Matching Schemata by utilizing Constraint Satisfaction Techniques

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

Johann Christoph Freytag
Humboldt-University Berlin
freytag@dbis.informatik.hu-berlin.de

Abstract

Semistructured data occurs when little effort is put into the organization of data. We have proposed a new, flexible approach for querying semistructured data, where queries are based on schema matching. In this paper we reduce the problem of finding matches for a given schema in a database to the well-studied Constraint Satisfaction Problem. To this end, the database is transformed into the domains and the schema into the variables. Various constraints representing the match semantics are introduced. We also discuss methods, heuristics and optimization ideas for solving these problems.

1 Introduction

Semistructured data is one of the challenging phenomena of the WWW age. HTML or BibTEXfiles are typical examples. In general, structural and content meaning are mixed.

A major concern is the finding of schema information to represent structural meaning. Usually there is initially no complete structural description of the data available. Therefore, query languages are defined in a way that they work without schema information. Nevertheless, such information is useful, for instance for optimization, but also to give users a clue about the content of the database.

In [BF98] we have proposed a new approach to semistructured data. The three key points of this approach are that answering a query *works without*, *benefits from* and *induces new* schema information. To achieve this, we lift query capabilities to the schema level and propose more sophisticated sorts of schemata. The reason for this is that we want to reuse user queries to generate schema information. This follows the simple observation that an answered query delivers metainformation, namely that some objects are the answer to the given query.

We use a general labeled graph data model with no special restrictions. We introduced schemata covering predicates, variables and paths. Queries are based on schema matching. Here we introduced pattern, focus and transformation queries. The focus of this paper lies on the discovery of these schema matches in a database graph, i. e. on how answering a query works without schema information. We solve this problem by reducing it to the well-studied Constraint Satisfaction Problem and look briefly on methods for solving such problems.

2 Reducing schema matching to the Constraint Satisfaction Problem

Finding schema matches in our approach is related to the Subgraph Isomorphism Problem which is known to be NP-complete in the general case [GJ79]. Nevertheless, the search space that occurs can be greatly reduced. Constraint Satisfaction Problems form a general class of search problems for which many techniques and heuristics exist.

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)

*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)

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 procedure We will illustrate the process of transforming a schema matching problem into a CSP in detail for the simplest sort of schema, the predicate schema. We use the database graph in Figure 1 and the schema in Figure 2 as an example. We see a person database with varying attributes and ancestor relationships. The schema matches everything that has an attribute arc leading to a node labeled 'Carpenter'.

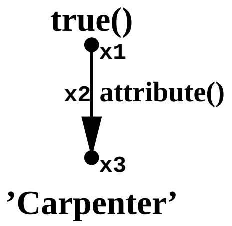

Figure 2: A simple Predicate Schema

The basic idea is the same as for the more sophisticated schema concepts. 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.

In general, let's assume we have a database object $o = (V^{(o)}, A^{(o)}, s^{(o)}, t^{(o)}, l^{(o)})$ and a schema $s = (V^{(s)}, A^{(s)}, s^{(s)}, t^{(s)}, l^{(s)})$. Here V and A denote the respective nodes and arcs, s and t denote the source and the target function for the arcs and l denotes the label function. The following transformation is performed:

1. From the database graph we deduce domains $D_V = V^{(o)}$ and $D_A = A^{(o)}$.

In our example (see Figure 1) we get:

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

2. A variable x_i is introduced for each element in $V^{(s)} \cup A^{(s)}$. As a shorthand, we use the variable as a synonym for the corresponding element and say e. g. that "x_i is an arc" instead of "x_i represents an arc". X is the set of these variables.
3. For each variable $x_i \in X$ we define $D_i = D_V$ if x_i is a node and $D_i = D_A$ if x_i is an arc. D is the set of the D_i.

From the schema graph (see Figure 2) we get:

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

4. For every variable x_i we introduce a unary constraint $C^{lab}_{(x_i)}$ ensuring the semantics of the predicates in the schema. A node or arc is in the constraint if its label makes the predicate of x_i true ($C^{lab}_{(x_i)} = \{(d_j) \in D_i | l^{(s)}(x_i)(l^{(o)}(d_j)) = true\}$).

