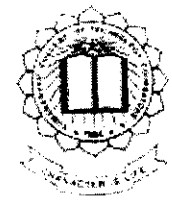


Effective Integration By Inter Attribute Dependency Graphs of Schema Matching

PROJECT REPORT

Submitted by

M.RAMKUMAR

Register No: 0820108012

in partial fulfillment for the award of the degree

of

MASTER OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 006

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated to Anna University, Coimbatore)

COIMBATORE – 641 006

Department of Computer science and Engineering

PROJECT WORK

PHASE II

MAY 2010

This is to certify that the project entitled

Effective Integration By Inter Attribute Dependency Graphs of Schema Matching

is the bonafide record of project work done by

M.RAMKUMAR

Register No: 0820108012

of M.E. (Computer Science and Engineering) during the year 2009-2010.

V.S. Akshaya

Project Guide

(Mrs. V.S. AKSHAYA)

Head of the Department

(Mrs. P. DEVAKI)

Submitted for the Project Viva-Voce examination held on 17/5/10

17/5/10

External Examiner

DECLARATION

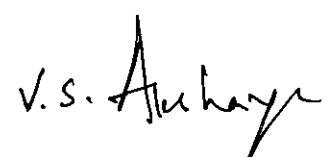
I affirm that the project work titled "**Effective Integration By Inter Attribute Dependency Graphs of Schema Matching**" being submitted in partial fulfillment for the award of M.E Computer Science and Engineering is the original work carried out by me. It has not formed the part of any other project work submitted for award of any degree or diploma, either in this or any other University.

M.RAMKUMAR

0820108012

I certify that the declaration made above by the candidate is true

Signature of the Guide,



Mrs. V.S. AKSHAYA

Senior lecturer

ACKNOWLEDGEMENT

I express my profound gratitude to our Chairman **Padmabhusan Arutselver Dr. N. Mahalingam B.Sc, F.I.E** for giving this great opportunity to pursue this course.

I would like to begin by thanking to **Dr.S.Ramachandran., Ph.D**, *principal* for providing the necessary facilities to complete my thesis.

I take this opportunity to thank **Dr. S.Thangasamy, Ph.D.,** *R & D Dean*, for his precious suggestions.

I take this opportunity to thank **Mrs. P. Devaki M.E., (Ph.D),** *Head of the Department*, Computer Science and Engineering, for his precious suggestions

I register my hearty appreciation to **Mrs. V.S. Akshaya M.E., (Ph.D),** *Senior lecturer*, my thesis advisor. I thank for her support, encouragement and ideas. I thank her for the countless hours she has spent with me, discussing everything from research to academic choices.

I thank all project committee members for their comments and advice during the reviews. Special thanks to **Mrs.V.Vanitha M.E., (Ph.D)** *Assistant professor*, Department of Computer science and Engineering, for arranging the brain storming project review sessions.

I would like to convey my honest thanks to **Mr. T. Chandra Vel Kumar (B.E)**, *Lab Assistant*, all **Teaching** staff members and **Non Teaching** staffs of the department for their support. I would like to thank all my classmates who gave me a proper light moments and study breaks apart from extending some technical support whenever I needed them most.

I dedicate this project work to my **parents** and my **friends**, without their love this work wouldn't possible.

TABLE OF CONTENTS

(An Autonomous Institution affiliated to Anna University, Coimbatore
Approved by AICTE - Accredited by NBA & NAAC with A- Grade)
Sathyamangalam - 638401. Erode District, Tamilnadu.

Department Of Information Technology

3rd National Conference on

Recent Trends in Information and Communication Technology

CERTIFICATE

This is to certify that Dr. /Mr. /Ms. [redacted] RAM KUMAR ..... of

..... KUMARAGURU COLLEGE OF TECHNOLOGY ..... has participated and presented a paper titled

EFFECTIVE INTEGRATION BY INTER ATTRIBUTE DEPENDENCY GRAPHS OF SCHEMA MATCHING ..... in

3rd National Conference on Recent Trends In Information And Communication Technology RTICT 2010, organized by

Sri Amman Institute of Technology, Sathyamangalam, during 9-10 April 2010.

Amrita

Amrita Wahi

Dr. A. Shanmugam

Principal

CSIR, New Delhi

ABSTRACT

Schema matching is one of the key challenges in information integration. It is a labor-intensive and time-consuming process. To alleviate the problem, many automated solutions have been proposed. Most of the existing solutions mainly rely upon textual similarity of the data to be matched. However, there exist instances of the schema-matching problem for which they do not even apply. Such problem instances typically arise when the column names in the schemas and the data in the columns are opaque or very difficult to interpret. It has been proposed a two-step technique to address this problem. In the first step, we measure the dependencies between attributes within tables using an information-theoretic measure and construct a dependency graph for each table capturing the dependencies among attributes. In the second step, it has been found that matching node pairs across the dependency graphs by running a graph-matching algorithm.

ஆய்வுச் சுருக்கம்

அமைப்பு முறைகள் பொருத்துதல் என்பது தகவல் ஒன்றுபடுத்தலில் மிக முக்கிய சவாலான ஒன்றாகும். இச்செயல்முறையை செய்வதற்கு கூடிய நேரமும் மற்றும் மிக ஆழமான வேலைத் தேவைப்படுகிறது. இந்தச் சிக்கல்களைத் தீர்க்க பல தன்னியக்கத் தீர்வு வழிகள் முன் மொழியப்பட்டுள்ளது. ஆனாலும் பட்டியலில் உள்ள நெடுவரிசைப் பெயர் தெரியாத பட்சத்தில் அமைப்பு முறைகள் பொருத்தும் திட்டம் சிக்கலாக உள்ளது. இந்த சூழ்நிலையில் இத்திட்டத்தில் முன்மொழிவது என்னவென்றால் முதலில் பட்டியலின் இடையில் உள்ள சார்வு பண்புகளை அளவீடு செய்து ஒரு வரைபடத்தை உருவாக்குகிறோம். அதற்குப் பின்னர் வரைபடத்தில் பொருந்தும் முனையங்களை வரைபடம் பொருந்தும் நெறிமுறைகள் மூலம் சார்வு வரைபடத்தில் கண்டு அறிகிறோம். பிறகு அணுகுமுறையின் துல்லியத்தையும் வரைபடம் பொருந்தும் நெறிமுறைகளின் கணிப்புச் சிக்கல்களை ஆராய்ந்து அமைப்பு முறைகளைத் துல்லியமாக பொருத்துகிறோம்.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVATIONS

SLUB	Structure-Level Usage-Based Matcher
ELUB	Element-Level Usage- Based Matcher
WGMP	Weighted Graph Matching Problem
VF2	Vector Fitting2
HC	Hill Climbing
MI	Mutual Information
ET	Entropy
A	Attribute
Q	Query

CHAPTER 1

PROBLEM DEFINITION

Schema matching is the process of identifying that two objects are semantically related while mapping refers to the transformations between the objects and schema describes the structure of a database.

Schema matching is one of the key challenges in information integration. It is a labor-intensive and time-consuming process. Here we are going to propose a two-step technique to address this problem. In the first step, we measure the dependencies between attributes within tables using an information-theoretic measure and construct a dependency graph for each table capturing the dependencies among attributes. In the second step, we find matching node pairs across the dependency graphs by running a graph-matching algorithm. In our previous work, we experimentally validated the accuracy of the approach. One remaining challenge is the computational complexity of the graph-matching problem in the second step. The problem instance we are facing is the weighted graph-matching problem to which no efficient solution has yet been found. In this project, we extend the previous work by improving the second phase of the algorithm incorporating efficient approximation algorithms into the framework.

CHAPTER 2

INTRODUCTION

The schema-matching problem at the most basic level refers to the problem of mapping schema elements (for example, columns in a relational database schema) in one information repository to corresponding elements in a second repository. For example, in the two schemas DB1.Student (Name, SSN, Level, Major, Marks) and DB2.Grad-Student (Name, ID, Major, Grades); possible matches would be: DB1.Student ≈ DB2.Grad-Student.

While schema matching has always been a problematic and interesting aspect of information integration, the problem is exacerbated as the number of information sources to be integrated, and hence, the number of integration problems that must be solved, grows. Such schema-matching problems arise both in "classical" scenarios such as company mergers and in "new" scenarios such as the integration of diverse sets of queriable information sources over the web.

Purely manual solutions to the schema-matching problem are too labor intensive to be scalable; as a result, there has been a great deal of research into automated techniques that can speed this process by either automatically discovering good mappings, or by proposing likely matches that are then verified by some human expert. In this project, we present such an automated technique that is designed to be of assistance in the particularly difficult cases in which the column names and data values are "opaque," and/or cases in which the column names are opaque and the data values in multiple columns are drawn from the same domain. Our

approach works by computing the "mutual information" between pairs of columns within each schema, and then using this statistical characterization of pairs of columns in one schema to propose matching pairs of columns in the other schema.

Table 1. Illustrates classification of schema-matching techniques based on the use of data interpretation and structural similarity. While all four classes of techniques are valuable in different domains, but we focusing on uninterpreted structure matching in this project.

Data Interpretation	Structural Similarity: Element-level	Structural Similarity: Structure-level
Interpreted	**Interpreted Element Matching** *Bayesian Classifier* *Neural Network* *Attribute Name Matcher* *Attribute Constraints Matcher*	**Interpreted Structure Matching** *Schema Graph Matching*
Un-interpreted	**Un-interpreted Element Matching** *Unique Value Count* *Frequency Distribution* *Information Entropy*	**Un-interpreted Structure Matching** *Mutual Information* *Dependency Graph Matching* *Causal Structure Matching*

Table 1 - Schema matching technique classification

CHAPTER 3

LITERATURE SURVEY

3.1. Usage-Based Schema Matching

3.1.1. Introduction:

Schema matching has long been one of the most important, yet difficult, problems in the area of data integration. With the exploding number of information systems, the need for schema matching solutions is growing. In life sciences, information integration is becoming a bottleneck limiting what scientists can accomplish. Businesses are relying on integration more than ever before. In disaster recovery situations, several entities and authorities need to rapidly exchange information. Schema matching is a key component in all such applications. Moreover, many of today's integration tasks have to cross country boundaries, thus adding a new dimension to this vexing challenge.

Existing techniques for schema matching are classified as either schema-based, instance-based, or a combination of both. In this work, we define a new class of techniques, called usage-based schema matching. The idea is to exploit information extracted from the query logs to find correspondences between attributes in the schemas to be matched. We propose methods to identify co-occurrence patterns between attributes in addition to other features such as their use in joins and with aggregate functions. Several scoring functions are considered to measure the similarity of the extracted features, and a genetic algorithm is employed to find the highest score mappings between the two schemas. Our technique is suitable for matching schemas even when their attribute names are opaque. It can further be combined with existing techniques to obtain more accurate results. Our experimental study demonstrates the effectiveness of the proposed approach and the benefit of combining it with other existing approaches.

