P-ISSN 2586-9000
E-ISSN 2586-9027
Homepage : *https://tci-thaijo.org/index.php/SciTechAsia* **Science & Technology Asia**
Vol. 22 No.3 July - September 2017 **Page: [93 - 111]**
Original research article

Enabling Fuzzy Logic to Enhance Automatic Schema Matching

Akarajit Tanjana[1,*] and Jian Qu [1]
Master of Science in Information Technology, School of Science and Technology, Shinawatra University, 99 Moo 10, Tambon Bangtoey, Amphoe Samkok, Pathum Thani 12160, Thailand

Received 1 April 2017; Received in revised form 19 May 2017
Accepted 14 June 2017; Available online 12 October 2017

ABSTRACT

Automatic schema matching is a process to find correspondences among different data attributes from either databases or XML schemas. Since there is an inconsistency for naming attributes, the schema matching which is done by humans is the most practical; however, it is time-consuming and incurs great expense. Therefore, automatic schema matching process has been extensively studied in the past. Most works still face many challenges such as abbreviation, synonym, hypernym, and structural problems. Some existing works take schema name, instance, data type and schema description as internal resources while other works employ external resources, such as several online dictionaries and ontologies, to increase accuracy for schema matching.

In this paper, we address automatic matching problems by employing abbreviation, synonym, and hypernym lists; furthermore, we propose a novel structure similarity algorithm. Finally, we propose to use fuzzy logic, a novel fuzzy scoring algorithm to increase the accuracy of our system. As comparing our systems with existing works on open data; we find that our system outperforms existing works with an f-measure of 90%.

Keywords: Automatic schema matching; Fuzzy logic

1. Introduction

Schema matching has been studied and enhanced by a lot of research in database, artificial intelligence, and information retrieval communities for over three decades [1]. This problem is still a challenge because there are no standards for naming each attribute, so different systems may refer to the same object by using different names. Thus, it is difficult for an automatic schema matcher to be successful. Moreover, it could be even more complex and difficult for human or automatic schema matcher to solve if there are many attributes or complex structure in database or XML schema. For example, first name and last name of the first schema could concatenate as a single attribute in the second schema. An attribute could represent in different forms by using

*Corresponding author: akarajit.tjn@gmail.com **doi**: 10.14456/scitechasia.2017.8

shortened name or abbreviations such as tel, phone or telephone. In addition, an attribute that refers to the same data could be named in different terms such as item, product or model. In contrast, same attribute name on two schemas could refer to different data. For example, "area" could refer to the physical location or space of a building. Those problems cause confusing to both human and software to select the correct pairs among matching process.

There are four major challenges on matching problems [1,2] . Firstly, only attribute names and schema structure do not describe clear meaning, such as which item or entity do they represent, so human analysts have to extract such meaning by reading unclear support documents to get more clues, which is time- consuming and expensive. Secondly, attribute names on one schema may not contain adequate clues and relationships among others. Paths and structures could add more clues, but they are incomplete. Thirdly, human analysts have to examine other attribute candidates to ensure that an expected attribute is the most appropriate one, which is time- consuming and can cause confusion. In particular, similar terms are found and no description is provided. Lastly, it could depend on creators who may name attribute on their own term such as a description of a house can be stored into house- style, house- info or house- description attribute which may be different to other creators and may also mislead users, thus it is difficult for automatic schema matcher to be successful.

As shown in Fig. 1, the relationship of two schemas, CIDX and APERTUM, can be formed in 1:1, 1:m or m:1. The relationships of related attributes are connected by thin lines, while the relationships of related groups of attributes are connected by thick lines. Several pairs of attributes could be matched easily if they are 1:1 relationship, such as " city" to " city"; " postcode" to "postalCode", but they could be difficult to match for m:1 relationship, such as "street1" and "street2" that have to be merged into one attribute " address." A group of attributes such as "Contact" on CIDX could refers to several groups of APERTUM as 1:m relationship. In contrast, " contactFunctionCode" and "contactEmail" attributes do not match to any attribute on APERTUM, but they seem similar to other attributes which contain the word "contact." Those relationships can distinguish easily by human because of their experiences, but those relationships would reduce accuracy to automatic schema matching process because of relaxation of an algorithm to map 1:m and m:1 relationship.

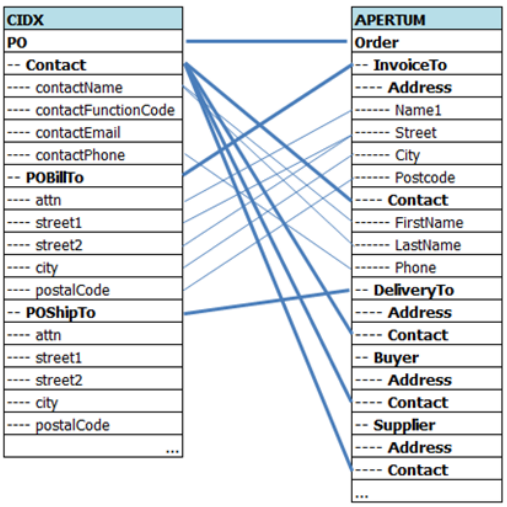

Fig. 1. Example of matching between two purchase order schemas.

According to several causes of heterogeneity of structure, naming and constraints, we come up with an idea to apply fuzzy logic on schema matching because this method is capable of handling uncertainty or vagueness of inputs by grading those inputs with membership functions then process them before making the final decision. The processes to grade inputs, to process graded data sets by fuzzy rules, then to compute final outputs are called fuzzifier, inference and defuzzifier respectively. This allows experts to add their judgment on several fuzzy rules to enhance qualities of their works.

Overview process of automatic schema matcher is shown in Fig. 2. The main process, at the middle, takes two schemas as inputs then it calculates the similarity of each attribute between both schemas. After that, suggested pairs are generated to users without or with a few of human adjustment. This matching process is necessary for several kinds of data activities such as data integration, data migration or building a global schema for master data management project. In particular, data integration project between in- house and out- of- the- box application, each of them uses their terminology and needs many experts to be involved on a matching task.

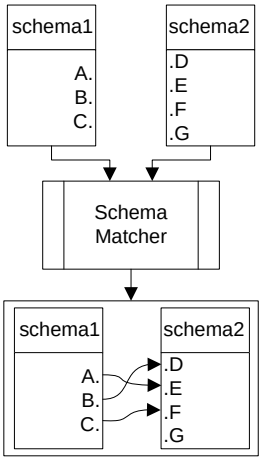

Fig. 2. Overview process of automatic schema matcher.

Without automatic schema matcher, a project could slip from the initial plan because of the needed time for a human to understand all data structure and meaning of each attribute. Moreover, mapping and validating tasks incur the same problems as matching task does.

This paper is organized as follows. In section 2, several prototypes and techniques are discussed. In section 3, our matcher prototype that employs fuzzy logic methodology is proposed. In section 4, results of our experiments and discussion are explained. Finally, conclusion and future work are provided.

2. Related Work

Schema matching solution can be roughly put into two groups: rule-based and learning-based [3,4]. The rule-based solution can be used immediately without providing training data set or previous matching result. This solution is almost straight forward, so the outcome of matching is very fast. In contrast, the learning- based solution takes many previous matching results as the training data. This solution would overcome the rule-based solution when it accumulates adequate information such as description and instance [9]. Sophisticated information retrieval techniques such as Naïve Bayes is capable of building "probabilistic rules" to distinguish between two attributes, but adequate information is difficult to prepare and compile [7, 9]. Moreover, users have to understand that method insightfully. Thus, our work only focuses on the rule- based solution because adding fuzzy logic over learning-based will add more complexity to it.

2.1 Rule-based matcher

For rule- based matchers, we can further classify into three major groups by different type of inputs: instance- based, schema-based, and hybrid [3].

