"University of Turku Technical Reports, No.13 - May 2017"

TRC-Matcher and enhanced TRC-Matcher

New Tools for Automatic XML Schema Matching

Lauri Mukkala | Jukka Arvo | Teijo Lehtonen | Timo Knuutila

Contact information:
University of Turku, Department of Future Technologies, 20014 Turun yliopisto, Finland,
firstname.lastname@utu.fi
ar.utu.fi

ISSN 2341-8028 | ISBN:978-951-29-6856-5

"University of Turku Technical Reports, No.13 - May 2017"

Abstract

Modern society depends on the access to a wide range of information that is located in heterogeneous data sources. Schema matching is a task of finding relationships among data source elements automatically. However, most of the existing schema matching software are semi-automatic meaning that they need a lot of interaction from an expert familiar with the systems being integrated. In this work, we propose a new hybrid matcher algorithm, called TRC-matcher, that is targeted for matching business oriented XML schemas with none or minor user assistance. When compared to previously published schema matching methods, the efficiency of the new algorithm is based on a new content profiling algorithm and on intelligent combination of matching results of multiple matching algorithms. In addition, an enhanced version of the TRC-Matcher is introduced that combines machine learning methods together with few new matching algorithms.

Keywords:
Schema Matching, Ontology Matching, WordNet, Neural networks, Machine learning, Automatic synonym dictionary generation

Turun yliopisto
University of Turku

Contents

1 Introduction

In the past decade, the demand for integrating different data sources has increased, but Integrating the data sources integrations can be problematic. The data sources are typically very heterogeneous, e.g., in shipbuilding industry, a shipyard usually has multiple subcontractors that store the same order data differently. For example, an element containing quantity can be named ***Quantity*** in system A and ***Qty*** in system B. In order to make the data usable for the shipyard, the differently named data has to be connected together.

One of the most important steps in integrating data sources is to combine schemas of data sources. A schema represents in a formal way the construction of data structures, defines data constraints, and states valid data structures and values [4]. XML schema and database schema are examples of schemas where the schema elements describe one column in a database. Additionally, some matching algorithms use ontologies instead of schemas, one standard ontology format being Web Ontology Language (OWL). The main difference between the schemas and ontologies is that ontologies are a formal way to define the structure of knowledge for various domains while the schemas describe the structure of data.

The process where semantic relationships between data source elements are identified is called schema matching. Schema matching is used in many types of applications, including data warehouses, web applications, reporting systems, health-care applications and semantic query processing. Currently, schema matching requires manual user assistance via graphical user interfaces, which may be sufficient when only a couple of data sources are integrated. The matching is typically done by a specialist with enough knowledge about the domain area and the schemas involved. However, extensive usage of user assistance can cause issues due to human errors, and the task becomes time consuming when the number of data sources increases.

A diverse range of approaches for automating the schema matching has been proposed to alleviate the issues of manual work requirements. Most of the currently available matching tools try to find semantic, structural or linguistic correspondences between schema elements. However, automated schema matching has still several problems: (1) schema labels can be ambiguous, and may use different languages; (2) schema labels can be non-standard words and contain abbreviations and acronyms; (3) in addition to determining which pairs are matches, algorithms have to determine which are not; (4) a single schema element can be one element in system A but a composite of multiple elements in system B (i.e. FirstName and LastName vs. FullName) and vice versa. Because schema matching has many challenges, the full

automation of the process is a difficult task.

In this paper, a new business oriented matching algorithm called TRC-Matcher based on XML schemas is introduced. The main target of the new algorithm is not to fully automate the schema matching process, but instead, to significantly reduce the amount of manual work that integration experts spend on database integrations. All automation in these processes improve the cost efficiency of integrations. TRC-matcher uses two main approaches for computing the matches. The first approach uses multiple linguistic methods such as synonym and abbreviation dictionaries in addition to lower level string matching algorithms. The second approach is used as a complementary approach when the first approach is uncertain about the quality of the matching results, and if the matched schema has data available. The second approach utilizes content profiling to improve the accuracy of the matching results by considering elements with similar content profiles as possible matches. Content profiles are constructed by using value length and composition (e.g. the value *abc123* is transformed into profile *llldddd*). What makes the proposed algorithm unique is the utilization of the content profiling, and the way of combining different methods into a hybrid matcher. To our knowledge, no previous algorithm combines the use of dictionary-based methods, string-based methods, and content profiling together. Enhanced TRC-Matcher algorithm is also presented that extends the TRC-mather algoritm with machine learning techniques and an automated synonym dictionary method. The enhanced TRC-Matcher algorithm is tailored to work with commercial test data which does not have much linguistic correspondences between the source and target matching pairs.

The remainder of this paper is organized as follows. In Section 2 we discuss recent related work. In Section 3 we present our method for schema matching using the synonym dictionary and content profiling. Section 4 shows results from test data set. Section 5 present the enhanced TRC-Matcher algorithm while Section 6 introduces few additional improvements to the enhanced TRC-Matcher method. Finally, in Section 7 we discuss future work while Section 8 presents our conclusions.

2 Related Work

The field of ontology and schema matching research is active and matching accuracy is improved every year. This can be seen from the results of Ontology Alignment Evaluation Initiative's (OAEI)[1] competition as the number of participants has increased and the results have constantly improved. There are also extensive surveys written on ontology matching algorithms that describe and analyze a large number of algorithms [17, 23–25].

Po & Sorrentino [12] proposed an automatic method aimed at discovering probabilistic lexical relationships from ontologies. They point out that non-dictionary words are difficult for dictionary-based methods. They try to solve that problem with schema label normalization so that, for example, abbreviations are normalized to full words.

Falcon-AO is an automatic tool for ontology matching, which uses two matcher methods: the first is a matcher based on linguistic matching and the second is a matcher based on graph matching [6]. Alhassan, Junaidu & Obiniyi developed and implemented an enhanced version of Falcon-AO with the use of a large lexical database, WordNet [1]. The proposed matcher uses several matching algorithms and Falcon-AO's built-in controller. The controller executes matching algorithms and combines similarities based on measures of linguistic comparability and structural comparability.

Most of the string matching algorithms cannot match synonyms and abbreviations. Lu et al. studied problems related to approximate string matching with synonyms [8]. Similarly, we also try to find answers to one of the biggest problems with synonyms: how to determine which two words are the most similar ones.