3.1.2. Existing System:

Existing techniques for schema matching are classified as either schema-based, instance-based, or a combination of both. Schema-level matchers only consider schema information, not instance data. The available information includes the usual properties of schema elements, such as name, description, data type, relationship types (part-of, is-a, etc.), constraints, and schema structure. Workings at the element (atomic elements like attributes of objects) or structure level (matching combinations of elements that appear together in a structure), these properties are used to identify matching elements in two schemas. Language-based or linguistic matchers use names and text (i.e., words or sentences) to find semantically similar schema elements. Constraint based matchers exploit constraints often contained in schemas. Such constraints are used to define data types and value ranges, uniqueness, optionality, relationship types and cardinalities, etc. Constraints in two input schemas are matched to determine the similarity of the schema elements. Instance-level matchers use instance-level data to gather important insight into the contents and meaning of the schema elements. These are typically used in addition to schema level matches in order to boost the confidence in match results, more so when the information available at the schema level is insufficient. Matchers at this level use linguistic and constraint based characterization of instances.

3.1.3. Enhanced Technique:

The goal of the usage-based schema matching technique is to exploit similarities in the query patterns to match attributes, which seem to play the same role in their respective databases. We are not claiming that the query patterns in the same domain will always be similar. However, like schema-based techniques, which will only be effective when the attribute names share some similarities, usage-based techniques will be most effective when the query patterns are close to each other; a requirement that is likely to be met in many situations. The same argument applies to other domains like healthcare, finance, and scientific databases. Our proposed technique has two main phases: feature extraction and matching. The feature extraction phase collects

information from the query logs characterizing the attributes' roles and their interrelationships. The matching phase examines several potential mappings, and assigns a score for each one of them. The scores are based on how well the features of the corresponding attributes match. The matching phase terminates by reporting the highest-score mapping (or mappings). During the feature extraction phase, the query log of each schema is scanned to collect both structure-level and element level features. The structure-level features are used by a Structure-Level Usage-Based matcher (SLUB), whereas the element-level features are used by an Element-Level Usage- Based matcher (ELUB). The structure-level features capture the usage relationships between attributes of the same schema. An attribute A appearing in a query Q would normally have a role in defining Q's answer. A can be part of the answer (A occurs in the select clause), or it can have a filtering role (A appears in the where or having clauses), a grouping role (A occurs in the group by clause), or an ordering role (A occurs in the order by clause). Furthermore, if two attributes co-occur in the same query, they potentially have a usage relationship whose type is defined by the role of each of them in defining the query's answer.

3.1.4. Advantages:

- This technique is suitable for matching schemas even when their attribute names are opaque or when they have different layouts.
- It is applicable to match complete schemas, rather than individual tables.
- Usage based schema technique can be combined with other matching techniques.

3.1.5. Disadvantages:

- The efficiency of the usage based schema technique is not much good as the existing.
- Cannot match individual tables

3.2. A Dynamical Systems Approach to Weighted Graph Matching

3.2.1. Introduction:

Graph matching is a fundamental problem that arises frequently in the areas of distributed control, computer vision, and facility allocation. In this work, we consider the optimal graph matching problem for weighted graphs, which is computationally challenging due the combinatorial nature of the set of permutations. Contrary to optimization-based relaxations to this problem, in this work we develop a novel relaxation by constructing dynamical systems on the manifold of orthogonal matrices. In particular, since permutation matrices are orthogonal matrices with nonnegative elements, we define two gradient flows in the space of orthogonal matrices. The first minimizes the cost of weighted graph matching over orthogonal matrices, whereas the second minimizes the distance of an orthogonal matrix from the finite set of all permutations. The combination of the two dynamical systems converges to a permutation matrix which provides a suboptimal solution to the weighted graph matching problem. Finally, our approach is shown to be promising by illustrating it on nontrivial problems. Graph matching plays a key role in many areas of computing from computer vision to networks where there is a need to determine correspondences between the components (vertices and edges) of two attributed structures. In recent years three new approaches to graph matching have emerged as replacements to more traditional heuristic methods. These new methods are:

- Least squares - where the optimal correspondence in determined in terms of deriving the best fitting permutation matrix between sets.
- Spectral methods - where optimal correspondences are derived via subspace projections in the graph Eigen spaces.
- Graphical models - where algorithms such as the junction tree algorithm are used to infer the optimal labeling of the nodes of one graph in terms of the other and that satisfy similarity constraints between vertices and edges.

3.2.2. Existing System:

Given two graphs with weights on edges, the weighted graph matching problem searches for an optimal permutation of nodes of one graph so that the difference between the edge weights is minimized. Graph matching problems arise frequently in computer vision, facility allocation problems, as well as distributed control. In computer vision, matching structural descriptions of an object to those of a model is formulated as a graph matching problem. In distributed control and distributed robotics, graphs are recently emerging as a natural mathematical description for capturing interconnection topology. Graph matching problems in this context can be used by a team of robots to reach a particular formation or minimize a distance from a particular formation. Finally, in facility allocation, graph matching is similar to the well known Quadratic Assignment Problem. In addition to its frequent appearance in various fields, weighted graph matching has also received a lot of attention due to its hardness. Since, it includes as a special case the largest common sub graph problem, which is NP-complete; it is also NP-complete. In particular, by its similarity to the quadratic assignment problem, problems with 20-25 nodes are considered very hard, and problems with more than 30 nodes are practically intractable. They treat weighted graphs with the same number of nodes and employ an analytic approach by using the Eigen-structure of adjacency matrices (undirected graph matching) or some Hermitian matrices derived from the adjacency matrices (directed graph matching). An almost optimal matching can be found when the graphs are sufficiently close to each other. The authors propose a Lagrangian Relaxation Network for the same problem. They formulate the permutation matrix constraints in the framework of deterministic annealing and achieve exact constraint satisfaction at each temperature within deterministic annealing.

3.2.3. Enhanced Technique:

In this work, we considered the problem of finding the optimal relabeling of the vertices of a graph so that its distance from some reference graph is minimized in the Frobenious norm sense. We relaxed the combinatorial nature of the problem by giving an equivalent representation

for the set of permutation matrices as the intersection of the space of orthogonal matrices with the set of element wise nonnegative matrices. This representation gave rise to defining two gradient flows on the space of orthogonal matrices, such that one minimizes the distance of the two graphs and the second converges to a permutation matrix. We discussed superimposing the two gradient flows to apply them to the weighted graph matching problem, as well as initialization and a hybrid scheme for combining the two dynamical systems. Our algorithm is provably correct and the simulations illustrate our theoretical results, as well as the high performance achieved when combined with the proposed heuristics.

3.2.4. Advantages:

- The high performance achieved when combined with the proposed heuristics.
- The first minimizes the cost of weighted graph matching over orthogonal matrices.
- The second minimizes the distance of an orthogonal matrix from the finite set of all permutations.

3.2.5. Disadvantages:

- It does not allow varied and distorted patterns to be matched which were not included in the training set.
- It does not support both statistical and structural information.

CHAPTER 4

EXISTING SYSTEM

4.1. Existing System & its Drawbacks:

- In the current work, we have only tested two simple distance metric Euclidean and normal. Distance metric learning is an emerging area of statistical learning in which the goal is to induce a more powerful distance metric from labeled examples.
- In our previous work, we proposed a two-step technique to address this problem. In the first step, we measure the dependencies between attributes within tables using an information-theoretic measure and construct a dependency graph for each table capturing the dependencies among attributes. In the second step, we find matching node pairs across the dependency graphs by running a graph-matching algorithm.
- In our previous work, we experimentally validated the accuracy of the approach. One remaining challenge is the computational complexity of the graph-matching problem in the second steps.
- This approach is called instance-based matching. Instance based matching also will work in many cases. For example, if we are deciding whether to match Dept in one schema to either DeptName or DeptID in the other, by looking at the column instances, one may easily find the mapping because DeptName and DeptID are likely to be drawn from different domains . Unfortunately, however, instance-based matching is also not always successful.
- Some of the attribute names will be clear candidates for matching, due to common names or common parts of names. Using the classification given in , such an approach is an example of

schema-based matching. However, for many columns, schema-based matching will not be effective because different institutions may use different names for semantically identical attributes, or use similar names for unrelated attributes.

- When instance-based mapping fails, it is often because of its inability to distinguish different columns over the same data domain and, similarly, its inability to find matching columns using values drawn from different domains. For example, EmployeeID and CustomerID columns in a table are unlikely to be distinguished if both the columns use similar IDs.

4.2. Existing System Methodologies:

The Integration process comprises of four main steps:

- **Pre integration**

  An analysis of schemas is carried out before integration to decide upon some integration policy. This governs the choice of schemas to be integrated, the order of integration, and a possible assignment of preferences to entire schemas or portions of schemas.

- **Comparison of the Schemas**

  Schemas are analyzed and compared to determine the correspondences among concepts and detect possible conflicts. Inter schema properties may be discovered while comparing schemas.

- **Conforming the Schemas**

Once conflicts are detected, an effort is made to resolve them so that the merging of various schemas is possible.

- **Merging and Restructuring**

Now the schemas are ready to be superimposed, giving rise to some intermediate integrated schema(s). The intermediate results are analyzed and, if necessary, restructured in order to achieve several desirable qualities.

4.3. Existing System Approaches:

Approaches to schema integration can be broadly classified as ones that exploit either just schema information or schema and instance level information

- **Schema-level matchers**

Only consider schema information, not instance data. The available information includes the usual properties of schema elements, such as name, description, data type, relationship types (part-of, is-a, etc.), constraints, and schema structure. Workings at the element (atomic elements like attributes of objects) or structure level (matching combinations of elements that appear together in a structure), these properties are used to identify matching elements in two schemas. Language-based or linguistic matchers use names and text (i.e., words or sentences) to find semantically similar schema elements. Constraint based matchers exploit constraints often contained in schemas. Such constraints are used to define data types and value ranges, uniqueness, optionality,

relationship types and cardinalities, etc. Constraints in two input schemas are matched to determine the similarity of the schema elements.

- **Instance-level matchers**

  Use instance-level data to gather important insight into the contents and meaning of the schema elements. These are typically used in addition to schema level matches in order to boost the confidence in match results, more so when the information available at the schema level is insufficient. Matchers at this level use linguistic and constraint based characterization of instances.

- **Hybrid matchers**

  Directly combine several matching approaches to determine match candidates based on multiple criteria or information sources.