Instance- based matching is an approach to determine correspondences by analyzing data patterns or data format of sample data. This approach is based on the fact that, schemas do not provide adequate information. Formats of data are capable of identifying between ISBN, telephone number or zip code due to the length of the data, data type, and symbols. For example, if "ANRED" and "LAND1" are matched to "FormOf-AddresCode" and "CountryCode" respectively, it's impossible for schema-only matcher to understand those words by using linguistic knowledge. QuickMig is an instance matcher to solve this problem by analyzing samples of instance using domain

ontology [15]. Although it could gain high precision, the recall is only above average because of several entity names used on SAP schemas are very obscure. If there are obvious data on two data sets, users can ignore schema name and search them by specifying sample data such as movie title or authors name, and then matching tools will create correspondence automatically. However, searching by number is still limited [10].

Schema- based matching will play around attribute name, data type, and description. In addition, recursive technique and graph would be required if a schema is structured more than one level such as a schema in XSD format. COMA is a framework to combine several matching techniques such as affix, n-gram, sounddex, edit distance, synonym and data type by using only information available on schema [5]. This work tries to find out the best strategy to combine results from several techniques including aggregation, direction, and selection. Although the results are good, users have to be familiar with this system a lot to tune up several parameters, adjust thresholds and add many entries on an auxiliary list until the results reach an acceptable level.

Compound names and abbreviations are commonly used for naming attributes in schemas. We can create a user-defined list for changing abbreviation or shorten form to long form, however, online data always provides richer information. Sorrentino et al. proposed a matcher taking both internal resources like user defined dictionary and several external resources such as WordNet and Abbreviation. com to examine abbreviation and they also studied methodology to select appropriate full words from a given abbreviation so that schema of any domain could take advantage of this work such as e-commerce and bibliographic data. Synonym and hypernym are also included in this work. The authors also proved that after normalization the quality of schema matching can be improved [6].

A matcher could be enhanced by combining several methods in a fixed or flexible way; those are hybrid and composite approach respectively. The hybrid approach is an approach to improve the quality of matchers by taking both schema information and instance to examine correspondences. This solution often comes up with a fixed strategy that is well tuned by human, but it would not fit for all problem domains. In contrast, a composite approach offers more flexibility to users to select algorithms for its library, allowing users to arrange the sequence of available algorithms and adjust thresholds to encounter different matching task. However, users need several experiments to get a good result.

There are two survey papers worth mentioning. They are survey paper of Ergard Rahm et al. in 2001 and in 2011. Both papers compiled available techniques and tools used for semantic matching area [3,4]. The first survey was published in 2001 to categorize several matchers while the second survey was published ten years later to refresh and also introduced several challenges remaining in this research area. Several new algorithms had been developed by taking further information such as relationships between elements in their structure by using graph technique. Query log of a database is capable of developing a taxonomy path on usage-based matching. The similarity of contents and similarity of links in documents can be measured by TF- IDF (term frequency-inverse document frequency). Strategies to solve the matching problem had also been proposed by self- tuning match workflow. Matching multiple schemas at once in the same domain using reused-based and holistic matching had been introduced. How users interaction with and provide feedback to the matching task had been tested by supporting GUI, incremental matching, Top-K matching and collaborative method. Finally, semantic tagging and conditional tagging were tested

to improve the result of finding correspondences by adding several constraints from instance such as if data on "productType" attribute refers to a book, "productCode" should map to ISBN.

In the recent survey, some schema matching tools are evolved to support ontology matching and many of them are not well maintain [17,18]. The academic tools are Cupid, COMA, Falcon-AO, RiMON, ASMOV, Agreement Maker and OpenII-Harmony, while, commercial tools are Altova Mapforce, IBM Infosphere, Microsoft BizTalk and SAP Netweaver.

Academic tools provide more sophisticated methods including using ontology to address the matching problem. On the other hand, commercial tools provide just simple methods based on linguistic matching, but they offer better GUI for manual matching task. In addition, ontology plays important role in matching tools because words and their relationship that are prepared systematically offer matching tools measure the similarity better than using a thesaurus. Relationships of words on ontology are much variety depends on creator such as broader, narrower or related [16].

2.2 Fuzzy Logic

Fuzzy logic was introduced by Lotfi Zadeh since 1965 based on the fact that human judge or draw conclusions by using approximate and vague inputs [11,12]. For example, humans can decide whether to turn on an electric fan or air-conditioner fuzzily on weather condition. Cold, cool, normal, warm and hot are grades for temperature. No precise value for individuals, but a human can define a rank for them easily.

Fig. 3 illustrates a typical block diagram of fuzzy logic. There are four main parts for a fuzzy logic system: fuzzifier, inference, fuzzy rules and defuzzifier.

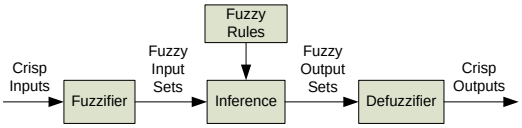

Fig. 3. Block diagram of fuzzy logic.

Fuzzifier is the entrance of a fuzzy system for receiving crisp inputs and transforms them to fuzzy input sets, graded by membership functions. Common fuzzy input sets consist of several values and those values are in real number between 0 and 1. One means full membership, zero means no relationship at all. For example, the average temperature in Bangkok is between 25-33 degrees Celcius; the temperature below 0 degree Celsius is not a member of the "Hot" membership at all, so fuzzy value is 0; the temperature is 35 degrees Celsius or above is certainly hot so the fuzzy value is 1 for "Hot" membership. In addition, a crisp value may produce several fuzzy input sets if membership functions overlap. For example, 30 degrees Celsius produces 0.7 membership of "Warm" and 0.3 membership of "Hot". After Fuzzy input sets are calculated, fuzzy output sets are proceeded by inference process that consists of several fuzzy rules, provided by experts.

Lastly, crisp outputs are calculated by defuzzifier process. Maximum value and center of gravity (COG) are standard methods of defuzzifier [11].

Fuzzy tools could be provided as an integrated module such as MatLab; dedicated software or even software library. jFuzzy Logic is a Java library that supports Fuzzy Control Language (FCL) defined in IEC1131-7 standard with graphical support to verify Fuzzy model [13].

Among a number of prototypes to address schema matching problem, we chose to create schema-based approach with fuzzy logic to examine attribute names of a pair of schemas because of following reasons. The first reason is the reliability of the result because this kind of matcher takes only

schema name and auxiliary lists, so it always produces consistent result. The second reason is schema-based approaches execute faster than instance-based approach because the whole structure of the schema needs just a few kilobytes in memory. In contrast, instant-based matchers take time for loading and analyzing data from files or database. The third reason is schema-based is capable of extending its capability easily by adding new terms into abbreviation, synonyms and hypernym list. That's similar to how humans learn linguistics.

3. Proposed Prototype

We propose to solve schema matching with two approaches, the first approach using abbreviation and synonym and weighted hyponym/ hypernym, and combining with our novel structure position algorithm. For the second approach, we try to further improve the quality of schema matching by adding our novel fuzzy logic system. Finally, we use bootstrap training to learn the rules for our fuzzy logic system to further increase the final results. The process of our matcher is shown in Fig. 4.

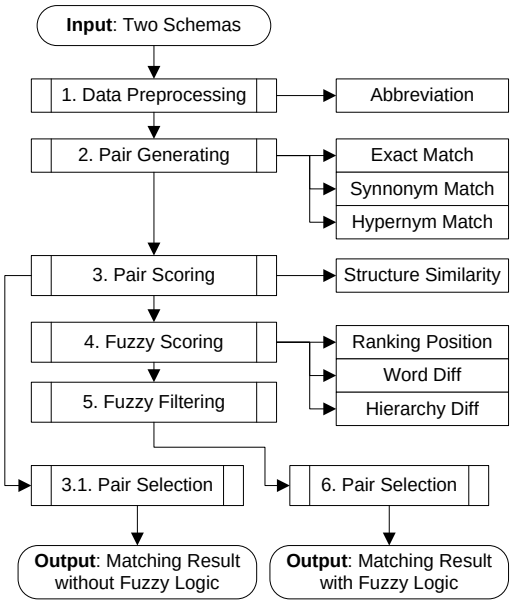

Fig. 4. Process of our matcher.

