Computer Science and Information Systems 15(2):295–320 https://doi.org/10.2298/CSIS170525002M

Exploring Instances for Matching Heterogeneous Database Schemas Utilizing Google Similarity and Regular Expression

Osama A. Mehdi[1], Hamidah Ibrahim[2],
Lilly Suriani Affendey[2], Eric Pardede[1] and Jinli Cao[1]

[1] La Trobe University, Computer Science and Information Technology, Bundoora, Victoria 3086, Melbourne, Australia
{O.mahdi, E.Pardede, J.Cao}@latrobe.edu.au
[2] University Putra Malaysia, Computer Science and Information Technology, Jalan Upm, 43400 Serdang, Selangor, Malaysia
{hamidah.ibrahim, lilly}@ upm.edu.my

Abstract. Instance based schema matching aims to identify correspondences between different schema attributes. Several approaches have been proposed to discover these correspondences in which instances including those with numeric values are treated as strings. This prevents discovering common patterns or performing statistical computation between numeric instances. Consequently, this causes unidentified matches for numeric instances which further effect the results. In this paper, we propose an approach for addressing the problem of finding matches between schemas of semantically and syntactically related attributes. Since we only fully exploit the instances of the schemas, we rely on strategies that combine the strength of Google as a web semantic and regular expression as pattern recognition. To demonstrate the accuracy of our approach, we have conducted an experimental evaluation using real world datasets. The results show that our approach is able to find 1-1 matches with high accuracy in the range of 93% - 99%. Furthermore, our proposed approach outperformed the previous approaches using a sample of instances.

Keywords: schema matching, instance based schema matching, Google similarity, regular expression.

1. Introduction

One of the vital tasks in database integration is schema matching. Schema matching is the task of identifying correspondences between schema attributes. Matching two schemas *S* and *T* requires deciding if two attributes *s* of *S* and *t* of *T* represent the same real-world concept. While humans may be able to easily discover if two attributes match or non-match, however it is difficult for machines to discover it, especially when these two attributes have semantic heterogeneity. For example, *s* and *t* can represent different concepts but have the same name. The opposite is also possible; *s* and *t* can represent the same concept but have different names. To solve the problem of finding the correspondences between schemas, available information could help to identify the semantics of schema elements and to detect their similarity. Three types of available information commonly used to determine the correspondences of schema matching are schema information, instances, and auxiliary information [1][2].

– Schema information: Various kinds of information, such as element name, description, data type, constraint, and schema structure, can be examined to characterize and compare the semantics of schema elements [3].
– Instances: In many applications, such as data integration and transformation, instances are available for the schemas to be matched and can also be exploited to characterize the contents and semantics of schema elements [4][5][6][7].
– Auxiliary information: This category comprises resources used to obtain information that can be utilized to detect similarities between schema elements. For example, utilizing dictionaries and thesauri such as WordNet, enables a search for semantic relationships like synonymy and hypernymy between element names [8].

During the process of schema matching, schema information which includes element name, description, data type, constraint, and schema structure are normally used by previous works in an attempt to achieve correct matching between schemas or even when the source and the target schema are nested relational, as in [9]. However, in some real world cases it may not be possible to use the information of schema structure. There are cases where information about the schema structure is not available such as in, fraud detection, crime investigation, counter-terrorism and homeland security [10][11]. In such scenarios, instances are the only option available that can be used for schema matching. Even though, schema information might be available however there are cases where it is worthless to be used for matching purpose. An example is when the schema attributes are abbreviations. For instance, the attribute name *CN* could be an abbreviation of Customer Name or Company Name while *SSN* is an abbreviation of Social Security Number. Hence, data instances can give an accurate characterization of the actual contents of schema attributes. Several approaches that utilized instances during the process of schema matching have been proposed [4-7][12-23]. These approaches focused on one main objective which is improving the accuracy of instance based schema matching in terms of precision (*P*), recall (*R*), and F-measure (*F*).

By analysing the instance based schema matching approaches, we observed that neural network, machine learning, theoretic information discrepancy and rule based have been utilized by these approaches [4][16][18][20][21][23]. The goal of these approaches is to discover correspondences between schema attributes whereby instances including instances with numeric values are treated as strings [10]. This prevents discovering common patterns or performing statistical computation between numeric instances. As a consequence, this causes unidentified matches for numeric instances and further reduces the quality of match results. Thus, for instance level approaches, an approach for identifying existing instance patterns must be deployed.

2. Related Work

Instance based schema matching examines instances to determine corresponding schema attributes. It represents a substitutional choice for schema matching. Even when substantial schema information is available, considering instances can complement schema based approaches with additional insights on the semantics and contents of schema attributes and can be beneficial in uncovering wrong interpretation of schema information, i.e. it would be helpful to disambiguate between schema level matches by matching the attributes whose instances are syntactically and semantically more similar. Neural network, machine learning, information theoretic discrepancy and rule based are approaches used for instance based schema matching.

Neural network is able to obtain the similarities among data directly from their instances and empirically infer solutions from data in the absence of prior knowledge for regularities. Neural network is employed to cluster similar attributes, whose instances are uniformly characterized using a feature vector of constraint based criteria. For instance based schema matching, the Back Propagation Neural Network (BPNN), which can acquire and store a mass of mappings between input and output, is ideal. However, neural network can be viewed as specific tool since it is trained based on domain-specific training data. It can only be used to resolve problems associated with that domain. Instance based schema matching approaches based on neural network [12] [13][17][19] achieved precision (*P*), recall (R), and F-measure (*F*) in the range of 65% - 96%.

Solutions that are based on machine learning generally employ methods such as Naïve Bayesian classification to enhance the accuracy of schema based matching. Learning-based solutions require a training data set of correct matches that may require a large training data set to determine the correct matches. Several approaches have been proposed [5][14-15] that employ machine learning techniques to first learn the instance, characteristics of the matching or non-matching attributes and then use them to determine if a new attribute has instances with similar characteristics or not. The precision (*P*), recall (*R*), and F-measure (*F*) achieved by these approaches are in the range of 66% - 92%.

Many approaches have applied the notion of information theoretic discrepancy such as mutual information and distribution values [4][16][20][21]. The main advantages of applying an information theoretic discrepancy approach are that its skillfulness and lack of constraints. However, approaches of information theoretic discrepancy need some probabilities of overlapping in the values being compared. Instance based schema matching based on information theoretic discrepancy achieved precision (*P*), recall (*R*), and F-measure (*F*) in the range of 45% - 92%.

Rule based approaches enjoy many benefits. The first benefit of using rule based would be, low cost and also no requirement for training as in learning-based techniques. The second benefit, its quick and concise method to capture valuable user knowledge about the domain. Instance based schema matching based on rules [13][18][24] achieved precision (*P*), recall (*R*), and F-measure (*F*) in the range of 72% - 87%.

