ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 8, August 2016

A Comparison between Two Variants of Sorted Neighborhood Method

Manju V J, Chinchu Krishna S

M.Tech Student, Dept. of I.T, Rajagiri School of Engineering and Technology, Kochi, India

Assistant Professor, Dept. of I.T, Rajagiri School of Engineering and Technology, Kochi, India

ABSTRACT: As an integral part of data cleansing, Duplicate detection is the process of finding multiple records that represent the same real-world entity in a dataset. Databases contains very large dataset and the duplicate detection needs to complete its process in very shorter time. The windowing methods, in particular Sorted Neighbourhood Method (SNM) is one of the most popular method used for duplicate detection. In recent years, a variety of improvements of SNM have been proposed. One among them is Progressive Sorted Neighborhood Method. One challenge that has emerged is that it is not straightforward to differentiate between the multiple variants of SNM existing today. Thus, in this paper, a comparison is made between the traditional SNM , its adaptive approach named DCS++ and the progressive SNM. The purpose of this comparative study is to show the difference between these methods in terms of their performance. The evaluation result shows that the progressive SNM finds most of the duplicates earlier than both the traditional Sorted Neighborhood and Adaptive DCS++ method.

KEYWORDS: Duplicate Detection; Sorted Neighborhood Method; Data Cleaning

I. Introduction

Data is one of the most important resources in a company. It represents the whole business knowledge about products, customers, suppliers and transactions and forms the main source of information. Since data changes in the course of daily business, faults occur and information invalidate. Steve Sarsfield emphasizes the growing of a company and its data volume as a major reason for quality issues. The insertion of new data items always bears the risk of producing errors and existing data items might become obsolete in the ever changing business environment. Redundant data management and the integration of different data sources may as well result in faulty data pools. Ignoring these errors forces a company to base its strategic decisions on incorrect information. The consequences are avoidable costs and competitive disadvantages.

Duplicate Detection, as an integral part of data cleansing, focuses on finding different representations of the same real world entity by comparing multiple records. A Duplicate Detection process is very costly due to extremely large data sets and compute-intensive record comparisons. At the same time, it may be very important to run duplicate detection within a limited amount of time.

In this paper, a Progressive Sorted Neighborhood Method (PSNM) is discussed which is a pay-as-you-go approach that identifies most duplicate pairs early in the process. Instead of limiting the overall time required to finish the whole duplicate detection process, progressive approaches try to limit the average time after which an arbitrary duplicate is found. A conservative duplicate detection algorithm collects all duplicates internally and emits them in the end. On the contrary, incremental algorithms already report intermediate results at execution time. Hence, they deliver first duplicates right from the start.

In this approach, partial results are gradually obtained as the duplicate detection process is in progress, so we can get at least some results faster. The partial results need not have to find all the records that belongs to the same real-world entity. The goal is to obtain as much of the overall result as quickly as possible [2]. The aim of progressive duplicate detection is to identify most matches early in the process. Therefore, the PSNM increase the efficiency of the duplicate detection and not the quality of the reported results. Traditionally, blocking and windowing techniques are used to decrease the complexity of the duplicate detection process . Now, one may argue that neither blocking nor windowing approaches may find all duplicates in a corpus. Therefore, terminating a progressive algorithm prematurely might be reasonable to find slightly less duplicates in a much shorter time. Early termination, in particular, yields more results on a progressive algorithm than on any traditional approach. Hence, the progressive approaches increase the efficiency especially for use case scenarios that do not need the complete result. A real world entity is often described by more than two records. A set of records that refers to the same real world entity is called a duplicate cluster, because all record pairs within this set are duplicates. By sorting or grouping the input records, the duplicate detection algorithm moves the records of a cluster closer together. Afterwards, the input data exhibits regions of higher and lower duplicate density. Using this observation, adaptive windowing techniques [5] dynamically adjust their window sizes depending on the amount of currently found duplicates. Duplicate Count Strategy++ (DCS++) is one such adaptive technique that can dynamically improve the efficiency of duplicate detection, but in contrast to the progressive techniques, they need to run for certain periods of time and cannot maximize the efficiency for any given time slot.

II. Variants of Sorted Neighbourhood Method

Extensive research has also been done on pair selection algorithms that try to maximize recall on the one hand and efficiency on the other hand [3]. The Sorted Neighbourhood Method is the most prominent pair selection approach that significantly reduced the complexity of the duplicate detection process. In this method, it first sort the records in the data list using a sorting key and move a fixed size window through the sequential list of records limiting the comparisons for matching records to those records in the window. If the size of the window is w records, then every new record entering the window is compared with the previous w - 1 records to find matching records. The first record in the window slides out of the window. It applies a fixed sized sliding window on a sorted set of records to select comparison candidates [6].