In the example x_1 is labeled with the predicate $true()$, so there is no constraint on D_V and we could actually skip this constraint. The arc x_2 is labeled $attribute()$. We assume that we can somehow evaluate this predicate and furthermore, that it is in this example true for all the arc labels in the database graph. x_3 is labeled $'Carpenter'$ which is a shorthand for the predicate $X = Carpenter$. The nodes v_5, v_7 and v_8 in the database graph carry this label.

$$\begin{aligned} C^{lab}_{(x_1)} =& \{(d_j) \in D_1 | true(l^{(o)}(d_j))\} = D_V \\ C^{lab}_{(x_2)} =& \{(d_j) \in D_2 | attribute(l^{(o)}(d_j))\} = D_A \\ C^{lab}_{(x_3)} =& \{(d_j) \in D_3 | Carpenter = l^{(o)}(d_j)\} \\ =& \{v_5, v_7, v_8\} \end{aligned}$$

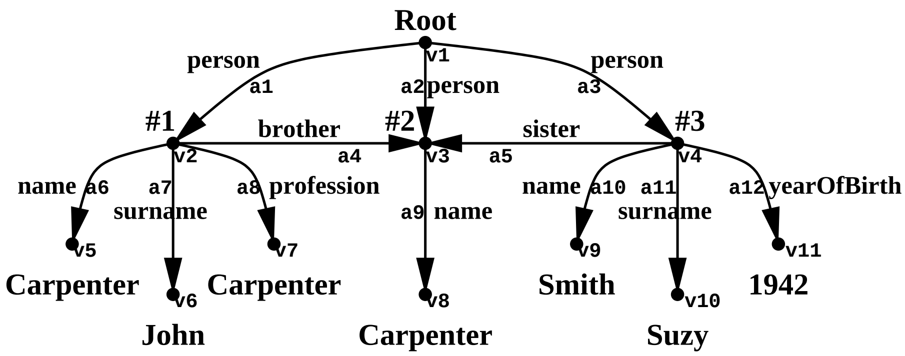

Figure 1: The sample database graph

5. For every variable $x_i \in A_S$ and its source and target node x_s and x_t we introduce two constraints $C^{src}_{(x_i,x_s)}$ and $C^{tar}_{(x_i,x_t)}$ that represent the structural behavior of the match (morphism property, $C^{src}_{(x_i,x_s)} = \{(d_j, d_v) \in D_i \times D_s | s^{(o)}(d_j) = d_v\}$, $C^{tar}_{(x_i,x_t)} = \{(d_j, d_v) \in D_i \times D_t | t^{(o)}(d_j) = d_v\}$).

We have only one arc in the schema graph, namely x_2. One should read the constraints as follows: "If x_2 is assigned to a_1 then x_1 has to be assigned to v_1" etc.

$$
\begin{aligned}
C^{src}_{(x_2,x_1)} =& \{(d_j, d_v) \in D_2 \times D_1 | s^{(o)}(d_j) = d_v\} \\
=& \{(a_1, v_1), (a_2, v_1), (a_3, v_1), (a_4, v_2), \\
& (a_5, v_4), (a_6, v_2), (a_7, v_2), (a_8, v_2), \\
& (a_9, v_3), (a_{10}, v_4), (a_{11}, v_4), (a_{12}, v_4)\} \\
C^{tar}_{(x_2,x_3)} =& \{(d_j, d_v) \in D_2 \times D_3 | t^{(o)}(d_j) = d_v\} \\
=& \{(a_1, v_2), (a_2, v_3), (a_3, v_4), (a_4, v_3), \\
& (a_5, v_3), (a_6, v_5), (a_7, v_6), (a_8, v_7), \\
& (a_9, v_8), (a_{10}, v_9), (a_{11}, v_{10}), (a_{12}, v_{11})\}
\end{aligned}
$$

6. C is the set of all introduced constraints.

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. Please note that if you want to ensure injectivity of the match, additional constraints must be introduced.

Variables and paths We briefly discuss variables and paths. The schema in Figure 3 matches every object where the name is the same as the profession. This is in the flavor of a join operation.

