1-2006

Discovery Agent : An Interactive Approach for the Discovery of Inclusion Dependencies

Dhaval B. Patel

MONTCLAIR STATE UNIVERSITY

Discovery Agent: An Interactive Approach for the Discovery of Inclusion Dependencies

by

Dhaval B. Patel

A Master's Thesis Submitted to the Faculty of

Montclair State University

In Partial Fulfillment of the Requirements

For the Degree of

Master of Science

January 2006

School College of Science and Mathematics
Department Computer Science

[redacted]

Robert Prezant, Ph.D.
Dean, College of Science and Mathematics

11/14/05
(Date)

Thesis Committee:

[redacted]

Andreas Koeller, Ph.D.
Thesis Sponsor

[redacted]

Katherine Herbert, Ph.D.
Committee Member

[redacted]

John Jenq, Ph. D.
Committee Member

[redacted]

Dorothy Deremer, Ph.D.
Department Chairperson

Discovery Agent: An Interactive Approach for the Discovery of Inclusion Dependencies

Dhaval Patel
Department of Computer Science, Montclair State University
Upper Montclair, NJ 07043, USA

Abstract

The information integration problem is a hard yet important problem in the field of databases. The goal of information integration is to provide unified views on diverse data among several resources. This subject has been studied for a long time. The integration can be performed using several ways. Schema integration using inclusion dependency constraints is one of them. The problem of discovering inclusion dependencies among input relations is NP-complete in terms of the number of attributes.

Two significant algorithms address this problem: *FIND*$_2$ by Andreas Koeller and *Zigzag* by Fabien De Marchi. Both algorithms discover inclusion dependencies among input relations on small scale databases having relatively few attributes. Because of the data discrepancy, they do not scale well with higher numbers of attributes.

We propose an approach of incorporating human intelligence into the algorithmic discovery of inclusion dependencies. To use human intelligence, we design an agent called the discovery agent, to provide a communication bridge between an algorithm and a user. The discovery agent demonstrates the progress of the discovery process and provides sufficient user controls to govern the discovery process into the right direction. In this thesis, we present a prototype of the discovery agent based upon the *FIND*$_2$ algorithm, which utilizes most of the phase-wise behavior of the algorithm and demonstrate how human observer and algorithm work together to achieve higher performance and better output accuracy.

The goal of the discovery agent is to make the discovery process truly interactive between system and user as well as to produce the desired and accurate result. The discovery agent can deliver an applicable and feasible approximation of an NP-complete problem with the help of suitable algorithm and appropriate human expertise.

DISCOVERY AGENT: AN INTERACTIVE APPROACH FOR THE DISCOVERY OF INCLUSION DEPENDENCIES

by
DHAVAL B PATEL

A THESIS
Submitted in partial fulfullment of the requirements
For the degree of Master of Science in
The Department of Computer Science in
the Graduate Program of
Montclair State University
January 2006

Acknowledgements

I would like to thank my advisor, Dr. Andreas Koeller, for his valuable guidance, indispensable advice and eminent support during this thesis at Montclair State University. I would also like to thank my dissertation committee members Dr. Katherine Herbert and Dr. John Jenq for their support and benevolence towards the completion.

I express sincere appreciation to Dr. Dorothy Deremer, the Chairperson of the Department of Computer Science at Montclair State University for her kind help. I also wish my thanks to Beverly Macaluso for her unforgettable help during my stay here at Montclair State University.

Finally I want to thank my sister and brother for their relentless help and support towards my entire Masters at Montclair State University.

Contents

List of Figures

1. Introduction

The amount of information in the world is rapidly increasing today. While storage of a huge amount of information itself is a problem, we are also facing the problem of integration of one or more information sources.

In a large organization, there are many different subdivisions which often produce and store their data using different database management systems (DBMS). Reasons for the use of diverse DBMS include a lack of coordination as well as very large size of organizations. This diversity creates big challenges in database integration. The database integration is a key process in many processes like the merger & acquisition of organizations, the adoption of enterprise architecture to facilitate services on the internet, and so on.

The goal of information integration is to provide an integrated and unified view of information residing at different sources. Integration is one of the core problems in database systems and has many application domains such as e-commerce, warehouse management, digital libraries and data mining. (Wiederhold, 1996; Knoblock and Levy, 1995; Widom, 1995; Hull, 1997)

Significant work has been done solving integration problems using different techniques. One of those techniques is schema integration in which we take input from several subschemas each of which were developed independently from each other and produce an output representing a global and unified schema (Batini, Lenzerini & Navathe, 1986). Initially, schema integration was performed with the help of meta-data or required human (expert) knowledge. Recent work has been carried out in schema integration focusing on an integration process taking actual data into account. In this process, the input is a collection of data built using different schema and format, and the goal is to produce an output which consists of global and unified schema.

In this thesis, we will concentrate on schema integration using inclusion dependencies. The inclusion dependency is an integrity constraint in relational databases (Date, 1981; Casanova, Fagin, Papadimitriou, 1984; Mannila and Räihä, 1986; Casanova, Tucherman and Furtado, 1988; Godfrey, Grant, Gryz and Minker, 1998; Leven and Vincent, 2000). It represents semantics of the database and plays a key role in various applications, such as relational database design and maintenance (Leven, 2000; Klettke, 1999), database reverse engineering (Casanova and de Sá, 1983; Markowitz and Makowsky, 1990;Petit, Toumani and Kouloumdjian, 1995), semantic query optimization (Paulley & Larson, 1994; Qian, 1996; Gryz, 1998; Cheng, Gryz, Koo, Leung, Liu, Quian and Schiefer, 1999), efficient view maintenance in data ware houses (Quass, Gupta, Mumick and Widom, 1996; Laurant, Lechtenbörger, Spyratos and Vossen, 1999) and so on.

Two significant algorithms developed in the field of discovery of inclusion dependencies are *FIND*$_2$ (Koeller & Rundensteiner, 2003) and *Zigzag* (De Marchi, Lopes & Petit, 2002). Both algorithms automatically discover a complete set of inclusion dependencies from the data in two arbitrary relational tables. While both algorithms perform well on smaller databases, they do not scale well for more than 20-30 attributes in some cases.

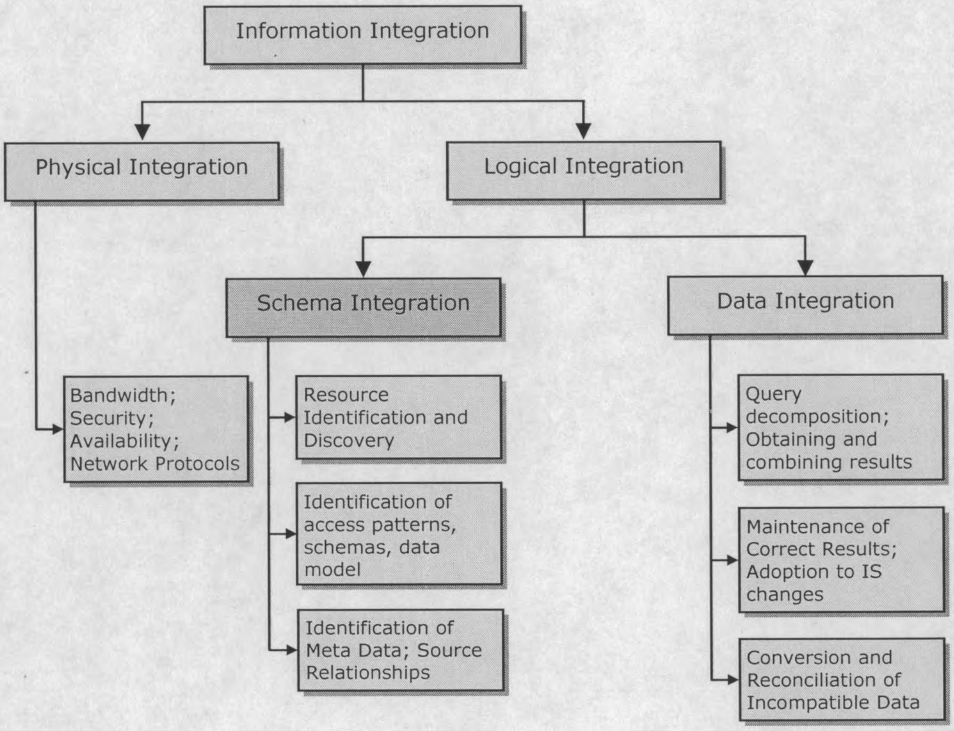
The goal of this thesis is to improve the performance and the accuracy of the overall discovery process by eliminating unwanted noise from input data using human intelligence. To incorporate human intelligence into the discovery process along with the algorithm computation, we introduce a software agent, called the discovery agent, to perform the discovery process. The main task of the discovery agent is to eliminate unwanted noise by involving human expertise. The automatic discovery of inclusion dependencies using an algorithm alone suffers from low quality in the result accuracy and high runtime of the algorithm computation.

In the following chapters, we first discuss inclusion dependencies and their importance in the field of data integration. Then we discuss the inclusion dependency discovery process using $FIND_2$ and *Zigzag* algorithms. Due to limited scope of the thesis, we concentrate more on the $FIND_2$ algorithm. However, the discovery agent can work with any discovery algorithm with moderate changes. After discussing the discovery process, we learn details about the discovery agent and its execution flow. Then we discuss about the user interface of the agent and see how it makes the discovery process interactive and effective.

2. Background

2.1 Introduction to Schema Integration

Figure 2.1 gives a conceptual idea about information integration. In the classification of the figure, our main focus will be exclusively on logical integration, not physical integration.



Information Integration
Physical Integration
Logical Integration
Schema Integration
Data Integration
Bandwidth; Security; Availability; Network Protocols
Resource Identification and Discovery
Identification of access patterns, schemas, data model
Identification of Meta Data; Source Relationships
Query decomposition; Obtaining and combining results
Maintenance of Correct Results; Adoption to IS changes
Conversion and Reconciliation of Incompatible Data

Fig. 2.1.1 Tasks in Information Integration (Koeller, 2001)

As mentioned in Fig. 2.1.1 and described in (Koeller, 2001), logical integration involves two major issues:

1. Obtaining necessary schema information to build logical schema model of final outcome (Schema Integration)
2. Providing integrated data to user based upon schema designed in step-1 (Data Integration)

In this thesis, we will concentrate on *Schema Integration*. There are many unsolved challenges involved in *Schema Integration* due to the fact that input information sources are developed independently of each other and there is no physical link between them.

In *Schema Integration*, the core problem is to develop a mapping between elements of two input schemas which are related semantically to each other. That mapping is scalled a *Match* and the process is called *Schema Matching* (Rahm & Bernstein, 2001).

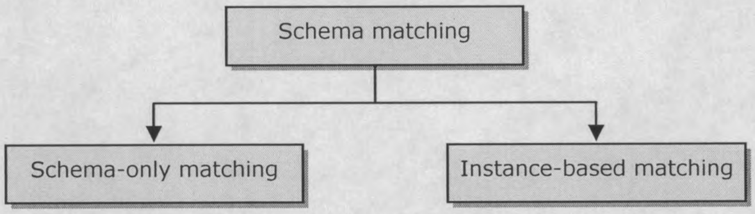


Fig. 2.1.2 *Schema Matching* (Rahm & Bernstein, 2001)

From figure 2.1.2, there are two ways to perform *Schema Matching*:

1. Schema-only matching
2. Instance-based matching

In *Schema-only matching*, the match is performed under the consideration of only meta data (schema information) from both input sources. Due to the fact that we are not taking data into account, we only have information such as entity names, data type, foreign-key relationships, constraints and other schema structures to perform integration. Here the match can be performed at the schema-element-level, in which we match each element of the first schema corresponding to an element of the second schema, or at the structure-level, in which we match a group of elements forming a structure from the first schema to a corresponding structure of elements from the second schema. Both approaches are often implemented through linguistic means (e.g. based upon name, textual descriptions of schema elements) or are constraint-based (e.g. based upon keys and relationships). (Rahm & Bernstein, 2001)

The main drawback of *Schema-only matching* is the frequent lack of proper information and documentation from both input schemas. Since both input schemas are developed independently from each other, consistent schema information is often difficult to obtain or unreliable. One solution to this problem is to use (human) domain experts to collect additional information regarding input schemas and apply such information in *Schema-only matching*.

In the second approach, *Instance-based matching*, the match is performed under consideration of the data itself. This approach is more generally applicable than schema-based matching and helpful even in the case of semi-structured data. Here the final schema can be derived from patterns discovered in the instance data. Both linguistic and statistical approaches are used. While instance-based matching is the only applicable method when schema information is missing, is can still benefit integration when substantial schema information is available. A combination of both approaches is promising.

2.2 Problem Definition

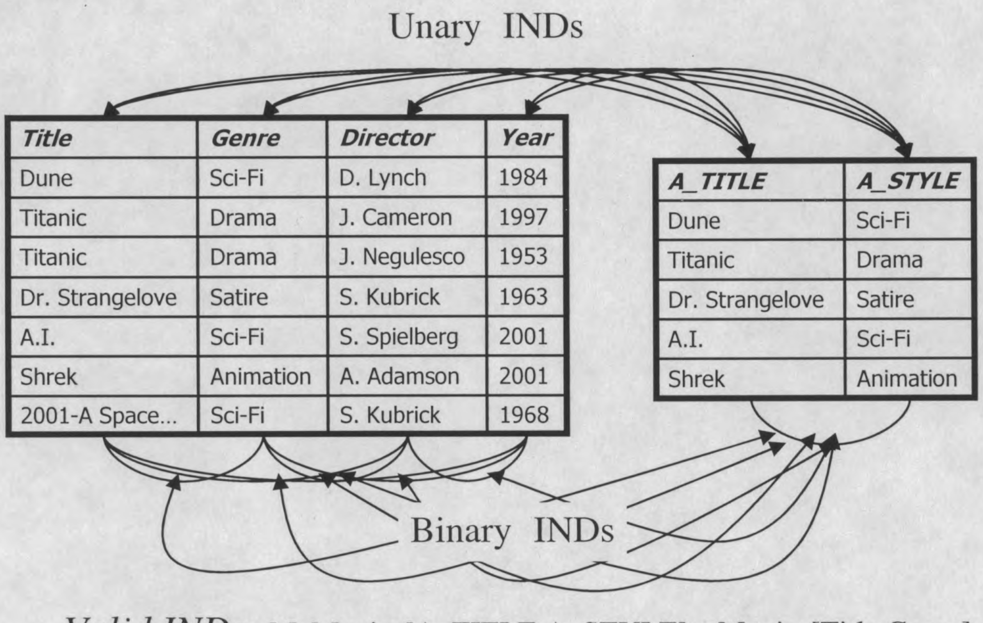
The problem addressed in this thesis is the discovery of Inclusion Dependencies (INDs) between tables in unknown relational databases, by comparing the data in two given tables.

