Scalable Computing: Practice and Experience
Volume 9, Number 4, pp. 303–314. http://www.scpe.org
ISSN 1895-1767

FUZZY CONSTRAINT-BASED SCHEMA MATCHING FORMULATION

ALSAYED ALGERGAWY, EIKE SCHALLEHN, AND GUNTER SAAKE*

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 function

1. Introduction. The deep Web (also known as Deepnet or the hidden Web) refers to the World Wide Web content that is not a part of the surface Web. It is estimated that the deep Web is several orders of magnitude larger than the surface Web [4]. As the number of deep Web sources has been increasing as the efforts needed to enable users to explore and integrate these sources become essential. As a result software systems have been developed to open the deep Web to users. *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 [22, 17]: 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 [10].

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 [17], COMA/COMA++ [6, 1], LSD [8], Similarity Flooding [20], OntoBuilder [13], QOM [12], BTreeMatch [11], S-Match [14], and Spicy [3]. 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 [9]. 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 [6, 17, 20] and others are learning-based [16, 7, 8]. 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 [25, 23].

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

*Department of Computer Science, Magdeburg University, 39106 Magdeburg, Germany (Alshahat|eike|sake@ovgu.de).

to hand to open this knowledge base for schema matching algorithms by reformulating the schema matching problem as a CP [24, 18, 5].

To summarize, we are in a need to a framework which is able to face the following challenges:

1. *formalizing the schema matching problem*: Although many matching systems have been developed 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 correspondences. On the other hand, not enough research has been done on formal basics of the schema matching problem.
2. *trading-off between schema matching performance aspects*: The performance of a schema matching system comprises two equally important factors; namely *matching effectiveness* and *matching efficiency*. The effectiveness is concerned with the accuracy and the correctness of the match result while the efficiency is concerned with the system resources such as the response time of the match system. Recent schema matching systems report considerable effectiveness [6], however, the efficiency aspects remain a missing area and represent an open challenge for the schema matching community. Improving schema matching efficiency results in decreasing matching effectiveness, so a trade-off between the two aspects should be considered.
3. *dealing with uncertainty of schema matching:* Schema matching systems should be able to handle uncertainty arises during the matching process from different sources. Recently, there has been renewed interest in building database systems that handle uncertain data and its lineage in a principled way, so a short rant about the relationship between databases that manage uncertainty and lineage and data integration systems appears. In addition to, in order to fully automate the matching process, we make use of extractor tools which extract different data models and represent them as a common model. The extraction process brings errors and uncertainties to the matching process

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 Section 3 in order to illustrate the scope of this paper. Section 4 shows how to formulate the schema matching problem as a constraint problem. Section 5 describes the related work. The concluding remarks and ongoing future work are presented in Section 6.

2. Preliminaries. This paper is based mainly on two existing bodies of research, namely graph theory [2] and constraint programming [24, 18, 5]. To keep this paper self-contained, we briefly present in this section the basic concepts of them.

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 2.1. *A Rooted Labeled Graph G is a 6-tuple G = (N_G, E_G, Lab_G, src, tar, l) where:*

- $N_G = \{n_{root}, n_2, \ldots, n_n\}$ *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, 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. 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 intelligence process which aims at mimicking the behavior of humans in finding semantic correspondences between schemas' elements. Therefore, constraint programming is a suitable scheme to represent the schema matching problem.*

Constraint programming is a generic framework for declarative description and effective solving for large, particularly 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 2.2. *A Constraint Satisfaction Problem* **P** *is defined by a 3-tuple* **P**=*(X,D,C) where,*

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

DEFINITION 2.3. *A Constraint* C_s *on a set of variables* $S = \{x_1, x_2, \ldots x_r\}$ *is a pair* $C_s = (S, R_s)$*, where* R_s *is a subset on the product of these variables' domains:* $R_s \subseteq D_1 \times \cdots \times D_r \rightarrow \{0, 1\}$*.*

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*.

