Heuristic Strategies for the Discovery of Inclusion Dependencies and Other Patterns*

Andreas Koeller[1]** and Elke A. Rundensteiner[2]

[1] Oracle Corporation, NEDC
Nashua, NH 03062, USA
koeller@acm.org
[2] Department of Computer Science, Worcester Polytechnic Institute,
100 Institute Road, Worcester MA 01609, USA
rundenst@cs.wpi.edu

Abstract. Inclusion dependencies (INDs) between databases are assertions of subset-relationships between sets of attributes (dimensions) in two relations. Such dependencies are useful for a number of purposes related to information integration, such as database similarity discovery and foreign key discovery.
An exhaustive approach at discovering INDs between two relations suffers from the dimensionality curse, since the number of potential mappings between the attributes of two relations is exponential in the number of attributes. For this reason, levelwise (Apriori-like) approaches at discovery do not scale beyond relations with 8 to 10 attributes. Approaches modeling the similarity space as graphs or hypergraphs are promising, but also do not scale very well.
This paper discusses approaches to scale discovery algorithms for INDs and some other similarity patterns in databases. The major obstacle to scalability is the exponentially growing size of the data structure representing potential INDs. Therefore, the focus of our solution is on heuristic techniques that reduce the number of IND candidates considered by the algorithm. Despite the use of heuristics, the accuracy of the results is good for real-world data.
Experiments are presented assessing the quality of the discovery results versus the runtime savings. We conclude that the heuristic approach is useful and improves scalability significantly. It is particularly applicable for relations that have attributes with few distinct values.

1 Introduction

In database research, and in particular in database design, modeling, and optimization, much emphasis has been placed on dependencies in databases. A vast field of research deals with functional dependencies (FDs), and many other dependencies between attributes of the same relation have been studied.

However, one type of dependency, called Inclusion Dependency (INDs) [1], is defined across *two* relations. The problem of IND *discovery*, which is addressed in this

* This work was supported in part by NSF grant #IIS9988776.
** This work was performed while the author was affiliated with Montclair State University, Montclair, NJ, USA

S. Spaccapietra et al. (Eds.): Journal on Data Semantics V, LNCS 3870, pp. 185–210, 2006.

paper, involves finding the minimal set of maximal inclusion dependencies between two relations from the data in the relations, rather than from implied or explicit knowledge about schemas (e.g., based on attribute names, features, or ontologies). IND discovery is also different from IND *inference* [1,2], which is the problem of discovering new INDs from known INDs, by using inference mechanisms such as transitivity, projection and permutation. IND discovery proceeds by querying or otherwise examining data across two relations without prior knowledge about those relations, in order to find inclusion patterns between them.

Solving IND discovery problems is interesting for a number of applications. INDs describe subset-relationships between projections (sets of attributes) of two relations, and can be thought of as related to the "specialization" relationship of object-oriented systems. For example, *foreign key constraints* are nothing but true (valid) INDs between a foreign key in one table and the associated key in another. Foreign key and functional dependency discovery [3] can be used to reorganize legacy database systems. In query rewriting, algorithms that answer queries over information spaces with partially redundant tables benefit from knowledge of INDs [4]. Examples can be found in the literature, e.g., query folding [5,6,7]. In the context of schema integration and matching [8], knowledge of redundancies across sources is essential. INDs represent such redundancies.

The problem of IND discovery is NP-hard [2], and enumeration algorithms are prohibitively slow, even for small real-world problems [9,10]. Since the problem is related to the discovery of functional dependencies [3] and association rule mining [11], proposals exist to adapt successful algorithms from those domains to the IND discovery problem [10]. In particular, those algorithms use a *levelwise strategy* [12], discovering single-attribute INDs first, then two-attribute (binary) INDs, then higher-order INDs. However, this approach does not scale beyond very modestly sized problems, as demonstrated in [9] and [10].

In previous work [13], the authors have proposed a scalable algorithm called FIND_2 that *discovers* INDs between unknown relations. Another similar algorithm, called *Zigzag*, has been independently proposed by de Marchi *et al.* [14]. The FIND_2 algorithm and the *Zigzag* algorithm approach the IND discovery problem from similar directions. They both observe that the solution to an IND discovery problem can be mapped to a *hypergraph*. Thus they can map the problem of IND discovery to a problem of discovering a hypergraph from limited knowledge of the hypergraph's nodes and edges. The algorithms employed in both approaches (hyperclique finding in FIND_2 and minimal traversal in *Zigzag*) are polynomial in the number of edges, and therefore exponential in the number of nodes in the hypergraph (since the number of edges in a general hypergraph of k nodes is bounded by 2^k). In the problem mapping applied in those algorithms, discovery problems over relations with 50 attributes (a common size) can easily lead to hypergraphs with hundreds of nodes, which for an algorithm running in exponential time in the number of nodes poses a serious problem [9,14]. Depending on the data in the source relations, even relations with 10–20 attributes can lead to unacceptably high runtimes or memory problems.

This paper deals with heuristic strategies to scale hypergraph-based IND-discovery algorithms beyond the sizes manageable in the basic hypergraph approach. The heuris-

tics reduce the size of hypergraph data structures involved in the discovery process by exploiting easily computable database statistics. While the non-heuristic FIND_2 and *Zigzag* algorithms find the exact problem solution, some of the strategies proposed here reduce the completeness of the solution. That is, the heuristics will sometimes prevent the finding of *all* INDs, but all INDs that *are* discovered will be correct, and often at least the *largest* IND in a given problem will be found.

It should be noted here that our work is orthogonal to manual or semi-automatic discovery of database relationships, as suggested by many research works [15,16] and implemented in many industrial software solutions. Our algorithms do not make use of domain knowledge such as ontologies, expert-supplied attribute relationship information, or use other schema-driven techniques. They exclusively use the *data* in the information sources to suggest relationships between databases.

The contributions of this paper are as follows: We identify and define "spurious" inclusion dependencies (INDs) as a major reason for performance problems in IND discovery. Then, we give a model of detecting such INDs. We also show how to derive heuristics based on this model, give additional suggestions as to the improvement of IND discovery, and present an experimental study of the advantages of our heuristic algorithm.

A preliminary version of this work was presented at the ODBASE 2004 conference [17]. In addition to the work published there, this paper contains: a discussion of the quality of results in heuristic IND discovery, a discussion of the use of the algorithm presented for discovery of patterns *other* than INDs, as well as four new experiments comparing our algorithm with an alternative technique, studying the effects of heuristics on quality, evaluating the DV heuristic specifically, and assessing the usefulness of the algorithm for non-IND pattern discovery.

The remainder of this paper is organized as follows: Section 2 reviews INDs and a hypergraph-based discovery algorithms for them. Section 3 introduces spurious INDs and motivates the concept. Section 4 introduces heuristics based on that notion and their application to IND discovery. Section 5 discusses how to assess the quality of the heuristic algorithm's results. Section 6 suggests the application of this algorithm to detect *approximate* relationships between tables, rather than INDs, which are based on exact subsets. Section 7 discusses experimental data to support our theoretical results and assess the algorithms. Sections 8 and Section 9 present related work and conclusions, respectively.

2 Background

2.1 Problem Definition

Our goal is to solve the problem of deducing all inclusion dependencies between two given relations solely from the data in the relations. Inclusion dependencies are defined as below.

Definition 1 (IND). *Let $R[a_1, a_2, \ldots, a_n]$ and $S[b_1, b_2, \ldots, b_m]$ 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 and denoted by $|\sigma|$. An IND $\sigma = (R[a_1, \ldots, a_k] \subseteq S[b_1, \ldots, b_k])$ is **valid** between two relations R and S if the sets of tuples in R and S satisfy the assertion given by σ.*

Due to its unclear semantics, we do not consider duplication of attributes on either side of the IND (i.e., we require sequences X and Y to be composed of distinct attributes). However, allowing duplicate attributes is possible and would not significantly increase the runtime of the algorithms presented here.

Casanova et al. [1] give a complete set of inference rules for INDs, observing that INDs are *reflexive*, *transitive* and invariant under *projection and permutation*. Permutation here refers to the reordering of attributes on *both* sides of the IND. For example, $R[AB] \subseteq S[KL] \equiv R[BA] \subseteq S[LK] \not\equiv R[BA] \subseteq S[KL]$. Transitivity is defined as usual, $R[X] \subseteq S[Y] \wedge S[Y] \subseteq T[Z] \Rightarrow R[X] \subseteq T[Z]$.

