Rich Schemata for Semistructured Data: Thesis proposal

André Bergholz*†
Humboldt-University Berlin
bergholz@dbis.informatik.hu-berlin.de

Abstract

Semistructured data is one of the new challenging research areas in the database community. We believe that the underlying problem is that of moving from content-based to structure-based querying. For the management of semistructured data we argue in favor of a modular approach based on schema matching. Based on a graph data model we introduce schemata covering predicates, variables and paths. On top we propose three kinds of queries, namely schema, focus and transformation queries. We show how schema matching can be performed without any additional knowledge by reducing the problem to the well-studied Constraint Satisfaction Problem. Finally we give preliminary ideas about how query processing can benefit from already existing schema information.

1 Introduction

Traditional database management requires design and ensures declarativity. With the growing amount of data, especially on the Internet, we are faced with data that is not well-designed but has rather little structure. This has led to the research area of semistructured data. Abiteboul calls it "data that is neither raw data nor strictly typed" [Abi97]. Examples for semistructured data include HTML files, BibTEXfiles or genome data. They share the following properties:

1. The structure may be irregular. The data material may be over- or underspecified in relationship to the schema.

*Contact author: André Bergholz, LFE Datenbanken und Informationssysteme, Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany (phone: +49-30 2093 3024)

†This research was supported by the German Research Society, Berlin-Brandenburg Graduate School in Distributed Information Systems (DFG grant no. GRK 316)

2. The structure may vary. Over a discourse world the granularity of the structure depends on the view.

3. The structure of the data may be implicit. The structure becomes clear only after analyzing the actual value of the data. In particular, structured text such as HTML or BibTeX files contain structural information that is implicit.

Additionally, Buneman remarks that some data really is unstructured, e. g. because the possibly existing structure is not discovered yet [Bun97]. To us, the key problem is to move from *content-based* querying, i. e. the UNIX `grep`-command or simple WWW search engines, to *structure-based* querying, i. e. querying in an SQL-like manner. Of course, many other problems arise too, such as new aspects in query processing or the integration of technologies such as XML.

In our approach we want to stress that the notion of schema is the most crucial one of all. A schema is something that describes other things. All other notions can be organized around it. Informally, a query is a tuple consisting of what we want (i. e. a schema) and how we want it. A view can be seen as a named schema (or maybe a named query).

If schema information is so important, then how can we get some? One possibility, the one used in traditional database management, is to design a database. A designer introduces an entity `Person` with attributes `name`, `surname` and so on. This is, as mentioned before, rarely possible in the semi- and unstructured world. How about another idea. Given just a bag of instances, there is initially not necessarily any schema known. The user poses the query "Give me anything that has a name and a surname." Let's assume this query can be answered and let's assume that this query can be expressed purely as a schema, then we gain the information that some objects conform to a schema. Although this is not exactly declarativity, because we don't know whether anything with a name and a surname is a person (maybe it could also be a pet?), but it's something that can help users to get an idea about the content of a database. Moreover, schema information is, of course, of benefit for query optimization. Our approach shall be guided by the following three principles:

1. Answering a query *works without* schema information.

2. Answering a query *benefits from* schema information.

3. Answering a query *induces new* schema information.

2 Related areas

Related work comes from at least the following areas: semistructured data including query languages and optimization, graph transformations and constraint satisfaction.

Projects in semistructured data were often originated in conjunction with information integration projects. This is due to the fact that often a low-level, purely syntactical integration model is needed. A good example is the Lore project [MAG$^+$97] which now aims at storing and querying semistructured data. It originated from the TSIMMIS project [CGMH$^+$94] that provided tools for integration of heterogeneous data. Both projects use a simple and flexible data model, the Object Exchange Model (OEM) [PGMW95], to represent the data, although Lore now starts to use XML. In OEM all objects are self-describing, there is initially no need for classes or schemata. For this model a query language named Lorel ("Lightweight Object Repository Language") [QRS$^+$95] has been developed. Syntactically, Lorel is similar to SQL/OQL. It supports simple queries, boolean connectors, subqueries and label markers to distinguish prefixes in paths. The semistructured flavor comes through by the introduction of general path expressions. They serve two purposes: One can use wildcards for paths, and one can define regular expressions over the labels. In a next step some schema information in the form of dynamically created and maintained DataGuides [GW97] is introduced. This helps the user to get a better view on the data, e. g. for query formulation. Additionally, DataGuides are useful for optimization, they can serve as an index.

