## Defining the XML Schema Matching Problem

for a Personal Schema Based Query Answering System

Marko Smiljanić, Maurice van Keulen, and Willem Jonker

### Contents

1 Introduction 2

2 Querying XML-Web using a personal schema 2
   2.1 The concepts personal schema and personal query 2
   2.2 The personal schema based query answering system 4
   2.3 Analogy between Google and the personal schema based query answering system 5
   2.4 Issues in a personal schema based query answering system 6

3 A framework for defining a schema matching problem 6
   3.1 Modeling XML schemas 7
      3.1.1 Features of the XML schema language 7
      3.1.2 Graph theory basics 7
      3.1.3 XML schema graph 9
   3.2 Generic matching problem is a constraint optimization problem 11
      3.2.1 Constraint programming basics 11
      3.2.2 Informal definition of a generic matching problem 12
      3.2.3 The definition of a matching problem as a COP 14
   3.3 Automated semantic matching 15

4 Definition of the XML schema matching problem 17
   4.1 Design of XML schemas 17
   4.2 Components of the semantic XML schema matching problem 18
   4.3 XML schema matching problem as a constraint optimization problem 19
   4.4 Additional constraints in the schema matching problem definition 21
   4.5 Search space size for an XML schema matching problem 21

5 Defining the objective function 22
   5.1 Hints 22
      5.1.1 Aspects of a hint 22
      5.1.2 Types of hint functions 23
      5.1.3 The use of properties in hint functions 24
   5.2 Composing hints 25
      5.2.1 Numerical hint functions only 25
      5.2.2 Examples of arithmetic approaches to hint composition 25
   5.3 Evaluation of semantic matching 26

6 Related research 27

7 Conclusion and future work 28
1 Introduction

XML brought several important qualities to data representation. Through the usage of tags, it combined schema and data information. Tag nesting enabled a simple representation of hierarchical relations. Such enrichments sparked off a new wave of research on how to improve querying and searching of data within XML documents.

The Internet is practically an endless collection of data being used simultaneously by millions of users. We expect that for a large part, this information will become available in XML. As such, it is a valuable source of information for other users, who need information finding services to guide them through the wealth of information.

In this report, we investigate a specific information finding approach – personal schema querying. The target environment for this approach is the XML-Web – an Internet based collection of XML data sources. Each data source in the XML-Web allows for querying of its data and access to an XML schema of that data.

In personal schema querying, users need not know the structure of XML-Web data. For querying, they use a self-defined XML schema, as their personal model of the ‘universe of discourse’. This personal model is, in a sense, imposed on the XML-Web instead of the other way around. Obviously, a personal XML schema is not likely to be the same as any of the schemas of the XML-Web’s data sources. Therefore, finding data corresponding to the personal schema requires it to be matched against the schemas of the data sources. This process is called XML schema matching.

In this report, we analyze the problem of personal schema matching. We define the ingredients of the XML schema matching problem using constraint logic programming. This allows us to thoroughly investigate specific matching problems. We do not have the ambition to provide for a formalism that covers all kinds of schema matching problems. The target is specifically personal schema matching using XML.

The report is organized as follows. Chapter 2 provides a detailed description of our research domain - the Personal Schema Query Answering System. In chapter 3, we introduce a framework for defining the XML schema matching problem. The XML schema matching problem is defined using this framework in chapter 4. An important component of the XML schema matching problem is the objective function, which is investigated in chapter 5. Chapter 6 presents the related research, with conclusions and further research being discussed in chapter 7.

Throughout the report, we use expressions like ‘schema matching’, ‘XML schema matching’ and ‘semantic XML schema matching’. Unless explicitly stated otherwise or strongly suggested by the context of the story, those expressions all refer to the same thing: semantic matching of XML schemas as used in personal schema querying. Furthermore, basic knowledge of the XML-schema language is assumed.

2 Querying XML-Web using a personal schema

In the following sections, the ideas behind personal schema based querying are introduced. Section 2.1 explains the concepts personal schema and personal query. Section 2.2 then describes a personal schema based query answering system (i.e., PSQ) and how a user interacts with this system. Section 2.3 compares PSQ with Google to illustrate PSQ’s novel approach. Chapter ends with section 2.4 that indicates some of the research issues related to schema matching in PSQ.

2.1 The concepts personal schema and personal query

The usual approach to exposing a large collection of heterogeneous distributed data to users is by defining an abstract view (see Fig. 1). The abstract view is used to hide the complexity of heterogeneity in the distributed data. Users benefit by being able to understand and learn to use the data without having to bother with the intricate details of each individual data source, which would be required if the data sources were accessed directly.
To support an abstract view, however, a mediator is needed. A mediator is a component that maps concrete data of many data sources to one uniform form, the abstract view. For example, a price comparison website\(^1\) concentrates product-related data from a large number websites, exposing it through one abstract view.

The concrete data itself remains distributed over the possibly thousands websites.

The exact structure of the abstract view is often predefined by a view designer. Hence, from the perspective of end-users looking for information, abstract views are static and unchangeable. The same holds for the links between the abstract view and the data sources.

![Figure 1: Abstract view.](image)

A solution to this would be to allow users to define the views themselves. Such views are not static and neither are the links between the views and the concrete data sources. Unfortunately, users would have to have in-depth knowledge of the data sources and their schemas in order to be able to define the view, something an abstract view was meant to avoid.

Therefore, we let users specify their own personal model of the ‘universe of discourse’. For this a different kind of mediator system is needed, namely one that can automatically establish the links between the personal view and the concrete data sources in an ‘on demand’ fashion and with no help from professional view designers.

We here investigate the latter scenario. We call the abstract view defined in such a scenario a personal schema. More precisely, the term personal schema stands for an abstract schema created by a user, possibly in an ad-hoc manner. The personal schema is a representation of a part of the user’s ‘universe of discourse’. It embodies the user’s current information need and expectation with respect to the structure of the information, which may be entirely different from the structure of the actual data in the distributed data sources of the XML-Web, of which the user does not need any knowledge.

After having defined a personal schema, the user is then allowed to ask queries over that schema. We call such a query, a personal query.

**Example 1.** Fig. 2 shows a simple personal schema on the left and a personal query on the right. The personal schema defines a structure related to countries and towns in those countries. The personal query on the right, given as an XPath [20] expression, looks for a name of the country that contains a town named “Amsterdam”.

For both the personal schema and query, we follow W3C’s XML-related recommendations [20]. Hence, we assume the personal schema and the schemas of the distributed data sources to conform to XML-schema. A personal query is assumed to be an XPath or XQuery expression.

We will now describe the basic architecture of a mediator system capable of providing answers to personal queries over personal schemas, i.e., the personal schema based query answering system.

\(^1\)For example, http://www.prijsindex.net
2.2 The personal schema based query answering system

Fig. 3 shows the basic architecture of the personal schema based query answering system. The rounded rectangles denote the three main components of the system. The rectangles on the left represent the data being exchanged between the user and the system. On the right, ‘data source’ circles represent distinct XML-Web data sources, with both XML data and XML schemas exposed.

The personal schema based query answering system (i.e., PSQ) is a system capable of answering personal queries posed over personal schemas. Hence, the primary inputs from the user are a personal schema and a personal query.

The architecture is described in a step-wise manner. Each step represents an action performed by either the system or the user of the system. Arrows in Fig. 3 indicate the data-flow related to each action.

Step 0. The PSQ collects metadata from data sources and stores it in the metadata repository. The most important kind of metadata is the schema describing the data in the data source. The part of the metadata repository storing those schemas is called schema repository.

Note that some systems [1] additionally collect and store (i.e., replicate) data extracted from the data source. This is not the case in our approach.

Step 1. The first step in solving a user’s information need, is the user supplying a personal schema and accompanying query. For reasons of simplicity of explanation, we will assume that the personal schema is used only by the schema matcher, and that the personal query is only used by the query evaluator. In practice, both schema and query can be used by both PSQ components.
Step 2. The PSQ’s schema matcher matches the personal schema against the concrete schemas stored in the schema repository. Schema matching automatically established links between the personal schema and the concrete schemas. Links that form a match between a personal schema and one or more concrete schemas are called a mapping. Schema matching is based on heuristics and may produce more than one possible mapping for a single personal schema. The PSQ is not capable of deciding which of the mappings is the ‘best’ in a given situation. It makes an approximation, again based on heuristics, by ranking the produced mappings.

Step 3. The user is then asked to evaluate the offered mappings and decide which of them will most probably lead to an answer corresponding with his or her information need. The selected mapping is sent to the system’s query evaluator.

Step 4. The system uses the selected mapping to convert the personal query to queries on one or more data sources. After evaluating these queries, mappings are used again to convert the query results back to the user’s personal schema, which represents the final answer to the query.

To illustrate the similarities and differences between personal schema-based querying and traditional querying of the web, we compare it to a classical web search engine (i.e., Google) in the following section.

2.3 Analogy between Google and the personal schema based query answering system

The analogy between Google and a personal schema based query answering system resides in the similarity in steps that a typical usage session entails. For PSQ those steps were described in Section 2.2. Table 1 puts PSQ’s steps of Section 2.2 alongside Google’s.

<table>
<thead>
<tr>
<th>Step</th>
<th>Google.com</th>
<th>PSQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0. Offline, the system is . . .</td>
<td>collecting and indexing documents</td>
<td>collecting schemas from the XML-Web</td>
</tr>
<tr>
<td>1. User forms a query using . . .</td>
<td>keywords with some logical operators and keyword based constraints</td>
<td>a personal schema and a personal query</td>
</tr>
<tr>
<td>2a. System performs . . .</td>
<td>keyword based document search</td>
<td>personal to concrete schema matching</td>
</tr>
<tr>
<td>2b. System calculates a . . .</td>
<td>ranked list of candidate document URLs with document descriptions</td>
<td>ranked list of candidate mappings with mapping descriptions</td>
</tr>
<tr>
<td>3. User selects a . . .</td>
<td>URL of a document to be retrieved</td>
<td>mapping to be used in the personal query evaluation</td>
</tr>
<tr>
<td>4. System . . .</td>
<td>retrieves the document</td>
<td>evaluates the query</td>
</tr>
</tbody>
</table>

Table 1: Google analogy.

