Fuzzy Constraint-based Schema Matching Formulation

Alsayed Algergawy, Eike Schallehn, and Gunter Saake

Department of Computer Science,
Otto-von-Guericke University,
39106 Magdeburg, Germany
{alshahat,eike,saake@iti.cs.uni-magdeburg.de}

Abstract. The deep Web has many challenges to be solved. Among them is schema matching. In this paper, we build a conceptual connection between the schema matching problem *SMP* and the *fuzzy constraint optimization problem FCOP*. In particular, we propose the use of the *fuzzy constraint optimization problem* as a framework to model and formalize the schema matching problem. By formalizing the *SMP* as a *FCOP*, we gain many benefits. First, we could express it as a combinatorial optimization problem with a set of soft constraints which are able to cope with uncertainty in schema matching. Second, 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, we could discover complex matches easily. Finally, we could make a trade-off between schema matching performance aspects.

Key words: Schema matching, Constraint programming, Fuzzy constraints, Objective functions.

1 Introduction

The number of deep Web sources has increased rapidly [3]. To open the deep Web to users software systems are needed to enable users to explore and integrate deep Web sources. Schema matching is the core task of these systems.

Schema matching is the task of identifying semantic correspondences among elements of two or more schemas. It plays a central role in many data application scenarios [12]: in *data integration*, to identify and characterize inter-schema relationships between multiple (heterogeneous) schemas; in *data warehousing*, to map data sources to a warehouse schema; in *E-business*, to help to map messages between different XML formats; in the *Semantic Web*, to establish semantic correspondences between concepts of different web sites ontologies; and in *data migration*, to migrate legacy data from multiple sources into a new one [9].

Due to the complexity of schema matching, it was mostly performed manually by a human expert. However, manual reconciliation tends to be a slow and

inefficient process especially in large-scale and dynamic environments. Therefore, the need for automatic schema matching has become essential. Consequently, many schema matching systems have been developed for automating the process, such as Cupid [12], COMA [5], LSD [6], BTreeMatch [10], and Spicy [2]. Manual semantic matching overcomes mismatches which exist in element names and also differentiates between differences of domains. Hence, we could assume that manual matching is a perfect process. On the other hand, automatic matching may carry with it a degree of uncertainty, as it is based on syntactic, rather than semantic, means. Furthermore, recently, there has been renewed interest in building database systems that handle uncertain data in a principled way. Hence a short rant about the relationship between databases that manage uncertainty and data integration systems appears. Therefore, we should surf for a suitable model which is able to meet the above requirements.

A first step in discovering an effective and efficient way to solve any difficult problem such as schema matching is to construct a complete 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 [5, 12, 14] and others are machine learning-based [6, 7]. However, formal specifications of problems being solved by these systems do not exist, or are partial. Little work is done towards schema matching problem formulation e.g. in [18, 16].

In the 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 case 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 paradigms*. 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 schema matching problem as a CP [17, 13, 4].

In this paper, we build a conceptual connection between the schema matching problem *(SMP)*and the *fuzzy constraint optimization problem (FCOP)*. On one hand, we consider schema matching as a new application of fuzzy constraints; on the other hand, we propose the use of the fuzzy constraint satisfaction problem as a new approach for schema matching. In particular, in this paper, we propose the use of the *FCOP* to formulate the *SMP*. However, our approach should be generic, i.e. have the ability to cope with different data models and be used for different application domains. Therefore, we first transform schemas to be matched into a common data model called rooted labeled graphs. Then we reformulate the graph matching problem as a constraint problem. There are many

benefits behind this formulation. First, we gain direct access to the rich research findings in the *CP* area; instead of inventing new algorithms for graph matching from scratch. Second, 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. Third, formalizing the *SMP* as a *FCOP* facilitates handling uncertainty in the schema matching process. Finally, we could simply deal with simple and complex mappings.

The paper is organized as follows: Section 2 introduces necessary preliminaries. Our framework to unify schema matching is presented in Sect. 3 to show the scope of this paper. 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* [1] and *constraint programming* [4, 13]. To keep this paper self-contained, we briefly present in this section the basic concepts of them.

