International Journal on Advanced Science Engineering Information Technology

Vol.6 (2016) No. 6
ISSN: 2088-5334

Record Duplication Detection in Database: A Review

Saleh Rehiel Alenazi[#], Kamsuriah[*]

[#]*Research Center for Software Technology and Management, Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi, Selangor, 43600, Malaysia*
E-mail: sala207@hotmail.com, kamsuriah@ukm.edu.my

***Abstract*— The recognition of similar entities in databases has gained substantial attention in many application areas. Despite several techniques proposed to recognize and locate duplication of database records, there is a dearth of studies available which rate the effectiveness of the diverse techniques used for duplicate record detection. The calculating time complexity of the proposed methods reveals their performance rating. The time complexity calculation showed that the efficiency of these methods improved when blocking and windowing are applied. Some domain-specific methods train systems to optimize results and improve efficiency and scalability, but they are prone to errors. Most of the existing methods fail to either discuss or lack thoroughness in consideration of scalability. The process of sorting and searching form an essential part of duplication detection, but they are time-consuming. Therefore this paper proposes the possibility of eliminating the sorting process by utilization of tree structure to improve the record duplication detection. This has added benefits of reducing the time required and offers a probable increase in scalability. For a database system, scalability is an inherent feature of any proposed solution, due to the fact that the data size is huge. Improving the efficiency in identifying duplicate records in databases is an essential step for data cleaning and data integration methods. This paper reveals that the current proposed methods lack in providing solutions that are scalable, high accurate, and reduce the processing time during detecting duplication of records in the database. The ability to provide solutions to this problem will improve the quality of data that are used for the decision-making process.**

***Keywords*— duplication detection algorithm, windowing, blocking, scalability, efficiency, data quality**

I. INTRODUCTION

Among the entity matching applications, one of particular importance is an identification of similar database records [31]. Recognizing and locating duplicate records is a process that involves probing databases for purposes of identifying records that, in the real world, represent an identical entity. Locating such duplications is impeded by the fact that they do not share a common key. One means of resolving this involves weeding out errors in the data, such as the misspelling of names; the lack of integrity restraints limiting human entries to ranges not much higher than 100 years; and recognizing varying conventions used for entering the same information. For example entering simple numerals rather than spelling numbers out [1].

The pre-processing stage is amongst the most crucial phases since that is where a database is scanned to detect and remove duplicate records. Redundancy in records often results from merging databases. While this can initially create undesirable duplicates, the combination of duplicate records can ultimately result in enriched data combining details listed in one of two original entries but not in both, thus providing more detailed data to be mined and analyzed for research [1].

Heterogeneity arises when data from several sources are combined and integrated. These are of two types, namely structural heterogeneity and lexical heterogeneity. Structural heterogeneity involves different means of separating fields for the same information, such as breaking up an address into two or three lines instead of listing all the data on a single line. Lexical heterogeneity involves different means of encoding the same information, such as spelling out a numerical street name rather than writing it in numerals; or using initials to abbreviate a person's middle name or even a first name [1], [2].

This paper discusses issues related to scalability and efficiency. These two factors are very important assessment criteria for duplication detection techniques. Since detection methods work with huge databases, scanning them can consume considerable time. Therefore, use of techniques that both work efficiently with such massive databases within acceptable time limits, and that are scalable according to the size of the database, are essential.

Previous studies used blocking or windowing as a means of finding duplication. However, such operations are time-consuming and lead to trade-offs about the size of window or block. In order to lessen the work involved in sorting, a tree structure implementation ensures data is already sorted.

Another challenge is a discovery of juxtaposed duplicated records, which itself demands further research. Previous studies have failed to concentrate sufficiently on scalability and efficiency: few have proposed solutions for those issues [2]. This paper aims to review the existing techniques used in detecting duplicate records. To discuss the issues, this paper is organized as follows: section II examines the main differences between the two famous "divide and conquers" techniques (blocking and windowing) as a means of finding duplication. Section III discusses the common similarity measurements that have been employed to detect duplication within database records. Section IV describes the main detection tools that are commonly used in many articles reviewed in this paper. Section V considers the issues related to efficiency and scalability of the currently proposed duplication detection techniques. Finally, Section VI concludes the paper.