XMap [18] is a multi-layer matching system which uses three different layers to compute matches. A terminological layer is used to compute similarities between the entity names within the ontologies. The similarities are computed by combining linguistic similarities with the semantic elements. A structural layer computes the similarity between the concepts i.e. by taking into account the element's position in the ontology and the element's constraints. An alignment layer aims to provide the final similarity matrix between the concepts. XMap uses synonym sets (synsets) from WordNet by selecting the synset with the greatest potential for matching elements with similar meanings.

Other top algorithms from the OAEI's competition include Mamba [19], Agree-

[1] http://oaei.ontologymatching.org/2015/

ment Maker Light (AML) [21] and GMap [20]. Mamba generates hypotheses about equivalences between labels and tokens but also about mappings between concepts and properties are considered to be true and wrong. Mamba first normalizes and splits the labels into tokens. Then several scoring algorithms (e.g. similarity, Levenshtein distance and appearence in the same synset) are used to evaluate and match token pairs.

GMap combines the sum-product network and the noisy-or model. GMap is an iterative algorithm which works in four steps. The first step is to calculate the lexical similarity based on edit-distance and external lexicons. The second step uses sum product networks to encode the similarities based on individuals and disjointness axioms and calculates the contribution through MAP inference. The third step is to use noisy-or model to encode the probabilistic matching rules. The last step is to use one-to-one constraint and crisscross strategy in order to obtain initial matches which are then iteratively improved.

AML derives from the AgreementMaker [22] but is more focused on the efficient matching of very large ontologies. AML consists of three layers. Matchers of the first layer compare concept features with the help of e.g. WordNet. The second layer uses structural properties of the ontologies to procude matches and the third layer combines the results of multiple matchers.

3 TRC-Matcher

TRC-Matcher is a hybrid method that computes matches using two complementary approaches. The main approach utilizes linguistic methods such as synonym and abbreviation dictionaries in addition to low level approximate string matching algorithms (i.e. Levenshtein distance [7], Jaro-Winkler distance [14], Monge Elkan algorithm [10], Longest Common Substring [5]). When the matched dataset contains data values in addition to field names, TRC-Matcher utilizes a content profiling algorithm to find matches for non-dictionary words. Elements with similar content profiles are marked as possible matches to support the overall match computation process. To our knowledge other matching algorithms do not use content-profiling due to the fact that OWL-files only contain ontologies but not values. If both algorithms are used, the final result is pruned from the union of all matches in the final combination phase.

3.1 Linguistic Matching

Linguistic matching is often used in schema matching, because usually at least some consistency can be found between the names of the two matched elements. Most commonly used linguistic methods are variations of approximate string matching algorithms (i.e. Jaro-Winkler distance [14], Levenshtein distance [7], Soundex [13]) and dictionary-based approaches. However, it is not easy to extend traditional similarity functions to handle the synonyms and abbreviations that often appear in XML schema matching. Therefore, the TRC-matcher utilizes a synonym dictionary to overcome this limitation [9].

In Algorithm 1 the input is assumed to be a subset of elements that a database integration expert has selected from the input elements, while the set of target elements is not reduced. Each input element is matched one at a time and the input element is first tokenized into meaningful tokens (e.g. tokens that are actual words) by splitting the elements using capital letters and special characters (i.e. OrderNumber = [Order, Number] and `Reference_Number` = [Reference, Number]). Hereafter, each token that is an abbreviation is replaced with the corresponding full word. The abbreviation correspondences are determined from STANDS4 Web Service [15].

The final step of the input element processing is to form a synonym dictionary where completed and meaningful input tokens are keys for lists of corresponding synonyms words. TRC-Matcher's approach uses WordNet dictionaries for synonyms [9].

Algorithm 1 Linguistic Matching Algorithm

```
1:  function ComputeSynonyms(Elements)
2:      Synonyms ← ∅
3:      for each Element in Elements do
4:          SPieces ← SplitInToWords(Element)
5:          SPieces ← FindAndCompleteAbbreviations(SPieces)
6:          Synonyms[Element] ← GetSynonyms(SPieces)
7:      end for
8:      return Synonyms
9:  end function
10:
11: function DoLinguisticMatching(MatchedElements)
12:     SourceSynonyms ← ComputeSynonyms(MatchedElements.source)
13:     TargetSynonyms ← ComputeSynonyms(MatchedElements.target)
14:     MatchedPairs ← ∅
15:     for each SourceElement in MatchedElements.source do
16:         SSynonyms ← SourceSynonyms[SourceElement]
17:         Matches ← ∅
18:         for each TargetElement in MatchedElements.target do
19:             TSynonyms ← TargetSynonyms[TargetElement]
20:             Matches ← Matches ∪ CommonSynonyms(SSynonyms, TSynonyms)
21:         end for
22:         MatchedPairs[SourceElement] ← PruneWithFuzzyLogic(Matches)
23:     end for
24:     return MatchedPairs
25: end function
```

WordNet provides an effective combination of traditional lexicographic information and modern computing. It is an online lexical database designed to be used by computer software. Nouns, verbs, adjectives, and adverbs are organized into synonym sets that are linked with semantic relations.

The same process to construct the synonym dictionary is also computed for the target elements (lines 1-9, Algorithm 1). The synonym sets of target elements are scanned for each input element and matches are generated for the element pairs that have at least 70% synonym correspondence. The threshold value was selected based on empirical testing of multiple data sets and threshold values. Additionally, the target element that has the highest amount of common synonyms generates one match. After all target elements have been processed, the match candidate list is pruned with Jaro-Winkler distance [14] and with the longest common substring algorithms whose implementations can be found e.g. from the FuzzyString open source code project [3]. The threshold parameter for the pruning is set to strong. When usage of synonym dictionaries do not result candidate matches for the target elements, the fallback method is to compute the same set of pruning algorithms for all possible input and output element pairs. In this case, the resulting set of matches is based on dictionary keys instead of synonyms.

3.2 Content Profiling

XML schemas are often used as an intermediate format between integrated systems in real life integration cases. Therefore, XML schema was chosen as the input format for TRC-matcher. The XML format additionally supports data values, and therefore data samples are utilized in our secondary matching approach.