Projection invariance of INDs is the key to discovery algorithms. By projection, a valid k-ary IND with $k > 1$ **implies** sets of m-ary valid INDs, with $1 \leq m \leq k$. Specifically, for a given valid IND $\sigma = R[X] \subseteq S[Y]$, the IND $\sigma' = R[X'] \subseteq S[Y']$ will be valid for any subsequence $X' \subseteq X$ and its corresponding subsequence $Y' \subseteq Y$. 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_m^k. A very important observation is that the validity of all implied m-ary INDs of a given IND σ is a *necessary but not 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 $R[A_1, A_2, A_3] \subseteq S[B_1, B_2, B_3]$, as can easily be seen through an example (Fig. 1).

R

A_1	A_2	A_3
1	4	7
2	5	8
3	6	9

S

B_1	B_2	B_3
1	4	7
2	5	8
3	6	-1
-1	6	9
3	-1	9

$R[A_1, A_2] \subseteq S[B_1, B_2]$ is valid.
$R[A_2, A_3] \subseteq S[B_2, B_3]$ is valid.
$R[A_1, A_3] \subseteq S[B_1, B_3]$ is valid.
$R[A_1, A_2, A_3] \subseteq S[B_1, B_2, B_3]$ is not valid.

Fig. 1. Validity of all implied INDs is not a sufficient validity test.

Due to the projection invariance, a set Σ of INDs between two relations can be described by a **cover of INDs**, denoted by $\mathcal{G}(\Sigma)$. Intuitively, this is a minimal set of INDs from which all INDs in Σ can be derived by projection, permutation, and transitivity. Naturally, $\mathcal{G}(\Sigma) \subseteq \Sigma$. With these observations, the IND discovery problem reduces to the problem of finding a *cover of INDs* for a given pair of relations.

2.2 IND-discovery Algorithms

Exhaustive Discovery Since $|\Sigma_m^k| = \binom{k}{m}$, the number of valid INDs implied by a single k-ary IND σ is exponential in k: $\sum_{m=1}^{k-1} \binom{k}{m} = 2^k - 2$. Furthermore, INDs are *not*

invariant under permutation of the attributes of just *one* side, but only if the attribute lists on both sides are permutated synchronously. That means for example that, when *discovering* INDs between two relations with k attributes, one has to test $k!$ potential INDs just for the hypothesis that the one relation is completely included in the other. Consequently, exhaustive enumeration algorithms are exponential and not feasible for IND discovery.

Since Apriori-like algorithm are the standard solution for many discovery problems (e.g., for association rules), the question arises whether such an algorithm might be appropriate for our problem. In fact, a levelwise algorithm [12] akin to the Apriori algorithms in association rule mining [11] has been proposed as a solution to this problem [10]. It discovers unary INDs first and then forms binary IND candidates from the valid unary INDs. Those INDs then have to be validated against the database. From the validated binary INDs, 3-ary INDs are formed, then tested, and so on. In the presence of a single sufficiently large valid IND σ, such an algorithm will have to discover $2^{|\sigma|} - 2$ implied INDs before even *considering* σ. This is clearly not a feasible approach. Experiments conducted by the authors (see Sec. 7) and de Marchi [10] both suggest that levelwise algorithms do not scale beyond a maximal IND size of 8–10.

Hypergraph-Based Discovery In general, the worst-case complexity of the problem is determined by the number of possible distinct INDs between two relations. However, in real-world problems, one expects to find a very low number of large distinct INDs (in fact, often just one), and possibly several small INDs (see also Sec. 5). Therefore, it is meaningful to find a *minimal cover* of valid INDs without even enumerating all valid INDs, reducing the complexity significantly.

For this purpose, the problem is mapped into a graph problem. We use a family of **k-uniform hypergraphs** which are graphs in which each edge is incident to exactly k nodes. 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-uniform 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.*

In analogy to above, a clique is a hyperclique in a 2-hypergraph.

To map our problem, we now map the set of valid INDs to a family 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 use the fact that, for $m = 2 \ldots k - 1$, any set Σ_m^k of INDs implied by a valid σ_k maps to a hyperclique in the corresponding hypergraph G_m. In other words, after an initial step of discovering low-arity INDs ($k = 1 \ldots 2$), we can form candidates for valid high-arity INDs by considering only those potential INDs that correspond to cliques in k-uniform hypergraphs for small k.

Algorithm FIND$_2$ Algorithm FIND$_2$ (Fig. 2) applies hyperclique-finding techniques to find inclusion dependencies (INDs). It was published as part of a dissertation [9] and also appears in [13]. Full details and derivations can be found in [18]. FIND$_2$ takes as input two relations R and S, with k_R and k_S attributes, respectively and returns a cover $\mathcal{G}(\Sigma)$ of INDs between R and S. The algorithm proceeds in stages enumerated by a parameter $k = 2, 3, \ldots$. It begins by exhaustively validating unary and binary INDs, forming a (2-uniform) hypergraph using unary INDs as nodes and binary INDs as edges (Step 1, $k = 2$). A clique-finding algorithm then determines all higher-arity INDs candidates (Step 2, candidates c_1 and c_2 in the figure). Since the clique property is necessary but not sufficient for the validity of a higher-arity IND (Sec. 2.1), each IND candidate thus discovered must also be checked for validity. Each IND that tests invalid (but corresponds to a clique in the 2-hypergraph) is broken down into its implied 3-ary INDs. They then form the edges of a 3-hypergraph (Step 3, $k = 3$). Edges corresponding to invalid INDs are removed from the 3-hypergraph.

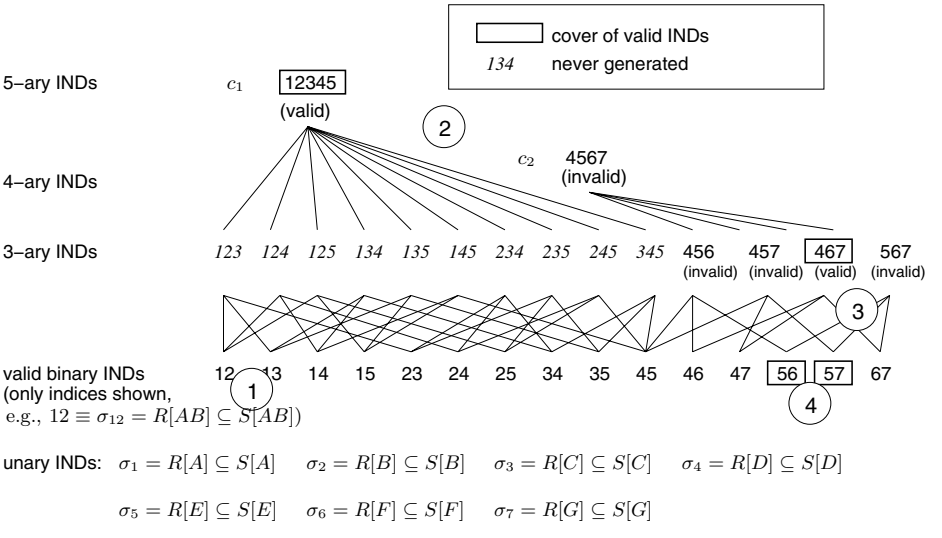

Fig. 2. Overview of the complete algorithm FIND$_2$.

Then, hypercliques are found in the 3-uniform hypergraph formed with unary INDs as nodes and 3-ary INDs as edges. Hypercliques found are new IND candidates. Invalidated IND candidates found in this step are broken down into 4-ary subsets ($k = 4$). The process is repeated for increasing k until no new cliques are found. At each phase, some small elements of the cover $\mathcal{G}(\Sigma)$ might be missed and are discovered by a cleanup process (Step 4, see also [18]). In all of our experiments using real data sets, the algorithm terminated for $k \leq 6$ (in Fig. 2, the algorithm terminates for $k = 3$).

Since the publication of FIND$_2$, de Marchi *et al.* have independently proposed a similar algorithm called *Zigzag* [14], which uses the same basic model as ours, but em-

ploys *minimal hypergraph traversals* [3,19] instead of clique-finding in order to generate large IND candidates. Furthermore, they introduce an optimization to the treatment of invalidated large IND candidates (e.g., c_2 in Fig. 2), in that they also attempt to validate such a failed IND candidate by projecting out single attributes from it, rather than restarting the discovery process for $k+1$. They make a decision as to which strategy to apply based on the number of tuples in relations R and S that violate the hypothesized IND.

3 The Semantics of Inclusion Dependencies

Attribute sets that stand in an IND to each other are not necessarily matches for the purpose of data integration. INDs can occur between attributes "by accident", especially if attributes have few distinct values and have similar or equal domains. Therefore, an algorithm enumerating all inclusion dependencies across two database tables is likely to produce some results that are not interesting for the purpose of data integration or schema matching.

