Duplicate Record Detection in Data Cleaning Using DCS++ Algorithm

Yin Yin Phyo, Thidar Win
University of Computer Studies, Yangon
yinyinphyo@ucsy.edu.mm, thidarwin@ucsy.edu.mm

Abstract

Duplicate Record Detection is a multiple record search process that represents the same physical entity in a dataset. It is also known as the record linkage (or) entity matching [1]. The databases contain very large datasets. Datasets contain duplicate records that do not share a common key or contain errors such as incomplete information, transcription errors and missing or differing standard formats (non-standardized abbreviations) in the detailed schemas of records from multiple databases. So, the duplicate detection needs to complete its process in a very shorter time. Duplicate detection requires an algorithm for determining whether records are duplicate records or not.

In this paper, calculate a similarity metric that is commonly used to find similar field items and use the Duplicate Count Strategy Multi-Record Increase (DCS++) Algorithm for approximately duplicate records detection over publication xml dataset.

1. Introduction

Nowadays, the amount of data within the datasets becomes more and more huge. Data errors or inconsistent data in these datasets also grow rapidly as the technology advances. In the economic world, invalid and duplicate data can be costly because it can affect the key decisions of many industry operations and the production of business organizations. Therefore, data needs to be good quality.

In order to improve the data quality, data cleaning is especially necessary when integrating disparate data sources [2]. When integrating data from different sources and implementing a data warehouse, organizations become aware of possible differences and systematic conflicts.

The problem of identifying duplicate records in the database is an important step in the data cleanup and integration process. Data reduction is the process of detecting and removing data errors, inconsistencies, and duplicate data. Duplicate detection is one of the solutions of data cleaning. It has two tasks to detect duplicate records efficiently and effectively:

(i) The representation of the data may vary slightly, so a specific similarity measure needs to be defined to compare pairs of records.

(ii) Not all records can be peer compared because the data set may be large.

To perform task (ii), a number of algorithms have been proposed that split the dataset and compare all pairs of records in each partition.

Sorted Neighborhood Method (SNM) is a known way to advance the window by classifying data based on the sorting key and comparing only the records displayed in the same window.

This paper proposes the Duplicate Count Strategy-Multi Record Increase Approach (DCS++), a variation of SNM and improvement of Duplicate Count Strategy (DCS). If a duplicate is found on the sorted dataset, it can also detect the other possible duplicates by comparing the next w - 1 record of that duplicate.

It can also reduce the comparing time by skipping windows for duplicates. Therefore, the proposed system can be faster and detects more duplicate records.

2. Related Work

Many researchers research on duplicate record detection with different efficient and effective blocking and windowing methods [3].

Ying Pei et al. [4] implemented the K-medoids clustering algorithm (IKMC) to solve the problem of detecting almost duplicate records. It is considered as one separated data object for every record in the database. It uses the Edit Distance method to get similarity values between records. Finally, clustering of these similarity values can detect duplicate records. The algorithm can automatically adjust the number of clusters by comparing the similarity value with a predefined similarity threshold. This algorithm shows good detection accuracy and high availability. Qiaoqiao Yang et al. [5] implemented the SNM algorithm based on some edit distances and variable windows to solve the shortcomings of the SNM algorithm. The algorithm proposed in this paper is based on the various edit distances and variable windows. The experiment's data set comes from the refrigeration industry management system. This proposed algorithm can efficiently recognize duplicate big data records. However, there is still the problem of improving the recall ratio and handling non-standard samples.

Jumoke Soyemi et al. [6] implemented a system for detecting duplicate records in a database using a simil matching algorithm. The Simil algorithm is based on calculating the similarity between two

strings. This proposed system can only be used to clean up data and prevent incorrect data from accessing the database.

3. Data Preparation

Data preparation is a necessary step in data cleanup before duplicate record detection process. The data preparation phase involves data parsing, data transformation, and standardization procedures. Data preparation techniques are also described in terms of ETL (extraction, transformation, loading) [7].

In the proposed system, the removing of XML tags in publication records is included in the data parsing procedure. Second, prior to the process of detecting duplicates, other data types are uniformly represented in standardization procedure. In this system, author name, date and title are standardized. Author names can be all authors participated in the publication. But only first author is extracted and formatted into first character of first name, dot (.) and last name only. The system extracts only year from date value and title must not be empty. Therefore, these preprocessed data fields can be easily used in key creation and detection.