3. The Proposed Approach

We have designed an approach for determining correspondences between schema attributes by exploring the instances of schemas. The proposed approach consists of four main phases as illustrated in Fig. 1. These phases are *analyzing instances*, *classifying schema attributes*, *identifying instance similarity*, and *identifying the match*, which are further explained in the following subsections:

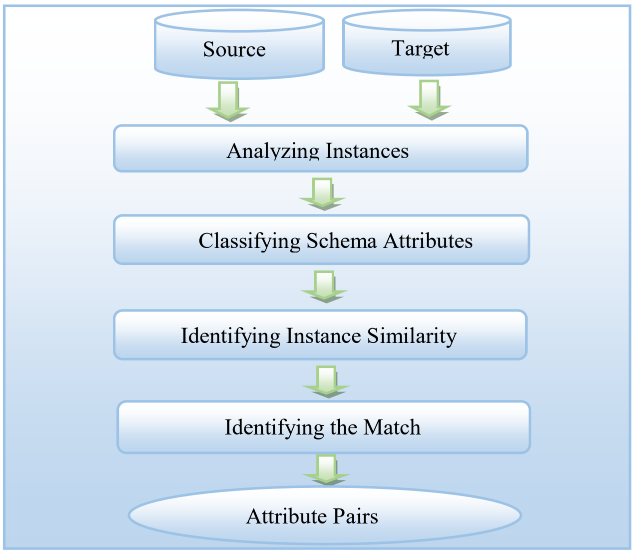

Fig. 1. The Phases of the Proposed Approach

3.1. Analyzing Instances

This phase aims at determining the data type of each attribute of both the target and source schemas. This is achieved by analyzing the characters of an instance selected randomly from each attribute of the schemas. We classify the data type of an attribute as *alphabetic*, *numeric* and *mix*. The *alphabetic* data type is for attributes whose instances consist of only alphabetic characters ([A...Z, a…z]), while the *numeric* data type is for attributes whose instances consist of only digit characters ([0…9]). The last type being the *mix* data type, is for attributes whose instances consist of combination of alphabetic, digit and special characters (e.g [-, /, \, .,]). This phase starts by randomly selecting an instance of an attribute and counts the number of characters for each data type, then checks whether the number is equal to the length of the instance or not. If the number of characters of a data type is equal to the length of the instance (without whitespace), then the data type of the instance is identified as alphabetic (if all the characters are alphabetic) or numeric (if all the characters are numeric).

Otherwise, if the number of characters of a data type is less than the length of the instance, then the data type of the instance is identified as *mix*. For example, the instance "New York" has seven *alphabetic* characters which is equal to the length of the instance (without whitespace), while the instance “255 Courtland” has three *numeric* characters and nine *alphabetic* characters which are not equal to the length of the instance which is twelve. Thus, “New York” and ”255 Courtland” are classified as *alphabetic* and *mix* data type, respectively.

3.2. Classifying Schema Attributes

After determining the data type of each attribute as discussed in the previous phase, the next step will be to classify the attributes that share the same data type in the same class. The main aim of this phase is to reduce the number of possible comparisons that needs to be performed during the matching process. The maximum number of classes created for each schema is based on the number of data types that have been determined from the previous phase. Table 1 shows an example to clarify this phase. The following instances “New York”, “Doctorate”, “255 Courtland”, “818/762-1221”, and ”49” have been classified into three data types which are *alphabetic*, *numeric*, and *mix*. Hence, three classes are created based on the identified data types. The class of alphabetic data type (*C_alpha*) includes the attributes of the instances “New York” and “Doctorate”, whereas the class of *mix* data type (*C_mix*) includes the attributes of the instances “255 Courtland” and “818/762-1221”, and the third class of *numeric* data type (*C_num*) includes the attribute of the instance “49”.

Table 1. Classifying Schema Attributes based on the Data Type.

Class of Alphabetic Data Type	
Attribute 1	Attribute 2
New York	Doctorate
Class of Mix Data Type	
Attribute 1	Attribute 2
255 Courtland	818/762-221
Class of Numeric Data Type	
Attribute 1	-
49	-

3.3. Identifying Instance Similarity

The aim of this phase is to compare the attributes in the same class that belong to different schemas, whether they are representing the same entity or not. Two tasks are carried out to find correspondences between attributes in each class. The first task utilizes regular expression for syntactic similarity while the second, utilizes Google for semantic similarity.

3.3.1 Regular Expression

Regular expression (known as regexes) is a way to describe text through pattern (format) matching and provides an easy way to identify text. Regular expression is a language used for parsing and manipulating text [25][26]. Furthermore, it's a string containing a combination of normal characters and special metacharacters or metasequences (*, +,?). Table 2 shows the most common metacharacters and metasequences in regular expression that are used in this work. Regular expression provides several advantages, as [27][29]:

– Being relatively inexpensive and does not require training or learning as in learning-based or neural network techniques.
– Provides a quick and concise method to capture valuable user knowledge about the domain.

Table 2. The Common Metacharacters in Regular Expression

Meta-character	Name	Matches
.	Dot	Matches any one character
[…]	Character class	Matches any one character listed
[^…]	Negated character class	Matches any one character not listed
?	Question	One allowed, but it is optional
*	Star	Any number allowed, but all are optional
+	Plus	At least one required; additional are optional
|	Alternation	Matches either expression it separates
^	Caret	Matches the position at the start of the line
$	Dollar	Matches the position at the end of the line
{X,Y}	Specified range	X required, max allowed

In general, regular expression of a given set can be determined by analyzing the pattern (format) of the instances. Having this regular expression, the correspondence attributes are detected by matching the regular expression with the instances of the attribute. Regular expression in this work is used to create patterns for both *numeric* and *mix* data types. In the next subsections, we will show how regular expression works for both data types.

3.3.1.1 Regular Expression for Numeric Data Type

This subsection explains the process of creating regular expression for the attributes with *numeric* data type. The attributes with *numeric* data type consist of instances with digits ranging from 0 – 9. when creating a regular expression for an attribute, the minimum and maximum values of the attribute will be required. Thus, three variables have been identified, namely: *nomin*, *nomax* and *uppervalue*. Initially, *nomin* and *nomax* are assigned the minimum and maximum values of the attribute, respectively. However, in the following iterations, the value of *nomin* is changed to the last *uppervalue* + 1. The *uppervalue* is a value which is greater than the value of *nomin* and less than the value of *nomax*; and is derived based on the following conditions:

- (i) when the *nomin's* length of digits is less than the *nomax's* length of digits, the *uppervalue* is the maximum value based on the *nomin's* length of digits and not greater than the value of *nomax*. For instance, if the *nomin's* length of digits is three (e.g. 345) then the *uppervalue* is 999. If the *uppervalue* is greater than the value of *nomax*, then the first digit of the *uppervalue* is changed to the first digit of *nomin* (399 for the above example). This is then checked against the value of *nomax*. If the new *uppervalue* is still greater than the value of *nomax* then the second digit of the *uppervalue* is changed to the second digit of *nomin* (349 for the above example). This process is repeated in which the next digit of the *uppervalue* is changed to the next digit of nomin until the condition stated in the definition of *uppervalue* is satisfied. However, if all the digits of *uppervalue* have been changed, i.e. the value of *uppervalue* is now equal to the value of *nomin*, and then the value of *nomax* is assigned to *uppervalue*. This is to reduce the number of iterations needed in identifying the *uppervalue*.
- (ii) when the *nomin's* length of digits is equal to the *nomax's* length of digits and the *nomin* has at least one zero digit on the right, the *uppervalue* is derived using the formula shown in equation (1). The equation (1) derives the closest *uppervalue* to the *nomax*. where *Sumz* is the result of *GetZeros* function (Step 13, Algorithm 1). If the equation (1) returns an *uppervalue* which does not satisfy the condition that we have stated earlier, then the steps as mentioned in (i) above are applied. For instance, if the *nomin's* length of digits is three (e.g. 120) and the *nomax's* length of digits is three (e.g. 123) then the *uppervalue* is 119 based on the equation (1). In this case, the value of *uppervalue* does not meet the definition of *uppervalue* which is greater than the value of *nomin* and less than the value of *nomax*. Then, the steps as mentioned in (i) above are applied to derive the value of *uppervalue*.

$$\text{uppervalue} = (\text{nomax} - (\text{nomax MOD Sumz} * 10) - 1) \qquad (1)$$