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

DEFINITION 2.4. *(Solution of a CSP) An assignment* Λ *is a solution of a CSP if it 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 *such that* $x_i \in X$ *and* $a_i \in D_i$*.*

Example 1. (Map Coloring) We want to color the regions of a map, shown in Fig. 2.1, 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 $CSP = (X, D, C)$ 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
- $C = \{C_{(x1,x2)}, C_{(x1,x3)}, C_{(x1,x4)}, C_{(x2,x4)}, C_{(x3,x4)}\}$ represents the set of constraints must be satisfied such that $C_{(xi,xj)} = \{(vi, vj) \in Di \times Dj | vi \neq vj\}$.

As shown in Example 1, there are a number of solutions to the specified CSP. Any one of them is considered a solution to the problem. However, in the schema matching field, we do not only search for any solution but also the best one. The quality of solution is usually measured by an application dependent function called the objective function. The goal is to find such a solution that satisfies all the constraints and minimize or maximize the objective function. Such problems are referred to as constraint optimization problems (COP).

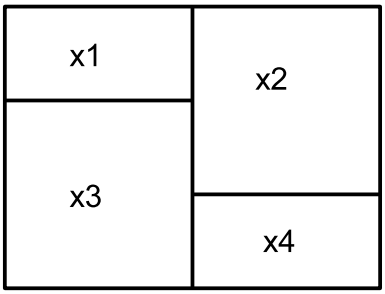


FIG. 2.1. *Map coloring example*

DEFINITION 2.5. *A Constraint Optimization Problem* $\boldsymbol{Q}$ *is defined by couple* $\boldsymbol{Q}$ *=(P,g) such that* $\boldsymbol{P}$ *is a CSP and* $g : D_1 \times \cdots \times D_n \rightarrow [0, 1]$ *is an objective function that maps each solution tuple into a value.*

Example 2. (Traveling Salesman) The traveling salesman problem is to find the shortest closed path by which city out of a set of n cities is visited once and only once.

While powerful, both CSP and COP present some limitations. In particular, all constraints are considered mandatory. In many real-world problems, such as the schema matching problem, 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 scheme and make use of soft constraints.* A way to circumvent inconsistent constraints problems is to make them fuzzy [15]. 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 [9], 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 2.6. *(Fuzzy Constraint) A Fuzzy Constraint* C_μ *on a set of variables* $S = \{x_1, x_2, \ldots, x_r\}$ *is a pair* $C_\mu = (S, R_\mu)$*, where the fuzzy relation* R_μ*, defined by* $\mu_R : \prod_{xi \in var(C)} D_i \mapsto [0, 1]$ *where* μ_R *is the membership function indicating to what extent a tuple v satisfies* C_μ*.*

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

DEFINITION 2.7. *A Fuzzy Constraint* C_μ *on a set of variables* $S = \{x_1, x_2, \ldots x_r\}$ *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 2.8. *A Fuzzy Constraint Optimization Problem* $\boldsymbol{Q}_\mu$ *is a 4-tuple* $\boldsymbol{Q}_\mu$*= (X, D,* $C\mu$*, 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 the light on our 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. As a result the need to 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 becomes essential. To this end, we suggest the following general phases to address the schema matching problem. Figure 2 shows these phases with the main scope of this paper. The four different phases are:

- importing schemas to be matched; *TransMat* Phase,

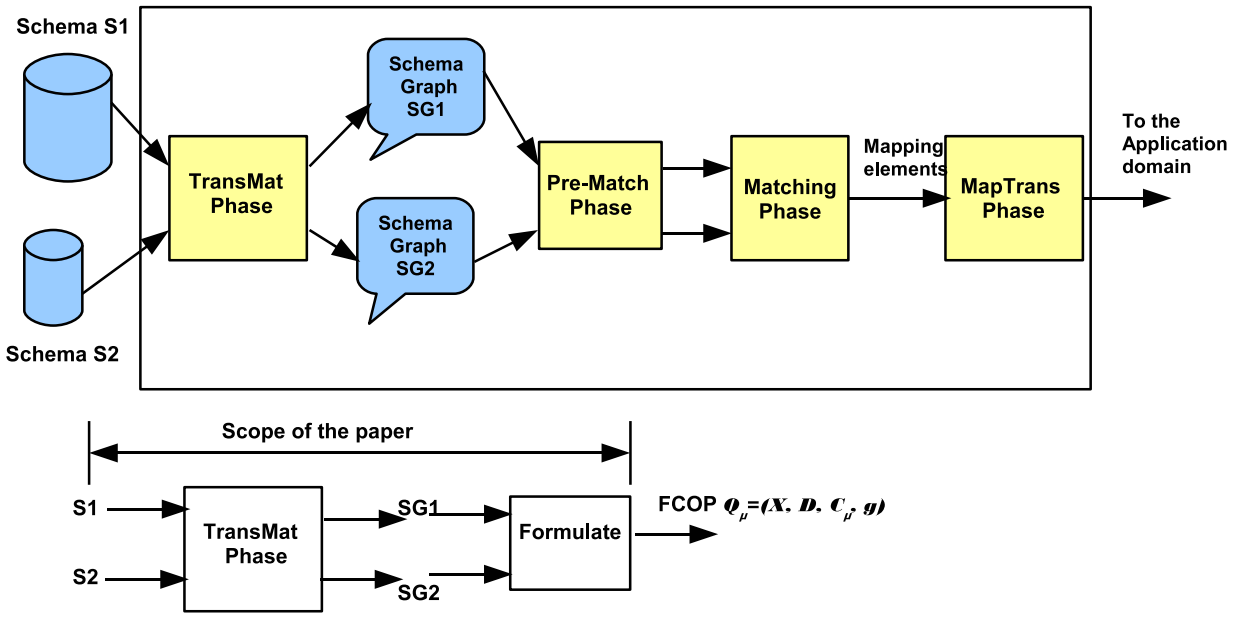


FIG. 3.1. *Matching Process Phases*

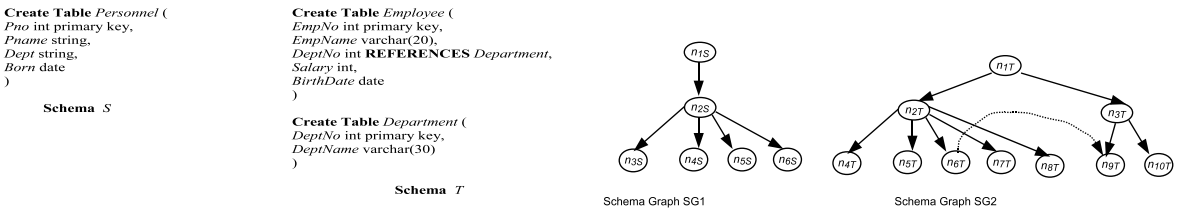
- identifying elements to be matched; *Pr-matching* Phase,
- applying the matching algorithms; *Matching* Phase, and
- exporting the match result; *MapTrans* Phase.

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 [25] to show that different data models could be represented by 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 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 into a common model in order to apply matching algorithms. We make use of rooted labeled graphs as the internal model. We call this phase *TransMat*; Transformation for Matching process.

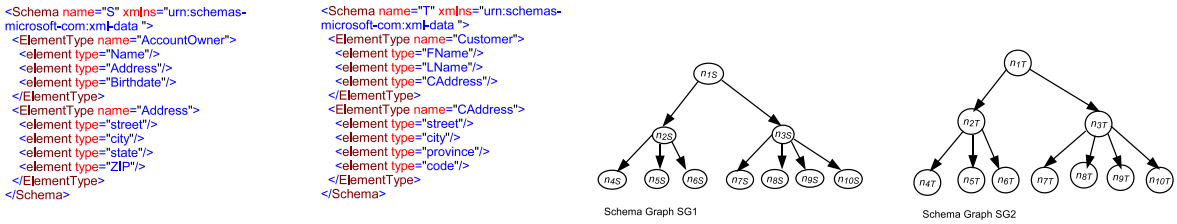
In general, to represent schemas and data instances, starting from the root, the schema is partitioned into relations and further down into attributes and instances. In particular, to represent relational schemas, XML schemas, etc. as rooted labeled graphs, independently of the specific source format, we benefit from the rules found in [25, 21]. 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,
$$\forall\ rfeat \in rfeatS\ \exists\ Lab \in Lab_{EG}$$