The main idea behind the content profiling algorithm is based on observation that matched schema elements usually contain similar data. However, the data is rarely the same in both systems, which denotes that the actual data values are not efficient in determining whether two elements match together. Therefore, our solution uses the lengths and content profiles of the data samples to determine potential matches. For example, elements such as phone number and zip code have typically fixed lengths and content profiles. The content profile is constructed by processing one alphanumeric value at time. If alphanumeric is an alphabetic character, letter 'l' is added to it's content profile. In case of numeric characters, 'd' is added correspondingly. Special character are skipped. For example:

$$GenerateDataProfile(value123) = lllllddd$$

Content profiling is not useful with elements whose data is free formed, e.g., in a description element of XML schema, as in those cases the content profiles are very different due to arbitrary text. TRC-matcher relies solely on linguistic matching when content profile matches are not found. However, content profile matching is particularly efficient in cases that have a lot of variation in element names, but the data sizes are fixed, such as order number (i.e. POrder, PNO and OrderNumber).

Algorithm 2 is the content profile matching algorithm. It computes profiles for each source and target element at the beginning. In the next phase, the content

Algorithm 2 Content Profile Matching Algorithm

```
1:  function ComputeContentProfiles(Elements)
2:      Profiles ← ∅
3:      for each Element in Elements do
4:          SValue ← GetSampleValue(Element)
5:          Profiles[Element] ← ComputeProfile(SValue)
6:      end for
7:      return Profiles
8:  end function
9:
10: function DoProfileMatching(MatchedElements)
11:     SourceProfiles ← ComputeContentProfiles(MatchedElements.source)
12:     TargetProfiles ← ComputeContentProfiles(MatchedElements.target)
13:     MatchedPairs ← ∅
14:     for each SourceElement in MatchedElements.source do
15:         SProfile ← SourceProfiles[SourceElement]
16:         Matches ← ∅
17:         for each TargetElement in MatchedElements.target do
18:             if SProfile = TargetProfiles[TargetElement] then
19:                 Matches ← TargetElement
20:             end if
21:         end for
22:         MatchedPairs[SourceElement] ← Matches
23:     end for
24:     return MatchedPairs
25: end function
```

profile of each source element is compared against the content profiles of target elements one at a time, and if the content profiles are equal, the target element is added to the candidate matches of the source element. In commercial data integration projects, it is typical that companies send one or two lines of data as an example for the integrator. In our implementation, the profile is based on the first value found from the sample data for a given source element. From scarce datasets, reliable mean values cannot be generated so using the first value is often the only feasible option.

3.3 Sorting the Candidates and Combination Phase

Algorithm 3 Algorithm for combining matches and selecting the best

```
1:  function CombineMatchesAndSelect(LinguisticMatches, ContentMatches)
2:      BestCandidate ← ∅
3:      if LinguisticMatches.Length > 0 and ContentMatches.Length > 0 then
4:          Matches ← LinguisticMatches ∩ ContentMatches
5:          if  Maches.Length = 0 then
6:              BestCandidate ← ContentMatches[0]
7:          else
8:              BestCandidate ← Matches[0]
9:          end if
10:     else if ContentMatches.Length > 0 then
11:         BestCandidate ← SelectRandomElement(ContentMatches)
12:     else if LinguisticMatches.Length > 0 then
13:         BestCandidate ← SelectBestCandidate(LinguisticMatches)
14:     end if
15:     return BestCandidate
16: end function
```

Algorithm 3 shows the final step of the TRC-matcher where the results of the linguistic and content profiling methods are combined. The number of synonyms that two elements share defines the quality of candidate match in the linguistic matching approach. This means that the pair with most shared synonyms is considered to be the best match. In the results of content profiling approach, pairs that have equal profiles are counted as candidate matches, but the content profiling algorithm cannot evaluate which pairs are the best candidates as the profile equality is only considered.

If both approaches share candidates, the best match is chosen by selecting the match with the highest amount of shared synonyms. The best match is selected similarly, if only the linguistic approach found candidate matches. In cases where only the content profiling found candidates, the match is selected in random from the set of candidates.

3.4 Comparison to Similar Methods

When compared to previous methods, the most similar algorithm to TRC-Matcher is the XMap from year 2015's OAEI competition. Both methods use WordNet dictionaries, but the key difference is that XMap selects the best synonym set whereas TRC-Matcher uses all synsets. With the one synset of XMap the algorithm is able to find obvious matches more reliably. However, with TRC-Matcher's wider synset more obscure matches can also be found.

One key difference between TRC-Matcher and most of the other matchers is that TRC-Matcher does not use structure (i.e. properties, types and cardinality restrictions) to match elements. This is due to the fact that in OWL files the structure is more important than in XML files which can be almost completely flat. The graph-based methods can be more suitable if structures are available. For example, neural networks can learn structural similarities (i.e. AML and GMap).

4 Evaluation

In this section, TRC-matcher is benchmarked with publicly available data sets and the obtained results are compared against previously published results.

4.1 Data Set

To evaluate TRC-matcher performance, conference data set from OAEI was used. This dataset has been used in OAEI evaluations of ontology matching technologies since 2004. The conference data set includes 16 ontologies from the field of conference organizations and 15 pairwise reference combinations between the ontologies. The goal of the conference test case is to find all correct correspondences within a collection of ontologies describing the domain of conferences [2]. The conference data set has been used widely, so we compare our results against previously published results.

4.2 Test content preprocessing

TRC matcher was initially developed for matching business oriented XML-files. In this section, conversion process for OAEI competition dataset from OWL ontology format to XML schemas is described.

The conversion algorithm tries to preserve majority of the information, because ontology files typically contain more information than XML schemas. However, OWL files do not contain data values. Thus, random synthetic data values were generated for each of the elements in order to benchmark our content profiling approach with the OAEI dataset. The efficiency of the TRC-matcher was stressed with and without the sample data elements.

The first preprocessing step is to convert OWL files to KRSS2 (Knowledge Representation System Specification) [11] which can be seen as a simple ontology file format. Here, the conversion preserves element names, attributes, and parent-child relationships, while, e.g., inheritance information is discarded.

In the second step, the resulting files are converted to XML files without data discards. After the data format conversion steps, ground truths for the matching elements are searched from the OAEI reference results. For each reference pair, a random content profile containing numbers and characters is also generated that match pairwisely together.

4.3 Evaluation Criteria

OAEI evaluates algorithms in terms of precision, recall, and F-measure. The same evaluation criteria is followed here, and we computed our new results based on the published matching formulas for every test case (presented in section 4.4) of the TRC-matching algorithm.

