Designing a Knowledge-based Schema Matching System for Schema Mapping

Sarawat Anam[1], Yang Sok Kim[2], Byeong Ho Kang[1] and Qing Liu[3]

[1]{Sarawat.Anam,Byeong.Kang}@utas.edu.au
School of Engineering and ICT, University of Tasmania, Hobart, Australia

[2]yangsokk@gmail.com
Department of Management Information Systems, Keimyung University, Korea

[3]Q.Liu@csiro.au
Autonomous Systems, CSIRO Computational Informatics, Hobart, Australia

Abstract

Schema mapping that provides a unified view to the users is necessary to manage schema heterogeneity among different data sources. Schema matching is a required task for schema mapping that finds semantic correspondences between entity pairs of schemas. Semi-automatic schema matching systems were developed to overcome manual works for schema mapping. However, such approaches require a high manual effort for selecting the best combinations of matchers and also for evaluating the generated mappings. In order to avoid such manual works, we propose a Knowledge-based Schema Matching System (KSMS) that performs schema mapping both at the element and structure level matching. At the element level matching, the system combines different matching algorithms using a hybrid approach that consists of machine learning and knowledge engineering approaches. At the structure level matching, the system considers hierarchical structure that represents different contexts of a shared entity. The system can update knowledge if schema data changes over time. It also gives facilities to the users to verify and validate the schema matching results by incremental knowledge acquisition approach where rules are not predefined. Our experimental evaluation demonstrates that our system is able to improve the performance and to generate the accurate results.

Keywords: Schema matching, schema mapping, knowledge-based approach, element level and structure level matching.

1 Introduction

Schema matching is necessary to overcome semantic heterogeneity problem as the schemas are designed by different people. It finds mappings between semantically related entity pairs of schemas. These mappings are used to integrate data residing in different sources, and to make knowledge discovery easy and systematic. Schema matching can be done at the element level and structure level. Element level matching only considers matching names of the entity pairs, and it can be done by string similarity metrics and text processing techniques. Different string similarity metric and text processing technique perform well for different schema data. This is because the schema data contains different characteristics such as identical, abbreviated, synonym and combined words. In addition, the techniques generate schema matching problems: false positive (if reported match by expert is false and predicted match by algorithm is true), and false negative (if reported match by expert is true and predicted match by algorithm is false). Therefore, it is necessary to combine these techniques effectively, and to handle the matching problems.

Some solutions have been proposed in the literature to combine the techniques. YAM (Duchateau et al., 2009) uses machine learning approach for combining the techniques at the element level. The system shows if false positives are high, then precision becomes low. If false negatives are high, then recall becomes low. Precision can be 1.0 if false positive becomes zero. If precision becomes high but recall becomes very low, then overall performance becomes very low. For this, it is very important to increase the value of recall. The system runs much iteration until the similarity scores between entities become stable and it removes some incorrect mappings (pre-defined). However, it takes much time to iterate many times, and it needs to rebuild a training model if schema data changes overtime.

Incremental knowledge engineering approach, Censored Production Rules (CPR) based Ripple-Down Rules (RDR) has been used by (Anam et al., 2014) for schema mapping. The approach uses the features created by the combination of string similarity metrics and text processing techniques for creating rules. However, the limitation of the approach is that it is time-intensive to create rules for mapping entity pairs one by one at the element level. In order to overcome the limitations of the above approaches, it is necessary to use a hybrid approach that combines both machine learning and knowledge engineering approaches at the element level. Element level matching does not only give proper results for schema mapping as it only considers matching names of the entities. So it is important to do structure level matching to get the accurate results. Structure level matching uses the result of element level matching and considers the hierarchical structure that represents different contexts of a shared entity. In order to get the final mapping results, it is necessary to combine the results of the element level and structure level matching using some aggregation functions. For determining the best suitable aggregation function, it is necessary to compare the performance of the functions.

In this research, we introduce a Knowledge-based Schema Matching System (KSMS) that matches schemas both at the element level and structure level and produces the final result. We use the following processes in the system:

- We use hybrid-RDR (Anam et al., 2015) approach at the element level matching. The approach consists of decision tree, J48 and incremental knowledge engineering approach, Censor Production Rules (CPR) based Ripple-Down Rules (RDR). We combine the similarity values of different string similarity metrics and text processing techniques for constructing features. These features are fed into J48 to generate matching results. If J48 generates some wrong matchings, then CPR based RDR is used for correcting and validating the matching results.
- We use graph matching algorithm, Similarity Flooding (Melnik et al., 2002) that matches schemas considering structural information to discover additional mappings.
- We combine the results of the element level and structure level using aggregation functions to get the final results. We compare the performance of the aggregation functions and choose the best one.

2 Basic Definitions

In this section, we give some basic definitions of the foundations of schema matching and mapping.

A **schema** is defined as a formal structure that represents a set of entities. Each schema entity has a name, a data type, a description (called annotation) as well as instances. The kind of schemas can be database schemas, XML-schemas, entity-relationship diagram, and ontology description.