Various Schema matching approaches

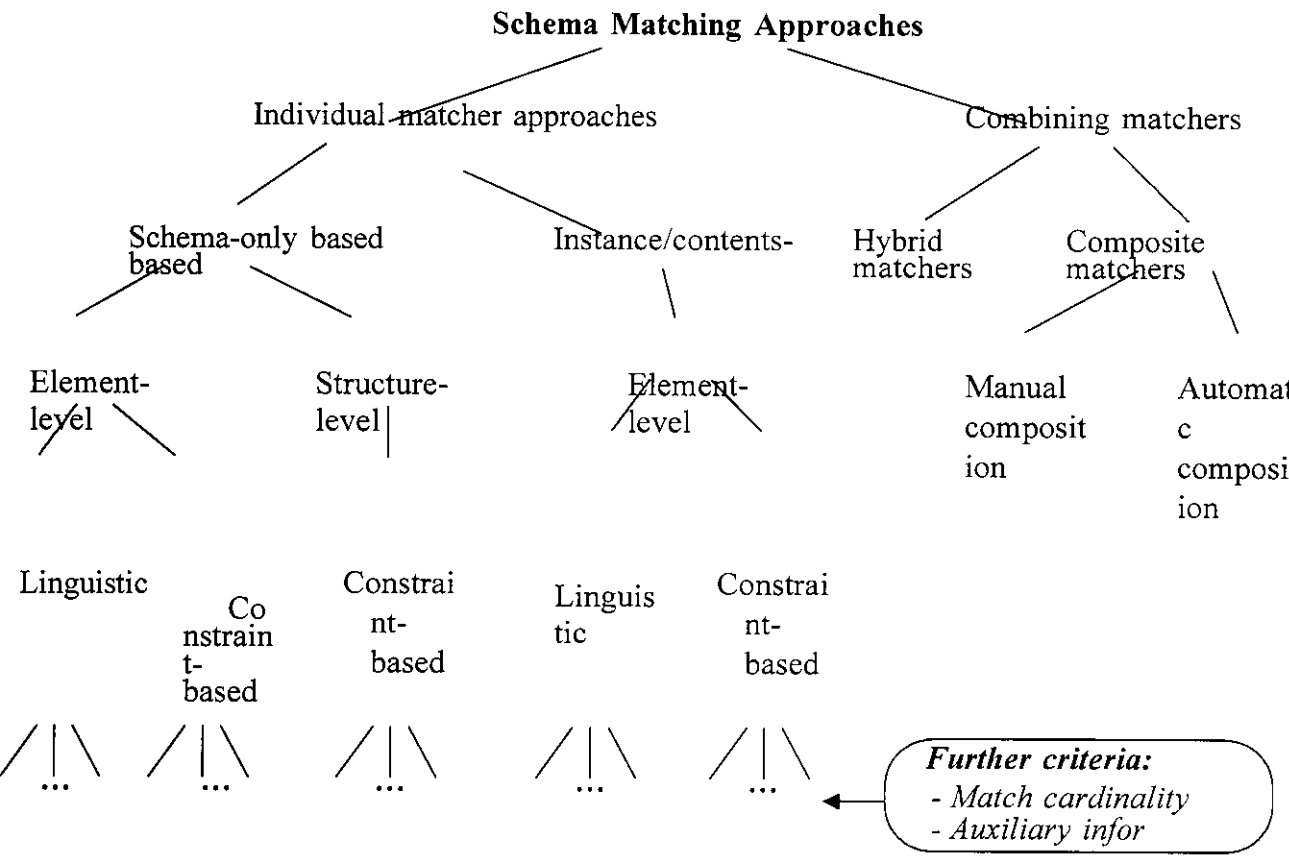


Fig 4.3. Classification of schema matching approaches

CHAPTER 5

PROPOSED SYSTEM

Schema matching is one of the key challenges in information integration. It is a labor intensive and time-consuming process. To alleviate the problem, many automated solutions have been proposed. Most of the existing solutions mainly rely upon textual similarity of the data to be matched. However, there exist instances of the schema-matching problem for which they do not even apply. Such problem instances typically arise when the column names in the schemas and the data in the columns are opaque or very difficult to interpret. In the current work, we have only tested two simple distance metric Euclidean and normal. Distance metric learning is an emerging area of statistical learning in which the goal is to induce a more powerful distance metric from labeled examples. We use in the proposed System; Mahalanobios distances are computed by linearly transforming the input space, then computing Euclidean distances in the transformed space. A well-chosen linear transformation can improve classification by decorrelating and reweighting elements of the feature vector. In fact, significant improvements will be observed. The technique that is in the existing system is an instance-based technique. It emphasize that claims is that this technique, is not the best of techniques to apply as a useful addition to a suite of automated schema mapping tools and proposing a new usage-based schema matching technique. The proposed technique exploits the usage information of the attributes in the query logs to find matches, in contrast to relying on the schema information or the data instances. This may be the only option for schema matching if the information needed by the two other techniques is not available or not reliable enough to achieve good matching quality. The

existing methods for weighted graph matching Algorithms for schema matching is compared with the new proposed methods and the performance and accuracy of the schema matching of the various techniques are compared and analyzed.

5.1. Objectives:

- The main objective of the project is to improve accuracy in the schema matching problem.
- To implement the new techniques of Graph matching algorithms and obtaining a better integration of the various attributes of the tables and databases and to bring about smooth and accurate database integration without any inappropriate data left unattended in middle without any integrity.

5.2. Advantages of Proposed System:

- Using Mahalanobios distances are computed by linearly transforming the input space, then computing Euclidean distances in the transformed space. A well-chosen linear transformation can improve classification by decorrelating and reweighting elements of the feature vector. In fact, significant improvements will be observed.
- An advantage of using dependency relations in schema matching is that this approach does not require data interpretation; that is, even if the data sets in the schemas to be matched use different encodings, we can still measure the dependency relations.
- As a result, the proposed matching technique can be applied to multiple unrelated domains without retraining or customization.

- By referring to matching techniques that are not dependent of data interpretation as uninterrupted matching, and make this precise in the next definition.
- Introducing a new criterion, data interpretation, in classifying schema-matching techniques. Along with structural similarity, we classify schema-matching techniques into four categories. Using this classification, we identify a new problem class that has not been addressed by existing techniques
- It is applicable to match complete schemas, rather than individual tables.

CHAPTER 6

IMPLEMENTATION OF PROPOSED SYSTEM

6.1. List of Modules:

- Modeling new Dependency Relation
- Implementing Enhanced Weighted Graph Matching algorithms for schema matching
- Implementing proposed new Graph Matching algorithms for schema matching
- Performance Comparison

6.1.1. Modeling New Dependency Relation:

The dependency graph function produces such dependency graphs by calculating the pair wise mutual information over all pairs of attributes in a table and structuring them in an undirected labeled graph.

The Entropy of X, H(X) is defined by:

$$H(X) = -E_X [\log (P(X))]$$

Let:

- X is a random variable (R.V).
- P(X) is the probability distribution of X.

- P(x) is the probability density of X.

ormally, the Mutual Information of two discrete random variables X and Y can be defined as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p_1(x)\, p_2(y)} \right).$$

here $p(x,y)$ is the joint probability distribution function of X and Y, and $p_1(x)$ and $p_2(y)$ are the arginal probability distribution functions of X and Y respectively.

ormally, the Euclidean Distance Metric for graphs A and B as:

$$D_M^U(A,B) = \sqrt{\sum_{i,j} (a_{ij} - b_{m(i)m(j)})2}$$

1.2. Implementing Enhanced Weighted Graph Matching algorithms for chema matching:

Hill Climbing Approach:

Hill climbing is a simple nondeterministic, iterative improvement algorithm. The C algorithm is simply a loop that moves, in each state transition, to a state where the most provement can be achieved. A state represents a permutation that corresponds to a mapping tween the two graphs. We limit the set of all states reachable from one state in a state nsition, to a set of all permutations obtained by one swapping of any two nodes in the rmutation corresponding to the current state. The algorithm stops when there is no next state

available that is better than the current state. As we can see, it is nondeterministic; depending on where it starts, even for the same problem, the final states may differ. To avoid being stuck in a local minimum after an unfortunate run, the usual practice is to perform some number of random restarts.

Hill Climbing Algorithm:

function Hill-Climbing (*problem*) **returns** a solution state

inputs: *problem* // a problem.

local variables: *current* // a node.

next // a node.

current ← Make-Node (Initial-State [*problem*]) // make random

loop do // initial state.

next ← a highest-valued successor of *current*

if Value [next] < Value [current] **then return** *current*

current ← *next*

end

6.1.3. Implementing proposed new Graph Matching algorithms for schema matching:

Vector Fitting2 (VF2) Algorithm:

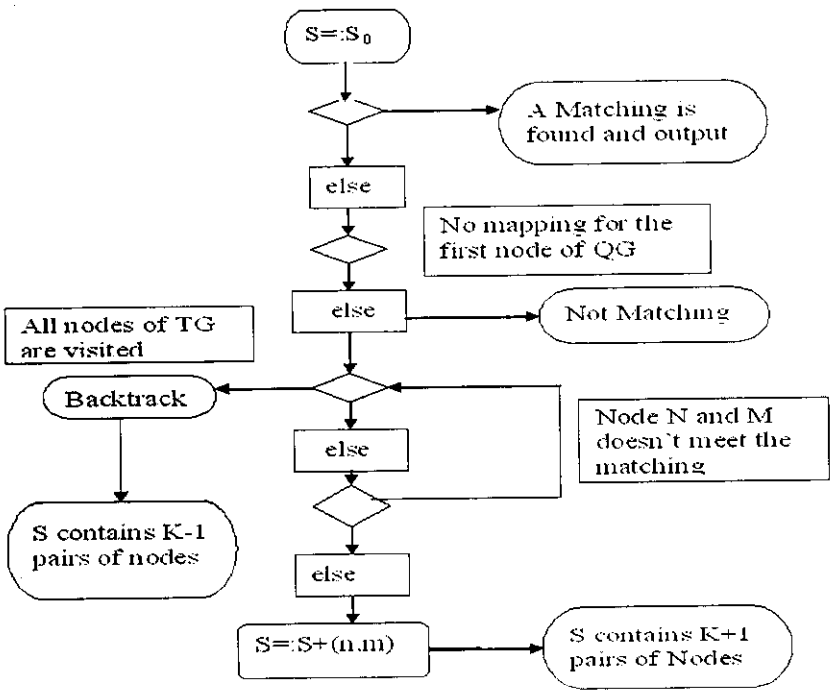


Fig 6.1.3. Flow Chart for VF2 Algorithm

Table 2. Shows the sample student database

Studname	Gender	Rollno	Address	Degree	Grade	Major	age
Kumar	Male	100	Trichy	B.E	A	CSE	22
Priya	Female	101	Tirupur	M.D	B	ECE	23
Raj	Male	102	Salem	M.E	C	ECE	24
Kani	Female	103	Pollachi	M.B.A	D	CSE	27
Ram	Male	104	Madurai	M.C.A	E	CAD	22
Maha	Female	105	Theni	M.E	A	ECE	25
Vijay	Male	106	Namakkal	M.E	C	CSE	26

