize the two-dimensional space of $(S_d, S_i)$ values into a uniform $n \times n$ grid, then determine the probability of obtaining a similarity value pair within each grid cell, and use an interpolation scheme to obtain the probability density function.

Formally, let $(\frac{k}{n} \leq S_d < \frac{k+1}{n}, \frac{l}{n} \leq S_i < \frac{l+1}{n})$, denote the $(k, l)$ grid cell, for $0 \leq k, l \leq n-1$. When either $k = n-1$ or $l = n-1$ (top and right boundary of the region), we can modify the grid cell definition to include the appropriate boundary of the grid as well. Every point in the $(S_d, S_i)$ space belongs to exactly one grid cell.

Let $p_{k,l} = \frac{N_{k,l}}{N}$ denote the number of records $N_{k,l}$ in the training set with similarity values in the $(k, l)$ grid cell, divided by the total number of records $N$. Similar approach is used to compute the counts and probabilities for each grid cell among the equivalents in the training set.

We now describe three different interpolation methods to obtain the density function $f(S_d, S_i)$ from $p_{k,l}$. The density function $f(S_d, S_i|D)$ is obtained in an anal-

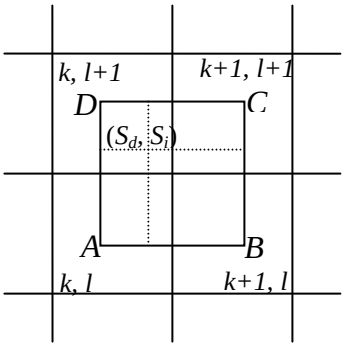


**Figure 5.2:** Illustration of linear interpolation method

ogous manner.

### Piece-wise Constant Interpolation

In the uniform or piece-wise constant interpolation method, the probability density function $f(S_d, S_i)$ takes a constant value within each grid cell. In other words, $f(S_d, S_i) = p_{k,l}$, where $(k, l)$ is the unique grid cell that contains the given point $(S_d, S_i)$.

### Linear Interpolation

As the name suggests, we use a linear interpolation of the $p_{k,l}$ values to obtain $f(S_d, S_i)$. Consider four adjacent grid cells $(k, l)$, $(k+1, l)$ $(k+1, l+1)$, and $(k, l+1)$ depicted in Figure 5.2 and let $A$, $B$, $C$, and $D$ be their respective midpoints as shown. Then the density function $f(S_d, S_i)$ is defined within square $ABCD$ as follows: $f(S_d, S_i) = w_C p_{k,l} + w_D p_{k+1,l} + w_A p_{k+1,l+1} + w_B p_{k,l+1}$, where $w_C = (\frac{k+1}{n} - S_d)(\frac{l+1}{n} - S_i)$,

$w_D = (S_d - \frac{k}{n})(\frac{l+1}{n} - S_i)$, $w_A = (S_d - \frac{k}{n})(S_i - \frac{l}{n})$, and $w_B = (\frac{k+1}{n} - S_d)(S_i - \frac{l}{n})$. Suitable modifications are done along the four boundaries to ensure that the density function is normalized (details omitted).

**Mixture of Gaussians Interpolation**

In the linear interpolation method the calculation of probability density for each point uses no more than four grid points. In the Gaussian interpolation method, we use a "mixture of Gaussians" model which uses the probabilities from all the grid cells to calculate the probability density function. Formally,

$$f(S_d, S_i) = \sum_{k=0}^{n-1} \sum_{l=0}^{n-1} p_{k,l} \times g(S_d - \frac{k+0.5}{n}, S_i - \frac{l+0.5}{n}), \tag{5.2}$$

where $g(x, y)$ is the two-dimensional Gaussian density function:

$$g(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{5.3}$$

and $(\frac{k+0.5}{n}, \frac{l+0.5}{n})$ is the center point of each grid cell. The standard deviation $\sigma$ of the Gaussian function must be chosen carefully; it is usually of the order of the grid cell size. If $\sigma$ is very small, say $\sigma = \frac{1}{10n}$, $g(x, y)$ will be almost an impulse and provide no smoothing. On the contrary, if $\sigma$ is too big, say $\sigma = \frac{5}{n}$ it may lead to over smoothing. We choose the same $\sigma$ for all grid cells, usually half the width of a grid cell.

If we only use direct similarity in equation (5.1), then $p(D|S_d, S_i)$, $f(S_d, S_i|D)$ and $f(S_d, S_i)$ will be replaced by $p(D|S_d)$, $f(S_d|D)$ and $f(S_d)$, respectively, and the two-dimensional space with $n \times n$ grid cells will become a one-dimensional space with

$n$ intervals. The computation of the probability density function is similar to that using both direct and indirect similarities.

Our experiments show that for Cora, $n = 200$ and piece-wise constant interpolation is a good choice; for CDDB, $n = 1000$ and mixture of Gaussians interpolation is a good choice.

### *5.4.3 Improving similarity/probability*

After we calculate similarity (for unsupervised learning) or probability (for supervised learning) values for all candidate record pairs, before we compare them with matching threshold, we can improve them using a "triangle relationship." We have two different methods for this improvement:

*Improve test set by itself*: For any triplet of records $(a, b, c)$, for unsupervised learning, if $S(a, c) < S(a, b) \times S(b, c)$, let $S(a, c) = S(a, b) \times S(b, c)$; for supervised learning, if $P(a, c) < P(a, b) \times P(b, c)$, let $P(a, c) = P(a, b) \times P(b, c)$.

*Improve test set by training set*: For any triplet of records $(a, b, c)$, if pair $(a, c)$ is in test set, pairs $(a, b)$ and $(b, c)$ are in training set, and both $(a, b)$ and $(b, c)$ are equivalent pairs, then set $P(a, c) = 1$.

After each candidate record pair is classified as equivalent or not, we do post-

processing to improve the classification result and merge the equivalent records.

## 5.5 Postprocessing

Our postprocessing step consists of three parts: inconsistency elimination, record updating, and equivalent elimination.

### *5.5.1 Inconsistency elimination*

The classification result often has inconsistent decisions; e.g., records triple $\{a, b, c\}$ such that pair $(a, b)$ and pair $(a, c)$ are classified as equivalent whereas pair $(b, c)$ is not. We call this kind of triple *inconsistent triangle*. For an inconsistent triangle in supervised learning, all three or less than three pairs or triangle edges may belong to the test set. Obviously, if some of edges of an inconsistent triangle is in training set, then we can make use of the known equivalent/nonequivalent label to modify the wrong decision of the other part of edges in test set. If all edges are in test set, then there are more options for possible changes to make the inconsistent triangles consistent. Generally transitive closure can be used for this situation.

**Make Use of training set (MT)**

For supervised learning, when an inconsistent triangle consists of records in both test set and training set, (for example, in one triangle, the edge in training set is nonequivalent, but the other two edges in test set are equivalent) we propose algorithm 7 to correct some labels as well as eliminate inconsistencies.

**Algorithm 7** Inconsistency elimination by making use of training set

**Input:** Classified candidate record pairs
**Output:** Classified candidate record pairs with inconsistency (between training set and test set partly) eliminated

```
1: set all pairs in test set to changeable
2: loop
3:     change = 0
4:     for each record a do
5:         for each record b where (a, b) is in candidate record pairs do
6:             for each record c where (a, c) and (b, c) are in candidate record pairs do
7:                 if one edge x of triangle (a, b, c) is in test set and the other two edges y, z are in training set then
8:                     if x is nonequivalent and changeable while both y and z are equivalent then
9:                         change x to equivalent and non-changeable
10:                        change = change + 1
11:                    else
12:                        if x is equivalent and changeable while one of y, z is equivalent and the other is nonequivalent then
13:                            change x to nonequivalent and non-changeable
14:                            change = change + 1
15:                        else
16:                            report error
17:                        end if
18:                    end if
19:                else
```

**Algorithm 7** Inconsistency elimination by making use of training set (continued)

```
20:                 if two edges x, y of {a, b, c} are in test set and the other edge z
    is in training set then
21:                     if one of x, y is equivalent and the other is nonequivalent,
    and z is equivalent, or if both x and y are equivalent and z is nonequivalent then
22:                         if one of x, y is changeable and the other is non-
    changeable then
23:                             change the changeable one to its opposite
24:                             change = change + 1
25:                         else
26:                             if both x and y are changeable then
27:                                 change the one with probability value closer to
    threshold to its opposite
28:                                 change = change + 1
29:                             else
30:                                 report error
31:                             end if
32:                         end if
33:                     end if
34:                 end if
35:             end if
36:         end for
37:     end for
38: end for
39: if change is the same as in the one in previous loop then
40:     exit
41: end if
42: end loop
```

Note that in an inconsistent triangle, if one or more pairs are in the training set we do not change its/their labels; for pais in the test set, we allow both type of changes — equivalent to nonequivalent and nonequivalent to equivalent. Therefore, the inconsistencies cannot always be eliminated.

**Transitive closure (TC)**

A common way to eliminate the inconsistency in classification result is using transitive closure. We present algorithm 8 to eliminate the inconsistency in classification triangles with all three edges in test set. In this algorithm, $S(x, y)$ is the direct

similarity value between records $x$ and $y$, $P(x, y)$ is the probability of records $x$ and $y$ being equivalent, and $t$ is current matching threshold. Our experiments showed that each time TC can eliminate all the inconsistent triangles with all three edges in test set.

**Algorithm 8** Inconsistency elimination by using transitive closure

**I**nput: Classified candidate record pairs
**O**utput: Classified new candidate record pairs with inconsistency (within test set) completely eliminated

```
1: loop
2:     change = 0
3:     for each record a do
4:         for each record b where pair (a, b) is in candidate record pairs do
5:             for each record c where pair (a, c) is in candidate record pairs do
6:                 if pair (b, c) is in candidate record pairs then
7:                     if pairs (a, b), (a, c) and (b, c) are in test set and two of them
   are equivalent pair and the other one is nonequivalent pair then
8:                         change the nonequivalent one to equivalent pair
9:                         change = change + 1
10:                    end if
11:                else
12:                    if both (a, b) and (a, c) are equivalent pair then
13:                        make (b, c) a equivalent pair
14:                        move (b, c) into candidate record pairs
15:                        S(b, c) = S(c, b) = max(S(b, c), t) (for unsupervised learn-
   ing) or P(b, c) = P(c, b) = t (for supervised learning)
16:                        change = change + 1
17:                    end if
18:                end if
19:            end for
20:        end for
21:    end for
22:    if change is 0 then
23:        exit
24:    end if
25: end loop
```

After applying this algorithm, the inconsistency in test set will be completely removed, but new inconsistent triangles (with edges in both test set and training set) could be generated. Experimental results show that there are not too many new

inconsistent triangles so that we can apply some postprocessing (see following section) to try to eliminate the new inconsistencies.

**General inconsistency elimination (IE)**

MT and TC are special inconsistency elimination methods. In MT, an inconsistent triangle has edges in both training set and test set. While in TC, all three edges of an inconsistent triangle are in test set. When neither MT or TC is applied, the number of inconsistent triangles generally is big. After applying MT and/or TC, this number can be significantly reduced, but the inconsistency often can not be completely eliminated. If MT is applied individually, the inconsistent triangle(s) with all the edges in test set often remains, while if TC is applied individually, the inconsistent triangle(s) with edge in both training and test set often remains. If both MT and TC are used, we have similar situation depending on the order they are applied. Therefore, we propose two general algorithms to eliminate the inconsistencies after we apply MT and/or TC. The first is algorithm 9.

The efficiency of algorithm 9 is not good because for each combination we must check all the triangles involved with the global set $E$ rather than a local group. Therefore, we propose an alternate one which is algorithm 10.

In algorithm 10, for each group, the inconsistent triangles in it can only be caused by the edges in it; therefore, to recount the number of inconsistent triangles for each combination, we just need to check the triangles involved with the edges

**Algorithm 9** Inconsistency elimination algorithm

**Input:** Classified candidate record pairs

**Output:** Classified candidate record pairs with inconsistency (partly) eliminated

1: For all the inconsistent triangles, collect all their edges in test set (let's call it set $E$);

2: Separate $E$ into groups each of size $n$ (e.g., $n = 10$);

3: For each group, try all the $2^n$ possible value combinations (each edge can take two different values: equivalent or nonequivalent) and count the number of inconsistent triangles for each combination;

4: For each group, choose a combination with least number of inconsistent triangle(s).

**Algorithm 10** Improved inconsistency elimination algorithm

**Input:** Classified candidate record pairs

**Output:** Classified candidate record pairs with inconsistency (partly) eliminated

1: Get a bipartite graph which is between inconsistent triangles and edges (in test set);

2: Get all the maximum connected subgraphs of this bipartite graph;

3: Group these subgraphs so that each group has $n$ edges and $n$ is as close as possible to some value $t$, say 10; For the subgraph with more than $t$ edges, we treat it as one group;

4: For each group, if its size is bigger than $t$, separate it into partitions, each of size $t$ and for each partition try all the $2^t$ possible value combinations; for each partition, choose the combination with least number of inconsistent triangle(s); if the group size is no bigger than $t$, try all the $2^t$ possible value combinations, and choose the combination with least number of inconsistent triangle(s).

in this group. As expected, algorithm 10 is much faster than algorithm 9. Our experiments show that its performance of inconsistency elimination is comparable to that of algorithm 9.

Note that in algorithm 10, we can group the subgraphs with different orders: bigger subgraph first, smaller subgraph first, or random. Experimental results show that bigger subgraph first gets best result.

Experimental results also show that for both algorithms 9 and 10, generally all or most of the inconsistencies can be eliminated.

**Nontransitive methods (NT)**

In MT, we try to eliminate inconsistent triangles with records in both test set and training set. In TC, we use transitive closure to eliminate the inconsistent triangles with all records in the test set. Instead of using transitive closure, we can use a method similar to that in MT for the triangles which are inside the test set. The general idea is as follows:

1. instead of allowing only nonequivalent to equivalent changes (as done in transitive closure), allow equivalent to nonequivalent change as well;

2. when more than one edge could be changed in a triangle, choose the one whose probability of being equivalent is closest to the matching threshold;

3. if there is only one edge in some inconsistent triangle which can be changed,

fix it after changing it (i.e., do not modify it in subsequent steps); otherwise do not fix it.