A similar, somewhat more theoretical project is the University of Pennsylvania's UnQL project. A query language UnQL ("Unstructured Query Language") [BDHS96] has been developed. Edge-labeled trees are used as the data model. The query language UnQL has a similar functionality as Lorel. As query results edge-labeled trees and labels can occur. One interesting aspect is the explicit treatment of restructurings of a database by introducing the `traverse`-command. Again, post-defined schemata for the purpose of optimization are introduced [BDFS97].

Much related work arises in the context of information extraction from the WWW. A main focus lies on query languages suited for the Web ([KS95], [MMM96], [LSS96], [AV97]). A very fundamental work on data stored in files is the one by Abiteboul, Cluet and Milo [ACM93]. Structured files are transformed to databases so that file querying and manipulating by using database technology becomes possible.

Graph transformations address the dynamic aspects of graphs. Systems are typically rule-based and can be used to model behavior. Two popular systems are PROGRES [Sch97] and AGG [AGG]. Particularly interesting from our point of view is how the matching of left-hand side graphs into the target graph is performed. PROGRES uses a database-like approach [Zue93] whereas AGG uses a constraint-based approach [Rud98].

Constraint Satisfaction Problems form a general class of search problems for which many techniques and heuristics exist. [Bar98] and [Kum92] provide an introduction to the field. They give various algorithms, heuristics and useful background information to efficiently solve CSPs.

3 Querying based on schema matching

Guided by the ideas presented in the introduction we propose a query language based on schema matching. We use labeled graphs, "the unifying idea in semistructured data" [Bun97], as the underlying syntax. Let $\mathcal{L}$ be an arbitrary set of *labels*. A tuple $G = (V, A, s, t, l)$ is a *($\mathcal{L}$-)labeled directed graph* if (V, A, s, t) is a total directed graph with a set of vertices V, a set of arcs A and two total functions s and t assigning each arc its source and target vertex. Furthermore, $l : V \cup A \longrightarrow \mathcal{L}$ is a total *label function* assigning each vertex and arc a label from $\mathcal{L}$.

In Figure 1 we present an example that we will use throughout the paper. It shows a semistructured database about persons having names, surnames, a year of birth, a profession etc. Furthermore, a sibling relationship is illustrated.

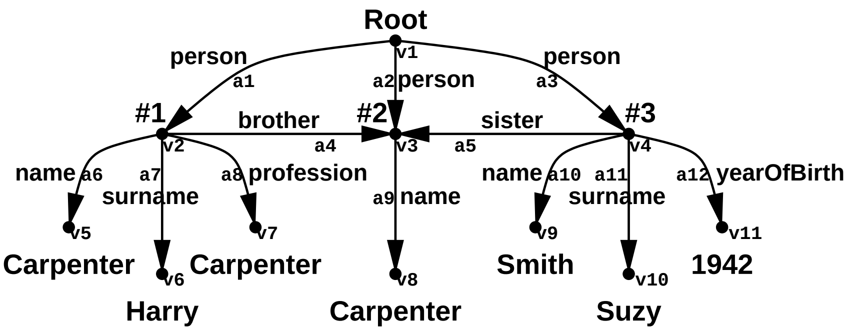

Figure 1: A labeled directed graph

Let's start with a simple notion of a schema and conformity between schema and object. Informally, a schema is an object that describes a set of objects. In the simpler syntactic framework of the label world the schema concept certainly exists as well. One label can describe a set of other labels. This is frequently done, data types, predicates and regular expressions are examples.

Now the first idea for schemata in the graph world would be to assign schemata from the label world to the elements of the graph. We choose predicates to be the label world schemata. Given a set of unary predicates $\mathcal{P}$, a *predicate schema* is an object where the elements are labeled with predicates from $\mathcal{P}$.

We give an example in Figure 2. Note that we treat a quoted constant c as an abbreviation for the predicate $X = c$.

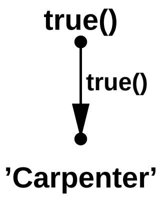

Figure 2: A simple predicate schema

In order to establish a relationship between the schema and the objects described by it we have to establish the notion of *conformity* between schema and object. In this simple case we say that an object o conforms to a schema s if there is an isomorphic embedding from s into o respecting the predicates.

In Figure 3 we show the same schema as in Figure 2 but together with its minimal matches in the database from Figure 1.