Formally, an inclusion dependency is defined as below:

Definition 1: IND

Let $R[a_1, a_2, \ldots, a_n]$ and $S[b_1, b_2, \ldots, b_m]$ are two input relations. Let X and Y be a sequence of k distinct attributes from R and S respectively, where $1 \leq k \leq \min(n.m)$. An inclusion dependency σ is a rule of the form $\sigma = R[X] \subseteq S[Y]$. Here k is called the *arity* of σ.

Note that an IND is simply an assertion, it might be true in a given database instance or it might not be true.



Title	Genre	Director	Year
Dune	Sci-Fi	D. Lynch	1984
Titanic	Drama	J. Cameron	1997
Titanic	Drama	J. Negulesco	1953
Dr. Strangelove	Satire	S. Kubrick	1963
A.I.	Sci-Fi	S. Spielberg	2001
Shrek	Animation	A. Adamson	2001
2001-A Space...	Sci-Fi	S. Kubrick	1968

A_TITLE	A_STYLE
Dune	Sci-Fi
Titanic	Drama
Dr. Strangelove	Satire
A.I.	Sci-Fi
Shrek	Animation

Valid INDs: MyMovies[A_TITLE,A_STYLE] ⊆Movies[Title,Genre]
MyMovies[A_TITLE] ⊆Movies[Title]
MyMovies[A_STYLE] ⊆Movies[Genre]

Fig. 2.2.1 Example of unary and binary INDs

Fig. 2.2.1 describes the unary (k=1) and binary (k=2) INDs from two relations MyMovies and Movies.

Definition 2: *valid* IND

An IND $\sigma = R[a_1, a_2, \ldots, a_n] \subseteq S[b_1, b_2, \ldots, b_m]$ between two relations R and S is *valid* if the sets of tuples in R and S satisfy the rule given by σ. Otherwise the IND is called *invalid* for R and S.

Casanova *et al.* (Casanova et al., 1982) describe a complete set of inference rules (*axiomatization*) for INDs. That means that, in a process similar to functional dependency inference, new INDs can be derived from existing INDs through the three rules stated below:

Axiom 1: *Reflexivity*

For any relation $R[a_1, a_2, \ldots, a_n]$, $R[a_1, a_2, \ldots, a_n] \subseteq R[a_1, a_2, \ldots, a_n]$ is *valid.*

Axiom 2: *Transitivity*

For any three relations $R[a_1, a_2, \ldots, a_n]$, $S[b_1, b_2, \ldots, b_n]$ and $T[c_1, c_2, \ldots, c_n]$, if an IND $R[a_1, a_2, \ldots, a_n] \subseteq S[b_1, b_2, \ldots, b_n]$ and an IND $S[b_1, b_2, \ldots, b_n] \subseteq T[c_1, c_2, \ldots, c_n]$ are *valid*, then an IND $R[a_1, a_2, \ldots, a_n] \subseteq T[c_1, c_2, \ldots, c_n]$ is also *valid.*

Axiom 3: *projection and permutation* (from (Casanova et al., 1982))

If $R[a_1, a_2, \ldots, a_n] \subseteq S[b_1, b_2, \ldots, b_n]$ is *valid*, then for any sequence $(i_1, i_2, \ldots, i_k)$ of distinct integers from $\{1, 2, \ldots, n\}$ $R[a_1, a_2, \ldots, a_k] \subseteq S[b_1, b_2, \ldots, b_k]$ is *valid.*

Definition 3: *derived* INDs

A *valid* IND σ can be *derived* from a set Σ of *valid* INDs, if σ can be obtained by repeatedly applying the above axioms on INDs taken from Σ.

From above definitions and axioms (in particular the projection-and-permutation axiom), it is clear that a valid *k*-ary IND with $k > 1$ naturally implies a set of valid *m*-ary INDs, for any $1 \leq m \leq k$, by projection. Note that the number of INDs implied by a *k*-ary IND is equal to the number of subsets of attributes in the IND, i.e., 2^k. This leads to a very high number of INDs that need to be discovered.

The problem of discovering inclusion dependencies among input relations has been proven to NP-hard in terms of the number of attributes of input relations (Kantola, Mannila, Räihä & Siirtola, 1992).

Since INDs can be derived from one another, a set of INDs can be completely described by a subset from which all INDs in the set can be derived.

Definition 5 – IND Cover.

Consider $\Sigma = \{ \sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_n \}$ is a set of *valid* INDs. A *cover* of Σ, denoted by $\in(\Sigma)$, is a set of valid INDs with the following properties:

(1) For all INDs among Σ, each IND can be *derived* from the *cover*.

(2) If any IND is removed from the *cover*, all INDs of Σ can no longer be derived from the *cover*.

In other words, the *cover* $\in(\Sigma)$ contains only those valid INDs from which *all* valid INDs in Σ can be derived. For details, see (Koeller and Rundensteiner, 2002).

Since the *cover* contains all information about inclusion dependencies between relations in a minimal number of INDs, the IND discovery problem can be reduced to the problem of finding the *cover* of *valid* INDs. While trying to find all *valid* INDs is impossible due to their large number, the *cover* can be found using efficient techniques.

Due to the fact that the maximal number of INDs is very high, an algorithm to find the *cover* should perform following tasks:

1. Select a number of INDs for *validation* (also knows as a *guess)*
2. Test those individual INDs for actual *validity*
3. Refine the *guess* and repeat.

We will now discuss two IND discovery algorithms, *FIND*$_2$ and *Zigzag*, which use different approaches to solve a discovery problem.

2.3 Algorithms developed in IND Discovery

As mentioned earlier, there are two known algorithms developed in the discovery of INDs.

(1) *Zigzag* by Fabien De Marchi (De Marchi, Lopes & Petit, 2002)

Definition 1: *search space C*

Let $R[a_1, a_2, \ldots, a_n]$ and $S[b_1, b_2, \ldots, b_n]$ are two input relations.. The *search space C* can be defined as:

$C = \{ R[a_1, a_2, \ldots, a_n] \subseteq S[b_1, b_2, \ldots, b_n] \}$ *and*

$\forall\, 1 \leq i < j \leq n, (a_i < a_j) \vee (a_i = a_j \wedge b_i < b_j)$

In words, C contains IND expressions whose sequences are sorted in order to naturally restrict the search space to only one permutation of each IND expression [see also Axiom 3, above].

Definition 2: *generalization*, *specialization*

Let *i* and *j* be two IND expressions, we say that *i generalize j* (or *j specialize i*), if *i* can be obtained by projection on *j*.

Definition 3: *positive border* or *positive cover*

Let $I \subseteq C$. The *positive border* or *positive cover* of *I* is the set of INDs which can be represented by its most *specialized* (largest) elements.

Definition 4: *negative border* or *negative cover*

Let $I \subseteq C$. The *negative border* or *negative cover* of *I* is the set of INDs which can be represented by the most *general* elements which it does not contain.

Definition 5: *pessimistic approach*

In the *pessimistic approach*, the algorithm walks through the IND search space from most general (smallest) INDs to most specific (largest) INDs. It involves discovery of high-arity INDs combining *valid* low-arity INDs and validating them against the database. This approach is not practical whenever large INDs have to be discovered. That is because in order to discover one *k*-ary IND, 2^k smaller INDs have to be discovered first.

Definition 6: *optimistic approach*

In the *optimistic approach*, the algorithm makes a speculation of a specific (large) IND and validates it against the actual database.

The *Zigzag* algorithm uses a combination of pessimistic and optimistic approaches. The algorithm is divided into two main sections:

1. Pessimistic exploration of general INDs up to a specified level.
2. "Zigzag" between *negative border* in construction and corresponding *positive optimistic border.*

The principle of this algorithm is to mix top-down and bottom-up approaches for finding the *positive border* of *valid* INDs.

First, the *pessimistic approach* is performed using a levelwise algorithm *M I N D* [8] up to specified level *k*. Using the result of this step two sets of INDs, a set SI_k which contains *valid* INDs and a set $UnSI_k$ which contains *invalid* INDs, are made. Then the algorithm computes *positive cover* $Co^+(I_k)$ and *negative cover* $Co^-(I_k)$ from SI_k and $UnSI_k$ respectively. Using the optimistic positive border generation algorithm [9], the *optimistic positive border* $Co^+(I_{opt})$ is then computed from $Co^-(I_k)$. The algorithm terminates when every element of $Co^+(I_{opt})$ is marked as true in previous passes i.e. when $Co^+(I_{opt})$ / $Co^+(I_k)$ is empty. Otherwise INDs of $Co^+(I_{opt})$ / $Co^+(I_k)$ are validated against the database. After validation, the *valid* INDs are added to $Co^+(I_k)$. The *invalid* INDs are then divided into two groups OptI and PessI. OptI contains INDs which are "almost true" and the rest are in PessI. The algorithm uses a special error measure mechanism to generate OptI and PessI. The INDs of OptI are traversed into a top-down manner, from the most specific to most general. While performing this traversal, $Co^+(I_k)$ and $Co^-(I_k)$ are updated accordingly. The INDs of PessI are validated against the database and based upon the result, $Co^+(I_k)$ and $Co^-(I_k)$ are updated. Again $Co^+(I_{opt})$ is computed from the new $Co^-(I_k)$ for next iteration. (De Marchi, Lopes & Petit, 2002)

Thus, the algorithm carries out *optimistic jumps* in the IND search space, performing "*Zigzags*" between the negative and positive borders in construction.

The algorithm performs well finding INDs among relations having a moderate amount of tuples and relatively few attributes.

(2) *FIND$_2$* by Andreas Koeller (Koeller, 2001)

The *FIND$_2$* algorithm uses a graph theory approach to discover INDs among the input relations by mapping an IND Discovery problem to the graph problem. Before we go into details of algorithm, we will give brief definitions of concepts used in the algorithm.

Definition 1 – *k*-hypergraph

A *k*-uniform hypergraph is a pair G = (V, E) where V is the set of nodes and E is the set of edges. An element $e \in E$ is a set with cardinality k of pairwise distinct elements from V, denoted by $\{v_1, v_2, \ldots, v_k\}$. An element e is called *k*-hyperedge and *k* is known as rank of graph G.

Definition 2 – Clique

A clique of graph G is a maximal complete subgraph of G (Harary 1994).

Definition 3 – hyperclique

Let G = (V,E) be a *k*-hypergraph. A hyperclique is a set $C \subseteq V$ such that for each *k*-subset S of distinct nodes from C, the edge corresponding to S exists in E. The cardinality of a hyperclique C, denoted by $|C|$, is the number of nodes in C.

The *FIND$_2$* algorithm maps a substantial part of the IND discovery problem to the Clique-Finding problem (also known as the Maximum Clique Problem), which is a famous NP-complete problem. More details on this mapping can be obtained from (Koeller & Rundensteiner, 2003). *FIND$_2$* uses a special algorithm called HYPERCLIQUE to find the hypercliques in *k*-uniforms hypergraphs. Due to limited scope of this thesis, details and description about the HYPERCLIQUE algorithm is omitted. For detail information, please refer (Koeller & Rundensteiner, 2003).

According to (Koeller & Rundensteiner, 2004), the *FIND$_2$* algorithm takes two relations *R* and *S*, with k_R and k_S attributes respectively and returns the cover of INDs between k_R and k_S by applying clique- and hyperclique-finding techniques using iterative process going from *m*=2 to k_S. The algorithm divides the whole operation into three phases (Fig. 2.3.1):

I BASELINE : finding valid unary and binary INDs
II CLIQUE: finding high-arity INDs from cliques
III CLEANUP: finding additional INDs from subsets of invalid cliques.

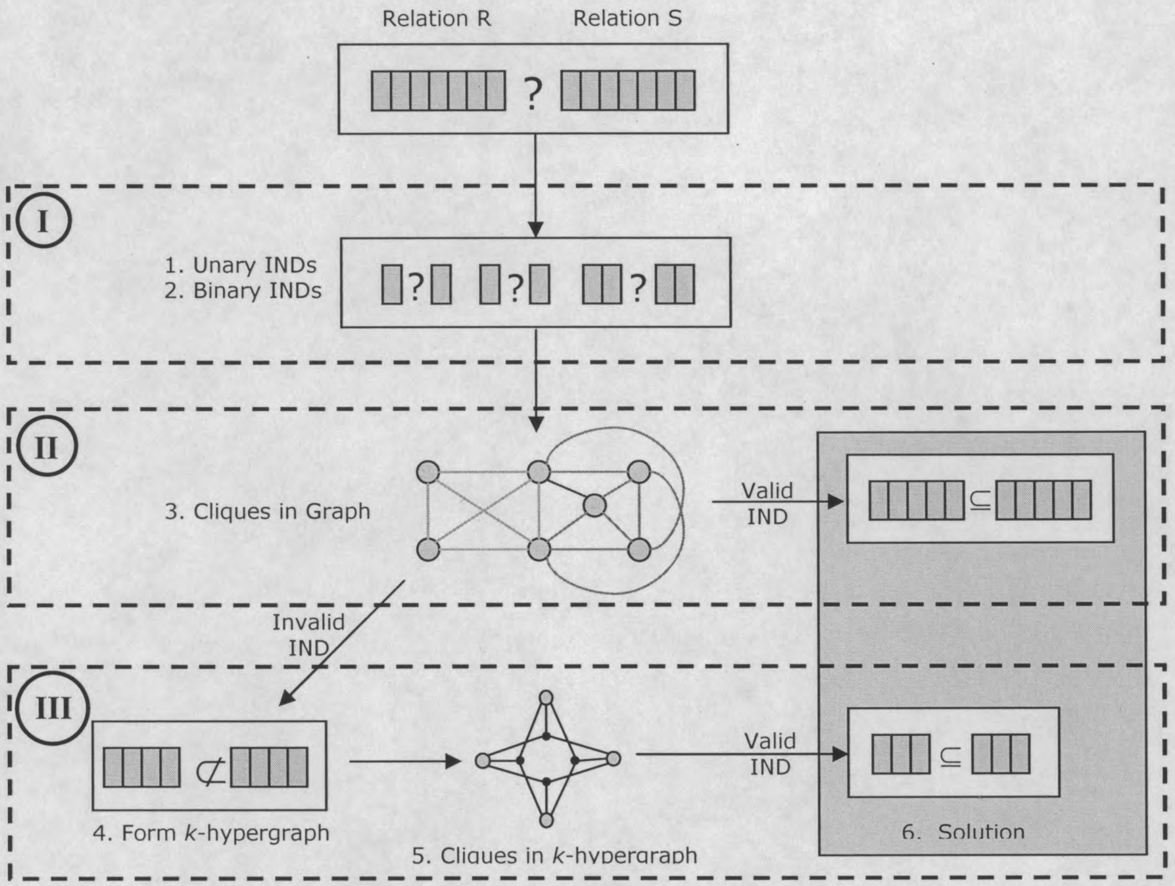


Fig. 2.3.1 Conceptual diagram of the *FIND*$_2$ algorithm

Phase-I: BASELINE

This is the first step in the *FIND*$_2$ algorithm in which we find unary and binary INDs from k_R and k_S in relations R and S. We first compute the unary INDs by combining k_R and k_S and then we validate them against the database. After validation, we have *valid* unary INDs which then serve as an input to find binary INDs. Again, the possible binary INDs are computed from the *valid* unary INDs and validated against the database.