(a) Two relational schemas (b) Schema graphs

FIG. 3.2. *Two Relational Schemas & their Schema Graphs (without labels)*



(a) Two XML schemas (b) Schema graphs

FIG. 3.3. *Two XML schemas & their schema graphs (without labels*

The following two examples illustrate that how these rules can be applied to different data models in order to make our approach a more generic approach.

Example 3. *(Relational Database Schemas)* Consider schemas S and T depicted in Fig. 3.2(a) (from [20]). The elements of S and T are tables and attributes. Applying the above rules, the two schemas *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

$$N_{GS} = \{n_{1S}, n_{2S}, n_{3S}, n_{4S}, n_{5S}, n_{6S}\}, \quad E_{GS} = \{e_{1-2}, e_{2-3}, e_{2-4}, e_{2-5}, e_{2-6}\},$$
$$Lab_{GS} = Lab_{NS} \cup Lab_{ES} = \{name, type, data\ type\} \cup \{part\text{-}of, associate\},$$

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 $l_S(e_{1-2}) = part\text{-}of$. Figure 3.2(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, is represented between keys. One association relationship is represented in Fig. 3.2(b) between the nodes n_{6T} and n_{9T} to specify a key/foreign key relation. Visually, association edges are represented as dashed lines.

Example 4. (XML Schemas) This example that we discuss illustrates how our unified schema matching framework copes with different choices of the models to be matched. Now consider two XML schemas in Fig. 3.3(a) (from [25]). 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. 3.3(b). The labels of nodes and edges are the same as Example 3.

Examples 3 and 4 illustrate that using Trans-Mat 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 problem. In this paper,

we extend the internal representation, schema graphs, and reformulate the graph matching problem as a fuzzy constraint optimization 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. There are two types of graph matching: *graph isomorphism* and *graph 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 4.1. *A Graph Morphism* $\phi : SG1 \rightarrow SG2$ *between two schema graphs*

$$SG1 = (N_{GS}, E_{GS}, Lab_{GS}, src_S, tar_S, l_S) \text{ and } SG2 = (N_{GT}, E_{GT}, Lab_{GT}, src_T, tar_T, l_T)$$

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 5. For the two relational schemas depicted in Fig.3.2(a) and its associated schema graphs shown in Fig.3.2(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 search tree algorithms may in the worst case become exponential in the size of the graph. Graph homomorphism has been proven to be NP-complete problem [19]. 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 are 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 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 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. 3.2(a) and its associated schema graphs shown in Fig. 3.2(b) as an example, taking into account that $|N_{GS}(= 6)| < |N_{GT}(= 10)|$ as follows:

- The set of variables X is given by $X = N_{GS} \bigcup E_{GS}$ where the variables from N_{GS} are called node variables X_N and from E_{GS} are called edge variables X_E
$$X = X_N \bigcup X_E$$
$$= \{x_{n1}, x_{n2}, x_{n3}, x_{n4}, x_{n5}, x_{n6}\} \bigcup \{x_{e1-2}, x_{e2-3}, x_{e2-4}, x_{e2-5}, x_{e2-6}\}$$
- The set of domain D is given by $D = N_{GT} \bigcup E_{GT}$, where the domains from N_{GT} are called node domains D_N and from E_{GT} are called edge domains D_E,
$= \{D_{n1}, D_{n2}, D_{n3}, D_{n4}, D_{n5}, D_{n6}\} \bigcup \{D_{e1-2}, D_{e2-3}, D_{e2-4}, D_{e2-5}, D_{e2-6}\}$ 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 $D_{e1-2} = D_{e2-3} = D_{e2-4} = D_{e2-5} = D_{e2-6} =$
$\{e_{1-2T}, e_{1-3T}, e_{2-4T}, \ldots, p_{1-2-4T}, \ldots\}$ (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 as the variable's domain. Let we have n schemas which are transformed into schema graphs $SG1, SG2, \ldots, SGn$ then $X = X_N \bigcup X_E$, $D_N = \sum_{i=2}^{n} D_{Ni}$, $D_E = \sum_{i=2}^{n} D_{Ei}$. Another benefit behind this approach is that our approach is able to discover complex matches of types *1:n* and *n:1* very easily. This can be achieved by allowing a value may have multiple values from its corresponding domain and a value may be assigned to multiple variables.