Since NT can change a label both ways, it cannot always eliminate all the inconsistencies.

Another nontransitive method is similar to TC, but instead of changing the nonequivalent edge to equivalent, we change one of the equivalent edges to nonequivalent. Again, we choose the one whose probability of being equivalent is closer to the matching threshold. Similar to TC, this method could eliminate all the inconsistencies. To distinguish these two different nontransitive methods, we call the former one NT1, and the latter one NT2.

**Combination of different methods for inconsistency elimination**

The different inconsistency elimination methods — making use of training set (MT), transitive closure (TC), and general inconsistency elimination (IE) — can sometimes produce contradictory decision to change some classification labels. We can combine them to get a better result. We tried seven different combinations: none, MT, TC, MT+TC, TC+MT, MT+TC+MT, and TC+MT+TC, where "+" means "followed by" and denotes execution order, "none" means neither MT or TC will be applied, and each of which is followed by an IE. Experiments show that none, TC, or TC+MT+TC always result in too many inconsistencies, while the other four combinations do not, although TC+MT is not stable and sometimes results in

too many inconsistencies. The latter four methods show some differences in the number of inconsistencies or in their ability to eliminate inconsistencies. We also tried MT+NT+IE. Detailed experimental results are presented in chapter 7.

We believe the reason different combinations have different inconsistency elimination ability is that different methods/combinations lead to different types of inconsistent triangles. From inspection of the inconsistent triangles in experiments, we see that for MT+TC, most inconsistent triangles consist of one nonequivalent edge in training set, and two equivalent edges in test set; whereas for MT, TC+MT and MT+TC+MT, most of inconsistent triangles consist of two equivalent edges and one nonequivalent edge and all of them are in test set. For MT+NT, most inconsistent triangles consist of one equivalent edge in training set and two other edges in test set with one equivalent and one nonequivalent. Apparently, when all edges of an inconsistent triangle are in test set, this inconsistency is easier to eliminate.

**Correlation clustering**

Correlation clustering is used when the relationship between the objects is known instead of the actual representation of the objects. For example, given a set of record pairs each of which is classified as equivalent or nonequivalent, the task is to cluster the records so that equivalent records are grouped together. So the process of correlation clustering is a little similar to our nontransitive methods. If nonequivalent records are grouped into one cluster then they are changed to be equiv-

alent; if equivalent records are separated into two clusters then they are changed to be nonequivalent. However, the objective of correlation clustering is different than that of our inconsistency elimination algorithms: correlation clustering tries to minimize the number of disagreements (the number of nonequivalent pairs inside clusters plus the number of equivalent pairs between clusters) or maximize the number of agreements (number of equivalent pairs inside clusters plus the number of nonequivalent pairs between clusters) (Bansal et al. [2002]); on the other hand, our inconsistency elimination algorithms try to minimize the number of inconsistent triangles. Although the objectives are different, it appears that correlation clustering could be a good approach to inconsistency elimination for entity resolution. We implemented the ideas from two different algorithms (Bansal et al. [2002] and Ailon et al. [2005]) for correlation clustering problem respectively and compared them with our methods with respect to the ability of inconsistency elimination for entity resolution (see chapter 7).

### *5.5.2 Record updating*

One of the main contributions of our entity resolution framework is that we propose two record updating algorithms to efficiently and effectively improve the classification result and find the records with "true" attributes for both unsupervised

and supervised learning. Also, the record updating algorithm for unsupervised learning can find a good matching threshold. The idea of our record updating algorithms is to make the dataset to have less number of different records by updating some records' attributes.

We present different record updating algorithms for unsupervised learning and supervised learning. Our record updating algorithm for unsupervised learning is presented in algorithm 11, and our record updating algorithm for supervised learning is described in algorithm 12.

In algorithm 11, step 1 to 7 can be repeated for several iterations, but for the last iteration, step 7 will not be executed. Generally one time of record updating is enough.

In algorithm 12, step 1 to 5 can be repeated for several iterations, but for the last iteration, step 5 will not be executed. Generally one time of record updating is enough.

We now discuss some details about these two algorithms:

1. How do we update the attributes of a record based on those of another record (updater record): We set the attributes of the record to be updated the same as those of the updater record. This leads to a more stable procedure as opposed to randomly or selectively updating part of the attributes of the record.

2. For record $r_1$, how to decide the set of records from which to select record $r_2$:

**Algorithm 11** Record updating for unsupervised learning

**Input:** All records in candidate pairs

**Output:** Matching threshold and records with "true" attributes which has equivalent

1: After record matching, for each record $r_1$ in candidate pairs, find a record $r_2$ which has biggest direct similarity $s$ with $r_1$; For matching threshold $t$, if $s \geq t$, go to step 2; else go for next record in candidate pairs;

2: For each of $r_1$ and $r_2$, calculate the average direct similarity $s_a$ for all the candidate record pairs including it, then mark the one with bigger $s_a$ as updater record of the other;

3: After all records in candidate pairs are processed in step 1 and 2, find all the records which does not have updater record but has "updatee" record(s);

4: Sort these records into a list $l$ with descending value of $s_a$;

5: In $l$, if two records $lr_1$ and $lr_2$ are equivalent and $lr_1$'s position is smaller than that of $lr_2$, remove $lr_2$ and add the records of which $lr_2$ is marked as updater record in step 2 into the records of which $lr_1$ is marked as updater record. After step 5 is done, we name the records in $l$ leading records which are the records with "true" attributes having equivalent. Leading records and those records which does not in any candidate record pair are the records with "true" attributes (we also call this kind of records unique records);

6: For each leading record $r$, use it to update all the other records which can be reached from $r$ along the updater transitive relationship (updating link);

7: Redo filtering and recalculate the direct similarity for each candidate record pair. If we do not simply use 0.5 as the matching threshold, we experimentally determine the new threshold as a good matching threshold which maximizes F1 score with the small set of training data.

**Algorithm 12** Record updating for supervised learning

**Input:** All records in candidate pairs

**Output:** Records with "true" attributes which has equivalent

1: Let $C$ be an empty set of clusters: $C = \Phi$. After record matching, for each record $r_1$ in candidate pairs, find a record $r_2$ which has biggest probability $p$ of being equivalent with $r_1$; For some threshold $t$, if $p \geq t$, go to Step 2; else go for next record in candidate pairs;

2: If neither $r_1$ or $r_2$ is in any cluster of $C$, create a new cluster $c$, then add $r_1$ and $r_2$ to $c$ and add $c$ to $C$; else if both $r_1$ and $r_2$ are in cluster $c$ of $C$, do nothing; else if $r_1$ is in cluster $c$ of $C$ but $r_2$ is not in any cluster of $C$, add $r_2$ to $c$; else if $r_1$ is not in any cluster of $C$ but $r_2$ is in cluster $c$, add $r_1$ to $c$;

3: After all records in candidate pair are processed in step 1 and 2, for each cluster in $C$, find its median; Median is the record which has biggest sum of direct similarities with other records in the same cluster. Then sort the clusters according to the descending order of their median's average sum of direct similarities with other records in the same cluster. If two clusters's medians are equivalent, then merge the cluster with lower index in the clusters into the cluster with higher index. For merged new cluster, median should be recalculated. Repeating this process until there are no clusters with the equivalent medians. Median records and those records which does not in any candidate record pair are the records with "true" attributes. We call these clusters updating clusters;

4: For each cluster in $C$, use its median to update other records in the same cluster;

5: Redo filtering and recalculate the probability for each candidate record pair.

We can choose $r_2$ from the set of all candidate records or from the records that have not been chosen as $r_1$ yet (in the current iteration). Experiments show that for Cora, the former option is better, while for CDDB, the latter option is better.

3. How to choose the threshold $t$: Simply speaking, we choose current matching threshold for $t$. We choose the threshold for which we get the best F1 score before we apply current updating. For unsupervised learning, we use 0.5 as threshold or we get a threshold which maximizes F1 score after record updating (with the help of a small training set). See section 5.4.1 for details.

4. For $k$-fold cross validation, after updating and redoing the filtering, we keep the previous candidate record pairs in their original fold. This also makes the updating procedure more stable.

5. How to get the final result: The record updating result converges very fast, and generally only one round of updating is needed to reach a stable result.

6. In algorithm 11, sometimes a record could have two marked updaters, which means this record could be reached from two different leading records. So which leading record should be used to update this record? In practice we sort the leading records list first with a descending order of average direct similarity mentioned in step 2, so the leading record with a bigger average direct similarity

will be used earlier, and we make sure that after a record is updated it will not be updated again by other leading record (in current iteration).

We will use the leading records (for unsupervised learning) or median records (for supervised learning) and those records which do not belong to any candidate record pair as the record with "true" attributes. We call this *equivalent elimination* which will be discussed in next section.

### *5.5.3 Equivalent elimination (Entity identification)*

After we have determined the equivalent/nonequivalent relation for record pairs, we merge the records to eliminate equivalent and find the "true" records.

Generally, the transitive closure could be found during inconsistency elimination which is similar to an updating link in algorithm 11 or updating cluster in algorithm 12. Making use of transitive relationship, in our method we "eliminate" the equivalent records and find the records with "true" attributes after record updating. We use leading records in the leading records list (for unsupervised learning) or median records in the updating clusters (for supervised learning) as the records with "true" attributes for its updating link/cluster and other records in the same updating list/cluster will be regarded as their equivalents.

Also, if a record is not in any candidate record pairs, then it will be treated as

a record with "true" attributes.

It is easy to show that our record updating algorithms can ensure that each record which has an equivalent will be some leading node or be updated by some leading node (proof omitted).

Our experimental results of applying ERUDITE to two public datasets will be presented in section 7.2.

# CHAPTER 6. PREDICTION OF MOVIE RATING

Another classification problem we studied is prediction of movie rating. We used our similarity measure for clustering on a movie dataset to predict user rating of IMDB movies. For each movie in the dataset, we got the user rating from the IMDB website. This was automatically done by our web information retrieval program. We studied prediction based on two versions of the user rating: unrounded and rounded.

## 6.1 Rounding

The user rating from IMDB website is a real number between 1.0 and 10.0. We rounded it to an integer between 1.0 and 5.0 as follows:

**Table 6.1:** IMDB movie rating rounding method

$$
\begin{aligned}
[1.0, 3.0) &\rightarrow 1.0 \\
[3.0, 5.0) &\rightarrow 2.0 \\
[5.0, 7.0) &\rightarrow 3.0 \\
[7.0, 8.6) &\rightarrow 4.0 \\
[8.6, 10.0] &\rightarrow 5.0
\end{aligned}
$$

## 6.2 $K$-Nearest-Neighbor Method

The simplest way to make prediction of the user rating is using the average rating of all other movies' ratings. Apparently, this method generally can not get satisfying result. In chapter 4, when we calculate $CoS$, we need to find the most similar movies for each movie first. So we use $K$-Nearest-Neighbor classification as another method which is simple but can provide better results. We tried different values for $K$: 5, 10, 15 and 20, and we tried different methods to combine the rating of each neighbor movie to generate the predicted rating:

1. **weighted**: For the sorted similar movies list (the most similar one first), each movie rating has a weight which is $q$ times the weight of the next movie in the list (exponential weighting). We tried different values for $q$: 1.0, 1.1, 1.2, 1.3, 1.4 and 1.5. Another weighting option is to use the similarity value of each similar movie as the weight (round them to make sure all of them sum to 1).

2. **most frequent category (majority voting)**: For the $K$ neighbor movies' categories, we choose the most frequently occurring one.

3. **median**: We choose the median category (since they are numbers which can be sorted) as the predicted value.

We use $n$-fold cross validation to calculate the RMSE (root mean squared error) of classification, where $n$ is the number of movies in the dataset. This means each movie is used as target movie one time and used as training movie $n - 1$ times. Detailed experimental results will be discussed in section 7.3.

# CHAPTER 7. EXPERIMENTAL RESULTS

In this chapter, we present our experimental results for clustering, entity resolution and movie rating prediction. In the beginning of each section, we will introduce the dataset(s) that we used. All the experiments were run on a PC with AMD Phenom II 2.9G CPU with 3.25G memory.

## 7.1 Clustering

We tested the $K$-medoid and fuzzy $K$-medoid clustering methods, with different number of initial clusters and different set of weights for the similarity measures. In each case, we calculated the validity indices. For the fuzzy $K$-medoid method, we set $p = 2$ as is usually done in the literature. For $K$-medoid, we also tested sampling when calculating $C_a$ and $S_a$.

We tested the interpretation of our clustering result for *Director* attribute without using it in our similarity measure. We use both attribute entropy and cluster entropy to evaluate how good the interpretation is.

We compared our clustering method with other existing clustering methods, especially graph clustering methods. We used the same datasets and evaluation measures that they used for the comparison.

### *7.1.1 Datasets*

**IMDB Movies**: The IMDB movie dataset (Imd) has more than 480K movies; we generally used a subset of it. The experiments reported here were done using the movies released from year 1991 to year 2005; we used movies with a certain minimum number of actors, actresses etc. to obtain our test dataset of 2571 movies. Its attributes include *Title*, *Name*, *Position* (*Actor*, *Director*, etc.), and *Role*. In our experiments, we used two kinds of attributes set: One is *Director*, *OtherPeople*, *Role*; The other is *Director*, *Actor-Role*, *OtherPeople*. *OtherPeople* includes all the people in that movie except Roles which actors/actresses played. *Actor-Role* means actor-role pairs. For indirect similarity, we use neighbor movies.

**Enron Emails**: We used a subset of Enron email dataset Enr which has 1000 emails. Its attributes include *from*, *to*, *message*, *date*. For indirect similarity, we use neighbor emails.

**Political Blogs**: It is a directed network of hyperlinks between 1490 weblogs on US politics (Pol). We use its *Value* attribute which indicates political leaning (0: left or liberal, 1: right or conservative) and the hyperlinks. Because *Value* is a single-value attribute, indirect similarity is not applicable in this case. The in-link and out-link were considered as two attributes in the similarity measure. The node move algorithm was applied on the clustering result.

**Cora Paper Citation Network**: It is a paper citation network from Cor with about 17604 papers. The citations for each paper (direct neighbors) were used as the only attribute in the similarity measure. The node move algorithm was applied on the clustering result.