Using *valid* unary INDs set V and *valid* binary INDs set E between input relation R and S, the algorithm then constructs a graph G=(V,E) (i.e., with the unary INDs as nodes and the binary INDs as edges) between them. Here any node in V which has no adjacent edges in E forms a clique of size 1 and corresponds to a unary IND that is part of the solution. Therefore, it is added to a specified *result* variable before further calculation.

Phase-II: CLIQUE

In this phase, the algorithm tries to find higher arity INDs from cliques. The HYPERCLIQUE algorithm is now used by the *FIND*$_2$ algorithm in this phase. First it computes the *m*-hypergraph G_m from nodes V and edges E_m (here m=2 which denotes binary level). The algorithm then computes hypercliques from graph G_m using a clique finding algorithm HYPERCLIQUE. It also validates each clique's implied IND against the database.

From the result of this validation, the set C_{tmp} is generated from the cliques that imply all *invalid* INDs, which then serves as an input to Phase-III. The cliques implying *valid* INDs are added to the solution.

From C_{tmp}, the algorithm generates m+1-ary INDs which serve as an input to the next iteration of the algorithm. If there are no m+1-ary IND found at this step, the algorithm terminates and returns *result* as a final answer.

Phase-III: CLEANUP

After the iteration process of Phase-II, the solution set would contain all valid INDs that have been found as (hyper)cliques of lower-arity INDs. However some maximal INDs may have been missed in this process. If an IND σ found by the clique-finding algorithm is invalid, it is still possible that its implied sub-INDs are valid and maximal. Therefore for the current step *m* all maximal *m*-ary sub-INDs of the invalid *m+1*-ary INDs in E_{m+1}.

For the given *m*, the algorithm then validates *m*-ary sub-INDs of the invalid *m+1*-ary INDs and adds the *valid* INDs to solution set. (Koeller & Rundensteiner, 2004)

For small arity INDs (arity<9), an "exhaustive" or "levelwise" algorithm is often faster due to lower overhead. However, it does not perform well for high arity INDs (arity>9). The *FIND*$_2$ algorithm shows a significant performance gain for INDs with arity>9. It also scales well while performing on larger relations.

Due to the limited scope of this thesis, we will concentrate on the *FIND*$_2$ algorithm in the further discussion. We will discuss limitations of the current algorithm and propose a *discovery agent* to overcome those limitations.

2.4 Autonomous Interface Agent

Before we discuss about the discovery agent, let's take a brief overview over an autonomous interface agent which is the key concept on which the discovery agent is built on.

An agent is an autonomous entity with an ontological commitment and agenda of its own. Recent studies in Human-Computer Interaction divide agents into two main categories:

1. Interface agents, in which the agent operates in the interface and assists users in operations
2. Autonomous agents, in which the agent operates on its own without user intervention.

Both *interface agents* and *autonomous agents* are designed and implemented to perform distinct operations. However in many cases, we need functionalities from both of them. An *autonomous interface agent* (Lieberman, 1997) is designed to incorporate features from both *interface agent* and *autonomous agent*. Lieberman explains the importance of *autonomous interface agents* and shows how useful they are when performing certain tasks, which needs both autonomous behavior and careful observation to achieve accurate results (Lieberman, 1997). He developed an autonomous interface agent, called "Letizia", which searches through the Web space as a continuous, cooperative venture between the user and a computer search agent (Lieberman, 1997).

In this thesis, we developed an interface agent for the purpose of discovering inclusion dependencies in databases.

3. An interactive approach for inclusion dependency discovery

As explained before, in phase-I, the *FIND*$_2$ algorithm discovers the unary INDs from the input relations using an enumeration technique. Using these unary INDs, it then discovers the binary INDs. Using the unary and the binary INDs as the ground work, the algorithm discovers high-arity INDs in higher phases.

Let $R[a_1, a_2, \ldots, a_n]$ and $S[b_1, b_2, \ldots, b_m]$ are two input relations. The formation of unary INDs can be described as below:

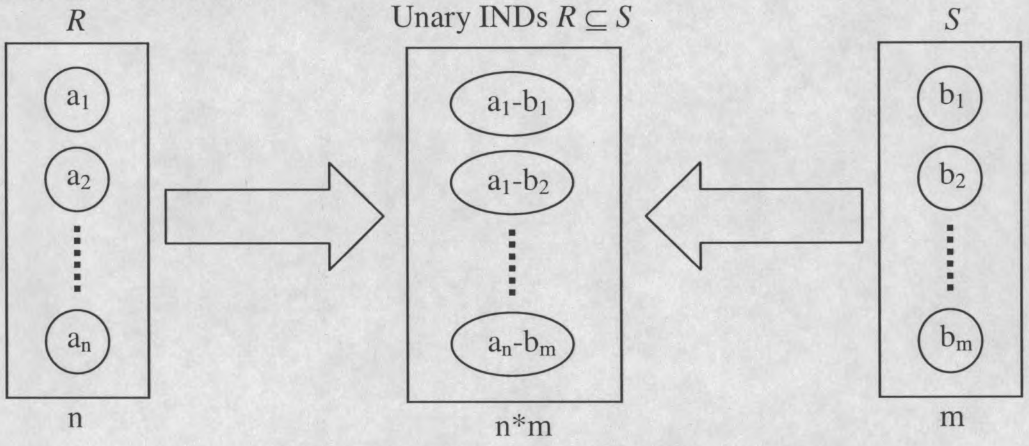


Fig. 3.1 The formation of Unary INDs

Here we are assuming all attributes from R and S are of the same type so that they can form a possible unary IND. n*m is the largest number of possible INDs between R and S. In general terms, if a relation R has total k_R attributes and a relation S has total k_S attributes, there are k_R* k_S unary INDs possible.

A binary IND is formed from two unary INDs as shown in fig. 3.2.

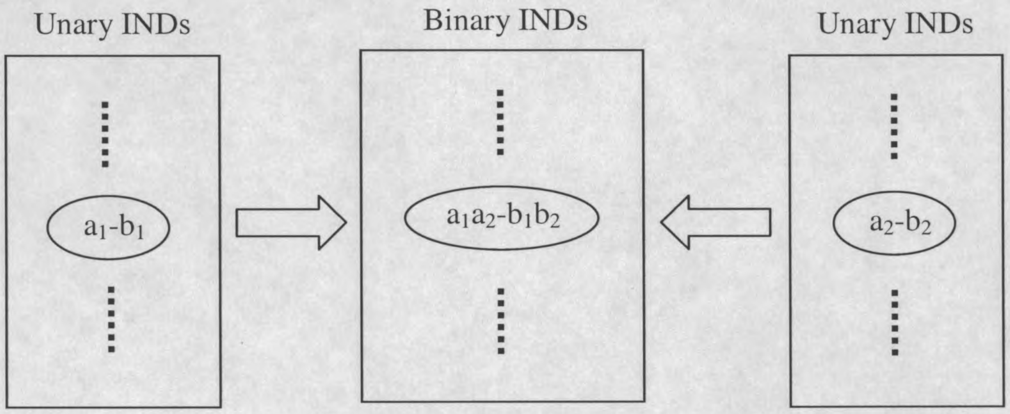


Fig. 3.2 The formation of Binary INDs

In words, the binary INDs can be generated from pairs of unary INDs by merging them together. So the number of binary INDs generated could be O(n^2) of the number of unary INDs.

Also if we merge two *valid* unary INDs and make one binary IND, it is not always true that resulting *binary* IND is also *valid*. Fig. 3.3 explains that scenario.

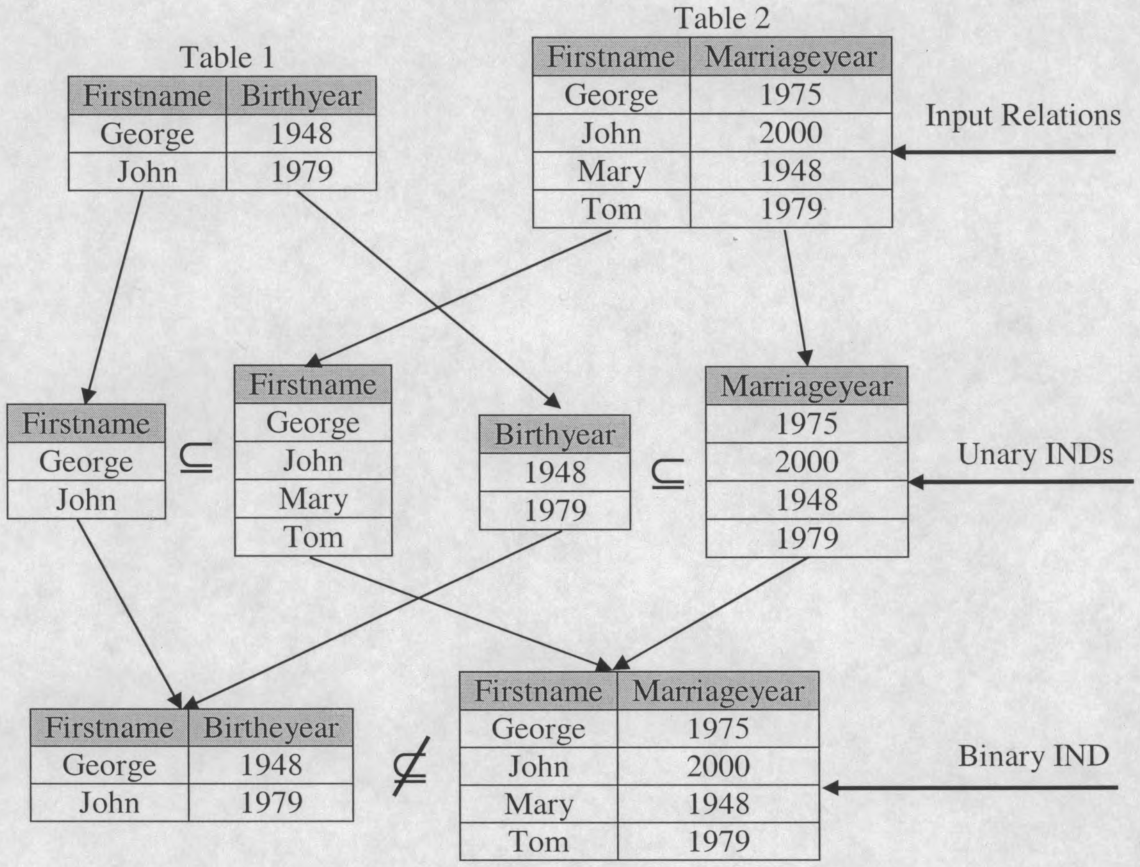


Fig. 3.3 Formation of *valid* unary INDs and *invalid* binary IND

From Fig. 3.3, we can see that two unary INDs Table1[Firstname] $\subseteq$ Table2[Firstname] and Table1[Birthyear] $\subseteq$ Table2[Marriageyear] are *valid* but the binary IND Table1[Firstname, Birthyear] $\subseteq$ Table2[Firstname, MarriageYear] is *invalid*. But still we have to check this validity against the database which consumes resources.

One important point to be noticed here is that in the formation of unary INDs the input is *attributes from relations* while in the formation of binary INDs the input is *formally discovered unary INDs*. Since two unary INDs are combined to form a binary IND, a moderate amount of unary INDs can generate a large amount of binary INDs. We need to check *validity* of each unary and binary INDs against the database. This validation can be resource and time consuming and due to that it affects the algorithm performance.

For example, if we have 10 attributes in both input relations there are 100 possible unary INDs and 4050 possible binary INDs (if we assume that all 100 unary INDs are *valid*). Now we need to run at least 4150 database queries to check the validity of unary and binary INDs. If we have 15 attributes in both input relations, the number of possible unary INDs is 225, the number of possible binary INDs is 22050 and the total database queries just to validate unary and binary INDs are 22275. Here the small addition of 115 unary INDs can result into a big addition of 18000 binary INDs and 18125 database queries. Thus the resource consumption in this process grows extremely fast.

If we follow the same brute-force enumeration approach to find high-arity INDs, we will face an exponential run-time and it will become practically impossible to find *valid* high-arity INDs. By using the HYPERCLIQUE algorithm and other intelligent computations in Phase-II, the *FIND*$_2$ algorithm does an excellent job generating high-arity INDs and at the same time eliminating many *invalid* INDs. However, depending on the data in the input relations, the algorithm will discovery many unary and binary IND candidates which will turn out later not to be part of the solution. In those cases, the algorithm will spend most of its resources in their validation and will not able to perform higher phases computations.

The main reason for this phenomenon is that a *valid* unary IND is a purely mathematical pattern (the values of one attribute form a subset of the values of another attribute) and may not represent a relationship of those attributes in a real world. Such INDs slow down the algorithm significantly and should not be included in the algorithm computations. This concept is defined more formally below:

Definition 1: *spurious* INDs

Whether an IND is *valid* or *invalid*, is decided by taking actual data into an account. Thus a *valid* IND suggests a relationship between attributes. In many cases, we accidentally found INDs between attributes representing two distinct entities but having similar data values. Since the IND is *valid* from the data perspective, it is still *irrelevant* in the real world. These INDs are known as *spurious* INDs. (Koeller, 2001)

The *spurious* INDs are generated mostly by accident due to similarity in data. Since the relationships based on these INDs do not hold in the real world, these INDs are considered as a noise to the algorithm. They should be eliminated for an accuracy of the result.