3.1 Why Improve IND Discovery?

Algorithms FIND_2 and *Zigzag* as described so far find the complete and correct solution to the IND-finding problem for two given relations. In principle, both algorithms first discover unary and binary INDs by enumeration and testing (called *pessimistic strategy* in [14]), and then form an *optimistic hypothesis* about the IND space by assuming that all high-arity INDs that *could* be valid based on the validated unary and binary INDs *are* in fact valid. That assumption makes both algorithms extremely sensitive to an overestimation of valid unary and binary INDs. A high number of such small INDs would cause many invalid larger IND candidates to be generated and tested against the database.

Also, several of the algorithms involved, in particular the hypergraph-based pattern discovery (hyperclique-finding in FIND_2, min-hypergraph traversal in *Zigzag*), have high complexity [19,18], and are fast only for sparse hypergraphs.

Therefore, the overall discovery strategy is sensitive to such overestimations and it is important to prune unnecessary INDs from the search space.

3.2 Spurious INDs

We will now motivate the concept of "overestimating" INDs. For this purpose, we define a notion of "accidental" or "spurious" INDs which are valid in the database but do not contribute significantly to finding a solution to our problem.

Definition 3 (Spurious IND). *An inclusion dependency $\sigma = R[A] \subseteq S[B]$ is called spurious iff (1) it is valid in the database and (2) does not reflect a* semantic relationship *between attribute sets A and B (i.e., A and B do not represent the same real-world dimensions).*

The exact meaning of "semantic relationship" depends somewhat on the context in which the IND discovery is used. For example, in schema matching, semantically related attributes would be mapped into the same attribute in the integrated schema. In query rewriting, a semantic relationship between two attributes would represent a redundancy between those attributes.

Often, spurious INDs occur when the domains of attributes are small (i.e., if attributes have many duplicate values), as the following example illustrates.

Example 1. Consider Fig. 3 for an example. The domains of three columns in table *Member* and two columns in table *Former* are "year", which is a domain with few values. The figure shows the cover $\mathcal{G}(\Sigma)$ of INDs for this problem.

Member

Name	Birthyear	MemberSince	MemberUntil
Jones	1940	1969	1989
Miller	1945	1960	1988
Myers	1960	1980	1988
Shultz	1969	1988	1989
Becker	1961	1989	

Former

Member	YOB	LeftIn
Myers	1960	1988
Shultz	1969	1989

Former[Member,YOB,LeftIn] ⊆ Member[Name,Birthyear,MemberUntil],
Former[**YOB,LeftIn**] ⊆ **Member**[**MemberSince,MemberUntil**]
Former[**LeftIn**] ⊆ **Member**[**MemberSince**]

Fig. 3. Accidental INDs introduced by small domains

Two low-arity INDs are part of the cover of INDs between **Former** and **Member**, shown in bold font in Fig. 3. However, in some sense, these INDs are intuitively "wrong". Note that they are not implied by any INDs with arity larger than 2. Therefore, the discovery algorithm will not need these INDs for finding INDs with arity > 2 and pruning them from the search space would speed up the algorithm while not significantly reducing the quality of its result.

3.3 Detecting Whether an IND Is Spurious

Algorithms FIND_2 and *Zigzag* both treat testing a single IND as an elementary operation with a binary result. A test for binary IND validity can simply be performed by formulating a database query. In SQL, one could employ the EXCEPT (set-difference) operator, since $R[A] \subseteq S[B] \iff \left(\left|R[A] \backslash S[B]\right| = 0\right)$. This however does not generate any information about the "spuriousness" of the IND.

In order to assess the probability for spurious INDs to occur we now look at a statistical model. Consider a sample N of size n obtained by sampling with replacement from a set K of k objects. Given a certain set $R \subseteq K$ of size $r \leq n$, consider the probability that all values in R are included in the sample N and denote it by $P(n, r, k)$. It can be computed by the following formula.

Theorem 1. *Consider a set $R = \{e_1, \ldots, e_r\}$ of r distinct elements from a universe K of k distinct elements. The probability that a random sample (obtained by sampling with replacement) of size n from K contains set R is*

$$P(n,r,k) = 1 - \frac{\sum_{i=1}^{r}(-1)^{i+1} \cdot \binom{r}{i} \cdot (k-i)^n}{k^n} = 1 - \sum_{i=1}^{r}(-1)^{i+1} \cdot \binom{r}{i} \cdot \left(1 - \frac{i}{k}\right)^n \quad (1)$$

Proof. There are k^n different samples of size n from k distinct elements (sampling with replacement). We compute how many of those do *not* contain R. A sample that does not contain R is missing at least one element from R. Let us denote by $A_{\overline{e}}$ the set of all samples that are missing element e. Then, the number of samples that do not contain at least one element from R is $r_0 = |A_{\overline{e_1}} \cup A_{\overline{e_2}} \cup \cdots \cup A_{\overline{e_r}}|$.
We now need to determine the size of the union of all those sets. The size of each $A_{\overline{e}}$ is $(k-1)^n$. In analogy, the size of $A_{\overline{e_1}} \cap A_{\overline{e_2}}$ (the set of all samples missing *two* given elements) is $(k-2)^n$, and so on. Since we can compute the sizes of their intersections, we can use the *inclusion-exclusion rule* of combinatorics[3], and get $r_0 = \sum_{i=1}^{r}(-1)^{i+1} \cdot \binom{r}{i} \cdot (k-i)^n$. We then get the probability $P' = \frac{r_0}{k^n}$ that a sample does *not* contain R. Therefore $P(n,r,k) = 1 - P'$, □

In order to now determine the probability of "spurious INDs', assume two relations R and S and the problem of assessing whether a valid IND $\sigma = R[A] \subseteq S[B]$ is spurious. Let A have r distinct values. Furthermore, set $n = |S|$, i.e., n is the number of (non-distinct) values in attribute B. One can argue that since the values in A are a subset of the values in B, the values in both attributes are from a common domain K with k distinct elements.

We are interested in the "chance" that attribute A just "happens" to be included in attribute B. This "chance" can be assessed by the probability that a sample (with replacement) of size n from K contains A, which is $P(n,r,k)$.

Now note that $\lim_{n\to\infty}(1 - \frac{i}{n})^n = e^{-i}$. Define $k = \frac{n}{c}$ and insert it into the rightmost term in Equation 1. Since $\lim_{n\to\infty}(1 - \frac{ci}{n})^n = e^{-ic}$, that means that for large n and k, the value of $P(n,r,k)$ depends approximately only on r and $c = \frac{n}{k}$.

In Table 1 we have listed the maximum value of c for which $P(n,r,k)$ remains lower than 5%, for different r. That is, for a given number of distinct values in an attribute A, we can estimate how likely it is that A is contained in an attribute B by chance, given the size of B and the size of the common domain of A and B. This is a measure of how likely $R[A] \subseteq S[B]$ is to be spurious.

Of course, the size of domain K is unknown. However, since we have assumed initially that $R[A] \subseteq S[B]$, we could assume that K is given by the distinct values in B. In this case, $n > k$ and thus $c \geq 1$. In this case, we get a $P < 0.05$ only if $r > 7$.

We conclude that inclusion dependencies where the included attribute has less than 6 or 7 distinct values have a high chance of being valid by statistical coincidence, rather than by semantic relationships between the attributes. We exploit this result to restrict the search space of our algorithm.

[3] This is a generalization of $|A \cup B| = |A| + |B| - |A \cap B|$. See also [9].

Table 1. Minimum number of distinct values to avoid spurious INDs.

r	$P(n,r,k) < 0.05$ for $c = n/k$ less than	r	$P(n,r,k) < 0.05$ for $c = n/k$ less than
2	0.25	7	1.06
3	0.46	10	1.35
4	0.64	20	1.97
5	0.80	50	2.85
6	0.93	100	3.53

4 Heuristics for IND-validity Testing

From the observations above, we have derived two heuristics which are useful in reducing the number of IND candidates considered in a discovery problem.

4.1 The Number-of-Distinct-Values (DV) Heuristic

Based on our definition of spuriousness, the DV heuristic states that an IND $R[A] \subseteq S[B]$ should *not* be used as a node or edge in a hypergraph in algorithm FIND_2 (or similar algorithms such as Zigzag) if the attribute (or attribute set) A has few distinct values (tuples). That is, this heuristic simply discards all inclusion dependencies in which the included attribute has less than n distinct values.

This method is supported by our theoretical results in Sec. 3.3, which state that $r = \delta(R[A])$ (the number of distinct values in attribute A) must be relatively large for the IND $R[A] \subseteq S[B]$ to not be considered spurious. From the theory, we would set a value of $n = 7$, a choice that is confirmed by our experiments.