To create a regular expression for an attribute with *numeric* data type, an interval is derived based on the values of *nomin* and *uppervalue* as well as the *nomin's* length of digits. Then a regular expression is created for that interval. This process, i.e. deriving interval and generating regular expression for that interval, is repeated until the *uppervalue* reached the value of *nomax*. The regular expressions of these intervals are combined as one regular expression using the | operator which represents the regular expression of the attribute. The following example clarifies the process of generating regular expression for an attribute with *numeric* data type. Let the values “7” and “123” represents the minimum, *nomin*, and maximum, *nomax*, values of an attribute, respectively. From the Table 3, we can notice that there are four iterations. In the first iteration, the *nomin* has only one digit, thus the *uppervalue* is equal to 9. From this, we generate a regular expression for the values in the range of 7 - 9 as [7 - 9]. The next step is to have an interval with two digits that starts with *nomin* equals to the last *uppervalue*

+ 1 (i.e. equal to 10 as shown in iteration 2). This step has *uppervalue*, which is equal to 99 as it is the maximum number of two digits and it is less than the *nomax*. A regular expression is generated for this interval as [1-9][0-9]. In the third iteration, the *nomin's* length of digits is equal to the *nomax's* length of digits as well as the *nomin* has two zero digits on the right. Using equation (1), the *uppervalue* is equal to 119. The process builds a regular expression for this interval as 1[0-1][0-9]. In the last iteration, the *nomin* is set to 120 as the start of the new interval (i.e. *nomin* = last *uppervalue* +1). Here, the *nomin's* length of digits is equal to the *nomax's* length of digits and the *nomin* has one zero digit on the right, thus the *uppervalue* is derived using equation (1) which is also equal to 119.

Table 3: The Mechanism of the RegEx for Numerical Domain

Iteration	Nomin	Uppervalue	RegEx	Accumulated RegEx
1	7	9	[7-9]	[7-9]
2	10	99	[1-9][0-9]	[7-9]|[1-9][0-9]
3	100	119	1[0-1][0-9]	[7-9]|[1-9][0-9]|1[0-1][0-9]
4	120	123	12[0-3]	[7-9]|[1-9][0-9]|1[0-1][0-9]|12[0-3]

However, we cannot use 119 as the *uppervalue* since the *uppervalue* should be greater than the value of *nomin* and less than the value of *nomax*. Here, the steps as described in condition (i) above are applied in which the *uppervalue* is set to 999 as it is the maximum number of three digits. However, this value cannot be considered as the *uppervalue* since it is greater than the maximum value. Thus, the first digit of the *uppervalue* is changed to the first digit of *nomin* which gives the value 199. For the same reason, 199 is not the value that meets the condition stated in the definition of *uppervalue*. Thus, 129 is then generated which is still greater than *nomax*. Since changing the third digit of 129 with the third digit of *nomin* does not satisfy the definition of *uppervalue*, therefore the *uppervalue* is set to *nomax*. In this stage, the regular expression is build for the interval of *nomin* and *nomax* as 12[0-3]. Fig. 2 depicts the details steps of generating regular expression for *numeric* data type. The steps 13 and 14 check whether the *nomin* has at least one zero at the end and then finds the *uppervalue* only if the *nomin* and *nomax* have the same length. As shown in Table 3, when the *nomin* = 100 and the *nomax* = 123, step 16 computes the next *uppervalue*. This step is repeated until the computed value of the *uppervalue* is greater than the *nomin* (step 17). Steps 20 - 32 perform the otherwise. These steps select the *uppervalue* that is less than the *nomax* to be within the interval as shown in Fig. 2 step 17. After computing an *uppervalue*, a regular expression is built for the current interval. This is performed by the function *Generating_RegEx* that takes as input the *nomin* and *uppervalue* of the interval or *nomin* and *nomax* for the last iteration. This function is depicted in Fig. 5.

Algorithm 1

Input: A set of attributes with *numeric* data type, $NC\_num = \{NA_1, NA_2, \ldots, NA_n\}$

Output: Set of regular expression, $Rex = \{rex_{NA1}, rex_{NA2}, \ldots, rex_{NAn}\}$

```
BEGIN
 FOR each NAi of NC_num  DO
  BEGIN
      Let nomax = the maximum value of attribute NA_i
      Let nomin = the minimum value of attribute NA_i
      Let Lmax = the length of the nomax
      Let Lmin = the length of the nomin
      rex_NAi = { },Sumz = 0
      finish = False
       WHILE (Not finish) DO
       BEGIN
         found = False
           Sumz = GetZeros (nomin, Lmin)
         IF (Lmax = Lmin AND Sumz  > 0)THEN
           BEGIN
         uppervalue = (nomax - (nomax MOD Sumz *10)- 1)
            IF (uppervalue > nomin)THEN
            found  =  True
           END
         IF (Not found) THEN
           BEGIN
           tlmin = Lmin
         While tlmin >0 DO/*Where tlmin = Lmin, Lmin - 1,…,1
             BEGIN
             upper = GetUpper (nomin,Lmin, tlmin)
             uppervalue= GetIntegerValue(upper)
             tlmin = tlmin  -  1
               IF  (uppervalue  <=  nomax) THEN
               found  = True
               break
              END
            END
         IF (found) THEN
         BEGIN
     rex_NAi=rex_NAi+Generating_ReEx(nomin,uppervalue)+"|"
             IF (uppervalue = nomax)THEN
             finish = True
           END
         ELSE
           BEGIN
           rex_NAi = rex_NAi +Generating_ReEx(nomin, nomax)
             finish  =  True
           END
           nomin  =  uppervalue  +  1
           END
        END
```

```
    END
```

GetZeros(*nomin*, *Lmin*: Calls the Find the Number of Zero's in the *nomin* Algorithm and returns an integer which is the number of '0' digits in the nomin.
GetUpper(*nomin*, *Lmin*, *tlmin*): Calls the *GetUpper* Algorithm and returns the upper value of the *nomin*.
GetIntegerValue is a build-in function in Java programming language that converts a char to an integer data type.
Generating_ReEx(*nomin*,*uppervalue*): Calls the *Generating_ReEx* Algorithm and returns the regular expression for the values between the *nomin* and *uppervalue*.

Fig.2. Generating *RegEx* for Numeric Attributes Algorithm

Algorithm 2

Input: *nomin*, *Lmin*
Output:Number of zeros in the right most of *nomin*, *sum*

```
BEGIN
   sum  =  0,  temp [ ]  =  " "
   temp = GetCharValue (nomin)
   WHILE (temp[Lmin - 1]) == 0) AND (Lmin - 1>=0)) DO
      EGIN
        sum  =  sum  + 1
        Lmin = Lmin  - 1
      END
END
```

GetCharValue is a build-in function in Java programming language that convert an integer to a char data type.