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

4.3. Constraint 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*. 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.

4.3.1. Syntactic Constraints.

1. ***Domain Constraints***: It states that a node variable must be assign a value or a set of values from its corresponding node domain, and an edge variable must be assigned a value from its corresponding 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 schema graph nodes 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 $x_{nt} \in X \exists$ two binary constraints $C^{src}_{\mu(x_{ei}, x_{ns})}$ and $C^{tar}_{\mu(x_{ei}, x_{nt})}$ 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 variables nodes x_{ni} and $x_{nj} \in X \exists$ a set of binary constraints describing the hierarchical relationships between schema graph nodes as follows:

(a) *Parent Constraint* $C^{parent}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of parent relationship, where

$$C^{parent}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N \mid \exists e(d_i, d_j) \ s.t. \ src(e) = d_i\}$$

(b) *Child Constraint* $C^{child}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of child relationship, where

$$C^{child}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N \mid \exists e(d_i, d_j) \ s.t. \ tar(e) = d_j\}$$

(c) *Sibling Constraint* $C^{sibl}_{\mu(x_{ni},x_{nj})}$ representing the structural behavior of sibling relationship, where

$$C^{sibl}_{\mu(x_{ni},x_{nj})} = \{(d_i, d_j) \in D_N \times D_N \mid \exists d_n \ s.t. \ parent(d_n, d_i) \wedge parent(d_n, d_j)\}$$

4.3.2. Semantic Constraints. The first constraint type considers only the structural and hierarchical relationships between schema graph nodes. In order to capture the other features of schema graph nodes such as the semantic feature we make use of the following constraint.

1. Label Constraints: $\forall x_{ni} \in X_N$ and $\forall x_{ei} \in X_E \exists$ a unary constraint $C^{Lab}_{\mu(xni)}$ and $C^{Lab}_{\mu(xei)}$ ensuring the semantics of the predicates in the schema such that:

$$C^{Lab}_{\mu(x_{ni})} = \{dj \in DN | lsim(lS(x_{ni}), lT(d_j)) \geq t\}$$
$$C^{Lab}_{\mu(x_{ei})} = \{dj \in DE | lsim(lS(x_{ei}), lT(d_j)) \geq t\}$$

where $lsim$ is a linguistic similarity function determining the semantic similarity between nodes/edges labels 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:

$$g = \text{mis}|\max(\sum\nolimits_{\text{set of constraints}} f_{\text{cost}} + \sum\nolimits_{\text{set of assignment}} f_{\text{energy}})$$

5. Related 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, such as Cupid [17], COMA/COMA++ [6, 1], LSD [8], Similarity Flooding [20], OntoBuilder [13], QOM [12], BTreeMatch [11], S-Match [14], and Spicy [3], have been developed and different approaches have been 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.