From this analogy, the improvements with respect to traditional web querying become apparent.

- The data model is upgraded from plain text to a structured data, i.e., XML.
- The query language is upgraded from keyword-based to a structured query language such as XPath or XQuery.
- Data granularity is upgraded (or should we say ‘downgraded’?) from document level to XML element level.
With the PSQ, the search engine paradigm for querying the Web remains the same. The user specifies his information need in the form of a template of the data he or she is interested in. The system proposes a list of possible ways to answer the question, one of which is selected by the user. Finally, the system retrieves the data.

The personal schema query answering system is a complex system. It incorporates many scientific and engineering challenges. We look into those challenges in some more detail in the following section.

2.4 Issues in a personal schema based query answering system

The PSQ performs complex tasks such as schema matching and distributed query evaluation. Furthermore, the PSQ is placed in the XML-Web environment (i.e., the Internet). Due to the volume of data of the XML-Web, technical specifics can have high impact on performance and must be accounted for. Consequently, a large number of interesting issues can be identified.

In this report focus is placed on schema matching as the most important part of the PSQ. The quality of schema matching directly influences the semantic relevance of the answer and determines user’s satisfaction. A well balanced trade-off between the quality of the matching and its operational efficiency is the ultimate goal in building a schema matcher for PSQ.

In this section, we present some important issues to be encountered on the path of achieving this goal. We distinguish the conceptual issues, i.e., those concerned with the quality of the solution and the performance related issues, i.e., those concerned with the efficiency of the solution. The non exhaustive list is given below:

- **Conceptual issues**: What are suitable languages for defining schemas and queries? Is the syntax/semantics of the query languages simple and transparent for the user? What is the semantics of such languages when used in schema matching? What is schema similarity and how to define schema matching? How to make schema matching transparent to the user and how to exploit user’s feedback? What is the output of the matching i.e. how to represent mappings?

- **Performance related issues**: How to be efficient in schema matching? How to store and reuse previous matching results? How to optimize simultaneous matching requests? How to ensure fast access to the metadata repository? How to ensure scalability of the repository with respect to its size and workload? How to dynamically balance the quality/performance ratio?

In the sequel we focus on defining the schema matching problem; a suitable and precise definition of schema matching is essential for investigating approaches to solving it. However, approaches for solving the matching problem are beyond the scope of this report.

3 A framework for defining a schema matching problem

In this chapter, we introduce a framework for defining the semantical XML schema matching problem. The framework incorporates three parts:

- a model for XML schemas (see Section 3.1),
- a framework for defining a generic matching problem (see Section 3.2), and
- a definition of automated semantic matching (see Section 3.3).

Based on these parts, the semantic XML schema matching problem, as encountered in the personal schema based query answering system, is defined in chapter 4.
3.1 Modeling XML schemas

The XML schema and DTD languages [20] are the two most commonly used languages for representing schemas of XML data. Between the two, XML schema is the more expressive one. We therefore use XML schema as our base schema language.

An XML schema can be seen as a graph, a common view in literature. A graph representing an XML schema is called a schema graph.

The model is described through methods for representing each of the XML schema language features. To this end, we will first introduce the elements of the XML schema language, then provide basic graph related definitions, and finally describe how to represent the former using the latter.

3.1.1 Features of the XML schema language

The XML schema language incorporates the following features.

1. The structure of an XML document is defined in an XML schema in terms of predefined hierarchical relationships between XML elements and/or attributes to which specific constraints concerning ordering, cardinality and participation are imposed (e.g., xs:element, xs:attribute, xs:sequence, xs:all, xs:choice, xs:minOccurs, xs:use, etc.).

2. The content of an XML document as found in elements or attributes can be restricted in an XML schema by defining it to take values from a domain of a predefined or user-defined datatype (e.g., xs:string, xs:simpleType, xs:restriction, xs:union, etc.).

3. Semantic invariants can be enforced in XML schema by imposing referential integrity or uniqueness constraints (e.g., xs:key, xs:keyref, xs:unique, etc.).

4. Semi-structured document regions are specified in XML schema by using wildcards specifying the location and properties of those regions (e.g., xs:any, xs:anyAttribute, etc.). Additionally, mixed content elements indicate free-form text.

5. Features supporting modularity and reusability in XML schema enable rapid schema development and reuse of, possibly adjusted, predefined schemas (e.g., xs:import, xs:include, xs:group, xs:extension, etc.).

6. Finally, documentation features facilitate human and machine understanding of an XML schema (e.g., xs:documentation, etc.).

3.1.2 Graph theory basics

In this section, we introduce some basic graph theory notation [2].

Def. 1 - Graph

A graph \( G \) is a 3-tuple \( G = (N_G, E_G, I_G) \) where:

- \( N_G = \{n_1, n_2, ..., n_i\} \) is a nonempty finite set of nodes,
- \( E_G = \{e_1, e_2, ..., e_j\} \) is a finite set of edges,
- \( I_G : E_G \rightarrow N_G^{(2)} \) is an incidence function that associates each edge \( e \in E_G \) with a set of two nodes \( \{u, v\} \in N_G^{(2)} \), where \( N_G^{(2)} = \{\{u, v\} | u, v \in N_G\} \), written as \( I_G(e) = \{u, v\} \).

We say edge \( e \) is incident to the end nodes \( u \) and \( v \). A function \( \text{Inc} : E_G \rightarrow N_G^{(2)} \) is defined as \( \text{Inc}(e) \equiv I_G(e) \), written as \( \text{Inc}(e) = \{u, v\} \).

Def. 2 - Directed graph

A directed graph \( G \) is defined as a graph \( G = (N_G, E_G, I_G) \) with the incidence function \( I_G \) redefined to:
Def. 3 - Degree of a node
The number of edges incident to a node in a graph \( G \) is called the degree of the node \( \text{deg}(n) \). In a directed graph we distinguish between \( \text{indeg}(n) \) and \( \text{outdeg}(n) \) representing the number of incident edges having node \( n \) as target and source node, respectively.

Def. 4 - Loop, parallel edges, reverse parallel edges
- Edge \( e \in E_G \) is a loop in graph \( G \) if and only if \( \text{Inc}(e) = \{n\} \) (i.e., \( \text{Inc}(e) = \{n, n\} \)), \( n \in N_G \).
- Edges \( e_1, e_2 \in E_G \) are parallel in graph \( G \) if and only if \( \text{Inc}(e_1) = \text{Inc}(e_2) \).
- Edges \( e_1, e_2 \in E_G \) are reverse parallel in directed graph \( G \) if and only if \( \text{source}(e_1) = \text{target}(e_2) \wedge \text{target}(e_1) = \text{source}(e_2) \).

Def. 5 - Walk, length of the walk, undirected walk, closed walk, path
- A Walk \( p \) in graph \( G \) is an alternating list of nodes and edges \( p = (n_0, e_{01}, n_1, e_{12}, n_2, \ldots, e_{l-1}, n_l) \), where for each edge \( e_{ij}, \text{Inc}(e_{ij}) = \{n_i, n_j\} \). Node \( n_0 \) is called the origin of the walk, \( \text{origin}(p) = n_0 \); node \( n_l \) is called the terminus of the walk, \( \text{terminus}(p) = n_l \).
- The length of a walk, \( \text{length}(p) \), is the number of edges in the walk.
- An directed walk in directed graph \( G \) is a walk \( p \) in \( G \), where \( p = (n_0, e_{01}, \ldots, n_i, e_{ij}, n_j, \ldots, e_{l-1}, n_l) \), such that for each edge \( e_{ij}, \text{source}(e_{ij}) = n_i \wedge \text{target}(e_{ij}) = n_j \).
- A closed walk \( p \) in graph \( G \) is a walk for which \( \text{origin}(p) = \text{terminus}(p) \).
- Contrary to the standard graph theory definition of a path, in this report, we use the term path for a walk.

Def. 6 - Partial graph, subgraph, partial subgraph
- A graph \( G' \) is a partial graph of a graph \( G \), if \( N_{G'} = N_G, E_{G'} \subseteq E_G \), and \( I_{G'} \) is such that \( \forall e \in E_{G'} : I_{G'}(e) = I_G(e) \).
- A graph \( G' \) is a subgraph of a graph \( G \), if \( N_{G'} \subseteq N_G, E_{G'} = \{e \in E_G \mid \text{Inc}(e) \cap \{N_{G'}\} \neq \emptyset \} \) and \( \forall e \in E_{G'} : I_{G'}(e) = I_G(e) \).
- A graph \( G' \) is a partial subgraph of graph \( G \) if \( G' \) is both partial graph and subgraph of graph \( G \). We depict partial subgraph relation using ‘\( \subset \)’ symbol: \( G' \subset G \).

Def. 7 - Labeled graphs
Let \( L \) be a set of labels. A graph \( G \) is said to be node-labeled, if it contains an additional bijective function \( LN_G : N_G \to L \). A graph \( G \) is said to be edge-labeled, if it contains an additional bijective function \( LE_G : E_G \to L \). A graph \( G \) can be both node-labeled and edge-labeled.
3.1.3 XML schema graph

We model XML schemas using schema graphs. Those are labeled directed graphs with property sets. This gives us the possibility of representing every XML schema feature as either a node or an edge of the graph, or to encode it using a node’s or an edge’s property.

Property sets for a graph $G = (N_G, E_G, I_G)$ are represented with function $PS_G$ as follows.

$PS_G : \{N_G \cup E_G\} \times \mathcal{P} \rightarrow \mathcal{V}$

where $\mathcal{P}$ is a set of properties, and $\mathcal{V}$ is a set of values including the null value. For example, we use $\mathcal{V} = \mathbb{R} \cup \mathbb{S} \cup \mathbb{U} \cup \{null\}$, where $\mathbb{R}, \mathbb{S}, \mathbb{U}$ are sets of real numbers, strings, and user-defined labels respectively.

For $n \in \{N_G \cup E_G\}$, $p \in \mathcal{P}$ and $v \in \mathcal{V}$ we can write $PS_G(n, p) = v$, or alternatively, if $G$ is clear from the context, $p(n) = v$.