Fig.3. Find the Number of Zero's in the *nomin* Algorithm

Algorithm 3

Input: *nomin*, *Lmin*, *tlmin*
Output:Upper value of *nomin*, *uppervalue*

```
BEGIN
   uppervalue [ ] =  " "
   uppervalue = GetCharValue (nomin)
   FOR j = 0 until tlmin - 1 DO  /* where j = 0, 1, …,
     uppervalue [(Lmin - 1) - j]  =  '9'
   END
```

Fig.4. *GetUpper* Algorithm

```
Algorithm 4
Input: X,Y
Output:Regular expression, vec
BEGIN
    vec  =  " "
    Let value1 =  GetCharValue (X)
    Let value2 =  GetCharValue (Y)
    Let len  =  length of value1
    FOR i = 0 until len DO   /* where i = 0, 1, …, len
     BEGIN
       IF (value1[i]  ==  value2[i]) THEN
        vec = vec + value1[i]   /* value2[i]
       ELSE
         BEGIN
           vec  =  vec  +   '['
           vec  =  vec  +  value1[i]
           vec  =  vec  +  '-'
           vec  =  vec  +  value2[i]
           vec  =  vec  +  ' ] '
         END
      END
    END
```

Fig.5. *Generating_ReEx* Algorithm

3.3.1.2 Regular Expression for Mix Data Type

This section presents the steps for generating regular expression for the attributes with *mix* data type. *Mix* data type includes *alphabetic*, *numeric* and *special characters*. The general idea is to divide an instance into a set of sub-tokens. Each sub-token is a sequential set of characters of a particular data type. Then, a regular expression is built for each sub-token of the instance. Finally, the regular expressions of each sub-token are combined as the regular expression of the instance. For example, the following instance "255 Courtland" can be divided into two sub-tokens which are "255" and "Courtland". The first sub-token "255" is considered as a sub-token of the *numeric* data type, since it consists of a sequential set of numeric characters. While, the second sub-token "Courtland" belongs to the *alphabetic* data type as it consists of a sequential set of alphabetic characters. Finally, we combine the regular expressions of each sub-token that are "*d*+" for the sub-token with numeric characters and "([*a-zA-Z*]+)" for the sub-token with the alphabetic characters as the final regular expression of the instance “255 Courtland”.

```
Algorithm 5
Input:A set of attributes with mix data types,NC_mix =
      {MA1,MA2,…,MAm}
Output:Set of regular expressions, Att_ReX ={Att_ReXMA1,
      Att_ReXMA2,…,Att_ReXMAm}
1. BEGIN
2. Att_ReX = {  },k = 0
3. FOR each MAi of NC mix DO
4.  Read an instance, I, randomly from MAi
5.   WHILE (k < length of I) DO
6.    BEGIN
7.      IF (Ik ∈ {A…Z, a…z}) THEN
8.       BEGIN
9.         While (k < length of I) AND (Ik ∈ {A…Z, a…z})
10.        k = k + 1
11.        rexMAi = rexMAi + "([a-zA-Z]+)"
12.       END
13.       ELSE (Ik ∈ {0..9}) THEN
14.        BEGIN
15.         While (k < length of I) AND (Ik ∈ {0..9}) DO
16.         k  = k + 1
17.         rexMAi = rexMAi + "\\d+"
18.        END
19.       ELSE (Ik ∈ special characters) THEN
20.        k = k + 1
21.        rexMAi = rexMAi + "special character"
22.       ELSE (Ik ∈  white space) THEN
23.        k = k + 1
24.        rexMAi = rexMAi + "\\s"
25.     END
26.     Att_ReXMAi = rexMAi
27.  END
28.  END
```

Fig.6. Generating *RegEx* for Mix Data Type Algorithm

Fig. 6 depicts the details steps of generating the regular expressions for the attributes with *mix* data type. The algorithm analyses each attribute, *MAi*, of the *mix* data type class, *NC_mix*, and selects randomly an instance, *I*, from each attribute, *MAi* (steps 3 and 4). The algorithm then checks each character of the selected instance whether it is *alphabetic*, *numeric* or *special character*, to determine the data type of each sub-token of the instance (steps 7, 13 and 19). If the character is an *alphabetic* character, the algorithm checks if there are a sequence of alphabetic characters (steps 9 and 10) and stop when the next character is not an *alphabetic* character. Then, step 11 considers the sequence of characters as a sub-token of *alphabetic* data type and assigned a regular expression of the sub-token to *rexMAi*. The same process is applied for *numeric* and *special characters* data types. Finally, once all characters of the instance have been checked, the final regular expression, *Att_ReXMAi*, is obtained (step 26).

3.3.2 Google Similarity Distance

The Google similarity uses the World Wide Web as a database and Google as a search engine. Google's similarity of words and phrases from the World Wide Web uses Google page counts, as shown in equation (2). Where $f(x)$ is the number of Google hits for the search term x, $f(y)$ is the number of Google hits for the search term y, $f(x, y)$ is the number of Google hits for both terms x and y together, and M is the number of web pages indexed by Google. The World Wide Web is the largest database on earth and the context information entered by millions of independent users averages out to provide automatic semantics of useful quality [30][31]. For instance, if we want to search for a given term in the Google web pages, e.g. "*Msc*", we will get a number of hits that is 108,000,000. This number refers to the number of pages where this term is found. For another term, "*Phd*", the number of hits for this term is 272,600,000. Furthermore, if we search for those pages where both terms" *Msc*" and "*Phd*" are found, that gives us 53,800,000 hits.

$$\text{GSD (x, y)} = \frac{max\ (log\ f(x), log\ f(y)) - log\ f(x, y)}{log\ M - min\ (log\ f(x), log\ f(y))} \qquad (2)$$

3.3.3 Google Similarity for Alphabetic Data Type

This approach calculates the semantic similarity score for the attributes with *alphabetic* data type that comprises instances consisting of only alphabetic characters ([A...Z, a…z]). This approach utilizes the Google similarity as explained in Fig. 7 illustrates the algorithm to find the semantic similarity in our proposed approach. The algorithm needs as input, classes of *alphabetic* data type from both source and target schemas that are constructed from the previous phase. The algorithm analyses each attribute of the source schema, *SNC_alph*, and each attribute of the target schema, *TNC_alph* (steps 6 and 7). Then, the similarity of two instances from the attributes of the different schemas is measured by calling the Algorithm 7 (step 14).

In Algorithm 7, step 2 presents the number of pages, M, indexed by Google, which is currently equal to 3,000,000,000. Steps 3, 4 and 5 are used to get the number of hits for the input instances. Then we apply the number of hits of the instances in the equation (2) (step 6). Returning to the Algorithm 6, in step 14 the similarity score is calculated by referring to Algorithm 7. If the similarity score is greater than the given *threshold1* (step 15), then the similarity score value is added to count (step 16). The *threshold1* in this work is set to 60, the same value used by previous work [34]. Then, the average similarity score for the instance $a_{k2,i}$ is calculated by dividing *count* with *Tlenght1* which is then added to the set $index_{k2}$ (step 18). For each element of $index_{k2}$ (step 20) the total average similarity score for the attribute SA_i is calculated. In step 22 the final similarity score is calculated by dividing the total average similarity score for the attribute SA_i with the number of instances of *SNC_alph*, *Tlenght2*. An average similarity score is calculated for each attribute of the source schema with each attribute of the target schema, i.e. there will be p x q average similarity scores based on our algorithm depicted in Fig. 7.

```
Algorithm 6
Input: A set of attributes of the source schema with alphabetic
       data type, SNC_alph = {SA1, SA2, …, SAp}, a set of
       attributes of the target schema with alphabetic data type,
       TNC_alph = {SB1, SB2, …, SBq}
Output: Set of similarity score, Sim_score = {scoreA1B1,
       scoreA1B2, …, scoreA1Bq, …, scoreA2B1, scoreA2B2, …,
       scoreA2Bq, …, scoreApB1, scoreApB2,…, scoreApBq}

BEGIN
    Let Tlenght1 = number of instances of TNC_alph of the
                   target schema
    Let Tlenght2 = number of instances of SNC_alph of the
                   source schema
    Outcome = 0, index_k2 = { }, sum = 0
    Let threshold1 = 60
      FOR each SA_i of SNC_alph DO
       FOR each SB_j of TNC_alph DO
       BEGIN
       FOR k2 = 0 until Tlenght2 - 1 DO
        BEGIN
         count = 0
         FOR k1 = 0 until Tlenght1 - 1 DO
          BEGIN
         Outcome = Get the Similarity (a_k2,i ∈A_i,b_k1,j ∈ B_j)
           IF (Outcome >= threshold1) THEN
           count = count + Outcome
           END
          index_k2 = index_k2 ∪ (count/Tlenght1)
          END
         FOR each element of index_k2 DO
         sum = sum + index_k2
         score_AiBj = (sum/Tlenght2) * 100
       END
      END

Get the Similarity(ak2, i, bk1, j):Calls Instance Similarity Score
Algorithm and returns the similarity score between the instance ak2, i
of attribute Ai and instance bk1, j of attribute Bj.
```