2.1 Graph Model

In this subsection we present formally rooted (multi-)labeled graphs used to represent schemas to be matched. More formally, we can define the labeled graph as follows:

Definition 1. *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.* $E_G = \{(n_i, n_j) | n_i, n_j \in N_G\}$ *is a finite set of edges.* $Lab_G = \{ Lab_{NG}, Lab_{EG} \}$ *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 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. And* $l : N_G \cup E_G \mapsto Lab_G$ *is a mapping label assigning a label from the given* Lab_G *to each node and each edge.*

2.2 Constraint Programming

Many 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 humans in finding semantic correspondences between two schemas' elements. Therefore, constraint programming is a suitable scheme to represent the SMP.*

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. We, here, concentrate only on constraint satisfaction problems *(CSPs)* and present definitions for *CSPs*, constraints, and solutions for the *CSPs*.

Definition 2. *A Constraint Satisfaction Problem* $\boldsymbol{P}$ *is defined by a 3-tuple* $\boldsymbol{P}$*=(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* $x_i \in X$*, and* $C = \{C_1, C_2, ..., C_m\}$ *is a nonempty, finite set of constraints on the variables of X.*

Definition 3. *A Constraint* C_s *on a set of variables* $S = \{x_1, x_2, ...x_r\}$ *is a pair* $C_s = (S, R_s)$*, where* R_s *is a subset on the product of these variables' domains:* $R_s \subseteq D_1 \times ... \times D_r \rightarrow \{0, 1\}$.

The number r of variables a constraint is defined upon is called arity of the constraint. 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*.

Example 1. (Map Coloring:) Let us assume we have a map comprising n countries. We want to color each country using one of four colors: *red*, *green*, *white*, or *blue* in a way that no two adjacent countries have the same color. This problem could be formulated as CSP $\boldsymbol{P}$*=(X,D, C)* where: $X = \{x_1, x_2, ..., x_n\}$ represents n countries, $D = \{D_1, D_2, ..., D_n\}$ represents the domains of the variables such that $D_1 = D_2 = ... = D_n = \{red, green, blue, white\}$, and Crepresents constraints which should be satisfied such that $C_{(x_i,x_j)} = \{(v_i, v_j) \in D_i \times D_j | v_i \neq v_j\}$.

Solving a *CSP* is finding assignments of values from the respective domains to the variables so that *all constraints* are satisfied. However, in the schema matching field, we do not need to find any solution but the best solution. The quality of a solution is usually measured by an application dependent function called objective function. The goal is to find such a solution that satisfies all the constraints and minimizes or maximizes the objective function respectively. Such problems are referred to as *Constraint Optimization Problems (COP).*

Definition 4. *A Constraint Optimization Problem* $\boldsymbol{Q}$ *is defined by couple* $\boldsymbol{Q}$ *=(P,g) such that* $\boldsymbol{P}$ *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, there are 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*. A way to circumvent inconsistent constraints problems is to make them fuzzy. The idea is to associate fuzzy values with the elements of the constraints, and combine them in a reasonable way.

A constrain, as defined before, is usually defined as a pair consisting of a set of variables and a relation on these variables. This definition gives us the availability to model different types of uncertainty in schema matching. In [8], authors identify different sources for uncertainty in data integration. Uncertainty in semantic mappings between data sources can be modeled by exploiting fuzzy relations while other sources of uncertainty can be modeled by making the variable set a fuzzy set. In this paper, we take the first one into account while the other sources are left for our ongoing work.

Definition 5. *A Fuzzy Constraint C_μ on a set of variables $S = \{x_1, x_2, ...x_r\}$ is a pair $C_\mu = (S, R_\mu)$, where the fuzzy relation R_μ, 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, while $0 < \mu_R(v) < 1$ means v partially satisfies C_μ.*

Definition 6. *A Fuzzy Constraint Optimization Problem $\boldsymbol{Q}_\mu$ 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.*

In the following section we shed light on our schema matching framework to determine the scope of schema matching understanding.

3 A Unified Schema Matching Framework

Most of existing schema matching systems deal 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 of the domain of schemas to be matched and independent of the model representations. To this end, we propose the following general phases to address the schema matching problem. Figure 1 shows these phases with the main scope of this paper. In the following subsection we introduce a framework for defining different data models and how to transform them into schema graphs.