The DV heuristic can only be used to test for *valid* INDs, i.e., an IND that is already considered invalid will not be affected. It may produce false negatives, i.e., declare INDs as spurious that are in fact not. Therefore, this heuristic has to be used carefully, as explained in Sec. 4.3.

4.2 The Attribute-Value-Distribution (AVD) Heuristic

The Attribute Value Distribution (AVD) heuristic has strong predictive power for many data sets. It is based on the hypothesis that two attributes A and B that form a non-spurious IND (i.e., are semantically related) have the same frequency distribution of values.

Obviously, this is strictly only true if A and B are both randomly taken from a common set of values. However, for the purpose of this paper, we are assuming that semantically related attributes *are* both taken from such a common set. Therefore, the additional assumption that they are *random* samples seems reasonable at least for some cases. The heuristic then states the following:

If the values of attributes A and B in a valid IND $\sigma = R[A] \subseteq S[B]$ do not *show the same value distribution, the attributes are not semantically related.*

That is, if the value distribution is found to be different, the σ can be considered spurious. If it is not different, no new information is gained about σ. This heuristic can produce false negatives when attributes that are actually semantically related are rejected due to the fact that they actually do not have similar frequency distributions. The statistical hypothesis testing itself, which is probabilistic in nature, may also lead to false negatives.

Performing Statistical Hypothesis Testing for AVD For the hypothesis test, we use the widely applicable χ^2-Test [20], in particular a χ^2-Test for independence. This test is designed to assess the independence of two categorical variables x and y. The χ^2-Test then tests under the null hypothesis that the two variables x and y are independent, i.e., that the value of variable x does not influence the value of variable y.

For our purpose we perform the following mapping: Given an IND $R[A] \subseteq S[B]$, we set $x = \{\mathrm{A}, \mathrm{B}\}$ (i.e., the *names* A and B) and $y = \delta(R[A]) \cup \delta(R[B])$, where $\delta(R[A])$ denotes the set of distinct values in attribute A of relation R. The *contingency table* used for the χ^2-Test is then filled with the *counts* of each distinct data value in each of the two attributes.

We are therefore testing for the null hypothesis: "the distribution of values in an attribute does not depend on the choice of attribute (out of $\{A, B\}$) from which the values are taken". If this hypothesis is rejected (i.e., if the value distribution *is* dependent on the choice of attribute), we conclude that the value distributions in the two attributes are different, and consequently an IND between them is spurious.

The attribute value distribution in a single attribute can be obtained easily through an SQL-query and can be pre-computed for all attributes. For larger INDs, attribute values can be concatenated to compute AVDs.

4.3 Incorporating Heuristics into the IND-checking Algorithm

The heuristic-based IND-checking function, called $\mathsf{CHECK}_\mathsf{H}$, is shown in Fig. 4. While the basic FIND_2 algorithm uses a simple database query to detect the validity of an IND, the heuristic algorithm, called FIND_H, uses this heuristic check function. $\mathsf{CHECK}_\mathsf{H}$ employs the DV and AVD heuristics introduced above, and also performs a simple check for compatible domains. Note that the AVD heuristic is only used when (1) the IND is valid in the database and (2) the DV heuristic rejects the IND. The intuition is that the AVD heuristic is a stronger test of spuriousness than the DV heuristic and can detect a semantic relationship (and thus "pass" the IND) where the DV heuristic failed. The CHECK-function performs a validity check of a single IND against the source database(s) through a database query and returns a Boolean value.

The computational complexity of IND-checking against the database is quite high, as a check involves computing a set difference, and is consequently of $O(n \log n)$ complexity in the number of tuples in the relations. De Marchi [10] proposes the use of an inverted index of data values in order to facilitate the computation of unary INDs only. This approach is not applicable for binary or higher-order INDs. Further improvements in the *testing* of INDs (rather than the generation of IND *candidates*) could be beneficial.

```
function CHECK_H(Relation R, AttList A of R, Relation S, AttList B of S)
    if (domains of R[A] and S[B] incompatible)
        return invalid
    else if (CHECK(R, A, S, B) = invalid)   //a check against the database
            return invalid
        else if (DV heuristic does not reject IND)
                return valid
            else if (AVD heuristic rejects IND)
                    return invalid  //false negative possible
                else return valid
```

Fig. 4. The heuristic IND-checking function $\mathsf{CHECK}_\mathsf{H}$ used by algorithm FIND_H

4.4 Detecting INDs in the Presence of False Negatives

Consider a complete graph (i.e., a graph with all possible edges) $G = (V, E)$. Then, the set of nodes V forms a clique in G. Now remove a single edge from E. Clearly, the clique property does no longer hold, but rather G will now contain at least two distinct maximal cliques. Those cliques are likely to have a substantial overlap (i.e., common set of nodes).

If any of our heuristics produces false negatives, some edges (or even nodes, i.e., unary INDs) of any graph or hypergraph considered by FIND_2 may be missing. The clique finding algorithms used by FIND_2 will then no longer find cliques that correspond to the maximal INDs in the problem given, but rather find only smaller subsets of those cliques. Simulations show that the removal of as few as 5 random edges from a clique of 40 or 50 nodes will generally produce a graph with around 20 distinct maximal cliques. However, those sub-cliques will often show substantial overlaps. Therefore, we use the following strategy: *When heuristics are used in* FIND_2 *that may produce false negatives (i.e., reporting non-spurious INDs as invalid), and* FIND_2 *reports several large, overlapping INDs, then we merge those INDs by computing the union of their nodes.*

Naturally, merging *all* INDs found by algorithm FIND_2 will in general not lead to a valid INDs, unless the (true) cover of INDs actually contains only one IND. Therefore, we merge INDs of decreasing size, starting from the largest, until adding another IND to the result will no longer produce a valid IND.

Our experiments show that the IND-merging heuristic is powerful enough to find large or maximal valid INDs even in cases when many underlying edges are pruned in earlier stages of heuristic discovery (Sec. 7).

5 Quality of Results in Heuristic IND Discovery

Discovery of INDs typically has the goal of discovering relationships between databases. As such, finding the *largest* set of related attributes between two given tables is an important subgoal. If that largest IND is large (has many attributes) compared to

any other INDs in the solution, it represents more information about the relatedness of the databases than the smaller INDs. The reason for this is that a 10-ary (or any high-arity) IND is very unlikely to hold by accident. If a high-dimensional IND is found between two tables, it most likely represents an actual semantic relationships between the attributes in the IND, rather than a random pattern that is true due to statistical coincidence.

See Table 2 for a typical result of an IND discovery. In the example, two tables have a set of 10 attributes each, which stand in an Inclusion Dependency relationship to each other. This is represented by the 10-ary IND discovered by the exact algorithm (left column in the table). The exact algorithm also found 7 more INDs, one unary one, 5 binary ones, and one 3-ary one, which are not implied by the 10-ary IND. Those 7 INDs are most likely spurious by our definition (Def. 3) since they do not seem to represent a semantic relationship between the tables. See also Fig. 5.

Table 2. Sets of Maximal Distinct INDs discovered by Heuristic FIND_H and Exact FIND_2 Algorithms.

IND Size	Exact Algorithm (FIND_2)	Heuristic Algorithm (FIND_H)
1	1	1
2	5	1
3	1	0
4	0	0
5	0	1
6	0	0
7	0	3
9	0	0
10	1	0

Furthermore, a large IND implies many smaller INDs (Sec. 2.1). In the example in Table 2, the 10-ary IND σ_1^{10} implies $2^{10} - 2 = 1022$ smaller INDs, whereas all the remaining maximal INDs ($\sigma_1^1 \ldots \sigma_1^3$) together imply only $5 \cdot (2^2 - 2) + 1 \cdot (2^3 - 2) = 16$ more smaller INDs, most of which are already subsumed by σ_1^{10} or are duplicates of each other.

On the other hand, the heuristic FIND_H algorithm did not find all of the INDs between the two tables (Fig. 2). Instead of σ_1^{10}, it found three 7-ary INDs, which are fragments of the 10-ary true IND. Note that the union of the attributes of the three INDs $\sigma_1^7 \ldots \sigma_3^7$ is exactly σ_1^{10}.