For example, consider two relations Table1[Firstname, Birthyear] and Table2[Firstname, Marriageyear] from fig. 3.3. Here the unary IND Table1[Birthyear] $\subseteq$ Table2[Marriageyear] is *valid* from the database point of view but since both attributes represent two distinct entities in the physical world, it is irrelevant from the real world physical-entities perspective and it is a *spurious IND*.

The automated process used by the algorithm to eliminate this unwanted noise is not sufficient for accurate IND discovery. We will learn more about this process in a later section and will see how it is not effective in all cases.

We propose an approach to use human intelligence and knowledge along with the algorithm computation for elimination of unwanted noise. Human intelligence is helpful in many automated processes. As a daily-life example, consider "Highway cruise control". A car on a road can drive at steady speed but is not capable of taking a decision when the traffic conditions change. At that time, the user takes control of the car. In this sense, our main focus would be implementing an agent which can work with an observer to eliminate unwanted noise generated by data discrepancy.

With the inspiration from Letizia (Section 2.4), we propose an approach to design an agent to discover inclusion dependencies among input relations using the $FIND_2$ algorithm. We refer to this agent as "The discovery agent" since it is mainly designed for the inclusion discovery process. We will now discuss the details about the discovery agent in next section.

4. The discovery agent

The discovery agent is built on the similar concept used by the *autonomous interface agent* (Lieberman, 1997). It provides a communication layer between an observer and an algorithm as well as between an algorithm and the actual data source(s). The core part of the agent is the *FIND*$_2$ algorithm and the other two layers (interfaces) are provided to facilitate the external communication. Fig. 4.1 gives the view of the discovery agent operation.

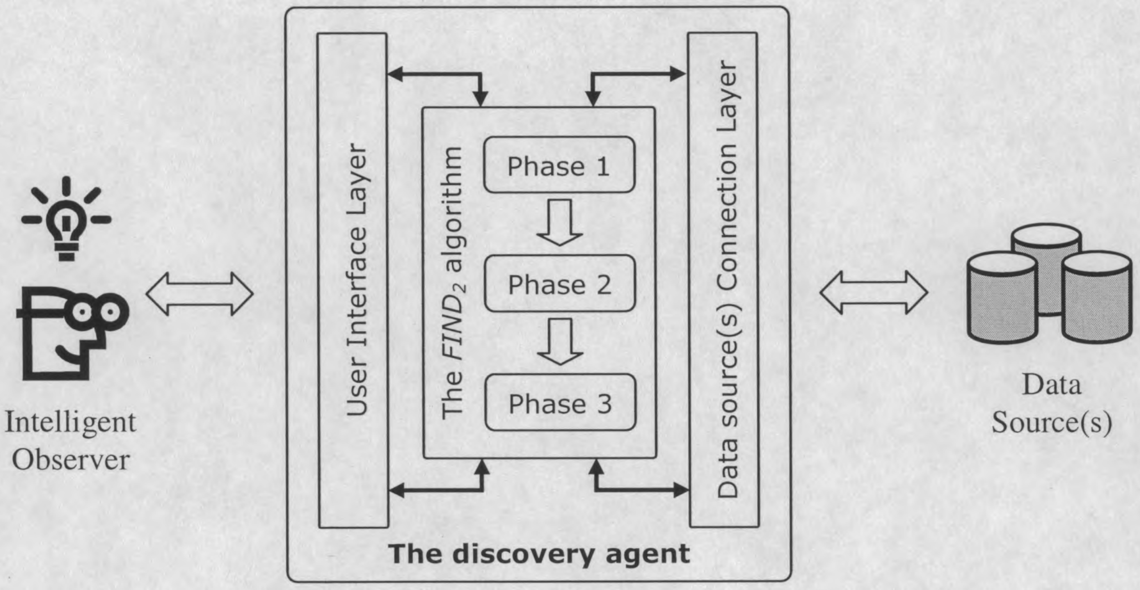


Fig. 4.1 The Outlook of the discovery agent

The task of the data source(s) connection layer is to provide the data to the algorithm upon request. The data can come from physical databases or from the Internet using different services. Since a human intervention is not required in this case, this interface can be considered as *autonomous*. The proper implementation of this interface can reduce the time spent on accessing and transferring data to the algorithm and thus it can improve the overall performance of the agent. In this thesis, the data is provided by a local database and the provided implementation of this interface with respect to the scope of the thesis is efficient in terms of data availability. Our main focus will be on the user interface layer.

Using the user interface, the discovery agent can take advantages of the observer's intelligence and knowledge in the discovery process. In the same manner, an observer is beneficial to supervise the discovery process and can drive the discovery process into the right direction using his knowledge and his decisions. In each phase, the algorithm needs an intelligent assistance and guidance to proceed in to the right direction. Using this layer the agent can provide the required assistance and guidance to the algorithm.

The discovery agent has certain characteristics and abilities as described below:

- ❑ The discovery agent provides a visual representation of the discovery process to an observer via the user interface layer. While the *FIND*$_2$ algorithm is begin executed, the visual representation contains the algorithm progress, intermediate results of different phases, current algorithm computation and so on.

  Due to this ability, an observer will be able to follow and supervise the algorithm's progress. It helps him making a right decision about the further computations.

- ❑ To accommodate the phase-wise behavior of the *FIND*$_2$ algorithm, the discovery agent should provide an ability to go back to the previous computational phase. With this ability, an observer should be able to "undo" the wrong computation. Also an observer should be able to stop and resume the process at any time.

- ❑ The discovery agent should provide an ability to save the progress and the result of its current stage. It can also open and resume the discovery process from the previously saved stage.

  It gives flexibility to an observer in working with the agent on a discovery process.

- ❑ The discovery agent performs computations on input relations based upon the *FIND*$_2$ algorithm which has exactly two relations as its input. So the agent should provide a way to an observer to specify those input relations. Due to this, an observer will be able to run the discovery process on specific relations that he wants.

  The input data source(s) may contain many relations. Performing a discovery process on each relation pair from input data source(s) can be very time and resource consuming. With the help of this ability, an observer can narrow down a discovery process to relations that he/she interested in.

- ❑ The *FIND*$_2$ algorithm uses certain input parameters which are independent to input relations but are used in many internal computations (for example parameters describing the maximal size of certain internal data structures). These parameters can affect the algorithm execution process. The agent should provide an ability to change these parameters visually and then perform the algorithm execution based upon that.

  Due to this ability, an observer can able to tune the algorithm based upon his input criteria and thus will be able to achieve the maximum performance.

- ❑ The implementation of the agent should be independent of the algorithm it the sense that it should work with any algorithm besides *FIND*$_2$. It can also provide a way to an observer to choose the algorithm that he wants. Based upon the chosen algorithm, the agent reflects the proper user interface which is specifically designed for that algorithm. It makes the agent capable of incorporating multiple discovery algorithms.

This feature makes the agent versatile to work with any discovery algorithms. It is very difficult to make one *generic* agent to work with any discovery algorithm, but it is possible to do that with the proper implementation environment and techniques.

- ❑ The final output of the discovery agent is inclusion dependencies between input relations and thus a measure by which one can identify how much input relations and which of their attributes are related to each other. This output is very helpful in the integration of the input relations or the input data sources. Based on this output, the agent or the system based on this agent should ask an observer to perform additional tasks.

  Due to this, an observer can not only discover how input relations or data sources are related to each other, but he can also perform the actual integration based on those results. Thus the agent or the system can be used as an actual information integration system.

Above characteristics give an overall idea about the effectiveness and the extendibility of the discovery agent. Though we cover most of the characteristics and the abilities of the agent, the agent is not limited to them. The actual implementation of the agent may vary from above points because of the limitation of implementation environment and techniques.

From the above discussion we can think about the impact of the discovery agent.

The essential functionality of the agent, an interactive IND discovery, has been implemented in a prototype. As discussed in the "Problem Scope" section, the main area where the *FIND*$_2$ algorithm needs help of an observer assistance is in phase-I (the discovery of unary and binary inclusion dependencies). So in this thesis, we concentrate on a version of the discovery agent which only focuses on the Phase-I of the *FIND*$_2$ algorithm. Then we discuss the implementation techniques and the usability of our version of the discovery agent.

5. Interaction of the agent with the *FIND$_2$* algorithm

5.1 Background

Before we go into detail about the discovery agent, let's take a quick look at phase-I of the *FIND$_2$* algorithm. Fig 5.1 gives us an outline of the phase-I computational steps.

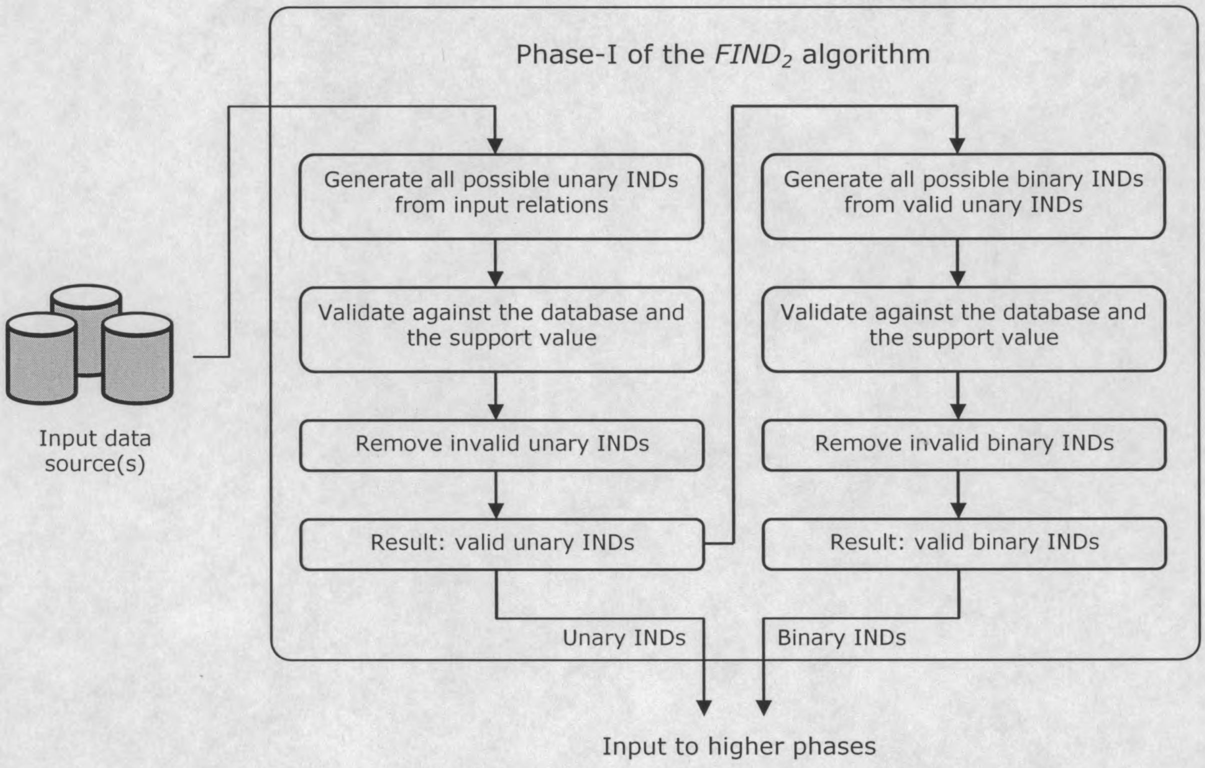

Phase-I of the FIND2 algorithm
Input data source(s)
Generate all possible unary INDs from input relations
Validate against the database and the support value
Remove invalid unary INDs
Result: valid unary INDs
Generate all possible binary INDs from valid unary INDs
Validate against the database and the support value
Remove invalid binary INDs
Result: valid binary INDs
Unary INDs
Binary INDs
Input to higher phases

Fig. 5.1.1 Outline of Phase-I of the *FIND$_2$* algorithm

As shown in Fig 5.1.1, first the algorithm starts processing with input relations and tries to find unary INDs. We already discussed the formation of the unary INDs before. The algorithm tries to discovery spurious INDs with the help of a *support* value, as defined below.

Definition: *Support* value

The support value is a heuristic measure to estimate the real-world semantic relationship between two sets of attributes that form a valid inclusion dependency. The *support* is a number composed of the number of distinct tuples in attributes/attribute sets and the distribution of actual values (a set of counts or multiplicities for each unique data value in each attribute) in attributes/attribute sets. (Koeller, 2001).

With the help of the support measure, the algorithm tries to discover semantic relationship between input attributes that stand in a *valid* inclusion dependency to each other. Similar numbers of distinct values and similar value distribution suggests a semantic relationship between attributes and therefore lead to higher support value. The lower support value can be caused by following reasons:

- ❑ The number of distinct values in the attributes/attribute sets is very low.
- ❑ The number of distinct values in one attribute/attribute set is significantly different from the number of distinct values in the other attribute/attribute set.
- ❑ The distribution of values in the two attributes/attribute sets is different.

