Web Information Integration Tool: Data Structure Modelling

Martin Rimnac
Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodarenskou vezi 2,
182 07 Prague 8,
Czech Republic
rimnacm@cs.cas.cz

***Abstract*— The paper describes a method for relational data model estimation from input web data and usage of this method. It includes also its principal limitations and shows the model usage for a more effective storage into a repository. The repository is implemented as the universal relation. The properties of the model are described as well.**

Keywords: data structure model, information retrieval

I. Introduction

Web pages contain a lot of information. The usage of web page content has been studied since the Internet boom; the well-known products are search engines or web autonomy robots.

The information retrieval "on the web" [1] approach continually develops: There are new methods for web pages mapping, a similarity evaluation between documents or new methods for page ranks. Many of them interpret a web page as a simple list of words, nothing else. Does there exist a kind of a web page, which could be read by humans and also could be processed by automatic tools - at a higher level - in a helpful way for humans?

One type of these pages could be a web interface to information systems or to content managers. Information is presented in this case as a view of a database and the view is formatted to the XHTML page, to the XML or RDF format document or to the plain text (CSV format). This kind of web pages is very important for the inverse task - information retrieval from a web page, and document transformation back into a database. A web robot providing data collection could include: a transformation tool, a repository, an integration tool and a reporter. The transformation tool extracts information from a document to a set of tuples. The tuples are stored in a repository, which is implemented as the universal relation. The integration process follows, synonyms at an attribute name level are processed. The last step is a portal reporter, which is used for viewing and searching information contained in the repository.

The database implementation of the universal relation ([2], [3]) and the operations in the repository are described in the paper. The paper includes also a description of a method, which automatically estimates a data structure model from an input dataset and which enables to store information in the repository in a more effective way. The motivation for the data structure model usage is the same as the normal forms for the relational data model. The obtained model could be used as a final relational data model by assuming input data completeness; in such a case the data structure model is the nearest to the logical data model. The data structure model describes the functional dependency system of the data stored in the repository, so the role of this model is similar to ontologies in the semantic web context.

In the past, decomposition methods leading to the relational data model modifications have been published. The input of these methods is usually a set of functional dependencies in the current model and requirements for new ones (normal forms, multi-level relations, etc.). These methods use the theory of sets or graph/hypergraph theory ([4], [5]) for relation and subrelation interpretation. There are many different approaches as finding the closure of the set [6], the step by step decomposition from one relation into its subrelations, removing redundant functional dependencies [7], converting the relational data model into a multi-level secure one [8]. All of these methods fulfil requirements such as minimal redundancy, representativeness or separativness [9]. The vertical [10] and horizontal [11] partitioning must be also included in this content.

Then there exist also new approaches in XML document integration [12], the first semantic information searching tools [13] and project Froogle [14], which all attempt to solve similar issues.

II. Universal Relation Implementation

The universal relation can be used to describe and store any object of a universum and its properties. The simplest implementation of the universal relation ([2], [3]) is the relationship creation between an attribute set, the active domains of these attributes and the numerator of tuples. This model of the universal relation can be extended by attribute name similarity relations (in clasical mathematical meaning) for the integration process. In any case all attribute values of any tuple stored in the repository are accessed in one step.

The main drawback of this simple model is functional dependencies non-inclusion leading to all tuple attribute values to be stored in a repository. But there could exist attribute/s with the same values in different tuples or with the value depending on the value of other attribute/s. In such a case,

the storage of all such values in a repository is redundant. It is the reason, why it is better to use the data structure model, which describes a functional dependency system in a repository and stores only attributes with different values or keys of subinstances. The possible handicap of this solution is not to retrieve values of all the attributes in one step.

A. Data Structure Modelling

The *data structure model* describes a functional dependency system. It uses a graph theory interpretation for functional dependencies: there is an oriented arc between attributes in the data structure model graph, if the second attribute functionally depends on the first one. The *covering matrix* is used for the storage of the arcs.