Schema mapping takes as input two schemas, each consisting of a set of discrete entities, and determines as output the relationships holding between these entities (Cate et al., 2013).

Schema matching is a process that discovers mappings between similar or same entity for a given entity using matching algorithms. We give an example of schema matching and mapping in the following:

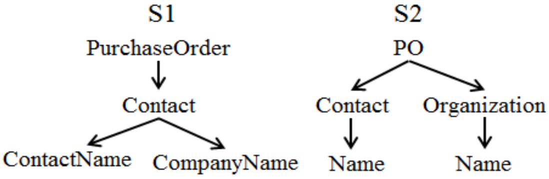

Fig. 1. Example of two schemas

For illustrating schema matching and mapping problem, we use two schemas, S1 and S2 representing the information of purchase order domain and the schemas are shown in **Fig.1**. These schemas contain different types of characteristics such as identical, abbreviated, synonym and combined words. Each schema consists of a set of schema entities. Similar types of schema entities are found in these datasets. For example, *PO* is an abbreviation of *PurchaseOrder* and *Company* is synonym of *Organization*. Schema matching is done at the element and structure level.

Element Level Matching considers only matching names of the entities. The basic techniques of this matching are string similarity metrics and text processing techniques. **String similarity metrics** compare the names of the schemas in order to produce a degree of similarity. **Text processing techniques** such as tokenization, abbreviation expansion and synonym lookup processes the names of the entities before matching. For example, *PO* is expanded to *PurchaseOrder* using abbreviation expansion.

Similarity measures produce numeric value ranging from 0 to 1 in normalized similarity metrics, schema mapping decision is Boolean – TRUE or FALSE. In order to take decision whether or not the source and the target entities are matched, a threshold value is specified. For example, Levenshtein string metric produces similarity value 0.4 between *ContactName* of S1 and *Name* of S2. If the threshold value is 0.4 for determining correct mapping that means the algorithm considers that all the pairs of entities with a confidence measure greater than or equal to 0.4 as correct mapping entities. Then the matching algorithm returns mapping decision to the user is TRUE. Another matching algorithm matches *CompanyName* in S1 and *Organization* in S2 using the combination of tokenization and synonym look up. First, *CompanyName* is tokenized as *{Company, Name}* and then *Company* and *Organization* are matched according to the meaning of the entities using synonym lookup and returns to the user that mapping decision is TRUE.

Only string similarity metrics and text processing techniques do not produce good performance for schema mapping. Therefore, it is necessary to use some combination functions such as machine learning algorithms, knowledge engineering approaches, neural network and hybrid approaches.

Structure level matching considers matching hierarchical structure of a full graph. In **Fig.1**, the hierarchical structure matching such as PurchaseOrder.Contact.contactName→ PO.Organization.Name is FALSE.
However, PurchaseOrder.Contact.companyName→ PO.Organization.Name is TRUE. This is because company and organization are matched according to the hierarchical context.

3 Related Works

There are some systems for schema mapping in the available literature. Lee and Doan developed a machine learning based approach, eTuner (Lee et al., 2007) to automatically tune schema matching systems to the problems. The approach handles relational schemas and considers only 1:1 mappings between schema pairs. It uses name matchers such as edit-distance and q-gram as terminological matchers. It can match source schema against synthetic schemas, for which the ground truth mapping is known, and can find a tuning in order to improve the matching performance of source schema against real schemas. It needs user assistance to improve the tuning quality by getting suggestion about the domain-specific perturbation rules. As the perturbation rules are known, so the mapping between original source and perturbed schema is also known. The approach is used for semantic matches and maintaining wrappers. However, the approach only considers source schema and ignores target schema, and tunes only small to moderate size schemas. Another problem is that the perturbation rules are static and so for different mapping problems, the generated gold standard does not differ much (Peukert et al., 2012).

Meta level learning (Eckert et al., 2009) is the first to recognize the need to have more schema features for creating adaptive processes. For this, the authors combine different matchers using machine learning techniques. They use the output of different matchers and additional features about the nature of the entities to be matched, as input for the learning approach. However, no suitable gold mappings are available for learning, and for this learned models often are not able to return results with a good quality. Besides, the learning approach easily overfits with the learning base, and the performance decreases significantly with increasing sizes of decision trees.

Duchateau et al. present an approach, MatchPlanner (Duchateau et al., 2008) for schema matching which uses a decision tree to combine the best suitable match algorithms. The approach inputs a set of schemas and a decision tree which is composed of match algorithms, and outputs a list of mappings which are validated by experts to find out whether the matching is correct or not. The feedback is used to feed into another decision tree for learning. YAM (Duchateau et al., 2009) is a machine learning based schema matching factory. In the learning phase, YAM considers users' requirement such as a preference for recall or precision, provided expert correspondences. It uses a Knowledge Base (KB) that consists of a set of classifiers, a set of similarity measures, and pairs of schemas which have already been matched. In the matching phase, the KB is used to match unknown schemas. In the system, users are asked to select appropriate classifiers. If the users do not have proper knowledge, then they depend on the default classifiers. However, the default classifiers often do not produce good performance. In addition, without proper knowledge, it is not easy to provide the preference between precision and recall. Machine learning techniques are promising for element similarity, but they need to rebuild the training model if schema data changes over time. Inversely, knowledge engineering approach encodes human knowledge directly, such that knowledgebase can be constructed with limited data.

