TOWARDS A HIGH PERFORMANCE MERGE SOLUTION FOR LARGE-SCALE DATASETS

(Research-in-Progress)
IQ Tools

Marília T. de Mello, Gustavo C. Frainer, José G. C. de Souza, Luiz C. R. Junior
Acxiom Brazil, Porto Alegre - RS
{marilia, gustavo, jose.camargo, luiz}@acxiom.com.br

Abstract: Solving the Merge&Purge problem is a crucial step for Data Quality applications since they depend on integrated views of entities. In the present work, we discuss a domain-independent solution to the problem of detecting duplicate records and its implementation in a real-world application. Our approach is based on the Standard Blocking strategy with some enhancements proposed to overcome the major challenge of deduplication in the industry scenario: maintaining good accuracy while processing large-scale datasets in a timely manner. Experimental results are given on both synthetic and real-world datasets. The performed experiments show that, with the proper configuration, it is possible to achieve high accuracy results, reaching an F-measure of 99.78%. Moreover, the performance evaluation shows that the proposed solution can be applied to large-scale real-world datasets, since it processed a dataset with 45 million records in about 3 hours in a simple PC, and a dataset containing 156 million records in about 5 hours and 40 minutes in a more robust machine.

Key Words: Merge&Purge, Standard Blocking, Data Quality

INTRODUCTION

The Merge&Purge problem, also referred to as data deduplication, duplicate detection or entity resolution, is the process of identifying records that refer to the same real-world entity in one or more data sources [9, 8]. Solving this problem is a crucial step for Data Quality applications since they depend on integrated views of entities. The quality of the data can have significant cost implications to a system that relies on information to function and conduct business [8].

Automatically detecting duplicate records is a process that improves data quality and integrity. It saves both time and money by: maximizing the value of the data; reducing operating costs and manual effort; enhancing customer service; avoiding the waste caused by duplicate mailings; ensuring a single customer view; delivering accurate information for better informed decision making; among others.

Data sources may contain duplicate data mostly due to variations caused by input errors, such as: misspellings, abbreviations, transpositions, missing or incomplete values, etc. Those errors constitute one of the main problems faced by the process of deduplicating data. A good solution must be able to detect duplicate records that refer to the same entity, in spite of any errors they may have. The other difficulty faced by the process is scalability, given that the number of record comparisons grows quadratically with the number of records in the input dataset (when all records have to be compared with each other). If the solution is to be used in a real-world application, this issue is of great concern since large scale datasets need to be processed.

In this paper, we discuss a domain-independent solution to the Merge&Purge problem and its implementation in a real-world application, as part of an enterprise data quality product in our company (the Merge&Purge Strategy platform). Our approach deals with the major challenge of deduplication in the industry scenario: achieving good accuracy while maintaining good performance, given that the size of the datasets that need to be processed is usually large. As the scope of the Merge&Purge solution in our company goes beyond the duplicate detection problem, our proposal will be referred to as Merge solution in the present work.

The main contribution of the present research is the provision of a Merge solution that meets the needs of the industry scenario. Our approach is based on the Standard Blocking strategy which was chosen due to its simplicity, relatively small number of parameters, and good accuracy/performance trade-off [3]. The solution's implementation, performed in Java, has no database or domain knowledge dependency. Furthermore, it provides the following advances, in relation to the basic Standard Blocking approach, which make it possible for the proposed system to overcome the difficulties described above:

1. Splitting the Inverted Index structure into *n* files;
2. Parallelization of the matching process;
3. Matching criteria definition with relevance weights;
4. Output generation as sets of duplicate records, instead of record pairs.

The performed experiments show that, with the proper configuration, high accuracy (both high Precision and high Recall) can be achieved with our approach. In some executions, it was able to reach an F-measure of 99.78%. Experimental results on accuracy are provided on both synthetic and real-world data. Moreover, as demonstrated in the performance experiments, the proposed solution has been successfully applied to real-world, large-scale datasets. A dataset with more than 45 million records was deduplicated in 3 hours and 15 minutes on an off-the-shelf machine, meeting the needs of our organization.