A. *Progressive SNM Algorithm*

Progressive SNM is based on traditional sorted neighborhood method. Progressive SNM [1] first sorts the data using a predefined sorting key with an intuition that records that are close in the sorted order are more likely to be duplicates than records that are far apart, because they are already similar with respect to their sorting key. The Progressive method then assumes that the distance of two records in their rank after sorting defines the likelihood of been a duplicate. The PSNM algorithm exploits this assumption by comparing records with their direct neighbors first. Then, the algorithm successively increases the distance between the comparison candidates. Since the algorithm is based on the Sorted Neighborhood Method, it practically starts sliding a window of size two over the sorted dataset in the first iteration. In a second iteration, the window size is three, in the third iteration four and so on until the final window size is reached. Thereby, each iteration only executes those record comparisons that have not been executed in a previous iteration. Using the LookAhead function, the PSNM algorithm dynamically changes the execution order of the comparisons.

Algorithm 1 shows the implementation of Progressive SNM. The algorithm mainly takes 5 input parameters: DS is a reference to the data set. The parameter maxWindow specifies the maximum window size and Key stands for the sorting key that defines the attribute which is to be used in the sorting step. The progressive strategy discussed here assume that all keys are single attribute keys. In principal, a key can be any combination of attributes. However, we focus on single attribute keys, because of simplicity reasons. Parameter Interval defines the enlargement interval of the iterations. In other words, the Interval defines how many window distance dist iterations should be executed on each loaded partition. For instance, if the parameter Interval is 3 then the algorithm loads the first partition to execute dist 1 to 3 sequentially, then it loads the second partition to execute the same interval of dist and so on until all the partitions have been loaded once. Afterwards, all partitions are loaded again to run dist 4 to 6 and so forth. This strategy reduces the number of load processes. The last parameter Rnum stands for the number of records in the data set.

As the entire dataset will not fit in main memory, PSNM operates on a partition of the dataset at a time. The set of records that the algorithm will load within the partition is calculated in Line 2. Thus partition size is the maximum number of records that fit in the memory. Line 3 defines an order array which stores the order of records with respect to the given sorting key key. In line 4, PSNM sorts the dataset DS by the sorting key key. Afterwards, PSNM linearly increases the size of the window from minimum of 2 to the maximum window size maxWindow in steps of Interval. Thus the PSNM algorithm selects promising close neighbors first and less promising far-away neighbors later on. PSNM reads the entire dataset once, for each of these progressive iterations. Since the load process is performed partition-wise, PSNM sequentially iterates (Line 6) and loads (Line 7) all partitions.

Algorithm 1 Progressive Sorted Neighborhood Method

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 8, August 2016

```
Step 1:  procedure PSNM(DS, maxWindow, Key, Interval, Rnum)
Step 2:      Partitions ← CalcPartitions(DS)
Step 3:      array order size Rnum as integer
Step 4:      order ← OrderBy(Key, DS)
Step 5:      for I ← 2 to ceil(maxWindow/Interval) do
Step 6:          for Partition € Partitions do
Step 7:              rec ← loadPartition(DS, Partition)
Step 8:              for dist € range( I,Interval,maxWindow) do
Step 9:                  for index ← 0 to |rec|- dist do
Step 10:                     dpair ← < rec[index], rec[index+dist]>
Step 11:                     if compare(dpair) then
Step 12:                     emit(dpair)
Step 13:                     lookAhead(dpair)
Step 14:                 end for
Step 15:             end for
Step 16:         end for
Step 17:     end for
Step 17: end procedure
```

To process a loaded partition, PSNM first iterates overall record rank distances dist that are within the current window interval I. In Line 9, PSNM then iterates all records in the current partition to compare them to their dist-neighbor. The comparison is executed using the compare(dpair) function in Line 11. If the compare function returns a true value, then a duplicate has been found and can be emitted. Then PSNM invokes the lookAhead(dpair) function, for progressively searching more number of duplicates in the neighborhood of the current duplicate.