Some systems have used knowledge engineering approach for schema matching. (Peukert et al., 2012) propose a self-configuring and adaptive schema matching system. It uses different terminological matchers such as name, datatypes, annotations, and synonyms using WordNet. In the structure level matching, it uses similarity propagation approach. The system depends on some features that are computed from input schemas and from intermediate mapping results. The features are then used in matching rules to select matchers, aggregation and selection operators. The rules represent expert knowledge on how to define or adapt schema matching processes. The matching process is iteratively extended, rewritten and executed in order to correct matching problems. However, the system predefines mapping rules such as starting, aggregation, rewrite, refine and selection. Therefore, the system faces problems when the viewpoint of two schemas is highly different. Second problem is that if some pre-defined mappings are incorrect and these methods are run only one time to produce new mappings, then the accuracy of new results will be unconfident. Third problem is that the system tunes matching processes manually and it does not split the process control flow based on the type of entities to be matched. Traditional rule-based systems require time-consuming knowledge acquisition as in those systems a highly trained specialist, the knowledge engineer, and the time-poor domain expert are necessary in order to analyze domain (Richards, 2009).

AMC (Peukert et al., 2011) is a schema and ontology matching framework where it is necessary for the users to provide an appropriate operator from different types of operators such as matcher, combination, selection, analyzer and blocking operators as input and to investigate individual results of individual operator. For this, users need to gather knowledge about the operators. If users want to use the default operator, then the operator may not handle different schemas of different domains.

In this research, we use Hybrid-RDR (Anam et al., 2015) approach that combines decision tree, J48 and incremental knowledge engineering approach, Censor Production Rules (CPR) based Ripple Down Rules (RDR) at the element level. In the approach, the KB is empty at the beginning, and the first rule is added to the KB by classifying a dataset using decision tree classification model. Then rules are added incrementally in order to solve schema matching problems such as false positives and false negatives. There are some advantages of the approach. First, only one classification model of decision tree is used in the approach, so it does not generate any over fitting problem. Second, rules are not pre-defined. Rules are created based on the features constructed from string similarity metrics and text processing techniques. Third, the approach does not need time consuming knowledge acquisition as rules are only created to correctly classify the wrongly classified cases produced by decision tree model. At the structure level matching, Similarity Flooding algorithm is used to match the hierarchical structure of a full graph.

4 KSMS Overview

The main components of KSMS system are described in **Fig.2**. The system discovers mappings between two schemas by element level and structure level matchers. The final mapping results are produced by using aggregation function. The functionalities of the system are described below:

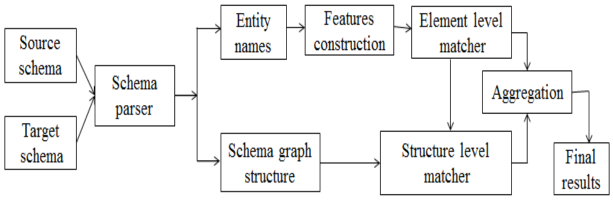

Fig.2. KSMS architecture

KSMS has been implemented in Java. It supports Graphical User Interface (GUI) for selecting schemas, displaying mapping knowledge created by feature construction process, classifying entities using J48 training model, creating rules for knowledge acquisition using features, checking satisfaction of rules, validating rules and also for saving rules to the Knowledge Base (KB).

In the system, any two schemas are first selected from repository. At the element level, input source and target schemas are parsed to extract names of the entities.

4.1 Feature Construction

Features of the entities are constructed using terminological matchers: text processing techniques and string similarity metrics. Feature construction processes are: **Step 1**, Cartesian product of the entities is generated. **Step 2**, three text processing techniques such as tokenization, abbreviation and acronym expansion, and synonym lookup are applied on the entities. **Step 3**, string similarity metrics are applied on the features of the attributes computed from the above two steps. We use string similarity metrics developed by two open source projects. For Levenshtein, JaroWinkler, Jaro Measure, TFIDF and Jaccard, we use open source library SecondString[1] and for Monge-Elkan, Smith-Waterman, Needleman-Wunsch, Q-gram and Cosine, we use SimMetric open source library[2]. Similarity values are normalized, such that the value within from 0 to 1, where 0 means strong dissimilarity and 1 means strong similarity. The threshold values for deciding schemas matching (true/false) are increased with 0.1 from 0 to 1. Another feature is created by using expert manual mapping (true/false). These features and features values are termed as attributes and cases respectively.

