ISSN No: 2454-9614

Parallel and Multiple Distributed Process for Progressive Duplicate Detection Model

S. Nivetha, M. Mohanasundari*

Department of Computer Science and Engineering,Vellalar College of Engineering and Technology
Erode,Tamilnadu, India.

*Corresponding Author: S. Nivetha

E-mail: vsnivi2010@gmail.com

Received: 10/11/2015, Revised: 12/12/2015 and Accepted: 05/03/2016

Abstract

Duplicate detection is the process of identifying duplicate records in a system. This paper proposes the minimization of average execution time in duplicate detection methods. Detection algorithms used are: 1) Progressive sorted neighborhood method 2) Progressive blocking algorithms. The aim of this paper is to process larger datasetin ever shorter time. A progressive duplicate detection algorithm improves the efficiency of finding duplicates in the shorter execution time. It improves the efficiency on detection in Parallel approach. It finds maximum number of duplicates in a short time periods and reduces average time. The execution time is minimized by sharing the resources in different location of memory.

Reviewed by* **ICETSET'16 *organizing committee*

1. Introduction

Data mining, or knowledge discovery, is the computer-assisted process of digging through and analyzing enormous sets of data and then extracting the meaning of the data. Data mining tools predict behaviors and future trends, allowing businesses to make proactive, knowledge-driven decisions.

Data mining tools are traditionally time consuming to resolve. They scour data bases for hidden patterns, finding predictive information that experts may miss because it lies outside their expectations. Data mining derives its name from the similarities between searching for valuable information in a large database. Although data mining is still in its infancy, companies in a wide range of industries including retail, finance, health care, manufacturing transportation, and aerospace are already using data mining tools andtechniques to take advantage of historical data. By using pattern recognition technologies and statistical and mathematical techniques to sift through warehoused information, data mining helps analysts recognize significant facts, relationships, trends, patterns, exceptions and anomalies that might otherwise go unnoticed. For businesses, data mining is used to discover patterns and relationships in the data in order to help make better business decisions.

Data are among the most important assets of a company. But due to data changes and sloppy data entry, errors such as duplicate entries might occur, making data cleansing and in particular duplicate detection indispensable. However, the pure sizes of present datasets render duplicate detection processes expensive. As independent persons change the product portfolio, duplicates arise. Although there is an obvious need for de-duplication, online shops without downtime cannot afford traditional de-duplication.

2. Related Work

Papenbrock, Heiseand Neumann **[1]** presents two novel approaches, progressive duplicate detection algorithms that significantly increase the efficiency of finding duplicates if the execution time is limited. They maximize the gain of the overall process within the time available by reporting most results much earlier than traditional approaches. Comprehensive experiments show that progressive algorithms can double the efficiency over time of traditional duplicate detection and significantly improve upon related work.

Progressive duplicate detection identifies most duplicate pairs early in the detection process.Instead of reducing the overall time needed to finish the entire process, progressive approaches try to reduce the average time after which a duplicate is found. Early terminations, in particular, then yields more complete results on a progressive algorithm than on any traditional approach.

Whang et.al. **[2]** Proposed a kind of progressive duplicate detection algorithm.

Entity resolution is the problem of identifying which records in a database refers to the same entity. In practice, many applications need to resolve large data sets efficiently, but do not require the ER result to be exact. For example, people data from the Web may simply be too large to completely resolve with a reasonable amount of work. As an example, real-time applications may not be able to tolerate any ER processing that takes longer than a certain amount of time.

This investigates how we can maximize the progress of ER with a limited amount of work using "hints," which give information on records that are likely to refer to the same real-world entity. A hint can be represented in various formats. It introduces techniques for constructing hints efficiently and techniques for using the hints to maximize the number of matching records identified using a limited amount of work. By using real data sets, the potential gains Pay-as-you-go approach compared to running ER without using hints.

Here the novel concept of hints, this can guide an ER algorithm to focus on resolving the more likely matching, records. Our techniques are effective when there are either too many records to resolve within a reasonable amount of time or when there is a time limit. Three types of hints that are compatible with different ER algorithms: a sorted list of record pairs, a hierarchy of record partitions, and an ordered list of records. In this work evaluated the overhead of constructing hints as well as the runtime benefits for using hints. The results suggest that the benefits of using hints can be well worth the overhead required for constructingand using hints and a general guidance for constructing and updating the "best" hint for any given ER algorithm.

3. Progressive SNM

Progressive sorted neighbourhood method sorts the input data using a pre-defined sorting key and only compares records that are within a window of records in the sorted order. The intuition is 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. More specifically, the distance of two records in their sort ranks (rank-distance) gives PSNM an estimate of their matching likelihood. The PSNM algorithm uses this intuition to iteratively vary the window size, starting with a small window of size two that quickly finds the most promising records. This algorithm is used for small datasets.

The PSNM algorithm calculates an appropriate partition size p Size, i.e., the maximum number of records that fit in memory, using the sampling function calc Partition Size (D).If the data is read from a database, the function can calculate the size of a record from the data types and match this to the available main memory. Otherwise, it takes a sample of records and estimates the size of a record with the largest values for each field. The algorithm calculates the number of necessary partitions pNum, while considering a partition overlap of W - 1 record to slide the window across their boundaries. The order-array stores the order of records with regard to the given key K. By storing only record IDs in this array; we assume that it can be kept in memory. To hold the actual records of a current partition, PSNM declares the record array.