From the definition of the support value, one can think a low support value as "no" relationship and a high support value as "maybe" or "strong" relationship between attributes. However since the computation of the support value is based on statistics, the result is not always reliable. For example:

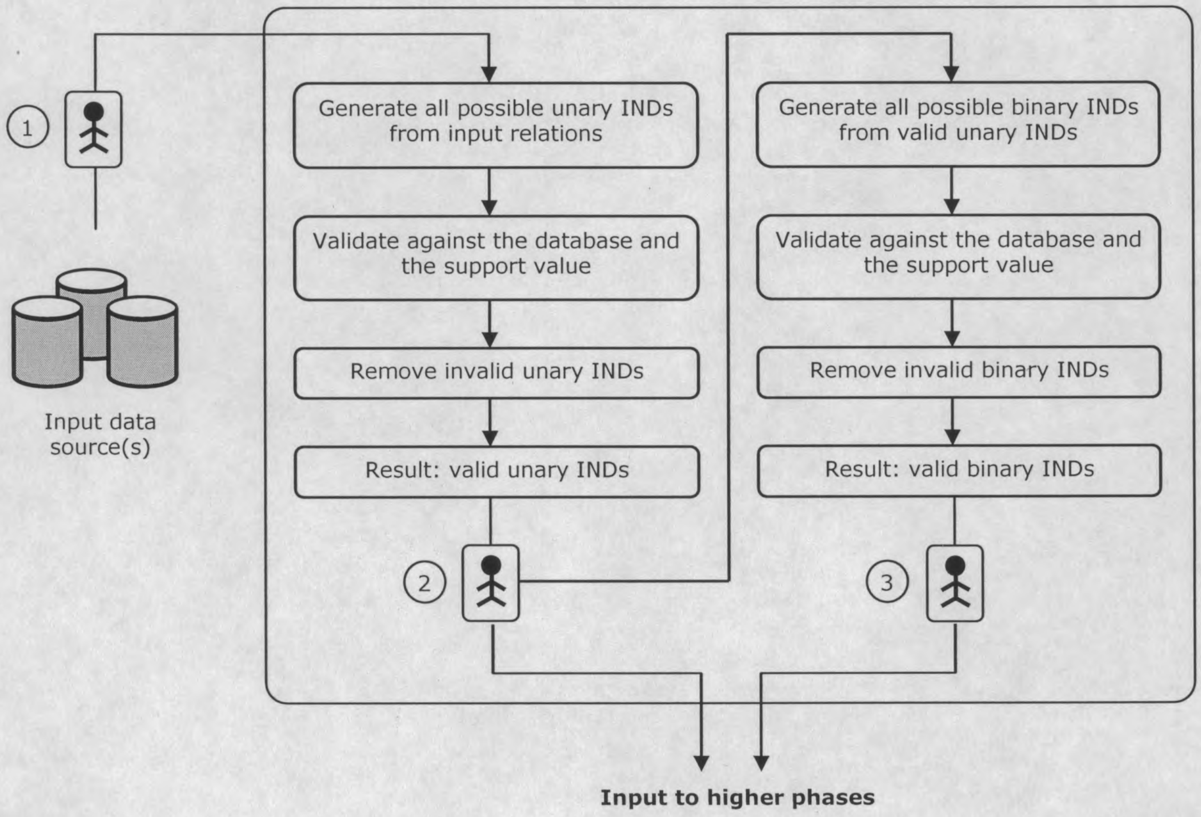
- ❑ A low support value is computed for an IND if the underlying attributes have very few values, suggesting that they are not related to each other in real world. For example, an attribute "MARRIED" has two distinct values TRUE and FALSE and an attribute "HOMEOWNER" also has two distinct values TRUE and FALSE. Based on these values, a low support value is computed, suggesting that "MARRIED" and "HOMEOWNER" represent two distinct entities in the real world and they are not related to each other. This IND would correctly be labeled as spurious due to its low support value. On the other hand, an IND relating two other attributes with two values each which *are* related would yield the same low support value, and could wrongly be labeled as spurious.
- ❑ In the similar manner, high support values are not reliable in all cases. If we have two attributes with 5 to 10 distinct values each, and they have similar distributions, the support value will be high but it may possible that attributes have different meaning in real world. We could have one attribute that has values from 1...6 representing the income group of a person (1 - under 10000, 2 - 10000 to 20000, and so on), and another attribute, from 1...6 representing the tax bracket of the person (1 - under 15%, 2 - 15-18%, 3 - 18-25%, and so on). In this case, the discovered inclusion dependency is spurious.

The support value suggests a semantic relationship between input attributes that have been found to form a valid IND in the database. The low and high score of the support values suggest how useful these INDs are (low being less useful and high being more useful). However as we have seen in above examples that these values are not always reliable. The algorithm can not easily decide on a threshold value for the support value above which all INDs are considered as non-spurious (i.e., semantically meaningful). An observer has to make that decision based upon the nature of attributes.

In the upcoming sections, we will discuss how the agent provides this facility to an observer. Using this facility, an observer can also include INDs having lower support values than the threshold value into the computation.

5.2 The agent overview

The discovery agent is specifically designed based upon this approach. We will discuss the design and implementation of the agent later but first let's take a look at the overall flow of the agent's execution process:



1
Input data source(s)
Generate all possible unary INDs from input relations
Validate against the database and the support value
Remove invalid unary INDs
Result: valid unary INDs
2
Generate all possible binary INDs from valid unary INDs
Validate against the database and the support value
Remove invalid binary INDs
Result: valid binary INDs
3
Input to higher phases

Fig. 5.2.1 Discovery agent execution flow

Fig. 5.2 gives the outlook of the execution process. Here the agent introduces the user intervention at 3 places (1, 2 and 3) as shown and as described below:

(1) First an observer decides the input relations on which he/she wants to perform the discovery process. The agent gives a choice to an observer to select only relations on which he/she wants to perform the discovery. It gives a good amount of flexibility to an observer.

(2) Here the agent asks an observer for his assistance to eliminate spurious INDs. The agent displays all INDs to an observer in a way that he/she will be able to identify spurious INDs. We will discuss more about the visualization of INDs in the next section. After an observer's decision, the resulting unary INDs are the *valid non-*

spurious unary INDs and they are used to find binary INDs as well as higher phases.

(3) In a similar manner, the agent seeks the observer's assistance in removing spurious binary INDs from among the valid binary INDs. After validating binary INDs against the database and calculating *support* measure, they are presented to an observer for review. An observer removes spurious binary INDs by careful examination of binary INDs and using user interface controls provided by the agent. These binary INDs are then used in higher phases of the $FIND_2$ algorithm.

With the observer's assistance and expert knowledge, the agent will be able to reduce most of spurious INDs at phase-I. Of course, the user interface and the visualization play an important role in this user-system interaction process. In following sections, we will discuss about visualization techniques and the user interface in sufficient details.

6. Visualization

In this section, we discuss the visualization techniques for unary and binary INDs. In this discussion we use following two input relations as a running example.

NAME	BIRTHYEAR	MEMBERSINCE	MEMBERUNTIL
Jones	1940	1969	1989
Miller	1945	1960	1988
Myers	1960	1980	1988
Shultz	1969	1988	1989
Becker	1961	1989	

Table 1: MEMBER

MEMBER	YOB	LEFTIN
Myers	1960	1988
Shultz	1969	1989

Table 2: FORMER

Fig 6.1 Input relations FORMER and MEMBER

Fig. 6.1 describes two input relations FORMER and MEMBER. We are trying to discover inclusion dependencies between tables MEMBER and FORMER. Fig 6.2 described the unique values of attributes in both relations:

TABLE	ATTRIBUTES	UNIQUE VALUES
MEMBER	NAME	Jones, Miller, Myers, Shultz, Becker
	BIRTHYEAR	1940, 1945, 1960, 1961, 1969
	MEMBERSINCE	1960, 1969, 1980, 1988, 1989
	MEMBERUNTILL	1988, 1989
FORMER	MEMBER	Myers, Shultz
	YOB	1960, 1969
	LEFTIN	1988, 1989

Fig 6.2 Unique values in MEMBER and FORMER

6.1 Visualization of unary INDs

As a simple optimization of the algorithm, only attributes with the same datatype (CHAR or NUMBER) are compared. Therefore, from fig. 6.1, we get a total of 7 possible unary INDs as described in fig. 6.1.1 below:

MEMBER[NAME] $\subseteq$ FORMER[MEMBER]
MEMBER[BIRTHYEAR] $\subseteq$ FORMER[YOB]
MEMBER[BIRTHYEAR] $\subseteq$ FORMER[LEFTIN]
MEMBER[MEMBERSINCE] $\subseteq$ FORMER[YOB]
MEMBER[MEMBERSINCE] $\subseteq$ FORMER[LEFTIN]
MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[YOB]
MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[LEFTIN]

Fig. 6.1.1 Possible unary INDs

The $FIND_2$ algorithm validates all of above unary INDs against the database. As a result, it found an invalid IND MEMBER[BIRTHYEAR] $\subseteq$ FORMER[LEFTIN] because unique values of BIRTHYEAR (1940, 1945, 1960, 1961, 1969) are different from unique values of LEFTIN (1988, 1989) and so they are assumed to represent two different physical entities. The same conclusion can be made for invalid IND MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[YOB]. These invalid INDs are removed from the result and the algorithm then concentrates on the remaining 5 valid INDs. The algorithm also computes support values for all 5 INDs which is helpful in further analysis. Fig. 6.4 describes the remaining 5 INDs with their support values:

No.	Unary INDs	Support value
1	MEMBER[NAME] $\subseteq$ FORMER[MEMBER]	0.4
2	MEMBER[BIRTHYEAR] $\subseteq$ FORMER[YOB]	0.4
3	MEMBER[MEMBERSINCE] $\subseteq$ FORMER[YOB]	0.4
4	MEMBER[MEMBERSINCE] $\subseteq$ FORMER[LEFTIN]	0.4
5	MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[LEFTIN]	0.67

Fig. 6.1.2 Valid unary INDs with support value

From fig. 6.1.2, there is only one IND, MEMBER[NAME] $\subseteq$ FORMER[MEMBER], is between the MEMBER attribute of FORMER and the NAME attribute of MEMBER with support value 0.4. Even though the support value is low, since there is no other INDs present which involves any of these two attributes we can say that this IND is valid and MEMBER and NAME both represents same physical entity. The data from Fig. 6.2 also support this premise.

On the other hand, the attribute YOB of FORMER is related with attributes BIRTHYEAR and MEMBERSINCE of MEMBER with two INDs MEMBER[BIRTHYEAR] $\subseteq$ FORMER[YOB] and MEMBER[MEMBERSINCE] $\subseteq$ FORMER[YOB] having same support value 0.4. Since it is very unlikely that one attribute in relation FORMER is related to *several* attributes in relation MEMBER, this result constitutes noise for the algorithm and it increases the algorithm's complexity and reduces the algorithm's accuracy. So for an accurate result, this noise should to be eliminated by using observer's assistance. By looking at data, the observer can conclude that the IND MEMBER[MEMBERSINCE] $\subseteq$ FORMER[YOB] is invalid in the real world because they both represent different physical entities. So he/she eliminates MEMBER[MEMBERSINCE] $\subseteq$ FORMER[YOB] from future computations using appropriate controls.

For the attribute LEFTIN of MEMBER, we found two INDs with different support values. The IND MEMBER[MEMBERSINCE] $\subseteq$ FORMER[LEFTIN] has support value 0.4 and MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[LEFTIN] has support value 0.67. Fig. 6.2 gives the reason for this difference. The unique values of attributes MEMBERUNTIL and LEFTIN are the same while the unique values of MEMBERSINCE and LEFTIN are different. Due to high support value, MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[LEFTIN] should be considered as non-spurious while the other can be considered spurious.

The unary inclusion dependencies establish relations between attributes of input relations. In that context, we can think of the unary inclusion dependencies between two input relations as edges and attributes from input relations as vertices of a bipartite graph. Since the core of the discovery agent, the *FIND*$_2$ algorithm, maps the IND discovery problem to a graph problem, we found the approach to visualize the unary inclusion dependencies as bipartite graph to be the most suitable.

The agent uses a new hybrid approach to display unary INDs as shown in fig. 6.1.3.

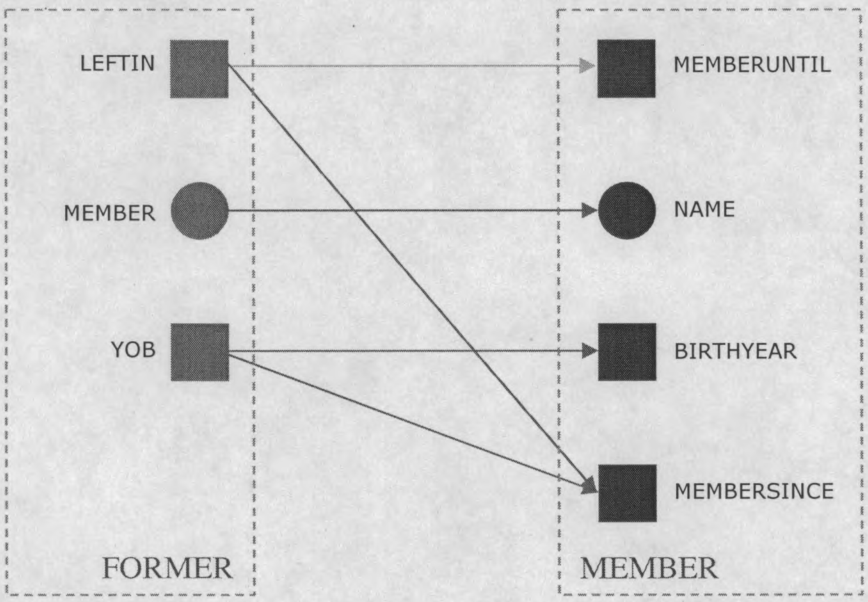


Fig. 6.1.3 The visualization of unary INDs

The approach used in Fig. 6.1.3 can be explained in following three categories:

(1) Vertex color distribution

Vertices are drawn using *k* colors to anticipate the idea of *k* groups of vertices in *k*-partite graph. Since we are comparing two relations here, we have *k* = 2 so we use only two colors. Vertices in red indicate the relation FORMER while vertices in blue indicate the relation MEMBER. Also vertices are drawn in such manner that all groups (2 here) do not overlap each other and display clear.

With the help of the color distribution, an observer will be able to visualize all input relations and their attributes, which are involved in the discovery process.

(2) Vertex shape distribution

In fig. 6.5, vertices are drawn using two geometrical shapes: circle and square. Here the shape of the vertex represents the data type of an attribute underneath. In that context, a circle is used to represent the data type 'VARCHAR' or textual data and a square is used to represent the data type 'NUMBER' or numeric data. Since there are only two distinct data types present from both input relations, we have only two distinct shapes. If we found more distinct data types, the agent will use more geometrical shapes.

Using this shape distribution, an observer will have a visual picture about all data types from input relations. By viewing distinct shapes, the observer can understand quickly why there are no inclusion dependencies between attributes of different types.

(3) Edge color distribution

To use observer's assistance, we use an approach of drawing edges using different colors to highlight the difference in support values of INDs. We use two basic colors "red" and "green". The color "red" represents the lowest support value and the color "green" represents the highest support value. INDs with support values between highest and lowest value are drawn using following technique:

1) We make a set of all support values, say S. So S = { s_1, s_2, ..., s_k } where s_1, s_2, ..., s_k are *k* support values and $s_1 \leq s_2 \leq \ldots \leq s_k$. So s_1 is the lowest support value and s_k is the highest support value.
2) We divide the color spectrum between red and green in to *k* different subparts and assign each part's color to corresponding support value. This distribution can be visualized as:

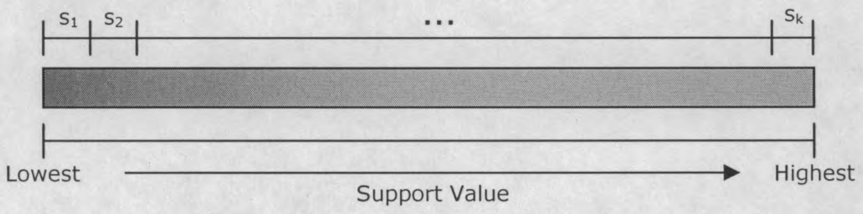


Fig. 6.1.4 Edge Color distribution

The color "red" is formed by RGB (255, 0, 0) and the color "green" is formed by RGB (0, 255, 0). Based upon the size of the support value collection, the color spectrum is constructed as shown in Fig. 6.1.4.