Fig.7. Find the Similarity for Alphabetic Data Type Algorithm

3.4. Identifying the Match

After we have analyzed the instances, classified the attributes, and performed the tasks of syntactic and semantic matching, the last phase of our proposed approach attempts to find the correct matching between the attributes that shared the same data type using the algorithm that is shown in Fig. 9. As shown in Fig. 9, the algorithm needs as input the classes of *numeric* and *mix* data types of the target schema for syntactic matching. As well as, the list of regular expressions that has been generated for each attribute of the source schema. While, for semantic matching the inputs are similarity scores. The algorithm starts by checking the type of class whether it is *numeric*, *mix* or *alphabetic* data type (steps 6 and 20). For *numeric* and *mix* data types the same process is performed, as they use the concept of regular expression. The algorithm analyses each element of the set of regular expressions (step 7) and the instances of each attribute, B_i, of the class of the target schema (steps 8-10). Then, step 11 counts the number of instances of B_i that matches with the regular expression. Step 12 measures the percentage of similarity for each attribute B_i with the regular expression. Then, if the maximum score among these percentages of similarity score is greater than the threshold value then we can conclude that there is a match between the regular expression which represents the attribute A_j of source schema with the attribute B_i of target schema (steps 15 and 16).

```
Algorithm 7
Input:Instance1, instance2
Output:  Google  similarity  score  between  Instance1  and
       Instance2,Google_Sim_Score
1.  BEGIN
2.   Let M = Number of pages indexed by Google
3.   x = number of hits in Google for Instance1
4.   y = number of hits in Google for Instance2
5.   Z = number of hits in Google for both Instance1
         and Instance2 together
6.
```

$$Google\_Sim\_Score = \frac{max\,(log\,f(x),\, log\, f(y)) - log\, f(x, y)}{log\, M - min\,(log\, f(x),\, log\, f(y))} * 100$$

```
7.    END
```

Fig.8. Instance Similarity Score Algorithm

On the other hand, for the *alphabetic* data type, the algorithm uses the list of similarity scores derived from the previous phase. The list of similarity scores contains the average similarity score for each attribute of the source schema with each attribute of the target schema. Hence, the algorithm gets the *highest_score* of similarity achieved between the attribute of the target schema and the attribute of the source schema (step 23) and if it is equals to or greater than the threshold2 (50), then these attributes are said to correspond to each other (steps 24 and 25).

```
Algorithm 8
Input: A set of classes from target schema, NC = {NC_alpha,
       NC_num, NC_mix}, a set of regular expressions for NC_num,
       Rex, a set of regular expressions for NC_mix, Att_ReX;
       Att_Sim_Score = {scoreA1B1, scoreA1B2, …, scoreA1Bq, …,
       scoreA2B1, scoreA2B2, …, scoreA2Bq,…, scoreApB1,
       scoreApB2,…, scoreApBq}
Output: Matching or Not

BEGIN
Let LRex_mix_num = Length of the list of Rex for
NC num /*or length of the list of Att ReX  for NC mix
Let  j = 0, i = 0, l  = 0
Let threshold2 = 50
 FOR each NC_l of NC DO
  IF (type of NC_l == "numeric" OR NC_l == "mix") THEN
FOR each elementj of Rex DO /*or elementj of Att_ReX or
NC_mix where  elementj is the regular expression of attribute Aj
FOR each Bi of NCl  DO
 FOR k = 0 until number of instances of NC_l DO
  IF (a_k, i  MATCH  rex_j of  Rex) THEN
  counter = counter + 1
  Percentage_j=counter/number of instances of NC_l*100
   END
  END
  IF (max(percentage_j)  > threshold2) THEN
  B_i MATCH  with  A_j
  END
  ELSE
   BEGIN
   FOR each A_k  DO      /*where k = 1, 2, ... , p
   FOR each B_l  DO    /*where l =  1, 2, ... , q
    BEGIN
    Let highest_score_Ak= max(score_AkB1,score_AkB2,…,
        score_AkBq)
     IF (highest_score_Ak >= threshold2) THEN
      A_k MATCH with B_l
  END
END
END
```

MATCH: is a build-in method in Java programming language that tells whether or not this value of *at*, *j* matches the given regular expression.

Fig. 9. Matching Generation Algorithm

4. Evaluation

4.1. Data Set

We used real-world data sets from two different domains: Restaurant and Census, both of which are available online [36][37]. Table 4 shows the Characteristics of data sets. For comparison purpose, we compared our proposed approach to [16][20][21] in terms of precision (*P*), recall (*R*), and F-measure (*F*). However, our proposed approach was not compared to some of the approaches that are reported in the related work section for several reasons, most importantly being that these approaches used data sets that are not accessible through the internet [4][15][17-18][22][23], and some of these approaches required specific rules [18][23][29] and user intervention [11-13] to perform the matching process.

Table 4. The Characteristics of Data Sets

Data Set	Restaurant	Census
Number of Attributes	5	11
Alphabetic Attributes	Name, Type of Food and City	workclass, education, relationship, race, sex, marital status, and native-country
Numeric Attributes	X	age, fnlwgt, Education-num and capital-gain and
Mix Attributes	Address, PhoneNumber	X
Number of Records	864	4320
Number of Instances	32561	358171

4.2. Measurements

The evaluation metrics considered in this work are precision (*P*), recall (*R*) and F-measure (*F*) that are shown in equations (3), (4) and (5), respectively. It is based on the notion of true positive, false positive, true negative, and false negative.

– *True positive* (*TP*): The number of matches (really matching) detected.
– *False positive* (*FP*): The number of matches (not really matching) detected.
– *True negative* (*TN*): The number of non-matches (really non-match) detected
– *False negative* (*FN*): The number of non-matches (really matches) detected.

$$Precision = |TP| / |TP| + |FP| \quad (3)$$

$$Recall = |TP| / |TP| + |TN| \quad (4)$$

$$\text{F-measure} = 2 * \text{Precision} * \text{Recall} / \text{Precision} + \text{Recall} \quad (5)$$

For each data set, we kept the number of attributes to 11 and 5 for Census and Restaurant, respectively. Each experiment was repeated 5 times, we then measured the precision (*P*), recall (*R*) and F-measure (*F*) and the average of all three measurements was deducted.

4.3. Results

We have conducted three analyses; (i) Analysis 1 which aims at identifying the optimal sample size of tuples, (ii) Analysis 2 aims to investigate and to prove that combining both Google similarity and regular expression as in our proposed approach achieves higher accuracy compared to utilizing Google similarity or regular expression separately and lastly, (iii) Analysis 3 which aims at comparing the performance of our proposed approach to that of the previous work with respect to precision (*P*), recall (*R*) and F-measure (*F*). The details of each analysis are presented in the following subsections. When evaluating the proposed approach, we created two sub-tables by randomly selecting the attributes from the original table of both data sets and used these two sub-tables as a source schema and target schema for the experiments. The number of attributes of each sub-table is equal to the number of attributes of the original table. However, these attributes might occur in different sequence and the same attributes might be selected more than once. These sub-tables were populated with instances selected randomly from the original table of the data sets. To represent real world cases, the number of instances of both sub-tables chosen randomly where different. We pretended that these sub-tables were two different tables that needed to have their schemas match [4][16] [20].