3.1 Process of our matcher without fuzzy Logic

3.1.1 Data Preprocessing

The schema files are originally provided in XDR format and many previous works convert them into graph or tree model [5, 19-23]. However, we transform those schemas into text into $X_1.X_2.X_3 \ldots X_n$ format by using XSLT scripts, where X_i refers to element name of the schema from root node to leaf node with a dot as the separator. Our format is easy to compare to the reference result and it can be validated with ease by string function. Then, our matcher prototype starts by reading two converted schema files and several auxiliary lists including abbreviation, synonym, and hypernym. After that, our matcher split each entry of both schemas from compound words to single words with capitalized segmentation method and then replace those single words by their matched abbreviation. It is worth noting that we have to do this before replacing because simply replacing attribute names by string function would fail if all characters of any abbreviations appear as a part of compound words such as "PostalCode" must not be replaced by "Po" to "PurchaseOrderCode", or "Desc" and "Description" must not repeat itself to "Descriptionription".

Abbreviation is commonly used for designing schema to simplify attribute names so that attributes are easy to read and also comply with some constraints of tools such as Oracle database, which allows only 30 characters for naming the database object like table name, index name or column name. But, abbreviating words or shorten word can add confusion if they are similar to well-known words, such as "column" or "color" both of them can be abbreviated as "col" [6]. Moreover, the words "Tel" or "phone" can both be referred to telephone number. Above examples shows that abbreviation can be a big problem for schema matching and sometimes cause matchers to produce wrong suggested candidates if they are not managed properly.

In order to address such abbreviation problem, we propose an abbreviation list for matching. Some of our abbreviations for our sample schemas are provided in Table 1.

Synonym list is created because humans can describe the same object by using different words so this is the necessity for any schema matcher to support this list [5-6], for example, "state" and "country"; "company" and "organization", those refer to the same thing. Therefore, we try to solve such synonym problem by making a synonym list.

Table 1. Sample of abbreviation list.

Abbreviation/ Shorten Form	*Long Form*
Po	PurchaseOrder
qty	Quantity
ord	Order
mod	Model
col	Color
post	Postal
uom	UnitOfMeasure
no	Number
attn	Attention
ref	Reference
sub	Supplier
desc	Description
tel	Telephone
phone	Telephone
del	Delivery

Hyponym/ hypernym is another problem for schema matching because the degree of relationship between two words would influence selecting candidates, such as "address" and "street". The word "address" usually contains several data include "house number", "street", "city" or "country" and "postcode". If they are compiled systematically, the list could be promoted to an ontology which contains a number of words and relationships to support matching task. Thus, we try to make an auxiliary list that stores the relationship of hyponym/ hypernym words.

Before and after applying abbreviation in both schemas is listed in Table 2 and Table 3 which replaced words are underlined in After section. In Table of CIDX, "PO" and "attn" are changed to "PurchaseOrder" and "Attention" respectively, while in Table of Apertum, "Post" and "Phone" are changed to "Postal" and "Telephone" respectively.

After expanding single words to long form by using abbreviation list, two more lists which are synonym and hypernym are loaded for next step: pair generating.

3.1.2 Pair Generating

Step 2 is dedicated to build candidate list by exact matching, synonym matching, hypernym matching. Partial matching result

Table 2. Partial of CIDX schema before and after applying abbreviation list.

Before
PO
PO.Contact
PO.Contact.contactName
PO.Contact.contactFunctionCode
PO.Contact.contactEmail
PO.Contact.contactTelephone
PO.POBillTo
PO.POBillTo.attn
PO.POBillTo.street1
PO.POBillTo.street2
PO.POBillTo.city
PO.POBillTo.postalCode
PO.POShipTo
PO.POShipTo.attn
PO.POShipTo.street1
PO.POShipTo.street2
PO.POShipTo.city
PO.POShipTo.postalCode

After
PurchaseOrder
PurchaseOrder.Contact
PurchaseOrder.Contact.contactName
PurchaseOrder.Contact.contactFunctionCode
PurchaseOrder.Contact.contactEmail
PurchaseOrder.Contact.contactTelephone
PurchaseOrder.PurchaseOrderBillTo
PurchaseOrder.PurchaseOrderBillTo.Attention
PurchaseOrder.PurchaseOrderBillTo.street1
PurchaseOrder.PurchaseOrderBillTo.street2
PurchaseOrder.PurchaseOrderBillTo.city
PurchaseOrder.PurchaseOrderBillTo.postalCode
PurchaseOrder.PurchaseOrderShipTo
PurchaseOrder.PurchaseOrderShipTo.Attention
PurchaseOrder.PurchaseOrderShipTo.street1
PurchaseOrder.PurchaseOrderShipTo.street2
PurchaseOrder.PurchaseOrderShipTo.city
PurchaseOrder.PurchaseOrderShipTo.postalCode

is shown in Table 4. To improve our matching process, we create an index for each schema to store all single words and all compound words in lowercase along with their position based on inputs string.

We store compound words in the index because of separating compound words may fail, for example, electronic mail could be named by following terms: "email", "E-mail" or "EMail". The two latter words are separated to "E" and "Mail" which fail when compare to "email".

Table 3. Partial of Apertum schema before and after applying abbreviation list.

Before
Order
Order.InvoiceTo
Order.InvoiceTo.Address.Name1
Order.InvoiceTo.Address.Street
Order.InvoiceTo.Address.City
Order.InvoiceTo.Address.PostCode
Order.InvoiceTo.Contact
Order.InvoiceTo.Contact.FirstName
Order.InvoiceTo.Contact.LastName
Order.InvoiceTo.Contact.Phone
Order.DeliverTo
Order.DeliverTo.Address.Name1
Order.DeliverTo.Address.Street
Order.DeliverTo.Address.City
Order.DeliverTo.Address.PostCode
Order.DeliverTo.Contact
Order.DeliverTo.Contact.FirstName
Order.DeliverTo.Contact.LastName
Order.DeliverTo.Contact.Phone

After
Order
Order.InvoiceTo
Order.InvoiceTo.Address.Name1
Order.InvoiceTo.Address.Street
Order.InvoiceTo.Address.City
Order.InvoiceTo.Address.PostalCode
Order.InvoiceTo.Contact
Order.InvoiceTo.Contact.FirstName
Order.InvoiceTo.Contact.LastName
Order.InvoiceTo.Contact.Telephone
Order.DeliverTo
Order.DeliverTo.Address.Name1
Order.DeliverTo.Address.Street
Order.DeliverTo.Address.City
Order.DeliverTo.Address.PostalCode
Order.DeliverTo.Contact
Order.DeliverTo.Contact.FirstName
Order.DeliverTo.Contact.LastName
Order.DeliverTo.Contact.Telephone

From observation, we notice the synonym and hypernym problem from matching results on Table 4. Some example of synonym problem are "BillTo" and "InvoiceTo"; "ShipTo" and "DeliverTo." To solve such kind of synonym problem, we take single words from a source attribute and then look for their synonyms from our proposed synonym list. After that, we try to match target attribute by using both exact and synonyms of single words. The attribute(s) from target schema is/are selected by this step. We treat the target attributes which contains exact words and synonym words equally by weighting them with 1.0.

Table 4. Mapping between CIDX and Apertum.