Table 2. Sample Student database

Table 3. Shows the corresponding Mutual information and Euclidean distance values

Columns	Table 1 MI Calculation	Table 1 MI Calculation	Euclidean Distance
01	0.3812309493079676 4	0.3812309493079676 4	0.0
02	2.8662657447607374	2.8662657447607374	0.0
03	2.9957322735539833	2.9957322735539833	0.0
04	2.5103550389677 73	2.5103550389677 73	0.0
05	2.48727356247666 7	2.9957322735539833	0.5084587110773162
06	2.9957322735539833	2.9957322735539833	0.0
07	2.930999009157 36	2.930999009157 36	0.0
12	7.3160031087656 52	7.3160031087656 52	0.0
13	7.0587277189907 01	7.0587277189907 01	0.0
14	8.9219281513711 27	8.9219281513711 27	0.0
15	9.4808659187743 38	7.0587277189907 01	2.4221381997836 37
16	7.0587277189907 01	7.0587277189907 01	0.0
17	7.1191651690529 17	7.1191651690529 17	0.0
23	3.1573875605959083	3.1573875605959083	0.0

Total Euclidean Distance: 5.617

Fig 6.1.3.1. Shows the Dependency Graph for the corresponding student database

Dependency Graph:

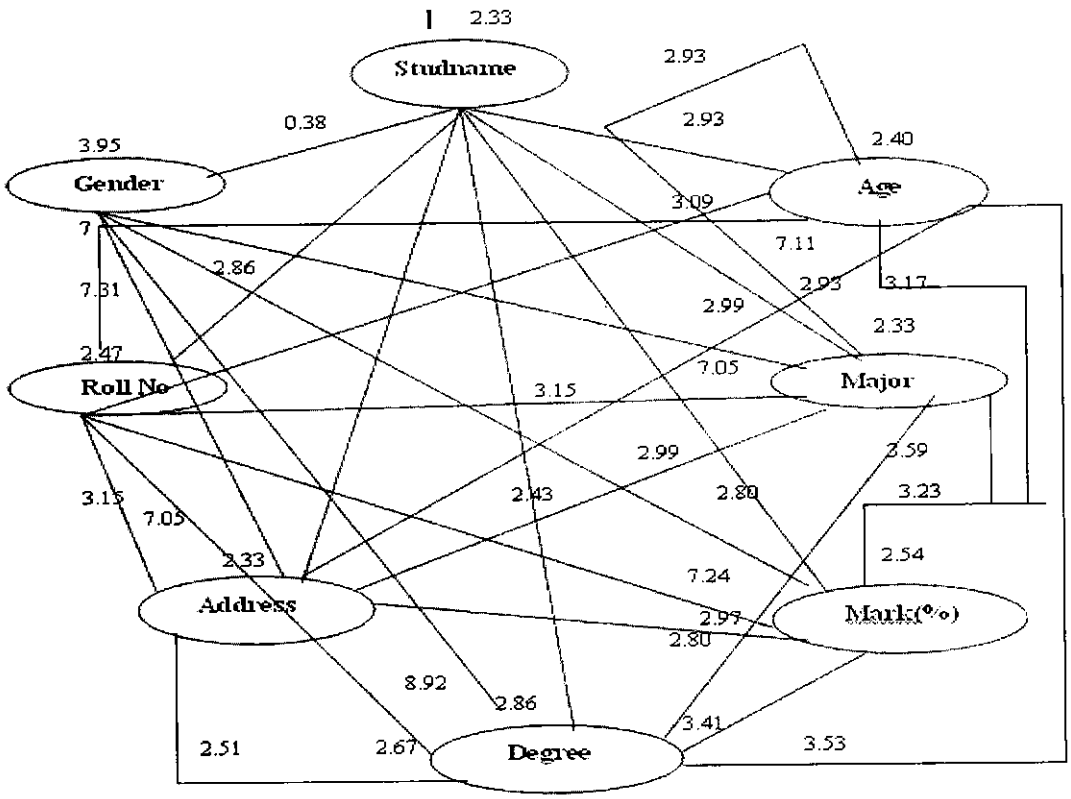


Fig 6.1.3.1. Dependency Graph

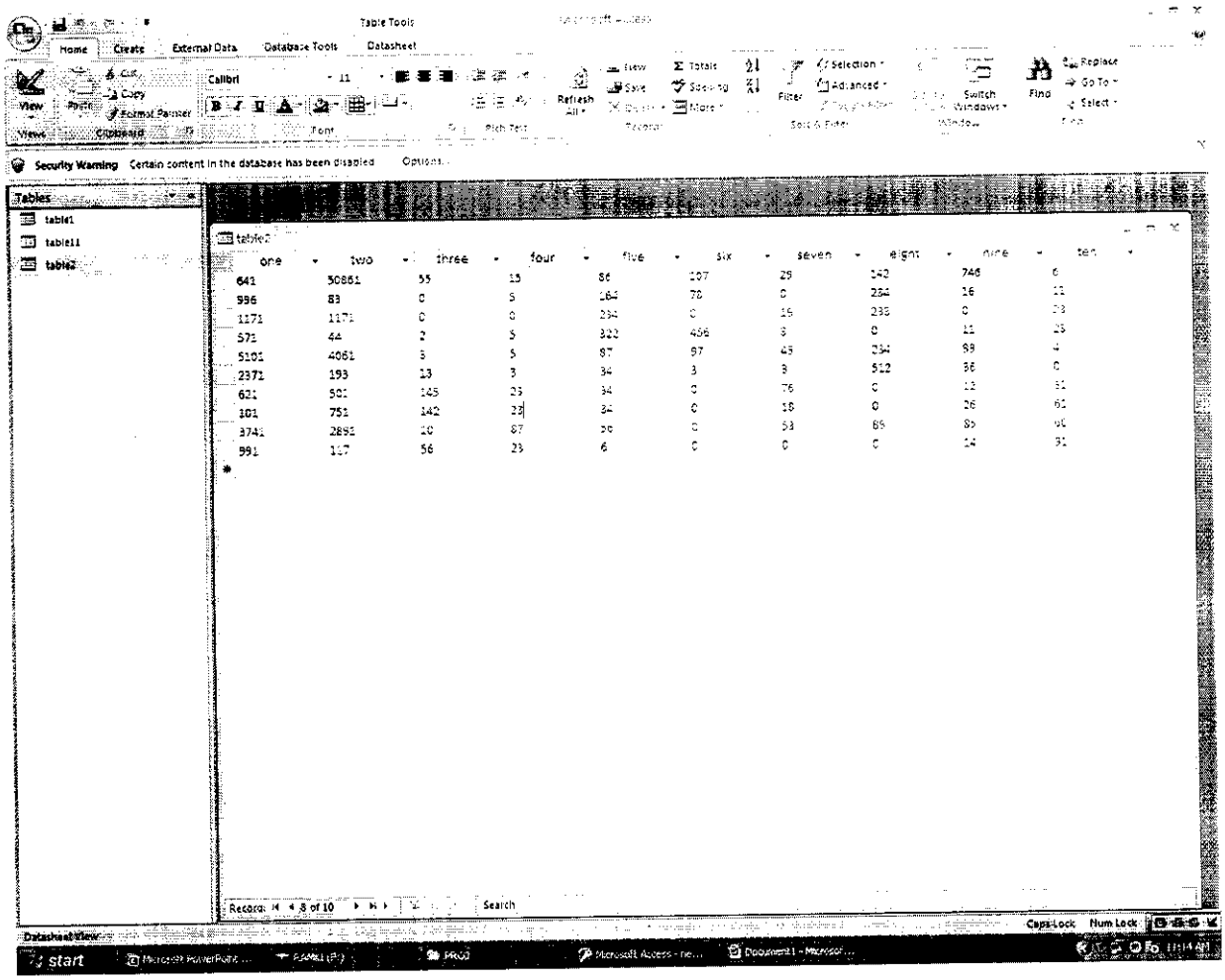
6.1.4. Performance Comparison:

Thus the proposed VF2 algorithm provides more accurate schema matching than existing HC algorithm by consuming less time and less error factor. But HC approach provides most promising results while VF2 is many times faster than HC and takes the advantage.