4.3.1 Analysis 1

In this analysis, we present the experiments of selecting the optimal sample size of tuples, which represents the size of samples that achieves acceptable results in terms of precision (*P*), recall (*R*), and F-measure (*F*). The optimal sample size is the number of tuples that are used during the phase of *identifying instance similarity* of instance based schema matching. For this analysis, several experiments have been conducted and designed in such a way that each experiment uses different size of samples starting from 5% of the actual table size. The size of samples is increased either 5% or 10% in the subsequent experiments. The experiments are ended when the precision (*P*), recall (*R*) and F-measure (*F*) are at least 96% which is close to the best results reported in the previous work [17]. From this analysis, we found that when the size of samples reached 50%, the results taken of precision (*P*), recall (*R*) and F-measure (*F*) are more satisfying than the results from previous work. Table 5 illustrates the size of samples considered in each experiment.

Table 5. Size of Samples for Each Experiment

Experiment	Size of Samples
Experiment 1-1	5%
Experiment 1-2	10%
Experiment 1-3	15%
Experiment 1-4	20%
Experiment 1-5	25%
Experiment 1-6	30%
Experiment 1-7	40%
Experiment 1-8	50%

The experiments are labeled as Experiment 1-1, Experiment 1-2, Experiment 1-3, Experiment 1-4, Experiment 1-5, Experiment 1-6, Experiment 1-7 and Experiment 1-8. These eight experiments used the same data sets. For each table, we kept the number of attributes to 11 and 5 for Census and Restaurant data sets, respectively. We repeated each experiment 5 times, measured the P, R and F and averaged these results.

4.3.1.1 Result of Analysis 1

We reported the precision (*P*), recall (*R*) and F-measure (*F*) for the experiments 1-1, 1-2, 1-3, 1-4, 1-5, 1-6, 1-7 and 1-8 as shown in Table 6 and Table 7. The percentage increases as the sample size increases. For example, the percentages are 69% and 70% for precision (*P*) and recall (*R*), respectively when the size of samples is 5%, however, when these percentages increased to 87% and 100% when the size of samples was 25%.

Although we have mentioned that acceptable results mean the results of precision (*P*), recall (*R*) and F-measure (*F*) are close to the best results as reported in previous work, however in this analysis the precision (*P*) is lower but the recall (*R*) and F-measure (*F*) are higher than those reported in the previous work [19]. Compared to the results shown in Table 6 for the Restaurant data set there is a slight different in the results of Census data set as shown in Table 7. For example, when the size of samples is 5% the precision (*P*) and recall (*R*) achieved for the Restaurant data set are 69% and 70% respectively, while for the Census data set, the precision (*P*) and recall (*R*) are 61% and 80%, respectively. The precision (*P*) and recall (*R*) increased to 81% and 96% respectively when the size of samples is 25%. The reason is due to the characteristics of Restaurant data set that consists of three attributes with alphabetic data type and two attributes with mix data types. From the results, we can conclude that 50% of the actual table size is the optimal sample size that represents the number of tuples that will be used during the phase of *identifying instance similarity* of instance based schema matching. Thus, we have stopped the experiments at this stage as the results achieved with the sample size of 50% outperformed the results reported in the previous works in terms of precision (*P*), recall (*R*), and F-measure (*F*).

Table 6. Results Related to the Restaurant Data Set for the Eight Experiments

Experiment (EX)	Size of Samples	Precision (*P*)	Recall (*R*)	F-measure (*F*)
Ex 1-1	5%	69%	70%	70%
Ex 1-2	10%	78%	80%	79%
Ex 1-3	15%	80%	94%	87%
Ex 1-4	20%	83%	98%	90%
Ex 1-5	25%	87%	100%	93%
Ex 1-6	30%	86%	100%	92%
Ex 1-7	40%	86%	100%	92%
Ex 1-8	50%	89%	100%	95%

Table 7. Results Related to the Census Data Set for the Eight Experiments

Experiment (EX)	Size of Samples	Precision (*P*)	Recall (*R*)	F-measure (*F*)
Ex 1-1	5%	61%	80%	69%
Ex 1-2	10%	70%	88%	78%
Ex 1-3	15%	74%	93%	85%
Ex 1-4	20%	76%	90%	82%
Ex 1-5	25%	81%	96%	88%
Ex 1-6	30%	86%	100%	92%
Ex 1-7	40%	91%	96%	93%
Ex 1-8	50%	97%	97%	97%

4.3.2 Analysis 2

This analysis aims to investigate and to prove that combining both Google similarity and regular expression, as in our proposed approach, achieves higher accuracy compared to utilizing Google similarity or regular expression separately. From the results that are shown in Fig.10 and Fig.11 the following can be concluded:

- Google similarity achieved better results in terms of precision (*P*), recall (*R*) and F-measure (*F*) for the Census data set compared to the Restaurant data set.
- Regular expression achieved better results in terms of precision (*P*), recall (*R*) and F-measure (*F*) for the Restaurant data set compared to the Census data set.
- For the Restaurant data set, Google similarity achieved better results with regards to precision (*P*) (60%) than regular expression (40%). However, regular expression achieved better results with regards to recall (*R*) (74%) than Google similarity (36%).
- For the Census data set, Google similarity achieved better results with regards to precision (*P*) (67%) than regular expression (38%). However, regular expression achieved better results with regards to recall (*R*) (71%) than Google similarity (47%).

These results are due to the characteristics of the data sets used in the experiments. The Restaurant data set consists of three attributes; *alphabetic* data type and two attributes with *mix* data types, while the Census data set consists of four attributes;

numeric data type and seven attributes with *alphabetic* data types. Google similarity is suitable at handling similarity between instances with *alphabetic* data type compared to instances with *numeric* and *mix* data types. For example, comparing the following instances "310/472-1211" and "818/585-0855" taken from the same attribute *PhoneNumber* of the Restaurant data set, the similarity score returned by Google similarity is 0.49, which indicates “not match” while these instances are from the same attribute. Thus, for the Restaurant data set, Google similarity is not able to find correct matches for the Address and *PhoneNumber* attributes while for the Census data set Google similarity is not able to find correct matches for the following attributes; *age*, *fnlwgt*, *Education-num* and *capital-gain*. While for regular expression, the opposite was observed. Regular expression is suitable at handling similarity between instances with *numeric* and *mix* data types compared to instances with *alphabetic* types. For example, comparing the following instances "*Canada*" and "*Bachelor*" taken from the attributes *native-country* and *education* of the Census data set, the result returned by regular expression is a match while these instances are from different attributes. Thus, for the Restaurant data set, regular expression is not able to find matches for the *Name*, *City* and *Type of Food* attributes while for the Census data set regular expression is not able to find matches for the following attributes: *workclass*, *education*, *relationship*, *race*, *sex*, *marital status*, and *native-country*.