We denote schema graphs as $\hat{G}$, e.g., $\hat{G} = (N_G, E_G, I_G, PS_G)$. Further more, all the operations performed on graphs can also be performed on schema graphs, with the $dom(PS_G)$ consequently being restricted or extended according to the changes of $N_G$ and $E_G$.

Example 2. A property set for XML schema includes the following properties $\mathcal{P} = \{kind, name, isElement, compositor, maxOccurs, \ldots\}$.

For the XML schema graph shown in Fig. 4 we can write $name(n_1) = \text{‘bank’}$, $maxOccurs(e_1) = 2000$, $compositor(n'_1) = \text{sequence}$, etc.

Proper labeling of schema graphs requires unique identifiers, such as those proposed in ‘XML Schema: Component Designators’ [20], for each node and edge. In our examples however, arbitrarily chosen labels are used to distinguish between the nodes and the edges (i.e., $n_1, e_1, \ldots$).

Schema matching can exploit the whole span of information contained within schemas. However, some schema features can be ignored in the matching process. For the sake of simplicity, in our models and matching algorithms, we omit features related to reusability, modularity, and derivation. Apart from these features, a schema graph is a complete representation of an XML schema, i.e., the one can be converted into the other and vice versa without loss of information. Additionally, we do not support recursive XML schemas, which we regard as beyond the scope of our current research.

Listed below are the rules used to transform an XML schema into its schema graph counterpart. The rules are defined in an informal way. An example schema graph given in Fig. 4 is used to support the discussion.

1. XML elements and attributes are represented by nodes in the graph. These nodes are called markup nodes. This is modeled with the kind property of the node (i.e., $kind(n) = \text{markup}$).

Further, the distinction between elements and attributes is encoded using isElement property of the markup node. This property can often be ignored in schema matching. Other properties of the markup node include its name, and datatype for the leaf nodes. All the rounded nodes in Fig. 4 are markup nodes.

2. XML model groups (i.e., sequence, choice and all) are represented using a so called model group node in the schema graph (e.g., $kind(n'_1) = \text{model group node}$). To distinguish between the three compositor types, a property compositorType is used. Additionally, for nodes $n$ having $compositorType(n) = \text{choice}$ all the edges exiting node $n$ (i.e., $source(e) = n$) have their choiceEdge property set to true (i.e., $choiceEdge(e) = \top$). Choice edges are graphically depicted with an arc drawn over them (see $e_4$ and $e_5$ in Fig. 4).

We will show later that some model group nodes can be removed from the schema.

3. The hierarchical structure of an XML schema (i.e., the parent-child relationships between elements) is modeled using so called implicit edges in the schema graph (i.e., $kind(e) = \text{implicit edge}$). Implicit edges are depicted with a solid line edges. The cardinality of the parent-child relationship is represented as a property of the implicit edge. Note that logically,

---

Footnote: In practice, this means that all reusability, modularity, and derivation based constructions are resolved and rewritten before matching is performed.
parent-child relationship in XML schema has a cardinality assigned to both of its ends (i.e., directions). First, `xs:minOccurs` and `xs:maxOccurs` specify the cardinality of the child with respect to its parent. This cardinality is physically modeled with `minOccurs` and `maxOccurs` properties of the edge (e.g., `maxOccurs(e1) = 2000`). The other cardinality direction is the child-parent cardinality, which is always 1 (one and only one) as there exists only one parent particle for any child particle. This cardinality is not explicitly encoded using edge properties.

4. Explicit relationships defined in XML schema by means of `xs:key` and `xs:keyref` pairs or similar mechanisms, are modeled using so called explicit edges (i.e., kind(e1) = explicit edge). An explicit relationship is defined in XML schema by specifying the related schema particles and a predicate function i.e., a join condition between the particles. Such relationships are bidirectional with specific and possibly different cardinalities in both directions. To preserve the compatibility with the implicit edges (those can model cardinalities in one direction only), explicit relations are represented using a pair of reverse parallel edges, now called explicit edges. Visually they are depicted as dashed lines. In Fig. 4, an explicit edge exists between nodes n1 (i.e., /bank/safe) and n8 (i.e., /bank/users/person) with predicate function `join($safe, $person) := $safe/number ∈ $person/safe_no` (a typical join condition).

Unlike cardinality for parent-child relationships, XML schema does not provide a tool for specifying the cardinality of explicit relationships. We will anticipate, however, a future existence of a language extension capable of expressing those cardinalities in XML schema. In our example, the explicit edge cardinalities specify that a ‘safe’ can belong to zero or one ‘person’, and a ‘person’ can have zero to ten ‘safes’.

5. Schema graphs can be reduced by removing the redundant model group nodes. Namely, in many cases, model group nodes node can be omitted without changing the perceptive meaning of the schema graph. When any node points to a particular composito
A constraint satisfaction problem

Def. 8 - Constraint satisfaction problem

problems.

the following, we more formally describe the basics of CP can be distinguished, such as constraint satisfaction and constraint optimization problems. In (i.e., without actually having to search the entire search space). Several classes of amounts to searching through the search space in such a way that a solution can be quickly found a vector space

A constraint satisfaction problem is a constraint optimization problem

3.2 Generic matching problem is a constraint optimization problem

To be able to analyze ways to solve matching problems, we first need to precisely define the matching problem, not for solving it. matching problems. We stress that the constraint programming framework will be only used for defining the matching problem, not for solving it.

3.2.1 Constraint programming basics

Constraint programming CP [14, 3] is a generic framework for problem description and solving. It strictly separates the declarative and operational aspects of problem solving. Informally, a CP problem is described in terms of variables which can take values from certain domains and for which a number of constraints (requirements) should hold. The domains effectively specify the search space. A solution is an assignment of all variables with values from the corresponding domains, i.e. a vector in the search space, for which all constraints hold. In this framework, solving a problem amounts to searching through the search space in such a way that a solution can be quickly found (i.e., without actually having to search the entire search space). Several classes of CP problems can be distinguished, such as constraint satisfaction and constraint optimization problems. In the following, we more formally describe the basics of CP needed for defining (schema) matching problems.

Def. 8 - Constraint satisfaction problem

A constraint satisfaction problem \( P \) is defined as a 3-tuple \( P = (X, D, C) \) where

- \( X = (x_1, \ldots, x_n) \) is a list of variables,
- \( D = (D_1, \ldots, D_n) \) is a list of finite domains, such that variable \( x_i \) takes values from domain \( D_i \). \( D \) is called search space for problem \( P \).
- \( C = \{c_1, \ldots, c_k\} \) is a set of constraints, where \( c_i : D \rightarrow \{\top, \bot\}, i = 1, k \), are predicates over one or more variables in \( X \).

Def. 9 - Valuation, partial valuation

A valuation \( \Theta_X \) for a list of variables \( X = (x_1, \ldots, x_n) \) and a search space \( D = (D_1, \ldots, D_n) \) is a vector \( \Theta_X = (\theta_1, \ldots, \theta_n) \) such that \( \theta_i \in D_i, i = 1, n \). We write \( \Theta_X \in D. \)

Any expression \( e(X) \) can be evaluated by substituting all variables in \( X \) with corresponding values from valuation \( \Theta_X \). The expression \( e(\Theta_X) \) becomes variable free, whereby its value can be calculated directly.

A partial valuation \( \Theta^P_X \) for a list of variables \( X = (x_1, \ldots, x_n) \) and a search space \( D = (D_1, \ldots, D_n) \)
is a vector \( \vec{\Theta}_X = (\theta_1, \ldots, \theta_n) \) such that \( \theta_i \in D_i \cup \{\text{null}\}, i = 1, n \). Variables in \( X \), not being assigned a null value by \( \vec{\Theta}_X \), comprise a partial variable list \( X' \), formalized as \( X' \subset X \). Further, \( D' \) is a restriction of \( D \) to variables in \( X' \), formalized as \( D' \subset D \). We write \( \vec{\Theta}'_X \in D' \).

**Def. 10 - Constraint satisfaction solution**

A solution for a constraint satisfaction problem \( P = (X, D, C) \) is any valuation \( \vec{\Theta}_X \in D \) such that satisfies all constraints in \( C \):

\[
\bigwedge_{i=1}^{k} c_i(\vec{\Theta}_X) = \top
\]

The definitions presented so far define a so called constraint satisfaction problem, or CSP. Typical for a CSP is that the goal is to find solutions to the problem whereby no solution is better than another. For schema matching, however, not all solutions should carry the same importance. Therefore, we use another class of CP problems, called a constraint optimization problem, or COP, which is an extension of a CSP with an objective function \( \Delta(X) \). The objective function provides a measure of how good a solution is, hence which of the proposed solutions is the best. The goal of constraint optimization is to find the solution with the minimal (or maximal) value for the objective function.

**Def. 11 - Defining a constraint optimization problem**

A constraint optimization problem \( P \) is a 4-tuple \( P = (X, D, C, \Delta) \) where

- \( X, D, C \) as in Def. 8,
- \( \Delta : D \rightarrow \mathbb{R} \) is a function assigning a numerical value to a valuation \( \vec{\Theta}_X \) (i.e., \( \Delta(\vec{\Theta}_X) \in \mathbb{R} \)).

For simplicity, we define that a smaller value for \( \Delta \) corresponds with a better solution.

**Def. 12 - Constraint satisfaction solution**

The definition of a solution is the same as in Def. 10. The best solution (or simply the solution) for a COP \( P \) is that valuation \( \vec{\Theta}'_X \) for which \( \Delta(\vec{\Theta}'_X) \) is minimal, i.e.,

\[
\exists \vec{\Theta}'_X \bullet (\vec{\Theta}'_X \text{ is a solution}) \land \Delta(\vec{\Theta}'_X) < \Delta(\vec{\Theta}_X)
\]

Insofar the declarative part of CP problems. The goal of constraint programming research is to develop techniques and algorithms for solving CP problems efficiently. A naive approach to solving COPs is the ‘generate, test and rank’ approach. A generator component systematically generates valuations by enumerating all possible combinations of values of domains \( D_i \). Each valuation is then tested against all constraints. If all constraints are satisfied, the valuation is declared a valid solution and ranked using the objective function.