Source: CIDX (Partial)	Target: APERTUM (Partial)
PO	Order
PO.Contact	Order.InvoiceTo.Contact
PO.Contact	Order.DeliverTo.Contact
PO.Contact	Order.Buyer.Contact
PO.Contact	Order.Supplier.Contact
PO.Contact.contactName	Order.InvoiceTo.Contact.FirstName
PO.Contact.contactName	Order.InvoiceTo.Contact.LastName
PO.Contact.contactName	Order.DeliverTo.Contact.FirstName
PO.Contact.contactName	Order.DeliverTo.Contact.LastName
PO.Contact.contactName	Order.Buyer.Contact.FirstName
PO.Contact.contactName	Order.Buyer.Contact.LastName
PO.Contact.contactName	Order.Supplier.Contact.FirstName
PO.Contact.contactName	Order.Supplier.Contact.LastName
PO.Contact.contactPhone	Order.InvoiceTo.Contact.Phone
PO.Contact.contactPhone	Order.DeliverTo.Contact.Phone
PO.Contact.contactPhone	Order.Buyer.Contact.Phone
PO.Contact.contactPhone	Order.Supplier.Contact.Phone
PO.POBillTo	Order.InvoiceTo
PO.POBillTo.attn	Order.InvoiceTo.Address.Name1
PO.POBillTo.street1	Order.InvoiceTo.Address.Street
PO.POBillTo.street2	Order.InvoiceTo.Address.Street
PO.POBillTo.city	Order.InvoiceTo.Address.City
PO.POBillTo.postalCode	Order.InvoiceTo.Address.PostCode
PO.POShipTo	Order.DeliverTo
PO.POShipTo.attn	Order.DeliverTo.Address.Name1
PO.POShipTo.street1	Order.DeliverTo.Address.Street
PO.POShipTo.street2	Order.DeliverTo.Address.Street
PO.POShipTo.city	Order.DeliverTo.Address.City
PO.POShipTo.postalCode	Order.DeliverTo.Address.PostCode

According to Table 4, we also found the hypernym problem; it is "attn" and "Name1". The word "Attention" could include title, company name, department name or just person name. Similar to "Address" which could contains house number, road name or city name. To solve this problem, we look up each attribute from source schema with hypernym list, and then

take their hypernyms to search for target attribute. We treat these attributes by weighting them with 0.8 because of an existing work claim that 0.8 is a good score for hypernym matching [5]. As be seen from Pseudo code 1. Each entry from schema S will be evaluated by taking its single words to look up their pairs in the index of schema T (line 8-11). If a single word on schema S match to single word on the index of schema T either exact or synonym (line 13- 15) matching, the score will be set to 1. But, the score will be set to 0.8 (line 17) if its match by hypernym. After every single word of the source entry is determined, then function merge at line 22 will combine redundant of target entries and also summarize wording score for each of them. At line 27, candidate pairs are prepared.

Table 5. Pair generating.

Pseudo 1: Pair Generating
List pairGenerating (schema S, schema T, synonym L1, hypernym L2)

```
1:  // Result List for candidate for each row of S
2:  List resultList List tmpPosList
3:
4:  // Store Position and Matching Score
5:  HashMap<String, Double > tmpPosList, posList
6:
7:  // each member in source schema
8:  For entry in S
9:      // leaf to root
10:     For nodeName in reverse ( entry )
11:         For SW in nodeName  // each single word
12:           // exact match
13:          tmpPosList.add (lookupIdx (T, SW, 1))
14:           // syn match
15:          tmpPosList.add (lookupIdx (T, SW, L1 ,1))
16:           //hyper match
17:          tmpPosList.add (lookupIdx (T, SW, L2, 0.8 ))
18:         Done
19:     Done
20:
21:     // Accumulate result of each candidate T
22:     posList = merge (tmpPosList)
23:
24:     // add T candidate List for each entry S
25:     For pos in posList
26:     Do
27:         resultList.add( entry + ":" + lookupEntry(pos, T));
28:     Done
29: Done
```

3.1.3 Pair scoring

Not only synonym and hypernym problem are found and taken care of on step 2, but we also notice the third problem that is the structural problem between leaf node and precedence nodes. The precedence nodes may contain clues on their name in exact words or synonym words, or even hypernym words. For example, "Bill" and "Invoice", "Ship" and "Delivery" which allow our matcher to create correspondences between source and target schema attribute precisely, for example, "PO. POBillTo. street1" and "Order.InvoiceTo.Address.Street"; "PO.PO ShipTo. street1" and "Order. DeliverTo. Address.Street."

Table 6. Scoring by Position.

Pseudo 2: Scoring by Position
Double scoringByPos (S_pos, T_pos, Curr_Pos)

```
1:  Set result = 0
2:  If Curr_Pos == 0 // Leaf Node
3:      If S_Pos == T_Pos // Both position are leaf  node
4:          result = 5
5:      Else
6:          result = 0
7:  Else  // Curr Pos is not leaf Node
8:      If S_Pos == T_Pos // Both position are same
        precedence nodes
9:          result = 1
10:     Else if ( S_Pos == T_Pos + 1)
11:         result = 0.2
12:     Else if ( S_Pos == T_Pos - 1)
13:         result = 0.2
14:     Else
15:         result = 0
16: End if
```

We solve this structural problem by proposing novel algorithm as shown in pseudo code 2 and demonstrate how our algorithm work in Fig. 5. This function takes three parameters including position of a single word on source schema, position of a single word on target schema, and current pointer of a node on source schema.

As can be seen from Pseudo code 2, the similarities of attribute and its candidate are evaluated, starting from the leaf node to its root node. If a single word from source schema matches to a single word on target schema at leaf node level, they will get 5 marks each (line 3-4). But matching on same

precedence level, such as parent to parent or grandparent to grandparent node, the score turn out to be 1 (line 8-9); and if they match to different level the score will take on only 0.2 (line 10-13). We assign the score 5, 1 and 0.2 because the normal compound words usually contain less than 5 single words [25]. We do not want score from precedence nodes to exceed a score on leaf nodes because, from our experiment, score on leaf nodes more important than their precedence nodes.

For example, if there is a pairs contains "PurchaseOrder.Header.orderDate" and "Order.OrderHeader.OrderNo", then the each single word of the first entry will be evaluated from the leaf node to root node. So, "date", "Order", "Header", "Order" and "Purchase" will be evaluated sequentially. There is no "Date" and "Purchase" in the second entry so score for them are zero. For "Order" at leaf node will get 5 marks because there is the same word on leaf node of the second entry. Then, "Header" is evaluated and gets 1 mark because this single word locates at parent node of the second entry. However, "Order" that come along with "Purchase" get 1.2 mark because there are two "Order" available on both non-leaf node. One is on the same level while another is on the different level. In total, this pair gets 5 + 1 + 1.2 = 7.2 marks.

After scoring, each attribute from source schema will get their candidates list taken from target schema. The top candidates will be promoted to the second stage list. Actually, our matcher also supports top-N policy, but we found that the top-1 is the most effective policy to match purchase order schemas on our experiment because most of the correct candidates are chosen without many of wrong candidates. For example, "PO.Contact.contactPhone" on Table 7 contains all correct candidates at the 1st rank and all wrong candidates are found at the 2nd rank.

3.1.4 Pair selection

The last step for non-fuzzy process is step 3.1 in Fig. 4. At this step, we enhance the matching result by sorting the target attribute and their source candidates with score obtain from step 3, and then pick up the top candidates for each target attribute. For example, "Order.DeliverTo.Contact" has 3 candidates those are "PO.Contact", "PO.Contact.contactFunctionCode" and "PO.Contact.contactEmail" with wording score 5.2, 6.0, and 6.0 respectively. So, the final candidates without using fuzzy logic are "PO.Contact.contactFunctionCode" and "PO.Contact.contactEmail" which are wrong. Actually, only "PO.Contact" is the correct answer. Limitation of word scoring is it discovers only candidate which contain the same word or its synonym of source attribute but word scoring does not give a penalty for extra single words. So, we need to measure several properties of each pair before select final result. Thus, we propose to enhance our matcher by employing fuzzy logic after word scoring step.

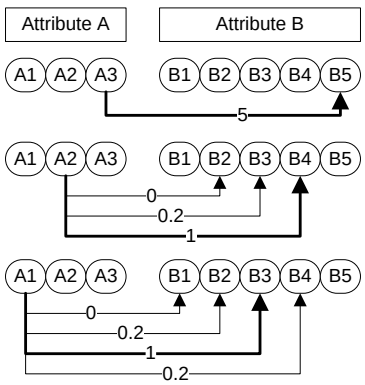

Fig. 5. Demonstration of our scoring by position algorithm.

