Understanding The Schema Matching Problem

Alsayed Algergawy, Eike Schallehn, Gunter Saake
Institute for Technical and Business Information Systems
Department of Computer Science
Otto-von-Guericke University
39016 Magdeburg
Germany

Abstract: Schema matching plays the central role in many applications that require interoperability between heterogeneous data sources. The best way to attain comprehensive understanding of the schema matching problem is to construct a complete, if possible, problem formulation. Schema matching has been intensively researched and many matching systems have been developed. However, specifications of the schema matching problem being solved by these systems do not exist, or if it exists do not take uncertainty problems into account. In this paper, we propose the use of the *fuzzy constraint problem (FCP)* as a framework to model and understand the schema matching problem. In an effort to achieve more generic approach, we first transform the schema matching problem into a graph matching problem by transforming schemas to be matched into a common model namely rooted labeled graphs. Then, with the aid of this common model, we formulate the graph matching problem into a fuzzy constraint problem. By formalizing the schema matching problem as a *FCP*, we could express it as a combinatorial problem with soft constraints which enables us dealing with inherent uncertainty in schema matching.

Key–Words: Schema matching, Rooted-labeled graphs, Constraint programming, Fuzzy constraints, Objective functions

1 Introduction

The rapid increase of information and communication technologies has made accessible large amount of information stored in different application-specific databases and web sites. The number of different information sources is rapidly increasing and the problem of semantic heterogeneities is becoming more and more severe. Semantic heterogeneity is the ambiguous interpretation of concepts, describing the meaning of data in heterogenous data sources [18, 22]. Schema matching plays the central role in solving the problem of semantic heterogeneities. *Schema matching is the task of identifying semantic correspondences between elements of two schemas*.

Schema matching is a critical and fundamental step in many data application scenarios [11]: in *data integration*, to identify and characterize inter-schema relationships between multiple (heterogeneous) schemas; in *data warehousing*, to map data sources to warehouse schema; in *E-business*, to help map messages between different XML formats; and in the *semantic web*, to establish semantic correspondences between concepts of different websites ontologies.

A first step in finding an effective and efficient way to solve any difficult problem is to construct a complete, possibly formal, problem specification. A suitable and precise definition of schema matching is essential for investigating approaches to solve it. Schema matching has been extensively researched, and many matching systems have been developed. Some of these systems are rule-based [4, 11, 13] and the other are learner-based [10, 5, 6]. However, formal specifications of problems being solved by these systems do not exist, or are partial. A little work is done towards schema matching problem formulation. Some of these works are found in [21, 17]. Hence, to understand and handle the complexity of the schema matching problem and to be able to advise an efficient algorithm to cope with the matching problem, a formal problem specification is required.

In the common rule-based approaches, a graph is used to describe the state of a modeled system at a given time, and graph rules are used to describe the operations on the system's state. As a consequence in practice, using graph rules has a worst complexity which is exponential to the size of the graph. Of course, an algorithm of exponential time complexity is unacceptable for serious system implementation. In general, to achieve acceptable performance it is inevitable to consequently exploit the special properties

of both schemas to be matched. Beside that, there is a striking commonality in all rule-based approaches; they are all based on *backtracking paradigm*. Knowing that the overwhelming majority of theoretical as well as empirical studies on the optimization of backtracking algorithms is based on the context of *constraint problem (CP)*, it is near to hand to open this knowledge base for schema matching algorithms by reformulating the problem as a CP [20, 12].

Due to the complexity of schema matching, it was performed manually by a human observer. However, manual reconciliation tends to be slow and inefficient process especially in dynamic environments such as the semantic web. Therefore, the need for automatic semantic schema matching has become essential. Consequently, many schema matching systems have been developed for automating the process, such as Cupid [11], COMA [4], and LSD [5, 6]. Manual semantic matching overcomes mismatches which exists in element names and also differentiates between differences of domains. Hence, we could assume that manual matching is a perfect process. In the other hand, automatic matching may carry with it a degree of uncertainty, as it is based on syntactic, rather than semantic, means. Therefore, we should model and represent uncertainty in the schema matching process.

In this paper, we propose the use of *fuzzy constraint problem (FCP)*, one specific type of the constraint problem [8], to formulate the schema matching problem. However, the schemas to be matched are represented in different data models. Therefore, we first transform these schemas into a common data model called rooted labeled graphs. Then reformulate the graph matching problem as a constraint problem. The main benefit of this approach is that we gain direct access to the rich research findings in the *CP* area; instead of inventing new algorithms for graph matching from scratch. Another important advantage is that the actual algorithm solution becomes independent of the concrete graph model, allowing us to change the model without affecting the algorithm by introducing a new level of abstraction. Moreover, formalizing the schema matching problem as a *FCP* facilitates handling inherent uncertainty in the schema matching process.

The rest of the paper is organized as follows: Section 2 introduces necessary preliminaries concerning graphs and constraint programming. Our framework to unify schema matching is presented in Sect. 3 to show the scope of this paper and presents how to transform schemas to be matched into schema graphs. Section 4 shows how to formulate the schema matching problem as a constraint problem. The concluding remarks and ongoing future work are presented in Sect. 5.

