ACMA documentation

Pablo Mazzucchi

December 6, 2014

Contents

Chapter 1

Introduction

Entity matching is a problem that concerns to many data management processes. If matching between entities represented by RDF individuals is considered, lists of attribute values with variable-length might be found for some properties, which lead to the problem of comparing multi-valued attributes, e.g. comparing author name lists for determining publication matching. This matching technique is more complex than comparing fixed-length records, but less complex than comparing XML documents. Instead of comparing a single string, representing the concatenation of these values, each value of one vector must be compared against all values of the other vector. In this document an heuristic to address the alignment and comparison process of multi-valued attributes is proposed and details about how to implement it in OYSTER are provided.

Before giving details about the implemented solution, lets start defining what Heuristic means. The etymology of the word is Greek and means "find" or "discover". In the psychology context, it refers to experience-based techniques for problem solving, learning, and discovery that find a solution which is not guaranteed to be optimal, but good enough for a given set of goals. Where the exhaustive search is impractical [1], heuristic methods are used to speed up the process of finding a satisfactory solution via mental shortcuts to ease the cognitive load of making a decision. Examples of this method include using a rule of thumb, an educated guess, an intuitive judgment, stereotyping, or common sense. Basically, heuristics are simple, efficient rules, learned or hard-coded by evolutionary processes, that have been proposed to explain how people make decisions, come to judgments, and solve problems typically when facing complex problems or incomplete information. These rules work well under most circumstances.

Doing an analogy with Computer Science [8, 10, 9] an heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution. This solution may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.

In Section 2 details about the heuristic proposed to compare multivalued attributes are given. Finally, in Section 3 a step by step guide to implement this method into OYSTER is provided.

[1] In the context of this document, the exhaustive search could be translated to all-against-all comparisons, and it is not efficient, rather than impractical

Chapter 2

Solution Model

To establish a degree of similarity between two attributes that contains multiple instances (multivalued attributes) requires many comparisons to be performed between the different values in order to finally aggregate them. In the publication domain, could be found that many authors collaborate in the same research. So for determining the similarity of the list of authors, given two papers, the following non trivial problems could appear: decide which authors should be compared, and, how to deal with the partial comparison to get a final result.

To attack that problem, the model showed in Figure 2.1 was designed. It consists of three stages: alignment, comparison and similarity aggregation.

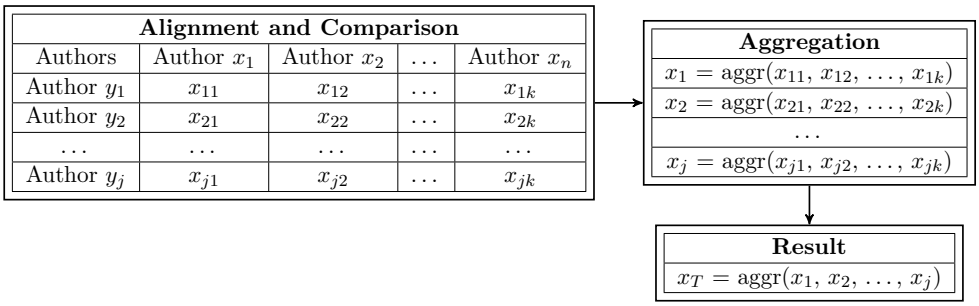

Figure 2.1: Multi-valued attributes Alignment and Comparison Process.

The strategies used in each stage are described in Sections 2.1,2.2 and 2.3. Finally, in Section 2.4 details about ACMA are provided, and in Section 2.5 some examples of how it works are shown.

2.1 Alignment

The alignment phase consists on arranging elements of both lists for having a comparison matrix where the diagonal $\{(1,1),...,(N,N)\}$ denotes the right comparisons to make. In some cases, multi-valued attributes are already ordered hence this step might not be necessary, e.g. publication author lists.

According to the clustering and blocking methods used in the Google Refine project [2], Fingerprint [7] resulted the most useful and promising approach to align multiple

attribute values in such a way that the comparison process performance is increased. Basically, Fingerprint splits a string in tokens and order them alphabetically.