BACKGROUND: STANDARD BLOCKING STRATEGY

When deduplicating a dataset, each record potentially has to be compared to all others in that set. The total number of potential record comparisons in this approach is $|D| \times (|D| - 1) / 2$, which is computationally unfeasible for large datasets due to its high performance cost. In order to reduce the number of comparisons, a blocking technique can be applied so that the input data are divided into blocks before executing the comparison process. This way, only the records in the same block are compared with each other. Standard Blocking [12] is the blocking strategy applied by the solution proposed in the present work.

In the Standard Blocking approach, the user must choose a blocking-key which can be formed by one or more fields from the dataset. The dataset will be split into several blocks according to the key values. All records having the same value in the blocking-key will be inserted into one block. Each record is inserted into one block only. The comparison process is then performed only within each block.

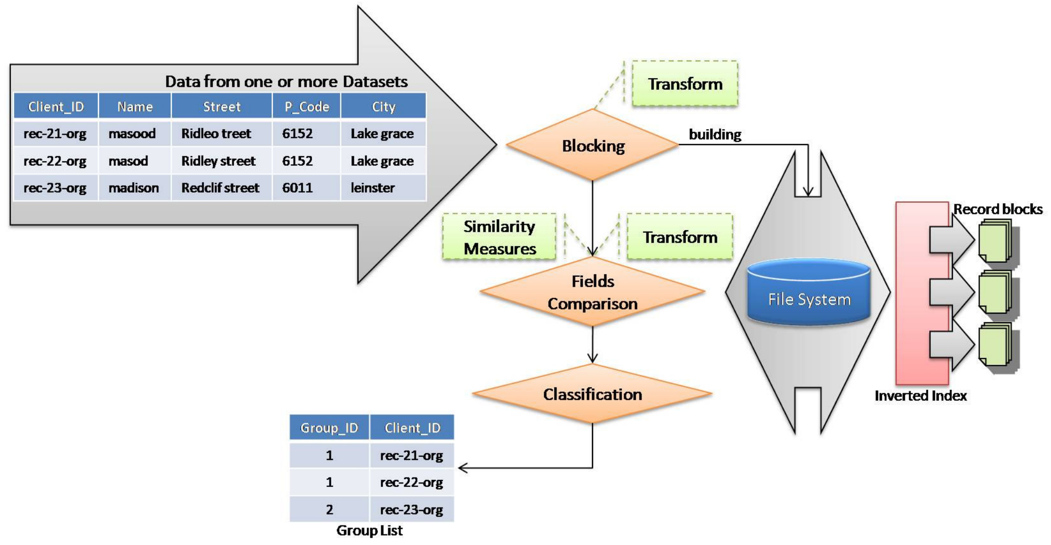
There are two major issues to be considered when choosing a blocking-key. The first one is the size of the generated blocks, which is directly related to the frequency distribution of the key values. If the resulting blocks have a large number of records, there is the possibility of performing more comparisons than necessary. On the other hand, if the blocks are too small, then true matches may be missed, reducing the strategy's accuracy. The second issue is the occurrence of errors in the values used as blocking-keys. This may also result in the separation of duplicate records into different blocks.

The Standard Blocking process is divided into two phases. In the first one, all records from the input source are read, the blocking-key values are created, and the records are inserted into a suitable index data structure. The second step is responsible for performing the comparison between each record in a block to all other records in the same block. Record pair classification as match or non-match is also performed in this phase.

SOLUTION

The Merge solution proposed in the present work was developed into a tool to be applied as a data quality service from our company. Summarizing, the Merge solution takes as input a set of records to deduplicate, which can be provided from one or more data sources that do not need to share the same structure, along with some configuration. Its deduplication strategy is then processed, generating a list of records with group identifiers as output. Besides the initial configuration, the application needs no human interaction. Moreover, it has no database or domain knowledge dependency. The main objective of our Merge solution is to maximize both accuracy and performance.