4. Field Matching Techniques

Field Matching Technique is the inner stage of duplicate detection while the outer stage of duplicate detection is applied as the record matching technique. Duplicate detection relies on string comparison techniques to resolve typographic changes in the string data and errors in the numeric data.

Techniques for matching fields in the context of duplicate record detection include:

- Character-based similarity measurement
- Token-based similarity measurement
- Similarity measurement of pronunciation
- Numerical similarity measurement

4.1. Character-based Similarity Measurement

Character-based similarity metrics handle typographical errors well. Some similarity measures are:

- Edit distance
- Affine gap distance
- Smith-Waterman distance
- Jaro distance metric and
- Q-gram distance

In this proposed system, Edit Distance (or) Levenshtein Distance Algorithm is used to calculate field matching similarity scores.

4.1.1. Edit Distance (or) Levenshtein Distance Algorithm

Edit Distance, a.k.a. Levenshtein distance [8], is the minimum number of edit operations on a single character that is required to transform the string one into string two. Three types of edit operations are possible. They are:

(i) Insertion: insert a character into the string.
(ii) Deletion: delete a character from the string.
(iii) Substitution: replace a character with another character.

Levenshtein Distance Algorithm is described as below:

Algorithm 1: Levenshtein Distance Algorithm

```
levenshtein (source, target : STRING ) : INTEGER
-- Minimum number of edit operations to turn source into target
local
   distance                : ARRAY_2 [ INTEGER]
   i, j, del, ins, subs    : INTEGER
do
   create distance.make (source.count, target.count)
   from  i := 0  until  i > source.count  loop
      distance [ i, 0 ]  :=  i ;
      i  :=  i + 1;
end
   from  j  := 0  until   j > target.count  loop
      distance [ 0, j ]  :=  j ;
      j  :=  j + 1;
end
   from  i := 1 until  i  >  source.count  loop
     from  j  :=  1 until  j  >  target.count  invariant
     -- for  all  p :0 … i, q : 0 …j -1  ,  we can turn source  [ 1..p]
      -- into target [ 1..q]  in  distance [p,q]  operations
      loop
        if  source[i]  =  target [j]  then
           distance [i, j]  :=  distance [ i  - 1, j  - 1]
        else
          deletion :=  distance [ i - 1 , j ]
          insertion :=  distance [ i , j - 1 ]
          substitution :=  distance [ i - 1 , j - 1 ]
          distance[ i , j ]  := minimum (deletion, insertion, substitution) + 1
        end
        j  := j + 1
   end
   i  :=  i  +  1
end
Result  :=  distance (source.count , target.count)
end
```

5. Duplicate Records Detection

There are three basic techniques for detecting duplicate records. They are:

(i) Knowledge-based technology
(ii) Probabilistic technology
(iii) Empirical methods

The proposed system focuses on empirical algorithms [9] which consist of sorting, blocking and windowing methods such as Sorted Neighborhood Method, its extensions and Priority Queue Algorithm, etc. Sorted Neighborhood Method is the most representative for windowing.

5.1. Sorted Neighborhood Method (SNM)

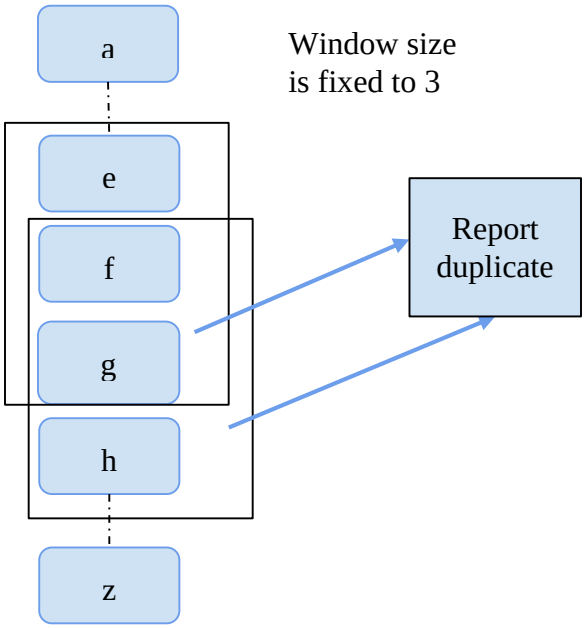


Figure 1: Sorted Neighborhood Method (SNM)

Using a sorted neighborhood approach can reduce the cost of comparing records and increase the efficiency of comparing records.