2.2 Comparison Strategies

The proposed comparison strategies try to minimize the number of comparisons between the values of the different attributes. The combination of comparison strategies determine which elements of the comparison matrix $\mathcal{M}$ are filled: the less comparisons, the better. Consider the lists of instances of each attribute $A_1[N]$ and $A_2[M]$, being $N \geq M$. The comparison strategies will determine which element of $A_1[i]$ ($0 < \mathrm{i} < \mathrm{N}$) and $A_2[j]$ ($0 < \mathrm{j} < \mathrm{M}$) will be compared. The result of the comparison (the individual similarity) is stored in $\mathcal{M}[i, j]$. The comparison strategies used were:

- **Row-Col-Deletion**: if $\mathcal{M}[i, j] \geq$ min-similarity, comparisons $\mathcal{M}[i, b]$ and $\mathcal{M}[a, j]$ are skipped for $i < a < N$ and $j < b < M$ respectively. This means that the comparison process will not consider anymore those cells included in the columns and rows where a match is found. This means that if a match is found in a row, all the remaining cells in that row will not be consider in the comparison process.
- **Row-Similarity-Threshold**: if $\mathcal{M}[i, j] \geq$ min-similarity, continue comparing in the next row of the matrix ($\mathcal{M}[i + 1, j]$), i.e. $\mathcal{M}[i, b]$ is left empty for $j < b < M$.

Additionally a stop criterion for avoiding unnecessary comparisons was defined:

- **Partial-Average-Threshold**: compare elements in $A_1[i]$ and $A_2[j]$ until reaching 80% of $\mathcal{M}$. If Max-Average (described in Section 2.3) is above min-similarity, continue with the comparison process; otherwise, stop.

Given the nature of the element comparison problem and based on the taxonomy described by [1], *Content-based* matching approaches were used to determine the similarity of two entities. The decision was to combine two techniques that compare atomic values: one for character-to-character comparison (*Jaro Winkler*), and another for token-to-token comparison (*Longest Common Substring (LCS)*). For combining the results of both of them, the *root mean square (RMS)* was used, to improve the similarity grade over a single comparator [5, 3]. Those instances with a similarity score above a defined threshold were considered similar.

2.3 Similarity Aggregation

Aggregation strategies try to determine the final similarity grade between the processed lists of instances. The strategies used are:

- **Average**: $\frac{\sum_{i=0}^{N}\sum_{j=0}^{M}\mathcal{M}[i,j]}{(N\times M)}$. In plain words, all the cells's values will be added up and the total of that sum, divided by the amount of cells or comparisons made.
- **Max-Average**: Max(Avg (x), Avg(y)), where $x = \sum_{i=0}^{N} Max(\mathcal{M}[i, *])$ and $y = \sum_{j=0}^{M} Max(\mathcal{M}[*, j])$. In plain words Max-Average chooses between the maximum average of the sum of maximum values obtained from column or rows.

Table 2.1: Subset of authors

	Bymaster, A	Marshall, B.	Ballal, D	Emborsky, C	Chapman, W. G	Feng, Z
Bymaster, A	1	0	0	0	0	0
García-Cuéllar, A. J	0	0.55	0.45	0.31	0.37	0.34
Marshall, B	0	1	0	0	0	0
Ballal, D	0	0	1	0	0	0
Emborsky, C	0	0	0	1	0	0
Chapman, W. G	0	0	0	0	1	0
Cox, K. R.	0	0	0	0	0	0.31
Feng, Z	0	0	0	0	0	1
Gong, K	0	0	0	0	0	0

Table 2.2: Subset not being useful

	Rodriguez-Gonzalez, C.	Martínez-Lauranchet, J.	Probst-Oleszewski, O. M.
Llamas-Terrés, A. R.	0.37	0.48	0.42
Morales-Hernández, A.	0.43	0.57	0.37
Rodriguez-Gonzalez, C. A.	1	0	0
Probst-Oleszewski, O. M.	0	0.37	1