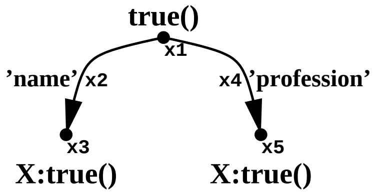

Figure 3: Adding variables

When constructing the corresponding CSP the only new thing is that we introduce an additional constraint $C^{var}_{(x_3,x_5)}$ linking the "related" variables x_3 and x_5 pairwise to each other.

$$
\begin{aligned}
C^{var}_{(x_3,x_5)} =& \{(d_i, d_j) \in D_3 \times D_5 | l^{(o)}(d_i) = l^{(o)}(d_j)\} \\
=& \{(v_1, v_1), (v_2, v_2), (v_3, v_3), (v_4, v_4), \\
& (v_5, v_5), (v_5, v_7), (v_5, v_8), (v_6, v_6), \\
& (v_7, v_5), (v_7, v_7), (v_7, v_8), (v_8, v_5), (v_8, v_7), \\
& (v_8, v_8), (v_9, v_9), (v_{10}, v_{10}), (v_{11}, v_{11})\}
\end{aligned}
$$

The schema in Figure 4 includes the arc x_2 that is supposed to match a path in the database. The schema matches everything that emanates from the root node via a path of arbitrary length and ends with an arc labeled $'name'$. It roughly corresponds to a `select *.name` statement in the Lore [MAG+97] language.

The first thing that we do when transforming this into a CSP is introducing a new domain, namely the domain of the *trails*. A trail is a walk where all the arcs are pairwise distinct. This ensures that the domain is always finite. Although strictly speaking not correct we use the term *path* instead of trail. For the database in Figure 1 we get:

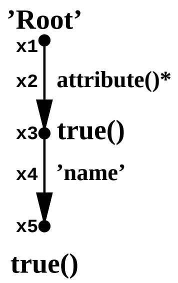

Figure 4: Adding paths

$$
\begin{aligned}
D_P = \{ & \epsilon, (a_1), (a_2), (a_3), (a_4), (a_5), (a_6), (a_7), \\
& (a_8), (a_9), (a_{10}), (a_{11}), (a_{12}), \\
& (a_1, a_4), (a_1, a_6), (a_1, a_7), (a_1, a_8), (a_2, a_9), \\
& (a_3, a_5), (a_3, a_{10}), (a_3, a_{11}), (a_3, a_{12}), \\
& (a_4, a_9), (a_5, a_9), (a_1, a_4, a_9), (a_3, a_5, a_9)\}
\end{aligned}
$$

where ϵ denotes the empty path. This domain is assigned to the variable x_2. The respective constraints have to be adapted. In the constraint $C^{lab}_{(x_2)}$ we must represent the fact that a path is in the constraint only if the predicate from x_2 is true for the labels of all arcs in the path. Source and target of nonempty paths can naturally be defined as the source of the first and the target of the last arc. For the empty path we only require that the source and the target node are the same. Thus, we represent the structural requirements in one ternary constraint rather than in two binary ones.

$$
\begin{aligned}
C^{src,tar}_{(x_2,x_1,x_3)} = & \{(d_j, d_s, d_t) \in D_2 \times D_1 \times D_3 | \\
& (d_j \neq \epsilon \wedge \\
& \quad s^{(o)}(d_j) = d_s \wedge t^{(o)}(d_j) = d_t) \\
& \vee (d_j = \epsilon \wedge d_s = d_t)\} \\
= & \{(\epsilon, v_1, v_1), (\epsilon, v_2, v_2), \ldots, \\
& ((a_1), v_1, v_2), ((a_2), v_1, v_3), \ldots, \\
& ((a_1, a_4), v_1, v_3), ((a_1, a_6), v_1, v_5), \\
& \ldots, ((a_3, a_5, a_9), v_1, v_8)\}
\end{aligned}
$$

Additionally we can put constraints on the minimum and maximum length of paths. This is, however, not necessary for our example, so we do not discuss this here.

3 Efficiently solving CSPs

A CSP is typically an exponential problem. Suppose we have a given CSP with m variables and assume for reason of simplicity that all underlying domains have the same size n. Then there exist n^m possible assignments for the variables. In our case, however, the news is not that bad because the number m of variables is determined by the schema and thus typically not too large.

Figure 5 shows a typical search space of a CSP. There are the variables with their respective domains and the constraints linking variables indicated on the left.

