Discovery of High-Dimensional Inclusion Dependencies[*]

Andreas Koeller
Dept. of Computer Science
Montclair State University
Upper Montclair, NJ 07043, USA
Andreas.Koeller@montclair.edu

Elke A. Rundensteiner
Dept. of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609, USA
rundenst@cs.wpi.edu

Abstract

Determining relationships such as functional or inclusion dependencies within and across databases is important for many applications in information integration. When such information is not available as explicit meta data, it is possible to discover potential dependencies from the source database extents. However, the complexity of such discovery problems is typically exponential in the number of attributes.

We have developed an algorithm for the discovery of inclusion dependencies across high-dimensional relations in the order of 100 attributes. This algorithm is the first to efficiently solve the inclusion-dependency discovery problem. This is achieved by mapping it into a progressive series of clique-finding problems in k-uniform hypergraphs and solving those. Extensive experimental studies confirm the algorithm's efficiency on a variety of real-world data sets.

1. Introduction

In this work, we are concerned with the discovery of meta-information (dependencies and interrelationships) in databases, and in particular with the discovery of inclusion dependencies (INDs).

Due to the nature of data and its generation, information is often stored in multiple places, with large amounts of redundancy (for example across different departments of a large enterprise or across multiple companies in the same field of business). When integrating data sources that are likely to be (even partly) redundant, such as in the EVE view integration system [6], a method to *discover* such redundancies is needed.

[*] This work was performed while Andreas Koeller was a research assistant at Worcester Polytechnic Institute. The work was supported in part by the NSF NYI grant #IRI 97–96264, the NSF CISE Instrumentation grant #IRIS 97–29878, and the NSF grant #IIS 9988776.

Also, applications in which data about similar real-world objects is collected independently will benefit greatly from redundancy discovery. For example, medical or pharmacological databases could be compared for similarities or overlaps, which could provide important information relevant for the discovery of treatments for certain diseases.

In general, the discovery of INDs will be beneficial in any effort to integrate or compare unknown databases. A manual extraction of INDs by domain experts is usually not feasible due to the large number of information sources in the world, the potentially high number of attributes in real-world relations, and a widespread lack of reliable meta-information about legacy databases. Furthermore, the general IND discovery problem has been shown to be NP-hard as a function of the number of attributes in the relations to be compared [4].

To the best of our knowledge, the algorithm we are presenting is the first that solves the IND-finding problem for large numbers of attributes. We are aware of only one other solution of the IND discovery problem, using a levelwise enumeration algorithm [3]. It shows good performance for relations with few attributes but does not scale well to numbers of attributes (and in particular maximal IND sizes) beyond 10.

The IND discovery problem is loosely related to the problem of association rule mining (ARM, [1]). However, with the exception of [7], ARM algorithms are *levelwise* algorithms in the sense that a "frequent itemset" of length k is only discovered after all itemsets of lengths $1 \dots (k-1)$ have in some sense been evaluated. Typically, ARM algorithms become feasible by using additional properties of association rules (most notably the concept of *support*), which are not available to us. For this reason, an adaptation of such algorithms to IND discovery would show significantly higher complexity than algorithm FIND_2.

In the remainder of this paper, we sketch the main idea of the FIND_2 process while a more extensive description including mathematical background and algorithmic details is given in [5].

2. Background

Inclusion dependencies are defined as below.

Definition 1 (IND) *Let $R[a_1, a_2, \dots, a_n]$ and $S[b_1, b_2, \dots, b_m]$ be (projections on) two relations. Let X be a sequence of k distinct attribute names from R and Y a sequence of k distinct attribute names from S, with $1 \leq k \leq \min(n, m)$. Then an **inclusion dependency (IND)** σ is an assertion of the form $\sigma = R[X] \subseteq S[Y]$. k is called the **arity** of σ. An IND $\sigma = (R[a_1, \dots, a_k] \subseteq S[b_1, \dots, b_k])$ is **valid** between two relations R and S if the sets of tuples in R and S satisfy the assertion given by σ.*