Hernáandez and Stolfo [10] describe the sorted neighborhood approach. This method involves three steps:

Step 1: Create sorting key: A key for each record in the dataset is assigned to each record. Keys are created by concatenating the values of two or more attributes.

Step 2: Sort the data: The records in the database are sorted based on the sorting key.

Step 3: Merge: A fixed size window is moved through the list of records sequentially to limit the comparison of records matching to those records in the window. Each new record that enters this window is compared to the previous record to find a "matching" record.

The Sorted Neighborhood Method uses fixed size windows. If the selection of window size is too small, some actual duplicate records may be lost and using larger window size will often result in unnecessary comparisons within the window.

The effectiveness of the sorted neighborhood approach depends greatly on the creation of sorting keys.

5.2. Duplicate Count Strategy

Duplicate Count Strategy is more than using fixed-size windows and provides windows that can be adjusted dynamically. The Duplicate Count Strategy is an improvement of the Sorted Neighborhood Method (SNM), and varies the size of the window depending on the number of identified duplicates.

5.2.1. Basic Duplicate Count Strategy (DCS)

Duplicate Count Strategy involves the following steps:

1. Sort the records by assigning a sorting key to each record.
2. Create a window with the initial window size w.
3. Compare the first record with all other records in the window.
4. Increase window size while $\frac{detected\ duplicates}{comparisons} \geq$ Ø
5. Slide the window (initial window size w)
6. Calculate transitive closure.

5.2.2. Enhancement of DCS: Duplicate Count Strategy-Multi Record Increase Approach (DCS++)

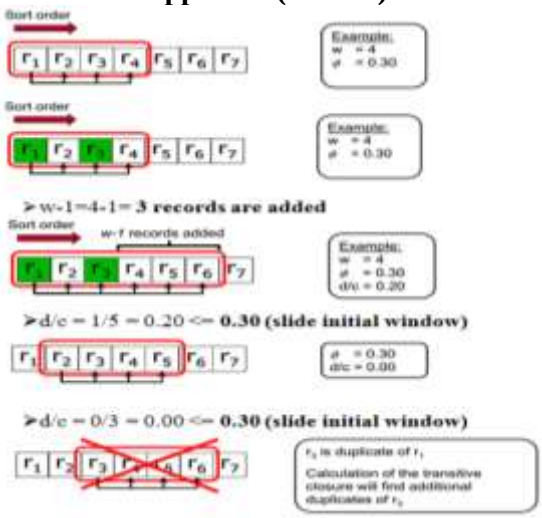


Figure 2: Duplicate Count Strategy ++(DCS++)

Idea 1: If each duplicate is found, the next w -1 records of this duplicate are added to the window.

Idea 2: Windows for duplicates have been omitted to save the comparisons.

- In **Figure 2**: Skip window for r3.
- Use the transitive closure to find additional duplicates.

- There is no loss in the window because the window for r1 covers all comparisons that r3 would have made.