II. Materials And Methods

Blocking and windowing are two core operations in record duplication detection. This section will discuss these two techniques and the process involves in detecting the similarities between records. These operations will be discussed in more details in section A and section B respectively. In blocking operation, records are partitioned into a disjoint partition, which means no record can be present in more than one partition. Then records will be compared within the block to detect any similarities. A blocking operation can be divided into two steps: The first step is building, which involves reading all record sets from a given data set, creating records, indexing them in the memory, and using that information to create blocking keys, such as by using an inverted index [1], [5]. Parameters are derived from the blocking keys for the inverted index. The inverted index list in question then receives all of the records containing an identical blocking-key value. The second step is retrieving, which involves retrieving record identifiers from the index one block at a time. Records in a given block are paired together in turns, and a classifier compares the numerical values of the vectors resulting from each pairing [1].

A. Blocking

Domain and duplication experts normally determine the blocking keys manually and normally choose general keys to optimizing the resulting quality. During this process of record comparison and matching, they reduce the required quantity of data [1]. Another means of reducing the number of comparisons is the Standard Blocking technique, which seeks identical blocking key values (BKVs) within which the record pairs are compared, as seen in Fig. 1. The proportion of comparisons is (N2/B), where N is the number of dataset records and B is the number of same-size blocks [1, 6].

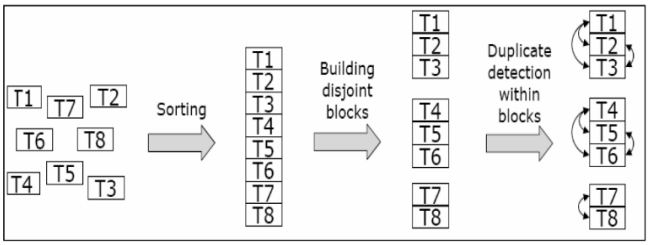

Fig. 1 Conventional blocking technique [7]

The number of blocks and their size are determined by the blocking-key selection, which is what the blocking technique relies on [1]. Choosing the optimal key is based on the optimal means of bringing duplicate candidates into the same block. Blocking can increase the data-comparison speed: but by bringing together records that should not be compared, it also has the potential to increase the number of false mismatches.

B. Windowing (Sorted Neighbourhood)

Hernández and Stolfo [8] developed the Sorted Neighbourhood (SN) technique, otherwise known as Windowing. By imposing limitations on the measures of similarity in the dataset, the number of potential matches is minimized. The following three steps outline the Windowing technique:

- Key creation: this process involves extracting appropriate attributes in order to compute a blocking key to apply to the dataset records under comparison. This can also be done with substring sequences of the attributes, which can be determined in a diversity of ways. The priority of the attributes is determined by which ones appear first in the key, and rank higher than those that appear later.
- Data Sorting: The blocking key value outlined above can be used to sort the dataset records.
- Limiting the comparisons within a window involves fixing the size of the window (w >1) and overlapping it throughout a series of records. Fig. 2 shows, if w = the size of window records, then adding records to a window and comparing each new one with previous records involves adding it to the preceding record w- 1.

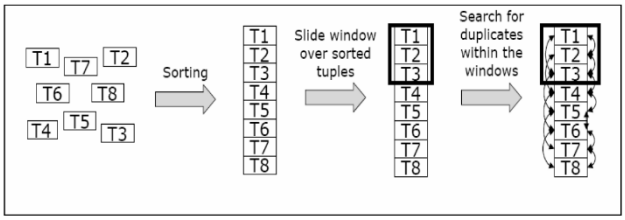

Fig. 2 Windowing [6]

The comparison of potential pairs of records is limited to w −1 by the use of the windowing method. If the number of records in the dataset is n, then the number of records in total is O (wn).

The selection of keys determines how accurate the windowing technique ends up being in practice. For example, a poor choice in a key with too small a window oversees records that should, in fact, be matched [1]. In contrast, a

poor choice in a key with too large a window results in too many undesirable comparisons [7].

Hernández and Stolfo offer a solution to this problem by merging the results of a multi-pass windowing technique that involves a sliding window and changing the key for each of a number of small-window periods [1], [8].