Table 7. Result of top-2 policy

Source: CIDX	Target: APERTUM	Rank	Correct (Y/N)
PO.Contact.contactPhone	Order.DeliverTo.Contact.Phone	1	Y
PO.Contact.contactPhone	Order.Supplier.Contact.Phone	1	Y
PO.Contact.contactPhone	Order.InvoiceTo.Contact.Phone	1	Y
PO.Contact.contactPhone	Order.Buyer.Contact.Phone	1	Y
PO.Contact.contactPhone	Order.InvoiceTo.Contact	2	N

Source: CIDX	Target: APERTUM	Rank	Correct (Y/N)
PO.Contact. contactPhone	Order.DeliverTo. Contact.Phone	1	Y
PO.Contact. contactPhone	Order.Buyer. Contact	2	N
PO.Contact. contactPhone	Order.Supplier. Contact	2	N
PO.Contact. contactPhone	Order.DeliverTo. Contact	2	N

3. 2 Process of our matcher with Fuzzy Logic

3.2.1 Fuzzy Scoring and Fuzzy Filtering

In step 4- 5, after we solve three problems of synonym, hypernym and structure, candidates for each source attribute are selected. However, the selection contains many wrong candidates. Thus, we design fuzzy logic system to filter wrong candidates by a fixed threshold.

The range of our fuzzy score is in real number between 0 and 1. If a value is 0 means very confident for a correct candidate while a value is 1 means not confident at all. As be seen in Fig. 6, the cut- off threshold is set to 0. 65 because we found that candidate pairs with the fuzzy score over 0. 65 are wrong candidates according to our observation in our experiment.

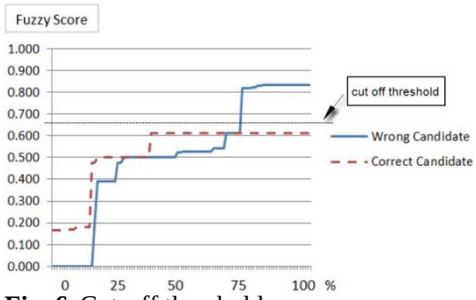

Fig. 6. Cut-off threshold

3.2.2 Pair Selection

In step 6, although fuzzy logic alone could discover and reject partial of wrong candidates, but we can enhance more by sorting by target attribute and fuzzy score, then pick up the top one for each target attribute. This sorting process has two benefits by removing some false candidates and also preserving 1: m relationship. For example, there are three candidates for "Order. DeliverTo. Contact" including "PO. Contact.contactEmail", "PO.Contact.contact FunctionCode" and "PO.Contact" which get fuzzy score 0.50, 0.61 and 0.17 respectively. Our matcher takes the lowest fuzzy score so " PO. Contact" is selected. In addition, "PO.POShipTo.street1" and "PO.POShipTo. street2" get the same fuzzy score is 0. 5 so both of them are preserved for "Order. DeliverTo. Address. Street" as 1: m relationship.

3.3 Detail design of our Fuzzy Logic

After we validate the result of linguistic matching, there are several wrong candidates occur for each entry of the source schema, so we examine the result and come up with three more properties to remove those wrong candidates. They are different by ranking, mismatch of single word and difference position of leaf node. In addition, we need only a value for the similarity of each candidate. Thus, we decided to create a fuzzy system with 3 inputs, one input per a membership function, and an output to get the value of similarity for each pairs.

As can be seen from Fig. 7- 9, Negative Large (NL) , Negative Medium (NM) , Negative Small (NS) , Zero (ZR) , Positive Small (PS) , Positive Medium (PM) and Positive Large (PL) are common terms to define fuzzy input but they are not restricted. And, their values do not actually refer to negative or positive number. They just indicate position far away from the middle of input range. The leftmost is NL, middle is ZR and rightmost is PL. One can define grade by low, medium and high, or poor, good and excellent for better understanding. How to calculate those three properties in detail are explained as following paragraph.

The first property is the ranking after linguistic matching. From observation, an attribute of a source schema could match to other attributes of a target schema which

does not have the highest wording score; the second highest score would actually be the correct one. For example, "PO. Contact. contactPhone" seems more similar to "PO. Header. Contact" than "PO. DeliverTo. Contact. telephone." However, in this case, only "PO. DeliverTo. Contact. telephone" is the correct answer. Thus, we set ranking difference (rnk) as one of our three properties to measure candidate characteristic. For fuzzy membership function, we employ trapezoid and triangle shape as depict in Fig. 7 allowing attributes at the 1st and 2nd rank to produce the same output so that we can use other properties to evaluate the similarity of candidate pairs further. But, candidates beyond 3rd rank can hardly be correct, so we let this membership function suppress candidates lower than 2rd rank.

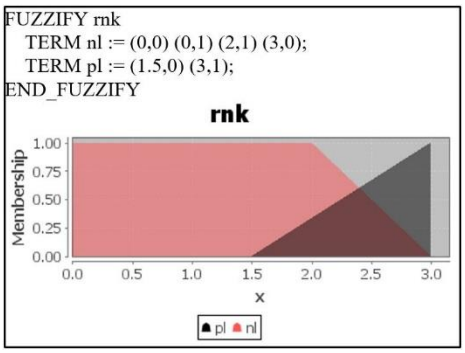

Fig. 7. Membership function of ranking**.**

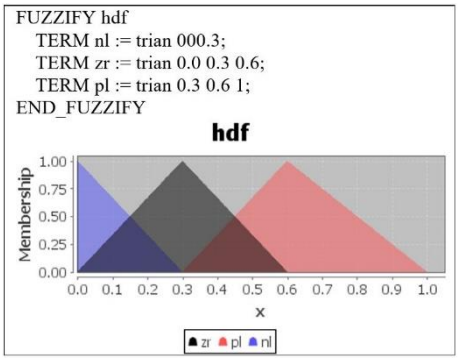

Fig. 8. Membership function of different on hierarchy.

The second property is hierarchy difference (hdf). After we investigate further, we found that if the depth of path from root node to leaf node is about the same, a pair is most likely to be matched. We propose Eq. (3.1) to get the second property to determine relative difference between two attributes on source and target schema. If the value of hdf is 0, it means both attributes are placed at the same level. The value of hdf is calculated by taking absolute value of difference between "a" and "b", then divided by max value between them. Whereas "a" is the depth level of a candidate from a source schema and "b" is the depth level of a target schema. The membership function of hdf is shown by FCL and visual graph in Fig. 8.

$$hdf = \frac{abs(a-b)}{\max(a,b)} \qquad (3.1)$$

The third property is word difference (wdf) to detect the number of unmatched words such as "FirstName" and "Name" is similar because matching on word "Name." However, "contactFunctionCode" and "Contact" are dissimilar because there are more than one word added including "Function" and "Code." We propose Eq. (3.2) to measure this property because most wrong candidates have many unmatched single words between source attributes. If the value of wdf is 0, it means both attributes are identical. The value of wdf is the proportion between single words that remain unchanged and the total number of single words of the attribute and each of its candidates.

$$wdf = \frac{(w_1 + w_2) - (2 * match(w_1, w_2))}{(w_1 + w_2)} \qquad (3.2)$$

The w_1 and w_2 are the number of single words on a leaf node in the source and target schema respectively. The matching function in (3.2) will return number of matched pairs of

single words. The membership function of wdf is shown by FCL and visual graph in Fig. 9.

After all membership functions are defined, the next thing to do is to design fuzzy rules. Total rules equal to the product of all classes across all membership functions, which in our case is 2×3×3 = 18. We can split the rules into Table 8 and Table 9, so they are much easier to define. Fuzzy rules for high ranking are provided in Table 8 while Table 9 contains fuzzy rules for low ranking. Although both tables seem similar, Table 9 contains more PL value than table 8, because pairs with low ranking are more likely to be wrong candidates than pairs with high ranking.