The term precision is the fraction of retrieved matches that are relevant. The numerical range of the precision term is [0,1], and the precision term is near to one when the amount of false matches is small with respect to true matches.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{4.1}$$

The term recall is the fraction of relevant matches that are retrieved in [0,1] numerical range. A high recall means that the algorithm returned most of the relevant matches. In automatic schema matching, both high precision and high recall are essential. If the algorithm returns false matches or if all relevant matches cannot be found, manual work is needed to fix the matches.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{4.2}$$

The term F-measure is the harmonic mean of precision and recall terms, and the F-measrure term is used to combine the results from precision and recall. Therefore, the F-measure term is used to evaluate schema matching algorithms.

$$F\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.3}$$

4.3.1 Results

The new results were computed using four different test setups. Two test setups computed the matching semi-automatically while two were fully automatic. In our first test setup, an integration specialist selected input elements from the source data. The algorithm then searched matches for the selected elements by using the both matching approaches. This setup is quite typical in industrial data source integrations, where the matched source components are known beforehand. In the second test setup, the source elements were again selected, but this time only the linguistic approach was used. This test case evaluated a situation where data values are not available.

The last two test setups were fully automatic. Matches for all source elements were searched without assistance from the integration specialist. As with the semi-automatic test cases, both of our approaches were used in case one, and the linguistic approach was only used in the case two. For the clarify, the following abbreviations are used for the four tests: WV (With Values), WOV (Without Values), FWV (Full With Values) and FWOV (Full Without Values).

The new and previously published results are listed in Figure 1. As can be seen, the F-score varied a lot between different tests. In some cases, almost a perfect

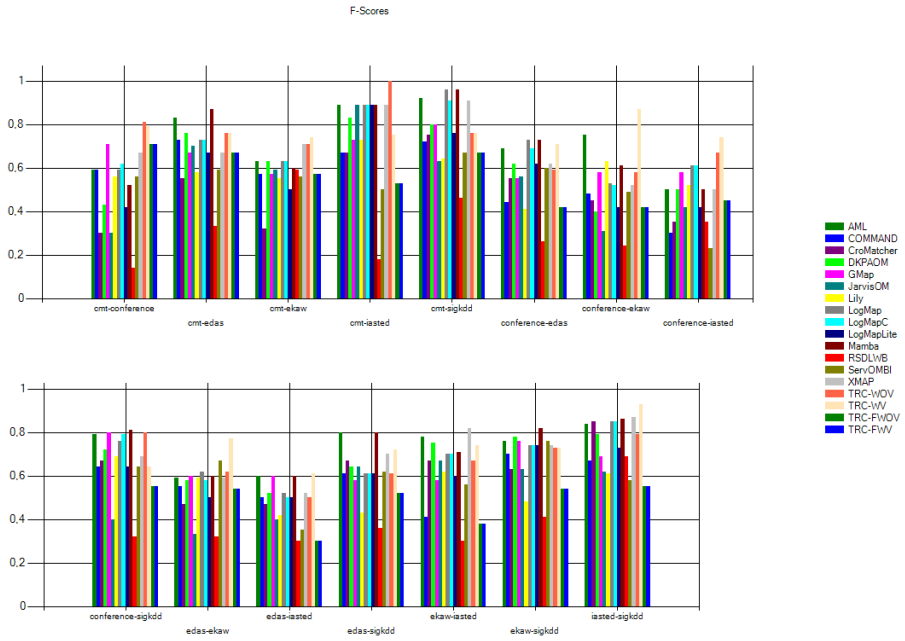

Figure 1: Comparison between evaluated matching algorithms.

F-score was achieved by all of the matching algorithms, while in some cases the mean F-score was only around 0.6. The lower F-score implies that the test data was difficult to match correctly. For example, when test sets had large amounts of elements or multiple elements had similar names, most of the algorithms performed poorly.

Our first test setup (WV) produced the best results in multiple tests. This was especially the case in large test cases where other algorithms generated false positives due to small amount of correct matches in the reference results. Due to this reason, both of our automatic test setups suffered from high false positive rates. The main reason for the poor performance is that our algorithm was originally designed for semi-automatic usage. However, typically the manual work does not take a long time, and does not always require integration experts to accomplish the source element selections.

In Table 1 the evaluated matching algorithms are sorted to decreasing F-score order. It is clear that the user assisted F-scores of TRC-matcher are higher when compared to non user assisted results as the set of source elements is smaller. Additionally, when data values are available, as XML-files often have, the data values can be effectively utilized in the matching process.

Name	F-Score Mean
TRC-WV	0.75
AML	0.73
Mamba	0.73
TRC-WOV	0.71
LogMap	0.70
LogMapC	0.69
XMAP	0.69
DKPAOM	0.65
GMap	0.65
LogMapLite	0.60
COMMAND	0.57
TRC-FWV	0.57
CroMatcher	0.56
Lily	0.56
ServOMBI	0.56
JarvisOM	0.54
TRC-FWOV	0.52
RSDLWB	0.35

Table 1: Matching algorithms ranked from the best to the weakest. Algorithms have been sorted with F-score mean computed from all test cases.

5 Enhanced TRC-matcher

This Section describes enhancements that were implemented to the TRC-matcher in order to improve the matching result reliability. The enhancements utilize optimization and learning algorithms to improve the accuracy of computed schema matching results. The implemented matcher contains two optimization methods and one learning method which are described in this Section. The initial results show that optimization and learning methods improve an F-measure score of matching results by 11% on average.

5.1 Terminology

- *Reliability value* is a scalar value in [0,1.0] range that describes how similar matched elements are to each other
- *Unit matcher* is an algorithm that compares names of matched elements, and returns the reliability value
- *Matcher weight* is a scalar value that is used to scale the reliability value of the unit matcher. Typically the sum of all unit matcher weights is normalized to 1.0
- *Training data* is a set of source and destination elements, and ground truth results which element pairs are correct matches. It should be noted that typically only a subset of source and destination elements generate matches

5.2 Used unit matchers

All algorithms use 5 unit matchers: *synonym matcher*, *longest common substring matcher*, *overlap coefficient matcher*, *simple string matcher*, and *content matcher*.

- The *synonym matcher* splits the elements to tokens, generates synonym sets for each token, and computes how many common synonyms the source and destination elements have with respect to total number of found synonyms.
- The *longest substring matcher* computes the longest substring of the source and destination elements and compares the length of the longest substring against the source and destination element lengths.