3) In our example, we have two support values 0.4 and 0.67. Since the size of the set is 2, we have drawn edges using only two colors red and green as seen in Fig. 6.1.3.
4) If the collection contains only one record (or all INDs have the same support value), we use "black" color for the edge.

The main objective of the color distribution is to develop a visual picture from which an observer has clear idea about which INDs should be consider as valid and also to eliminate spurious INDs.

By using proper user interface control, an observer can eliminate INDs with lower support values. At the same time, he/she can also choose any specific IND that he/she wants to include in final result. We will discuss more about the user interface details in next section.

After above analysis, we come up with 4 unary INDs as shown in Fig. 6.7, which are then used to find binary INDs.

No.	Unary INDs	Support value
1	MEMBER[NAME] ⊆ FORMER[MEMBER]	0.4
2	MEMBER[BIRTHYEAR] ⊆ FORMER[YOB]	0.4
3	MEMBER[MEMBERSINCE] ⊆ FORMER[YOB]	0.4
4	MEMBER[MEMBERUNTIL] ⊆ FORMER[LEFTIN]	0.67

Fig. 6.1.5 Resulting unary inclusion dependencies

6.2 Visualization of binary INDs

Fig. 6.1.5 display list of unary INDs from which binary INDs are generated as follows:

MEMBER[NAME, BIRTHYEAR] $\subseteq$ FORMER[MEMBER, YOB]
MEMBER[NAME, MEMBERSINCE] $\subseteq$ FORMER[MEMBER, YOB]
MEMBER[NAME, MEMBERUNTIL] $\subseteq$ FORMER[MEMBER, LEFTIN]
MEMBER[BIRTHYEAR, MEMBERUNTIL] $\subseteq$ FORMER[YOB, LEFTIN]
MEMBER[MEMBERSINCE, MEMBERUNTIL] $\subseteq$ FORMER[YOB, LEFTIN]

Fig. 6.2.1 Possible binary INDs generated from unary INDs

These binary INDs are then validated against the database. After validation, the algorithm computes the support value for each IND. As a result of the validation against the database, the binary IND MEMBER[NAME, MEMBERSINCE] $\subseteq$ FORMER[MEMBER, YOB] is removed. The data from Fig. 6.1 also supports this removal. Fig. 6.2.2 displays the resulting 4 binary INDs with their support values:

No.	Binary INDs	Support value
1	MEMBER[NAME, BIRTHYEAR] $\subseteq$ FORMER[MEMBER, YOB]	0.4
2	MEMBER[NAME, MEMBERUNTIL] $\subseteq$ FORMER[MEMBER, LEFTIN]	0.4
3	MEMBER[BIRTHYEAR, MEMBERUNTIL] $\subseteq$ FORMER[YOB, LEFTIN]	0.4
4	MEMBER[MEMBERSINCE, MEMBERUNTIL] $\subseteq$ FORMER[YOB, LEFTIN]	0.4

Fig. 6.2.2 Binary INDs with their support values

As seen from the formation of binary INDs, a binary is generated using two unary INDs. In that sense, the order of attributes must be preserved. For example, two unary INDs MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[LEFTIN] and MEMBER[MEMBERSINCE] $\subseteq$ FORMER[YOB] produce only one binary IND MEMBER[MEMBERUNTIL, MEMBERSINSE] $\subseteq$ FORMER[LEFTIN, YOB].

To visualize this behavior, we use *nodes to represent unary INDs* and an *edge connecting two nodes represents a binary IND* which is formatted using those two end node unary INDs.

For display purpose we use circle graph as shown in Fig. 6.2.3.

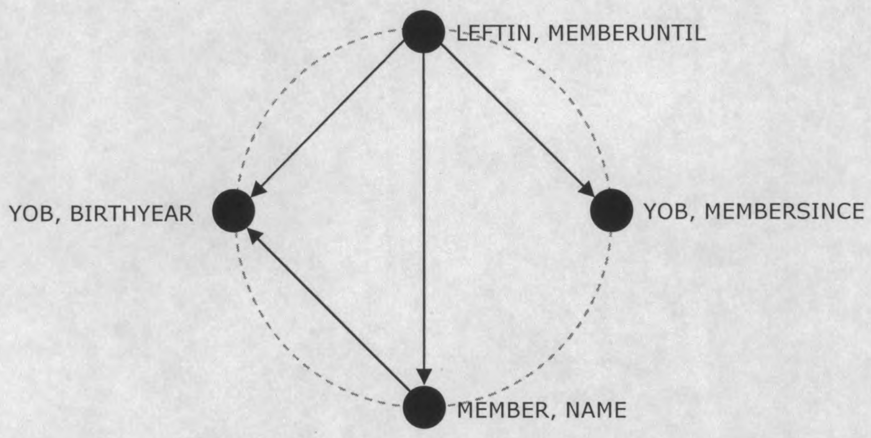


Fig. 6.2.3 Visualization of binary INDs. Nodes are the unary INDs from Fig. 6.1.5. Edges are the binary INDs from Fig. 6.2.1

The binary IND also contains the support value and it is represented by the color of the edge. The generation of color based upon the support value remains the same as we discussed in the case of unary INDs. The color here is black due to same support value.

The observer can now use this visualization to decide which binary INDs (edges) in this graph represent real-world patterns, i.e., relationships between two pairs of attributes across two relations.

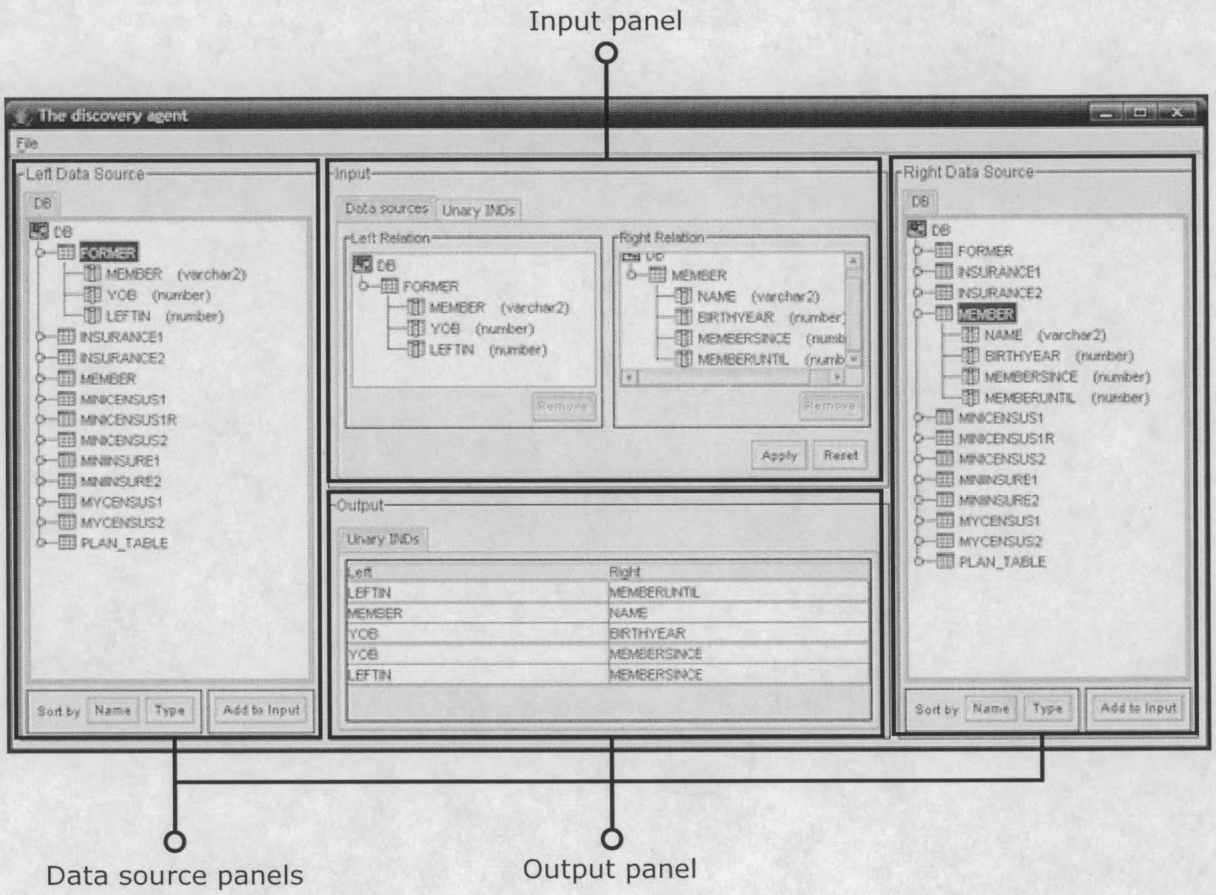
In this section, we discuss about visualization techniques used by the discovery agent to display unary and binary INDs. However the agent also provides user interface controls to an observer so that he/she can make significant changes. In the next section, we will discuss about the actual user interface of the discovery agent and how it is helpful to an observer making right decisions.

7. Agent walkthrough

So far we have discussed the design and execution flow of the discovery agent. In this section, we discuss about user interfaces layer of the agent. Though the agent has many screens from which it receives input from an observer, here we discuss three screens which play a major role in the execution of the agent. By looking at these three screens, a reader can visualize how the discovery agent works. These screens are listed as,

1. The agent's main screen
2. A dialog to visualize unary INDs
3. A dialog to visualize binary INDs

7.1 The main screen



Left	Right
LEFTIN	MEMBERUNTIL
MEMBER	NAME
YOB	BIRTHYEAR
YOB	MEMBERSINCE
LEFTIN	MEMBERSINCE

Fig. 7.1.1 The agent's main screen

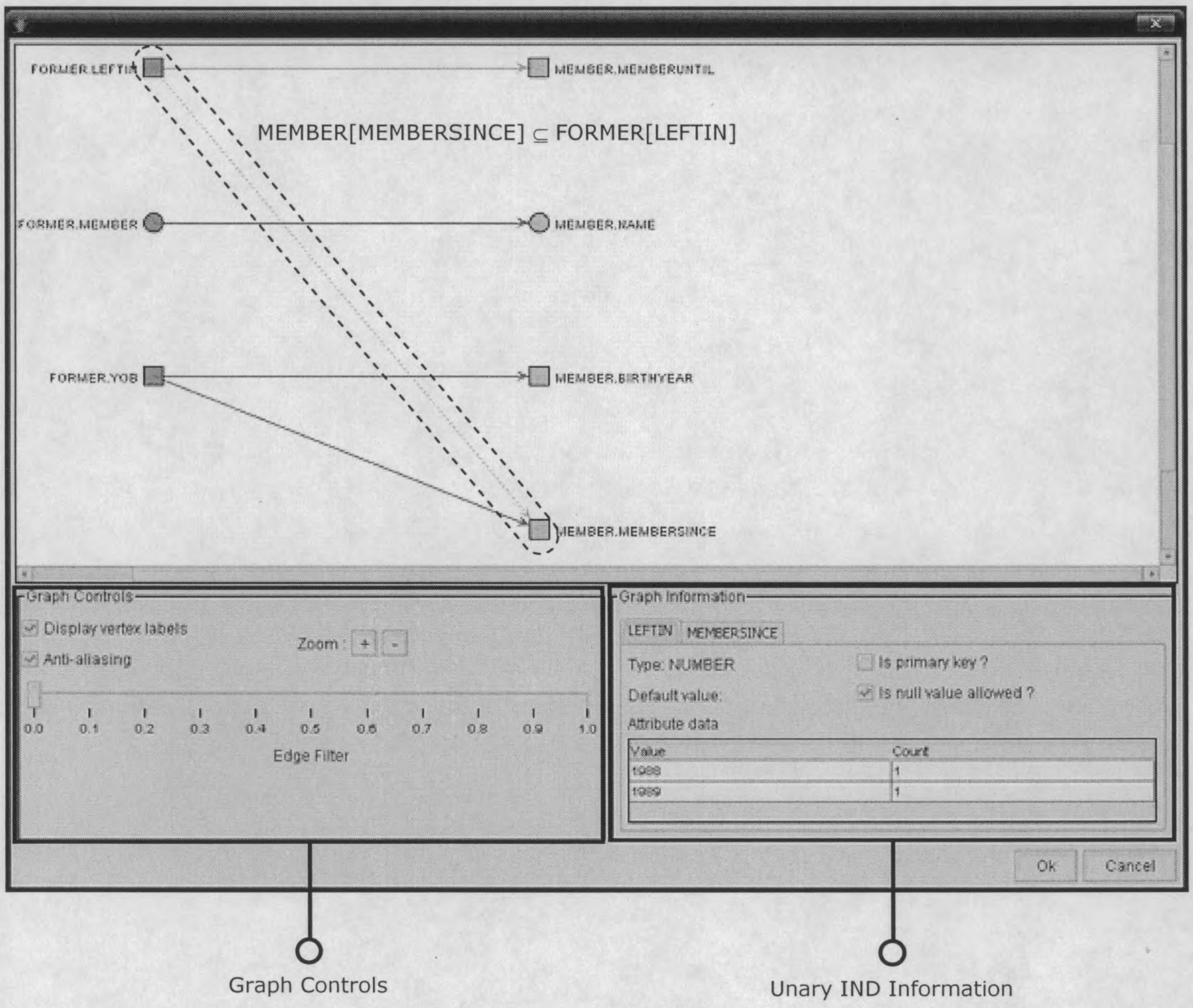
The main screen of the discovery agent is dividing into three main components:

1. The data source panels
   - There are two data sources panels from which an observer can choose input relations.
   - It displays a tree like structure for the data source(s). The root of the tree is the name of the data source. At the first child level, all relations belonging to this data source are displayed. All relation nodes have their attributes as child nodes. The tree structure provides ease of navigation to an observer.
   - The agent runs the discovery process with two input relations, one of which is at the left side and the other one is at the right side. Based upon this fact, we designed the main screen in such a way that the left data source panel provides relations for the left side and the right data source panel provides relations for the right side.
   - By arranging data source panels into left and right, it will be easy for an observer to choose input relations for the discovery process.
   - An observer can also perform attributes selection and thus can run the discovery process on attributes that he/she wants. To do this, an observer first selects attributes from any relation and then he/she can add them to the input panel by clicking on the "Add to Input" button.
   - Each data source panel provides sorting facility by which an observer can sort the data source tree based upon attributes name or attribute type (NUMBER, VARCHAR, TEXT and so on). It helps an observer deciding which relations he/she wants to include.