C. Comparison between Block & Slide Window

An in-depth comparison test that strives for algorithms capable of weeding out all but the record pairs most fit for comparison involves massive costs. Pairing up records that correspond to identical entities in the real world poses a challenge. Another challenge that presents itself is the costliness of a process attempting to accurately compute resemblances between pairs in larger datasets. With this in mind, it is a high priority to try to search, spot and locate duplicates with the most effective and efficient system.
The performance bottleneck for duplicate detection is related to the expense of comparison. To overcome these excessively expensive comparisons of all pairs of records, the best practice is to cautiously partition the records into smaller subsets and search for duplicates only within these partitions. Table 1 compares the blocking and windowing techniques in terms of working mechanism, occurrences, sizing, optimization, scalability, and efficiency.

TABLE I
COMPARISON OF BLOCKING AND WINDOWING TECHNIQUES

Process	**Blocking**	**Windowing**
Working mechanism	A set of records is partitioned into disjoint blocks by means of a blocking key.	The first phase involves assigning a sorting key to every record. The key need not be unique for purposes of blocking methods. The second phase involves sorting the records based on the respective key of each one. The third phase involves sliding a window of unvarying size across a list of records that has been sorted along the lines outlined above.
Comparison occurrences	Where the entire number of comparisons is limited to records located in the same partition. This acts to reduce the potential number of comparisons to a notable degree.	When determining the blocking method, the choice of which partitioning predicate to use ultimately characterizes the first two phases.
Sizing mechanism	The choice of which size partition to use is crucial to the creation of adequate partitioning. Given how easy it can be to locate duplicates, the ability to group duplicate candidates in the same partition is a reflection of how good the aforementioned decision is.	The contrast of efficiency and effectiveness is determined by the window size, generally between 10 and 30. Smaller window sizes are more efficient given their shorter run times, but larger window sizes are more effective given their greater thoroughness in spotting and locating duplicates.
Optimization mechanism	Optimal duplicate detection requires an iterative approach, especially in cases where duplicates have distinct partitioning attributes. The manifold approach involves distinct partition bases for each run. To eliminate any potential duplication, a transitive closure applies to all determined duplicates.	The Sorted Neighbourhood Method provides one means of cutting down on processing time by sorting records into clusters and dealing with them in a parallel fashion that makes for a smaller number of comparisons. As noted above, the apparent efficiency of too small a window size risks missing duplications that should be included. Conversely, the greater efficiency of too large a window size wastes time and resources with more comparisons than are needed.
Scalability	Technically, blocking isolates every block and scans it individually out of other blocks. Therefore, there is no overlapping. Blocking may perform better than windowing in terms of a number of comparisons. Only the number of blocks affects it, and that has been resolved by using parallel programming.	The window slides every time, which takes more time than blocking. This sliding technique will obstruct using parallel programming and may affect the scalability of windowing technique.
Efficiency	Both techniques suffer from size trade-off (size of window and block).	

Blocking and windowing have quite similar techniques. For instance, both try to decrease the number of comparisons by intelligently predicating the duplicated pairs. Both methods mainly depend on sorting to run duplicate detection effectively.

Despite their widely differing approaches, blocking and windowing have both become increasingly common means of dealing with the challenges mentioned. Respectively, they handle the problems by creating subsets that are disjoint to divide records. On one hand, boxing records up in a window to sort what lays within its range: on the other hand, some researchers such as Draisbach and Naumann merged both technologies [9]. They proposed an algorithm called Sorted

Blocks [9]. The performance of the sorted blocks with different datasets showed that this algorithm is capable of finding an equal number of duplicates, even though it makes fewer comparisons.

Sorted Blocks first order the records according to a key. It has assumptions, such as that the duplicated records are positioned in tight proximity to each other just as with the two previous approaches. However, unlike multi-pass techniques, which slide a window of unvarying size across a list of records, the sorted block approach makes a comparison of records that fit into disjoint partitions. Overlapping is used within sorted blocks to make sure that duplicated records are found. However, this additional complexity is linear.

The varying partition size of the sorted blocks approach provides advantages that windowing approaches lack with their unvarying window size. Most notable among these is the fact that it discriminates between records of parallel field magnitude and those of differing field magnitudes, which accommodates the comparison of fewer fields.