**Epinions Social Network**: It is who-trust-whom online social network of a general consumer review site Epinions.com. Members of the site can decide whether to "trust" each other. All the trust relationships interact and form the Web of Trust which was then combined with review ratings to determine which reviews are shown to the user (Epi). There were 75877 nodes and 405739 edges in the dataset we used. The direct neighbors of each member were used as the only attribute in the similarity measure. The node move algorithm was applied on the clustering result.

**Dolphin Social Network**: It is an undirected social network of frequent associations between 62 dolphins in a community living off Doubtful Sound, New Zealand (Dol). The direct neighbors of each dolphin were used as the only attribute in the similarity measure. The node move algorithm was applied on the clustering result.

**Condensed Matter Collaboration Network**: It is a weighted network of coauthorships between scientists posting preprints on the Condensed Matter E-Print Archive between Jan 1, 1995 and December 31, 1999 (Con). There are 16726 authors in this network. The direct neighbors of each author were used as the only attribute in the similarity measure. The node move algorithm was applied on the clustering

result.

### *7.1.2 Results of K-medoid Clustering*

For IMDB movie dataset, for direct similarity, we used the following properties: *Director*, *Actor*, *Actress*, *Role*, *Producer*, *Editor*, *Writer*, *Composer*, and *Cinematographer*. For simplicity, we group *Actor*, *Actress*, *Producer*, *Editor*, *Writer*, *Composer* and *Cinematographer* together into one property called *OtherPeople*. Thus we have three properties determining direct similarity, which together with the indirect similarity based on common neighbor movie, gives us four components in our similarity measure: *Director*, *OtherPeople*, *Role*, and *NeighborMovie*. We tried six different weight compositions for them: 1-0-0-0, 0-1-0-0, 0-0-1-0, 0-0-0-1, 0-0.9-0-0.1 and 0.1-0.8-0-0.1. Note that we consider *director* as a separate component since he/she plays a very important role in the style of a movie.

Figures 7.1 – 7.3 depict our validity indices — $CS$, $RoM$, and $CoS$, respectively — for different initial cluster numbers and similarity measures.

In Table 7.1, we summarize the $CoS$ measure by averaging the $CoS$ value over all iterations. We do this separately for the medoid nodes and all nodes, since we expect a higher value of $CoS$ for the medoid nodes (which are usually at the "cluster center"). In the left four columns of the table, we average $CoS$ over all similarity

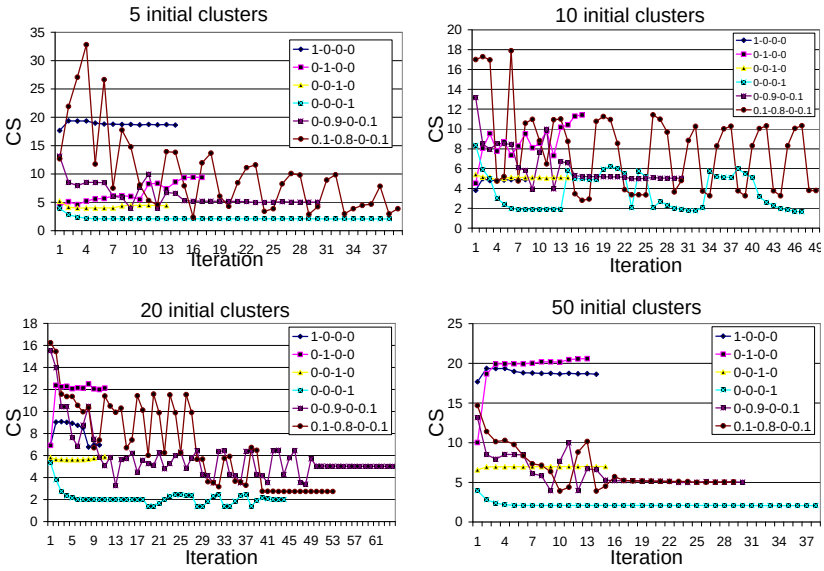


**Figure 7.1:** Initial cluster number vs. similarity measure vs. *CS*

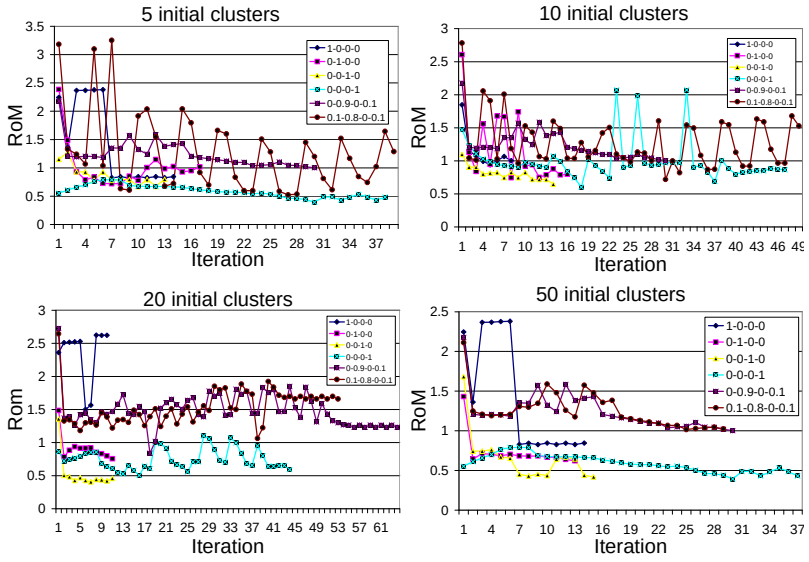


**Figure 7.2:** Initial cluster number vs. similarity measure vs. *RoM*

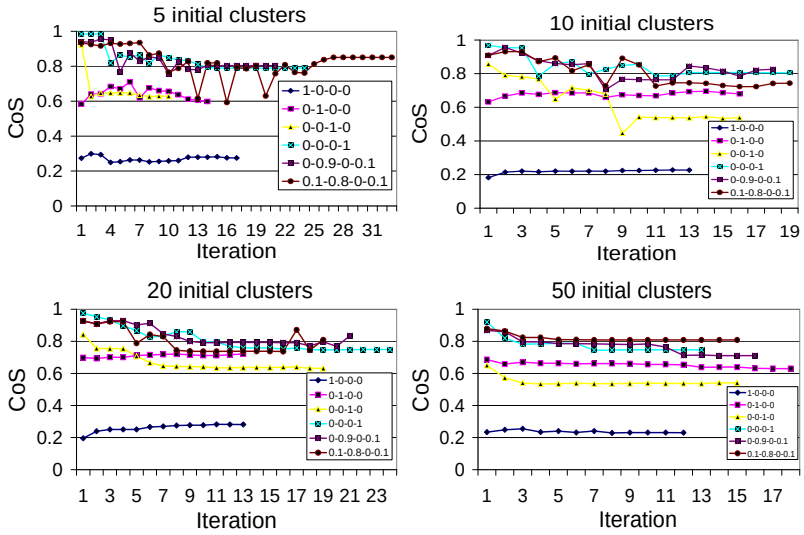


**Figure 7.3:** Initial cluster number vs. similarity measure vs. $CoS$

measure weights combinations and report this for different initial number of clusters, whereas in the right six columns of the table, we average $CoS$ over all initial number of clusters and report this for different similarity measure weights combinations. In the former case, we leave out the weight combination 1-0-0-0 while averaging for all nodes, since it has a much lower $CoS$ value as seen from the fifth column of the table. We also calculated the average for the maximum (over all iterations) $CoS$ value. For example, the $CoS$ average (over all variations of similarity measures) maximum value for K-medoid method is 0.90, 0.88, 0.88, 0.80, for 5, 10, 20, and 50 initial cluster numbers, respectively. The average maximum $CoS$ value can be used as a reasonable stopping criterion for the clustering process. In addition, we calculated

the weak $CoS$ value for each case; this represents the fraction of the top 5 (within same cluster) similar nodes to a given node that belong to the top 10 (within the entire dataset) similar nodes. Naturally, weak $CoS$ must be greater than strong $CoS$ (which are reported in Table 7.1) but less than 1. The weak $CoS$ values were 12% to 43% larger than strong $CoS$ values for all nodes; for the medoids, the increment was 4% to 25% over the corresponding strong $CoS$ values. It is evident from table 7.1 that $CoS$ has a bias towards smaller number of clusters. Moreover, we can see from the right three columns that indirect similarity helps improve $CoS$ value. Fuzzy version of the $K$-medoid algorithm performs similar to the $K$-medoid algorithm, in terms of $CoS$ value.

**Table 7.1:** Average $CoS$ values

| | | Initial cluster number | | | | Variation of similarity metric | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 5 | 10 | 20 | 50 | 1-0-0-0 | 0-1-0-0 | 0-0-1-0 | 0-0-0-1 | 0-0.9-0-0.1 | 0.1-0.8-0-0.1 |
| *K*-m | All nodes | 0.76 | 0.76 | 0.76 | 0.71 | 0.25 | 0.67 | 0.63 | 0.81 | 0.82 | 0.81 |
| | Medoids | 0.88 | 0.82 | 0.82 | 0.69 | 0.71 | 0.89 | 0.68 | 0.72 | 0.90 | 0.92 |
| Fuzzy | All nodes | 0.75 | 0.71 | 0.72 | 0.67 | 0.23 | 0.67 | 0.51 | 0.79 | 0.79 | 0.81 |
| | Medoids | 0.87 | 0.84 | 0.80 | 0.71 | 0.70 | 0.88 | 0.69 | 0.70 | 0.93 | 0.93 |

We studied the influence of postprocessing steps on the final number of clusters. Recall that postprocessing includes split and merge steps and each results in a change in the number of clusters. Figure 7.4 shows the variation of cluster number with iteration for different similarity measure weights combination in $K$-medoid clustering. Four different initial number of clusters — 5, 10, 20, and 50 — were used.

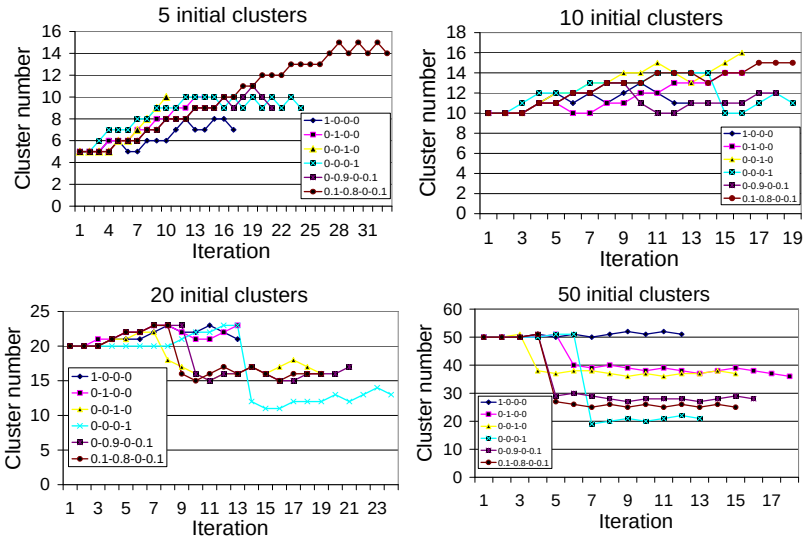


**Figure 7.4:** Cluster numbers vs. Iteration for different similarity measures and initial cluster numbers: $K$-medoid clustering

Figure 7.5 illustrates Theorem 4.1.2: when $\frac{C'_a}{C_a} \geq \frac{k_2(n-k_1)}{k_1(n-k_2)}$ (dark blue point (in line 1) is higher than red point (in line 2)) — iteration 1, 3, 6 and 9 — we have $\frac{S'_a}{S_a} \leq \frac{k_2(k_1-1)}{k_1(k_2-1)}$ (yellow point (in line 3) is lower than light blue point (in line 4)). Initial cluster number is 5 and similarity measure is 1-0-0-0 (for this measure clusters or roughly of equal size).

Figure 7.6 compares the clustering quality (with respect to $CS$ and $RoM$) by treating *Actor* and *Role* in pairs with those obtained by treating them individually. Different subsets of the original dataset were used; their sizes are as indicated in the legend.

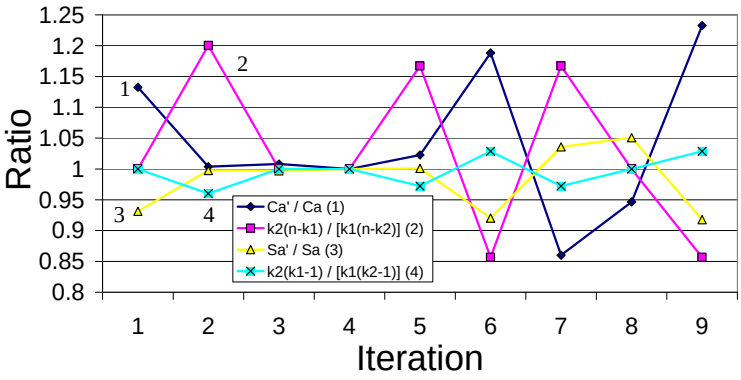


**Figure 7.5:** Illustration of Theorem 4.1.2

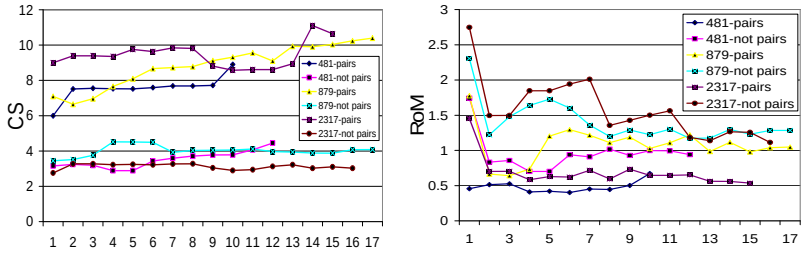


**Figure 7.6:** Effect of Actor-Role pairs on cluster validity indices

Figure 7.7 shows clustering results for Enron dataset. Initial cluster number is 5. Similarity measure is weighted combination for From&To, Message, Date, and CommonEmail.