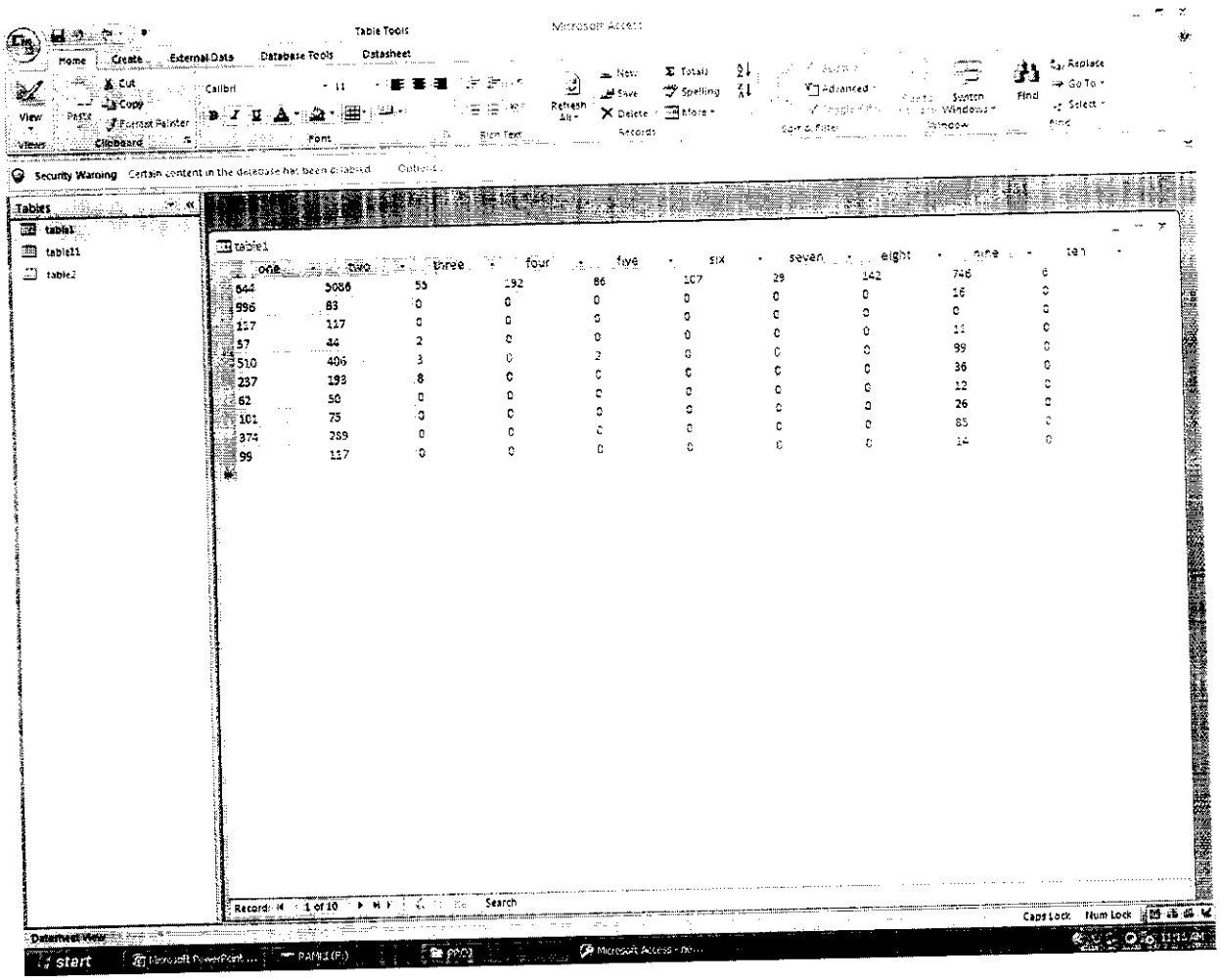
CHAPTER 7

EXPERIMENTAL RESULTS

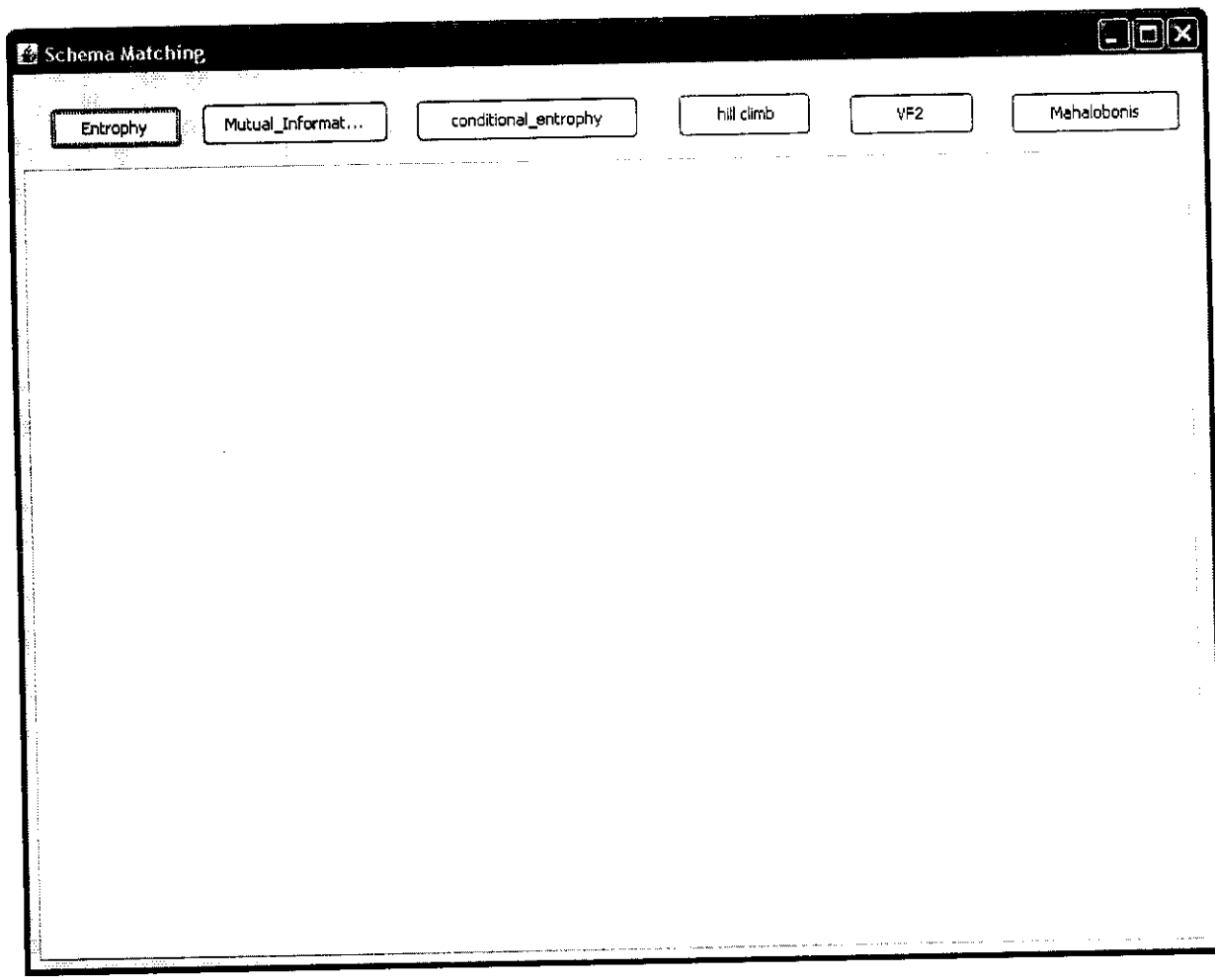
7.1 INPUT TABLE1 WITH OPAQUE COLUMN VALUES



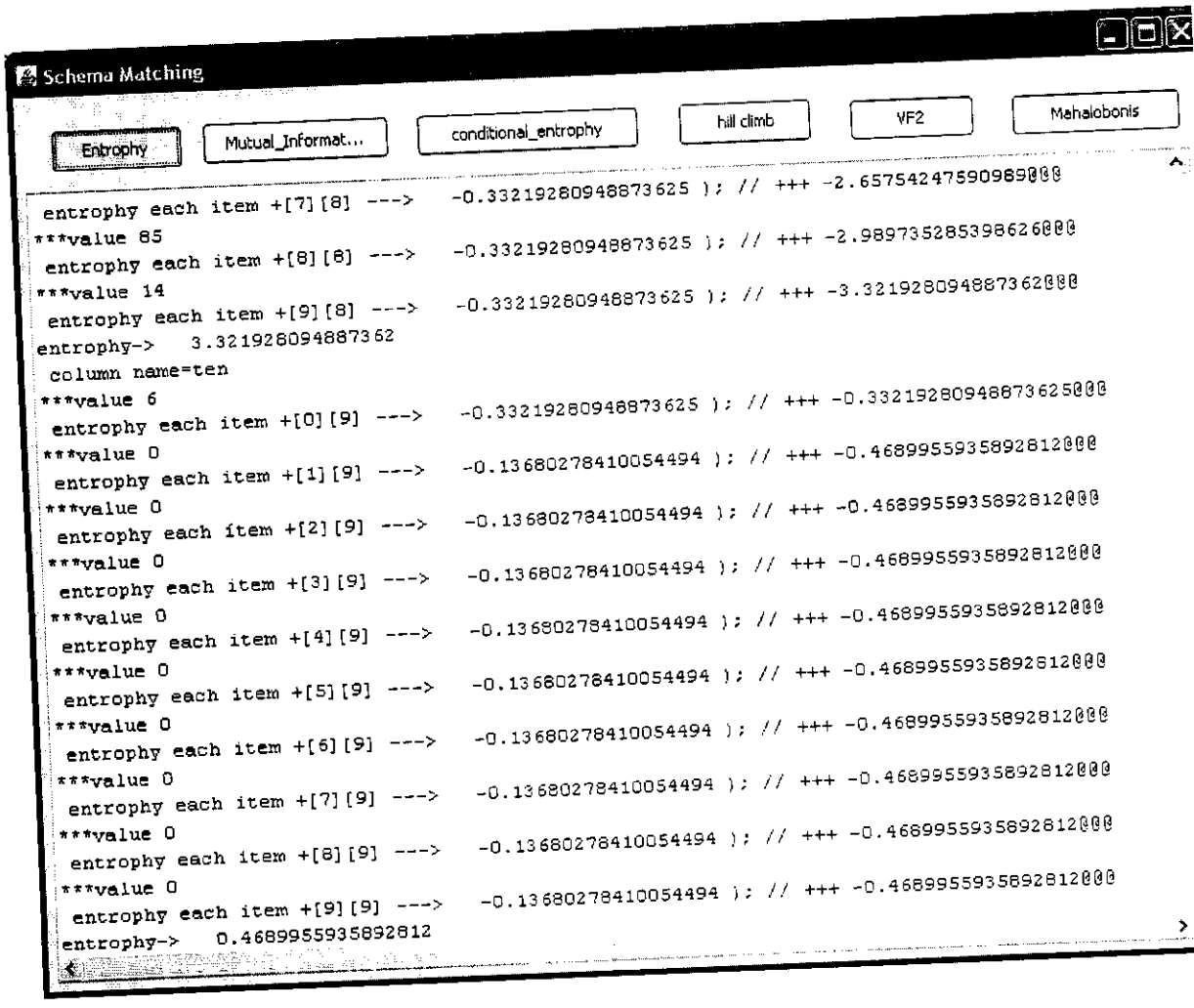
7.2 INPUT TABLE2 WITH OPAQUE COLUMN VALUES