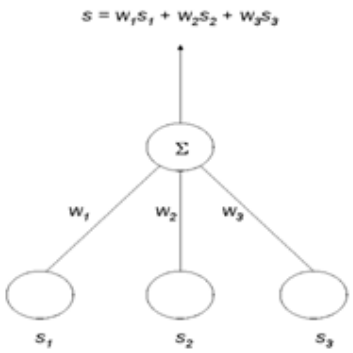

Figure 2: In this example, three unit matchers produce reliability values s1, s2, and s3. Each unit matcher has own scalar weight w1, w2, and w3 that are used to scale the reliability values. The output reliability value (s) of the three unit matchers is the weighted sum s = s1*w1+s2*w2+s3*w3

- The *overlap coefficient matcher* determines how many common characters input and destination elements have and compares the common character count to character counts of source and destination elements.
- The *simple string matcher* splits the source and destination elements to tokens and determines how many equal tokens the source and destination elements have.
- The *content matcher* generates a mask from source and destination data content, and compares the masks against each other to determine match similarity. The details of the content matcher were described in Section 3.2.

5.3 Optimization algorithms

Two optimization algorithms, logistics regression [26] and stable matching [27] methods were implement to the enhanced TRC-matcher. The algorithms were used to optimize matcher weights for the five used unit matchers in order to gain insight how much improvement the two optimization algorithms can provide for the matching results.

The initial setup for both optimization algorithms use weight set of { 0.2, 0.2, 0.2, 0.2, 0.2 }. This denotes that every unit matcher is considered equally important at the beginning. The weight of each unit matcher is then adjusted during the optimization process to produce more accurate matching results according to the training data. The basic idea how the weighted output result of multiple unit matchers is generated is illustrated in Figure 2.

Both optimization algorithms start by generating reliability matrices for each unit matcher. The reliability matrix stores the reliability value for every source and target elements. For example, if the source xml schema has 17 elements, and the

target xml schema has 20 elements, then the reliability matrix size is 17x20 for every unit matcher.

5.3.1 Logistics regression optimization

Logistics regression can be seen as a special case of generalized linear model [26], and therefore the logistics regression model can be used to optimize unit matcher weights in xml schema matching. In general, the logistics regression computes multiple iterations for the training data, and adjusts the unit matcher weights to directions that produce better results, typically at least after multiple iterations (local maximum).

The following formula is used to update k:th unit matcher weight for the iteration t+1 [28]:

$$w_k^{t+1} = w_k^t + \epsilon \sum_{i=1}^{n} y_i s_{ik} g(-y_i s_i) \tag{5.1}$$

where s_i is the weighted reliability value (output sum of all unit matchers), and $-y_i$ is the reference match result from the training data (0 or 1), s_{ik} is the reliability value of the k:th unit matcher, w_k^t is the k:th algorithm weight during the current iteration, $g(z) = \frac{1}{1+e^{-z}}$, is the sigmoid function [29], and ϵ is learning rate scalar.

The current implementation normalizes the updated matcher weights to $[0, 1.0]$ range after every iteration. The test results were generated by running 200 logistics regression iterations for every training data set.

5.3.2 Stable matching

Stable marriage or matching is the problem of finding a stable connection between two sets of elements given an ordering of preferences for each element [30]. The stable matching problem can be used for xml schema matching by generating ordering for element pairs. By using an intuition that the manually matched pairs from the training data should have the maximum reliability values for both rows and columns in the similarity matrices, the unit matcher weights can be optimized via the training data.

Suppose that the maximum reliability value for row i and column j in the reliability matrix are t_r and t_c, then the following weight update rule can be used to adjust the weights of the unit matchers:

$$\Delta w_i = \eta \sum_{d \in D} ((t_r - s_d) + (t_c - s_d)) s_{id} \tag{5.2}$$

where η is the learning rate, s_{id} is the reliability value of the i:th unit matcher for the training sample d, and s_d is the weighted output reliability value of multiple unit matchers.

The current implementation normalizes the updated matcher weights to [0,1.0] range after every stable matching iteration. The test results were generated by running 20 stable matching iterations for every training data set.

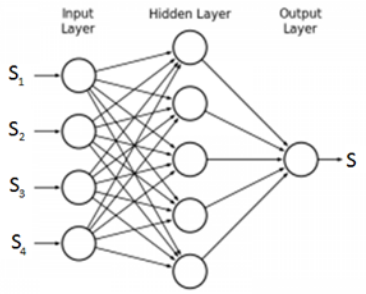

Figure 3: In this example image, four unit matchers provide reliability values (s_1 – s_4) as input to the three layer neural network. The multilayer neural network has relatively complex connectivity between hidden layers, and each neuron (circles) stores weights that are adjusted during the network training process.

5.3.3 Multilayer neural networks

Neural networks have a remarkable ability to derive meaning from complicated or imprecise data. This property can be used to extract patterns and detect trends that are complex to be noticed by either humans or other computer based techniques. A trained neural network can be thought of as an expert in the category of information it has been given to analyze. After the training, the neural network can be used to provide predictions of data that the network has not seen previously.

The problem of XML schema matching can be formulated to neural network friendly form. The selected approach has similarities to the optimization algorithms as can be seen from the Figure 3.

Probably the most significant code modification is to change all functions related to the neural network output values to produce 1.0 result value if the reliability value S is higher than 0.5, and otherwise 0.0. This clamps the neural network output values exactly to the training data result values (0.0 = non-match, and 1.0 = match). Otherwise, the neural network training did not seem to produce reliable matching results.

It is also essential to limit the training elements to contain reasonable amount of true and false matches. Otherwise, e.g., if the source xml schema contains 32 elements, the destination xml schema contains 44 elements, and if the ground truth matching results contain 15 true matches, the number of true matches is negligible when compared to 1408 matching pairs that the two xml schemas have in total. Therefore, if all elements from the training data are taught to the neural network, the network will easily learn all false matches, but no true matches as the amount of true matches is tiny when compared to total amount of training elements.

If the training data have N true matches, the test results presented in the next Section were generated with a ranking system that selects all N true matches from the reference results, and 2N non-matches whose reliability values are the highest. This selection strategy seems to generate reasonable good training sets that are able to improve matching results when compared to non-trained reference matcher.