3.1 Schema Graph

To make the matching process a more generic process, schemas to be matched should be represented internally by a common representation. This uniform representation reduces the complexity of the matching process by not having to cope with different representations. By developing such import tools, schema match

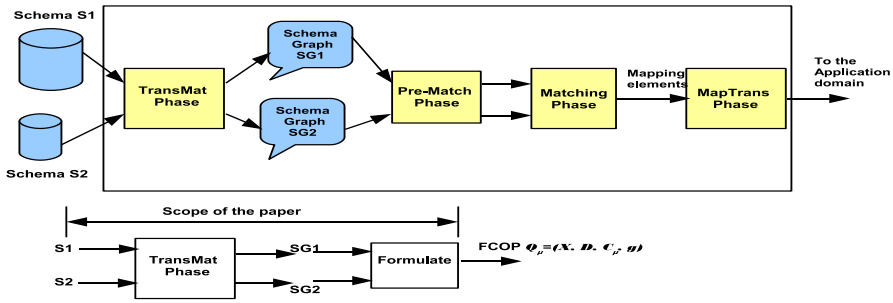


Fig. 1: Matching Process Phases

implementation can be applied to schemas of any data model such as *SQL*, *XML*, *UML*, and etc. Therefore, the first step in our approach is to transform schemas to be matched to a common model in order to apply matching algorithms. We make use of labeled graphs as the internal model. We call this phase *TransMat*; Transformation for Matching process.

In general, to represent schemas and data instances, starting from the root, the schema is partitioned into relations and further down into attributes and 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 [18, 15, 11]. These rules are rewritten as follows:

- Every prepared matching object in a schema such as the schema, relations, elements, attributes etc. is represented by a node, such that the schema itself is represented by the root node. Let schema S consist of m elements *(elem)*, then

$$\forall\ elem \in S\ \exists\ n_i \in N_G \wedge S \mapsto n_{root},\ s.t.\ 1 \leq i \leq m$$

- The features of the prepared matching object are represented by node labels Lab_{NG}. Let features *(featS)* be the property set of an element *(elem)*, then

$$\forall\ feat \in featS\ \exists\ Lab \in Lab_{NG}$$

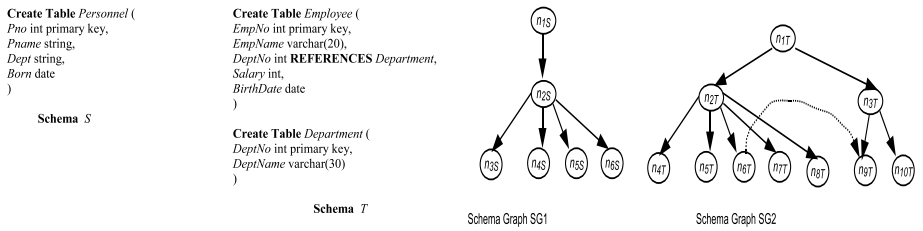
- The relationship between two prepared matching objects is represented by an edge. Let the relationships between schema elements be *(relS)*, then

$$\forall\ rel \in relS\ \exists\ e(n_i,\ n_j) \in E_G \text{ s. t. } src(e) = n_i \in N_G \wedge tar(e) = n_j \in N_G$$

- The properties of the relationship between prepared objects are represented by edge labels Lab_{EG}. Let features *rfeatS* be the property set of a relationship *rel*, then,

$$\forall\ rfeat \in rfeatS\ \exists\ Lab \in Lab_{EG}$$

Example 2. (Relational Database Schemas) Consider schemas S and T depicted in Fig. 2(a) (from [14]). The elements of S and T are tables and attributes. Applying the above rules, the two schemas *Schema S* and *Schema T* are represented



(a) Two Relational Schemas (b) Schema Graphs

Fig. 2: Two Relational Schemas & their Schema Graphs (without labels)

by *SG1* and *SG2* respectively, such that $SG1 = (N_{GS}, E_{GS}, Lab_{GS}, src_S, tar_S, l_S)$ where $N_{GS} = \{n_{1S}, n_{2S}, n_{3S}, n_{4S}, n_{5S}, n_{6S}\}$, $E_{GS} = \{e_{1-2}, e_{2-3}, e_{2-4}, e_{2-5}, e_{2-6}\}$, $Lab_{GS} = Lab_{NS} \cup Lab_{ES} = \{name, type, datatype\} \cup \{part-of, associate\}$. 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 2(b) shows only the nodes and edges of the schema graphs (SG2 can be defined similarly).