6. Overview of the Proposed System

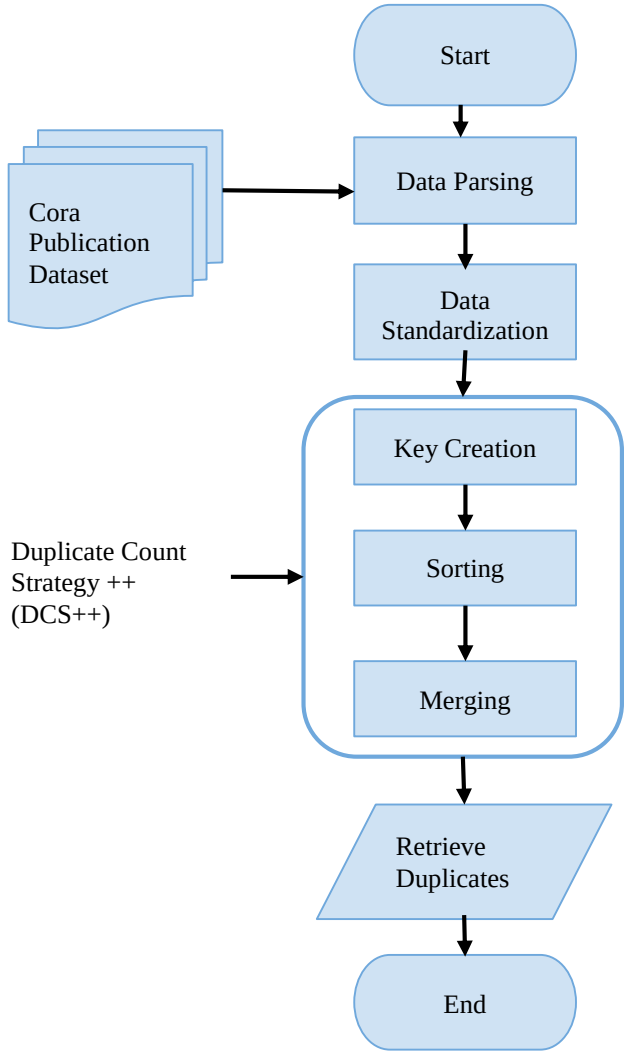


Figure 3: Overview of the proposed system

Figure 3 shows the overview of the proposed system. According to the figure, data preprocessing as data parsing and data standardization is performed to the input dataset. The proposed system can be classified by the two stages. In the first stage, field matching of each record is performed by using Levenshtein Distance Algorithm or Edit Distance Algorithm. In the second stage, duplicate record detection process is performed by using Duplicate Count Strategy-Multi Record Increase (DCS++) Algorithm and then duplicate records are detected.

6.1. Duplicate Count Strategy-Multi Record Increase Algorithm

In this experiment, the initial window size (w) is provided as 20 and DCS++ threshold (Ø) is recommended as $\frac{1}{w-1}$ not to miss any duplicates. The Duplicate Count Strategy-Multi Record Increase (DCS++) algorithm is described as below:

Algorithm 2. DCS++

```
1.  w: initial window size
2.  Ø: DCS++ threshold
3.  Require: w > 1 and 0 < Ø ≤ 1
4.  sort records by sorting key
5.  populate window win with first w records of
    records
6.  skipRecords ← null
7.  for j = 1 to records.length - 1 do
8.    if win[1] NOT IN skipRecords then
9.  numDuplicates ← 0
10. numComparisons ← 0
11. k ← 2
12.     while k ≤ win.length do
13.       if isDuplicate (win[1] , win[k] ) then
14.         emit duplicate pair (win[1] , win[k])
15.       skipRecords.add ( win[k] )
16.       numDuplicates ← numDuplicates + 1
17.       while win.length < k+w-1 and j + win.length
    < records.length do
18.         win.add (records [ j + win.length + 1] )
19.       end while
20.       end if
21.       numComparisons ← numComparisons+1
22.       if k = win.length and j + k < records.length
    and (numDuplicates / numComparisons ) ≥ Ø
    then
23.         win.add (records [j + k - 1])
24.       end if
25.       k ← k + 1
26.     end while
27.   end if
28.   win.remove(1)
29.   if win.length < w and j + k < records.length
    then
30.     win.add (records [j + k - 1])
31.   else
32.     while win.length > w do
33.       win.remove (win.length)
34.     end while
35.   end if
36.   j ← j + 1
37. end for
38. calculate transitive closure.
```

7. Performance Evaluation

In this system, the performance of the algorithm is measured using: **Recall**, **False Positive Error** (FP), **False Negative Error** (FN) and **Precision**.

1. **Recall**
   The percentage of duplicate records that the system correctly identifies. Recall percentage is computed by following equation:
   Recall = $\frac{no\ of\ identified\ duplicates}{no\ of\ actual\ duplicates} \times 100\%$ **(7.1)**

2. **False Positive Error (FP)**
   The percentage of records incorrectly identified as duplicates. FP percentage is defined as the equation:
   FP = $\frac{no\ of\ wrongly\ identified\ duplicates}{total\ number\ of\ identified\ duplicates} \times 100\%$ **(7.2)**

3. **False Negative Error (FN)**
   The percentage of duplicate records that the system does not detect. FN percentage is computed by following equation:
   FN = 100% − Recall **(7.3)**

4. **Precision**
   The percentage of information reported as relevant by the system that is correct.
   Precision percentage is defined as the equation:
   Precision = 100% − FP **(7.4)**

7.1. Experimental Environment

In order to evaluate the proposed algorithms, install XAMPP on this system. Apache service is started in XAMPP Control Panel as a local server.