It is observed that most of the datasets contain many large duplicate clusters. This observation is used to find good comparison candidates earlier by progressively searching for duplicate clusters. Since the pair selection algorithm cannot estimate potential clusters from the outset, the ranking of comparison candidates needs to be optimized at runtime. Therefore, the duplicate detection algorithm can use the information about previously found duplicates to identify further promising record pairs. Hence, the algorithm uses a lookAhead strategy to dynamically adjust the comparison order based on intermediate results. As the records within the same cluster are very close to each other in the record order, if (i, j) has been identified as a duplicate, then the record pairs (i+1, j) and (i, j+1) have a high chance of being duplicates of the same cluster as well. If a duplicate (i; j) has been detected in Line 11, the optimized PSNM algorithm additionally calls the LookAhead(dpair) function. This function schedules the comparison of the record pairs (i+1; j) and (i; j+1) and immediately emits all duplicates that have been found in this way. Furthermore, if any of the look-ahead comparisons finds another duplicate, an extra look-ahead is recursively executed. By recursively iterating larger neighborhoods around duplicates, a complete cluster can be found at once. To avoid comparing records redundantly in different lookAheads or in a following iterations, the algorithm stores all comparisons, which have been handled within a look-ahead, in a temporary data structure. Since the look-ahead is a recursive function, it may schedule comparisons that are beyond the given maxWindow. Therefore, the maximum look-ahead distance is limited to maxWindow

If the user do not terminate the detection process earlier, PSNM finishes when all intervals are processed and the maximum window size maxWindow is reached.

B. *Duplicate Count Strategy++ (DCS++)*

The major flaw of SNM is the use of fixed size window. If a window of smaller size is used, it may omit some real duplicates and if a window of larger size is used, it may perform many unnecessary comparisons. Duplicate Count Strategy (DCS++) overcome the use of fixed size window and comes out with windows that can be adapted dynamically. The Duplicate Count Strategy++ is an improvement of Sorted Neighborhood Method (SNM) and varies the size of the window based on the number of identified duplicates. As the window size is increased or decreased depending on the number of duplicates identified, the set of records that is being compared differs from the original SNM. The order in which the records are compared is also different from both the traditional SNM and PSNM.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 8, August 2016

Adapting the window size not only result in additional comparisons but also reduce the number of comparisons. However, adapting the window size should result in an overall higher effectiveness for a given efficiency or in a higher efficiency for a given effectiveness. Duplicate Count Strategy-Multi record increase (DCS++) uses the number of already detected duplicates as a signal for the window size [4]. If more duplicates of a record are found within its current window, the window is made larger. On the other hand, if no duplicate of a record is found within its window, then it assumes that no duplicates are there or the duplicates are very far away in the sorting order and works like SNM. In DCS++, records are sorted based on the sorting key. Then a window is created with initial window size w. The first record is compared with all other records in the current window. The window size is increased while the number of duplicates detected is greater than a threshold value. That is, for each detected duplicate, DCS++ add the next w-1 records of that duplicate to the window. Then the window slides with the initial window size w. To save the no of comparisons, DCS++ skip windows for the duplicates detected.

III. EVALUATION AND PERFORMANCE COMPARISON

The experiment has been done using the reduced real world dataset, CD-dataset containing 500 records. The CD-dataset contains different records of audio CDs and music . The experiment uses a window of size 20 for SNM, DCS++, and PSNM. The Progressive SNM is compared with an adaptive method called Duplicate count strategy++ (DCS++) and traditional SNM and the experiment shows Progressive Sorted Neighborhood method reports most duplicates much earlier than traditional SNM and Adaptive DCS++.

Time taken to find the most duplicates in Progressive Sorted Neighborhood Method is less than that of both the traditional SNM and adaptive DCS++. The results also shows that the adaptive approach can find more duplicates than both the progressive and traditional approach. This is because adaptive approach can dynamically vary the window size based on the number of identified duplicates. The results of the experiments are showed in fig.1. The X-axis indicate the time taken and Yaxis indicate the number of duplicates found by each method. In comparison to traditional SNM and its adaptive approach DCS++, progressive SNM satisfies improved early quality. For any arbitrary target time t at which results are needed, the output of a progressive SNM algorithm will be larger than the output of its corresponding traditional SNM algorithm. Target time t is typically smaller than the overall runtime of the traditional algorithm. PSNM also satisfies same eventual quality when compared to traditional SNM. Without early termination at t, if both a traditional algorithm and its progressive version finish their execution, they produce the same results. But when the adaptive version is also considered, it can find more results than the other two methods.