This means that using IND-merging, the heuristically found solution is essentially as useful as the exact one. However, the 7-ary INDs imply only a total of $3 * (2^7 - 2) = 372$ INDs, less than 40% of the total number of INDs implied by σ_1^{10}. Most of those INDs are also duplicates of each other since those three INDs have a 6-attribute overlap. Therefore, counting the *total* number of INDs in the solution is not a good measure for the quality of the result.

$$\begin{aligned}
\sigma_1^1 &= R[C] \subseteq S[B] \\
\sigma_1^2 &= R[B, J] \subseteq S[C, J] \\
\sigma_2^2 &= R[F, J] \subseteq S[I, J] \\
\sigma_3^2 &= R[H, J] \subseteq S[I, J] \\
\sigma_4^2 &= R[H, J] \subseteq S[F, J] \\
\sigma_5^2 &= R[E, H] \subseteq S[E, F] \\
\sigma_1^3 &= R[E, F, J] \subseteq S[E, H, J] \\
\sigma_1^{10} &= R[A, B, C, D, E, F, G, H, I, J] \subseteq \\
&\quad S[A, B, C, D, E, F, G, H, I, J]
\end{aligned}$$

Fig. 5. INDs discovered by Exact FIND_2 Algorithm.

$$\begin{aligned}
\sigma_1^1 &= R[C] \subseteq S[B] \\
\sigma_1^2 &= R[B, J] \subseteq S[C, J] \\
\sigma_1^5 &= R[B, C, D, G, I] \subseteq S[B, C, D, G, I] \\
\sigma_1^7 &= R[A, B, C, D, E, F, I] \subseteq \\
&\quad S[A, B, C, D, E, F, I] \\
\sigma_2^7 &= R[A, B, C, D, E, H, I] \subseteq \\
&\quad S[A, B, C, D, E, H, I] \\
\sigma_3^7 &= R[A, B, C, D, E, I, J] \subseteq \\
&\quad S[A, B, C, D, E, I, J]
\end{aligned}$$

Fig. 6. INDs discovered by Heuristic FIND_H Algorithm.

The primary difference between the exact and the heuristic algorithms is their treatment of unary and binary INDs, since the heuristics are not applied for higher-arity INDs. Therefore, we can also compare the counts of those unary and binary INDs as a measure of result quality. In the example above, the exact algorithm FIND_2 discovered 16 valid (not necessarily maximal) unary INDs. On the other hand, the heuristic FIND_H algorithm regarded 4 of those unary INDs as spurious. Those 4 INDs were actually not implied by the large IND σ_1^{10}, which means the heuristic correctly disregarded them. The FIND_H algorithm then generated only 62 possible binary INDs to test against the database, as opposed to 105 in the exact algorithm, which represents a 40% savings in runtime for this phase. However, the distinct-value heuristic for the binary INDs rejected 8 of the valid 46 binary INDs, some of which *were* implied by σ_1^{10}. Thus, the cause of quality loss in this case was the distinct-value heuristic for binary INDs.

6 Heuristics for Discovering IND-like Database Similarities

Algorithms FIND_H and FIND_2 discover INDs, which are strict set-inclusion patterns. However, they can also be used to discovery patterns that are not technically INDs, but rather "IND-like" pattern. In particular, the discovery of *similarities (near inclusion)* between tables rather than strict inclusion is possible if the similarities are strong enough.

The current algorithm uses SQL set-difference queries (see also Sec. 3.3) to detect inclusion of a given projection of the two tables in question. A projection $\pi_{\overline{A}}(R)$ on a table R is included in a projection $\pi_{\overline{B}}(S)$ in table S if the result of the relational query $\Delta = \pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)$ is empty. However, if Δ is not empty, its size $|\Delta|$ (i.e,. the number of tuples in the difference relation) can be an indicator for the relatedness of the projections. In its simplest form, a small $|\Delta|$ indicates a good relationship, while a large $|\Delta|$ suggests no relationship. A somewhat stronger heuristic is to use the ratio

of $|\Delta|$ to the number of distinct tuples in $\pi_{\overline{A}}(R)$ and/or $\pi_{\overline{B}}(S)$. This "relatedness" score can be used to rank IND-like patterns, which is important when at some stage in the algorithm, too many patterns are discovered to consider all. This ranking is also useful in the final output of the algorithm, as it gives additional information on whether a pattern discovered is real or not.

Partial Overlap Heuristic Determining whether two projections $\pi_{\overline{A}}(R)$ and $\pi_{\overline{B}}(S)$ of relations R and S are related if they do not satisfy an IND (i.e. if $\pi_{\overline{A}}(R) \not\subseteq \pi_{\overline{B}}(S)$) can be done in the following way:

- If $|\pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)| < c_1$, the projections are considered related. c_1 can either be a integer constant ($c_1 \geq 1$) or can be a function of the number of attributes in $\overline{A}$ (i.e,. the arity of the IND-like pattern). The rationale is that a very small number of "violating" tuples in the set difference between the two projections could be caused by noise in the data rather than a non-relatedness. An empirically found useful value for c_1 is $c_1 = 3$.
- If $|\pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)| < c_2 \cdot |\pi_{\overline{A}}(R)|$ the projections are considered related as well, with $0 < c_2 \leq 1$. The rationale here is that if the number of distinct tuples in the set-difference is less than a fraction of the number of distinct tuples in the left ("smaller") relation of the IND-like pattern, a relationship between the projections is likely. We experimented successfully with a $c_2 = 0.49$, which represents the fact that the smallest useful domain in a relational database must have a size of two (two distinct values, one of which could be null). In this way, for example, an attribute with a two-valued domain would not be considered related to another attribute unless there is a true IND between the two attribute sets (i.e., $|\pi_{\overline{A}}(R) \backslash \pi_{\overline{B}}(S)| = 0$), while two attribute sets $\overline{A}$ and $\overline{B}$ with larger domains could be considered related even if $\pi_{\overline{A}}(R) \not\subseteq \pi_{\overline{B}}(S)$.

If either of those two conditions is satisfied, the projections will be considered related, and treated like valid INDs. That is, they are then passed on to the other heuristics, filtering out spurious INDs, and then used in the $\mathsf{FIND}_{\mathsf{H}}$ algorithm.

While this heuristic works well for many cases (see Experiment 7, Sec. 7), the underlying assumption is that related tables have some data in common. With this simple scheme, a discovery of "relatedness" is possible if there is some extensional overlap between the relations to be compared. If the relations have no tuples in common, the use of set-difference queries is not meaningful for the discovery of relationships.

7 Experiments and Evaluation

7.1 Experimental Setup

Experiments were performed on several Linux-PCs with a dedicated machine running a relational database server. We obtained data from the UC Irvine KDD Archive (`http://kdd.ics.udi.edu`), specifically subsets of the CUP98, CENSUS, INSURANCE, and INTERNET data sets, which (converted into relational tables) had between 40 and 90 attributes each.

In order to "discover" inclusion dependencies, we used different projections and selections of each dataset and compared those to each other. An interesting feature of some of the data sets is that they have very small domains, as many of their attributes are categorical data. Furthermore, they are encoded as small integers, such that many unrelated attributes match each other (i.e., form spurious unary INDs). While one could "join" those columns with their "dimension tables" and obtain distinct domains, we left the tables as they were as a challenge to our algorithms. The effect was a high number of spurious INDs, which we could use to assess the performance of our solution.

7.2 Experiment 1: Comparison with Alternative IND Discovery Techniques

The performance of the FIND_2 algorithm (without heuristics) was compared with the previously published Apriori-like IND discovery algorithm [10], which serves as the baseline algorithm for this problem (and can be faster for very small problems). The latter algorithm was implemented in the same environment (Java over relational DB) as the FIND_2 algorithm. The test case consisted of a set of selections of the CENSUS table, with 500 rows each. Each table had 41 attributes. The total number of INDs between those tables varied, and the size of the largest IND between any of the tables tested also varied, between 5 and 16 attributes.

As can be seen from Fig. 7, the runtime of the FIND_2 algorithm is substantially shorter than that of the Apriori-like algorithm. The runtime recorded represents CPU time only; the number of database queries is also lower for the FIND_2 algorithm than for the Apriori-algorithm. As expected, the latter algorithm shows exponential runtime behavior in the size of the largest IND in the solution (note that the y-Axis is logarithmic). The runtime of FIND_2 is not affected by the size of the largest IND.

On the other hand, the runtime of FIND_2 does depend on the *size of the solution*, i.e., the number of distinct maximal INDs in the result (Fig. 8 shows data and linear regression curve; the correlation coefficient is $r^2 = 0.95$). As explained in Section 5, the size of the true solution is often small, but can be increased greatly by spurious INDs. Even though many of those spurious INDs are eventually purged from the search space in later phases of the algorithm, they slow down IND discovery significantly, and can even lead to aborted discovery runs due to memory problems. Here, using the heuristics proposed in this paper can help to speed up IND discovery.