The experimental environment is as follows:

- Operating System: Windows 8,
- CPU: Core i5@3 GHz,
- Memory: 8 GB

The software versions are as follows

- XAMPP version: 7.3.0 64 bit, and
- PHP version: 7.3.0

In the next sections, the Cora dataset and performance results with different thresholds are discussed.

7.2. Dataset Used

In this experiment, a Cora Dataset is used. This dataset contains bibliographic information for scientific papers. It provides 1,879 objects.

The Cora dataset is prepared by the original Andrew McCallum and his versions of this dataset are provided on his data web page. Many publications in record linkage and entity records over the years used these various versions of the Cora dataset.

```
<CORA>
<NEWREFERENCE id="1">
ahlskog1994a
<author>
M. Ahlskog, J. Paloheimo, H. Stubb, P. Dyreklev, M. Fahlman, O.
</author>
<title>Inganas and M.R.</title>
<journal>Andersson, J Appl. Phys.,</journal>
<volume>76,</volume>
<pages>893,</pages>
<date>(1994).</date>
</NEWREFERENCE>
    ...
</CORA>
```

Figure 4: Sample XML Cora Dataset

7.3. Performance Result

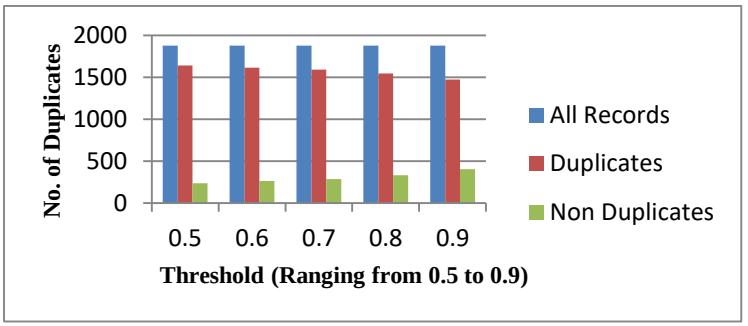


Figure 5: Execution Results of Duplicate Detection with window size 10.

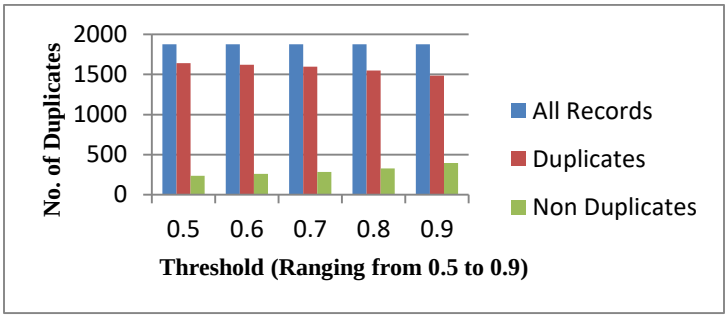


Figure 6: Execution Results of Duplicate Detection with window size 20.

The evaluation is performed by changing the threshold value from 0.5 to 0.9 at intervals of 0.1. This system is tested with different window sizes 10 and 20. **Figure 5** and **Figure 6** show the execution results of DSC++ with Edit Distance Algorithm for five threshold values.

Table 1 shows the performance evaluation results with window size 20 for five thresholds. The percentage of

recall and precision is high. Also, FP and FN is less in threshold values (0.5, 0.6, 0.7).