4.2 Element Level Matching

The extracted features including cases are fed into Hybrid-RDR approach. In the approach, knowledge base (KB) is empty at the beginning. First decision tree, J48 constructs a classification model using a small number of cases and uses the model for classifying the new cases. The decision tree rule is added in the KB as a first rule. Then users verify the results based on the expert manual mapping. If any case is wrongly classified (false positive/ false negative), new stopping rule is added to the KB to make the classification as NULL. The rule is created based on the features using a knowledge acquisition process of CPR based RDR (Kim et al., 2012). Knowledge acquisition is a process which transfers knowledge from human experts to knowledge based systems. The rule consists of one or more than one conditions. The condition has the form:

Attribute operator value

Where *attribute* is the feature, *operator* can be '=, !=, <, >, <=, >=', and *value* is the feature value. The conditions are added into a condition list to make rule. The rule is checked to determine whether it is satisfied by the current case or not. If the rule is satisfied, then the rule is validated on all the wrong classified cases to check whether other cases also satisfy the rule. The rule is saved in the KB as censor node which provides the classification of the wrongly classified cases as NULL. In order to correctly classify the NULL classified cases, alternative rules are added to the KB as child rules of the root rule for correctly classifying the cases as TRUE/FALSE.

The inference process is based on searching the KB represented as a decision list with each decision possibly refined again by another decision list. Once a rule is satisfied by any case, the process evaluates whether or not the exception rules are matched to the given case. If any exception rule is not satisfied, then the process stops with one path and one conclusion. However, if any exception rule is satisfied, the fired rule becomes zero according to censored conditions (Kim et al., 2012). Then other rules below the rule that was satisfied at the top level is evaluated. The process stops when none of the rules can be satisfied by the case in hand. The inference algorithm is the following:

```
1. Set lastFiredRule and CurrentRule as null
2. Get exceptionRule of rootRule
3. If exceptionRule is not null, set exceptionRule as currentRule
4. Evaluate inputCase with currentRule
   i. If inputCase satisfies currentRule, set currentRule as lastFiredRule and get exceptionRule of currentRule
      a. If exceptionRule is not null, set exceptionRule as currentRule and go to 4
   ii. Else get alternativeRule of currentRule
      a. If alternativeRule is not null, set alternativeRule as currentRule and go to 4
5. Stop inference process and return lastFiredRule
```

The mapping results produced by this approach at the element level are stored in a repository.

4.3 Structure Level Matching

At the structure level matching, input schemas are parsed and converted into graph data structure. Structure matching is used to adjust incorrect matches from matching phase, and it finds additional mappings. KSMS uses the results of element level to match schema graph structures based on a graph matching algorithm called Similarity Flooding (Melnik et al., 2002). The approach converts schemas into directed labelled graphs and uses

[1] http://secondstring.sourceforge.net

[2] http://sourceforge.net/projects/simmetrics

fix point computation to determine the matches between corresponding nodes of the graphs. It uses the concept that two nodes are matched based on the matching of neighborhood.

4.4 Final Results of Mapping

In this phase, we combine the mappings discovered from element level and structure level matching by weighted, average, minimum, maximum and harmonic mean aggregation methods. Different systems have used different aggregations function for combining mappings. In order to determine the best one, we compare the performance of all the aggregation functions. We define the similarity values found from element level matching and structure level matching by *esim* and *ssim* respectively. The aggregation functions are described below:

- **Weighted**: This strategy returns a weighted sum of the similarity values. The similarity value found from structure level matching is used as the threshold value which is the weight of element level matching, and the weight for structure level matching, W_{struct} is (1-threshold) (Ngo et al., 2011b). The weighted similarity of the entity pair, $e1$ and $e2$ is calculated as:

$$wsim(e1, e2) = W_{struct}\,.ssim(e1, e2) \\ + (1 - W_{struct}).esim(e1, e2)$$

This combination strategy is used in some matching systems (Do and Rahm, 2002, Ngo and Bellahsene, 2012, Madhavan et al., 2001).

- **Average:** The average similarity is calculated by dividing the sum of the similarity values of two string metrics for each name pair by the total number of similarity functions. Average value is calculated by the following function:

  Avg= (*esim*+*ssim*)/2

The matching systems which use this strategy are (Do and Rahm, 2002, Volz et al., 2009, Jimenez et al., 2009).

- **Minimum:** This strategy returns the minimum similarity value between two string metrics. Minimum value is calculated by using the following function:

  Min=Math.min (*esim*, *ssim*)

- **Maximum**: This strategy returns the maximum similarity value between two string metrics. Maximum value is calculated by using the following function:

  Max=Math.max (*esim*, *ssim*)

The combination strategies, minimum and maximum are used in some matching systems (Do and Rahm, 2002, Volz et al., 2009, Massmann and Rahm, 2008).

- **Harmonic mean:** Harmonic mean is calculated by the following function:

  Harmonic mean=2**esim***ssim*/ (*esim*+*ssim*)

This combination strategy is used in the systems (Do and Rahm, 2002, Ngo et al., 2011a).