7.3 Experiment 2: Performance and Quality Effects of Heuristics

This experiment was conducted to assess the runtime of the algorithm and the quality of its output for a given data set, with and without the use of heuristics. For this experiment, we used a 5000-tuple random subset CENSUS1 of data set CENSUS and a further random subset of 4500 tuples (90%) of CENSUS1, called CENSUS2 (i.e., $\mathsf{CENSUS2} \subset \mathsf{CENSUS1} \subset \mathsf{CENSUS}$). This choice was made to emulate a certain randomness in real-world data. We compared the performance and quality of algorithms FIND_2 and FIND_H. We used different projections on those tables, which all originally have 41 attributes. Figure 9 shows the runtime of algorithms FIND_2 and FIND_H, for different size projections, illustrating the large performance benefits of the heuristic strategy.

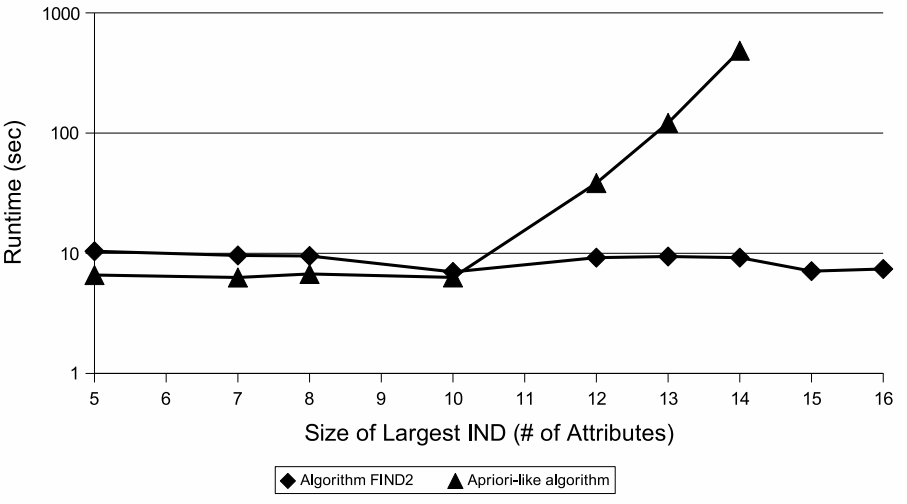

Fig. 7. Comparison of Algorithm FIND_2 with Apriori-like algorithm

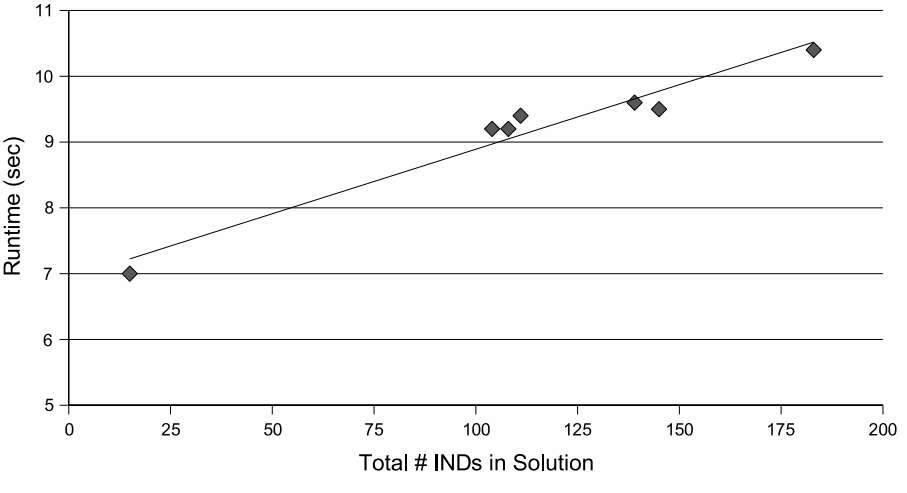

Fig. 8. Runtime Behavior of Algorithm FIND_2 under different size solutions.

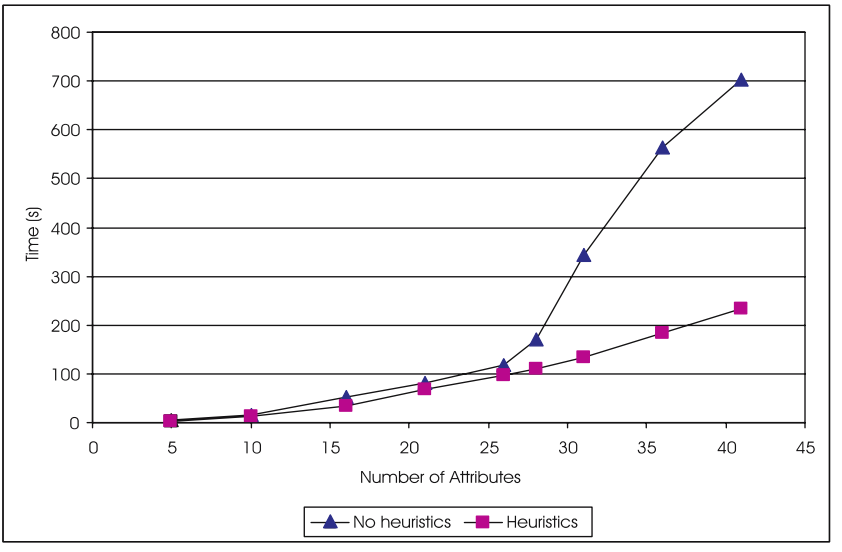

Fig. 9. Performance of algorithms FIND_2 and FIND_H, respectively, for discovering INDs between CENSUS2 and CENSUS1. The time for merging INDs is included in the runtime measurements.

There is a penalty in accuracy as a tradeoff for the lower runtime. The full cover of INDs is not found by the heuristic algorithm. Rather, FIND_H reports a maximum IND whose arity is about 70%-85% of the largest valid IND between the test data sets. However, through IND merging (Sec. 4.4), we still correctly find the largest IND in this data set. In other cases, the results of clique merging are not perfect as here, but still large INDs are found, as shown below.

7.4 Experiment 3: Assessing Result Quality for Heuristic FIND_H Algorithm

As explained in Sec. 5, the size of the largest IND discovered is a useful measure for the quality of the algorithm's performance. However, to assess the quality reduction of FIND_H, we conducted an experiment assessing the precision and recall of *unary* and *binary* INDs in the respective solutions.

For our test case of the CENSUS dataset, we recorded all unary and binary INDs that the heuristic and non-heuristic algorithms had considered and discovered, and compared with the true solutions.

See Table 3 for the results. The table contains the values for precision and recall for unary and binary INDs for both algorithms. Precision is computed in the usual manner as percentage of discovered INDs that are correct, while recall is computed as percentage of discovered INDs that are found by the algorithm.

In the test case, there was a single 41-ary IND to discover, such that the number of correct unary INDs was 41, and the number of correct binary INDs was $\binom{41}{2} = 820$. The

Table 3. Precision and Recall of Heuristic Algorithm FIND_H for Unary and Binary INDs

Algorithm	Unary INDs		Binary INDs	
	Precision	Recall	Precision	Recall
FIND_2	59%	100%	83%	100%
FIND_H	89%	100%	99%	97%

non-heuristic IND algorithm of course discovers all those INDs, but considers many more, spurious, INDs. In this experiment, the heuristics worked very well for unary INDs, achieving 100% recall and only considering very few INDs that turned out to be spurious. This is partially due to good performance of the AVD-heuristic which compares frequency distribution of values.

In other cases, the recall is not as good but still sufficient to discover large INDs efficiently. This experiment demonstrates the effect of the FIND_H algorithm: increase in precision of discovery of small INDs, at the expense of a reduction in recall.

7.5 Experiment 4: Effect of Low Numbers of Distinct Values in Data Set

In this experiment, we assess the quality of the heuristic algorithm in a data set with many spurious INDs. Table INSURANCE is such a data set, as it contains almost exclusively attributes with small integer domains (often less than ten distinct values) and consequently nearly 50% of its unary INDs are valid. For the full data set of 86 attributes, 4000 unary INDs are valid in the database, which would lead to a prohibitively large hypergraph with 4000 nodes.

In fact, the non-heuristic FIND_2 algorithm fails for this data set for all cases with more than 10 attributes, so no performance results for the non-heuristic algorithm can be reported for comparison.

Table 4 shows the quality achieved by the heuristic algorithm $\mathsf{CHECK}_\mathsf{H}$ for this case, for different size projections of table INSURANCE. Both the size of the largest IND found directly and the size of the largest *merged* IND are reported. The reason for the reduction in quality for larger relations is that in order for the algorithm to finish, we had to prune the search space by limiting the number of nodes and edges of the search hypergraph. The increase of quality for large relations may be due to the random projections of relations that were performed to obtain problem subsets.

The power of the IND-merging strategy (Sec. 4.4) becomes clear for very large relations, as the size of the largest discovered IND (relative to the size of the largest *existing* IND) actually increases.