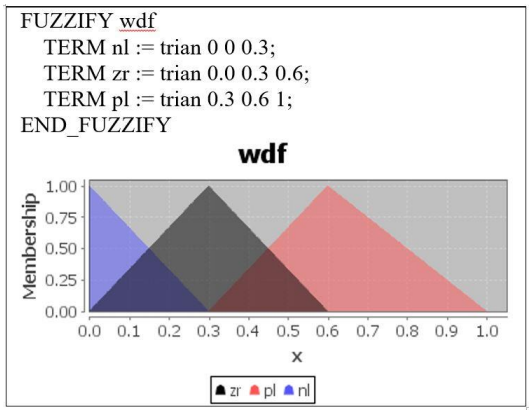

Fig. 9. Membership function of different on word.

Table 8. Fuzzy rule for high ranking.

Word Diff (WDF)	Hierarchy Difference (HDF)		
	NL	***ZR***	***PL***
NL	NL	NL	ZR
ZR	ZR	ZR	PL
PL	PL	PL	PL

Table 9. Fuzzy rule for low ranking

Word Diff (WDF)	Hierarchy Difference (HDF)		
	NL	***ZR***	***PL***
NL	NL	NL	PL
ZR	ZR	PL	PL
PL	PL	PL	PL

After tuning, we found that these fuzzy rules are adequate to reject wrong candidates. All of fuzzy rule for both tables are combined into Fig. 10 in FCL format, then we apply them to measure similarity for other schema pairs.

Each cell in Table 8 and Table 9; NL, ZR and PL means low error, medium error and high error respectively.

Normally, fuzzy rules are designed by one's experience or by examining the result after conducting several experiments. A good outcome would require several attempts but tuning is much easier than adjusting several thresholds. That is why fuzzy logic is widely adopted from home appliances to aerospace applications.

The last step in our fuzzy system is defuzzification which converts three fuzzy outputs to a crisp output. We create a simple defuzzifier function for those three inputs and also calculate centroid of output by center of gravity (COG) method. FCL and visual of our defuzzification is depicted in Fig. 11. sample values are listed on Table 10. Although COG method needs much more computation than Max Value method, COG produces much precious output. Some home appliance would define a table like Table 10 in its memory to avoid complexity of calculation.

```
RULE 1 : IF rnk IS nl AND hdf IS nl AND wdf is nl THEN out1 IS nl;
RULE 2 : IF rnk IS nl AND hdf IS nl AND wdf is zr THEN out1 IS zr;
RULE 3 : IF rnk IS nl AND hdf IS nl AND wdf is pl THEN out1 IS pl;
RULE 4 : IF rnk IS nl AND hdf IS zr AND wdf is nl THEN out1 IS nl;
RULE 5 : IF rnk IS nl AND hdf IS zr AND wdf is zr THEN out1 IS zr;
RULE 6 : IF rnk IS nl AND hdf IS zr AND wdf is pl THEN out1 IS pl;
RULE 7 : IF rnk IS nl AND hdf IS pl AND wdf is nl THEN out1 IS zr;
RULE 8 : IF rnk IS nl AND hdf IS pl AND wdf is zr THEN out1 IS pl;
```

```
RULE 9 : IF rnk IS nl AND hdf IS pl AND wdf is pl THEN
out1 IS pl;
RULE 10 : IF rnk IS pl AND hdf IS nl AND wdf is nl
THEN out1 IS nl;
RULE 11 : IF rnk IS pl AND hdf IS nl AND wdf is zr
THEN out1 IS zr;
RULE 12 : IF rnk IS pl AND hdf IS nl AND wdf is pl
THEN out1 IS pl;
RULE 13 : IF rnk IS pl AND hdf IS zr AND wdf is nl
THEN out1 IS zr;
RULE 14 : IF rnk IS pl AND hdf IS zr AND wdf is zr
THEN out1 IS pl;
RULE 15 : IF rnk IS pl AND hdf IS zr AND wdf is pl
THEN out1 IS pl;
RULE 16 : IF rnk IS pl AND hdf IS pl AND wdf is nl
THEN out1 IS pl;
RULE 17 : IF rnk IS pl AND hdf IS pl AND wdf is zr
THEN out1 IS pl;
RULE 18 : IF rnk IS pl AND hdf IS pl AND wdf is pl
THEN out1 IS pl;
```

Fig. 10. Fuzzy rule in FCL format.

```
DEFUZZIFY out1
   TERM nl := trian 0 0 0.5;
   TERM zr := trian 0 0.5 1;
   TERM pl := trian 0.5 1 1;
   // Use 'Center Of Gravity' defuzzification
method
   METHOD : COG;
   // Default value is 0 (if no rule activates
defuzzifier)
   DEFAULT := 0;
END_DEFUZZIFY
```

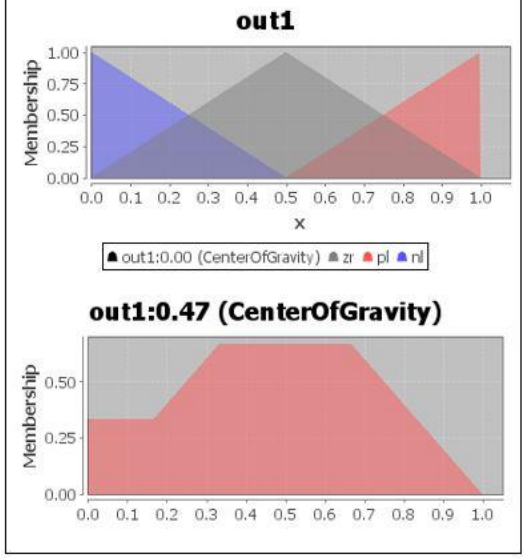

Fig. 11. Membership function of defuzzication method. (with inputs rnk = 1, hdf = 0.2, wdf = 0.2)

Table 10. Sample output from fuzzy logic.

Output	Input		
	RNK	***HDF***	***WDF***
0.16	1	0.0	0.0
0.18	1	0.2	0.0
0.47	1	0.0	0.2
0.47	1	0.2	0.2
0.52	1	0.2	0.4
0.49	1	0.4	0.2
0.49	2	0.2	0.2
0.52	2	0.4	0.4
0.82	3	0.4	0.4

Table 11. Quality of our matcher.

Test Set	Quality		
	Recall	***Precision***	***F-measure***
Conf1 (attn:name)	1.000 (29/29)	0.707 (29/41)	0.829
Conf1 + Fuzzy (attn:name)	0.862 (25/29)	0.862 (25/29)	0.862
Conf2 (-:name)	0.931 (27/29)	0.771 (27/35)	0.844
Conf2 + Fuzzy (-:name)	0.931 (27/29)	1.000 (27/27)	0.964

3.4 Bootstrap training

Table 11 is the result of our matcher for finding correspondence of the source schema in Table II to target schema on Table III and then evaluated with answer given in Table IV. There are four configurations to execute our matcher including conf1, conf1+fuzzy, conf2, conf2+fuzzy where conf1 and conf2 means configuration set 1 and configuration set2 without fuzzy logic whereas conf1+ fuzzy and conf2+ fuzzy means configuration set 1 and configuration set 2 with fuzzy logic. The difference of conf1 and conf2 is just the hypernym list. For conf1, the word "Attn" is extended to "attention" and also assigned to "name" as its hypernym because this word does not refer only to personal name but usually including title and company name. In contrast, conf2 "Attn: name" is removed.

Let us take a look at the results in Table 11, with the goal of finding 29 pairs of

attributes between the source and the target schema. The result of conf1, 41 pairs is suggested from our matcher. Although it shows outstanding recall because all answer is discovered, precision is only 0.707 because there are 12 wrong candidates selected, so the f-measure is just about 80%. After enabling fuzzy logic on conf1, number of suggested pairs is reduced to 25 so recall slightly drops to 0.862, but precision is increased up to 0.862. Thus, f-measure slightly increases by 4% to 0.864.