In this example, we exploit different features of matching objects such as *name, datatype,* and *type*. These features are represented as nodes' labels. These features shall be the input parameters to the next phase. For example, the *name* of a matching object in SG1 will be used to measure linguistic similarity between it and another matching object from SG2, its *datatype* is to measure datatype compatibility, and its *type* is used to determine semantic relationships. However, our approach is flexible in the sense that it is able to exploit more features as needed. Moreover, in this example, we exploit one structural feature *"part-of"* to represent structural relationships between nodes at different levels. Other structural features e.g. association relationship, that is a structural relationship specifying both nodes are conceptually at the same level, are represented between keys. One association relationship is represented in Fig. 2(b) between the nodes n_{6T} and n_{9T} to specify a key/foreign key relation. Visually, association edges are represented as dashed lines.

So far, recent schema matching systems directly determine semantic correspondences between 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

4.1 Schema Matching as Graph Matching

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 7. *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 conditions preserve both nodes and edges labeling information, while the third condition preserves graph's structure.

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

Example 3. For the two relational schemas depicted in Fig. 2(a) and its associated schema graphs shown in Fig. 2(b), the schema matching problem between schema S and schema T is converted into a homomorphic graph matching problem between *SG1* and *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 algorithm 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. Hence, choosing graph matching as a platform to solve the schema matching problem may be effective process but inefficient. Therefore, we propose transforming graph homomorphism into a *FCOP*.

4.2 Graph Matching as a FCOP

In the schema matching problem, we are trying to find a mapping among 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 the 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 to a certain extent. In order to obtain an equivalent constraint problem CP for a given schema matching problem (assuming that schemas to be matched are transformed into schema graphs) we utilize the followings rules:

1. take objects of one schema graph to be matched as the CP's set of variables,
2. take objects of other schema graphs to be matched as the variables' domain,
3. find a proper translation of the conditions that apply to schema matching into a set of fuzzy constraints, and
4. form 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(a) and its associated schema graphs shown in Fig. 2(b) 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

$$X = X_N \cup X_E$$
$$= \{x_{n1}, x_{n2}, x_{n3}, x_{n4}, x_{n5}, x_{n6}\} \cup \{x_{e_{1-2}}, x_{e_{2-3}}, x_{e_{2-4}}, x_{e_{2-5}}, x_{e_{2-6}}\}$$

– 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,

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

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. *the 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. *the 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 $e_{1-2} = e(n_1, n_2)$).

Using this formalization enables us to deal with holistic matching. This can be achieved by taking the objects of one schema as the variable set, while the objects of other schemas are the variable's domain. Let we have n schemas which are transformed into schema graphs *SG1, SG2,...,SGn* then $X = X_N \cup X_E$, $D_N = \sum_{i=2}^{n} D_{Ni}$, $D_E = \sum_{i=2}^{n} D_{Ei}$. Another benefit behind this approach is that our approach is able to discover complex matchings of types *1:n* and *n:1* very easily.

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

4.3 Constraints Construction

The exploited constraints 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 constraints is not included, and schemas may not completely capture the semantics of data they describe. Therefore, in order to improve performance and correctness of matching, additional information should be included. In this paper, we are concerned with both syntactic and semantic matching. Therefore, we shall classify constraints that should be incorporated in the *CP* model into: *syntactic constraints* and *semantic constraints*.