To reduce computational complexity, we use a small sample of the nodes in each cluster to compute the cohesion $C_a$ and separation $S_a$. In each case, we select a fraction (expressed as a sampling percentage) of the nodes from each cluster, using different sampling strategies. In one strategy we simply perform a uniform random

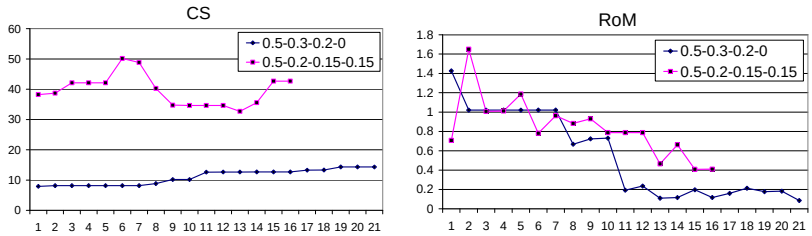


**Figure 7.7:** Clustering result for Enron dataset

sampling over each cluster (designated "random"). Our second sampling strategy was based on the similarity space for each cluster. For each cluster, we first choose either the medoid or a border node (node in cluster most dissimilar to medoid) as starting node (we designate this strategy as "medoid" or "border", respectively). We then select nodes based on its similarity with the starting node (uniform sampling in similarity space). Finally, we used a "mixture" strategy where we pick up the medoid, a border node and one other node as starting node and take the average over the three results. Figure 7.8 shows the relative root-mean-squared-error (RMSE) of $Ca$ and $Sa$ due to sampling. We present results for ten initial clusters and similarity measure weights 0-1-0-0; results are similar for other measures / cluster numbers. For each sampling percentage we report the average and standard deviation over ten runs (top and bottom row, respectively, of Figure 7.8).

We compute the similarity between the clustering result obtained with and without sampling based on the overlap between corresponding clusters. Figure 7.9

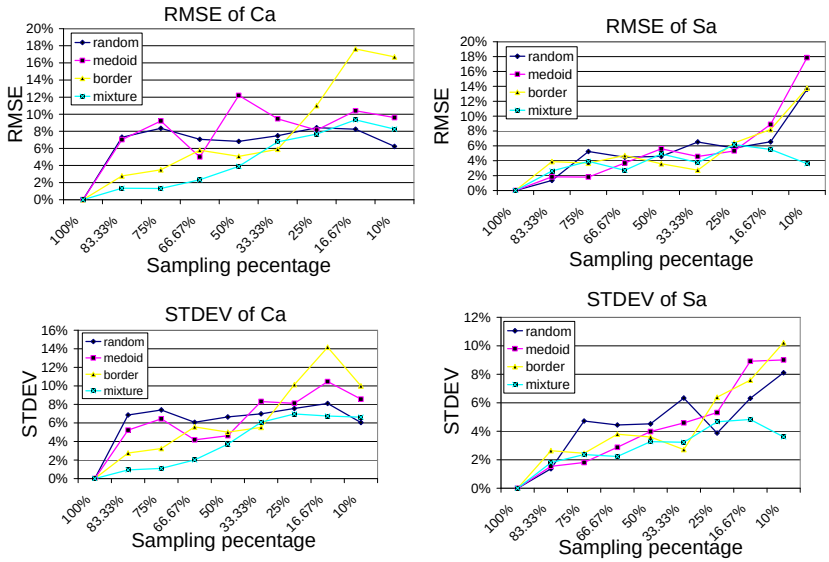


**Figure 7.8:** Sampling percentage vs. sampling strategy vs. sampling error

shows the similarity for ten initial clusters and similarity measure weights 0-1-0-0.

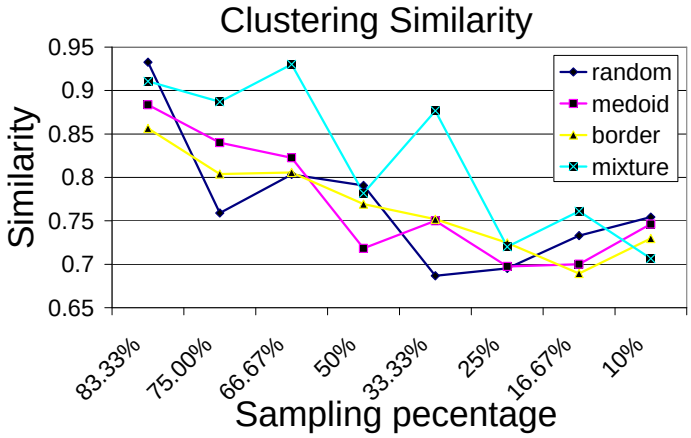


**Figure 7.9:** Sampling percentage vs. sampling strategy vs. clustering similarity

In our experiments, we observed that clustering time for 50% sampling is 30-60%

of that for 100% sampling (i.e. no sampling).

**Discussion**

For index $CS$, we can see from Figure 7.1 that it increases with iteration for weights 0-1-0-0 and 0-0-1-0; it also increases with initial cluster number for the same weights, especially for 0-1-0-0. We generally use the clustering result from the iteration which has the best $CS$ value; therefore from the perspective of peak $CS$ value, weights 0.1-0.8-0-0.1 is the best for 5 and 10 initial clusters, whereas weight 0-1-0-0 is the best for higher number of initial clusters. From Figure 7.2, the index $RoM$ is small for weights 0-1-0-0, 0-0-1-0, and 0-0-0-1, which means that the medoids are good cluster representatives. We also note that overall $RoM$ decreases with iteration, except for 20 initial clusters. From Figure 7.3, we note that $CoS$ is better when we use common neighbor movies in the similarity measure (last component non-zero). Similar results were obtained with 20 or 50 initial clusters. This indicates that inclusion of indirect similarity improves the similarity concentration quality ($CoS$ measure) of the clustering. We can see that both $CS$ and $CoS$ reach a high value (e.g., 10 for $CS$ and 0.8 for $Cos$), which indicates a good clustering quality.

In Figure 7.4 we observe that the number of clusters increases with postprocessing for 5 initial clusters and for some similarity measures with 10 initial clusters. Number of clusters increases first and then decreases for the case of 20 initial clusters and for some similarity measures with 10 initial clusters. For 50 initial clusters, we

see a decrease in the number of clusters for all cases (except one). Basically, we can see that 10∼20 would be a proper number of clusters for this dataset. Similar trend was observed for fuzzy $K$-medoid clustering which is not presented here.

Figure 7.6 shows that the clustering quality improves significantly by considering *Actor* and *Role* in pairs.

Figure 7.7 shows that common neighbor record component in similarity measure helps a lot for $CS$ for Enron dataset.

Overall, sampling error decreases with sampling percentage, which is to be expected. According to Figure 7.8, "random" and "border" sampling strategy is better for $C_a$, whereas for $S_a$ medoid sampling strategy is better. Although the mixture strategy looks the best for all the cases, we generally cannot use it because it is an average over three strategies and hence has added complexity. We present it here mainly for comparison. From Figure 7.9, overall clustering similarity decreases with sampling percentage, as expected. The similarity is quite high for all strategies.

### *7.1.3 Results of Multi-Medoid K-medoid Clustering and Interpretation for Director Attribute*

We present the results for these two in one section because we also want to see whether multi-medoid $K$-medoid clustering can get better interpretation of the clustering result with domain independent similarity measure. For this section, the

dataset we used is the IMDB movies directed by top 10 directors (ranked by the number of movies they directed) in year 1987 to 2006. Only the movie whose sum of numbers of director, other people and roles is no less than 10 were considered. It has 1576 movies. Because *Director* could not be used in the similarity measure, 1-0-0-0 weights combination was not used. Besides $CS$ and $RoM$, we used several more evaluation measures to evaluate how good the interpretation for *Director* is. Note that the "true label" for each record (*Director* name) is known.

**Cluster Entropy**: Boley [1998] introduced an information entropy approach to evaluate the quality of a set of clusters according to the original class labels of the data points (He et al. [2003]). For each cluster $c_i$, a cluster entropy $E_{c_i}$ is computed as

$$E_{c_i} = -\sum_j \frac{n(l_j, c_i)}{n(c_i)} \log \frac{n(l_j, c_i)}{n(c_i)}, \tag{7.1}$$

where $n(l_j, c_i)$ is the number of the samples in cluster $c_i$ with a predefined label $l_j$ and $n(c_i) = \sum_j n(l_j, c_i)$ is the number of samples in cluster $c_i$. We let $\log 0 = 0$ in equation (7.1). The overall cluster entropy $E_c$ is then given by a weighted sum of individual cluster entropies as

$$E_c = \frac{1}{\sum_i n(c_i)} \sum_i n(c_i) E_{c_i}. \tag{7.2}$$

The cluster entropy reflects the quality of individual clusters in terms of homogeneity of the data points in a cluster (a smaller value indicates a higher homogeneity,

and 0 means perfect). A drawback of cluster entropy is that it has a bias towards bigger number of clusters. To counter this deficiency, He et al. [2003] used another entropy measure called class entropy to measure how data points of the same class are represented by the various clusters created.

**Class Entropy**: For each class $l_j$, a class entropy $E_{l_j}$ is computed as

$$E_{l_j} = -\sum_i \frac{n(l_j, c_i)}{n(l_j)} \log \frac{n(l_j, c_i)}{n(l_j)}, \tag{7.3}$$

where $n(l_j, ci)$ is the number of samples in cluster $c_i$ with a predefined label $l_j$ and $n(l_j) = \sum_i n(l_j, c_i)$ is the number of the samples with class label $l_j$. The overall class entropy $E_l$ is then given by a weighted sum of individual class entropies as

$$E_l = \frac{1}{\sum_j n(l_j)} \sum_j n(l_j) E_{l_j}. \tag{7.4}$$

Opposite to cluster entropy, class entropy has a bias towards less number of clusters.

**Overall Entropy**: To overcome the drawbacks of cluster entropy and class entropy, He et al. [2003] defined a combined overall entropy measure:

$$E_{cl} = \alpha E_c + (1 - \alpha) E_l, \tag{7.5}$$

where $\alpha \in [0, 1]$ is the weight that balances the two measures. In our experiments, we set $\alpha = 0.5$.

**Cluster Purity** : The purity of a cluster represents the fraction of the cluster corresponding to the largest class of data points assigned to that cluster. Thus, the

purity of the cluster $c_i$ is defined as

$$P_{c_i} = \frac{1}{n(c_i)} \max_j n(l_j, c_i). \tag{7.6}$$

The overall cluster purity $P_c$ is then given by a weighted sum of individual cluster purities as

$$P_c = \frac{1}{\sum_i n(c_i)} \sum_i n(c_i) P_{c_i}. \tag{7.7}$$

**Class Purity**: Similar to class entropy, we define a measure called class purity as

$$P_{l_j} = \frac{1}{n(l_j)} \max_i n(l_j, c_i). \tag{7.8}$$

The overall class purity $P_l$ is then given by a weighted sum of individual class purities as

$$P_l = \frac{1}{\sum_j n(l_j)} \sum_j n(l_j) P_{l_j}. \tag{7.9}$$

**Overall Purity**: Cluster purity and class purity have drawbacks similar to cluster entropy and class entropy, respectively. Similar to overall entropy, we define a combined overall purity measure:

$$P_{cl} = \alpha P_c + (1 - \alpha) P_l, \tag{7.10}$$

where $\alpha \in [0, 1]$ is the weight that balances the two measures. In our experiments, we set $\alpha = 0.5$.

Note that entropy takes non-negative values, while purity takes values between 0 and 1 (inclusive). For entropy, we prefer smaller values, while for purity, we prefer bigger values.

Figure 7.10 shows the different entropies and purities vs. different weights combinations in similarity measure vs. number of medoid in each cluster.

In figure 7.10, "1-entropy" means it is entropy for the clustering result using $K$-medoid clustering, and "10-entropy" means it is entropy for the clustering result using multi-medoid $K$-medoid clustering (similarly for purity). "even" means it is the result for evenly distributed clustering. figure 7.10 shows that we got much better results than those with "even" distribution, except for 0-0-1-0 similarity measure, in which only *Role* was used for the calculation of similarities. It makes sense because it is not feasible to separate movies only based on roles. Figure 7.10 also shows that in most cases multi-medoid $K$-medoid (10 medoids in each cluster) performs better than $K$-medoid. Moreover, when each cluster has one medoid, indirect similarity improves cluster entropy, and also improves both class entropy and class purity greatly.