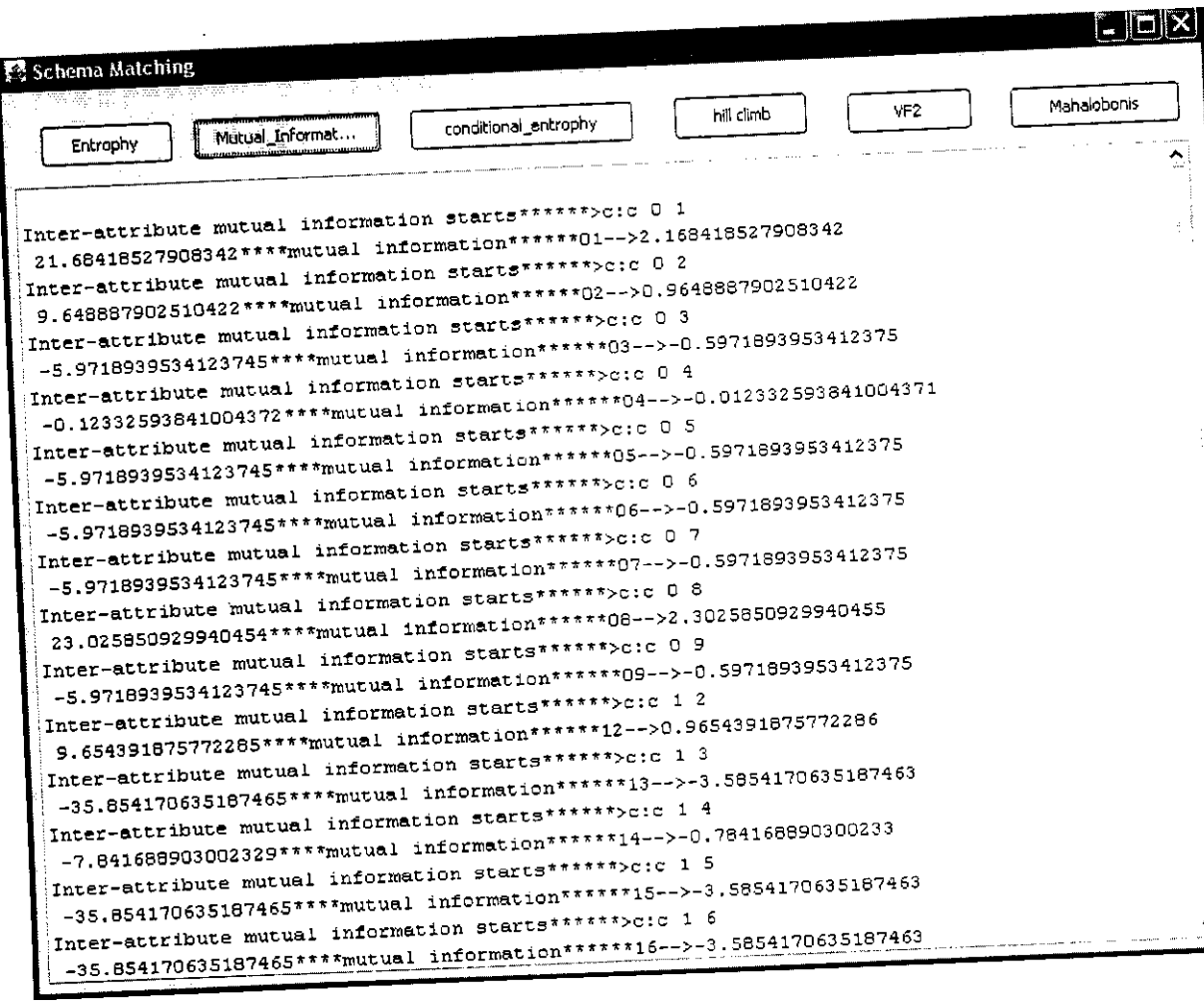
7.3 OUTPUT SCREEN FOR SCHEMA MATCHING



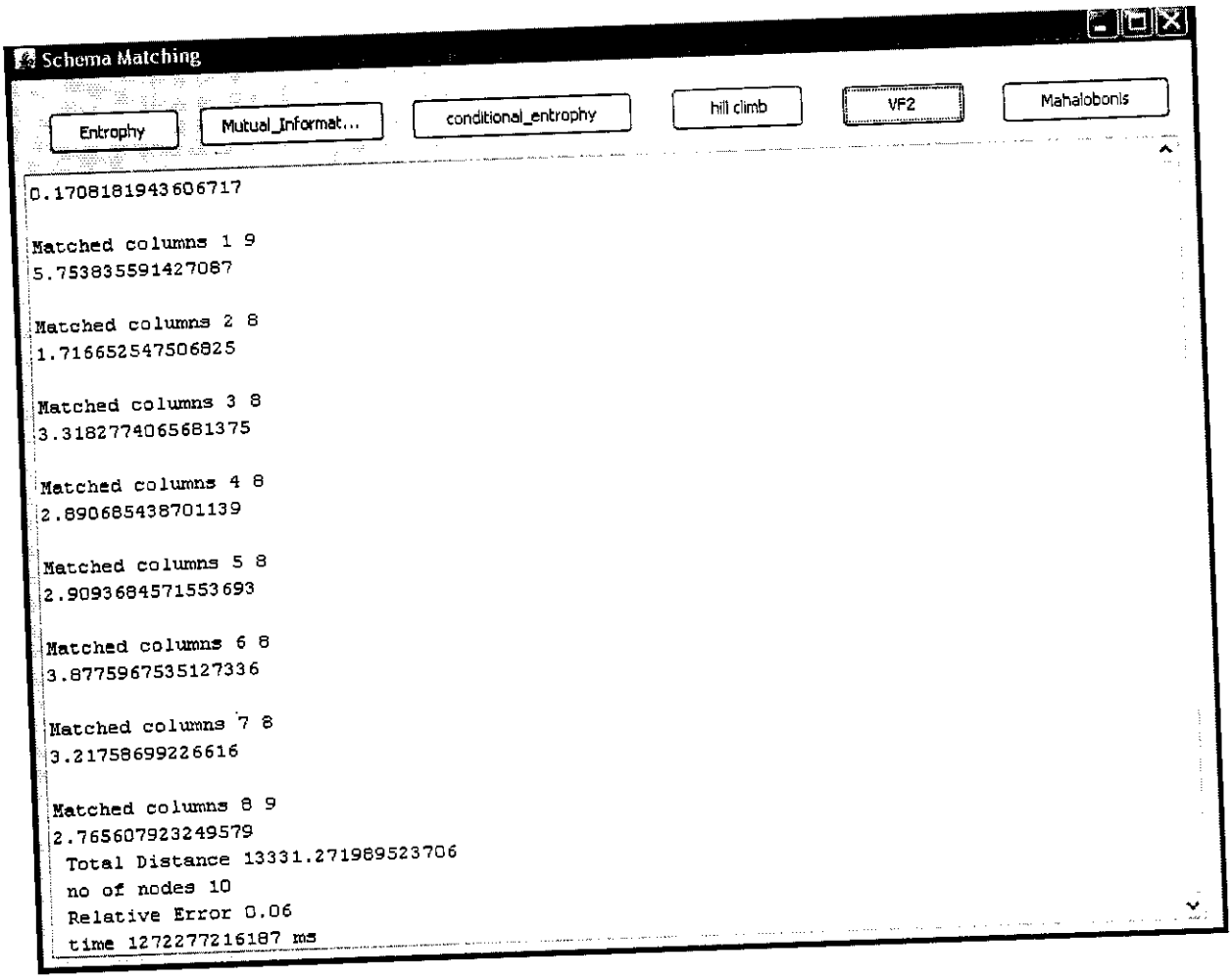
7.4 RETRIEVING ENTROPY VALUES



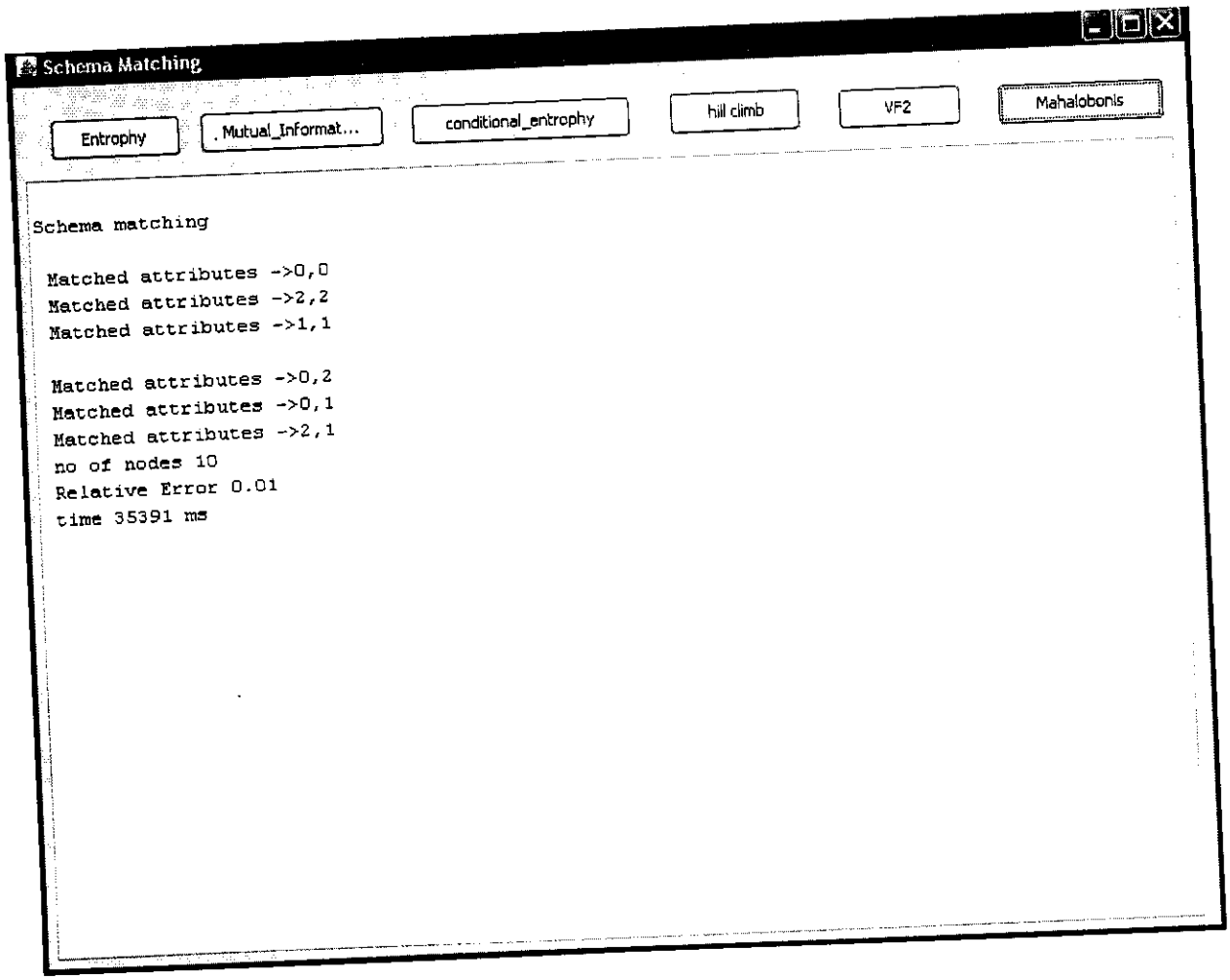
.5 RETRIEVING MUTUAL INFORMATION VALUES



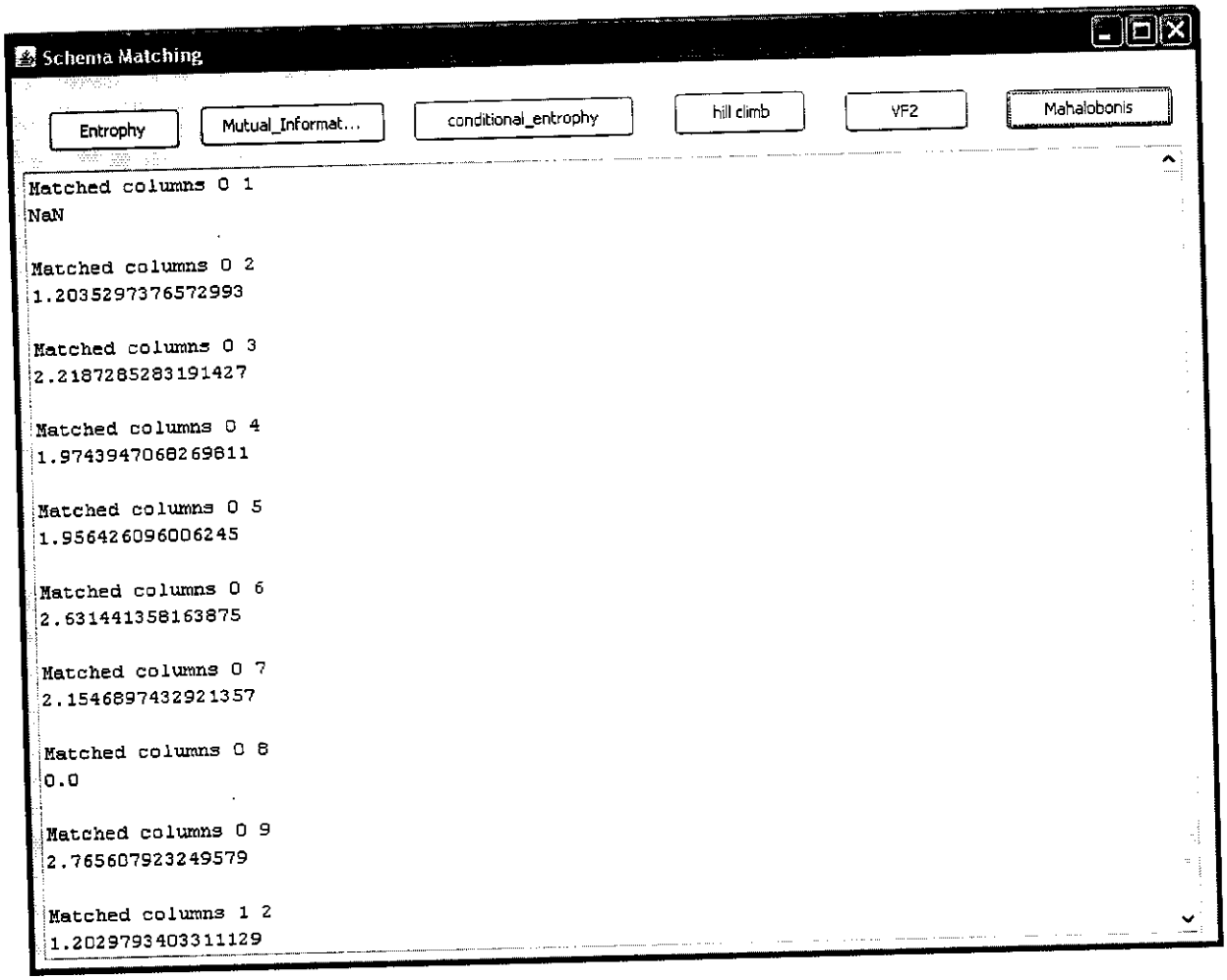
7.6 MATCHING COLUMNS FOR HC APPROACH



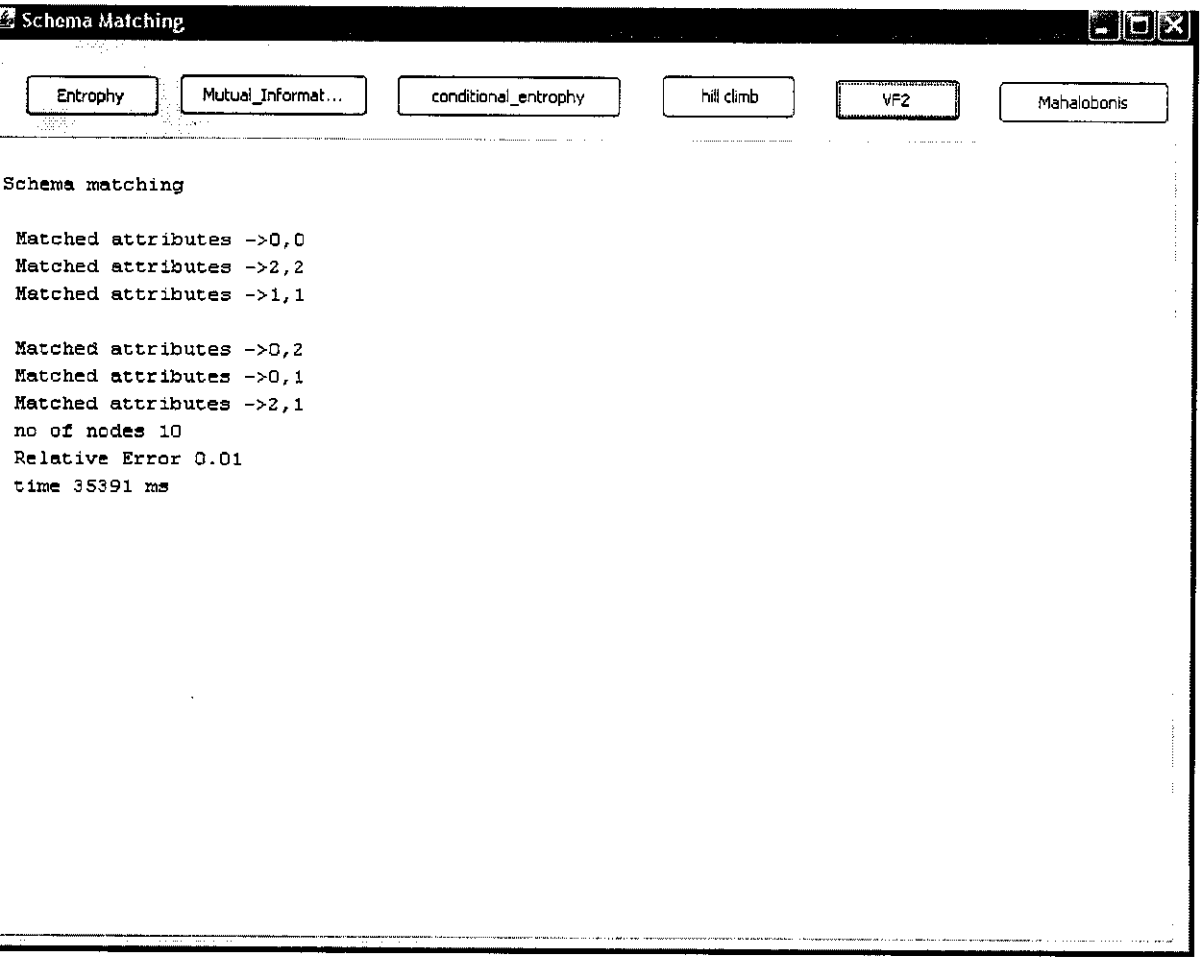
7.7 SCHEMA MATCHING FOR HC APPROACH



7.8 MATCHING COLUMNS FOR VF2 APPROACH



7.9 SCHEMA MATCHING FOR VF2 APPROACH



CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION:

The proposed two-step schema-matching technique works even in the presence of opaque column names and data values. In the first step, it is been measured with pair wise attribute correlations in the tables to be matched and constructed dependency graph using mutual information as a measure of the dependency between attributes. In the second stage, it has been found by matching node pairs across the dependency graphs by running a graph-matching algorithm. This work is the first to introduce an uninterpreted matching technique utilizing inter attribute dependency relations. It has been shown that while a single column uninterpreted matching such as entropy only matching can be somewhat effective alone, further improvement is possible by exploiting inter attribute correlations.

By investigating approximation algorithms for the matching problem and showed that an efficient implementation can be possible for this approach. Among the algorithms evaluated, the HC approach showed the most promising results. It is found close to optimal solutions very quickly, suggesting that the graph matching problems arising in our schema-matching domain are amenable to HC.

8.2 FUTURE ENHANCEMENTS:

A good deal of room for future work exists. In this work, it is been tested by only two simple distance metrics, Euclidean and normal. It is possible that more sophisticated distance metrics could produce better results. It would also be interesting to evaluate other dependency models using different uninterpreted methods which would produce more accurate schema match.