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:
   $$C^{src}_{\mu(x_{ei},x_{ns})} = \{(d_i, d_j) \in D_E \times D_N |\ src(d_i) = d_j\ \}$$
   $$C^{tar}_{(x_{ei},x_{nt})} = \{(d_i, d_j) \in D_E \times D_N |\ tar(d_i) = d_j\}$$
   - $\forall$ two variable nodes x_{ni} and $x_{nj} \in X_N\ \exists$ a set of binary constraints as follows:
     a) Parent Constraint $C^{parent}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of parent relationship, where
     $$C^{parent}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N |\ \exists\ e\ (d_i,\ d_j)\ s.t.\ src(e) = d_i\ \}$$
     b) Child Constraint $C^{child}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of child relationship, where
     $$C^{child}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N |\ \exists\ e\ (d_i,\ d_j)\ s.t.\ tar(e) = d_j\ \}$$
     c) Sibling 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 $x_i \in X_N$ and if $x_i \in X_E$:

$$C^{Lab}_{\mu(x_i)} = \{d_j \in D_N |\ lsim(l_S(x_i), l_T(d_j)) \geq t\ \}$$
$$C^{Lab}_{\mu(x_i)} = \{d_j \in D_E |\ lsim(l_S(x_i), l_T(d_j)) \geq t\},$$

   where *lsim* is a similarity function determining the semantics similarity between nodes/edges labels such name and t is a predefined threshold.

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. Moreover, each constraint is associated with a membership function $\mu(v) \in [0, 1]$ to indicate to what extent the constraint should be satisfied. If $\mu(v) = 0$, this means v totally violates the constraint and $\mu(v) = 1$ means v totally satisfies it. 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 associated with an optimization process which determines how good a solution is and depends on the object 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.

In this framework, we should consider two function components which constitute the objective function. The first is called *cost function* f_{cost} which determines the cost of a set constraint over variables. The second is called *energy function* f_{energy} which maps every possible variable assignment to a cost. Then, the objective function could be expressed as follows:

$$g = min|max(\sum_{setofconstraint} f_{cost} + \sum_{setofassignment} f_{energy})$$

5 Summary and Future Work

In this paper, we have introduced a fuzzy constraint-based framework to model the schema matching problem. Our approach is able to handle uncertainty in schema matching by exploiting fuzzy constraints. Moreover, our framework is generic which has the feature to deal with different schema representations by transforming the schema matching problem into graph matching. Instead of solving the graph matching problem which has been proven to be an NP-complete

problem, we reformulate it as a constraint problem. We have identified two types of constraints syntactic and semantic to ensure match semantics. We also shed light 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 the 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 our goal.

References

1. R. Babakrishnan and K.Ranganathan. *A textbook of graph theory*. Spring Verlag, 1999.
2. A. Bonifati, G. Mecca, A. Pappalardo, and S. Raunich. The spicy project: A new approach to data matching. In *SEBD*. Turkey, 2006.
3. S. C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, 2004.
4. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
5. H. H. Do and E. Rahm. COMA- a system for flexible combination of schema matching approaches. In *VLDB 2002*, pages 610–621, 2002.
6. A. Doan. Learning to map between structured representations of datag. In *Ph.D Thesis*. Washington University, 2002.
7. A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD*, pages 509–520, May 2001.
8. X. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB'07*, pages 687–698, 2007.
9. C. Drumm, M. Schmitt, H.-H. Do, and E. Rahm. Quickmig - automatic schema matching for data migration projects. In *Proc. ACM CIKM07*. Portugal, 2007.
10. F. Duchateau, Z. Bellahsene, and M. Roche. An indexing structure for automatic schema matching. In *SMDB Workshop*. Turkey, 2007.
11. F. Giunchiglia and P. Shvaiko. Semantic matching. *KER Journal*, 18(3), 2003.
12. J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB 2001*, pages 49–58. Roma, Italy, 2001.
13. K. Marriott and P. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, 1998.
14. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE'02*, 2002.
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. M. Smiljanic. *XML Schema Matching Balancing Efficiency and Effectiveness by means of Clustering*. PhD thesis, Twente University, 2006.
17. E. Tsang. *Foundations of Constraint Satisfaction*. Acadimic Press, 1993.
18. Z. Zhang, H. Che, P. Shi, Y. Sun, and J. Gu. Formulation schema matching problem for combinatorial optimization problem. *IBIS*, 1(1):33–60, 2006.