Another way we could evaluate the interpretation of our clustering result for *Director* attribute is to compare the validity indices values between our clustering results and "perfect" clusterings. The so-called "perfect" clustering means each cluster only contains the movies directed by some director. We used top 10 directors, so we had 10 clusters for "perfect" clustering. Note that the "perfect" clustering does

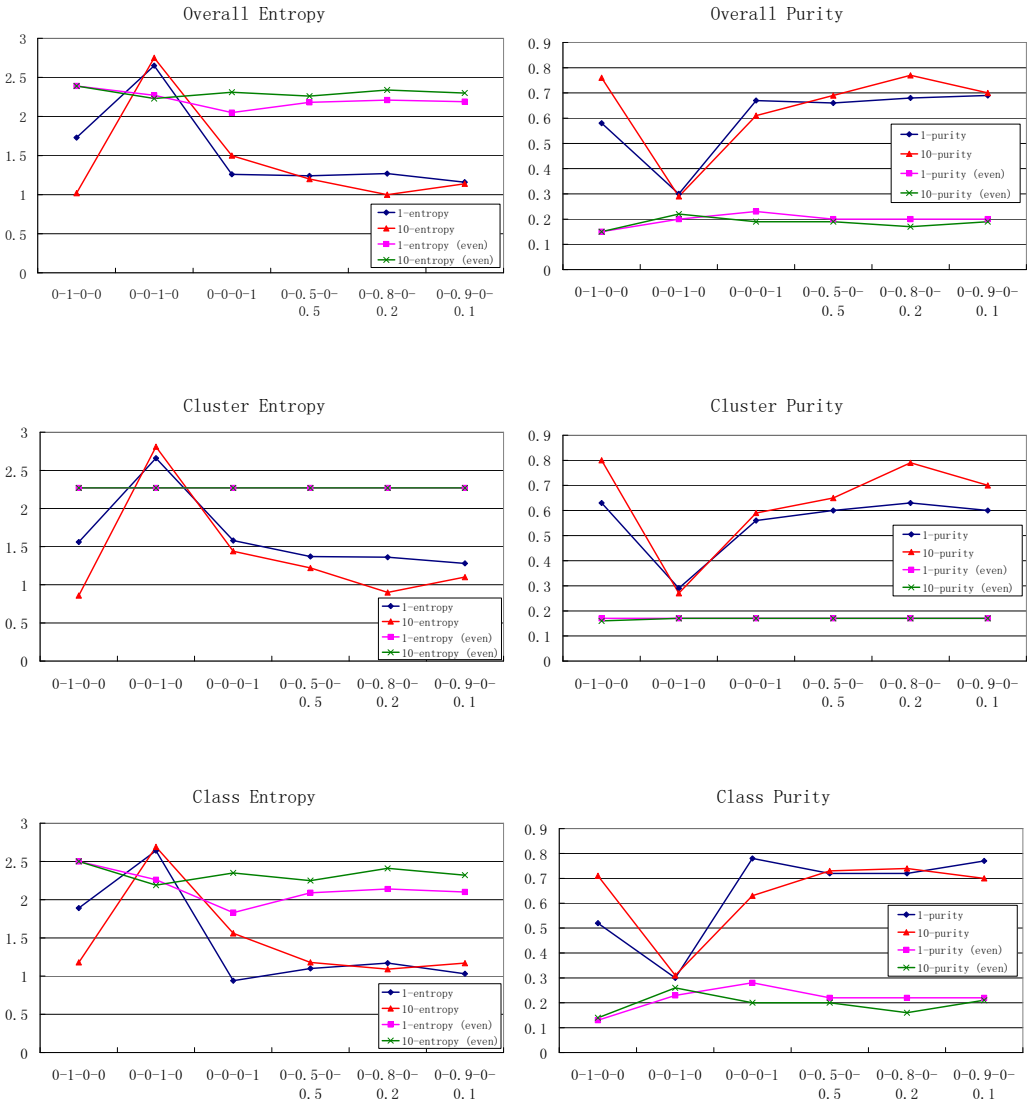


**Figure 7.10:** Entropy/Purity vs. similarity measures vs. number of medoid in each cluster

not necessarily have the best validity indices values unless using 1-0-0-0 similarity measure. But if we get a clustering whose validity indices values are close to those of "perfect" clustering, it could mean that this clustering has a good interpretation for

**Table 7.2:** Clustering result compared with "perfect" clustering for movies from year 1987 to 2006

| | **0-1-0-0** | | | | **0-0-1-0** | | | | **0-0-0-1** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | perfect | | our | | perfect | | our | | perfect | | our | |
| | m1 | m10 | m1 | m10 | m1 | m10 | m1 | m10 | m1 | m10 | m1 | m10 |
| *CS* | 39.63 | 39.63 | 13.27 | 22.90 | 22.07 | 22.07 | 98.68 | 72.69 | 4.90 | 4.90 | 2.87 | 3.02 |
| *RoM* | 0.54 | 0.65 | 0.30 | 0.30 | 1.99 | 2.02 | 2.44 | 1.56 | 0.61 | 0.57 | 0.40 | 0.35 |

*Director*.

Table 7.2 compares the perfect clustering with our clustering results for different similarity measures and number of medoid in each cluster, in terms of validity indices. "m1" means that each cluster has one medoid, and "m10" means that each cluster has 10 medoids.

Table 7.3 shows the same thing as table 7.2, except that it only used IMDB movies from year 1991 to 1995. It has 458 movies.

Table 7.2 and 7.3 show that in terms of validity indices, our clustering results are not very close to the perfect clustering. One positive result is that in most cases multi-medoid $K$-medoid gets better results than $K$-medoid in terms of validity indices. In table 7.3, multi-medoid method is better or not worse for all the results.

Figure 7.11 shows the validity indices values for different number of medoids

**Table 7.3:** Clustering result compared with "perfect" clustering for movies fromo year 1991 to 1995

| | **0-1-0-0** | | | | **0-0-1-0** | | | | **0-0-0-1** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | perfect | | our | | perfect | | our | | perfect | | our | |
| | m1 | m10 | m1 | m10 | m1 | m10 | m1 | m10 | m1 | m10 | m1 | m10 |
| $CS$ | 4.37 | 4.37 | 2.53 | 3.34 | 19.62 | 19.62 | 4.48 | 1868.56 | 1.43 | 1.43 | 0.89 | 1.03 |
| $RoM$ | 0.87 | 0.73 | 0.41 | 0.12 | 3.00 | 1.96 | 3.30 | 1.75 | 0.62 | 0.60 | 0.33 | 0.20 |

in each cluster and different similarity measures. We can see that for each similarity measure, overall, $CS$ value increases and $RoM$ value decreases as the number of medoids in each cluster increases, which is favored. For similarity measure 0-0-1-0 and 0-0-0-1, we have similar trends for $CS$ and $RoM$, but not as good as other similar measures.

Table 7.4 compares the running time of our $K$-medoid algorithm on IMDB movie dataset, for different number of medoid in each cluster. In the table, #medoid refers to the number of medoid in each cluster.

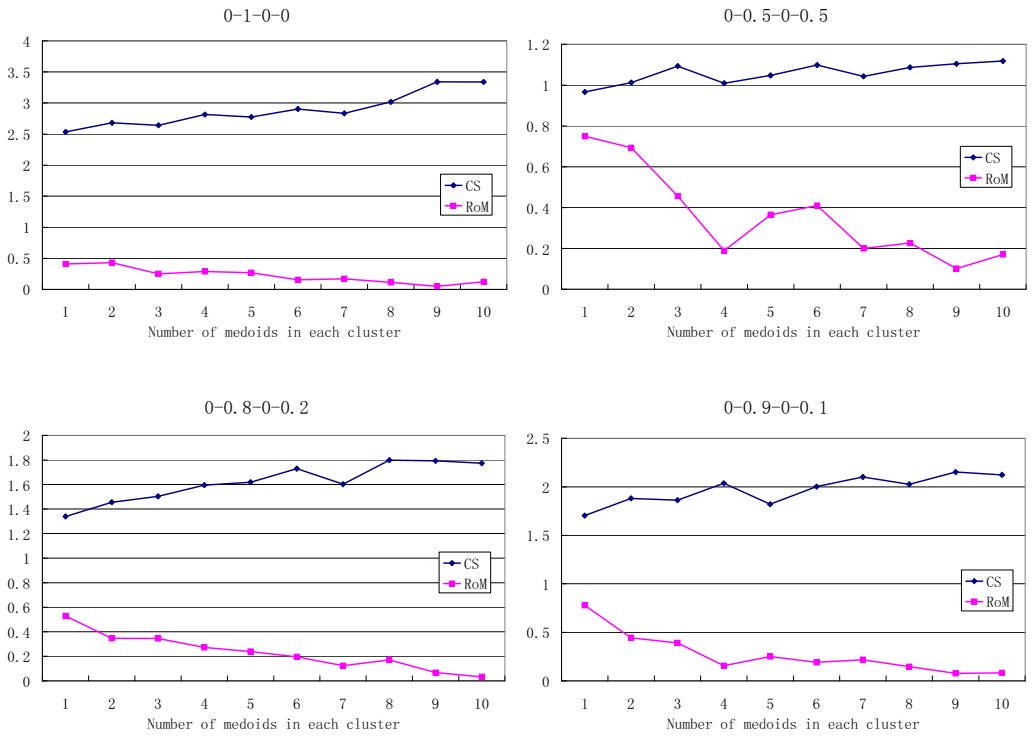


**Figure 7.11:** Validity indices vs. similarity measures vs. number of medoid in each cluster

### *7.1.4 Results of Comparison with Other Existing Graph-Based Clustering Methods*

Our clustering algorithms are designed to cluster unstructured or semi-structured data with categorical attributes. Although the $k$-mediod based algorithm itself is not graph-based, our similarity measure used in the $k$-mediod method has a graph-based explanation. Experimental results indicate that our $K$-medoid clustering algorithm also works well on graph style datasets, combined with our node move algorithm.

**Table 7.4:** Running time (second) of our $K$-medoid algorithm vs. number of medoid in each cluster

| #Medoid | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Time | 10 | 11 | 11 | 14 | 17 | 17 | 13 | 26 | 26 | 30 |

To compare with other existing graph clustering methods, we introduce three more evaluation measures here.

**Density** Zhou et al. [2009]: It is the ratio of the number of intro-cluster (original) edges to the number of all edges. It is defined as

$$density(\{c_i\}_{i=1}^{k}) = \frac{1}{|E|}\sum_{i=1}^{k}|(v_p, v_q)|v_p, v_q \in c_i, (v_p, v_q) \in E|, \tag{7.11}$$

where $c_i$ means $i$th cluster, and $E$ is edge set.

**Normalized Cut** Satuluri and Parthasarathy [2009]: The normalized cut (N-Cut) of a cluster $c$ in the graph $G$ is defined as

$$Ncut(c) = \frac{\sum_{v_i \in c, v_j \notin c} A(i,j)}{\sum_{v_i \in c} degree(v_i)}, \tag{7.12}$$

where $A$ is the adjacency matrix of the graph, $degree(v_i)$ is number of direct neighbors of node $v_i$. The normalized cut of a cluster is simply the number of edges that cut across this cluster and the rest of the graph, normalized by the total degree of all the nodes in the cluster. The average normalized cut of a clustering $C$ is the average of the normalized cuts of each of the constituent clusters, and lies between 0 and 1 —

smaller the average normalized cut, better is the clustering result. Average N-Cut score allows us to compare clustering results with different numbers of clusters.

$$AverageNcut(C) = \sum_{i=1}^{k} Ncut(c_i). \tag{7.13}$$

**Modularity** Newman and Girvan [2004]: Modularity is defined as

$$Q = \mathrm{Tr}\,\mathbf{e} - ||\mathbf{e}^2||, \tag{7.14}$$

where $\mathbf{e}$ is a $k \times k$ symmetric matrix $\mathbf{e}$ whose element $e_{ij}$ is the fraction of all edges in the network that link vertices in community $i$ to vertices in community $j$ (the network is divided to $k$ communities). $\mathrm{Tr}\,\mathbf{e} = \sum_i e_{ii}$ is the trace of this matrix which gives the fraction of edges in the network that connect vertices in the same community. $||\mathbf{x}||$ indicates the sum of the elements of the matrix $\mathbf{x}$. Modularity measures the fraction of the edges in the network that connect vertices of the same type (i.e., within-community edges) minus the expected value of the same quantity in a network with the same community divisions but random connections between the vertices. If the number of within-community edges is no better than random, we will get value 0. Values approaching 1, which is the maximum, indicate strong community structure. In practice, values for such networks typically fall in the range from about 0.3 to 0.7. Higher values are rare.

Table 7.5 compares our clustering result with the result in Zhou et al. [2009] for political blogs dataset. "# Cluster" means number of clusters. "m." means our node

**Table 7.5:** Clustering result compared with SA-Cluster in Zhou et al. [2009] for political blogs

| # Cluster | 3 | | | 5 | | | 7 | | | 9 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | Our method | | Zhou | Our method | | Zhou | Our method | | Zhou | Our method | | Zhou |
| | w/o m. | w/ m. | | w/o m. | w/ m. | | w/o m. | w/ m. | | w/o m. | w/ m. | |
| Density | 0.86 | 0.90 | 0.85 | 0.86 | 0.90 | 0.90 | 0.86 | 0.90 | 0.81 | 0.85 | 0.90 | 0.59 |
| Entropy | 0 | 0.03 | 0.69 | 0 | 0.03 | 0.07 | 0 | 0.03 | 0.07 | 0 | 0.03 | 0.09 |

move algorithm. We used 0.5-0.25-0.25-0 as our similarity measure, in which the four components are *Value*, *In-link*, *Our-link*, and indirect similarity. As we mentioned before, *Value* is a single-value attribute, so indirect similarity is not applicable.

From table 7.5, we can see that, before node move algorithm was applied, we got perfect entropy values for all results with different numbers of clusters. For density, except when number of clusters is 5, our method outperformed SA-Cluster. After applying node move algorithm, our method got better results for density on all numbers of clusters. For entropy, although the values got a little bit worse, our results were still much better than those of SA-Cluster.

Table 7.6 compares our clustering result with the result in Satuluri and Parthasarathy [2009] for Cora paper citation network dataset. "# c" means number of clusters. "max $|c|$" means the size of biggest cluster.

The average N-Cut evaluation measure prefers smaller number of clusters and

**Table 7.6:** Clustering result compared with MLR-MCL in Satuluri and Parthasarathy [2009] for Cora paper citation network

| | **Our method** | | **MLR-MCL** | | **Our method** | | **MLR-MCL** | |
|---|---|---|---|---|---|---|---|---|
| | # c | max $\|c\|$ | # c | max $\|c\|$ | # c | max $\|c\|$ | # c | max $\|c\|$ |
| | 628 | 559 | 670 | 783 | 670 | 119 | 670 | 258 |
| Avg. N-Cut | 0.35 | | 0.35 | | 0.38 | | 0.41 | |

bigger clusters. So when we compare with other methods, we need to consider these factors. The left set of columns in table 7.6 shows that we got the same average N-Cut value with their method with a smaller number of clusters but smaller size of the biggest cluster. The right set of columns shows that, with the same number of clusters, we got better result even if the size of the biggest cluster is smaller.

Table 7.7 compares our clustering result with the result in Satuluri and Parthasarathy [2009] for Epinions social network dataset. This table shows that we got better result even with bigger number of clusters.

Table 7.8 compares our clustering result with the result in Newman and Girvan [2004] for dolphin social network dataset. "random" means random clustering which randomly puts data records in each cluster to generate equal-sized clusters, and 0.51 is one of the best Modularity values obtained. This table shows that the method in Newman and Girvan [2004] got better result than our method for dolphin social

**Table 7.7:** Clustering result compared with MLR-MCL in Satuluri and Parthasarathy [2009] for Epinions social network

| | **Our method** | | **MLR-MCL** |
|---|---|---|---|
| # Cluster | 1388 | 1641 | 1632 |
| Avg. N-Cut | 0.37 | 0.44 | 0.45 |

**Table 7.8:** Clustering result compared with Newman and Girvan [2004] for dolphin social network

| **Method** | **Our method** | | **Newman and Girvan [2004]** |
|---|---|---|---|
| | $K$-medoid + node move | random + node move | |
| # Cluster | 3 | 4 | 5 |
| Modularity | 0.42 | 0.51 | 0.52 |

network dataset.