The trade-off is critical when it comes to time and accuracy. Therefore, many researchers recommended flexibility in the algorithm of records duplication. In other words, researchers let the user make decisions that are important to them, either by time or accuracy: this depends on their requirements. As such, Papenbrock, Heise, and Naumann recommended progressive duplicate detection [10]. The progressive approach utilizes parallel computing in order to reduce the time and increase scalability. Moreover, it allows users to interrupt the detection process and make modifications to improve the process. However, such an approach has many drawbacks: first, it assumes that the user has little knowledge about the cleaning process and of key generations. Such lack of knowledge may influence the accuracy of the result, as the user may select inefficient keys, or intervene erroneously. Besides, duplication detection process is a batch process due to the vast amount of records: interactivity is not recommended as it results in long user waiting time.

In essence, whichever technique is chosen for records detection, there are many considerations to bear in mind:

- The data itself determines the best approach to detect duplication.
- There is always a trade-off between accuracy and time. The higher level of accuracy wanted, the more time is required to do detailed detection. For those people who want instant results, it is better to apply progressive approaches where data comes out whenever available without waiting for the duplication process to finish. Technically, linear searching is more accurate because it compares each record with the entire dataset: however, it is impractical, as it takes too long. Therefore, a balance between time and accuracy is the main motivation for new approaches to duplication detection.
- Whatever record duplication algorithm is selected, it does not offer an optimal solution. Therefore, repeating duplication detection procedures with different keys or overlapping blocks or windows will enhance the accuracy of the results.
- The result of records duplication detection algorithm is significantly dependent on the quality of data transformation that has taken place in prior steps.
- Clustering is a way of reducing time by grouping records based on specific criteria, but again the quality is dependent on the prior steps. In other words, clustering is trying to use predictive criteria rather than criteria derived from the structure of the dataset such as name, telephone, or address. Criteria employed by clustering is developed by individuals who postulate data agreement based on previous experience. As an example, a data scientist might propose that the rich and young are more likely to purchase sports cars. Thus, sorting data based on criteria of age and wealth can assist duplicate deduction in such a group of records
- String comparison is a crucial component of those techniques for spotting matching information in duplicated records. A goal common to all approaches is to find similarity in the meaning of two strings, instead of absolute equality of characters entered, as character matches may exist for a number of reasons, including typographical errors and diverse means of entering the same information (e.g. abbreviations, numerals in place of spelled-out words for numbers, etc.).

D. Similarity measurements

Similarity measurements are proposed to identify whether there is a match between two database records. This match is used to make a decision whether the both records are identical. Many similarities methods were matching based on a number of similar characters in both records, while other methods matched based on similar words. Beside this, there are two types of duplication record detection methods: one related to a normal comparison, and another related to using machine learning or artificial intelligence.

It cannot be said that there is a single comprehensive comparison function that can provide a wide solution for record duplication detection in many domains. Vatsalan and Christen reported that there are three string comparison functions commonly used with medical records: the longest common substring approach, used since the earliest days of computer programming: the Levenshtein Edit Distance, pioneered in the 1960s; and the more recent Jaro-Winkler Distance, popularized in the 1990s. As with other contrasting means of drawing systematic comparisons, the right balance must be found along the continuum of efficiency and effectiveness - in the area of string comparison functions, this is generally referred to as specificity and sensitivity [3].

Several methods have been devised to carry out this task: however, each method tends to work well for particular types of errors, and none appears to offer a universal solution that fits all.

1) Character-Based Metrics: These metrics take into account the inverse similarity (or distance) between two strings of text to compare or approximately match them, or to carry out searches on "fuzzy strings". There are several variations on these string metrics; the most frequently used being described by Gomaa [11]. These variations are:

- Edit-Distance or Levenshtein Distance: Levenshtein introduced the Levenshtein Edit Distance concept that calculates a two-string distance (c1, c2) needed to minimize the number of operations used for deriving one string from another [12]. As for identifying typos, the measurement of edit distance is an effective tool, but it is not as useful for identifying other types of errors [13].
- Hamming Distance: Hamming distance is principally used for numerical fields of a fixed size, for example, zip codes or social security numbers. It counts the mismatches between the two numbers being compared when determining the lowest number of substitutions needed (or the lowest number of errors incurred) when deriving one string from another.
- Shingles (a type of n-gram): These are contiguous sub-sequences of tokens of length w (number of tokens in each shingle composing the set), which are a set of unique shingles that appear in a document, and are used to assess document similarity. The number of w shingles shared by paired-up documents can determine similarity according to this set of criteria [14]. However, Williams and Giles deemed it impractical in computational terms to calculate shingle set similarity for every document [15]. Therefore, they employed a method of producing a document sketch by h-hashing the shingles, locating the lowest value associated with each hash, and using these lowest values to construct a sketch. In order to estimate the similarity of the sketches, they overlapped them.
- Jaro: Considers common spelling errors, and is principally used for record linkage purposes.
- Jaro-Winkler: An extension of Jaro distance devised by William E. Winkler in 1990. This makes use of a prefix scale that gives more positive ratings to strings that match from the beginning for a set common prefix length, up to a maximum of four characters [11].
- q-gram: The q-gram similarity between the corresponding values is calculated, and the similarity is added with a weight to the similarity of the two products. Where q-gram Distance (s1; s2) is the number of different q-gram occurrences in s1 and s2 [17], [18].