Next, the result of conf2 is processed by removing "Attn: name" from hypernym list because we found that the word "name" is a very common that it causes several extra candidates to be added such as "Contact. FirstName" and "Contact. Last-Name." However, the right candidate is only "Address. Name1". After removing "Attn", suggested candidates drop from 41 to 35 pairs, so precision slightly increase from 0.707 to 0.771. Although the recall is not perfect this time because two correct pairs are missing from selection, "Attn" is matched to nowhere. This problem of conflict-hypernym could be solved by removing exact matched or synonym matched candidates, so that the remaining candidates will be selected as hypernym candidates. The result of conf2 after enabling fuzzy logic, the recall is unchanged, but precision reached 1, meaning all 27 suggested pairs are correct. Thus, it increased precision and boost f-measure up to 0.964.

4. Results and Discussion

Four parts are provided in this section including where the schemas we use for experiments, how we conduct the experiment, the result, and discussion.

4.1 Data Collection and Preparation

To ensure that our matcher can solve real world schema, we take five schemas of e-commerce from COMA 3.0 [14] which is the result of a study from the COMA research work [5]. The format of the data is in XDR format. However, XDR format is unpopular, thus we have to convert it into XSD format. The data contains a total of five schemas: Apertum, CIDX, Excel, Noris and Paragon. The characteristic of such data is provided in Table 12. The average depth level is 5. CIDX is the shallowest while Paragon is the deepest. Meanwhile, the average number of leaf node is about 80. The largest schema that contains 148 nodes is Apertum while CIDX is the smallest schema, which contains only 41 nodes. We will use 5 schemas to evaluate the quality of our matcher.

Table 12. Characteristic of five schemas.

Charac-teristic	Test Schema				
	1 Apertum	***2 CIDX***	***3 Excel***	***4 Noris***	***5 Paragon***
Max Depth	5	4	5	5	7
#Leaf Node	148	41	55	66	81

Not only the COMA provides sample schemas, it also provides partial answer in RDF format as well. Thus we use the answers to compare against our experiment result. Furthermore, we also compare our works with an existing work, openII (an open-source software) and Levenstein (an edit distance method) as the baseline.

OpenII is open-source contributed by MITRE Corporation for matching task in the large enterprise [23-24]. Not only the tool offers several matchers such as name similarity, documentation, mapping, thesaurus, extract, quick and wordNet matcher, but it also offers user-friendly GUI for users to review and adjust the result.

4.2 Evaluation Metrics

The matching quality is measured by three scores which are recall, precision and f-measure. Both recall and precision are independent quantity, whereas, f-measure combines precision and recall into a single quantity. While some papers [3, 13] measure

quality by overall properties instead of f-measure which allows value to be minus if a matcher performs bad result, F- measure is more common particularly in Ontology Alignment Evaluation Initiative (OAEI) competition since it was introduced in 2004. So, we select f- measure to represent dependent quality because of this reason. Recall measures the ability to discover correct pairs while precision measures the proportion of correct pairs against all suggested pairs. How to calculate precision, recall and f-measure is given in Eq. (4.1) – Eq. (4.3) , where A is the number of correct matches that a matcher did not discover, B is the number of correct matches discovered and C is the number of false matches that a matcher suggests wrong pairs.

$$Recall = \frac{B}{(A+B)} \quad (4.1)$$

$$Precision = \frac{B}{(B+C)} \quad (4.2)$$

$$F_{measure} = 2 \times \frac{(Precision \times Recall)}{(Precision+Recall)} \quad (4.3)$$

4.3 Experiment

For easy to reference, Apertum, CIDX, Excel, Noris and Paragon are denoted by numbers 1, 2, 3, 4 and 5 respectively. Seven test- cases between pairs of schema are denoted by representing the schema pairs as {2,1}, {2,3}, {2,4}, {2,5}, {3,1}, {3,4}, {3,5} in Table 13 and in Fig 11.

We start the experiment with an edit distance method that works well for finding similarity in the result of string matric comparison [8]. One of the most effective methods is Levenstein which produces a good result on record of a census, including first name, family name and address. We found that all three quality measures are very low since Levenstein method does not consider abbreviation. We tried to modify the method by adding abbreviation and shorten form by predefined abbreviation list, for example po: PurchaseOrder, ord: Order,post: Postal, no: Number, mod: Model, col: Color, uom:UnitOfMeasure etc. Then, we measured the similarity between full path of two attributes, one from source and one form target, and found that all the quality measures are still very low because a lot of different prefixes. Then, after we score full path plus parent node, the results on some datasets were improved as shown in the first section of Table 13, where the recall, precision and f-measure are about 20-50%. We can notice that set 4 {CIDX, Paragon}, set 5 {Excel, Apertum} and set 6 {Excel, Noris} are quite similar.

The second section in Table 13 is the result from OpenII. Surprisingly, the result from Levenstein is still better than this tool as shown in Table 13 because openII produces too much candidate which relate only parent and leaf node. In fact, all precedence is important for schema matching.

Table 13. Matching results.

Data Set		2,1	2,3	2,4	2,5	3,1	3,4	3,5
Levenstein	R	0.13	0.17	0.25	0.53	0.42	0.72	0.25
	P	0.17	0.27	0.20	0.63	0.60	0.65	0.27
	F	0.15	0.21	0.22	0.58	0.49	0.69	0.26
openII	R	0.25	0.34	0.19	0.28	0.21	0.44	0.20
	P	0.28	0.46	0.47	0.45	0.32	0.48	0.28
	F	0.26	0.39	0.27	0.35	0.25	0.46	0.23
NoFuzzy	R	0.91	0.85	0.94	0.88	0.75	0.82	0.90
	P	0.77	0.75	0.79	0.70	0.67	0.65	0.65
	F	0.83	0.80	0.86	0.78	0.71	0.73	0.76
WithFuzzy	R	0.89	0.85	0.91	0.88	0.75	0.80	0.90
	P	0.91	0.92	0.85	0.84	0.73	0.78	0.77
	F	0.90	0.88	0.88	0.86	0.74	0.79	0.83

(R = Recall, P = Precision, F= F-measure)

In the third section in Table 13, seven sets are executed without fuzzy logic show that the result of this method is quite high

because the synonym list was used to account for word similarity, such as: {tile: position}, {ship: shipment: deliver: delivery}, {item: product: model: line}, {bill: invoice}, {number:id:code} etc. However, the average precision is 20% less because of some wrong candidates.

The forth section in Table 13, seven sets are re- executed with enabling fuzzy logic, wrong candidates are taken out so precision and recall are both increased. As be seen in Fig. 12, f-measure improved in every case. However, recall slightly drop on three test sets which are 1 (CIDX, Apertum), 3 (CIDX, Noris) and 6 (Excel, Noris).

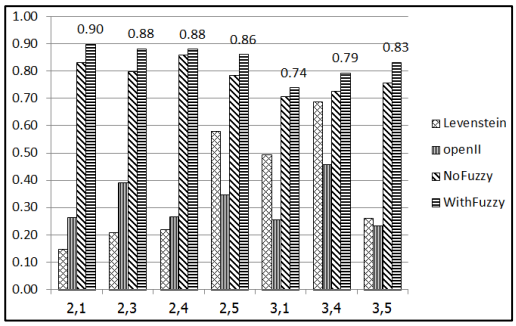

Fig. 12. Comparison of matching results.

5. Discussion

In this part, we will discuss on two cases why our matcher cannot discover some correct candidates and why some of correct candidates are rejected by fuzzy logic.

There are four reasons on the first case. The first reason is the existence of compound noun to define an entity name, for example, in data set1, which is the matching between CIDX and Apertum. The entity name " PoNumber" have to match to both "SupplierOrderReference" and "BuyerOrder Reference." Our matcher does not discover that because if we add "Reference" to this synonym {number: id: code}, it brings up many false candidates because number, id and code are commonly used for many entities. To address this problem in the future, our matcher has to implement a method to match simple word and compound word so that " number" can match to " OrderReference" correctly. The pairs of "count" and "NumberOfLines" also fail due to this reason.

The second reason is too many different words on two entities such as " totalValue" and " VAT_AmountTotal. " Even human cannot pair these candidates at the first glance, instances of data and description would be helped in this case. Besides, "UnitPrice:NetPerchPrice" also fall in this case.