	TRCM without learning	Logistics Regression optimization	LR improvement
edas-sigkdd	0,583333333	0,695652174	11,23 %
iasted-sigkdd	0,733333333	0,866666667	13,33 %
cmt-iasted	0,727272727	0,888888889	16,16 %
edas-ekaw	0,5	0,585365854	8,54 %
conference-edas	0,545454545	0,666666667	12,12 %
cmt-ekaw	0,631578947	0,666666667	3,51 %
conference-iasted	0,413793103	0,428571429	1,48 %
conference-ekaw	0,41509434	0,608695652	19,36 %
edas-iasted	0,43902439	0,625	18,60 %
cmt-edas	0,695652174	0,761904762	6,63 %
conference-sigkdd	0,689655172	0,814814815	12,52 %
ekaw-sigkdd	0,857142857	0,857142857	0,00 %
cmt-conference	0,642857143	0,740740741	9,79 %
ekaw-iasted	0,466666667	0,761904762	29,52 %
cmt-sigkdd	0,8	0,909090909	10,91 %
Average improvement			11,58 %

Figure 4: Summary table of F-measure scores for the logistics regression method.

	TRCM without learning	Stable Matching optimization	SM improvement
edas-sigkdd	0,583333333	0,695652174	11,23 %
iasted-sigkdd	0,733333333	0,838709677	10,54 %
cmt-iasted	0,727272727	0,727272727	0,00 %
edas-ekaw	0,5	0,585365854	8,54 %
conference-edas	0,545454545	0,625	7,95 %
cmt-ekaw	0,631578947	0,666666667	3,51 %
conference-iasted	0,413793103	0,421052632	0,73 %
conference-ekaw	0,41509434	0,608695652	19,36 %
edas-iasted	0,43902439	0,625	18,60 %
cmt-edas	0,695652174	0,761904762	6,63 %
conference-sigkdd	0,689655172	0,714285714	2,46 %
ekaw-sigkdd	0,857142857	0,857142857	0,00 %
cmt-conference	0,642857143	0,785714286	14,29 %
ekaw-iasted	0,466666667	0,761904762	29,52 %
cmt-sigkdd	0,8	0,909090909	10,91 %
Average improvement			9,62 %

Figure 5: Summary table of F-measure scores for the stable matching algorithm.

5.3.4 Results

The test results presented in this subsection are based on publicly available OAEI competition dataset [31] .The datasets are distributed in Web Ontology Language (OWL) data format [32]. The quality of matching results were evaluated using the formulas that were already described in Section 4.3.

Figure 4, Figure 5, and Figure 6 lists the detailed numbers of the F-measure scores from different OAIE datasets. As can be seen, the optimization and learning methods improved the F-measure scores from 9,62% up to 12,16% on average when compared to TRC matcher whose unit matcher weights were not optimized. One thing to note from the neural network learning results is that there are two cases where the neural network was not able improve the F-measure results. There are two possible explanations why the training did not succeed well in these cases. Either the training data generation strategy was not able to produce high quality training

	TRCM without learning	Neural Network learning	NN improvement
edas-sigkdd	0,583333333	0,8	21,67 %
iasted-sigkdd	0,733333333	0,965517241	23,22 %
cmt-iasted	0,727272727	0,857142857	12,99 %
edas-ekaw	0,5	0,666666667	16,67 %
conference-edas	0,545454545	0,6875	14,20 %
cmt-ekaw	0,631578947	0,727272727	9,57 %
conference-iasted	0,413793103	0,244897959	-16,89 %
conference-ekaw	0,41509434	0,744186047	32,91 %
edas-iasted	0,43902439	0,666666667	22,76 %
cmt-edas	0,695652174	0,761904762	6,63 %
conference-sigkdd	0,689655172	0,769230769	7,96 %
ekaw-sigkdd	0,857142857	0,583333333	-27,38 %
cmt-conference	0,642857143	0,692307692	4,95 %
ekaw-iasted	0,466666667	0,888888889	42,22 %
cmt-sigkdd	0,8	0,909090909	10,91 %
Average improvement			12,16 %

Figure 6: Summary table of F-measure scores for the neural network based learning method.

sets in these two cases, or the used unit matchers were not enough accurate (noisy) in these two cases for the neural network training. However, if the learned neural network weights were taken from other datasets, it seems that the neural network was able improve the F-measure results also in these two cases. Verifying and validating this assumption is left to future work.

6 Additional custom enhancements

The enhanced TRC-matcher was further customized for commercial requirements for a company called Integration House (IH) [33], who does commercial data system integrations

Integration House has an own Java-based environment where the futher enhanced multilayer neural network approach was implemented. The source code that was written is IH proprietary, but this chapter describes the implemented system on a high level. The basic idea of the Java-based custom implementation is similar to the neural network approach that is already described in the previous Subsection 5.3.3 multilayer neural network.

The implemented system contains three unit matchers from the previously described neural network based matcher: longest common substring matcher, synonym matcher, and substring equality matcher. The Java implementations of the three unit matchers are very similar to the previously written C#-based implementations, which denotes that the three Java-based three unit matchers are not explained in detail second time. The overall matching system also contains three other unit matchers that are IH proprietary. Therefore, the three other unit matchers are not described at all.

6.1 New test material

Previously used test material was generated from the publicly available OWL files, and the test files were converted with the semi-manual processes to appropriate internal data formats as described previously in Section 4.2. The semi-manual conversion process did not preserve the structure of the matcher pairs, i.e. the path parts of the matched pairs, and only the field names, the actual names of the data elements, were written out from the conversion process. The new Java âĂŞbased matcher implementation used new proprietary test material that IH generated. The new test data contained structural information also. The following example illustrates one structured element pair:

$$Customer/Order/Address \rightarrow Org/Name/ID/HomeAdd \tag{6.1}$$

In this example, $Customer/Order$ and $Org/Name/ID$ are called as path parts

of the matched elements while *Address* and *HomeAdd* are called as field names. The path parts represent the structure of the matched pairs while the field names are the actual names of the matched data elements. The new test data did not contain actual data values for the matched elements. Therefore, analysis of the data value similarities was not used in any of the used unit matcher.

6.2 Path and field name matching

As the new test material contains path parts, and as the path parts do not often contain almost any similarity in the correctly matched pairs, the previously implemented C#-based matching system did not perform very efficiently. This was due to the lack of similarity, especially in the path names. The field names contained more similarity, even though sometimes the field names were not possible to be matched without appropriate synonym and abbreviation databases.

6.3 Canonical destination model

The implemented matching system used a canonical destination model. This denotes that all source schemas were always mapped to one static destination schema. As the destination schema is static, synonym and abbreviation dictionaries were only generated for source schemas. From academic point of view, it is not fully obvious what the main advantages of canonical model are. The canonical model can be seen as one merged, large destination schema instead of multiple smaller destination schemas.

6.4 Path preprocessor

One major contribution for getting the matcher system to work with practical test data was the invention and implementation of a path preprocessor algorithm. The central idea of the path preprocessor method is to build a database for path synonyms and abbreviations from the correct reference results.