The complexity of the naive approach is exponential. More advanced techniques and algorithms are needed for real-life problems. Domain and constraint properties can be used to create intelligent generators. Such generators reduce the search space by not generating valuations for which it is obvious, or for which it can quickly be determined, possibly based on only a partial variable assignment, that those valuations cannot be a solution or a best solution. This reduces the number of ‘generate and test’ iterations, hence improves efficiency. Further improvement can be obtained by incorporating heuristics to try to generate those valuations first, that are likely to be correct and highly ranked. To this end, different algorithms have been proposed including variants of backtracking algorithms, and stochastic and AI algorithms.

We now proceed by focusing on generic matching problems and formalizing them using the CP framework.

### 3.2.2 Informal definition of a generic matching problem

As explained before, a matching problem can be modeled in the most general sense as a template object, that has to be matched with a set of target objects. A solution has to be selected from
the set of matches according to some criteria. In this section, we informally describe the generic matching problem in more detail.

Def. 13 - Matching problem

A matching problem $\Pi$ is defined as a 4-tuple $\Pi = (T, R, M_c, S_c)$ (see Figure 5) where

- $T$ is a template object,
- $R = \{to_1, \ldots, to_n\}$ is a repository (i.e., a set) of target objects,
- $M_c$ is a matching criterion, and
- $S_c$ is a selection criterion.

$M_c$ and $S_c$ are described below.

Example 3. Figure 6 illustrates a matching problem defined as follows: a circle $K$ needs to be matched against a set of squares $A_i, i = 1, k$ in order to find the one square that has the most similar area to that of a circle, whereby the circle must fit inside the square.

The circle $K$ is defined with its radius $r_1$ and the squares $A_i, i = 1, k$ are defined by their respective edge lengths $a_i, i = 1, k$. We use the ‘dot’-based object oriented syntax for representing properties of objects. For instance, we can express the former statements as $K \cdot \text{radius} = r_1$ and $A_i \cdot \text{edge} = a_i, i = 1, k$.

We will now define this as a matching problem by defining the four components of Def. 13 $\Pi = (T, R, M_c, S_c)$
• The template object $T = K$ (the circle).
• The repository $R = \{A_1, \ldots, A_k\}$ (the set of squares).
• The matching criterion $M_c$ defines how to match the template object against the set of target objects. It is defined as $M_c = \text{“similar area” and “circle must fit inside the square”}$.
• The selection criterion $S_c$ defines how to select the solution. For the example, it is defined as $S_c = \text{“pick the one with the most similar area”}$.

### 3.2.3 The definition of a matching problem as a COP

The informal definition of a matching problem can be formalized by defining it as a COP (see Def. 11). We have to fit a matching problem $\Pi = (T, R, M_c, S_c)$ into a COP problem $P = (X, D, C, \Delta)$. We approach this in a stepwise manner by formalizing the components of the matching problem one-by-one.

**Template object** A template object $T$ in $\Pi$ can be formalized through variables in $X$ and variable domains $D$ in $P$. More concretely, we can add a variable $x_t$ to $X$, and its domain $D_t = \{T\}$ to $D$. Note that this is a workaround due to inability of a COP to directly specify constant values such as an object like $T$ in this case. We shall introduce abbreviation notation to represent template objects in a COP later.

The matching problem $P = (X, D, C, \Delta)$ of example 3 is so far defined as:

$$
X = (x_t)
$$
$$
D = (D_t) \text{ where } D_t = \{K\}
$$
$$
C = \text{yet unknown}
$$
$$
\Delta(X) = \text{yet unknown}
$$

**Repository** A repository $R$ in $\Pi$ can be formalized through variables $X$ and variable domains $D$ in $P$. A variable $x_{to}$ is added to $X$, and its domain $D_{to} = R$ is added to $D$.

The matching problem $P = (X, D, C, \Delta)$ of example 3 is so far defined as:

$$
X = (x_t, x_{to})
$$
$$
D = (D_t, D_{to}) \text{ where } D_t = \{K\}, D_{to} = \{A_1, \ldots, A_k\}
$$
$$
C = \text{yet unknown}
$$
$$
\Delta(X) = \text{yet unknown}
$$

**Matching criteria** The matching criterion $M_c$ in $\Pi$ can be formalized through constraints $C$ and objective function $\Delta$ in $P$. $M_c$ expresses what makes a target object a desirable match for a template object. Formalization of $M_c$ is neither straightforward nor unique, because it involves a sometimes complex design process which cannot be generalized.

Example 3 identified $M_c$ as

• "similar area" and
• "circle must fit inside the square"

The two statements differ in nature. The first represents a calculation of ‘similarity’ between circle and square with respect to their ‘areas’. The second one is a predicate stating a boolean condition that must be satisfied for every solution.

In this specific problem, we can formalize the notion of ‘area similarity’ as a numerical function that calculates the difference between the areas of the template and target object. Hence, in this concrete case, $M_c$ can be formalized as an objective function $\Delta$ in $P$ as

$$
\Delta(X) = |x_t \cdot \text{radius}^2 - x_{to} \cdot \text{edge}^2|
$$
The second item of $M_c$ can be represented in $P$ by adding a constraint $c(X)$ to $C$. The constraint checks if the circle fits inside the square.

\[ c(X) = 2 \times x_t \cdot \text{radius} \leq x_{to} \cdot \text{edge} \]

The matching problem $P = (X, D, C, \Delta)$ of example 3 is so far defined as:

\[
\begin{align*}
X &= (x_t, x_{to}) \\
D &= (D_t, D_{to}) \text{ where } D_t = \{ K \}, D_{to} = \{ A_1, \ldots, A_k \} \\
C &= \{ c(X) \} \text{ where } c(X) = 2 \times x_t \cdot \text{radius} \leq x_{to} \cdot \text{edge} \\
\Delta(X) &= |x_t \cdot \text{radius}^2 \times \pi - x_{to} \cdot \text{edge}^2|
\end{align*}
\]

**Selection criterion** $S_c$ is not formalized explicitly through the components of $P$. Its importance lies in the fact that it specifies what kind of constraint problem we are dealing with. For example, $S_c$ can specify a matcher to find

1. one – any target object that satisfies the constraints,
2. all – all target objects that satisfy the constraints ordered by the value of the objective function,
3. best or top $N$ – those $N$ target objects that satisfy the constraints and have the lowest value for the objective function, or
4. some good – several target objects that satisfy the constraints and have a near optimal value for the objective function.

For the matching problem in example 3, we have identified $S_c$ to be “find the most similar one”, i.e., the best match. This specifies that we are dealing with a constraint optimization problem looking for the best solution. In the sequel, we will not formally specify $S_c$ unless it contributes to the clarity of the explanation.

**Abbreviation** To distinguish between the variables of template and target objects, we abbreviate the problem specification by moving the template object variable and respective domain out of the $X$ and $D$ lists. To illustrate this ‘cosmetic’ change, we show how it reflects to the example problem definition. Note that $C$ and $\Delta(X)$ are not affected.

- $X = (x_t, x_{to})$ is abbreviated to $X = (x_{to})_x$.
- $D = (D_t, D_{to})$ where $D_t = \{ K \}, D_{to} = \{ A_1, \ldots, A_k \}$ is abbreviated to $D = (D_{to})_K$.

To summarize, we have defined a matching problem through four components: template and target objects, matching criteria and a selection criterion. We have shown how those can be expressed through the COP formalism in terms of variables, variable domains, constraints and an objective function. All the described components are declarative in nature. They do not specify how to solve the problem, hence do not restrict a matcher in choosing an efficient method of computing the matching solution.

### 3.3 Automated semantic matching

We will proceed with a discussion on what is the meaning of ‘semantic’ in semantic matching. We, furthermore, show how semantic matching can be automated. Our definitions were inspired by the formal definitions of semantic matching in [9, 12].

**Def. 14 - Semantic schema, semantic matching**

A semantic schema $U$ is a human’s (i.e., user’s) representation of the Universe. A semantic schema is composed of semantic concepts $\alpha, \beta, \ldots$. Each user has an unique subjective semantic schema.
Fig. 7 illustrates the semantic world. It shows user A and user B and their respective semantic schemas $U_A$ and $U_B$. Semantic schemas cannot be made concrete, because they only exist ‘in the minds’ of users. As such, semantic schemas are abstract representations of human understanding of the Universe.

Semantic matching is an activity performed by the user aimed at expressing the similarity between semantic concepts in his semantic schema. Both matching and the notion of similarity are modeled through a subjective semantic distance function $\Lambda(\alpha, \beta), \alpha, \beta \in U$. Without loss of generality, we can assume that $\Lambda(\alpha, \beta)$ returns values in the range of $[0, 1]$ indicating the level of semantic similarity perceived by the user between semantic concepts $\alpha$ and $\beta$.

The semantic distance function is an abstract representation of the mental process through which a user realizes the level of similarity between semantic concepts. This function cannot be directly made concrete as it exists only in the mind of a user as well.

Def. 15 - Syntactic construction, syntactic matching

A syntactic construction $c$ is a formal, i.e., syntax-based concrete representation of some semantic concept $\alpha$. The reverse can also be said, that $\alpha$ is a representation of $c$, when concrete object $c$ already existed and the user created a mental model of it (i.e., semantic concept $\alpha$). A syntactic schema is a set of syntactic constructions.

Fig. 7 shows two syntactic schemas $S_1$ and $S_2$. In the example, user A is trying to understand how similar syntactic constructions $a$ and $b$ are by comparing semantic concepts $\alpha$ and $\beta$ in his mind, which represent $a$ and $b$, respectively.

Syntactic matching is the formal activity (that can, for example, be performed by a computer) aimed at calculating the syntactic similarity between two syntactic constructions belonging to any two syntactic schemas. Both notions of matching and syntactic similarity are modeled through an objective syntactic distance function $\Delta(a, b), a, b \in S_1, S_2$. Without loss of generality, we assume that $\Delta(a, b)$ returns values in the range of $[0, 1]$ indicating the level of syntactic similarity between syntactic concepts $a$ and $b$.