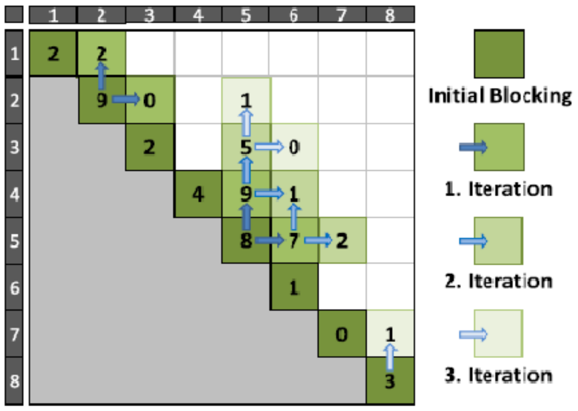
PSNM sorts the dataset D by key K. The sorting is done using progressive sorting algorithm. Afterwards, PSNM linearly increases the window size from 2 to the maximum window size W in steps of me. In this way, promising close neighbours are selected first and less promising far-away neighbours later on. For each of these progressive iterations, PSNM reads the entire dataset once. Since the load process is done partition-wise, PSNM sequentially iterates and loads all partitions.

To process a loaded partition, PSNM first iterates overall record rank-distances dist that are within the current window interval current. For I= 1 this is only one distance, namely the record rank-distance of the current main-iteration. PSNM then iterates all records in the current partition to compare them to their dist-neighbor. The comparison is executed using the compare (pair) function. Ifthis function returns “true”, a duplicate has been found and can be emitted.

4. Progressive Blocking

Progressive blocking is a novel approach that builds upon an equidistant blocking technique and the successive enlargement of blocks. Like PSNM, it also pre sorts the records to use their rank-distance in this sorting for similarity estimation. This algorithm is used for large and very dirty datasets.

It accepts five input parameters: dataset references D, key attribute K, maximum block range R, block size S and record number N. At first, PB calculates the number of records per partition p Size by using a pessimistic sampling function.



PB comparison

The algorithm also calculates the number of loadable blocks per partition b Per P, the total number of blocks bNum, and the total number of partitions pNum. PB then defines the three main data structures: the order-array, which stores the ordered list of record IDs, the blocks-array, which holds the current partition of blocked records, and the b Pairs-list, which stores all recently evaluated block pairs. Thereby, a block pair is represented as a triple of (blockNr1, blockNr2, duplicates Per Comparison).The b Pairs-list is implemented as a priority queue, because the algorithm frequently reads the top elements from this list.

The PB algorithm sorts the dataset using the progressive Magpie Sort algorithm. Afterwards, then it loads all blocks partition-wise from disk to execute the comparisons within each block. After the pre-processing, the PB algorithm starts progressively extending the most promising block pairs.

5. Attribute Parallel Method

The best key for finding the duplicate is generally hard to identify. Selecting good keys will increase the progressiveness. Multi-pass execution can be applied for progressive SNM. Key separation is not needed in PB algorithm.

Here all the records are taken and checked as a parallel processes in order to reduce average execution time. The overall records are kept in multiple resources after splitting. The intermediate duplication results are intimated immediately after found in any resources and returned to the main application. So the time consumption is reduced. Resource consumption is same as existing system but the data is kept in multiple resource memories.

6. Conclusion

Through this project, the efficiency of duplication detection is increased. This project introduced the progressive sorted neighbourhood method and progressive blocking. Both algorithms increase the efficiency of duplicate detection for situations with limited execution time; they dynamically change the ranking of comparison candidates based on intermediate results to execute promising comparisons first and less promising later. The project proposed a quality measure for progressiveness that integrates with existing measures. It uses multiple sort keys concurrently to interleave their progressive iterations. By analyzing intermediate results, both approaches dynamically rank the different sort keys at runtime, drastically easing the key selection problem. Almost all the system objectives that have been planned at the commencements of the software development have been net with and the implementation process of the project is completed. A trial run of the system has been made and is giving good results the procedures for processing is simple and regular order. The process of preparing plans been missed out which might be considered for further modification of the application.

References

[1] Papenbrock T., Heise A. and Naumann F. (2015), 'Progressive duplicate detection', Proc. IEEE Trans. Know. Data Eng., vol. 27, No. 5, pp. 1316-1329.

[2] Whang S. E., Marmaros D., Molina H. (2012),'Pay-as-you-go entity resoln', IEEE Trans. Know. Data Eng., vol. No 25.5, pp. 1111–1124.

[3] Draisbach U, Naumann F, Szott S, Wonneberg O. (2012),'Adaptive windows for duplicate detection', Proc. IEEE 28th Int. Conf. Data Eng., pp. 1073-1083.

[4] Draisbach U. and Naumann F. (2011), 'A generalization of blocking and windowingalgorithms for duplicate detection',Proc. Int. Conf. Data Knowl. Eng., pp. 18-24.

[5] Hassanzadeh O., Chiang F., Lee H.C., Miller R. J. (2009), 'Framework for Evaluating Clustering Algorithms in Duplicate Detection',Proc. Very Large Databases Endowment, Vol. 2, pp.1282-1293.

[6] Jeffery S.R., Franklin M.J., and Halevy A.Y. (2008), 'Pay-as-you-go user feedback for data space systems', Proc. Int. Conf. Manage. Data, pp. 847–860.

[7] Wallace M. andKollias S. (2008), 'Computationally Efficient Incremental Transitive Closure of Sparse Fuzzy Binary Relations ', Proc. IEEE Trans. Conf. Fuzzy Systems, Vol. 3, pp. 1561-1565.

[8] Elmagarmid A.K., Ipeirotis P.G., and Verykios V.S. (2007), 'Duplicate record detection: A survey', IEEE Trans. Know. Data Eng., Vol. 19, No. 1, pp. 1–16.

[9] Madhavan J., Jeffery S.R., Cohen S., Dong X., Ko D., Yu C. and Halevy A. (2007), ' Web-scale data integration: You can only afford to pay as you go' , Proc. Conf. Innovative Data Syst. Res, pp. 342-350.