Table 7.9 compares our clustering result with the result in Newman and Girvan [2004] for condensed matter collaboration network dataset. We used a much larger set of data and got better modularity result.

Table 7.10 shows the running time of our $K$-medoid algorithm and node move algorithm on political weblog dataset.

Table 7.11 displays the running time of node move step on several datasets.

**Table 7.9:** Clustering result compared with Newman and Girvan [2004] for condensed matter collaboration network

| | **Our method** | | **Newman and Girvan [2004]** | |
|---|---|---|---|---|
| | # cluster | dataset size | # cluster | dataset size |
| | 225 | 16726 | 13 | 145 |
| Modularity | 0.79 | | 0.72 | |

**Table 7.10:** Running time (second) of our $K$-medoid algorithm and node move algorithm on political blogs dataset

| Number of cluster | 3 | 5 | 7 | 9 |
|---|---|---|---|---|
| Time for clustering | 19 | 21 | 22 | 18 |
| Time for node move | $< 1$ | $< 1$ | $< 1$ | $< 1$ |

**Table 7.11:** Running time (second) of our node move algorithm on different datasets

| Dataset | Cora | Epinions | Dolphin | Condensed matter |
|---|---|---|---|---|
| Initial number of cluster | 750 | 5000 | 10 | 225 |
| Time for node move | 36 | 96 | $< 1$ | 12 |

## 7.2 Entity Resolution

Two publicly available datasets are used to test our proposed entity resolution methods. Our experiments were performed on these two sets of records separately, i.e., the records in each dataset constitutes input $D$ of our method. We use precision-recall curve and F1 to evaluate our experimental results. Area under the precision-recall curve (AUC) is a single metric calculated from the precision-recall curve. We also use the traditional F1 which is two times the ratio of the product of precision and recall to the sum of precision and recall. All experimental results are for all record pairs rather than only candidate record pairs. If some record pair is not a candidate record pair, then it will be automatically classified as nonequivalent.

### *7.2.1 Datasets*

We use two public datasets named Cora and CDDB in our experiments.

**Cora**: The Cora dataset includes bibliographical information about scientific publications. We downloaded it from Weis et al. [2009]. It contains 1878 citations which refer to 139 different research papers. It is an enlarged version of the dataset provided by McCallum [2005], which has 1295 citations referring to 122 papers. The Cora dataset has five attributes: Title, Author, Venue, Volume and Date. We cleaned it up in the preprocessing step by removing some punctuation marks including back-

slash, comma, period, colon, semicolon, etc. Also, we split the authors of each citation (if necessary) so as to represent each author as an Author attribute value for that citation record.

**CDDB**: The CDDB dataset includes 9763 (music) CDs randomly extracted from freeDB. We downloaded this from Weis et al. [2009] as well. Unlike the Cora dataset which has lots of equivalents, 9763 CDs in CDDB dataset refer to 9388 unique CDs which means that duplications are just a small fraction out of all the records. The CDDB dataset has seven attributes: Artist, Title, Category, Genre, Year, CDextra and Tracktitle. Similar to Cora, we split the artists of each CD (if necessary) so as to represent each artist as an Artist attribute value for that CD record.

### *7.2.2 Filtering and Domain-Optimized Direct Similarity Calculation for Cora and CDDB*

For Filtering, for Cora we choose Title and Author as $p_1$ and $p_2$, and use algorithm 4, because Title or Author each can distinguish a citation record well, and a citation generally has long title and many attributes. For CDDB we choose Artist and Title as $p_1$ and $p_2$, and use algorithm 3, because after inspection of the dataset, we feel that neither Artist or Title can distinguish a CD record well, and a CD generally has few artists and a short title.

The domain-specific optimizations we made for Cora dataset are as follows:

- For a candidate record pair $(i, j)$, if record $i$'s title contains some part of record $j$'s author, we move this part from $i$'s title to $i$'s author, and vice versa. Similarly, if record $i$'s author contains some part of record $j$'s title, we move this part for $i$'s author to $i$'s title, and vice versa;

- Replace abbreviation with its full word: For example, for venue name, we replace "PROC" with "PROCEEDINGS", "PHYS" with "PHYSICS", and "1ST" with "FIRST"; for date, we replace "jan" with "JANUARY";

- For two strings which are not the same and one is a substring of the other, give them some fixed similarity value (0.5 in our experiments);

- If one of the author names in a record pair is a one-character string or contains "et al" the similarity value of the pair is increased.

- For a candidate record pair $(i, j)$, if some attribute(s) is missing, we adjust the weights in equation (3.1). For example, if some attribute $p$ is missing, then its weight $w_p$ is set to zero, and the weights of other attributes are proportionately increased.

- If two records have the same authors and venue name but empty title, give them a high direct similarity value ($\frac{2}{3}$ in our experiments);

- If two records have the same title and venue name but empty author, set direct

similarity to 1;

- If two records have very similar title and venue name but empty author, and if their direct similarity is low, adjust it to a high value, (0.9 in our experiments).

The first two items are made in data preprocessing step. Because data preprocessing is commonly used for many data mining step, we will not discuss it in detail here.

The domain-specific optimizations we made for CDDB dataset are as follows:

- For two strings which are not the same and one string is a substring of the other, we set their similarity to some fixed value (0.75 in our experiments);

- For two strings which are not the same, contain some common substring, but neither one is a substring of the other, we set their similarity to some fixed value (0.5 in our experiments);

- For a candidate record pair $(i, j)$, if some attribute(s) is missing, we adjust the weights in equation (3.1). For example, if some attribute $p$ is missing, then its weight $w_p$ is set to zero, and the weights of other attributes are proportionately increased.

- If two CD's Tracktitle attribute is not empty but their Tracktitle field similarity is 0, set their direct similarity is set to 0.

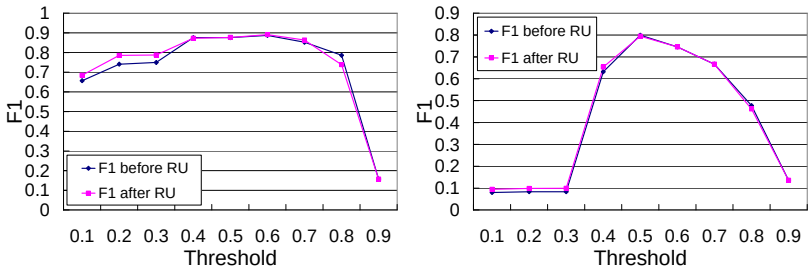


**Figure 7.12:** Unsupervised learning: Influence of matching thresholds on F1 (left: Cora, right: CDDB)

### *7.2.3 Unsupervised Learning*

For unsupervised learning, we only calculate direct similarity for record matching. Figure 7.12 shows the entity resolution results with different matching thresholds. In the following, "RU" means record updating. For Cora, the best matching threshold is around 0.6, and for CDDB, the best matching threshold is around 0.5. Also, we can see that record updating generally improves or does not hurt F1 score, especially when matching threshold is smaller than the best value.

Figure 7.13 shows the number of unique records (with "true" attributes which has equivalent) with different matching thresholds. For Cora, the actual number of unique records which has equivalent is 120, and this number for CDDB is 221. From the figure, we can see that the best matching threshold to get the actual value is about 0.55 for Cora and 0.65 for CDDB. Interestingly, this number increases with

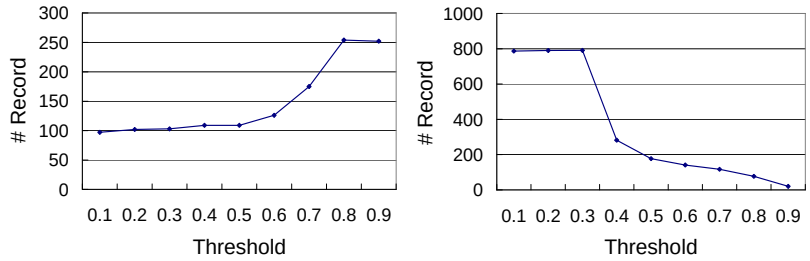


**Figure 7.13:** Unsupervised learning: Influence of matching thresholds on number of unique records with equivalent(s) (left: Cora, right: CDDB)

threshold for Cora, whereas it decreases with threshold for CDDB.

Table 7.12 shows we can use record updating to get best or quasi-best matching threshold with respect to F1. 0.1% means that we use 0.1% of all the candidate record pairs as training set to find the best matching threshold which maximizes the F1 score, and so on. Note that we only have about 6000 candidate record pairs for CDDB, so when we use 0.1% for Cora, we use 0.5% for CDDB. From this figure, we can see that after one record updating step, matching threshold stabilizes at 0.65 for Cora, and 0.49 for CDDB, no matter how small or large the training set is.

Table 7.13 shows the unsupervised learning results for both Cora and CDDB with the matching threshold obtained from record updating (with the help of a training set which is 1% of all the candidate record pairs). In the following, "#" refers to the number of unique records which has equivalent and "$t$" is the matching threshold.

**Table 7.12:** Unsupervised learning: Using record updating to get best threshold

| **Sample size** | **Cora** | | | | **CDDB** | | | |
|---|---|---|---|---|---|---|---|---|
| | Before RU | | After RU | | Before RU | | After RU | |
| | Threshold | F1 | Threshold | F1 | Threshold | F1 | Threshold | F1 |
| 0.1%/0.5% | 0.40 | 0.8757 | 0.65 | 0.8868 | 0.40 | 0.5956 | 0.49 | 0.6641 |
| 1% | 0.40 | 0.8757 | 0.65 | 0.8868 | 0.40 | 0.5956 | 0.49 | 0.6641 |
| 10% | 0.45 | 0.8807 | 0.65 | 0.8911 | 0.49 | 0.7993 | 0.49 | 0.7993 |
| 50% | 0.60 | 0.8965 | 0.65 | 0.9114 | 0.49 | 0.7993 | 0.49 | 0.7993 |
| 100% | 0.60 | 0.8965 | 0.65 | 0.9114 | 0.49 | 0.7993 | 0.49 | 0.7993 |

**Table 7.13:** Unsupervised learning results with best matching threshold

| **Step** | **Cora ($t$: 0.65)** | | **CDDB ($t$: 0.49)** | |
|---|---|---|---|---|
| | F1 | # | F1 | # |
| Before RU | 0.9108 | N/A | 0.7993 | N/A |
| After RU | 0.9178 | 150 | 0.7993 | 188 |

Combining Figures 7.12 and 7.13 with Tables 7.12 and 7.13, we can see that record updating does get best or at least quasi-best matching threshold for both datasets. Comparing Tables 7.12 and table 7.13, we see that we usually get a better (or no worse) result by using some training data and record updating to get a matching threshold followed by unsupervised learning without the help of training data using the threshold, than by using the training data all along, especially when training set is small.

For unsupervised learning, when we use domain-optimized direct similarity calculation and use the same thresholds for filtering algorithm, we usually do not get better results for Cora; but for CDDB, the F1 score improved by 1%-3%.

For unsupervised learning, in postprocessing, we only apply transitive closure (TC) to eliminate the inconsistent triangles in the classification result. MT can not be applied and IE is not necessary because no inconsistent triangles will be left after TC. Table 7.14 shows unsupervised learning results without applying TC. The matching threshold was obtained from record updating with the help of a training set which is 1% of all the candidate record pairs.

Table 7.15 shows the running time for unsupervised learning for both Cora and CDDB. In the table, "Pre" means preprocessing, "RM" represents record matching, "IE" is inconsistency elimination, and "RU" stands for record updating.

**Table 7.14:** Unsupervised learning results without TC

| **Step** | **Cora ($t$: 0.45)** | | **CDDB ($t$: 0.46)** | |
|---|---|---|---|---|
| | F1 | # | F1 | # |
| Before RU | 0.8837 | N/A | 0.8007 | N/A |
| After RU | 0.8835 | 116 | 0.7960 | 211 |

**Table 7.15:** Unsupervised learning: running time

| | Pre | RM | IE | RU |
|---|---|---|---|---|
| Cora | 3 | 1 | 13 | 0.1 |
| CDDB | 34 | 8 | 2 | 0.1 |

**Table 7.16:** Supervised learning: 5-fold cross validation with/without domain-optimized direct similarity calculation, using direct similarity only

| Step | Cora | | | | | | CDDB | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | w/o optimization | | | w/ optimization | | | w/o optimization | | | w/ optimization | | |
| | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # |
| Before RU | 0.9909 | 0.9805 | N/A | 0.9965 | 0.9916 | N/A | 0.7246 | 0.5940 | N/A | 0.8271 | 0.7128 | N/A |
| After RU | 0.9946 | 0.9911 | 132 | 0.9969 | 0.9944 | 119 | 0.7306 | 0.6152 | 213 | 0.7937 | 0.6351 | 201 |

### *7.2.4 Supervised learning*

Table 7.16 shows the supervised learning results with 5-fold cross validation for both datasets. It compares the results with and without domain-optimized direct similarity calculation. Only direct similarity is used to calculate probability of being equivalent. All results are average of three runs (same for Tables 7.16 – 7.19). For inconsistency elimination, unless otherwise specified, MT+TC+MT+IE is used.

Table 7.17 also shows the supervised learning results with 5-fold cross validation for both datasets, but both direct and indirect similarity are used to calculate probability of being equivalent.

Table 7.18 shows the supervised learning results with 3-fold cross validation for both datasets. Only direct similarity is used. From Tables 7.16 and 7.18, we can see that for both datasets F1 and AUC improves after record updating. Also,

**Table 7.17:** Supervised learning: 5-fold cross validation with/without domain-optimized direct similarity calculation, using both direct and indirect similarity

| Step | Cora | | | | | | CDDB | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | w/o optimization | | | w/ optimization | | | w/o optimization | | | w/ optimization | | |
| | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # |
| Before RU | 0.9910 | 0.9720 | N/A | 0.9967 | 0.9786 | N/A | 0.9281 | 0.8721 | N/A | 0.9569 | 0.9195 | N/A |
| After RU | 0.9940 | 0.9872 | 116 | 0.9968 | 0.9908 | 96 | 0.9313 | 0.9129 | 211 | 0.9607 | 0.9242 | 207 |

**Table 7.18:** Supervised learning: 3-fold cross validation with/without domain-optimized direct similarity calculation, using direct similarity only

| Step | Cora | | | | | | CDDB | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | w/o optimization | | | w/ optimization | | | w/o optimization | | | w/ optimization | | |
| | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # |
| Before RU | 0.9906 | 0.9774 | N/A | 0.9959 | 0.9885 | N/A | 0.7568 | 0.6485 | N/A | 0.7884 | 0.7105 | N/A |
| After RU | 0.9944 | 0.9891 | 134 | 0.9963 | 0.9923 | 117 | 0.7941 | 0.6673 | 219 | 0.7987 | 0.6471 | 203 |

domain-optimized direct similarity calculation helps.