An interesting problem when comparing multivalued attributes is the appearance of subsets of instances. For example, consider the Table 2.1, the list of authors that appears as columns are a subset of the list that appears as rows. If the Max Average aggregation is used, then the similarity grade would be 0.76. But could be that the list of authors of one publication is incomplete, as this example shows, and even the title and journal be the same, the final decision whether the publications that are being compared are the same or not be affected. That is the reason of why a new feature was added into the comparator: the capability of identifying if a list is a subset of the other list. With this new add-in, the similarity grade of the case showed in Table 2.1 rise to 1. But not always the subset utility would be helpful. Consider the case in Table 2.2. It is not a subset, as just 2 of the 3 authors that appear in the columns matched. But if the subset property is used when calculating the similarity, the result would be that the lists of authors match, and even they are similar (but not the same) the respective publications are not.

2.4 ACMA details

A selection of an alignment choice (original or Fingerprint), a combination of comparison strategies, and an aggregation style was considered an heuristic. In [6] several heuristics

were developed with the objective of increasing the effectiveness in the alignment and comparison process of multi-valued attributes. But in this document, just H_4 (or ACMA) is presented, as it was the one implemented in OYSTER. A summary and a brief description of it can be found in Table 2.3. Comparison strategies are decomposed in two criteria: one for choosing which elements to compare next, and another for determining when to stop comparing. The aim sought when designing ACMA was to reduce the amount of comparisons needed to determine the similarity or not between two multivalued attributes. It is expected that the comparison's strategies guide the heuristic to just compare the values of the attributes placed in the main diagonal, or to find the best possible match for one of them, and then continue looking the next instance's best matching candidate. If the values do not match, just with a few comparisons would be enough to determine that has not sense continuing with the comparison process.

Table 2.3: Summary of ACMA.

Heuristic	Alignment	Comparisons		Aggregation
ACMA	Original Order	Row-Col-Deletion + Partial-Average-Threshold		Max-Average
		Who's Next	**Stop Criteria**	
		if ($A_1 \wedge A_2$ are visible) then Row-Col-Deletion Row-Similarity-Threshold else if (j < M) then continue next column (j=j+1) else continue next row (i=i+1)	Partial-Average-Threshold	

2.5 Examples

In order to understand, highlight key aspects and see in detail how ACMA works, it was tested using two lists of authors, due author in the publication domain is considered a multivalued attribute. The first list was: *Moghaddam, S. J.; Clement, C.; De la Garza, M. M.; Zou, X.; Travis, E. L.; Young, H. W. J.; Evans, C. M.; Tuvim, M. J.;Dickey, B. F.*. The second author list used was: *Moghaddam, Seyed Javad; Clement, Cecilia G.; De la Garza, M. Miguelina; Zou, Xiaoyan; Travis, Elizabeth L.; Young, Hays W. J.; Evans, Christopher M.; Tuvim, Michael J.; Dickey, Burton F.*. It can be seen that the authors included in the second list have more detailed names than the authors of the former list. With the objective of showing the order and the amount of comparisons made by ACMA, a sub-index was included to all the similarity values obtained in the comparison process. The results obtained are in Table 2.4. As the list were similar, just a few comparisons were needed to align properly the values.

In Table 2.5 could be observed the results when the lists of authors were different. Not all the comparisons were performed to determine the inequality. It was enough to perform 80% of them.