Def. 16 - Understand, design

Understanding is the subjective capability of a user A to establish a represents (|=) relation between a syntactic construction $a \in S$ and a semantic concept $\alpha \in U_A$. Understanding is represented with a subjective mapping function $M_A$ that maps a syntactic construction onto an element of $U_A$

$$\forall a \in S, \alpha \in U_A : a |= \alpha \Leftrightarrow M_A(a) = \alpha$$

Note that the mapping function is abstract and cannot be directly made concrete as it exists only in the mind of the user.
Design is the capability of a user $A$ to do the reverse, i.e., create a syntactic representation $a$ for a semantic concept $\alpha \in U_A$. Design is represented by an inverse subjective mapping function, i.e., $M_A^{-1}(\alpha) = a$.

The definitions presented above show that all notions of the semantical world are abstract and subjective and cannot be made concrete directly. Semantics only exists in human minds. This means that automated semantic matching is, in principle, impossible. The best thing a computer can do, is to make use of syntactic constructions to simulate and approximate semantic matching, which is how we define automated semantic matching.

**Def. 17 - Automated semantic matching**

In light of Def. 14, Def. 15, and Def. 16, automated semantic matching is defined as the process in which an objective syntactic distance function $\Delta(a, b), a, b \in S_1, S_2$ is used to simulate/approximate the results of a user-dependent semantic distance function $\Lambda(M(a), M(b)), M(a), M(b) \in U$.

‘User-dependent’ in the definition above denotes the need to have a separate syntactic distance function for each user. Usually, this inherent requirement is relaxed, as we did above, by assuming all users have a sufficiently similar semantic conception of all syntactic constructions, i.e., there is a universe $U$ that is sufficiently similar to the $U_A$ of any user $A$. Optionally, one could add another parameter to the syntactic distance function, the user: $\Delta(a, b, A), a, b \in S_1, S_2, A \in Users$ simulating the semantic distance function $\Lambda_A(M_A(a), M_A(b)), M_A(a), M_A(b) \in U_A$.

## 4 Definition of the XML schema matching problem

In this section, the three components of the framework of section 3, the schema graph, matching formalized as a COP, and automated semantic matching, are combined to define the problem of semantic matching of XML schemas in terms of the formalism for defining constraint optimization problems. We proceed by introducing specific axioms that will turn our, so far generic, matching definitions into more specific ones. The axioms are related to the specifics of the design (see Def. 16) of an XML schema.

### 4.1 Design of XML schemas

We have defined the design activity to be the process that converts semantic concepts into their syntactic representations: syntactic constructions. The following axioms specify how we use schema graphs as syntactic constructions.

Schema graphs consists of nodes, edges and their properties. These are all syntactic constructions and are used to represent semantic concepts. For ease of understanding, we will assume that semantic concepts come in two variants: semantic entities (e.g., sun, work, address, size, price) and semantic relations (e.g., left of, is part of, is parent of, is child of). Furthermore, the composition of two or more semantic relations results in another semantic relation.

**Axiom 1 - A node represents a semantic entity**

Each **node** in a schema graph represents one **semantic entity** (i.e., one semantic concept $\alpha$ in a specific semantic schema $U$).

**Example 4.** Node $n_7$ in Figure 8 represents the semantic entity of a set of ‘users’. Node $n_1$ represents the semantic entity ‘bank’.

**Axiom 2 - An edge represents a semantic relation**

Each **edge** in a schema graph represents one **semantic relation**, such that the semantic relation relates semantic entities represented by the end nodes of the edge. The direction of the edge is the same as the direction of the semantic relation.

**Example 5.** Edge $e_4$ represents the semantic relation ‘bank serves a group of users’. The same edge in the opposite direction represents the inverse semantic relation ‘users are served by the bank’.
Figure 8: Bank schema

Def. 18 - A path represents a [composite] semantic relation

Any path (i.e., walk) in the schema graph represents one semantic relation, such that the semantic relation relates semantic entities represented by the end nodes of the path. Such a semantic relation is a composition of the semantic relations represented by the edges of the path. The path traversal direction determines the composition order of the semantic relations.

Example 6. Consider the path \( p = (n_1, e_6, n_7, e_7, n_8, e'_8, n_1) \) connecting the ‘bank’ (\( n_1 \)) and ‘safe’ (\( n_2 \)) nodes. This path represents the semantic relation: A bank is related to a number of safes in such a way that these safes are used by persons who are the members of the group of users served by the bank. Another path that connects the same end nodes is \( p' = (n_1, e_1, n_2) \). \( p' \) represents a different semantic relation: a bank is related to all the safes that are owned by the bank.

The axioms presented above do not specify how to use property sets to represent the additional aspects of semantic concepts. Concretely, it is not specified how a name for a node or a datatype for a value is selected, or how cardinalities are assigned to edges. Though some guidance on how to use such features exists in the form of data modeling guidelines, design is quite subjective and based on the designer’s ‘best practices’. This is an important source of heterogeneity in XML schema graph design.

4.2 Components of the semantic XML schema matching problem

In section 3.2.2, we defined a generic matching problem \( \Pi \) in terms of four components \( \Pi = (T, R, M_c, S_c) \). We proceed by relating aspects of XML schema matching to these components.

In XML schema matching, the template object \( T \) is a schema graph \( \hat{T} \), or more exactly, a personal schema graph. The repository of target objects \( R \) consists of many schema graphs belonging to the various data sources. The repository can be treated in two different manners: as a collection of independent schema graphs, or as one large schema graph. The same distinction can be made in the matching task (note that in both cases, ‘similarity’ refers to the similarity as defined in semantic matching). The matching task is either

- for a template schema \( \hat{T} \) find the most similar target schemas in \( R = \{\hat{t}_0, \ldots, \hat{t}_k\} \). The output of this matching approach is a list of concrete schemas from \( R \), namely the ones most similar to \( \hat{T} \).

- for a template schema \( \hat{T} \) find the most similar partial subgraphs (see Def. 6) \( \hat{t}_i, \ i = 1, k \) in \( R = \{\hat{R}\} \) such that \( \forall \hat{t}_i : \hat{t}_i \subseteq \hat{R} \), \( i = 1, k \). The output of this matching approach is a list of subgraphs of the one target object \( \hat{R} \) in \( R \). Subgraphs can in general be composed of nodes and edges from different concrete schema graphs participating in \( \hat{R} \).
In our research we adopt the second matching goal. We assume that in certain application domains, schemas are logically interconnected, and that the answer to a personal query should be obtained by joining data from several distinct data sources.

**Def. 19 - Template object in XML schema matching**
The template object \( T \) in an XML schema matching problem \( \Pi \) is the schema graph \( \hat{T} = (N_{\hat{T}}, E_{\hat{T}}, I_{\hat{T}}, PS_{\hat{T}}) \).

**Def. 20 - Target objects in XML schema matching**
A set of target objects \( R \) in an XML schema matching problem \( \Pi \) is a set of all possible schema graphs \( \hat{t} \) such that \( \hat{t} \subseteq \hat{R} \) (i.e., \( \hat{t} \) is a partial subgraph of \( \hat{R} \)).

**Def. 21 - Matching criteria in XML schema matching**
Matching criteria \( M_c \) in \( \Pi \) defines two aspects of the XML schema matching problem. First it restricts the structure of the target object \( \hat{t} \) where \( \hat{t} \subseteq \hat{R} \), based on its relation with the template object \( \hat{T} \):

1. for each node \( n \in N_{\hat{T}} \), there exist one corresponding node \( n' \in N_{\hat{t}} \). Node \( n' \) is a match node for \( n \) formalized as \( n' = \text{Match}(n) \),
2. for each path \( p \in \hat{T} \), there exists one corresponding path \( p' \in \hat{t} \) such that \( \text{origin}(p') = \text{Match}(\text{origin}(p)) \) and \( \text{terminus}(p') = \text{Match}(\text{terminus}(p)) \). Path \( p' \) is a match path for path \( p \) formalized as \( p' = \text{Match}(p) \).

   Based on the compositionality of semantic relations (see Def. 18), we simplify condition 2 into:

2. for each edge \( e \in E_{\hat{T}} \), there exist a path \( p' \in \hat{t} \) such that, \( \text{origin}(p') = \text{Match}(\text{source}(e)) \) and \( \text{terminus}(p') = \text{Match}(\text{target}(e)) \). Path \( p' \) is a match path for edge \( e \) formalized as \( p' = \text{Match}(e) \).

3. a fact that a template schema graph \( \hat{T} \) and a target schema graph \( \hat{t} \) meet the conditions 1 and 2 is formalized as \( \hat{t} = \text{Match}(\hat{T}) \).

Second, \( M_c \) in an XML schema matching problem \( \Pi \) defines a way to calculate the results of a user-dependent semantic distance function \( \Lambda(M(T), M(\hat{t})) \) (see section 3.3). As discussed, this is done by defining an objective syntactic distance function \( \Delta(T, \hat{t}) \) (see Def. 17), used to simulate \( \Lambda \). Having in mind that \( \hat{t} \) is defined as \( \hat{t} \) and that \( T \) is defined as \( \hat{T} \), the actual distance function \( \Delta \) calculates the syntactic distance between the template schema graph \( \hat{T} \) and the target schema graph \( \hat{t} \), e.g., \( \Delta(\hat{T}, \hat{t}) \). The ways to define \( \Delta \) function for semantic XML schema matching will be discussed in section 5.

Semantic XML schema matching does not imply any specific selection criteria. Any selection criterion of section 3.2.3 can be used.

### 4.3 XML schema matching problem as a constraint optimization problem

We have defined above a semantic XML schema matching as a generic matching problem \( \Pi = (T, R, M_c, S_c) \). We will now proceed by formalizing the problem \( \Pi \) as an COP problem \( P = (X, D, C, \Delta) \). As discussed in section 3.2 we have selected COP as a framework for a precise formal specification of matching problems, most suitable for further investigation.