Table 7.19 shows the supervised learning results with 3-fold cross validation for both datasets, but both direct and indirect similarity are used. From Tables 7.17 and 7.19 we can see that using indirect similarity often helps improve the classification result, especially for CDDB. We believe the fact that CDDB has much smaller fraction of equivalents causes its worse entity resolution results compared with Cora but bigger improvement with the help of indirect similarity.

**Table 7.19:** Supervised learning: 3-fold cross validation with/without domain-optimized direct similarity calculation, using both direct and indirect similarity

| Step | Cora | | | | | | CDDB | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | w/o optimization | | | w/ optimization | | | w/o optimization | | | w/ optimization | | |
| | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # | F1 | AUC | # |
| Before RU | 0.9900 | 0.9478 | N/A | 0.9953 | 0.9490 | N/A | 0.9376 | 0.8909 | N/A | 0.9584 | 0.9232 | N/A |
| After RU | 0.9925 | 0.9818 | 119 | 0.9956 | 0.9878 | 89 | 0.9418 | 0.9107 | 209 | 0.9614 | 0.9294 | 207 |

Table 7.20 compares the inconsistency elimination effect of different combinations of inconsistency elimination methods. We only use Cora for this comparison because generally there are no (or very few) inconsistencies in the classification result for CDDB. Value before/after "/" is the result before/after applying IE. "#" means the number of inconsistent triangles. "# $r$" means the number of unique records which has equivalent. All results are the average of three runs. Only direct similarity is used. We use algorithm 10 for IE and set $t = 5$. First, we can see that inconsistency elimination step improves classification accuracy, as well as generally eliminates all or most of the inconsistencies. Second, after record updating, the number of inconsistent triangles drops considerably. Third, all combinations show good inconsistency elimination ability except MT+TC+IE (see explanation in section 5.5.1). Fourth, MT+TC+MT+IE has the least number of inconsistencies and gets best classification accuracy. Finally, all combinations get good number of record with "true" attributes

**Table 7.20:** Supervised learning: inconsistency elimination

| Step | Before RU | | After RU | | |
|---|---|---|---|---|---|
| | F1 | # | F1 | # | # $r$ |
| None | .9754 | 254641 | .9780 | 185422 | 119 |
| MT+IE | .9959 | 266/15 | .9964 | 0/0 | 117 |
| MT+TC+IE | .9962 | 3/3 | .9967 | 0/0 | 120 |
| TC+MT+IE | .9915 | 164/24.3 | .9951 | 33/0 | 116 |
| MT+TC+MT+IE | .9965 | 10/0 | .9969 | 1/0 | 119 |

which has equivalent.

Table 7.21 compares our methods using transitive closure and nontransitive ideas, with methods using the idea of correlation clustering. All results are the average of three runs. This table shows that TC outperforms other methods. Like TC, NT without being followed by MT will lead to inconsistencies which are hard to be eliminated completely. We set $\delta = 0.14$ for the algorithm in Bansal et al. [2002].

Table 7.22 shows inconsistency elimination effect and time complexity with different group size for MT+TC+MT+IE, where group size is $t$ in algorithm 10. All results are average of five runs. We can see that the ability of inconsistency elimination decreases slowly as group size decreases, and the time needed to eliminate one inconsistent triangle decreases significantly as group size decreases. This table also

explains why we choose 5 as the group size in our experiments.

Table 7.23 compares the results of our method with those reported in the literature. These comparisons are all done on the Cora dataset since that is the dataset used in the literature. 25% means that randomly taking 25% record pairs as training set for each fold (totally 4 folds). Value before/after "/" is the result before/after applying domain-optimized direct similarity calculation. This table shows that our method outperforms comparable results in the literature.

Table 7.24 shows the running time for supervised learning (5-fold cross validation) for both Cora and CDDB. In the table, "CC1" stands for the correlation clustering method in Bansal et al. [2002], and "CC2" stands for the correlation clus-

**Table 7.21:** Supervised learning: inconsistency elimination with different methods

| **Step** | **Before RU** | | **After RU** | | |
|---|---|---|---|---|---|
| | F1 | # | F1 | # | # $r$ |
| MT+TC+IE | .9962 | 3/3 | .9967 | 0/0 | 120 |
| MT+NT1+IE | .9947 | 137/78 | .9963 | 8/8 | 126 |
| MT+NT2+IE | .9952 | 51/26 | .9961 | 36/23 | 119 |
| MT+CC Bansal et al. [2002] | .9886 | 3984 | .9926 | 4798 | 126 |
| MT+CC Ailon et al. [2005] | .8790 | 1730227 | .9654 | 1817095 | 114 |

**Table 7.22:** Supervised learning: Cora, inconsistency elimination vs. group size

| **Group size** | **10** | **5** | **4** | **3** |
|---|---|---|---|---|
| Percentage | 100% | 100% | 98.74% | 96.01% |
| Time (s) | 2.52 | 0.2 | 0.12 | 0.08 |

**Table 7.23:** Supervised learning: Cora, compare with other methods

| **Methods** | **5-fold (AUC)** | **3-fold (F1)** | **25% (F1)** |
|---|---|---|---|
| Our method | 0.991/0.994 | 0.994/0.996 | 0.982/0.988 |
| Other method | 0.988 (Singla and Domingos [2006]) | 0.947 (Wick et al. [2009]) | 0.92 (S. Rendle [2006]) |

tering method in Ailon et al. [2005]. Only direct similarity is used for record matching for Cora. Both direct and indirect similarity are used for record matching for CDDB.

## 7.3 Prediction of Movie Rating

The experiments reported here were done using the IMDB movies released from year 1991 to year 1995. By setting different thresholds and selecting movies with number of actors/actresses greater than the threshold, we obtained two datasets: one with a larger threshold with 512 movies and the other with a smaller threshold and

**Table 7.24:** Supervised learning: running time (second)

| | Pre | RM | MT | TC | IE | NT1 | NT2 | CC1 | CC2 | RU |
|---|---|---|---|---|---|---|---|---|---|---|
| Cora | 3 | 12 | 11 | 4 | 3 | 4 | 2 | 11 | 3 | 0.2 |
| CDDB | 34 | 25 | 0.3 | 0.2 | 0.2 | 0.2 | 0.2 | 69 | 7.5 | 0.1 |

5533 movies.

Figure 7.14 is the classification result when we use the original user ratings (without rounding). 5, 10, 15 and 20 refer to the value of $K$ in $K$-Nearest-Neighbor method.

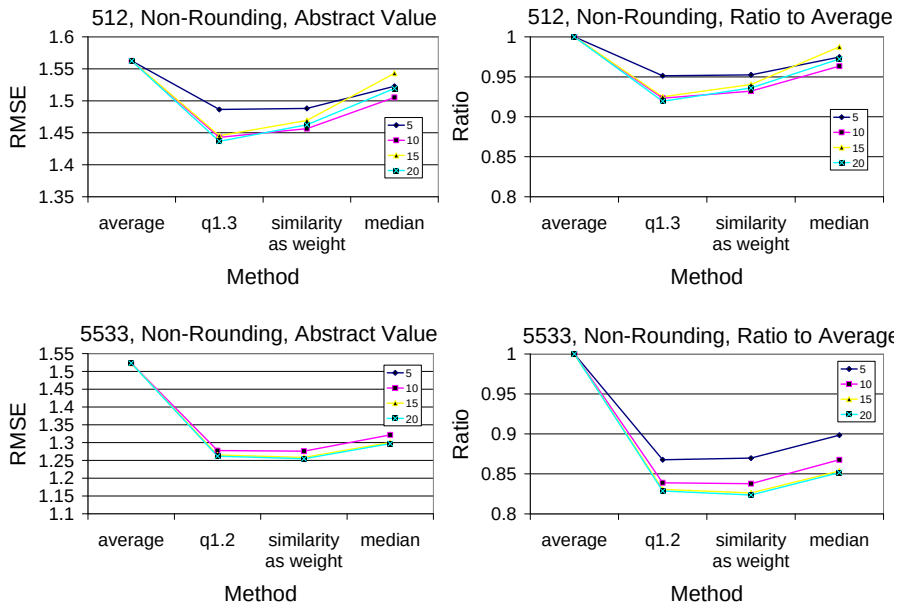


**Figure 7.14:** Nonrounding classification result

Figure 7.15 is the classification result when we use the rounded user ratings.

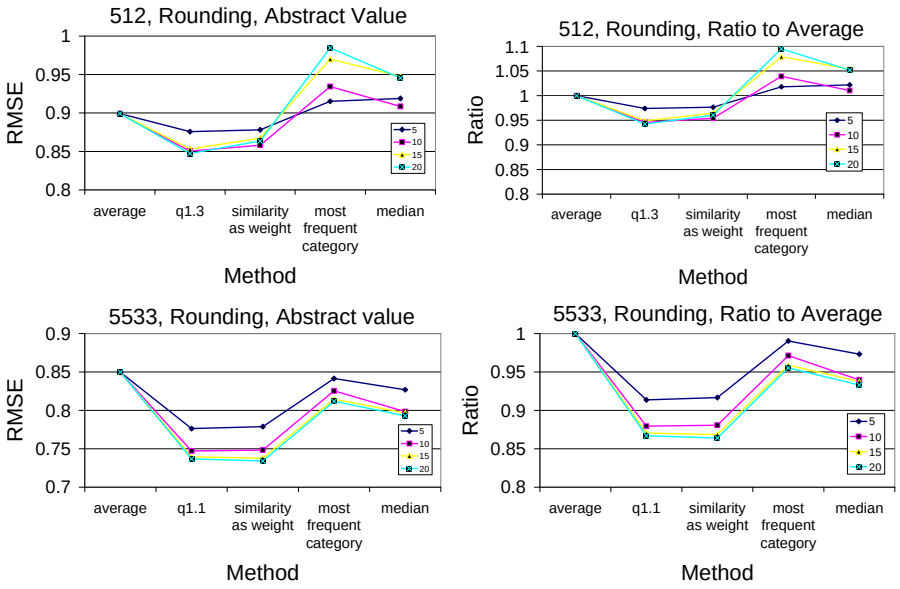


**Figure 7.15:** Rounding classification result

We can see that most of times "similarity as weight" method provides the best result compared with other methods.

# CHAPTER 8. CONCLUSION

The work in this dissertation focuses on three data mining research areas: data similarity, clustering, and entity resolution.

Data similarity measures how similar two data objects are. Based on our graph representation of data, we present a systematic data similarity measure in which direct similarity and indirect similarity are integrated in one graph explanation. We use this similarity measure in our clustering and entity resolution research.

Clustering techniques have been researched for a long time and have played an important role in many research and application fields. Our work on clustering focussed mainly on improving the clustering result. We explored different techniques: new similarity measure, new validity indices, improved clustering algorithm, new postprocessing method and algorithm. For clustering algorithm, we chose a classical clustering algorithm ($K$-medoid) as the basis of our clustering algorithm. We developed a post-clustering node move algorithm which showed great promise according to the experimental results on many different datasets. One other promising enhancement was to introduce multiple medoids into each cluster. We also experimentally investigated the interpretation of clustering result for an attribute which is not used in the similarity measure for clustering.

Entity resolution is closely related to data cleaning and data quality. We used similarity measure, direct record matching, and probabilistic model, together with some postprocessing steps in our entity resolution framework ERUDITE. Our method belongs to the traditional "merge-purge" method. The postprocessing steps mainly included record updating and inconsistency elimination, which can significantly improve the entity resolution result. Record updating played several different roles in our method. One interesting thing was that it can generate a good matching threshold for unsupervised learning. Our entity resolution method performs well on two different kinds of datasets: Cora has a lot of equivalent records, but CDDB has few equivalent records. We use similarity pattern instead of attribute value pattern in our probabilistic record matching model. Our inconsistency elimination can eliminate inconsistencies for different situations, especially for the inconsistencies with edge in both training and test set.

There are some open problems or questions remaining in our research. In our data similarity measure, direct similarity and indirect similarity can be explained on one graph explanation, but they could be integrated into a better unified form. K-medoid generally is not very fast — although we used sampling to address this issue, the speed of the algorithm would still be a problem for very large dataset. Moreover, when we used indirect similarity for clustering, the postprocessing steps often lead to oscillatory (nonconvergent) clustering results.

Our entity resolution work was conducted on single dataset, although it is not hard to adapt our method to multi-dataset scenario, for which schema matching may need to be applied first. The entity resolution results for CDDB (with much fewer equivalent records) are not as good as the results for Cora, which still has room for improvement. Although we use filtering to greatly decrease the size of the candidate record pairs set, the efficiency of pairwise record matching would still be a problem for very large datasets.

# Bibliography

Condensed matter collaborations 1999 dataset. URL http://www-personal.umich.edu/~mejn/netdata/.

Paper citation network dataset. URL http://www.cs.umass.edu/~mccallum/data.html.

Dolphin social network dataset. URL http://www-personal.umich.edu/~mejn/netdata/.

Enron email dataset. URL http://www-2.cs.cmu.edu/~enron/.

Epinions dataset. URL http://cs-www.cs.yale.edu/homes/mmahoney/NetworkData/.

The internet movie database. URL http://www.imdb.com/interfaces.

Political blogs network dataset. URL http://www-personal.umich.edu/~mejn/netdata.

Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 684–693, New York, NY, USA, 2005. ACM. ISBN 1-58113-960-8. doi: http://doi.acm.org/10.1145/1060590.1060692.

Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *ICDE'09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 952–963, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3545-6. doi: http://dx.doi.org/10.1109/ICDE.2009.43.

Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. In *Machine Learning*, pages 238–247, 2002.

Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal*, March 2008.

Johannes Berg and Michael Lässig. Cross-species analysis of biological networks by bayesian alignment. *Proceedings of the National Academy of Sciences*, 103(29): 10967–10972, July 2006. ISSN 0027-8424. doi: 10.1073/pnas.0602294103. URL http://dx.doi.org/10.1073/pnas.0602294103.

I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *SIAM International Conference on Data Mining*, pages 47–58, 2006.

Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Trans. Knowl. Discov. Data*, 1, March 2007. ISSN 1556-4681.

doi: http://doi.acm.org/10.1145/1217299.1217304. URL http://doi.acm.org/10.1145/1217299.1217304.

Mustafa Bilgic, Louis Licamele, Lise Getoor, and Ben Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *In Int. Symp. on Graph Drawing*, pages 505–507, 2005.

V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, and P. Van Dooren. *A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching*, volume 46:4. SIAM Review, 2004. ISBN 0036–1445.

Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 143–154, New York, NY, USA, 2005. ACM. ISBN 1-59593-060-4. doi: http://doi.acm.org/10.1145/1066157.1066175. URL http://doi.acm.org/10.1145/1066157.1066175.

Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE'07*, pages 746–755, 2007.

Daniel Boley. Principal direction divisive partitioning. *Data Min. Knowl. Discov.*, 2:

325–344, December 1998. ISSN 1384-5810. doi: 10.1023/A:1009740529316. URL http://dl.acm.org/citation.cfm?id=593421.593471.

D. G. Brizan and A. U. Tansel. A survey of entity resolution and record linkage methodologies. *Communications of the IIMA*, 6:41–50, 2006.

Peter Buneman. Semistructured data. *Tutorial in Symposium on Principles of Database Systems*, 1997.

Surajit Chaudhuri, Venkatesh Ganti, and Rajeev Motwani. Robust identification of fuzzy duplicates. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 865–876, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2285-8. doi: http://dx.doi.org/10.1109/ICDE.2005.125. URL http://dx.doi.org/10.1109/ICDE.2005.125.

Surajit Chaudhuri, Anish Das Sarma, Venkatesh Ganti, and Raghav Kaushik. Leveraging aggregate constraints for deduplication. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 437–448, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-686-8. doi: http://doi.acm.org/10.1145/1247480.1247530. URL http://doi.acm.org/10.1145/1247480.1247530.

Z. Chen, D. V. Kalashnikov, and S. Mehrotra. Adaptive graphical approach to entity

resolution. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 204–213, 2007. ISBN 978-1-59593-644-8.

Peter Christen. Febrl: a freely available record linkage system with a graphical user interface. In *Proceedings of the second Australasian workshop on Health data and knowledge management - Volume 80*, HDKM '08, pages 17–25, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc. ISBN 978-1-920682-61-3. URL http://dl.acm.org/citation.cfm?id=1385089.1385094.

Shu-Chuan Chu, John F. Roddick, and Jeng-Shyang Pan. A incremental multi-centroid, multi-run sampling scheme for k-medoids-based algorithms. Technical report, Proceedings of the Third International Conference on Data Mining Methods and Databases, Data Mining III, 2002.

Gao Cong, Wenfei Fan, Floris Geerts, Xibei Jia, and Shuai Ma. Improving data quality: consistency and accuracy. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 315–326. VLDB Endowment, 2007. ISBN 978-1-59593-649-3. URL http://dl.acm.org/citation.cfm?id=1325851.1325890.

D. J. Cook and L. B. Holder. *Mining Graph Data*. Hoboken, N.J John Wiley & Sons, Inc. (US), 2007. ISBN 9780471731900.

Aron Culotta and Andrew McCallum. Joint deduplication of multiple record types in

relational data. In *CIKM'05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 257–258, New York, NY, USA, 2005. ACM. ISBN 1-59593-140-6. doi: http://doi.acm.org/10.1145/1099554.1099615.

Xin Dong, Alon Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD'05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 85–96, New York, NY, USA, 2005. ACM. ISBN 1-59593-060-4. doi: http://doi.acm.org/10.1145/1066157.1066168.

Mohamed Elfeky, Vassilios Verykios, and Ahmed Elmagarmid. Tailor: A record linkage toolbox, 2002.

Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1):1–16, 2007. ISSN 1041-4347. doi: http://dx.doi.org/10.1109/TKDE.2007.9.

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

Ariel Fuxman, Elham Fazli, and Renée J. Miller. Conquer: efficient management of inconsistent databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 155–166, New York, NY,

USA, 2005. ACM. ISBN 1-59593-060-4. doi: http://doi.acm.org/10.1145/1066157.1066176. URL http://doi.acm.org/10.1145/1066157.1066176.

Helena Galhardas, Daniela Florescu, Dennis Shasha, Eric Simon, and Cristian-Augustin Saita. Declarative data cleaning: Language, model, and algorithms. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 371–380, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4. URL http://dl.acm.org/citation.cfm?id=645927.672042.

David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 721–732. VLDB Endowment, 2005. ISBN 1-59593-154-6. URL http://dl.acm.org/citation.cfm?id=1083592.1083676.

M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering validity methods: part I. In *ACM SIGMOD Record (ACM Special Interest Group on Management of Data)*, volume 31, pages 40–45, June 2002a.

M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering validity checking methods: part II. In *ACM SIGMOD Record (ACM Special Interest Group on Management of Data)*, volume 31, pages 19–27, September 2002b.

Ji He, Ah-Hwee Tan, Chew-Lim Tan, and Sam-Yuan Sung. *On Quantitative Evalu-*

*ation of Clustering Systems*. Kluwer Academic Publishers, 2003. URL citeseer.ist.psu.edu/he02quantitative.html.

X. D. Huang and W. Lai. Clustering graphs for visualization via node similarities. *Journal of Visual Languages & Computing*, 17(3):225–253, June 2006.

K. Jedidi. *Marketing Letters*, volume 9:4. Kluwer Academic Publishers, 1998.

Glen Jeh and Jennifer Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 538–543, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: http://doi.acm.org/10.1145/775047.775126. URL http://doi.acm.org/10.1145/775047.775126.

L. Kaufman and P. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. New York: John Wiley and Sons, 1990.

L. Kaufmann and P. Rousseeuw. *Finding Groups in Data: An Introdution to Cluster Analysis*. John Wiley and Sons, 1990.

Jon M. Kleinberg. Hubs, authorities, and communities. *ACM Comput. Surv.*, 31, December 1999. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/345966.345982. URL http://doi.acm.org/10.1145/345966.345982.

Nick Koudas, Amit Marathe, and Divesh Srivastava. Spider: flexible matching in

databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 876–878, New York, NY, USA, 2005. ACM. ISBN 1-59593-060-4. doi: http://doi.acm.org/10.1145/1066157.1066272.

J. Larson. *REGGAE Whitepaper*. Applied Technical Systems, Inc., 2008.

C. Legány, S. Juhász, and A. Babos. Cluster validity measurement techniques. In *AIKED'06: Proceedings of the 5th WSEAS International Conference on Artificial Intelligence, Knowledge Engineering and Data Bases*, pages 388–393, Stevens Point, Wisconsin, USA, 2006. World Scientific and Engineering Academy and Society (WSEAS). ISBN 111-2222-33-9.

E. A. Leicht, Petter Holme, and M. E. J. Newman. Vertex similarity in networks. *Phys. Rev. E*, 73:026120, Feb 2006. doi: 10.1103/PhysRevE.73.026120. URL http://link.aps.org/doi/10.1103/PhysRevE.73.026120.

RONG Li, Shunliang CAO, Yuanyuan LI, Hao TAN, Yangyong ZHU, Yang ZHONG, and Yixue LI. A measure of semantic similarity between gene ontology terms based on semantic pathway covering. *PROGRESS IN NATURAL SCIENCE*, 16 (17), 2006.

Y. Lifshits. Similarity search: a web perspective, October 2007. available at http://yury.name/algoweb/lifshits-retreat.pdf.

Hartmut S Loos, Dagmar Wieczorek, Rolf P Würtz, Christoph von der Malsburg, and Bernhard Horsthemke. Computer-based recognition of dysmorphic faces. *Eur J Hum Genet*, 11(8):555–60, 2003. ISSN 1018-4813. URL http://www.biomedsearch.com/nih/Computer-based-recognition-dysmorphic-faces/12891374.html.

W. Z. Lu, J. Janssen, E. Milios, and N. Japkowicz. Node similarity in networked information spaces. In *Proceedings of the 2001 conference of the Centre for Advanced Studies on Collaborative research*, IBM Centre for Advanced Studies Conference, page 11, Toronto, Ontario, Canada, 2001. IBM Press.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

A. McCallum. Cora citation matching, October 2005. URL http://www.cs.umass.edu/~mccallum/data/cora-refs.tar.gz.

Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM. ISBN 1-

58113-233-6. doi: http://doi.acm.org/10.1145/347090.347123. URL http://doi.acm.org/10.1145/347090.347123.

S. N. Minton and C. Nanjo. A heterogeneous field matching method for record linkage. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*, November 2005.

Manikandan Narayanan and Richard M. Karp. Comparing protein interaction networks via a graph match-and-split algorithm. *Journal of Computational Biology*, 14:892–907, 2007.

H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James. Automatic linkage of vital records. *Science*, 130:954–959, October 1959. ISSN 0036-8075.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004. doi: 10.1103/PhysRevE.69.026113. URL http://link.aps.org/doi/10.1103/PhysRevE.69.026113.

N. R. Pal and L. Jain. *Advanced Techniques in Knowledge Discovery and Data Mining*. Springer, 2005. ISBN 1-85233-867-9.

Uchang Park and Yeojin Seo. An implementation of xml documents search system based on similarity in structure and semantics. In *Proceedings of the International Workshop on Challenges in Web Information Retrieval and Integration*, pages 97–

103, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2414-1. URL http://dl.acm.org/citation.cfm?id=1105926.1106235.

Orion Penner, Vishal Sood, Gabriel Musso, Kim Baskerville, Peter Grassberger, and Maya Paczuski. Node similarity within subgraphs of protein interaction networks. *Physica A: Statistical Mechanics and its Applications*, 387(14):3801–3810, Jun 2008. URL http://www.sciencedirect.com/science/article/B6TVG-4S1C8BN-4/1/9ba1bd3da55823f17674b4737d8c943c.

Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 381–390, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4. URL http://dl.acm.org/citation.cfm?id=645927.672045.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294.

L. S. Thieme S. Rendle. Object identification with constraints. In *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM'06)*, pages 1026–1031, December 2006.

I. Saha and A. Mukhopadhyay. An improved crisp and fuzzy based clustering tech-

nique for categorical data. *International Journal of Computer Science and Engineering*, 2(4):184–193, Fall 2008.

Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, New York, NY, USA, 2002. ACM. ISBN 158113567X. doi: 10.1145/775047.775087.

Venu Satuluri and Srinivasan Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 737–746, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9. doi: http://doi.acm.org/10.1145/1557019.1557101. URL http://doi.acm.org/10.1145/1557019.1557101.

Erich Schubert, Sebastian Schaffert, and F. Bry. Structure-preserving difference search for xml documents. In *Extreme Markup Languages®'05*, 2005.

O¨ zgu¨r Simsek and David Jensen. Decentralized search in networks using homophily and degree disparity. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 304–310, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. URL http://dl.acm.org/citation.cfm?id=1642293.1642342.

P. Singla and P. Domingos. Entity resolution with markov logic. In *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM'06)*, pages 572–582, December 2006.

Alan F. Smeaton and Patrick J. Morrissey. Experiments on the automatic construction of hypertext from texts, 1995.

B. Stein and S. M. Eissen. Document categorization with majorclust. In *Proc. 12th Workshop on Information Technology and Systems, Tech. Univ. of Barcelona, Spain*, 2002.

A. Strehl and J. Ghosh. A scalable approach to balanced, high-dimensional clustering of market-baskets. In *Proceedings of the 7th International Conference on High Performance Computing (HiPC 2000)*, volume 1970 of *Lecture Notes In Computer Science*, pages 525–536. Springer, 2000.

John R. Talburt. *Entity Resolution and Information Quality.* Morgan Kaufmann, 2010.

P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Pearson Education, 2006. ISBN 0-321-32136-7.

M. Weis, F. Naumann, and F. Brosy. A duplicate detection benchmark for xml data, May 2009. URL http://www.hpi.uni-potsdam.de/naumann/projekte/

repeatability/duplicate_detection/a_duplicate_detection_benchmark_for_xml_data.html.

Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity resolution with iterative blocking. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 219–232, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-551-2. doi: http://doi.acm.org/10.1145/1559845.1559870.

M. L. Wick, A. Culotta, K. Rohanimanesh, and A. McCallum. An entity based model for coreference resolution. In *SDM'09*, pages 365–376, 2009.

W. E. Winkler. Data cleaning methods. In *Proceedings of the KDD-2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

William E. Winkler. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006.

Changjun Wu and Ananth Kalyanaraman. An efficient parallel approach for identifying protein families in large-scale metagenomic data sets. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 35:1–35:10, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9. URL http://dl.acm.org/citation.cfm?id=1413370.1413406.

X. Yin, J. Han, and P. S. Yu. Linkclus: Efficient clustering via heterogeneous semantic links. In *VLDB'06*, pages 427–438, 2006.

C. Zhao and K. Sivakumar. A graph-based similarity metric and validity indices for clustering non-numeric and unstructured data. In *Proceedings of The 2009 International Conference on Data Mining (DMIN 2009)*, pages 531–537, 2009.

C. Zhao and K. Sivakumar. An effective entity resolution method. In *Proceedings of The 2010 International Conference on Data Mining (DMIN 2010)*, pages 10–16, 2010.

C. Zhao and K. Sivakumar. Clustering for semi-structured data (to be submitted). *Knowledge and Information Systems (KAIS)*, 2011a.

C. Zhao and K. Sivakumar. Erudite: A general entity resolution framework (under review). *Knowledge and Information Systems (KAIS)*, 2011b.

Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Graph clustering based on structural/attribute similarities. *Proc. VLDB Endow.*, 2:718–729, August 2009. ISSN 2150-8097. URL http://portal.acm.org/citation.cfm?id=1687627.1687709.