One very important observation on INDs is that a k-ary IND with $k > 1$ naturally **implies** a set of m-ary INDs, with $1 \leq m \leq k$. That is, for a given valid IND $\sigma = R[\overline{A}] \subseteq S[\overline{B}]$ [1], the IND $\sigma' = R[\overline{A'}] \subseteq S[\overline{B'}]$ will be valid for any subset $\overline{A'} \subseteq \overline{A}$ and its corresponding subset $\overline{B'} \subseteq \overline{B}$. Such a set of m-ary INDs implied by a k-ary IND has a cardinality of $\binom{k}{m}$ and is denoted by Σ^k_m. Note that the validity of all implied k-ary INDs of a given IND σ is a necessary but not a sufficient condition for the validity of σ. For example, $(R[A_1] \subseteq S[B_1]) \wedge (R[A_2] \subseteq S[B_2]) \wedge (R[A_3] \subseteq S[B_3])$ does not imply $R[A_1, A_2, A_3] \subseteq S[B_1, B_2, B_3]$.

3. Mapping the IND Discovery Problem to a Graph Problem

The worst-case complexity of the problem is determined by the number of possible distinct INDs between two relations, which is exponential in the number of attributes in those relations [5, 4]. In our work, we instead make use of the fact that it is possible to find a minimal cover of valid INDs (i.e., a set of INDs from which all valid INDs can be derived by implication) without even enumerating all valid INDs, reducing the complexity significantly.

We propose a mapping of our problem into a more tractable graph problem. We use **k-uniform hypergraphs** which are graphs in which each edge is incident to exactly k nodes. Thus, standard undirected graphs can be considered "2-uniform hypergraphs". Furthermore, we extend the concept of *clique* (maximal connected subgraph) to hypergraphs.

Definition 2 (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. As a special case, a single node with no adjacent edges is a hyperclique of cardinality 1.*

[1]The notation $\overline{A}$ means a set of attributes.

In analogy to above, a clique is a hyperclique in a 2-hypergraph. The mapping from our problem to a graph problem is achieved as follows:

We first map the set of valid INDs to a set of hypergraphs G_m $(2 \leq m < k)$, by making all k-ary valid INDs hyperedges in a k-uniform hypergraph. The nodes of all hypergraphs (for any k) are formed by the unary INDs. For example, the first hypergraph for $k = 2$ has as its nodes all valid *unary* INDs, and as its edges all valid *binary* INDs.

We then show that, for $m = 2 \dots k-1$, any set Σ^k_m of INDs implied by a valid σ_k maps to a hyperclique in the corresponding hypergraph G_m. In other words, the only candidates for valid high-arity INDs are those that correspond to cliques in k-uniform hypergraphs for small k. Those graphs can be constructed after a relatively small number of IND validity checks on INDs of very small arity.

3.1. The Clique-Finding Problem

The **Clique-Finding Problem** (also called the Maximum Clique Problem) is a well known NP-complete graph problem. Efficient algorithms for reasonably sized graphs (i.e., 2-hypergraphs, with up to about 100 nodes) are given in the literature, e.g., [2]. With our definition of hypercliques (Def. 2), the Clique-Finding Problem extends naturally to k-hypergraphs.

The NP-complexity of the clique-problem is mainly due to the exponential number of possible cliques in a graph, although there are polynomial-time algorithms for some cases. We have developed and implemented an algorithm called HYPERCLIQUE that finds cliques in k-uniform hypergraphs and, while NP-complete, shows satisfactory performance for relatively sparse graphs with few cliques. Due to space limitations, we refer to [5] for details.

4. Algorithm FIND$_2$

We now briefly sketch out the algorithm FIND$_2$ (Fig. 1) which applies clique- and hyperclique-finding techniques to find inclusion dependencies (INDs). Full details and derivations can be found in [5]. FIND$_2$ takes as input two relations R and S, with k_R and k_S attributes, respectively and returns a generating set of INDs between attributes from R and S. The algorithm proceeds by first exhaustively validating unary and binary INDs, thus forming the first (2-uniform) hypergraph (Lines 01-02). A clique-finding algorithm then determines all higher-arity INDs candidates (Line 06). Since the clique property is necessary but not sufficient for the validity of a higher-arity IND, each IND candidate thus discovered must also be checked for validity (Line 09). Each IND that tests invalid (but is a clique in the 2-hypergraph) is broken down into its implied 3-ary INDs, which then form the edges of a 3-hypergraph (Line