2 Preliminaries

This paper is based mainly on two existing bodies of research, namely *graph theory* [2] and *constraint programming* [20, 12].

2.1 Graph Model

A *schema* is the description of the structure and the content of a model and consists of a set of related elements such as tables, columns, classes, or XML elements and attributes. There are many kinds of data models, such as relational model, object-oriented model, ER model, XML schema, etc. By schema structure and schema content, we mean its schema-based properties and its instance-based properties, respectively. In this subsection we present formally rooted (multi-)labeled directed graphs used to represent schemas to be matched as the internal common model.

A rooted labeled graph is a directed graph such that nodes and edges are associated with labels, and in which one node is labeled in a special way to distinguish it from the graph's other nodes. This special node is called the root of the graph. Without loss of generality, we shall assume that every node and edge is associated with at least one label: if some nodes (resp. edges) have no label, one can add an extra anonymous label that is associated with every node (resp. edge). More formally, we can define the labeled graph as follows:

Definition 1: *(Rooted Labeled Graph)* A rooted labeled graph G is a 6-tuple *G = (N_G, E_G, Lab_G, src, tar, l)* where:

- $N_G = \{n_{root}, n_2, ..., n_n\}$ is a finite set of nodes, each of them is uniquely identified by an object identifier (OID), where n_{root} is the graph root and satisfies the condition $parten(n_{root}) = NULL$.
- $E_G = \{(n_i, n_j) | n_i, n_j \in N_G\}$ is a finite set of edges, each edge represents the relationship between two nodes.
- Lab_G ={ Lab_{NG}, Lab_{EG} } is a finite set of node labels Lab_{NG} , and a finite set of edge labels Lab_{EG}. These labels are strings for describing the properties (features) of nodes and edges.
- *src* and *tar*: $E_G \mapsto N_G$ are two mappings (source and target), assigning a source and a target node to each edge (i.e. if $e = (n_i, n_j)$ then $src(e) = n_i$ and $tar(e) = n_j$).

- $l : N_G \cup E_G \mapsto Lab_G$ is a mapping label assigning a label from the given Lab_G to each node and each edge.
- $|N_G| = n$ is the graph size.

Now that we have defined a concrete graph model, in the following subsection we present basics of constraint programming

2.2 Constraint Programming

A lot of problems in computer science, most notably in artificial intelligence, can be interpreted as special cases of constraint problems. Semantic schema matching is also an intelligent process which aims at mimicking the behavior of human in finding semantic correspondences between elements of two schemas. Therefore, the constraint programming is a suitable scheme to interpret and understand the schema matching problem.

Constraint programming is a generic framework for declarative description and effective solving for large, particulary combinatorial, problems. Not only it is based on a strong theoretical foundation but also it is attracting widespread commercial interest as well, in particular, in areas of modeling heterogeneous optimization and satisfaction problems. There are two branches of constraint programming, namely *constraint satisfaction* and *constraint solving*. We, here, concentrate only on constraint satisfaction problem *(CSP)* and present definitions for *CSPs*, constraints, and solution for the *CSPs*.

Definition 2: *(Constraint Satisfaction Problem)* A constraint satisfaction problem **P** is defined by a 3-tuple **P**=*(X,D,C)* where,

- $X = \{x_1, x_2, ..., x_n\}$ is a finite set of *variables*,
- $D = \{D_1, D_2, ..., D_n\}$ is a collection of finite *domains*. Each domain D_i is the set containing the possible values for the corresponding variable $x_i \in X$,
- $C = \{C_1, C_2, ..., C_m\}$ is a set of *constraints* on the variables of X.

Definition 3: *(Constraint)* A constraint C_s on a set of variables $S = \{x_1, x_2, ...x_r\}$ is a relation on the product of these variables' domains: $C_s \subseteq D_1 \times ... \times D_r \rightarrow \{0, 1\}$. The number *r* of variables a constraint is defined upon is called arity of the constraint.

A constraint comprises the values a variable is allowed to take with respect to other variables. The simplest type is the unary constraint, which restricts the value of a single variable. Of special interest are the constraints of arity two, called *binary constraints*. A constraint that is defined on more than two variables is called a *global constraint*.

Solving a *CSP* is finding assignments of values from the respective domains to the variables so that all constraints are satisfied.

Definition 4: *(Solution of a CSP)* An assignment Λ is a solution of a *CSP* if is satisfies all the constraints of the problem, where the assignment Λ denotes an assignment of each variable x_i with the corresponding value a_i and $x_i \in X$ and $a_i \in D_i$.

In the schema matching field, we do not need to find any solution but the best solution. The quality of solution is usually measured by an application dependent function called objective function. The goal is to find such solution that satisfies all the constraints and minimize or maximize the objective function respectively. Such problems are referred to as *Constraint Optimization Problems (COP)*.

Definition 5: *(Constraint Optimization Problem)* A constraint optimization problem **Q** is defined by couple **Q** =*(P,g)* such that **P** is a CSP and $g : D_1 \times ... \times D_n \rightarrow [0, 1]$ is an objective function that maps each solution tuple into a value.