Table 2.4: Alignment and Comparison process performed by ACMA

Authors	Moghaddam, Seyed Javad	Clement, Cecilia G.	De la Garza, M. Miguelina	Zou, Xiaoyan	Travis, Elizabeth L.	Young, Hays W. J.	Evans, Christopher M.	Tuvim, Michael J.	Dickey, Burton F.
Moghaddam, S.,J.	$\mathbf{0.94}_{1}$								
Clement, C.		$\mathbf{0.92}_{2}$							
De la Garza, M. M.			$\mathbf{0.94}_{3}$						
Zou, X.				$\mathbf{0.91}_{4}$					
Travis, E.,L.					$\mathbf{0.91}_{5}$				
Young, H.,W. J.						$\mathbf{0.96}_{6}$			
Evans, C.,M.							$\mathbf{0.87}_{7}$		
Tuvim, M.,J.								$\mathbf{0.94}_{8}$	
Dickey, B.,F.									$\mathbf{0.95}_{9}$

Table 2.5: Alignment and Comparison process performed by ACMA when no similarity exists

Authors	Sampayo-Reyes, A.	Said-Fernandez, S.	Cerda-Flores, R.	Cortés-Gutiérrez, E.	Narro-Juárez, A.	Lozano-Garza, H.	González Garza-y Barrón, M. T.	Morales-Aguilera, G. A.
Moghaddam, S.,J.	$\mathbf{0.52}_{1}$	$\mathbf{0.43}_{2}$	$\mathbf{0.38}_{3}$	$\mathbf{0.35}_{4}$	$\mathbf{0.31}_{5}$	$\mathbf{0.33}_{6}$	$\mathbf{0.32}_{7}$	$\mathbf{0.46}_{8}$
Clement, C.	$\mathbf{0.33}_{9}$	$\mathbf{0.33}_{10}$	$\mathbf{0.4}_{11}$	$\mathbf{0.38}_{12}$	$\mathbf{0.3}_{13}$	$\mathbf{0.4}_{14}$	$\mathbf{0.34}_{15}$	$\mathbf{0.43}_{16}$
De la Garza, M. M.	$\mathbf{0.39}_{17}$	$\mathbf{0.42}_{18}$	$\mathbf{0.35}_{19}$	$\mathbf{0.36}_{20}$	$\mathbf{0.36}_{21}$	$\mathbf{0.55}_{22}$	$\mathbf{0.54}_{23}$	$\mathbf{0.53}_{24}$
Zou, X.	$\mathbf{0.33}_{25}$	$\mathbf{0.35}_{26}$	$\mathbf{0.27}_{27}$	$\mathbf{0.32}_{28}$	$\mathbf{0.39}_{29}$	$\mathbf{0.48}_{30}$	$\mathbf{0.35}_{31}$	$\mathbf{0.36}_{32}$
Travis, E.,L.	$\mathbf{0.42}_{33}$	$\mathbf{0.44}_{34}$	$\mathbf{0.4}_{35}$	$\mathbf{0.41}_{36}$	$\mathbf{0.39}_{37}$	$\mathbf{0.37}_{38}$	$\mathbf{0.37}_{39}$	$\mathbf{0.45}_{40}$
Young, H.,W. J.	$\mathbf{0.44}_{41}$	$\mathbf{0.3}_{42}$	$\mathbf{0.33}_{43}$	$\mathbf{0.36}_{44}$	$\mathbf{0.43}_{45}$	$\mathbf{0.45}_{46}$	$\mathbf{0.39}_{47}$	$\mathbf{0.45}_{48}$
Evans, C.,M.	$\mathbf{0.38}_{49}$	$\mathbf{0.45}_{50}$	$\mathbf{0.48}_{51}$	$\mathbf{0.39}_{52}$	$\mathbf{0.39}_{53}$	$\mathbf{0.39}_{54}$	$\mathbf{0.36}_{55}$	$\mathbf{0.43}_{56}$
Tuvim, M.,J.								
Dickey, B.,F.								

Chapter 3

ACMA in OYSTER

In Section 3.1, details about how and where to download the source code of ACMA are given, then in Section 3.2, the list of changes that should be performed in OYSTER to start using ACMA, and finally in Section 3.3 details of how to setup this new comparator.