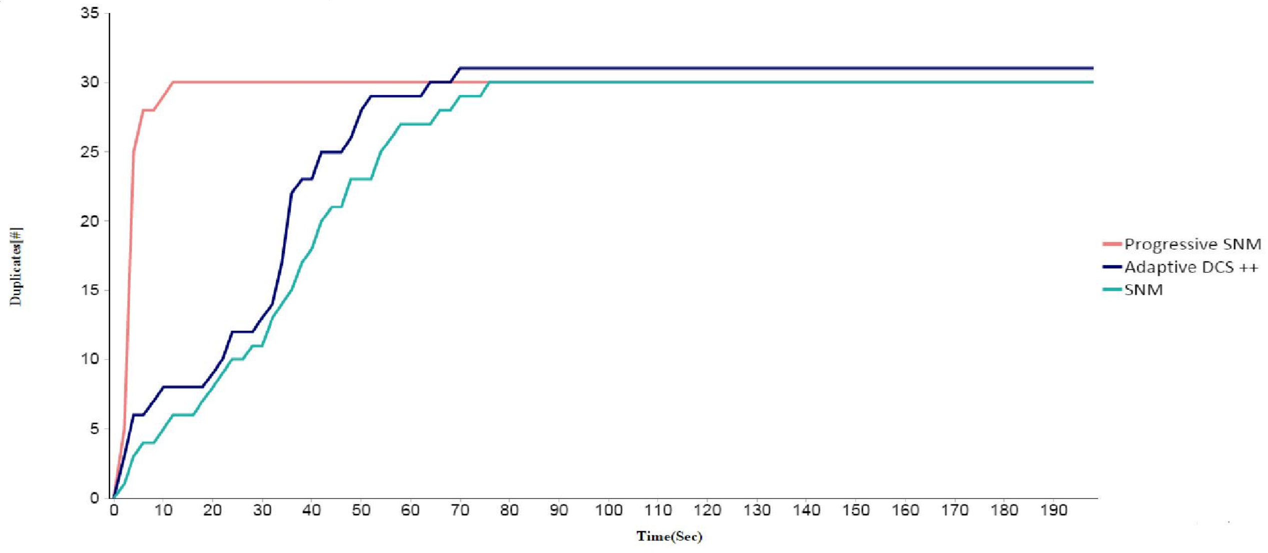

Fig.1. Performance Comparison of PSNM, Adaptive DCS++ and SNM

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 8, August 2016

IV. CONCLUSION AND FUTURE WORK

As the data set is increasing in a large volume, the need for an efficient duplicate detection algorithms becoming more and more important. In this paper we have experimented and compared traditional Sorted Neighborhood Method and two of its extensions named PSNM and DCS++. The experiments shows that PSNM finds most duplicates much earlier than both the adaptive DCS++ as well as the traditional SNM. The PSNM algorithm improves the efficiency of duplicate detection task for situations with reduced execution time. It can reduce the average time after which an arbitrary duplicate is found and thereby detecting duplicates earlier. It also shows DCS++ finds more results than PSNM and SNM as DCS++ adapts the window size based on the number of detected duplicates.

REFERENCES

1. T. Papenbrock, A. Heise, and F. Naumann, Progressive Duplicate Detection, IEEE Trans. Knowl. Data Eng., vol. 27, no. 5, pp. 13161329, May 2015.
2. S. E. Whang, D. Marmaros, and H. Garcia-Molina, Pay-as-you-go entity resolution, IEEE Trans. Knowl. Data Eng., vol. 25, no. 5, pp. 11111124, May 2012.
3. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, Duplicate record detection: A survey, IEEE Trans. Knowl. Data Eng., vol. 19, no. 1, pp. 116, Jan. 2007.
4. U. Draisbach, F. Naumann, S. Szott, and O. Wonneberg, Adaptive windows for duplicate detection, in Proc. IEEE 28th Int. Conf. Data Eng., pp. 10731083, 2012.
5. S. Yan, D. Lee, M.-Y. Kan, and L. C. Giles, Adaptive sorted neighborhood methods for efficient record linkage, in Proc. 7th ACM/ IEEE Joint Int. Conf. Digit. Libraries, pp. 185194, 2007.
6. M. A. Hernandez and S. J. Stolfo, Real-world data is dirty: Data cleansing and the merge/purge problem, Data Mining Knowl. Discovery, vol. 2, no. 1, pp. 937, 1998.
7. U. Draisbach and F. Naumann, A generalization of blocking and windowing algorithms for duplicate detection, in Proc. Int. Conf. Data Knowl. Eng., pp. 1824, 2011.
8. Cesario, Eugenio, et al. "An incremental clustering scheme for duplicate detection in large databases." 9th International Database Engineering & Application Symposium (IDEAS'05). IEEE, 2005.
9. J. Madhavan, S. R. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy, “Web-scale data integration: You can only afford to pay as you go,” in Proc. Conf. Innovative Data Syst. Res., 2007.
10. P. Christen, “A survey of indexing techniques for scalable record linkage and deduplication,” IEEE Trans. Knowl. Data Eng., vol. 24, no. 9, pp. 1537–1555, Sep. 2012.
11. C. Xiao, W. Wang, X. Lin, and H. Shang, “Top-k set similarity joins,” in Proc. IEEE Int. Conf. Data Eng., pp. 916–927, 2009.
12. O. Hassanzadeh and R. J. Miller, “Creating probabilistic databases from duplicated data,” VLDB J., vol. 18, no. 5, pp. 1141–1166, 2009.