The application was designed to be flexible, so that other features can be easily included in the future. Furthermore, other blocking techniques to perform the deduplication process can also be easily added (following the idea proposed by the Febrl system [5]), making it possible for the users to choose the desired approach to solve their problem. The currently provided strategy, illustrated in Figure 1, is based on the Standard Blocking approach. It is also executed in two steps: one for mapping the input records to an internal index structure, building the data blocks; and a second one to perform the comparison between the records within each block. No similarity calculations are performed during the first step.



Client_ID	Name	Street	P_Code	City
rec-21-org	masood	Ridleo treet	6152	Lake grace
rec-22-org	masod	Ridley street	6152	Lake grace
rec-23-org	madison	Redclif street	6011	leinster

Group_ID	Client_ID
1	rec-21-org
1	rec-22-org
2	rec-23-org

Figure 1: Merge solution complete process

In the first phase, an Inverted Index structure is used for storing the data blocks, as described in [6]. The (encoded) blocking-key values constitute the keys of the Inverted Index, and all records that have the same blocking-key value are stored into the same Inverted Index list. Differently from [6], instead of only providing phonetic encoding, we propose the possibility of performing other transformations to encode the blocking-key value, such as: *alpha*, *numeric*, *alpha-numeric*, *sorting*, *reversion*, and *substring*.

An enhancement proposed in our approach is in the storage of the Inverted Index structure. Instead of keeping all structure in main memory, which would be unfeasible for large datasets, the data blocks are stored in one or more files. The number of files in which the Inverted Index will be split into is configurable. Then, a memory control policy is employed to manage the loading of each block for the comparison process (in the second step).

Another improvement can be found in the second step. In order to achieve better performance, the comparison step, which often dominates the execution time, can be executed in parallel. This way, the proposed solution saves time by performing the comparisons in more than one block at the same time.

Each pair of records is compared based on match codes which are grouped into one or more matching criteria. Besides the standard configuration, or approach provides means for the user to specify different relevance weights to each criteria element, which increases the chances of detecting correct duplicates. This constitutes another advance provided by our solution.

The comparison result between two records is represented by a vector having the similarity degrees for each compared field weighted by specified relevance values. Each field comparison returns a number, between 0 (total dissimilarity) and 1 (exact math), representing a similarity degree. Each similarity vector is then processed by a classification strategy which determines if the records are similar or not. Fellegi Sunter is the strategy currently available in the application. In this classifier, the similarity degrees of each vector, weighted by relevance values, are summed into one matching result. Record pairs are classified as matches if the similarity result is above a specified threshold, and as non-matches otherwise. If two records are classified as a match, then both will be assigned to the same output group.

A fourth improvement is provided in the system's output generation. Instead of generating pairs of records, the basic Standard Blocking strategy was modified so that groups of records are created as part of the matching process. This way, without creating pairs of duplicate records, our approach avoids the need for a demanding post-processing step to generate the groups from those pairs.

The output created by our application has all records provided as input with an additional field for the group identifier. Each group must contain only records that refer to the same real-world entity. This way, the generated output is ready to be processed by a data quality solution without the need of any post-processing step over the data. In the specific case of our company, the generated output is processed by an application, named Purge, which has the objective of indicating which record better represents each group according to criteria previously defined by the user. Furthermore, if desired, the system cleans the dataset by only keeping one instance of each set of duplicates.

The Merge solution's input and output are read/generated through input and output connectors. They are responsible for specifying which type of source will be used as input and its corresponding output. Both connectors are provided as standard interfaces which can be implemented to attend to specific needs. For example, the input can be read from a text file or a database. Comma separated values (CSV) file format is currently being used for both input and output in our experiments.

Configuration

We aimed at providing a tool flexible in meeting each client's specific definitions of what is and what is not a duplicate. For this reason, the definition of some key parameters, such as blocking-key and matching criteria elements, must be provided as part of the application's configuration. The main difficulty in this task is identifying information that is relevant to duplicate detection in each dataset, without the use of any domain knowledge. Ideally, this choice should be made based on well defined statistic information about the dataset, such as the ones provided by a data profiling tool. Data profiling is the process of examining the data available in an existing source and collecting statistics about that data. Our company provides such a tool which is always applied to analyze the input data for deduplication.