APPENDICES

CODE FOR HILL CLIMBING ALGORITHM:

```
import java.sql.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class hillclimb
{

        Vector ccol=null; long start=0;

String mat[][]=null; String ecount[][]=null; double fff[]= new double[5];

String sprob[][]=null;//= new String[1000];  //prob of each item
double finalsum[]= new double[10];   //entrophy
Vector col= new Vector(); Vector col2= new Vector();
nt count=0;  double mff=0.0;
va.util.// Hashtable ht1 = new java.util.Hashtable();
```

```
Vector<Double> ht1 =new Vector<Double>(); //mutual information1
Vector<Double> ht2 =new Vector<Double>(); //mutual information2
Vector hp =new Vector();
Random rrac = new Random();
String tname="table1",tname2="table2";
Main sc =null;
public  hillclimb (Main sc ) throws Exception
{ this.sc=sc;

try
{ no1();
Connection theConn=null; Statement stmt12=null;
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
theConn = DriverManager.getConnection("jdbc:odbc:new");
stmt12 = theConn.createStatement();

      ResultSet r12 = stmt12.executeQuery("SELECT COUNT(*) AS rowcount FROM
"+tname+"");
      r12.next();

      count = r12.getInt("rowcount") ;
     r12.close() ;
     stmt12.close();
```

```
            theConn.close();
            mat    = new String[count][10];
            ecount =new String[count][10];
            sprob= new String[count][10];

} catch(Exception ex1)
{       //System.err.println("Exception: " + ex1.getMessage());
}

        String result = null;
                        String selectString;  String url = "jdbc:odbc:new";

            try {
        String className = "sun.jdbc.odbc.JdbcOdbcDriver";
                                Class.forName(className);
      Connection con = DriverManager.getConnection(url);

                        Statement stmt=null;
                       selectString = "select * from "+tname+"";

                                stmt = con.createStatement();
```

```
                    ResultSet rs = stmt.executeQuery(selectString);
        int i=0;
              while (rs.next())
              {
 //  String name = rs.getString(1);

for(int j=0;j<8;j++)
                    {
                    mat[i][j]=new String(rs.getString(j+1));
              //    System.out.print("\t"+mat[i][j]);
                    }     i++;


            } rs.close();
                  stmt.close();
                  con.close();

            } catch(Exception ex) {
                  //System.err.println("Exception: " + ex.getMessage());
            }
```

```
 try
                    {


//int sum12[]=0;


//String
name[]={"studfname","studlname","rollno","major","grade","age","gender","degree","level","ad
dress"};


{Connection theConn=null; Statement stmt12=null;
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    theConn = DriverManager.getConnection("jdbc:odbc:new");
                    stmt12 = theConn.createStatement();

        ResultSet rsColumns = null;
        DatabaseMetaData meta = theConn.getMetaData();
        rsColumns = meta.getColumns(null, null, tname, null);


   while (rsColumns.next())
   {
   String columnName = rsColumns.getString("COLUMN_NAME");
         col.add(columnName);
```

```
                } theConn.close(); rsColumns.close();
                 stmt12.close();

}

{Connection theConn=null; Statement stmt12=null;
                 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                 theConn = DriverManager.getConnection("jdbc:odbc:new");
                 stmt12 = theConn.createStatement();

      ResultSet rsColumns = null;
      DatabaseMetaData meta = theConn.getMetaData();
      rsColumns = meta.getColumns(null, null, tname2, null);

 while (rsColumns.next())
 {
 String columnName = rsColumns.getString("COLUMN_NAME");
      col2.add(columnName);


          } theConn.close(); rsColumns.close();
           stmt12.close();
```

```
}

for(int i=0;i<7;i++)
{
 Connection theConn=null; Statement stmt12=null;
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                    theConn = DriverManager.getConnection("jdbc:odbc:new");
                    stmt12 = theConn.createStatement();

        ResultSet rsColumns = null;
        DatabaseMetaData meta = theConn.getMetaData();
        rsColumns = meta.getColumns(null, null, tname, null);

   //while (rsColumns.next())
   {
for(int j=0;j<count;j++)
        { rsColumns.next();

  String columnName = rsColumns.getString("COLUMN_NAME");
       // col.add(columnName);
   // sc.jTextArea1.append("\n column name=" + columnName);

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

```
                theConn = DriverManager.getConnection("jdbc:odbc:new");

                stmt12 = theConn.createStatement();



        ResultSet r12 = stmt12.executeQuery("SELECT COUNT(*) AS rcount FROM
"+tname+" where "+columnName+"="'+mat[j][i]+"'");

                //      System.out.print("\t"+mat[i][j]+"  ");

                        r12.next();

                        int countt = r12.getInt("rcount") ;

                        // ecount[j][i]=""+countt;


                double cnt1=countt; double cnt2=count;

               //sprob[i][j]=""+(cnt1/cnt2)*Math.log(cnt1/cnt2);


                        r12.close() ;

                        stmt12.close();


        }//for

//      System.out.print("\n");

  }//while

       }//for

 } catch(Exception ex1)

  {

        //System.err.println("Exception: " + ex1.getMessage());
```

```
 // ex1.printStackTrace();
        }

//finalsum[i]= new  String[];

/*for(int i=0;i<finalsum.length;i++)
{
  System.out.print("\nentrophy  column  "+i+" "+ finalsum[i]);
}*/
//for(int i=1;i<2;i++)
int tablee=0;
ccol= new Vector();
while(tablee<2)
{
for(int i=1;i<=col.size();i++)  //column
{
        //System.in.read();
        //for(int j=i+1;j<i+2;j++)
        for(int j=i+1;j<=col.size();j++)
{ //System.in.read();
        //new
Vector vv1 = new Vector();
```

```
Vector vv2 = new Vector();

Vector vv = new Vector();

// new hillclimb();

}

}
```

CODE FOR VF2 ALGORITHM:

```
import java.sql.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class vf2
{
 int nodes=8;
 Vector xfirst =new Vector();
 Vector xsec =new Vector();
 vf2alg vf2=null;
 vf2(vf2alg vf2,Main sc )
 {  this.vf2=vf2;
   for(int i=0;i<nodes;i++)
   for(int j=i+1;j<nodes;j++)
   for(int k=0;k<nodes;k++)
   {
       if(i!=k && j!=k)
      {
                    System.out.println(" i j k " +i+" "+j+" "+k);
```

```
        xfirst.add(""+i+"@"+j+"@"+k);
        xsec.add(""+i+"@"+j+"@"+k);
            }
    }
     for(int i=0;i<nodes;i++)
     {
              for(int j=i+1;j<nodes;j++)
              {
                       Vector<String> atom1= new Vector<String>();
                       Vector<String> atom2= new Vector<String>();

    //System.out.println(" i j k " +i+" "+j+" "+k);
    String graphf[]=        (""+xfirst.get(i)).split("@");
    String graphs[]=        (""+xsec.get(i)).split("@");

    atom1.add(graphf[0]+","+graphf[1]);
    atom1.add(graphf[0]+","+graphf[2]);
    atom1.add(graphf[1]+","+graphf[2]);

    atom2.add(graphs[0]+","+graphs[1]);
              atom2.add(graphs[0]+","+graphs[2]);
    atom2.add(graphs[1]+","+graphs[2]);
```

```
String disp= isomorphicgraph(atom1,atom2);
System.out.println(""+disp);
sc.jTextArea1.setText("\nSchema matching");

if(disp .equalsIgnoreCase( "matched"))
 {
  sc.jTextArea1.append("\n \n Matched attributes ->"+graphf[0]+","+graphs[0]);
  sc.jTextArea1.append("\n Matched attributes ->"+graphf[1]+","+graphs[1]);
  sc.jTextArea1.append("\n Matched attributes ->"+graphf[2]+","+graphs[2]);
  sc.jTextArea1.append("\n \n Matched attributes ->"+graphf[0]+","+graphs[1]);
  sc.jTextArea1.append("\n Matched attributes ->"+graphf[0]+","+graphs[2]);
  sc.jTextArea1.append("\n Matched attributes ->"+graphf[1]+","+graphs[2]);

        /*      sc.jTextArea1.append("\n -Matched attributes ->"+graphs[1]);
  sc.jTextArea1.append("\n -Matched attributes ->"+graphs[1]);
                sc.jTextArea1.append("\n -Matched attributes ->"+graphs[1]);*/
          }
    }
     }
}
public  String isomorphicgraph(  Vector<String> atom1 ,  Vector<String> atom2)
{
```

```
int atomatch=0;  int isoshare=0;
double fa=(Double) vf2.isomorp1.get(atom1.get(0));
double fs=(Double) vf2.isomorp2.get(atom2.get(0));
if(fa<=fs)
isoshare++;
System.out.println("1-> "+fa+" , "+fs);
fa=(Double) vf2.isomorp1.get(atom1.get(1));
fs=(Double) vf2.isomorp2.get(atom2.get(1));
if(fa<=fs)
isoshare++;
System.out.println("2-> "+fa+" , "+fs);

fa=(Double) vf2.isomorp1.get(atom1.get(2));
fs=(Double) vf2.isomorp2.get(atom2.get(2));
if(fa<=fs)
isoshare++;
System.out.println("3-> "+fa+" , "+fs);



System.out.println("-->"+isoshare);
if(isoshare>=1)
{
atomatch=1;
```

```
    System.out.println(""+vf2.isomorp1.get(atom1.get(0))+" "+vf2.isomorp2.get(atom2.get(0)) );


    };
    if(atomatch==1)
    return "matched";
    else
    return "not";


  }
  public static void main(String args[])
  {
   //new vf2();
  }
}
```

CODE FOR DATABASE CONNECTION:

```
import java.sql.*;
public class database_conn
{
 Connection con=null;
 Statement stat=null;
 Statement stat1=null;
 Statement stat2=null;
 Statement stat3=null;
 Statement stat4=null;
 Statement stat5=null;
 Statement st=null;
 ResultSet result=null;
 ResultSet result1=null;
 //ResultSet rs,rs1,rs2;
 public database_conn()
 {
        try
        {
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         con=DriverManager.getConnection("jdbc:odbc:new");
```

```
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDAT
ABLE);

stat=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDA
TABLE);

stat1=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD
ATABLE);

stat2=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD
ATABLE);

stat3=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD
ATABLE);

stat4=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD
ATABLE);

stat5=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPD
ATABLE);

    }

    catch(Exception e)

    {

            System.out.println("Error in database_conn.java "+e);

        }

}
```

REFERENCES

[1] Jaewoo Kang, Jeffrey F.Naughton, "Schema Matching Using Inter attribute Dependencies", IEEE Transactions on knowledge and data engineering, Vol. 20, No. 10, October 2008.

[2] H.A. Almohamad and S.O.Duffuaa, "A Linear Programming Approach for the Weighted Graph Matching Problem", IEEEE Transactions on pattern analysis and machine intelligence, Vol. 15, No. 5, May1993.

[3] S.Gold, A.Rangarajan, "A Graduated Assignment Algorithm for Graph Matching", IEEE Transactions on pattern analysis and machine intelligence, Vol. 18, No. 4, April 1996.

[4] B.Chang, J.Han, "Discovery Complex Matching Across Web Query Interfaces: A Correlation Mining Approach", IEEE Transactions on KDD, Vol. 17, No. 6, June 1999.

[5] J.Kang and J.F.Naughton, "On Schema Matching with opaque column Names and Data Values", Proc.ACM SIGMOD '03, June 2003.

[6] E.Rahm and P.A.Bernstein, "On Matching Schemas Automatically", The VLDB J, vol.10, No 4.December 2001.

[7] Hazem Elmeleegy', Mourad Ouzzani2, and Ahmed Elmagarmid' 2 Usage-Based Schema Matching, ICDE 2008.

[8] S.Castano, V.Antonellis, Vimercati, "Global Viewing of Heterogeneous Data Sources", IEEE Transactions on knowledge and data engineering, Vol. 13, No. 2, March 2001.

[9] T. Milo and S. Zohar, "Using Schema Matching to Simplify Heterogeneous Data Translation", Proc.of the 24th VLDB Conference, 1998.

[10] A. Doan, P. Domingos, A.Y. Halevy, "Reconciling Schemas of Disparate Data Sources: A Machine –Learning Approach", Proc. ACM SIGMOD, 2001.