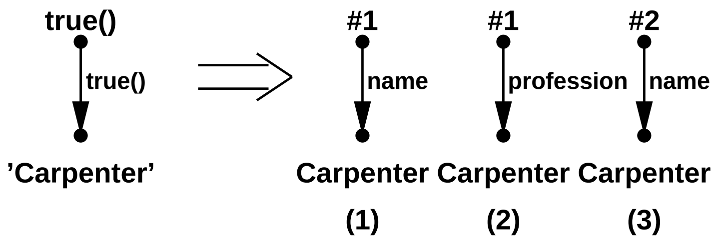

Figure 3: The predicate schema and its minimal matches

Adding variables is in the flavor of a join operation. We give an idea of what is meant by providing an example (see Figure 4).

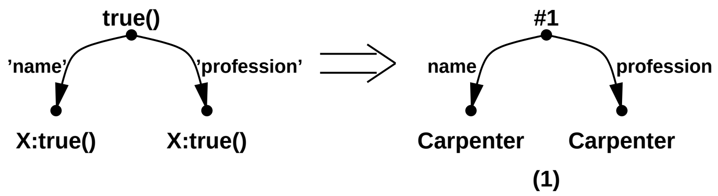

Figure 4: Adding variables

Paths are an important concept in semistructured data. This is due to the fact that sometimes information is represented at different levels of granularity. We demonstrate the idea again by presenting an example (see Figure 5).

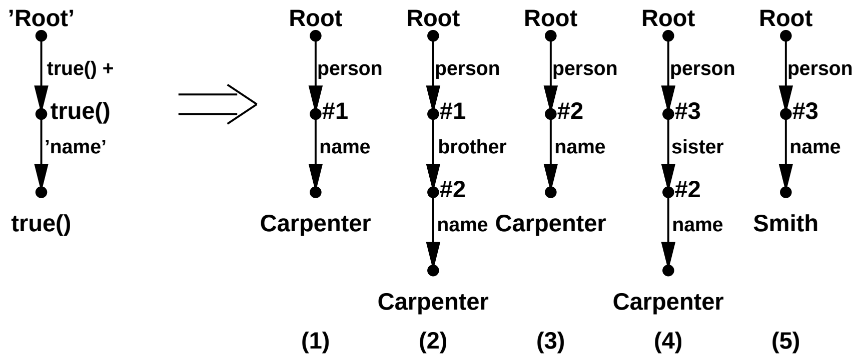

Figure 5: Adding paths

A schema in itself already forms the most simple kind of query. It queries all subobjects of a database that conform to it. A *schema query* is a tuple $q = (s)$ where s is a schema. The *answer* to q with respect to a database o is the set of minimal matches of s in o.

As an example you can imagine any of the schemata from the previous sections. With a schema we can formulate conditions. This roughly corresponds to the selection in the relational world. However, we would like to have something that is comparable to the projection. A *focus query* is a tuple $q = (s_1, s_2)$ where s_1 is a schema and s_2 is a subobject of s_1. The *answer* to q with respect to a database o is the union of the minimal matches of s_2 over all minimal matches of s_1 in o. The example in Figure 6 queries for the surnames of all persons with the name 'Carpenter'. The subschema s_2 is indicated by circles around the elements (in this case just one node) of the superschema s_1.

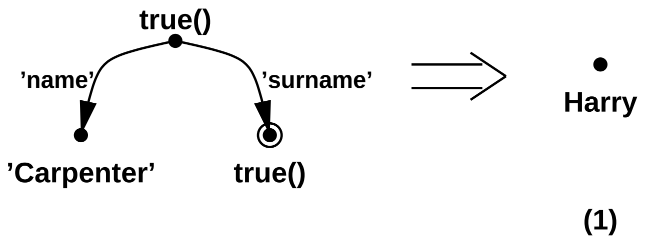

Figure 6: A focus query

However, sometimes we want to restructure the answer completely. Therefore we introduce the *transformation query* where you can give a graph structure that you prefer and compute new labels by using terms over the old ones. A *transformation query* is a tuple $q = (s, t)$ where s is a schema and t is an object labeled with terms over the elements in s. The *answer* to q is built by creating for every match of s in o a new object isomorphic to t, labeled with the evaluated terms of t instantiating the terms by using the match. The example in Figure 7 queries for the age of Suzy Smith. The age is computed from the year of birth. Please note, that schema and focus queries can be