**Example 7.** We will use Fig. 9 as our example. Personal schema graph \( \hat{T}_1 \) as shown in Fig. 9a) is defined as follows.

\[
\hat{T}_1 = (N_{\hat{T}_1}, E_{\hat{T}_1}, I_{\hat{T}_1}, PS_{\hat{T}_1}), \text{ where}
\]

\[
N_{\hat{T}_1} = \{n_1, n_2, n_3\}
\]

\[
E_{\hat{T}_1} = \{e_1, e_2\}
\]
\[ I_{\hat{T}_1} = I_{\hat{T}_1}(e_1) = (n_1, n_2), \quad I_{\hat{T}_1}(e_2) = (n_1, n_3) \]
\[ PS_{\hat{T}_1} = \text{concrete properties are not relevant for this example} \]

\[ \hat{T}_1 \]

\[ \hat{X} \]

\[ \hat{R} \]

Figure 9: a) template schema graph, b) target schema graph variables, c) repository schema graph

A repository schema graph \( \hat{R} = (N_{\hat{R}}, E_{\hat{R}}, I_{\hat{R}}, PS_{\hat{R}}) \) is shown in Fig. 9c). Let \( P_{\hat{R}} \) be the set of all paths in \( \hat{R} \).

**Def. 22 - COP formalization of template schema graph \( \hat{T} \)**

The template schema graph \( \hat{T} = (N_{\hat{T}}, E_{\hat{T}}, I_{\hat{T}}, PS_{\hat{T}}) \) can be modeled in a COP problem \( P = (X, D, C, \Delta) \) as follows:

- for each node \( n_i \in N_{\hat{T}} \), a node variable \( t_{n_i} \) and a domain \( D_{n_i} = \{ n_i \} \) are added to \( P \),
- for each edge \( e_i \in E_{\hat{T}} \), an edge variable \( t_{e_i} \) and a domain \( D_{e_i} = \{ e_i \} \) are added to \( P \).

**Example 7 continued.** Based on Def. 22, we have defined \( P \) so far as

\[ X = (\text{undefined})_{t_{n_1}, t_{n_2}, t_{n_3}, t_{e_1}, t_{e_2}} \]
\[ D = (\text{undefined})_{n_1, n_2, n_3, e_1, e_2} \]
\[ C = \{ \text{undefined} \} \]
\[ \Delta(X) = \text{undefined} \]

A target schema graph \( \hat{t}_0 \) is represented in the same manner as a template schema graph \( \hat{T} \). Variables needed to describe a target schema graph are illustrated in Fig. 9b), with graphs \( \hat{X} \) and \( \hat{T}_1 \) being isomorphic.

**Def. 23 - COP formalization of target schema graph \( \hat{t}_0 \)**

Def. 20 defined a target object to be a schema graph \( \hat{t}_0 \) such that \( \hat{t}_0 \subset \hat{R} \). Def. 21 further restricted the structure of \( \hat{t}_0 \) based on its relation with \( \hat{T} \). Based on Def. 20 and Def. 21 the target schema graph \( \hat{t}_0 \) can be modeled in a COP problem \( P = (X, D, C, \Delta) \) as follows:

- for each node \( n_i \in N_{\hat{T}} \) a node variable \( x_{n_i} \) and a domain \( N_{\hat{R}} \) are added to \( P \),
- for each edge \( e_i \in E_{\hat{T}} \) a path variable \( x_{p_i} \) and a domain \( P_{\hat{R}} \) are added to \( P \),
- for each edge \( e_i \in E_{\hat{T}} \), and \( I_{\hat{T}}(e_i) = (n_o, n_t) \), a constraint \( ic_i(X) := \text{origin}(x_{p_i}) = n_o \land \text{terminus}(x_{p_i}) = n_t \) is added to \( P \). We will jointly denote the conjunction of all such constraints with \( IC(X) \) (i.e., incidence constraints):

\[ IC(X) = \bigwedge_{k=1}^{\text{card}(E_{\hat{T}})} \text{ic}_k(X) \]
Example 7 continued. Def. 23 extends the definition of $P$ to

\[
X = (x_{n_1}, x_{n_2}, x_{n_3}, x_{p_1}, x_{p_2})_{t_1, t_2, \ldots, t_{n_1}, t_{n_2}} \\
D = (N_{R_l}, N_{R_i}, N_{R_i}, P_{R_l}, P_{R_i})_{n_1, n_2, n_3, e_1, e_2} \\
C = \{IC(X)\} \\
\Delta(X) = \text{undefined}
\]

Def. 24 - COP formalization of a matching criteria $M_c$

The objective syntactic distance function $\Delta$ defined as a part of $M_c$ (see Def. 21) is directly represented, i.e., reused in $P$. As both the template $T$ and the target $to$ schema graphs are represented through variables in $X$ (see Def. 22 and Def. 23), we can directly transform a function of the form $\Delta(T, to)$ into an identical function of the form $\Delta(X)$, assuming the function $\Delta(X)$ can use the property set functions $PS_T$ and $PS_{to}$.

The details on how to design an objective function $\Delta(X)$, that is, an objective syntactic distance function $\Delta(T, to)$ will be discussed in chapter 5.

4.4 Additional constraints in the schema matching problem definition

So far, we have defined one constraint $IC(X)$ in $P$ (see Def. 23). We have not considered the fact that a schema graph is not just an ordinary graph. Schema graphs additionally encode specific semantic aspects of the XML schema language. These can be modeled in $P$ by adding new constraints.

Example 8. Choice edges $e_4$ and $e_5$ in Fig. 8 are mutually exclusive. That means that there will never exist an XML documents having both the $<\text{small}/>$ and the $<\text{large}/>$ elements within one $<\text{size}>$ element. Semantically, this can be interpreted as follows. The semantic relation represented by path $p = (n_5, e_4, n_4, e_5, n_6)$ has no meaning, since the semantic sub-relations represented by $e_4$ and $e_5$ are mutually exclusive. The path $p$ wrongly reads as: "for all the small sizes of the object, give me the large sizes of the same object".

We can define constraints in $C$ to filter valuations that include paths that have no correct semantical interpretation.

Example 9. Implicit edges, like $e_7$ in Fig. 8, are used to represent both an implicit semantical relation and its inverse. Explicit semantical relations, however, e.g. ‘person-safe’, are represented using two edges in the schema graph: one edge representing the semantical relation (e.g., $e_{x1}$) and the other representing its inverse (e.g., $e'_{x1}$). For this reason, explicit edges should never be ‘traversed’ in the reverse direction, e.g., the path $p = (n_2, e'_{x1}, n_3)$ is not valid, while the path $p' = (n_2, e_{x1}, n_3)$ is.

A repository schema graph contains cyclic paths, i.e., closed walks. For example, path $p = (n_7, e_7, n_8, e'_{x1}, n_2, e_{x1}, n_8, e_9, n_{10})$ in the bank schema graph (see Fig. 8) includes a cyclic subpath $p_c = (n_8, e'_{x1}, n_2, e_{x1}, n_8)$. Cyclic paths rarely have desirable semantical interpretation. In the example, the path reads as ‘person’ is related to a ‘person’ such that they both use the same ‘safe’ - in this case, it is an identity relation. A way to control the usage of cyclic paths is to define suitable constraints that fully or conditionally disallow their usage.

The precise formal definition of constraints requiring validity of paths is rather complex and would not contribute to the clarity of the explanation. We, hence, omit them from the text.

Other kinds of domain knowledge can also be represented through constraints. Constraints restrict the search space for the matching problem so may benefit the efficiency of the search process. On the other hand, if too complex, constrains introduce additional computing complexity to the problem solver.

4.5 Search space size for an XML schema matching problem

Variables $X$, variable domain $D$, and constraints $C$ are the, so far defined, ingredients of the XML schema matching problem. Without an objective function those are sufficient to treat XML
schema matching problem as a constraint satisfaction problem (i.e., CSP), discussed in section 3.2.1. CSP is solved by enumerating all the correct solutions. In XML schema matching that would be a set of all the correct mappings of a personal schema to target schema graphs, that is, mappings of a template schema graph to partial subgraphs of the repository schema graph. The number of those mappings can be used to estimate objective function’s search space, i.e., the number of mappings amongst which the best one has to be found using the objective function.

Having the template schema graph $\hat{T}$ with $|N_{\hat{T}}|$ nodes and $|E_{\hat{T}}|$ edges, and the repository schema graph $\hat{R}$ with $|N_{\hat{R}}|$ nodes and an average number of noncyclic paths between two nodes equal $|P_{\hat{R}}|$, then the number of the correct solutions to an CSP XML schema matching problem can be calculated as $O(|N_{\hat{R}}| |N_{\hat{T}}| \times |P_{\hat{R}}| |E_{\hat{T}}|)$.

Even with the modest sizes of the personal schemas and the repository the number of correct solutions to the given CSP becomes impractical. This implies usage of the objective function, not only as a tool to rank the enumerated mappings, but also as a tool to further optimize the search algorithms, preventing the need for full search space enumeration.

5 Defining the objective function

In chapter 4 we defined the XML schema matching problem. The key component is the objective function simulating human reasoning on similarity between representations of semantical concepts behind XML schemas, as defined in the section 3.3. In this chapter, we will analyze approaches for designing such objective functions. Most of the approaches existing today are based on the usage of heuristics, which we call hints, also known as clues.

5.1 Hints

Hints are heuristic ideas on how to represent fragments of semantic reasoning using functions over syntactic constructions. They calculate the estimated semantic similarity between two representations, based on their syntactic properties.

Example 10. Consider the following hint: *If two nodes have the same name or parts of their names are the same, then they are more likely to be semantically similar, than those nodes whose names are different.*

This hint addresses two syntactic constructions, i.e., nodes, and their name property, to make a judgment on their semantic similarity. Following this hint, a node with name ‘Auto’ is more similar to another node ‘auto’, than to a node with name ‘airplane’, for example. However, this hint will also find that ‘Auto’ is more similar to node ‘Automatic’ then to node ‘Car’.

The previous example reveals the fact that a hint does not have to be completely correct. In other words, a hint is often a simplification of the semantic reasoning process. The usual approach to overcome this problem in schema matching, is to combine many different hints assuming that the majority of the hints will guide the objective function to an adequate approximation of semantic similarity.

