Discovery of Constraints and Data Dependencies in Relational Databases (Extended Abstract)*

Siegfried Bell & Peter Brockhausen
Informatik VIII University Dortmund
44221 Dortmund Germany
email: {*bell*, *brockh*}@ls8.informatik.uni-dortmund.de

1 Introduction

Data dependencies are the most common type of semantic constraints in relational databases which determine the database design. Despite the advent of highly automated tools, database design still consists basically of two types of activities: first, reasoning about data types and data dependencies and, second, normalizing the relations. Automatic database design may serve as a process to support database designers with a dependencies proposing system, which may help to design optimal relation schemes for those cases where data dependencies are not obvious. The so called dependency inference problem is described in [Mannila and Räihä, 1991] as: Given a relation r, find a set of data dependencies which logically determines all the data dependencies which are valid in r.

Unfortunately, it is impractical to enumerate all data dependencies and to try to verify each of them. Alternatively, a second approach to discovery is to avoid unnecessary queries by inferring as much as possible from already verified data dependencies. A third approach is to draw inferences not only from verified data dependencies but also from invalid data dependencies, the so called independencies. In this paper we will follow this approach. A second problem is that real world databases are known to be very large. Therefore they only can be accessed via a database management system.

We present a rough sketch of our main algorithms and in the last section a comparision with some similiar approaches.

Our system can be seen at the first glance as an optimized version of CLAUDIEN regarding functional dependencies, [Dehaspe et al., 1994]. But there are differences: first, in CLAUDIEN the relationship between the dependencies is based on θ–subsumption and the verification of the hypotheses on theorem proving. In our approach, the relationship of the dependencies is based on an axiomatization of FDs and UINDs. The verification is done by the database management system which groups the rows. This offers several advantages: First, theorem proving is for this purpose too powerful and we can infer dependencies by transitivity which is really simple. Second, we can find dependencies in relational databases, which can not be stored in the main memory as PROLOG assertions.

* The full paper is available from the authors.

2 Discovering Data Dependencies

2.1 Value Restrictions

Value restrictions are the upper and lower bounds of attribute domains. We select the minima and maxima for all attributes in all relations with the corresponding SQL statements. The SQL statement uses the normal order on numbers for numerical attributes and the lexicographic order on the character set for attributes of a symbolic type. Since it is possible to compute the two values in one query, the overall costs are $\mathcal{O}(n * m)$. Throughout this section n denotes the number of attributes in all tables and m the maximal number of tuples in the table which possesses the most.

2.2 Unary Inclusion Dependencies

A naive algorithm for computing inclusion dependencies has a time complexity of $\Theta(n^2 * m^2)$. It generates exactly $\frac{n*(n-1)}{2}$ database queries, if the corresponding UINDs are valid or not. In contrast our algorithm has a overall time complexity of $\mathcal{O}(n^4 + n^2 * m^2)$.

At a first glance, this result looks strange because of the $\mathcal{O}$–notation. But our algorithm has one very important property. Given a fixed sequence of attributes for hypotheses testing, our algorithm always poses a minimal number of database queries for the discovery of UINDs, by exploiting discovered UINDs and the transitivity of UINDs, and hence it saves all superfluous queries to the database, cf. [Brockhausen, 1994]. The correctness of the algorithm is considerably based on the axiomatization of UINDs and UINIs, cf. [Bell, 1995].

2.3 Functional Dependencies

To determine functional dependencies we have integrated two main ideas, namely to exploit the transitivity of FDs and to concentrate on the computation of most–general FDs. For a discussion in full detail on the algorithm and the design choices being made see [Brockhausen, 1994]. Our algorithm uses a top–down and breadth–first search strategy. We should mention that we also use information of the database system on primary keys, indexes and null values and the discovered inclusion dependencies to reduce the search space in our algorithm.

Figure 1 lists the statement and the condition which must hold. The clue is the GROUP BY instruction. The computational costs of this operation are dependent on the database system, but it can be done in time $\mathcal{O}(m * log\ m)$. However, the worst case time complexity of every such an algorithm is exponential, due to the results of [Mannila and Räihä, 1991].

3 Evaluation and Conclusions

We compared our algorithm with two approaches presented at the AAAI workshop on knowledge discovery in databases in 1993. Savnik and Flach call their method "bottom–up induction of functional dependencies from relations", see

```
1.    SELECT SUM (COUNT (DISTINCT $A_1$)),
                 SUM (COUNT (DISTINCT B))
      FROM R
      GROUP BY $A_1, \ldots, A_n$                  $=: a_1, b$
2.    $a_1 = b \Rightarrow A_1 \ldots A_n \rightarrow B$
```

Fig. 1. A SQL–statement for the Computation of Functional Dependencies

[Savnik and Flach, 1993]. Briefly, they start with a bottom–up analysis of the tuples and construct a negative cover, which is a set of FIs. Therefore they have to analyze all combinations between any two tuples.