Character-based measurements compare character to character. These may be affected by the size of the record, which may influence the overall performance of duplication detection when using this approach. However, the accuracy is considerable because all available characters are scanned.

2) Token-Based Metrics: When it comes to typographical errors, the Character-Based Similarity Metrics work really well. The downside of this is that typographical conventions may lead to certain unsolicited rearrangement of words (e.g. “Walter Evans” vs. “Evans, Walter”). In cases like this, the Character-Level Metrics fail to net similarities of the components. This is where Token-based Metrics come in and try to compensate for this problem. [11], [17].

3) Numeric Metrics: Even though there are numerous methods currently used for detecting similarities in string-based data, methods seeking out these similarities in numeric data are extremely dated. The numbers are characteristically regarded as strings or simple range queries, which pinpoint numbers that have similar values. Koudas et al. suggest that for future research the direction should point towards considering data together with its type and its distribution [18]. Another direction to be considered is the extension of the notion of cosine resemblance when it comes to numeric data, and how well it works in detecting duplicates [11], [14], [20].

In terms of scalability and performance, there are few differences among these three metrics. Some have better performance than others in certain domains, but not considerable differences. Moreover, it was noticed that methods that use clustering offer better performance, due to very good categorization for records that facilitate finding duplicates easily. The challenge comes with developing methods that minimize the unnecessary check of duplication, provide very good response time, and simultaneously have scalability. However, it seems hard to achieve this triple benefit. Yancey demonstrated that, for purposes of matching names, the Jaro-Winkler metric performed well, and, for purposes of matching tokens and characters, the metric used by Bilenko et al. performed well. The Monge-Elkan metric performed the best for distance metrics, while the SoftTF performed the best for IDF metrics [16]. Given that most metrics performed well in some areas but poorly on others, some called attention to the necessity of metrics that can more flexibly accommodate a variety of similarity comparisons.

Increasing the efficiency of comparing single records can possibly increase the efficiency of spotting duplicates. Moreover, there are certain fields that, if scanned for duplications first, could clarify that the records are not duplicates before resources are allocated to scanning the documents in their entirety. Thus, it is imperative to decide the field comparison for the two records as soon as possible to eliminate time wasting.

In order to curb time wastage, “the global likelihood ratio for the full agreement for each of the identifiers should be calculated” [17]. Throughout the duration of the comparison, the maximum collective evidence that is most likely to be amassed from that point forward will be an indication of the rate and/or quality of improvement in the holistic likelihood ratio, in the case of methodical continual comparisons.

Verykios et al. proposed a group of techniques to minimize the number of record comparisons [22]. The techniques involve applying a feature subset selection algorithm in order to decrease the dimensionality of the input set. This pre-processing step accelerates the process of comparison by limiting its search to the more manageably sized subset of fields in the records being compared [21], [22]. Verykios et al. similarly advocate smaller model trees by pruning the derived decision trees themselves, thereby speeding up and improving the accuracy of the categorization of record pairs [22].

E. Duplicate Record Detection Techniques

The techniques described previously are all capable of being employed to pair up and match fields one at a time in a given record. However, the practical database has records consisting of multiple fields, which complicates the process of spotting and locating duplications and necessitates a choice between deeming the record pairs matches or mismatches [23], [24].

For multiple field records, there are many techniques to detect duplication. Elmagarmid et al. categorized them as follows [17]:

- Techniques that involve machine-learning models, in addition to probabilistic methods, to learn by themselves the best ways to pair up records for potential matches.
- Methods guided by various types of domain knowledge, often complemented with distance metrics (of a generic variety), to match records in ways requiring less machine learning.