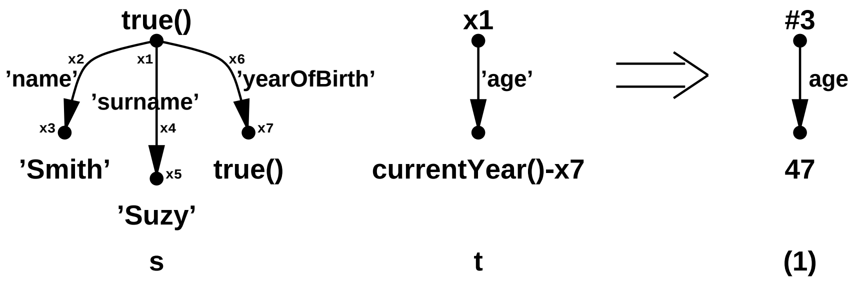

Figure 7: A transformation query

expressed as transformation queries.

Initial ideas of this work have also been published in [BF98].

4 Answering queries by utilizing Constraint Satisfaction techniques

We reduce this problem to the Constraint Satisfaction Problem. The reason for this is twofold. First, the area of constraint satisfaction is well-studied and thus, there exist quite a few algorithms and heuristics for solving these problems. Second, the solution is quite general and we can thus in the future integrate richer schemata without too many complications.

Constraint satisfaction deals with solving problems by stating properties or constraints that any solution must fulfill. A *Constraint Satisfaction Problem (CSP)* is a tuple (X, D, C) where

- X is a set of *variables* $\{x_1, \ldots, x_m\}$,
- D is a set of finite *domains* D_i for each variable $x_i \in X$ and
- C is a set of *constraints* $\{C_{S_1}, \ldots, C_{S_n}\}$ restricting the domains of the variables. Here the $S_i = (x_{S_{i_1}}, \ldots, x_{S_{i_k}})$ are arbitrary tuples of variables from X and each C_{S_i} is a relation over the crossproduct of the domains of these variables ($C_{S_i} \subseteq D_{S_{i_1}} \times \cdots \times D_{S_{i_k}}$).

Solving a CSP is finding assignments of values from the respective domains to the variables so that all constraints are satisfied. We are usually interested in finding all solutions of a CSP.

The basic idea is as follows and can be depicted from Figure 8. The database graph is transformed into suitable domains and variables are introduced for the elements in the schema. Furthermore, constraints representing the match semantics are introduced. They can be categorized into the ones that represent the label part and the ones that represent the structural part of the match semantics.

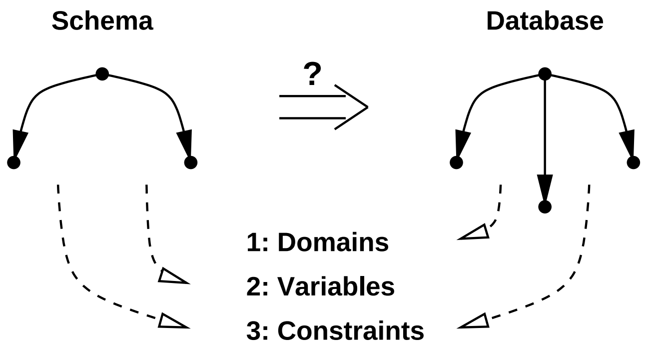

Figure 8: Schema matching by Constraint Satisfaction

We depict the domains of the vertices and arcs from the database graph in Figure 1.

$$D_V = \{v_1, v_2, v_3, \ldots, v_{11}\}$$
$$D_A = \{a_1, a_2, a_3, \ldots, a_{12}\}$$

The example schema in Figure 9 (the same as in Figure 2) gives us the variables and the domain assignments.

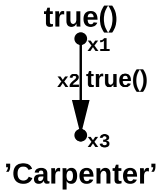

Figure 9: A simple predicate schema

$$\begin{aligned} X &= \{x_1, x_2, x_3\} \\ D_1 = D_3 &= D_V \\ D_2 &= D_A \end{aligned}$$

Constraints are derived from the labels in the schema ...

$$\begin{aligned} C^{lab}_{(x_1)} &= \{(v_1), (v_2), (v_3), \dots, (v_{11})\} \\ C^{lab}_{(x_2)} &= \{(a_1), (a_2), (a_3), \dots, (a_{12})\} \\ C^{lab}_{(x_3)} &= \{(v_5), (v_7), (v_8)\} \end{aligned}$$