11). Edges corresponding to invalid INDs are removed from the 3-hypergraph (Line 05). Then, our algorithm HYPERCLIQUE finds new IND candidates, in the manner described above (Line 06), with invalid INDs broken down into 4-ary subsets, and so forth for increasing k. The process is repeated until no new cliques are found. At each phase, some small INDs might be missed and are discovered in line 13 (see [5]). In all of our experiments using real data sets, the algorithm terminated for $k \leq 6$.

```
01 : Set $V \leftarrow$ genValidUnaryINDs$(R, S)$
02 : Set $E_2 \leftarrow$ genValidBinaryINDs$(R, S, V)$
     //initialize result set with unconnected nodes
     //(i.e., cliques of size 1)
03 : Set $res \leftarrow \{v \in V | \text{degree}(v) = 0\}$
04 : for $m \leftarrow 2 \ldots k_S - 1$
05 :     Graph $G_m \leftarrow (V, \text{validINDs}(E_m))$
06 :     Set $I \leftarrow$ genCliquesAndCheckAsINDs$(G_m)$
07 :     Set $C_{tmp} \leftarrow \emptyset$
         //collect invalid cliques into $C_{tmp}$
08 :     forall $(c \in I)$
09 :         if $(c$ is valid $\wedge |c| \geq m)$ $res \leftarrow res \cup c$
10 :         if $(c$ is invalid $\wedge |c| \geq (m + 1))$
                     $C_{tmp} \leftarrow C_{tmp} \cup c$
         //generate edges for the next hypergraph $G_{m+1}$
11 :     $E_{m+1} \leftarrow$ genKAryINDsFromCliques
             $(m + 1, C_{tmp})$
12 :     if $(E_{m+1} = \emptyset)$ return $res$
13 :     $res \leftarrow res \cup$ genSubINDs$(m, E_{m+1}, res)$
14 : return $res$
```

Figure 1. Algorithm FIND$_2$.

A complete explanation of the algorithm's functions, including a correctness and complexity discussion, can be found in [5].

5. Discussion

We implemented algorithm FIND$_2$ in Java over Oracle 8i relational databases (using JDBC). We ran a large suite of experiments, comparing our algorithm to well-known levelwise strategies ("Apriori" class of algorithms), as well as testing the algorithm's performance on multiple data sets obtained from the UC Irvine KDD Archive.

We found that algorithm FIND$_2$ finds large INDs (50-100 attributes in relations of 5,000-100,000 tuples) in reasonable time (minutes to a few hours). As one example, the discovery of a 30-ary IND in a training set (two relations of 41 attributes each, with 4500 and 5000 tuples, respectively) took about 350 seconds. Note that the implementation used standard SQL queries to determine IND validity and significant speedups are possible with more careful implementation. Also, the algorithm scales linearly in the size of base relations, even in the simple SQL-based implementation used. The full set of experiments can be found in [5].

6. Conclusions

In this paper, we have proposed an algorithm called FIND$_2$ for the problem of discovering inclusion dependencies over high-dimensional databases. With our solution, it is possible to automatically compare two databases with known schema, but unknown interrelationships, and identify inclusion dependencies between their attributes. As our algorithm discovers database interrelationships, it is useful for a variety of purposes, such as the identification of database duplicates or the comparison of large multi-dimensional databases. The discovery of inclusion dependencies is a hard problem, with inherent NP-complexity [4]. By mapping the problem to a set of graph problems, we achieve a significant improvement in performance over the naïve algorithm. Due to space limitations, we can only give a very brief overview over the work. A more detailed treatment can be found in [5].

References

[1] R. Agrawal and S. Ramakrishnan. Fast algorithms for mining association rules. In *Proc. Intl. Conf. on Very Large Databases (VLDB)*, pages 487–499, 1994.

[2] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, September 1973.

[3] F. de Marchi, S. Lopes, and J.-M. Petit. Efficient algorithms for mining inclusion dependencies. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, pages 464–476, 2002.

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

[5] A. Koeller and E. A. Rundensteiner. Discovery of high-dimensional inclusion dependencies. Technical Report WPI-CS-TR-02-15, Worcester Polytechnic Institute, Dept. of Computer Science, 2002.

[6] A. J. Lee, A. Koeller, A. Nica, and E. A. Rundensteiner. Data Warehouse Evolution: Trade-offs between Quality and Cost of Query Rewritings. In *Proceedings of IEEE International Conference on Data Engineering*, Special Poster Session, page 255, Sydney, Australia, March 1999.

[7] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 12(3):372–390, May/June 2000.