F. EM Algorithm

According to Jaro, estimates for ML of data that is incomplete should be obtained using an EM algorithm [25]. When a subset of the desired training data is not available for computing conditional probabilities, this provides a practical second option. A good deal of information in the data set can also be harnessed [26]. This also computes the E-step and the M-step iteratively on a data set that is incomplete to create the parameter set θ= (m, u, p). The steps outlined in Dumpster, Laird, & Rubin are as follows [26]:

- Give initial values of θ. These values can be any values since the algorithm is not particularly sensitive to the starting values.
- Compute the E-step using the values of θ.
- Compute the M-step to re-estimate the values of θ based on the values from Step B.
- Repeat Step B and Step C until the convergence of the values of θ.

Estimating mi requires that the record pairs match and that this is reflected in the EM algorithm. A blocking technique typically allows each similar record pair to be grouped into its own block and uses these pairs to compute the λ comparison vector.

G. Rule-Based Technique

Wang and Madnick put forth a rule-based technique that uses heuristic rules defined beforehand in order to determine whether two records match [27], [28]. Techniques of this variety give a one or zero as a weight for each attribute, unlike techniques based on probabilistic methods, which determine the weight of each attribute based on the input. For instance, Wang and Madnick might define rules such as the following [27]:

IF age < 22
THEN status = undergraduate
ELSE status = graduate
IF course_id = 564 or course_id = 579
THEN student_major = MIS

Records referring to an identical entity in the real world are clustered by rules of this sort, and this sometimes leads to incorrect results, given the heuristic nature of their derivation [27].

III. Results And Discussion

This paper has reviewed, analyzed and compared existing approaches in order to suggest the most effective approach in terms of efficiency and scalability. Finding such approach to improve scalability is important because the algorithm able to maintains acceptable response times, even when the size of data processed is massively increased. Many methods suffer from the drawback of quadratic computation, which arises from pairwise distance calculations that affect their scalability. Adaptive tree-based indexing approaches were proposed to reduce such disadvantages [3].

To prove efficacious, duplicates detection should maintain the efficiency of the system and scalability, principally when dealing with large datasets. The standard blocking technique is developed by Baxter et al. [6], and the sorted neighbourhood technique is developed by Hernández & Stolfo [8]. These two techniques share the characteristic of sorting a dataset before applying a window or block to it. They both combine dataset attributes to create a candidate key [1]. Choosing the candidate key wisely is significant in order to avoid missing duplicated records.

Identifying and locating duplicated records can be a sub-optimal process when it involves pairing up all records of a database. If for instance, each record in one data set |A| is paired in turn with every record in a second data set |B|, then the combination of |A|×|B| could produce heavy processing loads: if each dataset has a large number of records, such as 1,000,000 each, this would require comparing 1012 record pairs. Given how relatively small a set of duplication errors would be found after so great an expenditure of resources and time, more efficient methods are not simply more desirable but far more practical.

For efficiency and scalability, grouping data sets into clusters or blocks provide one way of meeting the challenge to increase efficiency. A particularly useful solution involves defining criteria and threshold values, breaking datasets into blocks, and building pairs of candidates within a single block, for use with a Blocking Key Value (BKV) approach. This approach to greater efficiency is geared to not compromise the effectiveness [28], [29].

The reviewed techniques concentrated on the accuracy of duplication detection and focused far less on scalability and performance. There is a trade-off between accuracy and performance. Character-based methods give more details about the similarities, and enable the decision maker to choose whether to consider it duplication or not. However, this takes considerable time as they go character by character. Conversely, term-based methods focus on finding similarity between tokens and take less time in comparison. Then again, they do not give the same leeway to the decision maker as character-based methods.

This paper insinuates an approach of data cleaning based on a tree structure. This approach will assist in determining duplicated data once the new data comes in. This happens due to the fact that the records are saved next to each other based on a specific key. Therefore, a new record arrives is

automatically positioned to the closest key and this helps to implement a duplication determining method, such as Jaro. In a tree structure, it is highly possible to locate duplicated records within the same sub-tree. However, the block or window size selection option will be eliminated. On the other hand, the searching time required will be significantly lower due to searching in small areas (the sub-tree). In addition, there is an increase in accuracy as there is not any chance of any duplicated record being undetected or left behind (as with block and window discussed previously). The tree structure introduces a good solution in terms of accuracy and efficiency but also has a fundamental flaw; which is scalability in cases of data presented and saved in memory. This, however, can resolve by saving the tree structure to a hard disk and is only retrieved it when necessary.