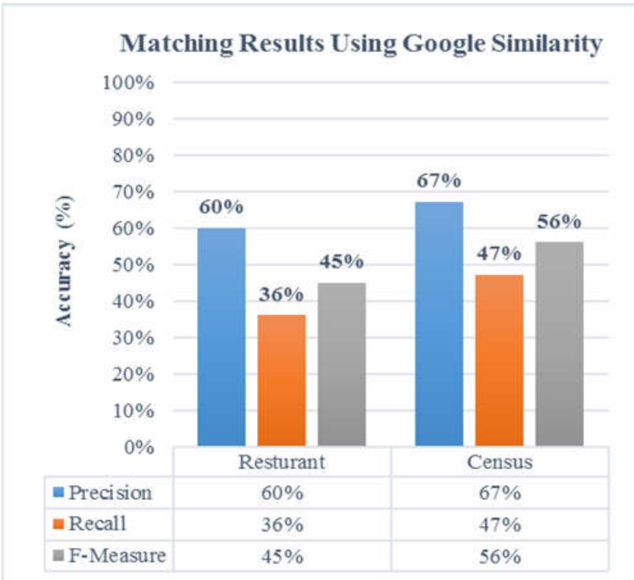

Fig.10.Matching Results using Google Similarity

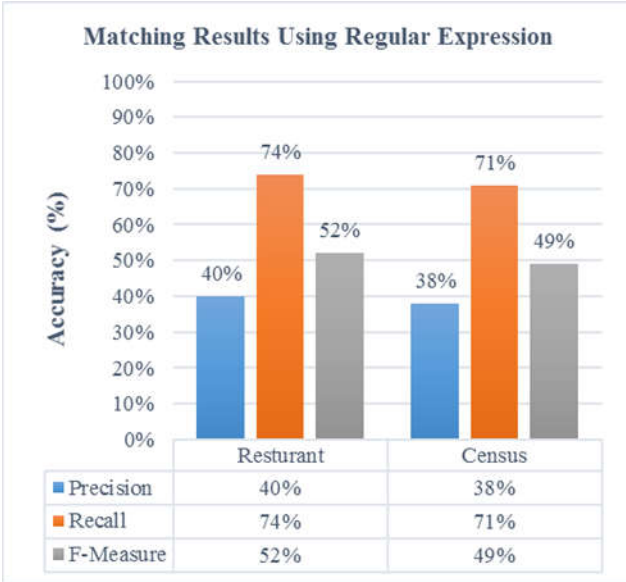

Fig.11. Matching Results using Regular Expression

4.3.3 Analysis 3

In this analysis, we focus on the performance of our proposed approach and compare it to the previous works taking into account, precision (*P*), recall (*R*), and F-measure (*F*). Fig.12 and Fig.13 show the results of accuracy in terms of precision (*P*), recall (*R*) and F-measure (*F*) for the proposed approach of instance based schema matching.

From the results seen above, the following can be concluded: (i) we achieved 96% for precision (*P*) and 93% for recall (*R*) for the Restaurant data set, while with Census data set, scores of 99 % for precision (*P*) and 97% for recall (*R*) were achieved. The size of samples used is 50% of the actual table size, which has been identified through the experiments conducted in the Analysis 1. For comparison purpose, we compared our approach to the previous approaches proposed by [16][20][21]. We evaluated [21] approach based on the two data sets, namely: Restaurant and Census. Fig. 12 and Fig. 13 show the results of our proposed approach compared to the [21] in terms of precision (*P*), recall (*R*) and F-measure (*F*). From these results the approach proposed by [21] achieved low accuracy (66%, 68% and 67% for precision (*P*), recall (*R*), and F-measure (*F*) respectively) for the Restaurant data set. While for the Census data set the approach by [21] achieved 83%, 74% and 78% for precision (*P*), recall (*R*) and F-measure (*F*), respectively. This is due to the fact that [21] approach depends on the existence of common/identical instances between the compared attributes. Furthermore, Fig. 13 shows the matching results using Census data set of our proposed approach compared to the approaches proposed by [16][20] in terms of precision (*P*), recall (*R*) and F-measure (*F*).

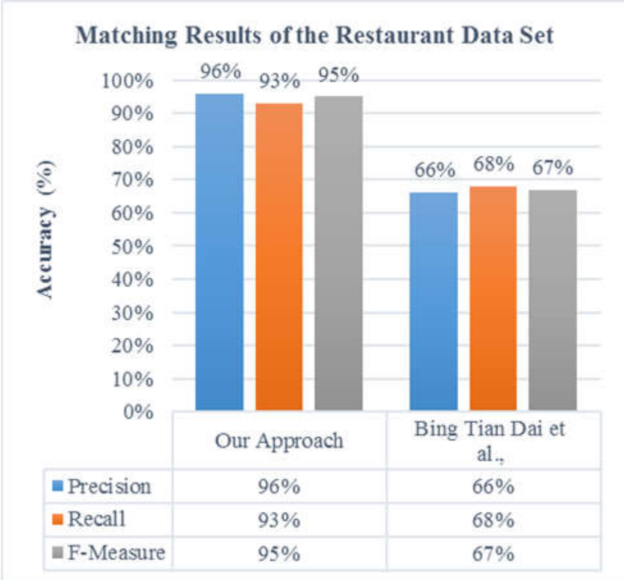

Fig.12. Matching Results of the Restaurant Data Set

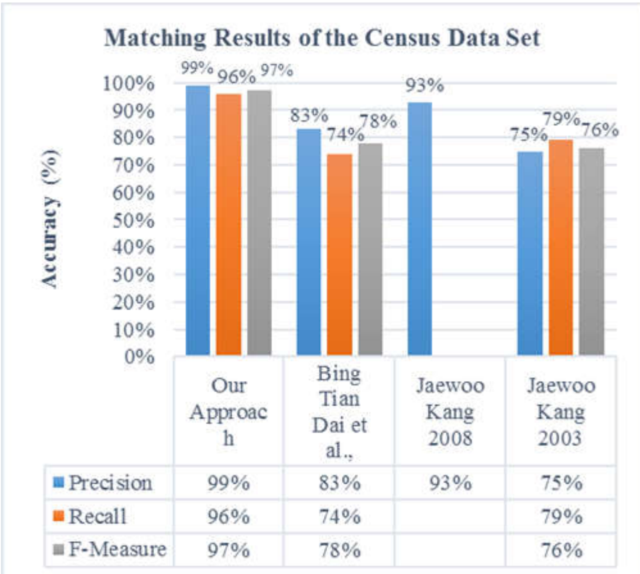

Fig.13. Matching Results of the Census Data Set

From these results, we can conclude that our proposed approach achieved better results although only a sample of instances were used instead of considering the whole instances during the process of instance based schema matching as used in the previous works [16][20][21].

5. Conclusion

In this paper, we proposed an instance based schema matching approach to identify 1-1 schema matching. Our proposed approach adopts strategies based on Google similarity as a web semantic and regular expression as pattern recognition. Our experimental results show that our proposed approach is able to identify 1-1 matches with high accuracy in terms of precision (*P*), recall (*R*) and F-measure (*F*) although only a sample of instances is used instead of considering the whole instances during the process of instance based schema matching. In the near future, we plan to extend our proposed approach to handle complex schema matching (n-m), since identifying complex matches is a more challenging problem.

References

1. Hai, D. (2007). Schema matching and mapping-based data integration: Architecture, approaches and evaluation. VDM Verlag.
2. Aničić, N., Nešković, S., Vučković, M., & Cvetković, R. 2012. Specification of data schema mappings using weaving models. *Computer Science and Information Systems*, 9(2), 539-559.
3. Madhavan, J., Bernstein, P. A., & Rahm, E. 2001. Generic Schema Matching with Cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases* (*VLDB 01*), San Francisco, CA, USA, pp. 49-58.
4. Liang, Y. 2008. An Instance-Based Approach for Domain-Independent Schema Matching. In *Proceedings of the 46th Annual Southeast Regional Conference on XX* (*ACM-SE* 46). ACM, New York, USA, pp. 268-271.
5. Feng, J., Hong, X., & Qu, Y. 2009. An Instance-Based Schema Matching Method with Attributes Ranking and Classification. In *Proceedings of the 6th international conference on Fuzzy systems and knowledge discovery*, IEEE Press, NJ, USA, Vol. 5, pp. 522-526.
6. Szymczak, M., Bronselaer, A., Zadrożny, S., & De Tré, G. 2016. Content Data Based Schema Matching. In Challenging Problems and Solutions in Intelligent Systems (pp. 281-322). Springer International Publishing.
7. Tian, A., Kejriwal, M., & Miranker, D. P. 2014. Schema matching over relations, attributes, and data values. In Proceedings of the 26th International Conference on Scientific and Statistical Database Management (p. 28). ACM.
8. Yatskevich, M., & Giunchiglia, F. 2004. Element Level Semantic Matching Using WordNet. In *Proceedings of Meaning Coordination and Negotiation Workshop*, ISWC.
9. Rull, G., Farré, C., Teniente, E., & Urpí, T. 2013. Validation of schema mappings with nested queries. *Computer Science and Information Systems*, 10(1), 79-104.
10. De Carvalho, M. G., Laender, A. H., GonçAlves, M. A., & Da Silva, A. S. 2012. An Evolutionary Approach to Complex Schema Matching. *Information Systems*, Vol. 38, No. 3, pp. 302-316.
11. Christen, P. 2012. Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution and Duplicate Detection. Springer Publishing Company, Incorporated.