The third reason is inadequate information on schema like " Header. Contact.companyName" of a source schema can be matched to 8 target attributes, which are "InvoiceTo. Address. Name1", "Invoice To. Address. Name2", "SupplierTo. Address. Name1" , " SupplierTo. Address. Name2" , " DeliveryTo. Address. Name1" , " Delivery To.Address.Name2", "Buyer.Address.Name 1", and "Buyer. Address. Name2". There is only a clue on simple word "Name" which requires other information like instance, description or human judgment to confirm of correct correspondences.

The last reason is top-1 policy of our word scoring which could make a correct candidate stay at the 2nd or 3rd rank if there is a more similar entity name on a different path available. Although our matcher can determine correct candidates from different rank, we found that top- 1 policy still produces the result at an acceptable level for both precision and recall.

Next, let continue to discuss on the second case. Actually, our fuzzy logic has rejected only a single pair for each dataset 1, 2 and 6.

On data set1 {CIDX, APERTUM}, "Order.POLine.LineNo" has to match one of these three candidates including " PO. POLines. Item. partNo", "PO. POLines. Item. line" and "PO. POLines. Item"; with fuzzy score is 0.388, 0.503 and 0.503 respectively. After step 6 in Fig. 4, sorting by target row and fuzzy score along with top-1 policy will remove both attributes with score 0. 503

which contains the right candidate. " PO. POLines. Item. line" is the correct answer.

On data set 3 {CIDX, Noris} also fail with the same reason as data set 1. Two of source attributes " PO. POLines. Item. line" and "PO.POLines.Item" are promoted to the semi-final list with fuzzy score 0.425 and 0.170 respectively. So, "PO.POLines.Item" is accepted for "PurchaseOrder.Line.lineNo" because of depth of "PO.POLines.Item" is equal to the answer while " PO. POLines. Item" is one level deeper. For human, we can keep those of them for further validation because both of the candidates are very similar.

On test set 6 {Excel, Noris}, there are seven candidates in the semi-final list for "PurchaseOrder.Items.Item.partDescription." However, "PurchaseOrder.Items.Item" is the most similar, even though the correct candidate is " PurchaseOrder. Line. product Name" so the former is accepted and the latter is rejected. In this case, it depends on users for choosing the most appropriate entry. Further validation on instance would be required.

6. Conclusion

Our schema matcher comprises two main parts; word scoring and fuzzy logic by defining fuzzier, inference, fuzzy rule and defuzzifier in the FCL. We start by analyzing schema structure, expanding abbreviation, indexing, scoring, applying synonym and hypernym to reach an acceptable level of recall, then we tune our matcher further by employing fuzzy logic to remove wrong candidates with a cut-off threshold.

From our experiments, after enabling fuzzy logic, the result of our schema matcher shows that the quality of precision is increased by 10% on every test sets. Meanwhile, the f-measure is up to 90% on maximum and 80% on average.

The major benefit of implementing fuzzy logic for schema matching is the ability to adjust rules and grading inputs by membership functions. Adjusting rules are easier to understand than changing several thresholds and parameters. However, the fuzzy system will be complicated if there are many inputs and many classes for each membership function because many rules may be required. Thus, designing fuzzy logic system, selecting inputs and developing membership function for a fuzzy system is crucial.

There are several things that can be done enhance our matcher in the future such as compound word matching, conditional rules, and implementing a GUI to support users to select candidates manually. Compound word matching is an attempt to increase the quality of recall in case where a simple word has correspondences to compound word such as "total:numberOf" or "id:ReferenceNo." Next, conditional rule is the way to match candidates where leaf node would be exactly the same, but they are on different topics. For example, first name and last name are related to people, not company or organization. Lastly, GUI is necessary for users to be able to explore options so that correct candidates can be examined with ease.

Acknowledgements

The first author would like to express his gratitude to Advance Info Services (AIS) for the scholarship and also for supporting their employees to further their education.

References

[1] Doan AH, Alon Y. Halevy. Semantic integration research in the database community: A brief survey. AI Mag 2005;26(1):83.

[2] Halevy A. Why your data won't mix. Queue 2005;3(8):50-8.

[3] Rahm E, Philip A. Bernstein. A survey of approaches to automatic schema matching. VLDB Journal 2001; 10(4):334-50.

[4] Bernstein PA, Jayant M, Erhard R. Generic schema matching, ten years later.

Proceedings of the VLDB Endowment 2011;4(11):695-701.

[5] Do HH, Erhard R. COMA: a system for flexible combination of schema matching approaches. In: VLDB Endowment. Proceedings of the 28th international conference on Very Large Data Bases; 2002.

[6] Sorrentino S, et al. Schema label normalization for improving schema matching. Data Knowl Eng 2010;69(12): 1254-73.

[7] Manning CD, Prabhakar R, Hinrich S. An Introduction to Information Retrieval. Cambridge University Press; 2009.

[8] Cohen W, Pradeep R, Stephen F. A comparison of string metrics for matching names and records. Kdd workshop on data cleaning and object consolidation. 3rd ed. 2003.

[9] Madhavan J, et al. Corpus-based schema matching. In: Data Engineering 21st International Conference on IEEE. Proceedings of ICDE; 2005. p. 57-68.

[10] Qian L, Michael JC, Jagadish HV. Sample- driven schema mapping. In: ACM. Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data; 2012. p. 73-84.

[11] Bryan LA, Eric AB. Programmable Controllers: Theory and Implementation. Industrial Text Company. 2nd ed. 1997.

[12] Zimmermann H. Fuzzy set theory-And its applications. 4th ed. 2001.

[13] Cingolani P, Jesús AF. FuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. Int J Comput Int Sys 2013;6tems 6.sup1 2013: 61-75.

[14] Database Group Leipzig. COMA 3. 0 [Internet]. Leipzig: Universität Leipzig; 2016 [cited 2016 Dec 13]. Available from: http: / / dbs. uni- leipzig. de/ de/ Research/ coma.html.

[15] Drumm C, et al. Quickmig: automatic schema matching for data migration projects. In: Conference on information and knowledge management. Proceedings of the 16th ACM conference; 2007 p. 107-16.

[16] Roussey C, et al. An introduction to ontologies and ontology engineering. Ontologies in Urban Development Projects. London: Springer; 2011. p. 9-38.

[17] Shvaiko P, Jérôme E. Ontology matching: state of the art and future challenges. IEEE Transactions on knowledge and data engineering. New York: IEEE; 2013. p. 158-76.

[18] Mukkala L, et al. Current State of Ontology Matching. In: A Survey of Ontology and Schema Matching. University of Turku Technical Reports. 4th ed. Turku: Turun yliopisto; 2015.

[19] Madhavan JPA, Bernstein R, Erhard R. Generic schema matching with cupid. Proceedings of the 27th International Conference on Very Large Data Bases; 2001 Sep 11-14; Roma, Italy. 2001. p. 49-58.

[20] Melnik S, Hector GM, Erhard R. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: Data Engineering 2002. Proceedings of the 18th International Conference on IEEE; 2002 p. 117-28.

[21] Giunchiglia F, Pavel S, Mikalai Yatskevich. S-Match: an algorithm and an implementation of semantic matching. Proceedings of the European semantic web symposium; Heidelberg, Germany: Springer Berlin; 2004. p. 61-75.

[22] Aumueller D, et al. Schema and ontology matching with COMA++. Proceedings of the 2005 ACM SIGMOD international conference on Management of data. 2005. p. 906-8.

[23] Smith K, et al. The role of schema matching in large enterprises [Internet]. 2009 [cited 2016 Dec 13]. Available from: https://arxiv.org/abs/0909.1771.

[24] Seligman L, et al. Openii: an open source information integration toolkit. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data; 2010 Jun 6-11; IN, USA. New York: ACM; 2010.

[25] Juhasz BJ. The processing of compound words in English: Effects of word length on eye movements during reading. Lang Cognitive Proc 2008;23(7-8):1057-88.