3.1 Download ACMA

1. Download the source code from GitHub [1].
2. Everything could be extracted, but just the folder named as *src* would be enough to start using ACMA. All the classes and methods needed by OYSTER are there.
3. Create a folder inside the Oyster project called *ACMA*. It should be in the following path:

$$\backslash workspace\backslash Oyster\backslash src\backslash edu\backslash ualr\backslash oyster\backslash utilities\backslash ACMA \tag{3.1}$$

4. Copy all the extracted folders (*blocking, core, source* and *string matching*) into 3.1.

3.2 Modifying OYSTER

Once the *java* files had been added into the OYSTER project, the following changes should be performed on the class *OysterCompareDefault*, located in $Oyster\backslash src\backslash edu\backslash ualr\backslash oyster\backslash association.matching$, as showed in Figure 3.1.

[1] https://github.com/pmazzu/ACMA_OYSTER

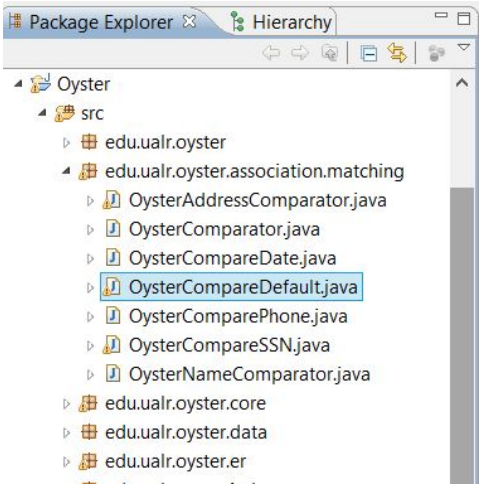

Figure 3.1: OYSTER class that should be modified .

1. First is necessary to import following ACMA packages, as shown by Figure 3.2.

```
import edu.ualr.oyster.utilities.acma.core.ACMA;
import edu.ualr.oyster.utilities.acma.core.Comparison;
import edu.ualr.oyster.utilities.acma.log.LogAdministrator;
```

```
import edu.ualr.oyster.utilities.Tanimoto;
import edu.ualr.oyster.utilities.TverskyIndex;
// BEGIN MODIFICATION
import edu.ualr.oyster.utilities.acma.core.ACMA;
import edu.ualr.oyster.utilities.acma.core.Comparison;
import edu.ualr.oyster.utilities.acma.log.LogAdministrator;
// END MODIFICATION
import java.util.Locale;
import java.util.logging.Logger;
```

Figure 3.2: Packages that should be imported.

2. Then the ACMA comparator should be defined as a private variable, inside the class *OysterCompareDefault*, as shown by Figure 3.3.

```
private ACMA acma;
```

3. Inside the method *OysterCompareDefault*, the ACMA comparator should be added in the list of used comparators, as shown in the Figure 3.4.

```
"SMITHWATERMAN", "SCAN", "SUBSTRLEFT", "SUBSTRRIGHT", "SUBSTRMID", "
    PSUBSTR", "NICKNAME", "ACMA"};
```

4. Then, a new object of the ACMA class should be created, as shown by Figure 3.5.

```
cma = new ACMA();
```

5. In the method *getMatchCode*, the following variables should be defined, as shown by Figure 3.6

```
/** Nickname/Alias lookup */
private OysterNickNameTable nnTable;

// BEGIN MODIFICATION
private ACMA acma;
// END MODIFICATION

// <editor-fold defaultstate="collapsed" desc=" UML Marker ">
// #[regen=yes,id=DCE.D4C8F28B-5411-E71A-048B-3814127D8C66]
// </editor-fold>
/**
```

Figure 3.3: Variable definition.

```
String [] arr = {"INITIAL","TRANSPOSE","LED","QTR","JACCARD","SORENSEN",
    "TANIMOTO","TVERSKY","SOUNDEX","DMSOUNDEX","IBMALPHACODE","MATCHRATING",
    "NYSIIS","CAVERPHONE","CAVERPHONE2","METAPHONE","METAPHONE2","NEEDLEMANWUNSCH",
// BEGIN MODIFICATION
    //"SMITHWATERMAN","SCAN","SUBSTRLEFT","SUBSTRRIGHT","SUBSTRMID","PSUBSTR","NICKNAME"};
    "SMITHWATERMAN","SCAN","SUBSTRLEFT","SUBSTRRIGHT","SUBSTRMID","PSUBSTR","NICKNAME",
 "ACMA"};
// END MODIFICATION
```