... and the structure of the schema.

$$\begin{aligned} C^{src}_{(x_2,x_1)} = \{&(a_1, v_1), (a_2, v_1), (a_3, v_1), (a_4, v_2), (a_5, v_4), (a_6, v_2), \\ &(a_7, v_2), (a_8, v_2), (a_9, v_3), (a_{10}, v_4), (a_{11}, v_4), (a_{12}, v_4)\} \\ C^{tar}_{(x_2,x_3)} = \{&(a_1, v_2), (a_2, v_3), (a_3, v_4), (a_4, v_3), (a_5, v_3), (a_6, v_5), \\ &(a_7, v_6), (a_8, v_7), (a_9, v_8), (a_{10}, v_9), (a_{11}, v_{10}), (a_{12}, v_{11})\} \end{aligned}$$

Our sample CSP has the solutions (v_2, a_6, v_5), (v_2, a_8, v_7) and (v_3, a_9, v_8) for the variables (x_1, x_2, x_3). They correspond to the matches of the schema as depicted in Figure 3. Please note that if you want to ensure injectivity of the match, additional constraints must be introduced.

More details about this part of the work (e. g. variables and paths) and about techniques for solving CSPs can be found in [BF99].

5 Using schemata for optimization

In the previous section we show how to process a query without any schema information given. In this section we want to give rather preliminary ideas about how to make use of already present schema information for query optimization. The general idea is that it is probably faster to match an incoming schema against the already present schemata rather than against the database itself. If schema containment occurs then the query processing may be speeded up. Of course, the whole process also depends on the number of schemata that the incoming one is matched against.

A schema s_1 contains a schema s_2 if for all databases o all matches of s_2 are also matches of s_1. If s_1 contains s_2

1. All matches of s_2 are also matches of s_1. If we want to find the matches of s_1 and have already the ones for s_2 we can present those without further thinking as matches of s_1. There may be more matches for s_1, though.

2. Matches of s_2 can only be found among the matches of s_1. If we want to find the matches of s_2 and have already the ones for s_1 we don't need to consider the rest of the database.

In Figure 10 we see three schemata. They contain one another from left to right.

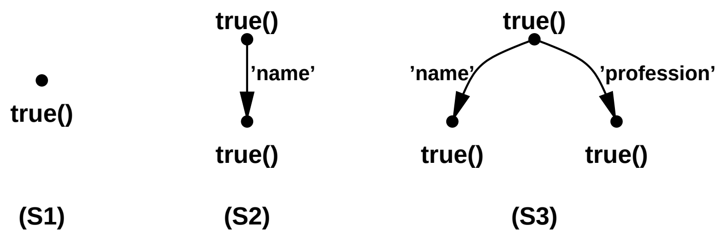

Figure 10: Schema containment

Let's assume we have the notion of containment for predicates, too. p_1 contains p_2 if for all labels x the implication $p_2(x) \longrightarrow p_1(x)$ holds. Informally, a schema s_1 contains another schema s_2 if s_1 is a subgraph of s_2 and the predicates of s_1 contain the respective predicates of s_2 and the paths in s_1 are no longer than those in s_2. The other direction of this implication does not hold.

We want to reduce the problem of testing schema containment again to the Constraint Satisfaction Problem. Furthermore, we want to find "good" schemata to materialize. Materializing "small" schemata that contain everything is probably not a good idea. Vice versa, materializing "large" specific schemata is probably not a good idea as well, since testing for containment will be almost as expensive as finding schema matches in the database directly. Thus, we are looking for criteria to assess schemata. These criterion should probably be based on the number of matches for a schema. Initial ideas are presented in the table below. We investigate some criteria, starting with the number of minimal matches. It seems that "small" schemata will always have more matches than larger ones. So we multiply the number of matches with the size of the schema. To say something about the selectivity of the labels we also give the number of structural matches, which is the number of matches not considering any labels. Finally we divide the latter two. A summary of these criteria applied to the schemata in Figure 10 can be found in Table 1. Which of the criteria are useful and what boundaries should be considered as "good" will be investigated in the future.

Assessment criterion	S_1	S_2	S_3
# of (minimal) Matches	11	3	1
# of Matches × Size	11	9	5
# of Structural matches	11	12	30
Matches × Size / Structural matches	1	3/4	1/6

Table 1: Ideas for assessing the usefulness of a schema

6 Conclusion

In this paper we present the ideas that have been developed so far in our project about the management of semistructured data. In general, semistructured data occurs when little effort is put into the organization of the data. In contrast to other approaches we argue in favor of queries based on schema matching. We feel that this approach is more natural and modular and gives us the potential to extract schema information from user queries.