While powerful, both CSP and COP present some limitations. In particular, all constraints are considered mandatory. In many real problems, such as dealing with uncertainty, often appear constraints that could be violated in solutions without causing such solutions to be unacceptable. If these constraints are treated as mandatory, this often causes problems to be unsolved. If these constraints are ignored, solutions of bad quality are found. This is a motivation to extend the CSP schema and make use of *soft constraints*. We differentiate between different types of constraints. A constraint is crisp when it is either completely satisfied or completely violated. A constraint is a fuzzy when it is allows for intermediate satisfaction degrees [14, 8].

Definition 6: *(Fuzzy Constraint)* A fuzzy constraint C_μ is represented by the fuzzy relation R_f, defined by $\mu_R : \prod_{x_i \in var(C)} D_i \rightarrow [0, 1]$ where μ_R is the membership function indicating to what extent a tuple *v* satisfies C_μ,

- $\mu_R(v) = 1$ means *v* totaly satisfies C_μ,
- $\mu_R(v) = 0$ means *v* totaly violates C_μ,
- $0 < \mu_R(v) < 1$ means *v* partially satisfies C_μ.

Obviously, crisp constraints are included in the model, involving values 0 and 1 only.

Definition 7: *(Fuzzy Constraint Optimization Problem)* A fuzzy constraint optimization problem (FCOP) is a 4-tuple $\boldsymbol{Q}_\mu$= *(X, D, Cμ, g)* where *X* is a list of variables, *D* is a list of domains of possible values for the variables, C_μ is a list of fuzzy constraints each of them referring to some of the given variables, and *g* is an objective function to be optimized.

The following examples illustrate the above definitions.

Example 8: *(CSP: Map Coloring)* We want to color the regions of a map in a way that no two adjacent regions have the same color. The actual problem is that only a certain limited number of colors is available. Let's we have four regions and only three colors. We now formulate this problem as CSP ***P***=*(X,D, C)* where

- $X = \{x_1, x_2, x_3, x_4\}$ represents the four regions,
- $D = \{D_1, D_2, D_3, D_4\}$ represents the domains of the variables such that $D_1 = D_2 = D_3 = D_4 = \{red, green, blue\}$, and
- $C = \{C_{(x_1,x_2)}, C_{(x_1,x_3)}, C_{(x_1,x_4)}, C_{(x_2,x_3)}, C_{(x_2,x_4)}, C_{(x_3,x_4)}\}$ represents the constraints which should be satisfied such that $C_{(x_1,x_2)} = \{(v_1, v_2) \in D_1 \times D_2 | v_1 \neq v_2\}$. The other constraints are defined in the same way.

Example 9: *(FCOP)* Consider we have three variables each has two available values *a* and *b*. It is required to find the best 3-character word subjected to the following fuzzy constraints $C_{\mu(x_1,x_2)} = \{(v_1, v_2) \in D_1 \times D_2 | \mu_R(a,a) = 0, \mu_R(b,b) = 0.7, \mu_R(a,b) = 1, \mu_R(b,a) = .5\}$, $C_{\mu(x_2,x_3)} = \{(v_2, v_3) \in D_2 \times D_3 | \mu_R(a,a) = 0.3, \mu_R(b,b) = 1, \mu_R(a,b) = 0.1, \mu_R(b,a) = 1\}$. This problem can be formulated as FCOP $\boldsymbol{Q}_\mu$= *(X, D, C_μ, g)* where:

- $X = \{x_1, x_2, x_3\}$ represents the three characters,
- $D = \{D_1, D_2, D_3\}$ represents the doamins of the variables such that $D_1 = D_2 = D_3 = \{a, b\}$
- $C_\mu = \{C_{\mu(x_1,x_2)}, C_{\mu(x_2,x_3)}\}$ represents the fuzzy constraints defined above, and
- *g* is the objective function to detemine the best solution

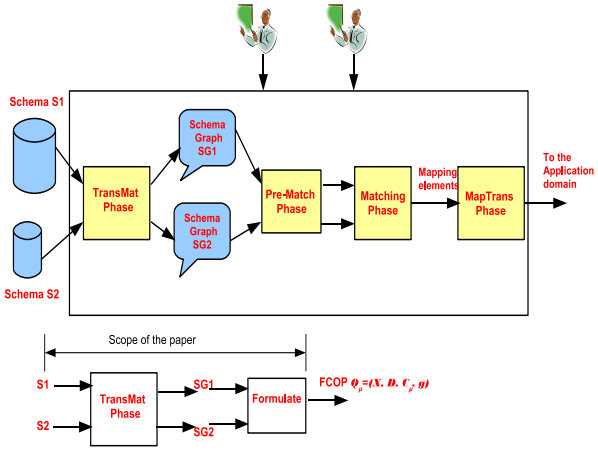

Figure 1: Matching Process Phases

The mentioned examples are used to explain how to formulate different problems from different domains as constraint problems. In the following section we shed the light on our unified schema matching framework to determine the scope of schema matching understanding.