Def. 25 - Hint

A hint in schema matching is a function, called hint function, whose parameters are syntactic constructions. The output of the function is an approximation of the semantic similarity between two semantic concepts.

The definition of a hint is in essence the same as the definition of the objective function. As discussed in the sequel, an objective function may be constructed from several hint functions.

5.1.1 Aspects of a hint

When used in a matcher of a personal schema query answering system (see section 2.2), each hint influences the behavior of the system in several ways.
The quality aspect of a hint is a property of the hint related to how useful the hint is in schema matching. If the hint provides a stable and consistent semantic approximation, then the hint will be valuable in automating the schema matching process. On the contrary, a hint with low precision for semantic comparison, will contribute less or even harm schema matching quality. The quality of hint also depends on the specific subjective properties of the personal schema and the schema repository, hence a hint behaves differently in different queries asked by different persons. A way to relax this problem is to allow users to moderate the behavior of the matcher by modifying the query and the way in which hints are used. For this to be possible the users of the system must have a transparent understanding on how hints work and a full comprehension of the matching results.

The performance aspect of a hint indicates how costly it is to use the hint in schema matching. Hints that are computationally expensive, or whose parameters are hard to extract from schema graphs, significantly degrade schema matching efficiency. In scenarios with large schema repositories, this can be a decisive factor. On the other hand, such hints can may significantly reduce the search space for a matching problem and thus improve the performance.

The query evaluation performance aspect of a hint indicates the influence of a specific hint on the query evaluation phase in PSQ. For example, if a hint favors shorter paths in schemas, the evaluation of the associated queries would probably be faster due to a reduction in path steps during executing of the query. A user may want to specify a hint that favors matches that mean fast query evaluation. The investigation of this aspect is, however, behind the scope of this report.

5.1.2 Types of hint functions

Hints are formalized through hint functions $\delta : A \rightarrow B$. Hints can be defined on different levels giving different choices for $A$. Further more, hints can quantify the matching quality in several ways, giving different choices for $B$.

The domain of a hint function

Hint functions can use as their inputs any of the syntactic constructions, and their combinations, existing in schema graphs being matched, that is, nodes, edges, paths, and the whole schema graphs. Having formalized those constructions as variables of $X$ in an COP (see Def. 11), we can more precisely say that hints use as their input a partial variable list $X' \subseteq X$ (see Def. 9). This renders $A = D'$ where $D'$ is a subset of $D$.

Example 11. Having an XML schema problem with $X = (n_1, n_2, \ldots, n_k, t_1, t_2, \ldots, t_k)$ a node level hint function can be defined as $\delta(X') := \text{WordNetDistance(name}(t_i), \text{name}(n_i))$, where $X' = (n_i)_{i \in [1, k]}$.

This hint function, as indicated with $i \in [1, k]$, can be used with any template node and the corresponding target node to calculate their approximate semantic distance. We assume here that WordNetDistance function is implemented using the WordNet lexical database.

Having in mind that $X' \subseteq X$, we can safely re-declare every hint function $\delta(X')$ to $\delta(X)$. The latter will have $D$ for its domain, but will use only the values assigned to variables existing in $X'$.

The codomain of hint functions

The approximated semantic similarity between two syntactic constructions can be expressed in several ways, possibly yielding a different hint function codomain.

- $B = \mathbb{R}$

3see http://www.cogsci.princeton.edu/~wn/
The most common choice for a codomain of a hint function is a set of real numbers. Often, $B$ is normalized to the interval $[0, 1]$.

**Example 12.** For a hint: "the shorter the path, the more similar it is to an edge", a hint function can be expressed as:

$$\delta(X') = \text{length}(p_i),$$

where $X' = (p_i)$ and function $\text{length}$ calculates the length of a path.

- $B = \{\text{true, false}\}$

A hint function can be a predicate. Predicate hints define a property that makes a target object more similar to a specific template object in comparison with other template objects not having this property.

**Example 13.** For a hint: "nodes that have an equal number of children are more similar then those that do not", a hint function can be expressed as:

$$\delta(X') = (\text{childrenNo}(t_i) = \text{childrenNo}(n_i)),$$

where $X' = (n_i)_{i \in [1, k]}$, $k = |N_T|$ and function $\text{childrenNo}$ calculates the number of children of a node.

- $B = (A, \preceq)$, where $\preceq$ is a total order on $A$

Such hint function is an ordering function. It only orders a set of target objects on their similarity with the template object without quantifying the similarity level.

**Example 14.** If the process of matching is semi-automatic, a user may be asked to assist. For example, a user may specify that the template object ‘bench’ is semantically similar to ‘chair’, ‘furniture’ and ‘object’ in the following order:

$$\delta(\text{‘bench’}, \text{‘chair’}) \preceq \delta(\text{‘bench’}, \text{‘furniture’}) \preceq \delta(\text{‘bench’}, \text{‘object’})$$

**Implementing the hint function**

A hint function can be implemented using various techniques, the most common ones are listed below.

- arithmetic expressions,
- iterative techniques,
- rule based techniques,
- machine learning approaches,
- pattern matching techniques,
- hybrid techniques.

**5.1.3 The use of properties in hint functions**

Depending on the inputs of a hint function, the properties that can be used in defining the hint function differ. The more complex the input object, the more properties are available. Some example properties are:

For nodes, available properties are: node name, type of the node content, number of children, absolute position in the schema graph, etc.

For paths, available properties are: path length, the traversal direction of the edges used in the path, the cardinality of the path steps, etc.

For schema graphs, available properties are: the size of the schema graph, the number of edges in the schema graph, the number of ‘extra’ nodes in paths compared to the template schema, etc.

Some of the mentioned properties are not readily available in the schema graph and need to be calculated at the time of the hint function evaluation. Those properties are called derived properties.
The properties are in practice divided in three groups, *textual*, *structural*, and *hybrid*. *Textual* properties are mostly related to name property of the nodes. In XML schemas, node names are the strongest indicators of the semantic concept being represented with the node.

In hints, node names can for example be used for:

- syntactic comparison, i.e., names are treated as strings and compared for, e.g., common substrings, or
- semantic comparison, when synonymy, homonymy and lists of abbreviations are used to establish semantical similarity.

*Structural* properties include the analysis of different aspects of structural organization of the schema graph. For example, siblings, parent, descendant, child, and other relations in template and target structures can be used in comparison.

In [8], one of the proposed matching approaches uses the so-called ‘NamePath’. It is a hint that creates a ‘long name’ by concatenating all the node names along a certain path. Such ‘long name’ is then used to perform complex string based comparison. This makes it *hybrid* property, an example of combined structural and textual data exploitation.

5.2 Composing hints

In order to have single objective function in an XML schema matching problem, and still use different hint functions, those hint functions have to be composed. As discussed previously, hint functions can differ in various ways, and making them work together in a meaningful way is a major problem.

In essence, we need a comprehensive algebra, that allows us to compose hint functions into new, i.e., composite, hint functions. Due to the heuristic nature of hint creation and composition, development of such algebra is a rather complex issue and is out of the scope of our research. In composing hint functions we will rely on a ‘best practices’ approach.

5.2.1 Numerical hint functions only

It is not clear how to compose functions with different kinds of codomains (see section 5.1.2) in a meaningful way. The usual approach in related research [8, 13] is to use only numerical hint functions from the start, i.e., if necessary approximate the predicate and order types with the numerical one. The downside of this function transformation is a ‘semantical error’ that can be introduced in the process. For example, transforming an order function that returns $\delta_{ord}(\text{‘tiny’}) \preceq \delta_{ord}(\text{‘very big’}) \preceq \delta_{ord}(\text{‘huge’})$ into a numerical one returning $\delta_{num}(\text{‘tiny’}) = 0.2$, $\delta_{num}(\text{‘very big’}) = 0.4$, $\delta_{num}(\text{‘huge’}) = 0.6$ is semantically misleading, as it wrongly approximates that semantic distance between ‘tiny’ and ‘very big’ is the same as semantic distance between ‘very big’ and ‘huge’. Intuitively, a more precise approximation would have $\delta_{num}(\text{‘very big’}) = 0.55$.

Numerical functions are easiest to compose. We assume numerical hint functions in the sequel.

5.2.2 Examples of arithmetic approaches to hint composition

Let $\delta_{synt}(X)$ be a function that calculates the syntactic difference of node names. Let $\delta_{card}(X)$ be a function that calculates cardinality difference between an edge of a template and a path in a target schema graphs. Both functions return values in the range $[0, 1]$. The smaller values indicate smaller difference, i.e., greater similarity.

We want to combine the two hint function to produce a composite hint function called $\delta_{comp}(X)$. Below we present some composition approaches as described in [8].

1. \textit{Min}

$$\delta_{comp}(X) = \min(\delta_{synt}(X), \delta_{card}(X))$$
Only one of the values, the smaller one, is returned. This is an optimistic approach. If the two hints represent mutually exclusive ideas, then using Min recognizes that only one of the hints can provide low distance value at a time.

2. Max
\[ \delta_{\text{comp}}(X) = \max(\delta_{\text{synt}}(X), \delta_{\text{card}}(X)) \]

This is a pessimistic approach. It assumes that hints can make error in underestimating the syntactic distance. Thus, a bigger value is selected to stay on the safe side.

3. Average
\[ \delta_{\text{comp}}(X) = \frac{1}{2} \cdot \delta_{\text{synt}}(X) + \frac{1}{2} \cdot \delta_{\text{card}}(X) \]

In here both source hints are given equal influence on the result of the composed hint.

4. Weighted composition
\[ \delta_{\text{comp}}(X, \alpha) = \alpha \cdot \delta_{\text{synt}}(X) + (1 - \alpha) \cdot \delta_{\text{card}}(X) \]

Hints are given an unequal importance in the result of the composed hint. The relative importance of the hints is controlled through an additional parameter \( \alpha \). Proper tuning of this parameter is an additional problem.

Other approaches to hint composition exist. In general, any approach that transforms inputs into outputs can be used in semantic matching.