Figure 3.4: List of comparators.

```
String threshold_acma = "";
String list_comparators = "";
String aggrMode = "";
```

6. Then, the logic to parse the input parameters should be added, as shown by Figure 3.7. Note that the parameters should be comma separated, and the first parameter that should be sent is the threshold, then the list of comparators and finally the aggregation method. More details are given in the Section 3.3.

```
} else if (matchType.toUpperCase(Locale.US).startsWith("ACMA(")){
                matchType = matchType.trim().substring(5, matchType.
                    length()-1);

    String [] temp = matchType.split("[,]");
    threshold_acma = temp[0].trim();
    list_comparators = temp[1].trim();
    aggrMode = temp[2].trim();

    matchType = "ACMA";
}
```

7. Finally the ACMA method is invoked, as shown by Figure 3.8.

```
} else if (matchType.equalsIgnoreCase("ACMA") && acma.
    multivalued_attr_similarity_calc(s, t, threshold_acma,
    list_comparators, aggrMode)){
        result = tempMatchType;
```

8. The ACMA invocation should be added twice, as OYSTER's logic check whether an operator is defined or not. There is just a little modification compared with the fist invocation, as could be appreciated in Figure 3.9.

```
            metaphone = new Metaphone();
            metaphone2 = new DoubleMetaphone();
//            metaphone3 = new Metaphone3();

//            needlemanWunsch = new NeedlemanWunsch();
            smithWaterman = new SmithWaterman();

            nnTable = new OysterNickNameTable();
            substr = new CharacterSubstringMatches();
            scan = new Scan();

            // BEGIN MODIFICATION
            acma = new ACMA();
            // END MODIFICATION
        }
```

Figure 3.5: Creation of the ACMA comparator object.

```
public String getMatchCode (String s, String t, String matchType) {
    String result, tempMatchType = matchType, direction = "", charType = "",
            upperCase = "", order = "";
    int sLen = 0, tLen = 0, length = 0, start = 0;
    float qTRThreshold = 0.25f, ledThreshold = 0.8f;
    // BEGIN MODIFICATION
    String threshold_acma = "";
    String list_comparators = "";
    String aggrMode = "";
    // END MODIFICATION
    float match = 1f, mismatch = 0f, gap = 0f, alpha = 0f, beta = 0f;
    boolean not = false;
```

Figure 3.6: Definitions of more variables.

```
} else if (matchType.equalsIgnoreCase("ACMA") && !acma.
    multivalued_attr_similarity_calc(s, t, threshold_acma,
    list_comparators, aggrMode)) {
        result = tempMatchType;
```

3.3 ACMA Setup

Finally, to compare multivalued attributes, the OYSTER attributes XML file should be set up. The rules are defined there (for more details refer to [4]), so for those that use multivalued attributes, the ACMA comparator could be added as shown by Figure 3.10. The parameters that should be passed are:

1. Threshold: numeric value in the domain [0-1]. It depends on the comparison method used.

2. Comparison method: as described in Section 2.2, the implemented method is *rms*.

3. Aggregation: *STRICT* if it is desired that just the aggregation method described in Section 2.3 without certifying that the values of one of the attributes are a subset of the other attribute's values or *SUBSET* if it is required to verify that the values of one of the multivalued attributes are a subset of values of the other.

```
            length     = Integer.parseInt(temp[2].trim());
            upperCase  = temp[3].trim();
            order      = temp[4].trim();

            matchType = "SCAN";
        // BEGIN MODIFICATION
        //}
        } else if (matchType.toUpperCase(Locale.US).startsWith("ACMA(")){
            matchType = matchType.trim().substring(5, matchType.length()-1);

            String [] temp = matchType.split("[,]");

            threshold_acma = temp[0].trim();
            list_comparators = temp[1].trim();
            aggrMode = temp[2].trim();

            matchType = "ACMA";
        }
```