3 A Unified Schema Matching Framework

Each of the existing schema matching systems deals with the schema matching problem from its point of view, but we need a generic framework that unifies the solution of this intricate problem independent on the domain of schemas to be matched and independent on the model representations. Therefore, we suggest the following general phases to address the schema matching problem. Figure 1 shows these phases with the main scope of this paper. The four different phases are: importing the schemas to be matched; *TransMat Phase*, identifying the elements to be matched; *Pr-matching Phase*, applying the matching algorithm; *Matching Phase*, and exporting the match result; *MapTrans Phase*.

In the following subsection we introduce a framework for defining different data models and how to transform them into schema graphs. This part follows the same procedure found in [21] to show that different data models could be represented by schema graphs. More details about our framework is found in [1]

3.1 Schema Graph

To make the matching process more generic process, the schemas to be matched should be represented

internally by a common representation. This uniform representation reduces the complexity of matching process by not having to cope with different schema representations. By developing such import tools, schema match implementation can be applied to schemas of any data model such as *SQL*, *XML*, *UML*, and etc. Therefore, the first step in most schema matching approaches is to transform the schemas to be matched to a common model in order to applying the matching algorithm. Most of these approaches choose graph data structure as the internal representation [4, 3, 7, 11, 13, 16]. The choice of graph as an internal representation for schemas to be matched has many motivations. First, graphs are well-known data structure and have its algorithms and implementations. Second, by using the graph as a common data model, the schema matching problem is transformed into another standard problem graph matching problem. We also make use of rooted labeled graphs as the internal model for the schemas to be matched. We call this phase as *TransMat*; Transformation for Matching process.

In general, to represent the schemas and data instances, starting from the root, the schema is partitioned into relations (elements) and further down into attributes and data instances. In particular, to represent relational schemas, XML schemas, etc. as rooted labeled graphs, independently of the specific source format, we benefit from the rules found in [21, 15, 9]. These rules are:

- Every prepared matching object in a schema such as schema, relations, elements, attributes etc. is represented by a node in the schema graph, such that the schema itself is represented by the root node. Let schema S consists of m elements *(elem)*, then
  $\forall elem \in S \; \exists n \in N_G$ and $S \mapsto n_{root}$
- The features of the prepared matching object are represented by node labels Lab_{NG}. Let features *(featS)* are the properties set of an element *(elem)*, then
  $$\forall feat \in featS \; \exists Lab \in Lab_{NG}$$
- The relationship between two prepared matching objects is represented by an edge of the schema graph. Let the relationships between schema elements are *(relS)*, then
  $$\forall rel \in relS \; \exists e(n_i, n_j) \in E_G \text{ such that } src(e) = n_i \in N_G \text{ and } tar(e) = n_j \in N_G$$
- The properties of the relationship between prepared objects are represented by edge labels Lab_{EG}, then, $\forall rfeat \in rfeatS \; \exists Lab \in Lab_{EG}$

Create Table *Personnel* (
Pno int primary key,
Pname string,
Dept string,
Born date
)

Schema *S*

Create Table *Employee* (
EmpNo int primary key,
EmpName varchar(20),
DeptNo int **REFERENCES** *Department*,
Salary int,
BirthDate date
)

Create Table *Department* (
DeptNo int primary key,
DeptName varchar(30)
)

Schema *T*

Figure 2: Two Relational Schemas

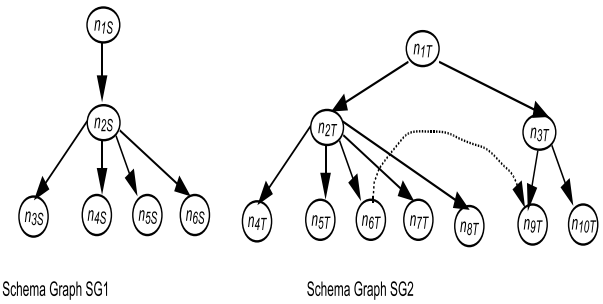

Figure 3: Schema Graphs of Two Relational Schemas(without labels)

Example 10: *(Relational Database Schemas)* Consider schemas S and T depicted in Fig. 2 (from [13]). The elements of *S* and *T* are tables and attributes. From Fig.1 and applying the above encoding rules, we obtain the schema graphs SG1 and SG2. Therefore, two relational schemas are transformed into schema graphs, thus *Schema S* and *Schema T* are represented by *SG1* and *SG2* respectively, such that $SG1 = (N_{GS}, E_{GS}, Lab_{GS}, src_S, tar_S, l_S)$ where

- $N_{GS} = \{n_{1S}, n_{2S}, n_{3S}, n_{4S}, n_{5S}, n_{6S}\}$
- $E_{GS} = \{e_{1-2}, e_{2-3}, e_{2-4}, e_{2-5}, e_{2-6}\}$
- $Lab_{GS} = Lab_{NS} \cup Lab_{ES} = \{name, type, datatype\} \cup \{part - of\}$, Table 1 represents these labels.
- src_S, tar_S, l_S are mappings such that $src_S(e_{1-2}) = n_{1S}$, $tar_S(e_{2-3}) = n_{3S}$ and $l_S(e_{1-2}) = part - of$

Figures 3 shows only the nodes and edges of the schema graphs (SG2 can be defined similarly).