B. Basic Properties of Functional Dependencies

If attributes depend mutually, they have the same size of their active domains.

$$A_1 \rightarrow A_2 \wedge A_2 \rightarrow A_1 \Rightarrow \|\mathscr{D}_\alpha(A_1)\| = \|\mathscr{D}_\alpha(A_2)\| \quad (1)$$

The set of functional dependencies has a transitive property:

$$A_1 \rightarrow A_2 \wedge A_2 \rightarrow A_3 \Rightarrow A_1 \rightarrow A_3 \quad (2)$$

The functional dependency can exist only in the case, when the size of the depending attribute active domain is not greater than the size of the active domain of the attribute on which it depends.

$$A_1 \rightarrow A_2 \Rightarrow \|\mathscr{D}_\alpha(A_1)\| \geq \|\mathscr{D}_\alpha(A_2)\| \quad (3)$$

For complex-attribute functional dependencies, if a value of the (complex-)attribute H depends on the (complex-)attribute G, then the attribute H depends also on the attribute G extedend with a random added attribute (these dependencies are called trivial).

$$G \rightarrow H \Rightarrow G' \rightarrow H \quad \forall G' \supset G \quad (4)$$

The consequence of (4) is the data structure model contains a complex-attribute non-trivial functional dependency with m attributes on one side, if these m attributes have no dependency between them.
The observation is that if a tuple of a new entity type with all new attributes is added into a repository, these new attributes are mutually depended. Thanks to (4), there is no complex-attribute functional dependency in a new added entity type tuple.

C. Tuple insertion into a repository

In previous sections, the universal relation implementation and basic properties of functional dependencies were described. Now, let us try to insert several tuples of one entity type (with n attributes) into an empty repository.

For the first tuple, there is no problem, the tuple will be added into a repository and all functional dependencies between n attributes will be added into the data structure model. The structure model will contain $n(n-1)$ functional dependencies, self-attribute or other trivial dependencies are not considered.

Let us assume the second tuple has some attributes with the same value as the first one and some with different. Several functional dependencies will be corrupted, because there exist attributes with different sizes of active domains. These corrupted dependencies will be removed from the data structure model.

Now the question, how a functional dependency corruption is determinated, arises. The $o(n)$ test for the functional dependency $X \rightarrow Y$ corruption after the n-th tuple addition is

$$\exists i < n \quad x_i = x_n \wedge y_i \neq y_n \quad (5)$$

One way is to perform a test for all functional dependencies in the data structure model. The second, more effective, is to create the data structure model *skeleton*. Only the skeletons reflecting the order of attributes given by the active domain size (3) of non-complex attributes will be considered:

$$\|\mathscr{D}_\alpha(A_i)\| < \|\mathscr{D}_\alpha(A_j)\| \Rightarrow i < j \quad (6)$$

The observation is that the skeleton containing functional dependencies with minimal distance between the attribute active domain sizes is the most useful. These dependencies are located nearby a diagonal of the covering matrix. The skeleton also contains functional dependencies, which forks the path in the skeleton (the skeleton without these arcs is a tree). All the functional dependencies not containing in the skeleton will be called *non-selected*. The result of this observation enables, by usage of (2, 6), making a test for functional dependency corruption only for the dependencies in the skeleton. If the test fails, all dependencies with the same right or left side must be tested.

In the case of functional dependency in the skeleton being corrupted, the relation is divided into subrelations according to the new data structure model.
In the situation, when there are m attributes with no dependencies between them, the complex-attribute functional dependency determination is enabled as a consequence mentioned after (4). All dependencies between the already existing attributes and the newly added complex-attribute are tested.
On success, the new comlex-attribute functional dependencies are added into the data structure model. The location of the new complex-attribute in the covering matrix is important for the model skeleton, the complex-attribute hierarchy must be added to the ordering criterion (6). All possible subparts of the new complex-attribute present in the model are moved up into the higher hierarchy level. It means the so called *virtual attribute* having active domain equal to the Cartesian product of all concerned attributes is created. This enables to use a graph skeleton algorithm instead of a hypergraph one. The modified criterion for the attribute order is:

$$\|\mathscr{D}_\alpha(G_i)\| < \|\mathscr{D}_\alpha(G_j)\| \wedge G_i \nsubseteq G_j \quad \Rightarrow \quad i < j \quad (7)$$

$$G_i \subset G_j \quad \Rightarrow \quad i > j \quad (8)$$

where G_i represents complex-attribute.