7.6 Experiment 5: Number-of-Distinct-Tuples Parameter in DV Heuristic

The Distinct-Value (DV) heuristic rejects valid INDs as spurious when the number of distinct values is lower than a certain threshold n. A study of the statistic effects of this heuristic is given in Sec. 3.3. In this experiment, we varied the parameter n, whose

Table 4. Size of largest IND discovered relative to size of largest valid IND in a difficult case. In cases marked "N/A", the algorithm did not finish.

# of Attributes	Algorithm		
	Non-heuristic	Heuristic	Heuristic w/ IND-merging
10	100%	100%	100%
20	N/A	95%	95%
30	N/A	50%	50%
40	N/A	33%	33%
52	N/A	38%	38%
64	N/A	41%	44%
86	N/A	36%	53%

default value is 7, from 0 to 15. An $n = 0$ represents the exact (non-heuristic) algorithm. Table 5 shows the higher-arity INDs that the FIND_H algorithm found for a test case from the CENSUS experiment set, for different n.

For this experiment, there was a single true IND to be discovered, with 41 attributes (column 1). Two effects are apparent: First, the size of the largest IND discovered by the heuristic algorithm FIND_H decreases as the DV heuristic declares more and more small INDs spurious. Second, the algorithm also discovers more INDs, with different sizes, such that the solution becomes "spread out". For $n > 7$, the solution quickly deteriorates, as predicted by the theory. Note that for this experiment, IND-merging (i.e., computing the union of the attribute sets in all discovered INDs) yielded the "true" solution of a 41-ary IND.

Furthermore, the algorithm actually becomes slower for larger values of n. One reason is that the size of the discovered solution (i.e, the number of minimal unique INDs) increases (see also Fig. 8). Another reason is that the (time-consuming) AVD-heuristic (Sec. 4.2) is used more often as the DV-heuristic declares more INDs spurious (since the AVD-heuristic is applied to INDs rejected by the DV heuristic).

7.7 Experiment 6: Accuracy of the χ^2-Test and the AVD Heuristic

The *attribute value distribution (AVD) heuristic* relies on the assumption that attributes that stand in an inclusion relationship to one another are semantically related and thus show a similar distribution of their values. This will be true if the two relations in question are actually random samples of some larger real-world data set. However, if algorithm FIND_2 is run on two relations R and S, with one or both of R and S being selected from a larger set D on a predicate $(R = \sigma_{C_1}(D) \vee S = \sigma_{C_2}(D))$, the value distribution in some attributes in R might be different from the value distribution in some attributes in S.

Thus, we performed a number of experiments in which we generated subsets of our data sets using *predicates* rather than random sampling. The expectation is that the AVD heuristic will produce many false negatives in the presence of such predicates, which

Table 5. Large INDs discovered with Different Thresholds for the DV Heuristic

IND Size	Declare IND spurious if $n \leq$						
	0	4	6	7	8	10	15
≤ 24							4
25						1	1
26							3
27							1
28						2	4
29							2
30						1	2
31					1		
32				2	1	6	
33			1	2	5	2	
34			1	2	4		
35			4	1			
36			2	1			
37		2					
38							
39		2					
40							
41	1						
Runtime (sec)	650	381	356	334	388	408	455

motivates the design to only run this heuristic after the DV heuristic has already rejected an IND (Sec. 4.3).

Table 6 shows the quality (ratio of size of largest IND found to size of largest existing IND) of the result in data set INTERNET for four different predicates. The data set represents a survey in Internet usage data, and we selected the following four attributes for predicates: *gender*, *household income*, *country/state of origin* (encoded in a single attribute in the original data source), and *major occupation*, with conditions that had selectivities between 0.45 and 0.8. For example, selecting tuples with a predicate such as `GENDER<>'female'` will change the value distribution of the values in the other columns if they are gender-specific. Likewise, a predicate such as `HOUSEHOLD_INCOME<75,000` will probably change the value distribution in the other columns of this table, which represents an Internet usage survey.

We performed similar experiments with our other data sets and found that the AVD heuristic helps to find between 50% (data set CUP98) and 10% (data set INSURANCE) larger INDs than the algorithm with only the DV heuristic, averaged over several different predicates. This experiment shows that using the AVD heuristic gives better results (i.e., more accurate large INDs) in most of our experimental cases in which it was actually applied. It never reduces the quality of the result due to the way it is used in algorithm $\mathsf{CHECK}_\mathsf{H}$.

Table 6. Relative size of largest discovered IND, with subsets selected by predicate.

Attribute	Predicate	with AVD	without AVD
GENDER	`<>'female'`	81%	43%
HOUSEHOLD_INCOME	`< 75000`	94%	43%
COUNTRY	`='US' AND state <= 'North Carolina'`	85%	42%
MAJOR_OCCUPATION	`<>'other'`	88%	43%

7.8 Experiment 7: Discovering Non-exact Relationships

In this experiment, we tested the hypothesis established in Sec. 6, that the FIND_H algorithm can be used to discover approximate relationships between tables that are not exact inclusions of one another.

Our test case consisted of data from the US Census database[4], with the goal of letting the algorithm discover that the census data from two small states (North and South Dakota) are related. As explained in Sec. 6, due to the use of set-difference queries at the lowest level, the algorithm in its current form can not be used to compare distinct relations; some overlap is required. We therefore generated two overlapping tables by introducing tuples of each state's microcensus table into the other. We obtained two tables with about 5,000 tuples each, which had an intersection of about 3,500 tuples, and about 1,500 unique tuples each. For this experiment, we then projected those tables to 15 randomly selected attributes. We then let the FIND_2 and the FIND_H algorithms attempt to discover the one-to-one attribute correspondence between the two tables (i.e., a single 15-ary inclusion-dependency-like pattern, which implies 15 unary and 105 binary patterns).

While the non-heuristic FIND_2 algorithm generated 126 unary and over 6000 binary patterns, and subsequently did not finish, the FIND_H algorithm performed very well. It generated only 37 unary and 275 binary patterns (with the DV and AVD heuristics in place), well within the capabilities of the clique-finding algorithm. It finished after 248 seconds, and found a 13-ary relationship between the two input tables (after merging). The two attributes not found both had only 2 values in their domains and were highly correlated, making them indistinguishable for the algorithm.

This experiment suggests that the FIND_H algorithm can be used to discover relationships between database even in the presence of substantial noise, or even if the tables only partially overlap rather than form subsets of one another.

8 Related Work

There is substantial work on the discovery of patterns in databases. Much work is concentrated on functional dependencies (FDs), such as Lim and Harrison [21].

[4] One-percent microcensus:
`ftp://ftp2.census.gov/census_2000/datasets/PUMS/`.

An important related paper is by Kantola, Mannila *et al.* [2]. The authors describe an algorithm for discovering functional dependencies and also mention inclusion dependencies. However, no algorithm for IND discovery is given, and only a very rough upper bound for the complexity of the IND-finding problem is presented (in addition to a proof of NP-completeness of the problem).

Much database pattern discovery uses the concept of *levelwise search*, which has a well known instantiation in the *Apriori*-algorithm for association rule mining [11]. Mannila and Toivonen [12] give a theory of levelwise searches, and introduce the concept of *borders of theories* for discovery algorithms.

Zaki [22] uses levelwise search as well as the idea of cliques (but not hypercliques) for association rule mining. In this paper, the author also mentions *clique-merging*, which is similar to our *IND-merging*.

Hypergraphs have been used in other areas of databases and data mining. For example, Mannila and Räihä [3] give an algorithm for the discovery of functional dependencies that maps the problem to a hypergraph traversal.

Inclusion dependencies have been widely studied on a theoretical level. Fundamental work is done by Casanova, Fagin and Papadimitriou [1]. They present a simple axiomatization for INDs. While their work focuses on inference of INDs, not discovery, we use their "projection and permutation" axiom as the basis for the FIND_2 algorithm. Casanova et al. further prove that the decision problem for INDs (i.e., deciding whether a given IND can be derived from a given set of INDs through inference) is PSPACE-complete. Chandra and Vardi [23] prove undecidability of the problem. Mitchell [24] developed inference rules for INDs. No discovery on the data-level is mentioned in that body of work.