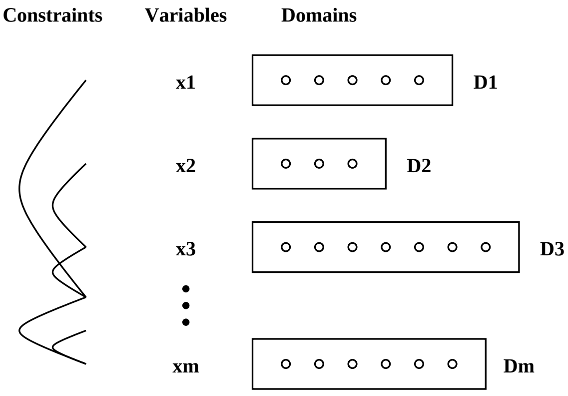

Figure 5: The search space of a CSP

When solving a CSP there are more things to consider then the actual search algorithm. Let's assume we use a simple Backtracking algorithm. An initially empty assignment is expanded by assigning one variable at a time. In each step the validity of the constraints is tested. It is now possible to reduce the search space. When a contradictory partial assignment is found, we usually get rid of an exponential subproblem by not needing to find assignments for the remaining uninstantiated variables. This principle leads to the "First-Fail" heuristics. The aim is to produce a partial contradictory assignment as fast as possible. This is a particularly good heuristics when as in our case all solutions to the problem have to be found and thus the whole search space has to be considered. An important role in this heuristics play the ordering in which the variables get instantiated (the so called *variable ordering*). On one hand we want to fail as fast as possible. Thus, it is good to consider variables with many and "strong" constraints first. On the other hand once a fail is achieved we want to get rid of an as big as possible part of the search space. Thus, it is good to consider variables with small domains first.

In our prototypical implementation we are currently investigating four optimization ideas:

1. Ensure injectivity of solution not by constraints but by a separate test (both, naive and hash-based).

2. Use the label constraints as domains (static domain reduction). This is analogous to the CSP technique of node consistency, that tries reduce domains by eliminating values that violate unary constraints.

3. Sort the variables (both, by increasing domain size and by decreasing number of constraints).

4. Perform dynamic domain reduction. If a variable representing an arc is instantiated, it is immediately clear that the variables corresponding to the source and the target node of this variable can only have one specific value. Vive versa, if a variable representing a node is instantiated, the domains of the variables corresponding to the incoming and outgoing arcs can also be reduced substantially. Note that this is different from the idea of Lookahead, since we do not reduce the domains by going through the constraints but rather by explicitly computing reduced domains.

Preliminary results suggest that one is not a good idea, two and three are, especially combined together, and four may be a good idea, if used only from a certain domain size and from a certain number of already instantiated variables on.

4 Related work

Core issues concerning semistructured data are discussed in [Abi97] and [Bun97]. Query languages include Lorel [MAG$^+$97] and UnQL [BDHS96]. In [GW97] the creation and maintenance of a DataGuide, an ad hoc complete schema for a semistructured database, is described. Another notion of schema is introduced in [BDFS97].

Graph pattern matching plays an important role in the area of graph transformation. In [Rud98] constraint satisfaction techniques are used for simple graph pattern matching. A more database-like approach to this problem can be found in [Zue93].

For an introduction to the field of constraint satisfaction you can refer e. g. to [Bar98] or [Kum92]. They give various algorithms, heuristics and useful background information to efficiently solve CSPs.

5 Conclusion

In this paper we have presented how to find matches for previously introduced schemata in a database graph. In our approach this schema matching forms the base for query processing. We solve the problem by reducing it to the Constraint Satisfaction Problem. To this end, we transform the database graph into domains and the schema graph into variables. We introduce constraints representing the label and the structural part of the match semantics. Finally, we briefly discuss methods, heuristics and optimization ideas for solving these kinds of CSPs. We believe that the advantages of our approach lie in its generality, richer schemata can be integrated without too many complications, and in the fact that the area of Constraint Satisfaction is a well-studied one where many algorithms and heuristics exist. Our approach is reasonably efficient since the search space grows exponential in the size of the schema but only linear in the size of the database.

Future work lies on how to test schema containment in order to make use of it for optimization. We imagine that schema information can come not only from a database designer but can also be extracted from already answered queries. In this context we need to develop some criterion for measuring the quality of a schema.

References

[Abi97] S. Abiteboul. Querying semi-structured data. In *Proceeding 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.

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

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

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

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

[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.

[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.

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