We use labeled graphs as the data model in our approach. Within this syntactic framework we define schemata covering predicates, variables and paths. Based on these schemata we introduce the queries. Schema queries return the instances conforming to a schema. A focus query is similar to the relational projection operation. With a transformation query the answer can be built in accordance with the users wishes.

We show how matches for schemata can be found in a database graph by applying techniques from the constraint satisfaction area. We took this approach because there are already many optimization techniques known for these types of problems and because it can probably easily be adapted for a richer schema semantics in the future.

Finally we present preliminary ideas about utilizing schema containment for query optimization. If a schema contains another one then all matches of the second one are also matches of the first one or, in other words, matches of the second one can only be found among the matches of first one. The underlying idea is, of course, to match any incoming query first against already existing schemata in order to speed of the query processing.

Future work lies first and foremost in a prototype implementation of our ideas. We plan to use the constraint solver ECLiPSe [Ecl] as the basis of our implementation. Furthermore, details about the management of the schemata, i. e. about how they can be used for query processing and about assessing the usefulness of schemata, are still unclear.

Acknowledgement

The author wishes to thank all members of the Berlin-Brandenburg Graduate School in Distributed Information Systems, in particular Johann Christoph Freytag and Felix Naumann, for many interesting discussions and useful comments.

References

[Abi97] S. Abiteboul. Querying semi-structured data. In *Proceeding of the International Conference on Database Theory*, 1997.

[ACM93] S. Abiteboul, S. Cluet, and T. Milo. Querying the file. In *Proceedings of the VLDB Conference*, 1993.

[AGG] The AGG-Homepage, `http://tfs.cs.tu-berlin.de/projekte/gragra/agg/`.

[AV97] S. Abiteboul and V. Vianu. Queries and computation on the web. In *Proceedings of the International Conference on Database Theory*, 1997.

[Bar98] R. Bartak. Guide to constraint programming, 1998. `http://kti.ms.mff.cuni.cz/ ~bartak/constraints/`.

[BDFS97] P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory*, 1997.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[BF98] A. Bergholz and J. C. Freytag. A three-layer approach to semistructured data. In *6th International Workshop on Theory an Application of Graph Transformations (TAGT)*, November 1998.

[BF99] A. Bergholz and J. C. Freytag. Matching schemata by utilizing constraint satisfaction techniques. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats (in conjunction with ICDT'99)*, January 1999.

[Bun97] P. Buneman. Semistructured data. In *Proceedings of the PODS Conference*, 1997.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogenous information sources. In *Proceedings of the Information Processing Society of Japan (IPSJ) Conference*, pages 7–18, October 1994.

[Ecl] ECLiPSe - The ECRC Constraint Logic Parallel System, `http://www.icparc.ic.ac.uk/eclipse/`.

[GW97] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the VLDB Conference*, August 1997.

[KS95] D. Konopnicki and O. Shmueli. W3QS: A query system for the World-Wide Web. In *Proceedings of the VLDB Conference*, pages 54–65, 1995.

[Kum92] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[LSS96] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. A declarative language for querying and restructuring the web. In *Proceedings of the 6th International Workshop on Research Issues in Data Engineering (RIDE)*, February 1996.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

[MMM96] A. O. Mendelzon, G. A. Mihaila, and T. Milo. Querying the World-Wide Web. In *Proceedings of the Conference on Parallel and Distributed Information Systems (PDIS)*, 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogenous information sources. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 251–260, March 1995.

[QRS+95] D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. Querying semistructured heterogeneous information. In *Proceedings of the International Conference on Deductive and Object-Oriented Databases*, pages 319–344, 1995.

[Rud98] M. Rudolf. Utilizing constraint satisfaction techniques for efficient graph pattern matching. In *6th International Workshop on Theory an Application of Graph Transformations (TAGT)*, November 1998.

[Sch97] A. Schuerr. Programmed graph replacement systems. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations*, volume 1. World Scientific, 1997.

[Zue93] A. Zuendorf. A heuristic for the subgraph isomorphism problem in executing PROGRES. Technical Report AIB 93-5, RWTH Aachen, 1993.