For every unique element path that is stored in the canonical model, the path preprocessor database stores all source paths that are mapped to certain destination element in the canonical model. This denotes that every canonical model path element, that is unique, has a list of source paths that are correct matches to the canonical model path in some training files.

The path preprocessor method generates an active dictionary for the best path synonyms that have been frequently been matched to unique paths in the canonical model. The active dictionary generation is required, as the unique paths in the canonical model may have been matched to multiple source paths. Therefore, a heuristic has to be used to select paths to the active path dictionary. Few heuristics were tested during the fast implementation work of the path preprocessor.

At the end, the used heuristic selected the path that has the highest match count, e.g., the source path that was matched to the unique path of the canonical model

most often. Additionally, the source path was not allowed to be longer that the destination path. If there were multiple source paths that had equal match counts, the shortest source path was selected to the active synonym path dictionary. There was also a low level threshold that limited source path selection to paths that were mapped more than twice to the unique destination path in the canonical model.

The implemented path preprocessor generated the database for all source path mappings and generated the active path synonym dictionary automatically from the used training files. The path preprocessor also contained load and save functionalities for the main and active path synonym databases. By having the load and store capabilities, only the new training files need to be processed when the path preprocessor databases are being updated.

6.5 Field name synonym dictionary for the synonym unit matcher

In addition to the path synonym dictionary, a field synonym dictionary is generated automatically from the training data. The field synonym data is used in the synonym unit matcher. The main idea of the field synonym dictionary is to provide a synonym database for the remaining parts of the path and field parts that the path preprocessor does not handle properly.

Before the field synonym dictionary generation begins, the source paths are processed with the path preprocessor. This denotes that the path preprocessor replaces the path names that are found from the active path synonym dictionary. The path preprocessor is run for all three TRC based unit matchers to make the path parts more similar in case of potentially correct matches. After the path preprocessor has modified the source paths, the path parts and the field names, that are not equal to the destination entry in the canonical model, are inserted to the field synonym dictionary in order to produce correct matches with the synonym unit matcher. When the source path part is shorter than the mapped destination path from the canonical model, the field name of the source path is stored as the key in synonym dictionary, and the remaining path parts of the source and the field name of the destination canonical model entry are inserted to the data entry slot in the dictionary. Otherwise, only the field part is inserted to the data field when the field parts are different. If the same key, the source field name, occurs in multiple source paths, all non-equal destination field names, and potential path parts are inserted to the data entry of the corresponding dictionary slot.

6.6 Summary of the matching dataflow

The overall dataflow of the Java-based matching system has many similarities to the neural network based matching system that was described previously. The biggest difference to the previously described data flow is the usage of the path preprocessor algorithm. The path preprocessor method modifies parts of the source paths, and the

path preprocessor is executed for all of the three unit matcher that were implemented by a TRC employee.

By replacing the source paths parts via synonym paths that are found from the active path synonym dictionary, the path parts of true matches are made similar to the destination paths according to the training data. As the synonym path dictionary is generated only from the true matches of training data, the number of false positive matches is reduced. This is due to significant differences between the path parts of the matched elements. Thus, the path preprocessor often improves the reliability of finding true matches and reduces the probability of finding false true matches.

After the source path parts have been processed with the path preprocessor algorithm, the three unit matchers start to compute the match reliability values for the incoming source and destination pairs. All unit matchers compute the internal similarities for both path and field parts separately and use weighed linear combinations to produce the output reliability values per unit matcher. In the previous C#-based implementation, unit matchers compute the reliability values based on the field names only as the path parts were lost during the input data conversion process. The computation of the path part similarity as a part of the unit matcher reliability value is another significant change when Java-based implementation is compared to the previous C#-based implementation.

The output reliability values from the unit matchers are then fed to the multilayer neural network to produce the final output reliability value for the matched element pair. The multilayer neural network use internal weights that have been produced during the neural network training process. The usage of the multilayer neural network is similar in both C# and java-based implementations. Actually, the Java-based implementation uses the same multilayer neural network implementation as the C#-implementation, expect that the C#-code has been ported to Java.

7 Discussions and Future Work

Although our software is mainly targeted for semi-automatic usage, the partial automation is an improvement over the current real-world situation, where all data source integrations are done manually. By using the TRC-matcher, the integration specialist only selects the elements that need to be matched, and the manual matching work that is the most time consuming part of the matching process is not needed. The long term plan is to improve the whole integration process to a level where almost anyone could perform the matching process without the need of integration specialists.

Our short term topic for future work is to improve the robustness of our linguistic matching approach in the fully automatic mode. The biggest challenge in the full automation is to develop algorithms for pruning out irrelevant source elements. At the moment most of our false positives are due to the elements that do not need to be matched at all.

After the enhancements were implemented to the basic TRC matcher algorithm, the robustness of matching results improved significantly when good training data was available for the neural network based enhancements and automatic synonym dictionary algorithm. Especially the automated generation of the synonym dictionary improved the reliability of enhanced matching system, because quite often the input data path and field name did not have much linguistic correspondence to the target data path and field name. When all major enhancements were enabled, the integration specialist was not required to reduce the amount of potential matching pairs with the used test material.

One interesting topic for future work would be to benchmark the enhanced TRC matcher system with publicly available test data sets, compute reliability scores and compare the results against previous published scores. Typically the publicly available test data sets contain more linguistic correspondence between the source and target matching pairs when compared to the real world test data sets that were used in the made evaluations. Therefore, publicly available test data sets would stress the enhanced TRC matcher from different viewpoint.

Another interesting topic for future work would be to collect a large data set from commercial integration cases and to stress the enhanced TRC matcher algorithm with various real world integration cases. It is likely that the large commercial test data set would reveal some new weaknesses from the enhanced TRC matching system. However, it would be very interesting to develop new innovative solutions to overcome the potential weaknesses as publicly available test data sets seem to have

more similarities in the matching pairs than the commercial test data set that were used in development of the enhanced TRC matcher.

Depending on how well the enhanced TRC matcher would work on large commercial data sets, one additional avenue for future work would be to study how far the neural network training and automated synonym dictionary generation could be generalized. Potentially some grouping based on integration case type could be required for good matching performance due to specialized dictionaries and used neural network weights.