Most of the existing work [22] define match as a function that takes two schemas (models) as input, may be in the presence of auxiliary information sources such as user feedback and previous mappings, and produces a mapping as output. A schema consists of a set of related elements such as tables, columns, classes, or XML elements and attributes. A mapping is a set of mapping elements specifying the matching schema elements together. Each mapping element is specified by 4-tuple element $\langle ID, S_i^1, S_j^2, R\rangle$ where ID is an identifier for the mapping element that matches between the element S_i^1 of the first schema and the element S_j^2 of the second one and R indicates the similarity value between 0 and 1. The value of 0 means strong dissimilarity while the value of 1 means strong similarity. But, in general, a mapping element indicates that certain element(s) of schema $S1$ are related to certain element(s) of schema $S2$. Each mapping element can have an associated mapping expression which specifies how the two elements (or more) are related. Schema matching is considered only with identifying the mappings not determining the associated expressions.

In the work of A. Doan [7], they formalize the schema matching problem as four different problems:

1. *The basic 1-1 Matching*; given two schemas S and T (representations), for each element s of S, find the most semantically similar element t of T, utilizing all available information. This problem is often referred as a one-to-one matching problem, because it matches each element s with a single element. For example, the $\langle$*ID1, S.Address, T.CAddress, 0.8*$\rangle$ mapping element indicates that there a mapping between the element S.*Address* of schema S and the element T.*CAddress* of schema T with a degree of similarity 0.8.
2. *Matching for Data Integration*; given source schemas *S1, S2,...,Sn* and mediated schema T, for each element s of Si find the most similar element t of T.
3. *Complex Matching*; let S and T be two data representations. Let O =*{O1, O2,...,Ok}* be a set of operators that can be applied to the elements of T according to a set of rules R to Figure 2: Matching Function construct formulas. For each element s of S, find the most similar element t, where t can be either an element of T or a formula from the elements of T, using O and R.
4. *Matching for Taxonomies*; given two taxonomies of concepts S and T, for each concept node s of S, find the most similar concept node of T.

For each of these problems, Doan shows input information, solution output, and the evaluation of a solution output. In general, the input to a problem can include any type of knowledge about the schemas to be matched and their domains such as schema information, instance data, previous matchings, domain constraints, and user feedback.

Zhang and et. el. [25] formulate the schema matching problem as a combinatorial optimization problem. The authors cast the schema matching problem into a multi-labeled graph matching problem. The authors propose a meta-meta model of schema: multi-labeled graph model, which views schemas as finite structures over the specific signatures. Based on this multi-labeled schema, they propose a multi-labeled graph model, which is an instance of multi-label schema, to describe various schemas, where each node and edge can be associated with a set of labels describing its properties. Then they construct a generic graph similarity measure based on the contrast model and propose an optimization function to compare two multi-labeled graphs. Using the greedy algorithm, they design an optimization algorithm to solve the multi-labeled graph matching problem.

Gal and et al. [13] propose a fuzzy framework to model the uncertainty of the schema matching process outcome. The framework aims at identifying and analyzing factors that impact the effectiveness of schema matching algorithms by reducing the uncertainty of existing algorithms. To specify their belief in the mapping quality, the authors associate a confidence measure with any mapping among attributes' sets. They use the framework to define the monotonicity property as a desired property of the schema matching problem, so one can safely interpret a high confidence measure as a good semantic mapping.

The recent work for [23] introduces a formal specification for the XML matching problem. The authors define the ingredients of the XML schema matching problem using constraint logic programming. Matching problems can be defined through variables, variable domains, constraints and an objective function. They distinguish between the constraint satisfaction problem and constraint optimization problem and show that the optimization problem is more suitable for the schema matching problem. They make use of combination of clustering methods and the branch and bound algorithm to solve the schema matching problem.

In our formulation approach, we have some common and distinct features with the other related work. The common features include transforming schemas to be matched into schema graphs, i. e. rooted labeled graphs, and making use of the constraint programming as a framework to extend the graph matching problem into a

constraint optimization problem. However, our approach introduces distinctly the use of fuzzy constraint in order to reflect the nature of the schema matching problem. As well as the use of the fuzzy constraint enables us to model uncertainty in the schema matching process.