Example 11: *(XML Schemas)* The relational data model does not capture all the features of semi-structured or unstructured data. Semi-structured data does not possess regular structure as well as missing

Table 1: The Labels of two Schema Graph Nodes

OID	Labels of Schema Graph SG1			Labels of Schema Graph SG2		
	name	*data type*	*type*	*name*	*data type*	*type*
1	Schema S	-	schema	Schema T	-	schema
2	Personnel	-	table	Employee	-	table
3	Pno	int	attribute	Department	-	table
4	Pname	string	attribute	EmpNo	int	attribute
5	Dept	string	attribute	EmpName	varchar(20)	attribute
6	Born	date	attribute	DeptNo	int	attribute
7				Salary	int	attribute
8				BirthDate	date	attribute
9				DeptNo	int	attribute
10				DeptName	varchar(30)	attribute

```
<Schema name="S" xmlns="urn:schemas-
microsoft-com:xml-data ">
 <ElementType name="AccountOwner">
  <element type="Name"/>
  <element type="Address"/>
  <element type="Birthdate"/>
 </ElementType>
 <ElementType name="Address">
  <element type="street"/>
  <element type="city"/>
  <element type="state"/>
  <element type="ZIP"/>
 </ElementType>
</Schema>
```

```
<Schema name="T" xmlns="urn:schemas-
microsoft-com:xml-data ">
 <ElementType name="Customer">
  <element type="FName"/>
  <element type="LName"/>
  <element type="CAddress"/>
 </ElementType>
 <ElementType name="CAddress">
  <element type="street"/>
  <element type="city"/>
  <element type="province"/>
  <element type="code"/>
 </ElementType>
</Schema>
```

Figure 4: Two XML Schemas

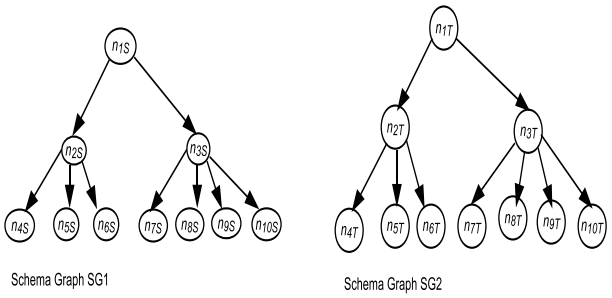

Figure 5: Schema Graphs of Two XML Schemas(without labels)

or duplicated fields are allowed. Typical examples are HTML and XML.

This example that we discuss illustrate how our unified schema matching framework copes with different choices of the models to be matched. Now two XML schemas in Fig. 4 (from [21]). The schemas are specified using the XML language deployed on the website *biztalk.org* designed for electronic documents used in e-business. The schema graphs (without labels) of these schemas are shown in Fig.5. The labels of nodes and edges are the same as Example 10.

Examples 10 and 11 illustrate that using *TransMat* phase aims at matching different schema models. The matching algorithm (*Matching Phase*) does not have to deal with a large number of different models. The matching algorithm only deals with the internal representation. So far, recent schema matching systems directly determine semantic correspondences between two schemas elements as a graph matching. In this paper, we extend the internal representation, schema graphs, and reformulate the graph matching problem as a constraint problem.

4 Schema Matching as a FCOP

The goal of schema matching is to identify the semantic correspondences between elements of two schemas. We will illustrate the process of transforming the schema matching problem into a FCOP. We first describe the schema matching problem as a graph matching problem and then reformulate it as a constraint problem.

4.1 Schema Matching as Graph Matching

The schemas to be matched are transformed into rooted labeled graphs and, hence, the schema matching problem is converted into graph matching. Two types of graph matching exist *isomorphism* and *homomorphism*. In general, a match of one graph into another is given by a *graph morphism*, which is a mapping of one graph's object sets into the other's, with some restrictions to preserve the graph's structure and its typing information.

Definition 12: *(Graph Morphism)* A graph morphism $\phi : SG1 \rightarrow SG2$ between two schema graphs $SG1 = (N_{GS}, E_{GS}, Lab_{GS}, src_S, tar_S, l_S)$ and $SG2 = (N_{GT}, E_{GT}, Lab_{GT}, src_T, tar_T, l_T)$ is a pair of mappings $\phi = (\phi_N, \phi_E)$ such that $\phi_N : N_{GS} \rightarrow N_{GT}$ (ϕ_N is a node mapping function) and

$\phi_E : E_{GS} \rightarrow E_{GT}$ (ϕ_E is an edge mapping function) and the following restrictions apply:

1. $\forall n \in N_{GS} \; \exists \, l_S(n) = l_T(\phi_N(n))$
2. $\forall e \in E_{GS} \; \exists \, l_S(e) = l_T(\phi_E(e))$
3. $\forall e \in E_{GS} \; \exists$ a path $p' \in N_{GT} \times E_{GT}$ such that $p' = \phi_E(e)$ and $\phi_N(src_S(e)) = src_T(\phi_E(e)) \wedge \phi_N(tar_S(e)) = tar_T(\phi_E(e))$.