5 Experimental Design

5.1 Datasets

Five XDR schemas of purchase order domain, such as CIDX, EXCEL, NORIS, PARAGON and APERTUM obtained from www.biztalk.org are used for this evaluation study. We denote the schema datasets CIDX, EXCEL, NORIS, PARAGON and APERTUM by C, E, N, P, and A respectively. These schema datasets are used for schema mapping evaluation and terminological matching evaluation (Peukert et al., 2011). These schema datasets contain different types of characteristics such as identical words, combined words, abbreviated words and synonym words. Each schema dataset contains 35 (E), 30 (C), 46 (N), 82 (A), 59 (P) entities.

5.2 Experimental Procedure

In this research, we experiment ten matching tasks one-by-one using all combinations of five schema datasets such as C-E (first matching task is to deal with two datasets, CIDX and EXCEL), C-N, C-P, C-A, E-N, E-P, E-A, N-P, N-A and P-A. We take the Cartesian product of the schema datasets for ten matching tasks separately. The sizes of Cartesian product of the matching tasks are 1050 (C-E), 1380(C-N), 1770(C-P), 2460(C-A), 1610(E-N), 2065(E-P), 2870(E-A), 2714(N-P), 3772(N-A) and 4838(P-A) entity pairs respectively. We denote the matching tasks C-E, C-N, C-P, C-A, E-N, E-P, E-A, N-P, N-A and P-A by D1, D2, D3, D4, D5, D6, D7, D8, D9 and D10 respectively.

In the evaluation approach, we feed the datasets in to the static decision tree, dynamic decision tree (DT) approaches and Hybrid-RDR approach. The approaches learn a new model by including newly available data. We use the decision tree to compare the performance to the existing approaches as some systems, YAM (Duchateau et al., 2009) and MatchPlanner (Duchateau et al., 2008) use decision tree as a combination method. Here we divide the decision tree into static and dynamic decision tree. In the static decision tree, one dataset is used for building a training model and another dataset is used for testing. In the dynamic decision tree, one dataset is used for building a training model and test the test dataset. Then two datasets are combined and used for building a training model and test the test dataset. Incrementally, all the datasets except the test dataset are used for building a training model and test the test dataset.

We perform ten experiments to get the performances (precision, recall and F-measure) of the static decision tree, dynamic decision tree (DT) and Hybrid-RDR approaches. In all experiments, we randomly select datasets for training and testing. For example, we select D1 for training and D10 for testing, D7 for training and D3 for testing, D4 for training and D9 for testing. In such a way, we select the datasets for training and testing. The evaluation processes of the approaches are described below:

5.2.1 Static DT

In the static decision tree approach, we create decision tree model, $\boldsymbol{ML_0}$ for D1 and test D10. Then we create

$\boldsymbol{ML_1}$ for D2 and test D10. In this way, we create $\boldsymbol{ML_2}$ for D3 to $\boldsymbol{ML_8}$ for D9 and test D10. For other combination, we create $\boldsymbol{ML_0}$ for D7 and test D3, $\boldsymbol{ML_1}$ for D8 and test D3. In this way, we create $\boldsymbol{ML_8}$ for D1 and test D3.

5.2.2 Dynamic DT

In the dynamic decision tree approach, we create decision tree model, ML_0 for D1 and test D10. Then we incrementally add other datasets like D1+D2, D1+D2+D3 for creating decision tree models, ML_1, ML_2 respectively and test D10. In this way, we add all nine datasets for creating decision tree model, ML_8 and test D10.

For all decision tree approaches, we consider 10-fold cross validation. 10-fold cross validation means that the data is split into 10 groups where nine groups are considered for training and the remaining one group is considered for testing. This process is repeated for all 10 groups. For all experiments using decision tree, we use WEKA (Hall et al., 2009) data mining and machine learning toolbox.

5.2.3 Hybrid-RDR

In the Hybrid-RDR approach, we create decision tree model, ML_0 for D1 and test D10. We also test D2 and find some wrong classified cases. Then we refine the decision tree rule by adding censor/exception/stopping rule, $Rule_0$ and again classify the cases by adding alternative rule, $Rule_0$. The censor rules are added as censor nodes of decision tree in the KB and alternative rules are added as parent rules in the KB. The ML_0+$Rule_0$ is then used for testing D10 and also for testing D3. We add rule, $Rule_1$ again for the wrong classified cases of D3, and ML_0+$Rule_0$+$Rule_1$ is used for testing D10. In such a way, we incrementally add rules for all nine datasets, ML_0+$Rule_0$+$Rule_1$+…+$Rule_8$ and test D10.

5.3 Evaluation Metrics

As this task is a classification task, we use the following conventional metrics: precision = $\frac{TP}{TP+FP}$, recall = $\frac{TP}{TP+FN}$ and F-measure = $\frac{2*precision*recall}{precision+recall}$, where TP is True Positive (hit), FP is False Positive (false alarm, Type I error) and FN is False Negative (miss, Type II error). For a specific threshold value, we calculate TP, FP and FN by comparing manually defined matches (R) with the predicted matches (P) returned by the matching algorithms according to (Jimenez et al., 2009).