This modified criterion allows represention of complex-attribute functional dependencies (in a special case the multi-valued ones) in the data structure model. Of course, the condition on the domain size (3) holds for complex-attribute dependencies too.

III. Data Structure Model

This section deals with the advanced properties of the data structure model. The model reflects a functional dependency system in a repository. The relations in a repository are automatically divided into subrelations, if necessary.

Generally, the data structure model in time of initialization contains $n(n-1)$ functional dependencies, but implicitly contains $n!$ of them (all possible functional dependencies, including the trivial ones). The functional dependencies in the model can be removed with a new tuple insertion. When no new instance is available, the data structure model may be similar (or same) to the logical one. The assumption of no next tuple is very important, but it is not sufficient. The non-existing in the real world - but present in the model - dependencies are caused by limitations of resources. This kind of limitation could be resolved by a complex integration process, but it does not guarantee a solution in all cases.

On the other hand, the data structure model evolution is monotonic (down going number of possible functional dependencies) and the model is unique to a set of tuples (no dependency on a tuple order). Formally, the models could be ordered by number of all (skeleton, non-selected and trivial) functional dependencies (denoted $|M|$) implied by the data structure model. Then it binds:

$$|M_0| = n! \tag{9}$$

$$M_i < M_j \Rightarrow |M_i| > |M_j| \tag{10}$$

$$\forall M_k : M_0 \leq M_k \leq M_I \leq M_\infty \tag{11}$$

In this notation, the estimated data structure model M_k after the $(k+1)$-th tuple insertion is located by the model ordering between the model after initialisation M_0 and the model M_I containing information from all integrated data sources, which can have more functionaly dependencies that the logical one M_∞.

IV. Implementation and Experiment

According to the given ideas, a prototype was implemented in a PostgreSQL [15] database system. The prototype was tested on an experimental data set containing information about rates at an exchange office during two days. The data were extracted from CSV format file downloadable from the internet [16]. The tested set has 62 tuples, 31 per a day.

The graph in the Fig. 1 presents the relationship between the number of the functional dependencies contained in the model and a number of already stored tuples in a repository. Each column is divided into two groups and each group is separated into two parts. The upper group of the functional dependencies represents the non-selected functional dependencies (denoted NS), the bottom one the selected (denoted S) functional dependencies, i.e. used in the model skeleton, which must be tested after each new tuple is added. The righttop-leftbottom hatching parts are the mutually functional dependencies, i.e. $A_i \rightarrow A_j$ and $A_j \rightarrow A_i$ and $i > j$; the reverse hatching parts are others.