The first two condition preserve both nodes and edges labeling information, while the third condition preserves the structure of the graph.

The graph matching is an isomorphic matching problem when $|N_{GS}| = |N_{GT}|$ otherwise it is homomorphic. Obviously, the schema matching problem is homomorphic problem.

Definition 12 reflects only one type of matching cardinality called *individual matching* (one-to-one). Individual matching implies that one graph object from the first schema graph is associated with one graph object from the other schema graph. The other type of matching cardinality is *complex matching* which implies one (or a set) graph object(s) of one schema may be associated with a set of graph objects of the other schema.

Example 13: For the two relational database schemas depicted in Fig. 2 and its associated schema graphs shown in Fig. 3, the schema matching problem between schema *S* and schema *T* is converted into a homomorphic graph matching problem between schema graph *SG1* and schema graph *SG2*.

Graph matching is considered to be one of the most complex problems in computer science. Its complexity is due to two major problems. The first problem is the computational complexity of graph matching. The time required by backtracking in a search tree algorithms may in the worst case become exponential in the size of the graph. The second problem is the fact that all of the algorithms for graph matching mentioned so far can only be applied to two graphs at a time. Therefore, if there is more than two schemas that must be matched, then the conventional graph matching algorithms must be applied to each pair sequentially. For applications dealing with large databases, this may be prohibitive. Graph homomorphisms have been proven to be NP-complete problem [19]. Hence, choosing graph matching as platform to solve the schema matching problem may be effective process but inefficient. Therefore, we propose transforming graph homomorphism into a *FCOP*.

Now that we have defined a graph model and its homomorphism, let us consider how to construct a *FCOP* out of a given graph matching problem.

4.2 Graph Matching as a FCOP

In the schema matching problem, we are trying to find a mapping between the elements of two schemas. Multiple conditions should be applied to make these mappings valid solutions to the matching problem, and some objective functions are to be optimized to select the best mappings among matching result. The analogy to constraint problem is quite obvious: here we make a mapping between two sets, namely between a set of variables and a set of domains, where some conditions should be satisfied. So basically, what we have to do to obtain an equivalent constraint problem CP for a given schema matching problem (knowing that schemas to be matched are transformed into schema graphs) are:

1. take the objects of one schema graph to be matched as the CP's set of variables,
2. take the objects of the other schema graph to be matched as the variables' domain,
3. find a proper translation of the conditions that apply to a schema matching into a set of constraints, and
4. form the objective functions to be optimized.

We have defined the schema matching problem as a graph matching homomorphism ϕ. We now proceed by formalizing the problem ϕ as a FCOP problem $Q_\mu = (X, D, C_\mu, g)$. To construct a FCOP out of this problem, we follow the above rules. Through these rules, we take the two relational database schemas shown in Fig. 2 and its associated schema graphs shown in Fig. 3 as an example, taking into account that $|N_{GS}|(= 6) < |N_{GT}|(= 10)$

- The set of variables X is given by $X = N_{GS} \cup E_{GS}$ where the variables from N_{GS} are called *node variables* X_N and from E_{GE} are called *edge variables* X_E
  $X = X_N \cup X_E = \{x_{n1}, x_{n2}, x_{n3}, x_{n4}, x_{n5}, x_{n6}\} \cup \{x_{e12}, x_{e23}, x_{e24}, x_{e25}, x_{e26}\} = \{x_{n1}, x_{n2}, x_{n3}, x_{n4}, x_{n5}, x_{n6}, x_{e12}, x_{e23}, x_{e24}, x_{e25}, x_{e26}\}$
- The set of domain D is given by $D = N_{GT} \cup E_{GT}$, where the domains from N_{GT} are called *node domains* D_N and from E_{GT} are called *edge domains* D_E

  $D = D_N \cup D_E = \{D_{n1}, D_{n2}, D_{n3}, D_{n4}, D_{n5}, D_{n6}\} \cup \{D_{e_{1-2}}, D_{e_{2-3}}, D_{e_{2-4}}, D_{e_{2-5}}, D_{e_{2-6}}\}$

$= \{D_{n1}, D_{n2}, D_{n3}, D_{n4}, D_{n5}, D_{n6}, D_{e_{1-2}}, D_{e_{2-3}}, D_{e_{2-4}}, D_{e_{2-5}}, D_{e_{2-6}}\}$
where $D_{n1} = D_{n2} = D_{n3} = D_{n4} = D_{n5} = D_{n6} = \{n_{1T}, n_{2T}, n_{3T}, n_{4T}, n_{5T}, n_{6T}, n_{7T}, n_{8T}, n_{9T}, n_{10T}\}$ (i.e.*node domain* contains all the second schema graph nodes) and $D_{e_{1-2}} = D_{e_{2-3}} = D_{e_{2-4}} = D_{e_{2-5}} = D_{e_{2-6}} = \{e_{1-2T}, e_{1-3T}, e_{2-4T}, ..., p_{1-2-4T}, ...\}$ (i.e. *edge domain* contains all the available edges and paths in the second schema graph)(the edge e_{1-2} reads the edge extends between the two nodes n_1 and n_2 such that $e_{1-2} = e(n_1, n_2)$).