6 Evaluation Results

Performance of the static decision tree, dynamic decision tree and Hybrid-RDR approaches depends on the features of the datasets which are created using string similarity metrics and text processing techniques. The performance of Hybrid-RDR approach also depends on the efficient knowledge acquisition. We compute performance in terms of precision, recall and F-measure. Precision estimates the reliability of the match predictions and recall specifies the share of real matches. During schema mapping, manually matching schemas of two heterogeneous data sources and false identified matches by algorithms are handled by humans. The burden of deleting false identified matches is much easier than creating manual matches among thousands of schemas (Stoilos et al., 2005). As for calculating recall value, manually identified matches are necessary, so recall value is very important. Only precision or recall cannot estimate the performance of match algorithms (Cheng et al., 2005). So it is necessary to calculate the overall performance or F-measure of Hybrid-RDR approach and both static and dynamic decision tree using both precision and recall. For this, we determine the best performing classification system based on the optimized F-measure (Marie and Gal, 2008) for almost all experimental datasets.

6.1 Schema Mapping Results at the Element Level

At the element level, the names of the entities are matched by static decision tree, dynamic decision tree and Hybrid-RDR approaches. For all the above three approaches, we perform ten experiments and compute average performance of the experiments. In all experiment, we randomly select datasets for training and testing. We compare the performance of the approaches to other approaches, AMC (Peukert et al., 2011), COMA (Do and Rahm, 2002), FALCON (Hu et al., 2008), RONDO (Melnik et al., 2003) based on F-measure. The performance, F-measures of these approaches are found from AMC. All F-measure of the approaches are described in **Table 1**. The *Datasets* column describes the datasets used for the experiments. The other columns, *AMC*, *COMA*, *FALCON* and *RONDO* represent F-measure of these approaches. We denote static decision tree, dynamic decision tree and Hybrid-RDR by *S_DT*, *D_DT* and *HRDR* respectively. The schema mapping result found from element level matching is described in **Table 1**.

Datasets	**AMC**	**COMA**	**FALCON**	**RONDO**	**S_DT**	**D_DT**	**H-RDR**
D1	0.44	0.42	0.38	0.41	0.81	0.85	0.90
D2	0.71	0.63	0.62	0.43	0.74	0.87	0.89
D3	0.59	0.51	0.55	0.53	0.65	0.78	0.85
D4	0.52	0.46	0.35	0.47	0.62	0.76	0.87
D5	0.45	0.42	0.42	0.41	0.74	0.82	0.86
D6	0.65	0.60	0.70	0.60	0.67	0.84	0.90
D7	0.51	0.48	0.44	0.45	0.66	0.79	0.88
D8	0.55	0.50	0.54	0.55	0.64	0.76	0.85
D9	0.41	0.34	0.39	0.28	0.68	0.75	0.83
D10	0.30	0.31	0.25	0.25	0.56	0.60	0.80
AVG	**0.51**	**0.48**	**0.47**	**0.44**	**0.68**	**0.79**	**0.86**

Table 1. F-measures compariosn of the approaches

In **Table 1**, we compare performance, F-measure of some previous approaches to the static decision tree, dynamic decision tree and Hybrid-RDR. We find that our approaches show better performance compared to AMC, COMA, FALCON and RONDO. The average performances of these approaches are 0.51, 0.48, 0.47 and 0.44 respectively, whereas for static decision tree, average performance is 0.68. Though static decision tree improves performance compared to the previous approaches, but the performance is still low. F-measure is calculated from precision and recall. The reason of low precision means high false positive values, and low recall means that the false negative numbers are very high. In order to increase the performance, we use dynamic decision tree which adds datasets gradually to the previous datasets for building training model and use the model for handling some false positives and false negatives. The approach improves the average performance up to 11% compared to static decision tree, but it is necessary to handle more false positives and false negatives to increase the performance. For this, we use Hybrid-RDR that handles the problems by efficient knowledge acquisition. The performance of Hybrid-RDR is reasonably high compared to other approaches for all datasets. The average performance of Hybrid-RDR is 0.86 which improves 18% and 7% compared to static and dynamic decision tree respectively.

The performance of the algorithms depends on the characteristics of the datasets such as identical, abbreviated, and synonym and combined words. If training dataset contains large number of abbreviated words, but test dataset contains large number of synonym words, then performance becomes low. For increasing the performance of dynamic decision tree, it is necessary to build models again with more datasets to correctly classify the schema data. Sometimes building model with a large amount of datasets may not improve the performance by classifying the schemas correctly because the learning approach easily overfits with the learning base. However, for the Hybrid-RDR approach, performance is improved by incrementally adding rules for solving false positives and false negatives.

6.2 Schema Mapping Results at the Structure Level

Only element level matching does not produce good results. In order to improve the performance and produce accurate results, we have performed structure level matching. The mapping result of structure level matching is shown in **Table 2**.