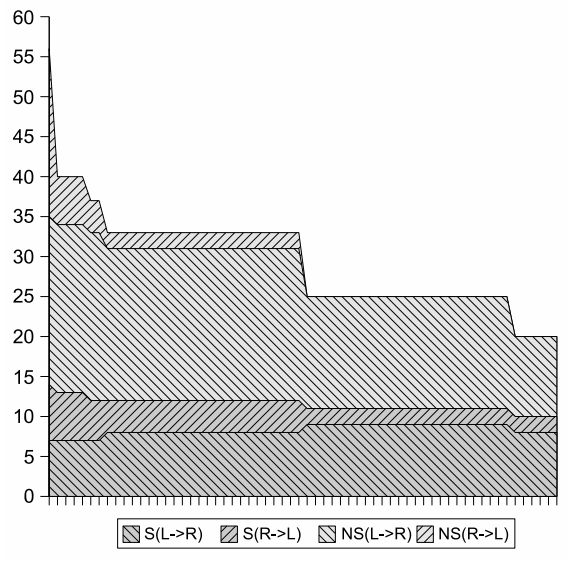

Fig. 1. Number of Dependencies

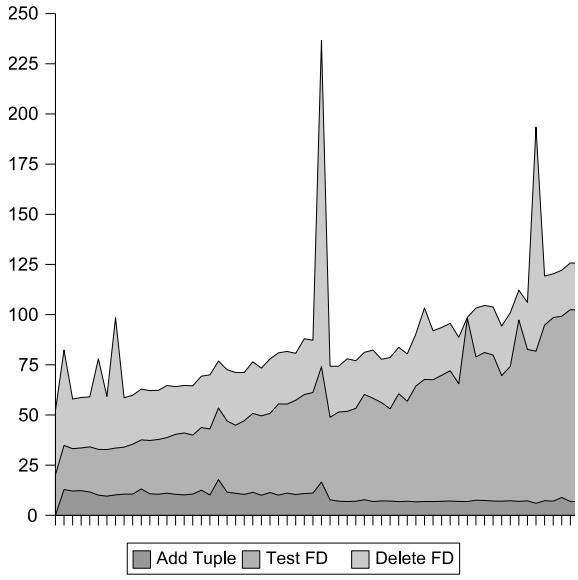

Fig. 2. Benchmark t[ms]

The observation, that the model skeleton contains at most $2n$ functional dependencies, can be proved by the graph theory combined with (2) and (4). The number is getting lower, the model estimation is monotonic (11). The figure corresponds with the model theory given before.

The model estimation process benchmark is given in the Fig. 2. The process is separated into three phases. The first is the tuple addition into a repository; it includes the attribute or

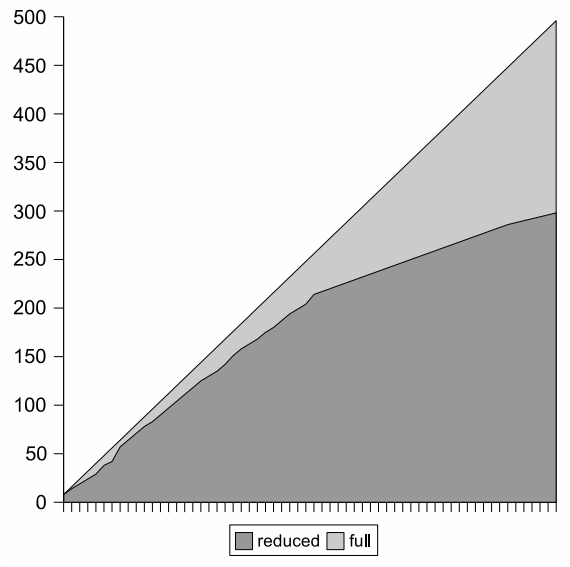

Fig. 3. Items in the repository

the value record creation, if necessary, see the time difference between the first and second half of the dataset. The second step is the corruption of dependencies in the skeleton testing; if some corrupts, all transitive dependencies with the same left or right side are tested and all together, in the case of the corruption, are removed from the model. The test for the non-selected ones and removing of corrupted is made in the third phase.

The most complex operation was the addition of the first tuple of the second day information, because a complex-attribute is tested (Delete FD peak).

The Fig. 3 compares two ways of a dataset storage into a repository. The first one, without the model estimation, is linearly increasing, all pairs attribute-value are stored. The second one uses the model estimation and stores only pairs of the decompositioned relations. The difference is straightforward in the second half of the dataset, which contains information about the second day and all the time invariant attribute values are already stored.

V. Conclusion

The paper shows another way of mining web pages different from the semantic web ideas. This can be used as a connection between classical web pages and the formats for semantic web; it joins these technologies. It uses the same documents as the humans, integrates information from sources and only provides the collected information for human with no deliberation.