6. Summary and Future Work. In this paper, we have investigated an intricate problem; the schema matching problem. In particular, we have introduced a fuzzy constraint-based framework to model the schema matching problem. To this end, we build a conceptual connection between the schema matching problem and fuzzy constraint optimization problem. On one hand, we consider schema matching as a new application of fuzzy constraint optimization, and on the other hand we propose the use of fuzzy constraint optimization as a new approach for schema matching.

Our proposed approach is a generic framework 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. As well as, we make use of the fuzzy constraints in order to enable us modeling uncertainty in the schema matching process. 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] D. Aumueller, H. H. Do, S. Massmann, and E. Rahm, *Schema and ontology matching with COMA++*, in SIGMOD Conference, 2005, pp. 906–908.

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

[3] A. Bonifati, G. Mecca, A. Pappalardo, and S. Raunich, *The spicy project: A new approach to data matching*, in SEBD, Turkey, 2006.

[4] S. C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, *Structured databases on the web: Observations and implications*, SIGMOD Record, 33 (2004), pp. 61–70.

[5] R. Dechter, *Constraint Processing*, Morgan Kaufmann, 2003.

[6] H. H. Do and E. Rahm, *COMA- a system for flexible combination of schema matching approaches*, in VLDB 2002, 2002, pp. 610–621.

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

[8] A. Doan, P. Domingos, and A. Halevy, *Reconciling schemas of disparate data sources: A machine-learning approach*, SIGMOD, (2001), pp. 509–520.

[9] X. Dong, A. Halevy, and C. Yu, *Data integration with uncertainty*, in VLDB'07, 2007, pp. 687–698.

[10] C. Drumm, M. Schmitt, H.-H. Do, and E. Rahm, *Quickmig - automatic schema matching for data migration projects*, in Proc. ACM CIKM07, Portugal, 2007.

[11] F. Duchateau, Z. Bellahsene, and M. Roche, *An indexing structure for automatic schema matching*, in SMDB Workshop, Turkey, 2007.

[12] M. Ehrig and S. Staab, *QOM- quick ontology mapping*, in International Semantic Web Conference, 2004, pp. 683–697.

[13] A. Gal, A. Tavor, A. Trombetta, and D. Montesi, *A framework for modeling and evaluating automatic semantic reconciliation*, VLDB Journal, 14 (2005), pp. 50–67.

[14] F. Giunchiglia, M. Yatskevich, and P. Shvaiko, *Semantic matching: Algorithms and implementation*, Journal on Data Semantics, IX (2007).

[15] H. W. Guesgen and A. Philpott, *Heuristics for fuzzy constraint satisfaction*, in ANNES '95, 1995, pp. 132–135.

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

[17] J. Madhavan, P. A. Bernstein, and E. Rahm, *Generic schema matching with cupid*, in VLDB 2001, Roma, Italy, 2001, pp. 49–58.

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

[19] S. Medasani, R. Krishnapuram, and Y. Choi, *Graph matching by relaxation of fuzzy assignments*, IEEE Trans. on Fuzzy Systems, 9 (2001), pp. 173–182.

[20] 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.

[21] 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 (2005), pp. 462–497.

[22] E. Rahm and P. A. Bernstein, *A survey of approaches to automatic schema matching*, VLDB Journal, 10 (2001), pp. 334–350.

[23] M. Smiljanic, M. van Keulen, and W. Jonker, *Formalizing the xml schema matching problem as a constraint optimization problem*, in DEXA, K. V. Andersen, J. K. Debenham, and R. Wagner, eds., vol. 3588 of Lecture Notes in Computer Science, Springer, 2005, pp. 333–342.
[24] E. Tsang, *Foundations of Constraint Satisfaction*, Acadimic Press, 1993.
[25] Z. Zhang, H. Che, P. Shi, Y. Sun, and J. Gu, *Formulation schema matching problem for combinatorial optimization problem*, IBIS, 1 (2006), pp. 33–60.

Edited by: Dominik Flejter, Tomasz Kaczmarek, Marek Kowalkiewicz
Received: February 3rd, 2008
Accepted: March 19th, 2008
Extended version received: June 17th, 2008