Datasets	Precision	Recall	F-measure
D1	0.98	0.94	0.96
D2	0.94	0.91	0.92
D3	0.93	0.95	0.94
D4	0.97	0.94	0.95
D5	1.00	0.89	0.94
D6	0.96	0.91	0.93
D7	0.95	0.93	0.94
D8	0.91	0.94	0.92
D9	0.95	0.91	0.93
D10	0.90	0.92	0.91
AVG	**0.95**	**0.92**	**0.93**

Table 2. Performance of KSMS at the structure level matching

In **Table 2,** we show that the performance of structure level matching in terms of precision, recall and F-measure. Precision is higher than recall in most of the datasets. This is reasonable when we consider structure level instead of element level. We compare this F-measure to the F-measure of the element level matching, and we find that average F-measure has been improved up to 7% when we consider the hierarchical structure at the structure level matching. The average precision, recall and F-measure of all the datasets in the purchase order domain are 0.95, 0.92 and 0.93 respectively.

6.3 Final Mapping Results by Aggregation functions

In order to combine the schema mapping results produced by element level and structure level matchers, and to produce the final results, we use aggregation functions on the F-measure. The final schema mapping results are shown in **Table 3** where the columns *Datasets, Harm, Avg, Min, Max, Weighted* describe information about datasets, HARMONIC MEAN, AVERAGE, MINIMUM, MAXIMUM and WEIGHTED aggregation results.

Datasets	Harm	Avg	Min	Max	Weighted
D1	0.93	0.93	0.90	0.96	0.90
D2	0.90	0.91	0.89	0.92	0.89
D3	0.89	0.90	0.85	0.94	0.86
D4	0.91	0.91	0.87	0.95	0.87
D5	0.90	0.90	0.86	0.94	0.86
D6	0.91	0.92	0.90	0.93	0.90
D7	0.91	0.91	0.88	0.94	0.88
D8	0.88	0.89	0.85	0.92	0.86
D9	0.88	0.88	0.83	0.93	0.84
D10	0.85	0.86	0.80	0.91	0.81
AVG	**0.90**	**0.90**	**0.86**	**0.93**	**0.87**

Table 3. Final mapping results

In **Table 3,** we find that MAXIMUM gives the highest and MINIMUM gives the lowest schema mapping results compared to other aggegation functions. As MAXIMUM takes the highest value and MINIMUM takes the lowest value between two values, we do not consider the results. We compare the results among other three functions. We find that WEIGHTED provides the lowest aggregation result. AVERAGE gives slightly better mapping results compared to HARMONIC MEAN for some datasets such as D2, D3, D6, D8 and D10. Therefore, the final average mapping performance, F-measure is 0.90.

7 Discussion

Schema mapping can be done by machine learning or knowledge engineering approaches at the element level. Machine learning approach is promising for element similarity, but it needs to rebuild a training model if schema data changes over time. Inversely, knowledge engineering approach encodes human knowledge directly such that knowledge base can be constructed with limited data, but it needs time consuming knowledge acquisition. In order to overcome the limitations, we have used Hybrid-RDR approach that combines machine learning algorithm, J48 and knowledge engineering approach, CPR based RDR in our system, KSMS. The advantage of Hybrid-RDR is that it needs only one training model to classify new schema data. If the model gives wrong classification, then rules are added incrementally in order to handle the problem. The approach increases performance incrementally with the help of knowledge acquisition and decreases rule addition over time.

However, only element level matching is not sufficient for schema mapping. This is because it is necessary to consider the hierarchical structure of a full graph in order to improve the performance and produce accurate results. For this, we have added the features of performing structure level matching in KSMS. Finally, we have used some aggregation functions for combining the results of both element level and structure level matching.

8 Conclusion and Future Works

In this research, we have presented a Knowledge-based Schema Matching System (KSMS) which has performed schema mapping both at the element and structure level. In order to show the ability of the system, we have used 5 XDR datasets from purchase order domain. Experimental results have shown that the system determines good performance both at the element and structure level. The final schema mapping result is determined by the average aggregation function. There are some advantages of our system compared to the existing systems. First, it is not necessary to select the best combination of matchers. Second, Knowledge base is empty at the beginning. That means the system does not need any initial expert correspondences from the users. Third, rules are not predefined. Rules are created based on the features constructed from element level matchers. Fourth, over fitting problem does not occur in the system as only one decision tree model is used for classifying schemas. Fifth, the system does not need time consuming knowledge acquisition as rules are only created to correctly classify the wrongly classified cases produced by decision tree model. Finally, the system can handle the schema matching problems: false positives and false negatives using knowledge acquisition. So users do not need to add, delete or modify schema mapping results manually.

In future, we will adapt our system for ontology mapping. Then we will experiment more datasets from other domains such as conference, bibliography and anatomy.