Figure 3.7: Parsing the input parameters of ACMA.

```
        result = tempMatchType;
    } else if (matchType.equalsIgnoreCase("PSubStr") && substr.properSubString(s, t, length))
        result = tempMatchType;
    } else if (matchType.equalsIgnoreCase("SCAN") && scan.compareScan(s, t, direction, charTy
        result = tempMatchType;
// BEGIN MODIFICATION
    } else if (matchType.equalsIgnoreCase("ACMA") && acma.multivalued_attr_similarity_calc(s,
        result = tempMatchType;
// END MODIFICATION
    } else {
        result = "X";
    }
    } else {
        if (matchType.equalsIgnoreCase("Initial") && !(s.charAt(0) == t.charAt(0)
                && ((sLen == 1 && tLen > 1) || (sLen > 1 && tLen == 1)))) {
            result = tempMatchType;
```

Figure 3.8: Calling ACMA method.

The order in which the parameters were passed must be respected.

```
        } else if (matchType.equalsIgnoreCase("SubStrMid") && !substr.mid(s, t, start, length)
            result = tempMatchType;
        } else if (matchType.equalsIgnoreCase("PSubStr") && !substr.properSubString(s, t, leng
            result = tempMatchType;
        } else if (matchType.equalsIgnoreCase("SCAN") && !scan.compareScan(s, t, direction, ch
            result = tempMatchType;
       // BEGIN MODIFICATION
        } else if (matchType.equalsIgnoreCase("ACMA") && !acma.multivalued_attr_similarity_cal
            result = tempMatchType;
       // END MODIFICATION
        } else {
            result = "X";
        }
    }
    return result.toUpperCase();
```

Figure 3.9: The second call of the ACMA method.

```
<Rule Ident="2">
    <Term Item="Title" MatchResult="LED"/>
    <Term Item="Authors" MatchResult="ACMA(0.82,rms, SUBSET)"/>
    <Term Item="ISSN" MatchResult="Exact"/>
</Rule>
<Rule Ident="3">
    <Term Item="Title" MatchResult="LED"/>
    <Term Item="Authors" MatchResult="ACMA(0.82,rms, STRICT)"/>
    <Term Item="Journal" MatchResult="LED"/>
</Rule>
```

Figure 3.10: Rule definition.

List of Figures

List of Tables

Bibliography

[1] Carina Friedrich Dorneles, Rodrigo Gonçalves, and Ronaldo Santos Mello. Approximate data instance matching: a survey. *Knowledge and Information Systems*, 27(1):1–21, April 2010.

[2] Google. Google Refine Project, 2012.

[3] Shaun J Grannis, J Marc Overhage, and Clement McDonald. Real world performance of approximate string comparators for use in patient matching. *Studies in health technology and informatics*, 107(Pt 1):43–7, January 2004.

[4] Fumiko Kobayashi and John Talburt. *Oyster v3.3 User Guide*. ERIQ.

[5] Hanna Köpcke, Andreas Thor, and Erhard Rahm. Comparative evaluation of entity resolution approaches with FEVER. In *Proc. 35th Intl. Conference on Very Large Databases (VLDB)*, 2009.

[6] Pablo N. Mazzucchi and Hector Ceballos Cancino. An Aligment Comparator for Entity Resolution with Multi-valued Attributes. 2014.

[7] Tom Morris and David Huynh. FingerPrint Method, 2010.

[8] Allen Newell and Herbert A Simon. Computer science as empirical inquiry: Symbols and search. *Communications of the ACM*, 19(3):113–126, 1976.

[9] Allen Newell and Herbert A. Simon. Computer science as empirical inquiry: Symbols and search. *Commun. ACM*, 19(3):113–126, March 1976.

[10] Judea Pearl. *Heuristics*. Addison-Wesley Publishing Company Reading, Massachusetts, 1984.