2. The input panel
   - The input panel has a tabbed interface to display input to various subsections of the phase-I of the algorithm.
   - The first tab displays two input relations in a tree-like structure. It also provides the "remove" button by which an observer can remove relations or attributes.
   - When an observer is done selecting relation and attributes, he/she starts the discovery process by clicking on the "Apply" button.
   - The "Reset" button clears out the input section of the current tab.
   - After clicking on "Apply", the agent runs the algorithm and discovers all unary INDs. It then displays all unary INDs into the output panel and also into the input panel in another tab (called "Unary INDs"). The "Unary INDs" tab displays all unary INDs in tabular form. This tab also gives control to analyze all unary INDs in graphical form which we will discuss in next few minutes.
   - In a similar manner, the input panel also displays a binary INDs tab and provides control to analyze them.
   - The main purpose behind the input panel is to display the section where the algorithm needs a human assistance.

3. The output panel
   - The output panel displays an output of the algorithm at each sub level of Phase-I into a tabbed interface.
   - As a first step, it displays all unary INDs into tabular form in a tab named "Unary INDs".
   - In the same manner, it displays all binary INDs into other tab named "Binary INDs"
   - This section only displays the intermediate results of the algorithm for an observer. An observer can only view the information here. He/she can only change the information in the input panel.

7.2 A dialog to visualize unary INDs



[✓] Display vertex labels
[✓] Anti-aliasing

Type: NUMBER
Default value:
[ ] Is primary key ?
[✓] Is null value allowed ?
Attribute data

Value	Count
1988	1
1989	1

Fig. 7.2.1 Unary IND visualization dialog

- Built on the visualization technique that we discussed before, fig. 7.2.1 displays the agent's dialog by which an observer can visualize all unary INDs.
- With the IND visualization, the screens also have few user controls as follows:

1. Edge Filter:

   It is most helpful control for an observer. With the help of a slider, an observer can exclude edges (which represent unary INDs here) with low support values. When an edge is excluded, the corresponding IND is also excluded from future computations. Excluded edges are drawn using light dotted lines. For example, when the slider points on 0.3, the graph filters out edges (INDs) that have support values less than 0.3.

2. Display vertex labels

   Using this control, an observer can show/hide vertex labels on the graph.

3. Anti-aliasing

   Using this control, an observer can turn on/off the graph anti-aliasing. It is helpful to view the graph in low resolution display monitors.

4. Zoom control

   As the name suggests, this controls helps an observer zooming in and out in the graph. When you have a large number of nodes and edges, this feature is helpful to view details of the graph.

- An observer also can exclude edges manually by a single right mouse click on a normal edge. By default, all edges (INDs) are included in the graph and they are displayed using dark solid lines. When you right-click on a solid (included) edge explicitly, the edge (IND) will be marked as excluded and displayed in a light dotted pattern. When you click on a dotted (excluded) edge, it is included and drawn as a dark solid shape. Using this feature, an observer is able to manually select/deselect INDs which he/she wants to include in- or exclude from computations.

- Apart from the control section, the dialog also displays graph related information in "Graph Information" section in Fig. 7.2.1.

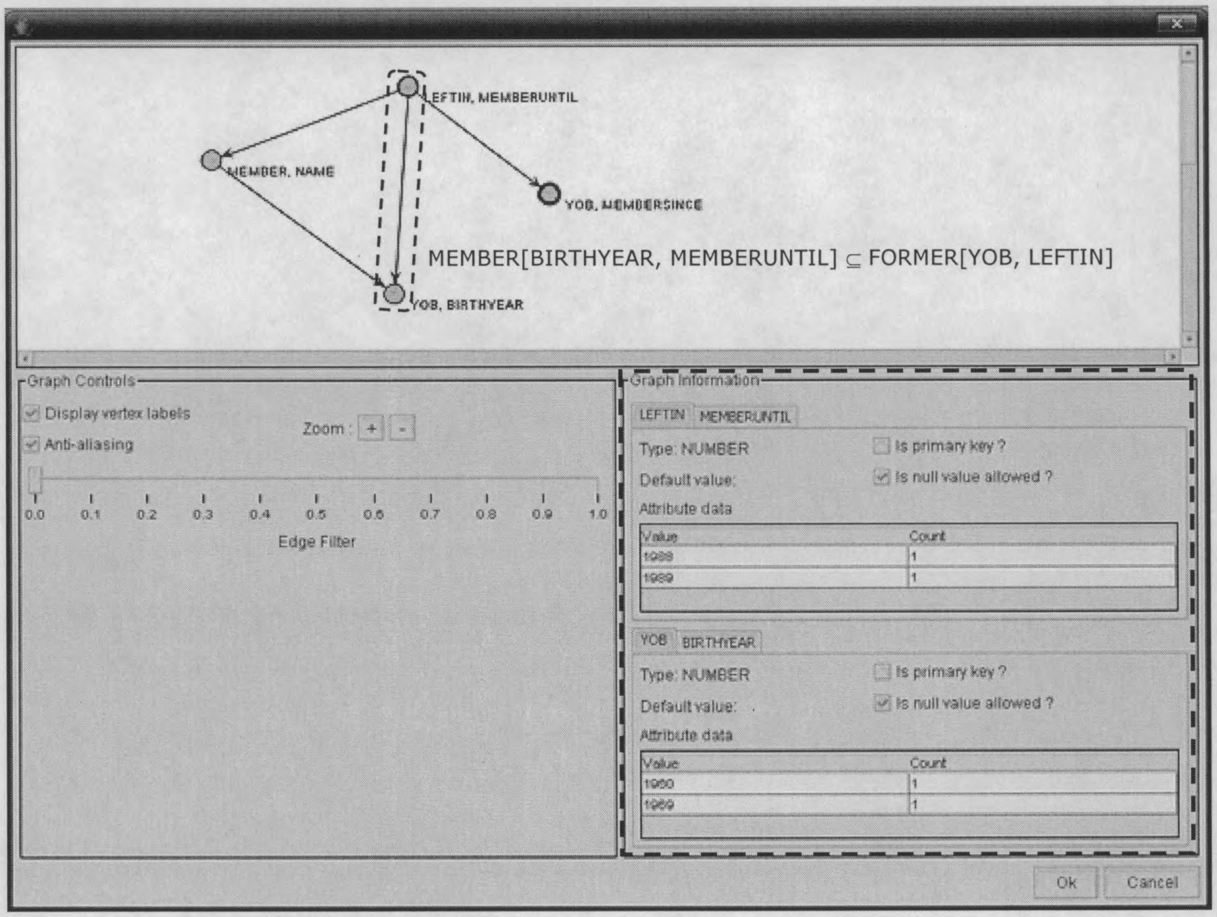
- When you select any node in the graph, this section displays information such as type, default value, whether it is primary key or not and whether the null value is allowed for that particular attribute represented by the node. Also it displays distinct values of the attribute ("Value" column in fig. 7.2.1) and how many times a particular value appears ("Count" column in fig. 7.2.1) for that attribute in the relation. With the help of this information, an observer can think about an actual entity represented by that attribute.

- In the similar manner, when you select any edge, it displays information about attributes that are involved in that particular IND. In fig. 7.2.1, the information about unary IND MEMBER[MEMBERSINCE] $\subseteq$ FORMER[LEFTIN] is displayed. As you can see there are two tabs LEFTIN and MEMBERSINCE which represents two attributes LEFTIN and MEMBERSINCE which are involved in the IND. When you select any of the tabs, it will display all information about that particular attribute.

- With the help of this information, an observer can have necessary information about attributes to decide whether an IND is also *relevant (i.e., non-spurious)* in the real world. The tab layout provides the ease of the usage and helps an observer with his/her decision.

- When you click on any node of the graph, the graph will highlight that node and it will also highlight all connected nodes. This feature is very helpful to determine the number of relationships in which the particular node (attribute or unary IND) is involved.

7.3 A dialog to visualize binary INDs



MEMBER[BIRTHYEAR, MEMBERUNTIL] ⊆ FORMER[YOB, LEFTIN]

[✓] Display vertex labels
[✓] Anti-aliasing

LEFTIN | MEMBERUNTIL

Type: NUMBER
Default value:
Attribute data

[ ] Is primary key ?
[✓] Is null value allowed ?

Value	Count
1988	1
1989	1

YOB | BIRTHYEAR

Type: NUMBER
Default value: .
Attribute data

[ ] Is primary key ?
[✓] Is null value allowed ?

Value	Count
1960	1
1969	1

Fig. 7.3.1 Binary IND visualization dialog

- Fig. 7.3.1 displays the binary IND visualization dialog.
- We already discussed the visualization techniques for binary INDs. The graph is constructed here is based upon those techniques only.
- Also the graph controls serve the same purpose as they served in unary IND visualization dialog. The graph interacts with an observer in the same manner as we discussed in the unary IND visualization dialog. Due to fact that here we are considering binary INDs, there are few changes in visually which we discuss next.
- Here the node in the graph now represents a unary IND. So by selecting any node, an observer can view all the attributes involved in the unary IND in graph information section. Since we have two attributes involved in any unary IND, there are two tabs representing them in graph information section (similar to what we discussed in unary IND visualization dialog).
- Here an edge represents a binary IND. So when you select any edge, an observer will see two parts in the "Graph Information" section which displays two unary INDs. As

shown in fig. 7.3.1, the binary IND MEMBER[BIRTHYEAR, MEMBERUNTIL] $\subseteq$ FORMER[YOB, LEFTIN] is selected in the graph. Since it is generated from two unary INDs MEMBER[BIRTHYEAR] $\subseteq$ FORMER[YOB] and MEMBER[MEMBERUNTIL] $\subseteq$ FORMER[LEFTIN], these two INDs are displayed in the "Graph Information" section.

- The dialog uses same interface as we discussed in unary IND visualization to maintain consistency in viewing.

The goal of the agent is to provide a superior user interface by which an observer can review the information quickly which helps him/her making intelligent decision. We believe that by using proper display techniques and with the ease of usage, an observer will be able to figure out and remove spurious INDs from further computation which further increases the reliability of the result of the overall discovery process and the speed of the discovery algorithm.

8. Related work

Inclusion Dependencies and Functional Dependencies are two most significant integrity constraints in relational databases. They play a major role in data integration (Calì, Calvanese, De Giacomo and Lenzerini, 2004; Calvanese, De Giacomo, Lenzarini, Nardi and Rosati, 2001; Fernandez, Florescu, Kang, Levy and Suciu, 1998; Fernandez, Florescu, Levy and Suciu, 1999; Arenas, Bertossi and Chomicky, 1999; Fagin, Kolaitis, Miller and Popa, 2003; Popa, Velegrakis, Miller, Hernández and Fagin, 2002). The problem of integrating two data sources has been studied for a long time and in this section we provide an overview of the work that has been done in many area to achieve the goal of integration.

There are many projects built for integration of heterogeneous databases. ARTEMIS (Castano, de Antonellis & de Capitani de Vemercati, 2001; Castano, De Antonellis, 1999) and Microsoft's CUPID (Madhavan, Bernstein & Rahm, 2001) are example of performing integration using schema-based matching approach. ARTEMIS is used as a component of a heterogeneous database mediator, called MOMIS (Mediator envirOnment for Multiple Information Sources) (Bergamaschi, Castano and Vinicini, 1999; Beneventano, Bergamaschi, Castano, Corni, Guidetti, Malvezzi, Melchiori and Vincini, 2000; Bergamaschi, Castano, Vincini, Beneventano, 2001). CUPID is a hybrid matcher based on both element- and structure-level matching. SemInt (Li & Clifton, 2000) and LSD (Doan, Domingos & Halevy, 2001) are example of performing integration using instance-based matching approach.

Casanova, Fagin & Papadimitriou (1982) defined the term inclusion dependency and worked on rules dealing with how to obtain inclusion dependencies, how to derive other inclusion dependencies from already derived inclusion dependencies and how to obtain inclusion dependencies from functional dependencies. In their work, they also established the inference rules which form the basis of all algorithm for the discovery of high-arity inclusion dependencies.

The discovery problem also appears in data mining, namely the problem of the discovery of association rules (Agrawal and Ramakrishnan, 1994). Much work in the literature deals with implementation issues and optimizations on Agrawal's algorithm (Bayardo, 1998; Lin and Kedem, 1998; Han et al., 2000; Zaki, 2000) or generalizations of the problem (Mannila and Toivonen, 1997). (Koeller and Rundensteiner, 2004)

Inclusion dependency discovery can also be used in the database design. Ling & Goh (1992) demonstrated how inclusion dependencies can be used to model important database constraints leading towards better design of database schemas. Ling & Goh also proposed a new normal form, called *Inclusion Normal Form (IN-NF)*. Levene & Vincent (2000) further studies normal forms based upon INDs and derived the formal definition of new form *Inclusion Dependency Normal Form (IDNF)* by providing sufficient and necessary semantics.

Functional Dependencies are also important in data integration. Lim & Harrison (1997), Bell & Brockhausen (1995), Huhtala et al. (1998), Knobbe & Adriaans (1996) and Savnik & Flach (1993) have all worked on the discovery of Functional Dependencies and showed the importance of the result in the data integration. While functional and inclusion dependencies are related to each other, the results of this thesis apply strictly to inclusion dependency discovery only.

Kantola, Mannila, Räihä & Siirtola (1992), Bell & Brockhausen (1995) and Knobbe & Adriaans (1996) provide some theoretical basis for the discovery of various patterns, including functional and inclusion dependencies, in databases. As we have already discussed, *FIND*$_2$ by Dr. Andreas Koeller (Koeller & Rundensteiner, 2003) and *Zigzag* by F. De Marchi (De Marchi, Lopes & Petit, 2002) are most recent algorithms developed in the field of inclusion dependency discovery.