Acknowledgement

Autonomous Systems, Digital Productivity and Service Flagship, and the Tasmanian node of the Australian Centre for Broadband Innovation are assisted by a grant from the Tasmanian Government which is administered by the Tasmanian Department of Economic Development, Tourism and the Arts.

References

Anam, S., Kim, Y. and Liu, Q. (2014): Incremental Schema Mapping. *Knowledge Management and Acquisition for Smart Systems and Services.* Springer International Publishing.

Anam, S., Kim, Y.S., Kang, B.H. and Liu, Q. (2015): Schema Mapping Using Hybrid Ripple-Down Rules. *the Thirty-Eighth Australasian Computer Science Conference, ACSC 2015.* Sydney, Australia: CRPITT.

Cate, B.T., Dalmau, V. and Kolaitis, P.G. (2013): Learning schema mappings. *ACM Transactions on Database Systems (TODS),* 38**,** 28.

Cheng, W., Lin, H. and Sun, Y. (2005): An efficient schema matching algorithm. *Knowledge-Based Intelligent Information and Engineering Systems*, Springer, 972-978.

Do, H.-H. and Rahm, E. (2002): COMA: a system for flexible combination of schema matching approaches. *Proceedings of the 28th international conference on Very Large Data Bases,* VLDB Endowment, 610-621.

Duchateau, F., Bellahsene, Z. and Coletta, R. (2008): A flexible approach for planning schema matching algorithms. *On the Move to Meaningful Internet Systems: OTM 2008.* Springer.

Duchateau, F., Coletta, R., Bellahsene, Z. and Miller, R.J.(2009): Yam: a schema matcher factory. *Proceedings of the 18th ACM conference on Information and knowledge management*, ACM, 2079-2080.

Eckert, K., Meilicke, C. and Stuckenschmidt, H. (2009): Improving ontology matching using meta-level learning. *The Semantic Web: Research and Applications.* Springer.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H. (2009): The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter,* 11**,** 10-18.

Hu, W., Qu, Y. and Cheng, G. (2008): Matching large ontologies: A divide-and-conquer approach. *Data & Knowledge Engineering,* 67**,** 140-160.

Jimenez, S., Becerra, C., Gelbukh, A. and Gonzalez, F. (2009): Generalized mongue-elkan method for approximate text string comparison. *Computational Linguistics and Intelligent Text Processing.* Springer.

Kim, Y.S., Compton, P. and Kang, B.H. (2012): Ripple-down rules with censored production rules. *Knowledge Management and Acquisition for Intelligent Systems.* Springer.

Lee, Y., Sayyadian, M., Doan, A. and Rosenthal, A.S. (2007): eTuner: tuning schema matching software using synthetic scenarios. *The VLDB Journal—The International Journal on Very Large Data Bases,* 16**,** 97-122.

Madhavan, J., Bernstein, P.A. and Rahm, E. (2001): Generic Schema Matching with Cupid. *Proceedings of the 27th International Conference on Very Large Data Bases.* Morgan Kaufmann Publishers Inc.

Marie, A. and Gal, A. (2008): Boosting schema matchers. *On the Move to Meaningful Internet Systems: OTM 2008.* Springer.

Massmann, S. and Rahm, E.(2008): Evaluating Instance-based Matching of Web Directories. WebDB, 2008. Citeseer.

Melnik, S., Garcia-Molina, H. and Rahm, E. (2002): Similarity flooding: A versatile graph matching algorithm and its application to schema matching. *18th International Conference on Data Engineering*, IEEE, 117-128.

Melnik, S., Rahm, E. and Bernstein, P.A. Rondo (2003): A programming platform for generic model management. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ACM, 193-204.

Ngo, D., Bellahsene, Z. and Coletta, R. (2011a): A generic approach for combining linguistic and context profile metrics in ontology matching. *On the Move to Meaningful Internet Systems: OTM 2011*. Springer.

Ngo, D.H. and Bellahsene, Z. (2009): YAM++:(not) Yet Another Matcher for Ontology Matching Task. BDA'2012: 28e journées Bases de Données Avancées, 2012.

Ngo, D.H., Bellahsene, Z. and Coletta, R. (2011b): YAM++--Results for OAEI 2011. ISWC'11: *The 6th International Workshop on Ontology Matching*, 228-235.

Peukert, E., Eberius, J. and Rahm, E. (2011): AMC-A framework for modelling and comparing matching systems as matching processes. *27th International Conference on Data Engineering (ICDE)*, IEEE, 1304-1307.

Peukert, E., Eberius, J. and Rahm, E. (2012): A self-configuring schema matching system. *28th International Conference on Data Engineering (ICDE)*, IEEE, 306-317.

Richards, D. (2009): Two decades of ripple down rules research. *The Knowledge Engineering Review*, 24**,** 159-184.

Stoilos, G., Stamou, G. and Kollias, S. (2005): A string metric for ontology alignment. *The Semantic Web–ISWC*, Springer.

Volz, J., Bizer, C., Gaedke, M. and Kobilarov, G. (2009): *Discovering and maintaining links on the web of data*, Springer.