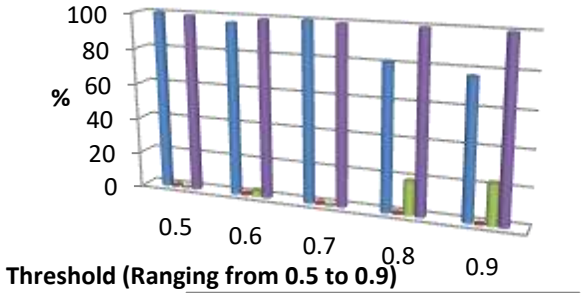
So, it determines that this system identified duplicate records being correctly in threshold values (0.5, 0.6, 0.7). Although precision and FP are good in threshold values (0.8 and 0.9), other percentage values of recall and FN are no good because the percentage of duplicate records being correctly identified by the system is less.

Table 1: Results of Performance Evaluation

DCS++ Algorithm with Edit Distance Algorithm (window size = 20)					
Sr. No	**Threshold**	**Recall%**	**FP%**	**FN%**	**Precision%**
1.	**0.5**	100	0.91	0	99.09
2.	**0.6**	96.55	0.85	3.45	99.15
3.	**0.7**	100	0.74	0	99.26
4.	**0.8**	80.95	0.58	19.05	99.42
5.	**0.9**	76.92	0.35	23.08	99.65

In **Figure 7**, the system assumes that the threshold values (0.5, 0.6 and 0.7) are better than other threshold values (0.8 and 0.9) for duplicate detection because these are less in the percentage of FP, FN and high in the percentage of precision and recall.

Among them, threshold value 0.7 is the best result for duplicate detection in this system.



	0.5	0.6	0.7	0.8	0.9
Recall %	100	96.55	100	80.95	76.92
FP%	0.91	0.85	0.74	0.58	0.35
FN%	0	3.45	0	19.05	23.08
Precision%	99.09	99.15	99.26	99.42	99.65

Figure 7: Performance Evaluation of Duplicate Detection

8. Conclusion and Future Work

The system used DCS ++ to detect duplicate records from the Cora publication xml dataset. This system has exceeded using a fixed window size.

As the result of performance evaluation threshold value 0.7 is the best result for duplicate detection of Cora dataset in this system. DCS ++ detects more duplicates by adding the next w-1 records of this duplicate to the window for each detected duplicate. Time is critical in data cleaning of a large database. Usage of duplicate detection DCS++ method is to reduce the time taken on each comparison by skipping windows for duplicates. There is some limitation in this system. This system can only be used in a homogeneous source dataset such as XML format because we need the key creation for some fields of the dependent domain. The future work is to detect duplicates in domain-independent data cleaning.

References:

[1].H. B. Newcombe, J. M. Kennedy, S. J. Axford, and A. P. James, *"Automatic linkage of vital records."* Science, vol. 130, pp. 954-99, 1959.

[2].ZM Guo, AY Zhou, *"Research on data quality and data cleaning: A survey[J]"*, Journal of Software, 2002, 13(11): 2076-2082.

[3].Draisbach, Uwe and Felix Naumann. *"A Comparison and Generalization of Blocking and Windowing Algorithms for Duplicate Detection."*, 2009.

[4] .P. Ying, X. Jungang, C. Zhiwang, and S. Jian, *"IKMC: An Improved K-Medoids Clustering Method for Near-Duplicated Records Detection"*, in Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on, Wuhan, 2009, pp. 1 -4.

[5].Q. Yang,* Z. Guo, K. Wang, *"The SNM Algorithm Based on a Variety of Edit Distance and Variable Window"*, The 7th International Conference on Computer Engineering and Networks, 2017, CENet2017.

[6].Jumoke Soyemi, James Adegboye, *"Database Record Duplicate Detection System using Simil Algorithm,"* International Journal on Computer Science and Engineering (IJCSE), 2018, vol 10.

[7].Kimball, Ralph; Joe Caserta, *"The data warehouse ETL toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data"*, John Wiley & Sons, 2004.

[8].*Levenshtein V I. Binary codes capable of correcting deletions, insertions and reversals[J]*. Soviet Physics Doklady, 1966, 10(1):707-710.

[9].M. Rehman and V. Esichaikul, "*Duplicate Record Detection For Database Cleansing*," in

Machine Vision, 2009. ICMV '09. Second International

Conference on , Dubai, 2009 , pp. 333 - 338.

[10].Hernández, Mauricio Antonio; Salvatore Joseph Stolfo (jan 1998). *"Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem"*.Data Mining and Knowledge Discovery 2 (1): 9--37.