Human intelligence has been proven as an effective solution for achieving desired results. Vijayshankar Raman and Joseph M. Hellerstein demonstrate the use of human intelligence in an interactive data cleaning system, Potter's Wheel (Raman, Hellerstein, 2001). "The Control Project" (Hellerstein, Avnur, Chou, Hidber, Olston, Raman, Roth, Haas, 1999) uses human-computer interaction techniques in the data analysis.

In this thesis, we combine the area of inclusion dependency discovery with the idea of independent software agents. Lieberman (1997) defined autonomous and interface agents and showed the importance of the collaboration between them. He explained design principles of such agents with the Letizia system. The WebWatcher from Armstrong, Freitag, Joachims & Mitchell (1995) and the LIRA from Balabanovic & Shoham (1995) are closest projects to Letizia which involve the interaction between agents and users.

In this thesis, we proposed an approach to make an agent built on the concept used by autonomous interface agents (Lieberman, 1997) for the inclusion dependency discovery process. Up to our knowledge, the closest project to our work is the *DBA Companion Project* which is maintained by Fabien De Marchi, Stephane Lopes and Jean-Marc Petit. The *DBA Companion Project* provides the understanding of logical database constraints from which logical database tuning can be achieved. This project differs from the discovery agent in the sense that it involves functional dependencies while the discovery agent only concentrates on inclusion dependencies. The goal of the discovery agent is to provide a unified data integration solution based upon the collaborative efforts of the inclusion dependency discovery algorithm and the human intelligence.

9. Conclusion and future enhancements

The discovery of inclusion dependencies is a hard problem in the field of the data integration. In thesis, we showed that with collaborative efforts of suitable algorithm intelligence and proper human expertise, a software agent can produce effective result towards finding an applicable solution of an NP-complete problem.

In this thesis, we propose an approach of using agents in the discovery of inclusion dependencies. The discovery agent that we presented in this thesis demonstrates the collaborative efforts of the algorithm and the observer to deliver a comprehensive inclusion dependency discovery solution.

There is a large gap between research solutions and industrial applications based on those solutions. Here we are trying to reduce this gap by introducing the discovery agent and its interactive and semi-automated process of inclusion dependency discovery. The aim of the discovery agent to provide a data integration solution based on the discovery of inclusion dependency.

In this thesis, we discussed many characteristics of the discovery agent. We demonstrated the effectiveness and usefulness of the agent by implementing its small scale prototype. Our work mainly involves the user interface layer but the development of the agent is not limited to it. Compatibility between different forms of data sources (e.g,. XML databases rather than just relational databases), ability to work with two different forms of data sources at a same time, and better interaction with the discovery algorithm used are some of the major future enhancements of the discovery agent.

10. References

Agrawal, R. and Ramakrishnan, S. (1994). 'Fast Algorithms for Mining Association Rules'. *Proc. Intl. Conf. on Very Large Databases (VLDB).* pp. 487-499.

Arenas, M., Bertossi, L.E. and Chomicky, J. (1999). Consistent query answers in inconsistent databases. *Proceedings of the 18th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'99).* pp. 68-79.

Armstrong, R., Freitag, D., Joachims, T.& Mitchell, T. (1995). WebWatcher: A Learning Apprentice for the World Wide Web. In *AAAI Spring Symposium on Information Gathering, Stanford, CA.*

Balabanovic, M. & Shoham, Y. (1995). Learning Information Retrieval Agents: Experiments with Automated Web Browsing. In *AAAI Spring Symposium on Information Gathering, Stanford, CA.*

Batini, C., Lenzerini, M., and Navathe, S.B. (1986). A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323-364.

Bayardo, R. J. (1998), 'Efficiently mining long patterns from databases'. In: L. Haas and A. Tiwary (eds.): *Proceedings of SIGMOD,* Vol. 27(2). pp. 85-93.

Bell, S. and Brockhausen, P. (1995). Discovery of Data Dependencies in Relational Databases. *Technical Report. LS-8 Report 14, University of Dortmund.*

Beneventano, D., Bergamaschi, S., Castano, S., Corni, A., Guidetti, R., Malvezzi, G., Melchiori, M. and Vincini, M. (2000). Information integration: the MOMIS project demonstration. *Proceedings of 26th International Conference on Very Large Databases (VLDB).* pp. 611-614.

Bergamaschi, S., Castano, S. and Vincini, M. (1999). Semantic integration of semistructured and structured data sources. *ACM SIGMOD Record* 28(1). pp. 54-59.

Bergamaschi, S., Castano, S.,Vincini, M. and Beneventano, D. (2001). Semantic integration of heterogeneous information sources. *Data Knowledge Engineering* 36(3). pp. 215-249.

Calvanese, D., De Giacomo, G., Lenzarini, M., Nardi, D. and Rosati, R. (2001). Data integration in data warehousing. *International Journal of Cooperative Information Systems* 10(2). pp. 237-271.

Calì, A., Calvanese, D., De Giacomo, G. and Lenzerini, M. (2004). Data integration under integrity constraints. *Information Systems, 29(2)*. pp. 147-163

Casanova, M.A. and de Sá J.E.A. (1983). Designing entity-relationship schemes for conventional information systems. *Proceedings of the Third International Conference on the Entity-Relationship Approach to Software Engineering*. pp. 265-277.

Casanova, M.A., Fagin R. and Papadimitriou C.H. (1982). 'Inclusion Dependencies and their Interaction with Functional Dependencies'. *Proceedings of ACM Conference on Principles of Database Systems (PODS)*. pp. 171-176.

Casanova, M.A., Tucherman, L. and Furtado, A-L. (1988). Enforcing inclusion dependencies and referential integrity. *Proceedings of the 14th International Conference on Very Large Databases*. pp. 38-49.

Castano, S., De Antonellis, V. (1999). A schema analysis and reconciliation tool environment. *Proceedings of International Database Eng Appl Symp (IDEAS), IEEE Computer*. pp. 53-92.

Castano, S., de Antonellis, V. & de Capitani de Vemercati, S. (2001). Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering, 13(2)*. pp. 277-297

Cheng, Q., Gryz, J., Koo, F., Leung, T.Y.C., Liu, L., Qian, X. and Schiefer, B. (1999). Implementation of two semantic query optimization techniques in DB2 universal database. *Proceedings of the 25th International Conference on Very Large Databses*. pp. 687-698.

Doan, A. H., Domingos, P. & Halevy, A. Y. (2001). Reconciling schemas of deiparate data sources: A machine-learning approach. *Proceedings of the ACM SIGMOD International Conference on Management of Data, USA*. pp. 509-520.
Date., C.J. (1981). Referential integrity. *Proceedings of Seventh International Conference on Very Large Databases*. pp. 2-12.

De Marchi, F., Lopes S., and Petit J.-M. (2002). 'Efficient Algorithms for Mining Inclusion Dependecies'. *Proceedings of International Conference on Extending Database Technology (EDBT) pp. 464-476.*

Fagin, R., Kolaitis, P.G., Miller, R.J. and Popa, L. (2003). Data exchange: semantics and query answering. *Proceedings of the 9th International Conference on Database Theory (ICDT 2003)*. pp. 207-224.

Fernandez, M.F., Florescu, D., Levy, A. and Suciu, D. (1999). Verifying integrity constraints on web-sites. *Proceedings of the 16h International Joint Conference on Artificial Intelligence (IJCAI'99)*. pp. 614-619.

Fernandez, M.F., Florescu, D., Kang, J., Levy, A. and Suciu D. (1998). Catching the boat with Strudel: experiences with a web-site management system. *Proceedings of the ACM SIGMOD International Conference on Management of Data.* pp. 414-425.
Han, J., Pei, J. and Yun, Y. (2000). 'Mining Frequent Patterns without Candiate Generation'. *SIGMOD Record (ACM Special Iterest Group on Management of Data)* 29(2). pp. 1-12.

Harary, F. (1994). *Graph Theory.* Reading, MA: Addison-Wesley, 1994.

Hellerstein, J.M., Avnur, R., Chou A., Hidber C., Olston C., Raman V., Roth T. and Haas P.J. (1999). Interactive Data Analysis: The Control Project. *IEEE Computer Society Press.* pp. 51-59.

Huhtala, Y., Karkkainen, J., Porkka, P. & Toivonen (1998). Efficient discovery of functional and approximate dependencies using partitions. In *Proceedings of IEEE International Conference on Data Engineering.* pp. 392-401.

Hull, R. (1997). Managing semantic heterogeneity in databases: A theoretical perspective. *Proceedings of PODS-97.*

Godfrey, P., Grant, J., Gryz, J. and Minker, J. (1998). Integrity constraints: semantics and applications. *J. Chomicki, G. Saake (Eds.), Logics for Databases and Information Systems, Kluwer Academic Publishes, Boston.* pp. 265-306.

Gryz, J. (1998). Query folding with inclusion dependencies. *Proceedings of the 14th IEEE International Conference on Data Engineering.* pp. 126-133.

Kantola M., Mannila H., Räihä, K.J., and Siirtola H. (1992). Discovering functional and inclusion dependencies in relational databases. *International J. of Intelligent Systems,* 7:591-607.

Klettke, M. (1999). Reuse of database design decision. *Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling. Vol. 1727.* pp. 213-224.

Knobbe, A.J. and Adriaans, P.W. (1996). Discovering Foreign Key Relations in Relational Databases. *Proceedings of the 13th European Meeting on Cybernetics and Systems Research, Vienna.* Pp 961-966

Knoblock, C. and Levy, A. (eds.) (1995) *AAAI Symposium of Information Gathering from Heterogeneous, Distributed Environments,* number SS-95-08 in AAAI Spring Symposium Series.

Koeller, A. (2001). Integration of Heterogeneous Databases: Discovery of Meta Information and Maintenance of Schema Restructuring Views. *PhD Thesis*, Worcester Polytechnic Institute, Worcester, MA, USA.

Koeller, A. and Rundensteiner, E.A. (2002). 'Discovery of High-Dimensional Inclusion Dependencies'. Technical Report WPI-CS-TR-02-15, Worcester Polytechnic Institute, Dept. of Computer Science.

Koeller, A. and Rundensteiner, E.A. (2003). 'Discovery of High-Dimensional Inclusion Dependencies'. *Proceedings of IEEE International Conference on Data Engineering.* Banglor, India. pp. 683-685.

Koeller, A. and Rundensteiner, E.A. (2004). 'Scalable Discovery of Inclusion Dependecies'. *Unpublished Report.*

Krishnamurthy, R., Litwin, W. and Kent, W. (1991). Language features for interoperability of databases with schematic discrepancies. *SIGMOD Record (ACM Special Interest Group on Management of Data), 20(2)*, pp. 40-49.

Laurent, D., Lechtenbörger, J., Spyratos, N, and Vossen, G. (1999). Complements for data warehouses. *Proceedings of the 15th International Conference on Data Engineering.* pp. 490-499.

Leven, M. & Vincent, M.W. (2000). Justificaiton for Inclusion Dependency Normal Form. *In IEEE Transactions on Knowledge and Data Engineering (TKDE),* 12(2), pp. 281-291.

Li, W. & Clifton, C. (2000). SemInt: A tool for identifying attribute correspondences in heterogeneous databases using neural network. *Journal of Data and Knowledge Engineering, 33(1).* pp. 49-84.

Lieberman, Henry (1997). "Autonomous interface agents". *Proceedings of the SIGCHI conference on Human factors in computing systems.* pp. 67 – 74

Lim, W. & Harrison, J. (1997). Discovery of Constraints from Data, for Information System Reverse Engineering. In *Proc. Of Australian Software Engineering Conference (ASWEC '97), Sydney, Australia.*

Lin, D.-I and Kedem, Z. M. 1998. 'Pincer Search: A new algorithm for discovering the maximum frequent set'. *International Conference on Extending Database Technology, EDBT'98.* pp. 385-392.

Ling, T W and Goh, C.H. (1992). "Logical database design with inclusion dependencies". *In Proceedings of the Eight International Conference on Data Engineering*, pp. 642-649.

Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with CUPID. *Proceedings of the 27th International Conference on Very Large Databases.* pp. 49-58.

Mannila, H. and Toivonen, H. (1997). ‘Levelwise Search and Borders of Theories in Knowledge Discovery’. *Data Mining and Knowledge Discovery* 1(3). pp. 241-258

Mannila, H. and Räihä, K.J. (1986). Inclusion dependencies in database design. *Proceedings of the Second IEEE International Conference on Data Engineering*. pp. 713-719.

Markowitz, V.M. and Makowsky, J.A. (1990). Identifying extended entity-relationship object structures in relational schemas. *IEEE Trans. Software Engineering*. pp. 777-790.

Myers, B.A. (1998) "A Brief History of Human Computer Interaction Technology." ACM interactions. Vol. 5, no. 2, March, 1998. pp. 44-54

Paulley, G.N. and Larson, Per-Åke. (1994). Exploiting uniqueness in query optimization. *Proceedings of the 10th International Conference on Data Engineering*. pp. 68-79.

Petit, J-M., Toumani, F. and Kouloumdjian, J. (1995). Relational database reverse engineering: a method based on query analysis. *International Journal of Cooperative Information Systems 4*. pp. 287-316.

Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A. and Fagin, R. (2002). Translating Web data. *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2002)*. pp. 598-609.

Qian, X. (1996). Query folding, in: S.Y.W. Su (Ed.). *Proceedings of the 12th IEEE International Conference on Data Engineering*. pp. 48-55.

Quass, D., Gupta, A., Mumick, I.S. and Widom, J. (1996). Making views self-maintainable for data warehousing. *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*. pp. 158-169.

Rahm E., Bernstein P. (2001). A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases, 10(4)*, 334-350

Raman, V. and Hellerstein, J.M. (2001). Potter's Wheel: An Interactive Data Cleaning System. *Proceedings of the 27th International Conference on Very Large Data Bases*. pp. 381-390.

Savnik, I. & Flach, P.A. (1993) Bottom-up induction of functional dependencies from relations. IN Piatetsky-Shapiro, G., editor, *Proc. Of AAAI-93 Workshop: Knowledge Discovery in Database*. pp. 174-185.

Widom, J. (1995). Special issue on materialized views and data warehousing. *IEEE Bulletin on Data Engineering,* 18(2).

Wiederhold, G. (1996) Special issue: Intelligent integration of information. *Journal of Intelligent Information Systems.* 6(2/3).

Zaki, M.J. (2000). 'Scalable Algorithms for Association Mining'. *IEEE Transactions on Knowledge and Data Engineering (TKDE),* 12(3). pp. 372-390.