8 Conclusions

In this paper, a novel schema matching algorithm based on linguistic methods and content profiling was presented. The algorithm works in five steps: (1) element names are tokenized based on special characters and capital letters, (2) synonyms are searched for tokens from abbreviation dictionaries and WordNet synonym sets, (3) name-based matches are computed for the tokenized elements with linguistic matching algorithm, (4) content profiling algorithm searches elements with similar content profiles when data values are available, and finally (5) the matching results are combined and the best match is selected.

The presented algorithm works most efficiently in cases where user assistance is used to select the input elements and elements have data values available. We demonstrated the effectiveness of TRC-matcher by using the OAEI's Conference dataset and by comparing the new results against the results from 2015's competition. In many test cases, TRC matcher produced accurate results, but the most challenging cases for TRC-matcher were data sets that contained a large number of elements while only few elements need to be matched. In such cases, the number of false positives appeared to be high.

Several enhancements for the basic TRC matcher algorithm were also presented. The most significant enhancements were the neural network based training method and automated synonym dictionary generation algorithm. These enhancements improved the accuracy of matching results significantly when compared to the basic algorithm. In general, very encouraging preliminary results were achieved using the commercial test data where basic linguistic methods performed poorly due to lack of similarity in the source and target matching pairs.

As publicly available training data sets can be quite different from commercial test data sets, development of new innovative matching solutions is still required to build commercial level matcher systems. New solutions even for limited integration case types can provide significant cost savings for commercial data system integrations.

Acknowledgements

The research has been carried out during the MARIN2 project (Mobile Mixed Reality Applications for Professional Use) funded by Tekes (The Finnish Funding Agency for Innovation) in collaboration with partners; Defour, Destia, Granlund, Infrakit,

Integration House, Lloyd's Register, Nextfour Group, Meyer Turku, BuildingSMART Finland, Machine Technology Center Turku and Turku Science Park.

Bibliography

[1] Alhassan, B.B., Junaidu, S.B., Obiniyi, A.: Extending an ontology alignment system with a lexical database. In: Scientific Research Journal 3(1), 12–17 (2015)

[2] Dragisic, Z., Eckert, K., Euzenat, J., Faria, D., Ferrara, A., Granada, R., Ivanova, V., Jiménez-Ruiz, E., Kempf, A.O., Lambrix, P., et al.: Results of the ontology alignment evaluation initiative 2014. In: Proceedings of the 9th International Workshop on Ontology Matching Collocated with the 13th International Semantic Web Conference (ISWC 2014) (2014)

[3] FuzzyString - Approximate String Comparison in C#. `https://fuzzystring.codeplex.com/`

[4] Garcia-Molina, H.: Database systems: the complete book. Pearson Education India (2008)

[5] Gusfield, D.: Algorithms on strings, trees and sequences: computer science and computational biology. Cambridge university press (1997)

[6] Jian, N., Hu, W., Cheng, G., Qu, Y.: Falcon-ao: Aligning ontologies with falcon. In: Proceedings of K-CAP Workshop on Integrating Ontologies, 85–91 (2005)

[7] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. In: Soviet physics doklady. vol.10, 707–710 (1966)

[8] Lu, J., Lin, C., Wang, W., Li, C., Wang, H.: String similarity measures and joins with synonyms. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, 373–384. ACM (2013)

[9] Miller, G.A.: Wordnet: A lexical database for english. Commun. ACM 38(11), 39–41 (Nov 1995), `http://doi.acm.org/10.1145/219717.219748`

[10] Monge, A.E., Elkan, C.P.: The webfind tool for finding scientific papers over the worldwide web. In: Proceedings of the 3rd International Congress on Computer Science Research, 41–46 (1996)

[11] Patel-Schneider, P., Swartout, B.: Description logic specification from the krss effort. (1993)

[12] Po, L., Sorrentino, S.: Automatic generation of probabilistic relationships for improving schema matching. Information Systems 36(2), 192–208 (2011)

[13] Russell, R., Odell, M.: Soundex. US Patent 1261167 (1918)

[14] Winkler, W.E.: String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. ERIC (1990)

[15] STANDS4 Web Services: Abbreviations API. http://www.abbreviations.com/abbr_api.php

[16] Zarembo, I., Teilans, A., Rausis, A., Buls, J.: Assessment of name based algorithms for land administration ontology matching. Procedia Computer Science 43, 53–61 (2015)

[17] Mukkala, L., Arvo, J., Lehtonen, T., Knuutila, T.: Current State of Ontology Matching. A Survey of Ontology and Schema Matching. University of Turku Technical Reports 4 (2015)

[18] Djeddi, W. E., Khadir, M. T., Yahia, S. B.: XMap: Results for OAEI 2015. OAEI (2015)

[19] Meilicke, C: MAMBA-Results for the OAEI. OAEI (2015)

[20] Li, W., Sun, Q: GMap: Results for OAEI. OAEI (2015)

[21] Faria, D., Martins, C., Nanavaty, A., Oliveira, D., Sowkarthiga, B., Taheri, A., Cruz, I. F: AML Results for OAEI 2015. OAEI (2015)

[22] Cruz, I. F., Antonelli, F. P., Stroe, C.: AgreementMaker: efficient matching for large real-world schemas and ontologies. Proceedings of the VLDB Endowment, 2(2), 1586-1589. (2009)

[23] Shvaiko, P., Euzenat, J.: Ontology matching: state of the art and future challenges. In: Knowledge and Data Engineering. IEEE Transactions, pp. 158-176 (2013).

[24] Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. Journal on Data Semantics IV. Springer, 146-âĂŞ171 ()2005).

[25] Bernstein, P., Madhavan, J., Rahm, E.: Generic schema matching, ten years later. Proceedings of the VLDB Endowment, 695-701 (2011)

[26] Walker, SH., Duncan, DB.: Estimation of the probability of an event as a function of several independent variables. Biometrika. 54: 167-178 (1967))

[27] Gusfield, D.; Irving, R. W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press. p. 54 (1989)

[28] Jason Rennie, Lecture notes, 2003. http://people.csail.mit.edu/jrennie/writing/lr.pdf

[29] Duan S„ Fokoue A., and Srinivas K.: One Size Does Not Fit All: Customizing Ontology Alignment Using User Feedback. Proceedings of the 9th International Semantic Web Conference on The Semantic Web, 177–192, 2010.

[30] https://en.wikipedia.org/wiki/Stable_marriage_problem

[31] http://oaei.ontologymatching.org/2015/

[32] https://en.wikipedia.org/wiki/Web_Ontology_Language

[33] http://integrationhouse.fi/