- Still missing in our *FCOP* are the proper set of constraints to enforce matching properties on the solutions and a set of objective functions to determine a measure of how good a solution is.

In the following subsections, we demonstrate how to construct both constraints and objective functions to obtain a complete problem definition.

4.3 Constraints Construction

For constraints to be of most use, they should reflect the goals of schema matching. Schema matching based only on schema element properties has been attempted. However, it does not provide any facility to optimize matching. Furthermore, additional constraint information, such as semantic relationships and other domain constrains is not included, and schemas may not completely capture the semantics of data they describe. Therefore, in order to improve the correctness of matching, additional information should be included. In this paper, we are concerned with both syntactic and semantic matching. Therefore, we could classify constraints that should be incorporated in the *CP* model into: *syntactic constraints* and *semantic constraints*. In the following, we consider only the constraints construction while the fuzzy relations of fuzzy constraint are not consider since it depends on the application domain. For example, as shown below, domain constraints are crisp constraints, i.e. $\mu_C(v) = 1$, while the structural constraints are soft constraints with different degree of satisfaction.

Syntactic Constraints

1. Domain Constraint: It states that a node variable must be assigned a value (or a set of values) from a node domain, and an edge variable must be assigned a value from the edge domain. That is $\forall x_{ni} \in X_N$ and $x_{ej} \in X_E \exists$ a unary constraint $C^{dom}_{\mu(x_{ni})}$ and $C^{dom}_{\mu(x_{ei})}$ ensuring domain consistency of the match, where
   $C^{dom}_{\mu(x_{ni})} = \{d_i \in D_{Ni}\}$
   $C^{dom}_{\mu(x_{ei})} = \{d_i \in D_{Ei}\}$
2. Structural Constraints: There are many structural relationships between nodes in schema graphs such as:
   - Edge Constraint: It states that if an edge exists between two variable nodes, then an edge (or path) should exist between their corresponding images. That is, $\forall x_{ei} \in X_E$ and its source and target nodes are x_{ns} and x_{nt} $\exists$ two binary constraints $C^{src}_{\mu(x_{ei},x_{ns})}$, $C^{tar}_{\mu(x_{ei},x_{nt})}$ representing the structural behavior of matching, where:
     $C^{src}_{\mu(x_{ei},x_{ns})} = \{(d_i, d_j) \in D_E \times D_N | src(d_i) = d_j\}$
     $C^{tar}_{(x_{ei},x_{nt})} = \{(d_i, d_j) \in D_E \times D_N | tar(d_i) = d_j\}$
   - Parent Constraint: $\forall$ two variable nodes x_{ni} and $x_{nj} \in X_N \exists$ a binary constraint $C^{parent}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of parent relationship, where
     $C^{parent}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N | \exists e(d_i, d_j) s.t. src(e) = d_i\}$
   - Child Constraint: $\forall$ two variable nodes x_{ni} and $x_{nj} \in X_N \exists$ a binary constraint $C^{child}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of child relationship, where
     $C^{child}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N | \exists e(d_i, d_j) s.t. tar(e) = d_i\}$
   - Sibling Constraint: $\forall$ two variable nodes x_{ni} and $x_{nj} \in X_N \exists$ a binary constraint $C^{sibl}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of parent relationship, where
     $C^{sibl}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N | \exists d_n s.t. parent(d_n, d_i) \wedge paren(d_n, d_j)\}$

Semantic Constraints

1. Labeled Constraints: $\forall x_i \in X \exists$ a unary constraint $C^{Lab}_{\mu(x_i)}$ ensuring the semantics of the predicates in the schema such that:
   if $x_i \in X_N : C^{Lab}_{\mu(x_i)} = \{d_j \in D_N | l_S(x_i) = l_T(d_j)\}$
   if $x_i \in X_E : C^{Lab}_{\mu(x_i)} = \{d_j \in D_E | l_S(x_i) = l_T(d_j)\}$

The above syntactic and semantic constraints are by no means the contextual relationships between elements. 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, constraints introduce additional computational complexity to the problem solver.

4.4 Objective Function Construction

The objective function is the function to be optimized depending on the object parameters (also referred to as search space parameters). The objective function constitutes the implementation of the problem to be solved. The input parameters are the object parameters. The output is the objective value representing the evaluation/quality of the individual. In the schema matching problem, the objective function simulates human reasoning on similarity between schema graph objects. Most of the recent approaches are based on the usage of heuristics called clues.

A clue is the methodology of how to compute semantic similarity between schema graph objects based on their features. Thus, a clue is a function whose inputs are the features of schema graph objects and its output is the semantic similarity. Obviously, each clue influences schema matching performance. If a clue produces a high precision (recall) matching result, it would be improve schema matching effectiveness and if a clue needs expensive computations, it would regrade schema matching efficiency. Therefore, the design of clues is an intractable process.