De Marchi *et al.* first proposed a levelwise algorithm for IND discovery [10]. The algorithm is competitive for very small problems, especially due to the use of an inverted index for unary IND discovery, but suffers from the dimensionality curse for IND sizes beyond about 8. More recently, deMarchi *et al.* proposed the *Zigzag* algorithm [14] which is very similar to the FIND_2 algorithm presented by the authors in [9,13]. There are significant differences such as the hypergraph model (we use k-uniform hypergraphs vs. de Marchi's general hypergraphs) and the discovery algorithm (our hypercliques vs. de Marchi's minimal hypergraph traversals). In addition, de Marchi treats invalid large IND candidates (such as c_2 in Fig. 2) differently from us, by attempting to validate them by removing single attributes. The choice of strategy is guided by a heuristic based on the number of tuples violating the IND property in the proposed IND. His ideas are orthogonal to ours, and we expect that a pooling of ideas might lead to an overall more optimized algorithm. In any case, the results from this paper would apply equally to FIND_2 and *Zigzag*.

There is substantial related work on the mathematical foundations of some of the heuristics that we have used to restrict problem spaces in our algorithm. Work on the theory of attribute value distributions can be found in [25] and [26]. The statistical χ^2-Test itself is described in statistics textbooks such as [20].

Schema integration is not limited to the discovery of INDs. In fact, there is a very large body of work in meta-data driven (as opposed to data-driven) schema integration.

Rahm and Bernstein [8] give an overview over some recent schema-integration projects; an earlier survey is [27].

Larson *et al.* [28] give a theory in which they infer attribute equivalence by a variety of indicators, such as domains, maximal and minimal values, and some constraints imposed by the (relational) database system. Their work is complementary to ours in some sense but ignores the actual data inside the attributes. Therefore, it is very sensitive to the availability and correctness of their assumed constraints.

More ideas on schema matching are contained in the *SemInt* project [29], in which attribute equivalence is inferred based on 20 different features of an attribute, five of which (minimum, maximum, average, coefficient of variance, standard deviation) are based on data but represent very simple properties and apply only to numeric attributes. These 20 dimensions are then used to train a neural network classifier for inferring attribute relatedness. Doan et al. [30] use a similar machine-learning approach to infer related schema elements in semistructured databases.

Kang and Naughton [31] present another schema matching approach, in which they map each of two relations into a graph and then perform graph matching to achieve schema matching. They use the assumption that attributes with similar entropy are related and also take intra-relational mutual information of attributes into account. The entropy heuristic applies to all data types and is somewhat related to our AVD measure, but is only a one-dimensional measure which incurs many false positives. The authors report that their approach does not scale beyond 15–20 attributes due to the deterioration of their heuristic.

9 Conclusion

In this paper, we have proposed heuristics that help to scale hypergraph-based inclusion dependency discovery algorithms [13,14]. We have shown that significant performance benefits are possible by applying the concept of *spurious* IND. This concept is used to reduce the problem size for exponential-complexity algorithms. This strategy makes it possible to automatically discover overlaps between almost any pair of real-world size relations. Even relations with many meaningless single-attribute overlaps (introduced by domains with few and accidentally identical values between those attributes) can be used for robust discovery.

Applications of this work lie in database integration (particularly, schema matching), reorganization, and query optimization. It could also be potentially beneficial in other application domains, since exponential-complexity mapping problems are common in subset and similarity discovery problems.

A potential direction into which to take this work is a further generalization of the problem, moving away from the discovery of *exact subsets* between relations and towards true *similarity*. This would entail relaxing the assumptions (1) that all tuples in the "included" relation actually exist in the other and (2) overcoming the problem that values across the attributes must match exactly for an inclusion dependency, both of which are receiving some attention in the research community already While the first problem is addressed in this paper and the $\mathsf{FIND}_{\mathsf{H}}$ algorithm can be used for discovery tasks in this category, see for example [31] for the second problem.

References

1. Casanova, M.A., Fagin, R., Papadimitriou, C.H.: Inclusion dependencies and their interaction with functional dependencies. In: Proceedings of ACM Conference on Principles of Database Systems (PODS). (1982) 171–176
2. Kantola, M., Mannila, H., Räihä, K.J., Siirtola, H.: Discovering functional and inclusion dependencies in relational databases. International J. Of Intelligent Systems **7** (1992) 591–607
3. Mannila, H., Räihä, K.J.: Algorithms for inferring functional-dependencies from relations. Data & Knowledge Engineering **12** (1994) 83–99
4. de Marchi, F., Lopes, S., Petit, J.M., Toumani, F.: Analysis of existing databases at the logical level: the DBA companion project. SIGMOD Record (ACM Special Interest Group on Management of Data) **32** (2003) 47–52
5. Lee, A.J., Nica, A., Rundensteiner, E.A.: The EVE approach: View synchronization in dynamic distributed environments. IEEE Transactions on Knowledge and Data Engineering (TKDE) **14** (2002) 931–954
6. Gryz, J.: Query folding with inclusion dependencies. In: Proc. Intl. Conf. on Data Engineering, IEEE Computer Society (1998) 126–133
7. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. Proceedings of IJCAI (2003) 16-21
8. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal: Very Large Data Bases **10** (2001) 334–350
9. Koeller, A.: Integration of Heterogeneous Databases: Discovery of Meta-Information and Maintenance of Schema-Restructuring Views. PhD thesis, Worcester Polytechnic Institute, Worcester, MA, USA (2001)
10. de Marchi, F., Lopes, S., Petit, J.M.: Efficient algorithms for mining inclusion dependencies. In: Proceedings of International Conference on Extending Database Technology (EDBT). (2002) 464–476
11. Aggarwal, C.C., Yu, P.S.: Online generation of association rules. In: Proceedings of IEEE International Conference on Data Engineering. (1998) 402–411
12. Mannila, H., Toivonen, H.: Levelwise search and borders of theories in knowledge discovery. Data Mining and Knowledge Discovery **1** (1997) 241–258
13. Koeller, A., Rundensteiner, E.A.: Discovery of high-dimensional inclusion dependencies. In: Proceedings of IEEE International Conference on Data Engineering, Bangalore, India, IEEE (2003) 683–685
14. de Marchi, F., Petit, J.M.: Zigzag: A new algorithm for mining large inclusion dependencies in databases. In: 3rd Intl. Conf. on Data Mining, Melbourne, Florida, IEEE (2003) 27–34
15. Mitra, P., Wiederhold, G., Jannink, J.: Semi-automatic integration of knowledge sources. In: Proc. of the 2nd Int. Conf. On Information Fusion (FUSION'99), Sunnyvale, California (1999)
16. Beneventano, D., Bergamaschi, S., Castano, S., et al.: Information integration: The MOMIS project demonstration. In: International Conference on Very Large Data Bases. (2000) 611–614
17. Koeller, A., Rundensteiner, E.A.: Heuristic Strategies for Inclusion Dependency Discovery In: Proceedings of 3rd International Conference on Ontologies, Databases and Applications of Semantics (ODBASE) (2004) 891–908
18. Koeller, A., Rundensteiner, E.A.: Discovery of high-dimensional inclusion dependencies. Technical Report WPI-CS-TR-02-15, Worcester Polytechnic Institute, Dept. of Computer Science (2002)

19. Demetrovics, J., Thi, V.D.: Some remarks on generating armstrong and inferring functional dependencies relation. Acta Cybernetica **12** (1995) 167–180
20. Rice, J.A.: Mathematical Statistics and Data Analysis. 2nd edn. Duxbury Press (1995)
21. Lim, W., Harrison, J.: Discovery of constraints from data for information system reverse engineering. In: Proc. of Australian Software Engineering Conference (ASWEC '97), Sydney, Australia (1997)
22. Zaki, M.J.: Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering (TKDE) **12** (2000) 372–390
23. Chandra A.K., Vardi, M.Y.: The implication problem for functional and inclusion dependencies is undecidable. SIAM J. Comput. **3** (1985) 671–677
24. Mitchell, J.C.: Inference rules for functional and inclusion dependencies. In: Proceedings of ACM Symposium on Principles of Database Systems, Atlanta, Georgia (1983) 58–69
25. Mannino, M.V., Chu, P., Sager, T.: Statistical profile estimation in database systems. ACM Computing Surveys **20** (1988)
26. Hon, W.C., Zhang, Z., Zhou, N.: Statistical inference of unknown attribute values in databases. In: Proceedings of International Conference on Information and Knowledge Management. (1993) 21–30
27. Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys **18** (1986) 323–364
28. Larson, J.A., Navathe, S.B., Elmasri, R.: A theory of attribute equivalence in databases with application to schema integration. IEEE Transactions on Software Engineering **15** (1989) 449–463
29. Li, W., Clifton, C.: SemInt: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. Data and Knowledge Engineering **33(1)** (2000) 49–84
30. Doan, A., Domingos, P., Halevy, A.: Learning source description for data integration. In: Proceedings of the Third International Workshop on the Web and Databases (WebDB), Dallas (2000) 81–86
31. Kang, J., Naughton, J.F.: On schema matching with opaque column names and data values. Proceedings of SIGMOD (2003) 205–216