IV. CONCLUSIONS

Since errors involving duplication entry occur routinely, data cleaning of this sort is a crucial part of regular operations. The overview of techniques discussed here displays a range of approaches for searching database records in order to spot and locate these mistakes. There still remains room to expand the repertoire of techniques used in present-day systems. Currently, there are two key approaches for duplicate record detection which are blocking and windowing. Both approaches operate in a simple and fast duplication detection process and can be applied to databases with millions of records. Such techniques should have a balance between efficiency and accuracy. The majority of duplicate detection systems in use at present advocate many algorithms to accelerate the duplicate detection process. The key component should be the flexibility of using methods that adapt and change, independently and automatically, over time when detecting different patterns throughout the duplicate detection process.

Reviewed papers showed weaknesses in terms of scalability, with barely any mention of how far their methods can be scaled. Scalability is crucial because of the nature of duplication detection methods that involve sorting and searching. It is an acknowledged fact that sorting and searching are time-consuming. Therefore, it is recommended that methods be identified that at least eradicate sorting and principally focus on searching of duplications. This paper proposed the tree-based approach for detecting duplications for the following reasons: First, a tree is inherently sorted; there is no need for any sorting. This feature arranges records with relatively similar keys next to each other in a single sub-tree. As a result, the implementation of the Jaro method becomes easy. Second, the accuracy is very high as all duplications will be in one place. Consequently, there will be no records left behind as opposed to with block and window. Finally, there is only one flaw with this method, scalability; the tree grows too fast. However, this can be resolved by saving the tree onto a hard disk. This will be further investigated in future.

The authors regard character-based methods as the best starting point for this task. The output of this study will improve the parameters of evaluation for duplication detection algorithms. This paper pointed out that accuracy, performance, and scalability are considerable factors to evaluate the detection algorithms. The argument was based on the fact that the working domain is a warehouse. Therefore, it is characterized by huge size, and duplicated data for working methods which should be scalable, faster, and accurate. As future works, this study will analyze Jaro method and improve this algorithm in order to minimize running time and increase accuracy in record detection duplication. Providing solutions to this problem will reduce duplication of records in the area of data management, data warehousing, customer relationship management and data integration.

ACKNOWLEDGMENT

The authors would like to thanks the Faculty of Information Science and Technology, Universiti Kebangsaan, Malaysia, for giving the opportunity to conduct this research. This research is funded by Universiti Kebangsaan Malaysia under Fundamental Research Grant Scheme FRGS/1/2014/ICT07/UKM/02/3 and DPP-2015-019.

REFERENCES

[1] O. H. Akel, "A Comparative Study of Duplicate Record Detection Techniques," Master Thesis, Middle East University, Amman, Jordan, 2012.

[2] A. Skandar, M. Rehman, and M. Anjum, "An Efficient Duplication Record Detection Algorithm for Data Cleansing," International Journal of Computer Applications, 127(6), pp.28-37, 2015.

[3] D. Vatsalan and P. Christen, "Privacy-preserving matching of similar patients," Journal of Biomedical Informatics, Vol. 59, pp. 285-298, February 2016.

[4] W. E. Winkler, "Record linkage software and methods for merging administrative lists," US Bureau of the Census, 2001.

[5] P. Christen, "Improving data linkage and duplication quality through nearest neighbor based blocking." in Proceedings of Thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07), 2007.

[6] R. Baxter, P. Christen, and T. Churches, "A Comparison of Fast Blocking Methods for Record Linkage." ACM SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation, Washington DC, pp. 25-27, 2003.

[7] U. Draisbach and F. Naumann, "A comparison and generalization of blocking and windowing algorithms for duplicate detection." in Proceedings of the International Workshop on Quality in Databases (QDB), August 2009, p. 51-56.

[8] M. A. Hernández and S. J. Stolfo, "Real-world data is dirty: Data cleansing and the merge/purge problem," Data Mining and Knowledge Discovery, 2(1), pp. 9-37, 1998.