In the next step they use a top–down search approach similar to ours in order to discover the functional dependencies. They check the validity of a dependency by searching for FIs in the negative cover. Schlimmer also uses a top–down approach, but in conjunction with a hash–function in order to avoid redundant computations [Schlimmer, 1993].

Algorithm by	Database	$\|r\|$	$\|R\|$	$\|X\|$	Time
Savnik and Flach	Lymphography	150	19	7	9 min.
Schlimmer	Breast Cancer	699	11	4	1 h 14 min.
Bell and Brockh.	Lymphography	150	19	7	> 33 h
Bell and Brockh.	Breast Cancer	699	11	11	8 Min. 53 sec.
Bell and Brockh.	Breast Cancer	699	11	4	4 Min. 19 sec.

Table 1. Comparison of the Experimental Results from [Savnik and Flach, 1993] and [Schlimmer, 1993] with the algorithm FUNCTIONAL DEPENDENCIES.

But in contrast to our algorithm, in both articles mentioned, the authors do not use a relational database like OracleV7 or any other commercial DBMS. They even do not use a database at all. And this has some important effects on the results, which will be discussed in the next paragraph. Table 1 shows a summary of their results, where $|r|$ denotes the number of tuples, $|R|$ the number of attributes, $|X|$ the maximal number of attributes on the left–hand side of a FD and time is the time needed for the discovery of a most–general–cover. For comparison reasons we introduced such a bound on the number of attributes in our algorithm.

First, our algorithm cannot detect the FDs in the lymphography domain in reasonable time, because we do not hold the data in main memory like Savnik and Flach. And since most of the FDs are really long, for some attributes the shortest most–general FDs have already seven attributes on the left side, the search space and the overhead for the communication with the database is to big. But it cannot be said that our approach is inferior to the one of Savnik and Flach, because the circumstances are to different, namely the presence or absence of a database for the storage of the tuples.

Second, in the breast cancer domain our algorithm is really fast, more than seventeen times faster than Schlimmer's algorithm. Even without any bound on the length of the FDs it is still eight times faster and it uses a database.

Database	$\|r\|$	$\|R\|$	$\|X\|$	Time	N
Books	9931	9	9	4 h 44 min.	25
Books	9931	9	6	4 h 40 min.	25
Books	9931	9	3	2 h 10 min.	20

Table 2. Summary of the results of the algorithm FUNCTIONAL DEPENDENCIES.

But of course the two domains above are not typical database applications. Table 2 shows the results of our algorithm with respect to a real database, the library database of the computer science department. Here it becomes obvious that our pruning criterions are efficient, because with a bound of six attributes and without any bound the time needed is nearly the same. The differences are neglectable because there are many more users working on the network and the results are only reproducible within some bounds. But apart from the known primary key of the database the discovered FDs are semantically meaningless.

In summary, one can say that the algorithm which we present in our work has one important advantage over the two approaches mentioned above. The algorithm is capable of dealing with great amounts of data, because we use a real database for the storage. And as a side effect, because we use standard SQL–statements for the discovery of FDs, our approach is portable and we can use any database which "understands" SQL as a query language.

Acknowledgment: This work is partly supported by the European Community (ESPRIT Basic Research Action 6020, project Inductive Logic Programming) and the Daimler–Benz AG, Contract No.: 094 965 129 7/0191.

References

[Bell, 1995] Bell, S. (1995). Inferring data independencies. Technical Report 15, University Dortmund, Informatik VIII.

[Brockhausen, 1994] Brockhausen, P. (1994). Discovery of functional and unary inclusion dependencies in relational databases. Master's thesis, University Dortmund, Informatik VIII. in german.

[Dehaspe et al., 1994] Dehaspe, L., Laer, W. V., and Raedt, L. D. (1994). Applications of a logical discovery engine. In Wrobel, S., editor, *Proc. of the Fourth International Workshop on Inductive Logic Programming*, GMD-Studien Nr. 237, pages 291–304, St. Augustin, Germany. GMD.

[Mannila and Räihä, 1991] Mannila, H. and Räihä, K.-J. (1991). *The design of relational databases.* Addison-Wesley.

[Savnik and Flach, 1993] Savnik, I. and Flach, P. (1993). Bottum-up indution of functional dependencies from relations. In Piatetsky-Shapiro, G., editor, *KDD-93: Workshop on Knowledge Discovery in Databases.* AAAI.

[Schlimmer, 1993] Schlimmer, J. (1993). Using learned dependencies to automatically construct sufficient and sensible editing views. In Piatetsky-Shapiro, G., editor, *KDD-93: Workshop on Knowledge Discovery in Databases.* AAAI.