The ultimate goal of hint composition is to end up with one hint function that incorporates all desired hints, namely the objective function. Hints and the hint composition are essential for matching quality.

In the following section we will describe a technique to evaluate the quality of an objective function.

### 5.3 Evaluation of semantic matching

Hints are used to build semantic schema matchers. A semantic schema matcher tries to approximate human reasoning on the similarity of semantic concepts. Hence, the best way to evaluate the quality of a schema matcher’s output is to use human evaluation.

In [7], a number of semantic schema matching systems are compared with respect to the techniques they use for result evaluation. One of those techniques, based on the calculation of precision and recall, is described below.

Having an XML schema matching problem \( \Pi = (T, R, M_c, S_c) \) (see Def. 13), let \( M_{\text{all}} \) be the set of all possible correct mappings of template \( T \) onto target objects \( to \) in repository \( R \).

Let \( M_{\text{system}} \) be the set of semantic mappings generated by the automatic matching system. Having human evaluators, let \( M_{\text{human}} \) be the set of all the semantically correct mappings detected by humans. Due to the human subjectivity, it is often the case that more than one person is involved in preparing \( M_{\text{human}} \) set. Set \( M_{\text{human}} \) is complete, i.e., no semantically correct mappings exist outside of this set, and no semantically incorrect mappings are included in \( M_{\text{human}} \).

Fig. 10 illustrates the sets defined above.

![Figure 10: Human and machine generated mappings](image)

**Precision** and **recall** are quality measures calculated as follows:

\[ \text{Precision} = \frac{|M_{\text{system}} \cap M_{\text{human}}|}{|M_{\text{system}}|} \]
\[ \text{Recall} = \frac{|M_{\text{system}} \cap M_{\text{human}}|}{|M_{\text{human}}|} \]
• **Precision** is the ratio between the numbers of semantically correct mappings existing in $M_{\text{system}}$ and the total number of mappings in $M_{\text{system}}$. It gives an indication of how many incorrect answers are being generated by the matcher.

\[
\text{precision} = \frac{|M_{\text{system}} \cap M_{\text{human}}|}{|M_{\text{system}}|}
\]

• **Recall** is the ratio between the number of semantically correct mappings existing in $M_{\text{system}}$ and the total number of semantically correct mappings, i.e., mappings in $M_{\text{human}}$. It gives an indication of how many correct answers are being missed by the matcher.

\[
\text{recall} = \frac{|M_{\text{system}} \cap M_{\text{human}}|}{|M_{\text{human}}|}
\]

To evaluate the quality of a matching system, both the precision and recall have to be considered. Alternatively, a combined measure of the two, can be used.

• **F-measure**

\[
\text{F-measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\]

F-measure combines the precision and recall giving them the equal importance. This is a most common variant of a more generic combining function $F_{\text{measure}}(\alpha)$, which parametrizes the relative importance between precision and recall through the parameter $\alpha$.

• **Overall**

\[
\text{Overall} = \text{recall} \cdot (2 - \frac{1}{\text{precision}})
\]

The overall measure was developed to quantify the post-match editing efforts needed to modify the generated result, i.e., $M_{\text{system}}$, into the correct result, i.e., $M_{\text{human}}$.

The process of matching result evaluation, using precision and recall, is expensive and error-prone, as it involves significant human participation. The effort to manually create the set $M_{\text{human}}$ exponentially increases (see section 4.5) with the size of the set $M_{\text{all}}$. Building the $M_{\text{human}}$ set for a repository of schemas with thousands of nodes becomes thus practically impossible. Consequently, the evaluation results, so far reported, are based on rather small sets of schemas built of at most few hundred nodes.

Using only precision and recall is not sufficient to evaluate the other aspects of matching, like the duration and the cost of the training phases, or preparation of auxiliary data for matching support. Other evaluation methods are also being used [7].

Nevertheless, schema matching systems [17, 9] report the usefulness of using automatic matching techniques based on hints, at least as a support to human matching efforts. We can conclude that a personal schema based query answering system (see chapter 2), when using the same matching techniques, provides similar matching quality and benefits.

### 6 Related research

Schema matching has been attracting significant attention [17] as it finds application in many different areas. In data warehousing: source data formats have to be matched against a data warehouse data format [16]; In enterprise application integration, legacy applications’ calls are matched against predefined B2B web-services [19]; In E-commerce, different formats for data exchange are matched against each other; In data integration systems, similar to data warehousing, various data sources are matched against one common virtual view [1, 6].

Semantic schema matching solutions roughly fall in two categories [9]: rule based and learner based solutions. In the former, different heuristics are used to design distance measuring functions for schema particles. In our PSQ framework, these functions are called hint functions. They are combined within the objective function and used to approximate semantic similarity, similar to many other system [8, 15, 13, 6, 17]. Learner based systems use machine learning techniques for semantic similarity approximation. Multiple ‘learners’ are used [9], each based on different hint. Though not currently planned, no restrictions exist for our PSQ system to implement some of its hint functions using machine learning techniques, or to use machine learning techniques to combine hint functions.
Semantic matching can benefit from techniques for structural matching. In [18] matching is performed using an approximate tree embedding algorithm. The word ‘approximate’ depicts the capability of the system to match an edge of the template tree to a path in a target tree, which is one of the capabilities of our more generic PSQ system. Structural similarity of XML documents and DTD’s as described in [5] can also be used in schema matching.

Most of the work on semantical schema matching does not use formal frameworks for schema matching. Systems are built on the intuition that syntactic schema similarity leads to semantic schema similarity, and the problems involved are not formally modeled. However, in [12] a formal definition of semantic matching is defined. In [9], the restriction of this definition is used. We have modified the definitions of [9, 12] in creating the one given in section 3.3. Going a step further, we have also defined a formal framework for defining a schema matching problem. Our framework formalizes a schema matching problem as a constraint optimization problem. This was inspired by [4] where schema matching was defined as a constraint satisfaction problem used to define problems of classic query processing. By ‘upgrading’ to a constraint optimization problem, we use the addition of the objective function to fully formalize hint based matching problems.

We use schema matching as a part of semistructured data querying. Two approaches to handling semistructured querying exist. One uses XML query languages such as XPath or XQuery [20], or those described in [10, 4] to define strict queries, that include the specification of semistructured features expected to be encountered in the answer. Such languages define a query in a form of a constrained pattern. A query evaluator retrieves all the substructures of the XML documents that fit the query pattern and satisfy the constraints. Users are expected to, at least partially, know the structure of the data used in querying, and plan for their semistructuredness. Results are usually not ranked, i.e., no means are provided to measure how well an answer fits the query. In our research we pursue a different approach, in which user does not have to know the structure of the targeted data, nor to incorporate semistructuredness in the query. Instead, the query evaluation system processes the query in a relaxed manner. Answers rarely match the query isomorphically, and the system quantifies the similarity between the query and the different answers in terms of some distance functions. We found similar reasoning in [18].

7 Conclusion and future work

This report describes a novel querying approach for the XML-Web, based on the usage of a personal schema and a personal query. Users are liberated from the need to know the structure of the available XML data, by providing their self-defined virtual XML views, i.e., personal schemas, over which they can ask personal queries. The personal schema based query answering system (i.e., PSQ) is responsible for providing them with answers to such queries. We have shown that PSQ can be considered as an extension of a traditional Internet search engine. Through PSQ we have identified the importance of the ad-hoc semantic XML schema matching, i.e., matching of personal schemas to large XML schema repositories. We have placed schema matching in the focus of this report.

XML schema matching attracted much attention and several matching systems exist. Matching solutions were developed using different kinds of heuristics, but usually, without a prior formal definition of the problem they are solving. They focus on improving matching quality with the efficiency aspect ignored. In this report, we have introduced a framework, that can formally specify the ingredients of an XML schema matching problem encountered by the PSQ. We have combined and extended the existing related research and created a formal representation of the XML schema matching problem in the form of a constraint optimization problem (i.e., COP), known in the theory of constraint programming [14]. Matching problems can now be defined through variables, variable domains, constraints and an objective function. We have separately described the objective function and the, so called, hint functions, in chapter 5, as these capture most of the heuristics involved in defining the matching problem. Accompanied with a flexible graph based model for representing schema information, the COP based formalism creates a complete framework for defining the semantic XML schema matching problem. This is a valuable
tool for the development of the efficient XML schema matchers, i.e., development of the efficient solving methods for an XML schema matching problem. That is our next investigation effort.

The formal problem representation increased our understanding of how schema matching problem maps to a generic optimization problems, consequently opening a path to the adoption of different existing optimization algorithms. Clear differentiation between the components of the matching problem, helped understand how each of them influences matching performance. It also guided the initial design of the schema matcher prototype we are currently building.

Future work

Our future work is directed toward the development of the efficient XML schema matching system as needed in the PSQ. Naively, such system could solve the schema matching problem using the *generate–test–rank* loop. All possible mappings would first be generated, then tested for their compliance with the constraints, and then ranked using the objective function. Having to search for a solution in a large XML schema repository, the exponential complexity of this approach, makes it useless. The research challenge is to select a solving method that will significantly reduce the search space. Generators must be ‘intelligent’ to avoid the generation of the mappings that will not comply with constraints, or those that will rank low. Ad-hoc computation should be minimized by using precomputed data. Precomputed data should guide the generators to more efficiently traverse the search space.

Most of the needed ‘intelligence’ comes from the hint functions described in this report. Especially the monotonic hint functions, like those based on path length, can be used to quickly cut the search space. For such hint functions appropriate bounding functions can be developed and used as described in the Branch and Bound algorithm [11]. The non-monotonic hint functions can be handled through the application of non-deterministic algorithms. Related issues of variable and value ordering greatly influence the solving efficiency and will be investigated. We plan to investigate hints based on length, cardinalities, path direction and keywords.

Having large amounts of stored XML schema data, including the precomputed data derived from schemas, the need for the application of fast data access methods is apparent. Having a complete schema matching method, we will investigate the appropriate data structures for improving this, lower level, efficiency. A limiting factor might be the need to have the repository data structures incrementally updateable, due to dynamic nature of the XML-Web.

References