[9] U. Draisbach and F. Naumann, "A generalization of blocking and windowing algorithms for duplicate detection." in Proceeding of Data and Knowledge Engineering (ICDKE)," 2011, p. 18-24.

[10] T. Papenbrock, A. Heise, and F. Naumann, "Progressive duplicate detection," IEEE Transactions on Knowledge and Data Engineering, 27(5), pp. 1316-1329, 2015.

[11] W. H. Gomaa and A. A. Fahmy, "A Survey of Text Similarity Approaches," International Journal of Computer Applications (0975 - 8887) Volume 68(13), April 2013.

[12] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," Soviet Physics Doklady 10(8): pp. 707-710, 1996.

[13] V. Wandhekar and A. Mohanpurkar, "Validation of Deduplication in Data using Similarity Measure," International Journal of Computer Applications, 116 (21), 2015.

[14] Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," Computer Networks and ISDN Systems, 29(8), 1157-1166, 1997.

[15] K. Williams and C. L. Giles, "Near Duplicate Detection in an Academic Digital Library," in Proceedings of the ACM symposium on Document engineering (DocEng), 2013, pp. 91-94,

[16] W. E. Yancey, “Evaluating string comparator performance for record linkage,” Statistical Research Division Research Report,2005.
[17] K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, “Duplicate record detection: A survey,” IEEE Transactions on Knowledge and Data Engineering, 19(1), pp.1-16, 2007.
[18] R. V. Bezu, S. Borst, R. Rijkse, J. Verhagen, D. Vandic, and F. Frasincar, “Multi-component similarity method for web product duplicate detection,” In Proceedings of the 30th Annual ACM Symposium on Applied Computing ACM, 2015 ,p. 761-768.
[19] D. Mrozek, B. Socha, S. Kozielski, and B. Małysiak-Mrozek, “An efficient and flexible scanning of databases of protein secondary structures,” Journal of Intelligent Information Systems, 46(1), 213-23, 2016.
[20] B. Khan, A. R. S. D. Shah, and S. Khusro, “Identification and Removal of Duplicated Records,” World Applied Sciences Journal, 13(5), pp.1178-1184, 2011.
[21] D. Koller and M. Sahami, “Hierarchically classifying documents using very few words,” Technical Report, Stanford InfoLab,1997.
[22] V. S. Verykios, A. K. Elmagarmid, and E. N. Houstis, “Automating the approximate record-matching process,” Information Sciences, 126(1), pp. 83-98, 2000.
[23] X. Wang, “Matching records in multiple databases using a hybridization of several technologies,” Master Dissertation. Department of Industrial Engineering. University of Louisville, KY, USA, 2008.
[24] C. Conrad, “Predicting Political Donations Using Data Driven Lifestyle Profiles Generated from Character N-Gram Analysis of Heterogeneous Online Sources,” Master of Electronic Commerce Thesis, Dalhousie University, Canada, 2015.
[25] J. S. Murray, “Probabilistic Record Linkage and Deduplication after Indexing, Blocking, and Filtering,” Journal of Privacy and Confidentiality, 7(1), pp.3–24, 2016.
[26] P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm.” Journal of the Royal Statistical Society. Series B (methodological), pp. 1-38, 1977.
[27] J. R. Wang and S. E. Madnick, “The inter-database instance identification problem in integrating autonomous systems.” In Proceedings of the Fifth International Conference on Data Engineering, 1989. p. 46-55.
[28] M. Kejriwal and D. P. Miranker, “On the Complexity of Sorted Neighborhood,”1501.01696, Cornell University, 2015.
[29] J. J. Tamilselvi and V. Saravanan, “Detection and elimination of duplicate data using token-based method for a data warehouse: A clustering based approach,” International Journal of Dynamics of Fluids, 5(2), pp. 145-164, 2009.
[30] F. N. Mahamood and A. Ismal, “Semantic Similarity Measurement Methods: State of Art,” Research Journal of Applied Sciences, Engineering and Technology, 26, pp. 415–430, 2014.
[31] S. Ramya and C. Palani Nehr, “An Efficient Duplicate Detection Based on Navie Block Detection Algorithm,” Middle-East Journal of Scientific Research 24 (Techniques and Algorithms in Emerging Technologies), pp. 291-296, 2016.