12. Li, W. S., & Clifton, C. 1994. Semantic Integration in Heterogeneous Databases using Neural Networks. In *Proceedings of the 20th International Conference on Very Large Data Bases* (*VLDB 94*), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1-12.
13. Li, W. S., & Clifton, C. 2000. SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases using Neural Networks. *Data and Knowledge Engineering*, Vol. 33, No. 1, pp. 49-84.
14. Doan, A., Domingos, P., & Halevy, A. Y. 2001. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data* (*SIGMOD* 01), ACM, New York, NY, USA, Vol. 30, No. 2, pp. 509-520.
15. Berlin, J., & Motro, A. 2001. Autoplex: Automated Discovery of Content for Virtual Databases. *Cooperative Information Systems*, Springer Berlin Heidelberg, pp. 108-122.
16. Kang, J., & Naughton, J. F. 2003. On Schema Matching with Opaque Column Names and Data Values. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (*SIGMOD* 03), New York, NY, USA, pp. 205-216.
17. Li, Y., Liu, D. B., & Zhang, W. M. 2005. Schema Matching using Neural Network. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence* (*WI '05*). IEEE Computer Society, Washington, DC, USA, pp. 743-746.
18. Bilke, A., & Naumann, F. 2005. Schema Matching using Duplicates. In *Proceedings of the 21st International Conference on Data Engineering* (*ICDE 05*), IEEE Computer Society, Washington, DC, USA, pp. 69-80.
19. Yang, Y., Chen, M., & Gao, B. 2008. An Effective Content-Based Schema Matching Algorithm. In *Proceedings of the 2008 International Seminar on Future Information Technology and Management Engineering* (*FITME 08*). IEEE Computer Society, Washington, DC, USA, pp. 7-11.
20. Kang, J., & Naughton, J. F.. 2008. Schema Matching using Interattribute Dependencies. *Knowledge and Data Engineering*, IEEE Transactions, Vol. 20, No. 10, pp. 1393-1407.
21. Dai, B. T., Koudas, N., Srivastava, D., Tung, A. K., & Venkatasubramanian, S. 2008. Validating Multi-Column Schema Matchings by Type. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering* (*ICDE 08*). IEEE Computer Society, Washington, DC, USA, pp. 120-129.
22. Feng, J., Hong, X., & Qu, Y. 2009. An Instance-Based Schema Matching Method with Attributes Ranking and Classification. In *Proceedings of the 6th International Conference on Fuzzy Systems and Knowledge Discovery*, IEEE Press, NJ, USA, Vol. 5, pp. 522-526.
23. Chua, C. E. H., Chiang, R. H., & Lim, E. P. 2003. Instance-Based Attribute Identification in Database Integration. T*he VLDB Journal*, Vol. 12, No. 3, pp. 228-243.
24. Zapilko, B., Zloch, M., & Schaible, J. (2012, November). Utilizing regular expressions for instance-based schema matching. In Proceedings of the 7th International Conference on Ontology Matching-Volume 946 (pp. 240-241). CEUR-WS. org.
25. Chicago
26. Kleene, S. C. 1951. Representation of Events in Nerve Nets and Finite Automata. Automata Studies, Princeton University Press, Princeton, NJ, pp. 3-42.
27. Friedl, J. E. 2006. Mastering Regular Expressions. O'Reilly Media, Inc.
28. Doan, A., & Halevy, A. Y. 2005. Semantic Integration Research in the Database Community: A Brief Survey. AI Magazine, Vol. 26, No. 1, pp. 83-94.
29. Stubblebine, T. 2007. Regular Expression Pocket Reference: Regular Expressions for Perl, Ruby, PHP, Python, C, Java and. NET. O'Reilly.
30. Goyvaerts, J., & Levithan, S. 2009. *Regular Expressions Cookbook*. O'reilly.
31. Cilibrasi, R. L., & Vitanyi, P. M.. 2007. The Google Similarity Distance. *Knowledge and Data Engineering*, *IEEE Transactions*, Vol. 19, No. 3, pp. 370-383.
32. Cilibrasi, R., & Vitanyi, P. 2004. Automatic Meaning Discovery using Google. *Technical Report*, University of Amsterdam, National ICT of Australia, pp. 1-31.

33. Rahm, E., & Bernstein, P. A. 2001. A Survey of Approaches to Automatic Schema Matching. The VLDB Journal, Vol. 10, No. 4, pp. 334-350.
34. Blake, R. 2007. A Survey of Schema Matching Research. *College of Management Working Papers*, University of Massachusetts Boston, Paper 3.
35. Partyka, J., Parveen, P., Khan, L., Thuraisingham, B., & Shekhar, S. 2011. Enhanced Geographically Typed Semantic Schema Matching. *Web Semantics: Science*, Services and Agents on the World Wide Web, Vol. 9, No. 1, pp. 52-70.
36. http://www.cs.cmu.edu/~mehrbod/RR/
37. http://archive.ics.uci.edu/ml/datasets.html

Osama A. Mahdi is a PhD candidate at the Department of Computer Science and information technology La Trobe University, Melbourne, Australia. He obtained his M.Sc. in database from the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia in 2014. His B.Sc. degree in Computer Science from Babylon University, Iraq in 2009. His current research interests include Data Stream Mining, Concept Drift and Data Integration (Schema Matching).

Hamidah Ibrahim is currently a professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. She obtained her PhD in computer science from the University of Wales Cardiff, UK in 1998. Her current research interests include databases (distributed, parallel, mobile, bio-medical, XML) focusing on issues related to integrity constraints checking, cache strategies, integration, access control, transaction processing, data stream, data analytic, and query processing and optimization; data management in grid and knowledge-based systems.

Lilly Suriani Affendey is an Associate Professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM). She received her Bachelor of Computer Science from University of Agriculture, Malaysia in 1991 and MSc. in Computing from the University of Bradford, UK in 1994. In 2007 she received her PhD in Database Systems from University Putra Malaysia. Her research interest includes multimedia databases, video retrieval, data science and big data analytics.

Eric Pardede received his PhD in Computer Science from La Trobe University, Melbourne, Australia. He is currently a Senior Lecturer in the Department of Computer Science and Information Technology at La Trobe University, Australia. He has wide range of teaching and research experience. His current research interests include data analytics, higher education pedagogy and IT entrepreneurship.

Jinli Cao received her PhD in Computer Science from Department of Mathematics and Computing University of Southern Queensland, Australia in 1997. She is currently a senior lecturer in Department of Computer Science and Computer Engineering of La Trobe University. She has wide range of teaching and research experience. Her current research interests include Data Quality, Big Data Analytics, Recommendation Systems and Query Mining.

Received: May 25, 2017; Accepted: February 20, 2018.