In the paper, the implementation of the universal relation, which is useful for the integration process, was described. The main aim of the paper was to show how it is possible to perform the tuple storage into a repository more effectively. It consists in a data structure model creation, which contains all functional dependencies and affects the storage in a repository. The very important effective functional dependency testing was also described.

The paper also includes ideas on the described relational data model estimation, shows problems and limitation of this approach. The reason for the described method usage is given in the experimental part.

The ideas presented in the paper were implemented, the solution has been tested on real web sources in the CSV format and the results were included.

In the future, the automatic generation of the synonymous attributes in different sources is planned to be added. It will start the integration process needed for a better global model estimation. The next work plan is to create a tool finding parts of related information in complex web presentations and automatically adding them into a repository. The knowledge stored in a repository will increase, which may require the distribution of the problem.

Acknowledgment

The work was supported by the project 1ET100300419 of the Program Information Society (of the Thematic Program II of the National Research Program of the Czech Republic) "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation" and partly by the Institutional Research Plan AV0Z10300504 "Computer Science for the Information Society: Models, Algorithms, Applications".

References

[1] A.A.Barfourosh, M.L. Anderson, H.R.M.Nezbad, D Perlis "Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition" http://citeseer.ist.psu.edu/barfourosh02information.html [online]. 2002.

[2] S.M.Kuck, Y. Sagiv "A Universal Relation Database System Implemented Via the Network Model", in *Symposium on Principles of Database Systems*, pp. 147–157. 1982.

[3] D.Bednarek, D.Obdrzalek, J.Yaghob, F. Zavoral "Access Rights Definition and Management in an Information System based on a DataPile Structure" in *ITAT 2004, Workshop on Information Technologies - Application and Theory*, 2004.

[4] G. Ausiello, A. D'Atri, M. Moscarini, "Chordality Properties on Graphs and Minimal Conceptual Connections in Semantic Data Models", in *Symposium on Principles of Database Systems*, pp. 164–170. 1985.

[5] Bruno T. Messmer, Horst Bunke "Efficient Subgraph Isomorphism Detection: A Decomposition Approach", in *IEEE Transactions on Knowledge and Data Engineering*. pp.: 307-323. 2000.

[6] P.A. Bernstein, J.R. Swenson, D.C. Tsichristzis, "A Unified Approach to Functional Dependecies and Relations", in *SigMod*, pp. 237–245, 1975.

[7] J. Biskup, U. Dayal, P.A. Bernstein, "Synthesizing Independend Database Schemas", in *SigMod*, pp. 143–150, 1979.

[8] F. Cuppens, K. Yazdanian "A Natural Decomposition of Multi-level Relations", in *IEEE Symposium on Security and Privacy*, pp. 273-284. 1992.

[9] G. Grahme, K. Räihä, "Database Decomposition into Fourth Normal Form", in *Conference on Very Large Databases*, pp. 186–196, 1983.

[10] B.N. Shamkant, R. Minyoung, "Vertical Partitioning for Database Design – a Graphical Algorithm", in *SigMod*, pp. 440–450, 1989.

[11] L. Bellatreche, K. Karlapalem, A. Simonet, "Algorithms and Support for Horizontal Class Partitioning in Obect–Oriented Databases", in *Distributed and Parallel Databases, 8*, Kluwer Academic Publisher. pp. 115–179, 2000.

[12] D. Rosaci, G. Terracina, D. Ursino, "A Framework for Abstracting Data Sources Having Heterogennous Representation Formats", in *Data & Knowledge Engineering*, vol. 48, pp. 1–38. 2004.

[13] "Swoogle", *http://www.swoogle.org [online]*.

[14] "Froogle", *http://froogle.google.com [online]*.

[15] "Postgres", *http://www.postgresql.com [online]*.

[16] "Ceska narodni banka", *http://wdb.cnb.cz/CNB_TXT/KURZY.K_CURRTXT [online]*.