Definition 14: *(Clue Function)* A clue function f_c is a function which determines the semantic similarity between schema graph objects based on their features. $f_c : i_c \mapsto [0, 1]$ s.t. $i_c \subseteq D$.
The clue function f_c can exploit different methods and techniques such as arithmetic expression, string functions, iterative techniques, machine learning approaches, ect. The input of f_c makes use of schema graph object features including labels of nodes Lab_N and labels of edges Lab_E. While, the output of this fuction can be quantified in different quantities for example numerical, logical. But the most used is the numerical which is normalized to the interval [0,1].

In order to get an objective function from different used clues functions, those clues function have to be composed. Many approaches to clue composition exist. In general, any approach that transforms inputs into outputs can be used in semantic similarity.

5 Summary and Future Work

Schema matching is a fundamental process in many domains dealing with shared data such as data integration, data warehouse, E-commerce, semantic query processing, and the web semantics. Matching solutions were developed using different kind of heuristics, but usually without prior formal definition of the problem they are solving. Although many matching systems have been developed and different approaches are proposed to solve the schema matching problem, but no complete work to address the formulation problem. Schema matching research mostly focuses on how well schema matching systems recognize corresponding schema elements. On the other hand, not enough research has been done on formal basics of the schema matching problem.

In this paper, we have introduced a formal definition for the schema matching problem. This definition is based on transforming the schema matching problem into graph matching by representing schemas to be matched as schema graphs. Instead of solving the graph matching problem which has been proven to be NP-complete problem, we reformulate it as a constraint problem. Therefore, the schema matching problem can be understood as searching values for variables from finite domains when the given constraints are (partially) satisfied and an objective function is optimal. We have identified two types of constraints syntactic and semantic to ensure match semantics. We also shed lights on how to construct objective functions.

The main benefit of this approach is that we gain direct access to the rich research findings in the CP area; instead of inventing new algorithms for graph matching from scratch. Another important advantage is that the actual algorithm solution becomes independent of the concrete graph model, allowing us to change the model without affecting the algorithm by introducing a new level of abstraction.

Understanding schema matching problem is considered the first step towards an effective and efficient solution for the problem. In our ongoing work, we will exploit constraint solver algorithms to reach to our goal.

References:

[1] A. Algergawy, E. Schallehn, and G. Saake. A unified schema matching framework. In *19. GI-Workshop on Foundations of Databases*, pages 58–62. Bretten, Germany, May 2007.

[2] R. Babakrishnan and K.Ranganathan. *A textbook of graph theory*. Spring Verlag, 1999.

[3] H. H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. In *Proceedings of the 2nd Int. Workshop on Web Databases*, October 2002.

[4] H. H. Do and E. Rahm. Coma- a system for flexible combination of schema matching approaches. In *P. A. Bernstein et al., editors, VLDB 2002: proceedings of the Twenty-Eighth International Conference on Very Large Data Bases*, pages 610–621, August 2002.

[5] A. Doan. Learning to map between structured representations of datag. In *Ph.D Thesis*. Washington University, 2002.

[6] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD*, pages 509–520, May 2001.

[7] A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AAAI AI Magazine, Special Issues on semantic Integration*, 25(1):83–94, 2005.

[8] D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence*, 6:287–309, 1996.

[9] F. Giunchiglia and P. Shvaiko. Semantic matching. *KER Journal*, 18(3), 2003.

[10] W. Li and C. Clifton. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data and Knowledge Engineering*, 33:49–84, 2000.

[11] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB '01*, pages 49–58. Roma, Italy, Sept. 2001.

[12] K. Marriott and P. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.

[13] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, February 2002.

[14] P. Meseguer, N. Bouhmala, T. Bouzoubaa, M. Irgens, and M. Sanchez. Current approaches for solving over-constrained problems. *Constraints*, 8:9–39, 2003.

[15] L. Palopoli, D. Rossaci, G. Terracina, and D. Ursino. A graph-based approach for extracting terminological properties from information sources with heterogeneous formats. *Knowledge and Information Systems*, 8:462–497, 2005.

[16] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large Data Bases*, 10(4):334–350, Dec. 2001.

[17] M. Smiljanic. *XML Schema Matching Balancing Efficiency and Effectiveness by means of Clustering*. PhD thesis, Twente University.

[18] X. Sun and E. Rose. Automated schema matching techniques: An exploratory study. *Res. Lett. Inf. Math. Sci.*, 4:113–136, 2004.

[19] R. K. Swarup Medasani and Y. Choi. Graph matching by relaxation of fuzzy assignments. *IEEE TRANSACTIONS ON FUZZY SYSTEMS*, 9(1):173–182, FEBRUARY 2001.

[20] E. Tsang. *Foundations of Constraint Satisfaction*. Acadimic Press, 1993.

[21] Z. Zhang, H. Che, P. Shi, Y. Sun, and J. Gu. Formulation schema matching problem for combinatorial optimization problem. *Interoperability in Business Information systems (IBIS)*, 1(1):33–60, 2006.

[22] H. Zhao. Semantic matching across heterogeneous data sources. *Communication of the ACM*, 50(1):45–50, January 2007.