Having that information available, the decision of which fields must be chosen to configure the application is facilitated. For example, fields that have a high percentage of records with missing values are not suitable for being used as blocking-keys. It should be pointed out that the configuration has a strong influence on the results obtained by the solution. That happens because the size of the generated blocks depends upon the frequency distribution of the record values used as blocking-keys. If a very large block is generated, a very large number of comparisons will need to be performed, negatively affecting the application's performance. On the other hand, if similar records are inserted into different blocks, true matches will be missed, negatively affecting the application's accuracy.

If the proper settings are found for each dataset, better results can be achieved both in terms of accuracy and performance. The proper configuration would be the one that generates a small number of candidate matches while covering as many true matches as possible.

Blocking-Key

The blocking-key determines which records will be compared (as it determines which records will be put in the same block). One or more attributes (record fields) can be used to compose a key. If more than one field is selected, their values are concatenated to form the key value. The separation of the input dataset into blocks of records is performed based on those values. Records that have the same value for the blocking-key are inserted into the same data block. However, the big majority of the datasets present errors in the values of their record fields.

In order to deal with this issue, our Merge solution provides a set of preprocessing criteria (previously listed) which can be applied on the values before they are used as blocking-keys. This way, not only equal blocking-key values, but also similar ones are grouped into the same block. Besides that, this feature also provides means for creating more efficient blocking-keys. That is, flexible keys able to perform data blocking according to the specific characteristics from the dataset.

The ideal fields to form a key should have no records with missing values (or at least a low percentage), besides having the same values for the records that must be matched. Moreover, the frequency distribution of the field values should also be taken into account. Key values with many repetitions through the records in the dataset will contribute to large data blocks, increasing the number of record comparisons.

Matching Criteria

Matching criteria, in their turn, define which fields will be compared between each pair of records. The choice of which attributes must be used depends on what constitutes a duplicate. The main problem that must be overcome when comparing pairs of records is the presence of noise and small differences between the ones that must be matched together. In order to obtain better matching results, each record comparison in our approach is performed based on one or more matching criteria, which allow for the definition of relevance weights according to each field's reliability. A matching criterion is a set of match codes. Each criteria element can be one match code or another matching criterion.

A match code is formed by: (1) the definition of the record field(s) that must be used to compare pairs of records; (2) the transformation criteria that must be applied to the field values (optional); (3) a similarity measure to perform the comparison; (4) a similarity threshold; (5) a relevance weight; and (6) the similarity degree that must be applied when there is a missing value (in one or both the records being compared). The transformation criteria available are the same as the ones provided for the blocking-key building (as mentioned before). The similarity measures that can be applied in our Merge solution are based on string comparison techniques. The ones currently provided are: *Equal Matcher*, *Hamming Distance*, *Jaro*, *Levenshtein*, and *Smith Waterman*. A survey about the quality of similarity functions is given in [7].

The similarity threshold defines if the values being compared are similar or not. When a similarity degree is computed between two values, the result must be higher than the predefined threshold for them to be classified as matched. Otherwise, the score will be set to zero. The relevance value, in its turn, denotes the importance of the match code in the similarity computation. It plays a key role in the matching computation, making it possible for the most reliable information to contribute more to the similarity criteria composition. Threshold and relevance values must also be defined for each matching criterion.

The similarity degree for a matching criterion is represented by a vector containing a similarity degree for each criteria element, weighted by the corresponding relevance value. The resulting similarity is computed by the classifier according to Equation 1, in which the similarity degrees (w), weighted by their relevance values (rel), are summed into one matching result. The relevance value of each element must be normalized to a range between 0 and 1. The matching result must also be higher than the threshold value determined for that criterion. Otherwise, the result will be set to zero.

$$Matching\ Result = w_1 * rel_1 + w_2 * rel_2 + \ldots + w_n * rel_n \tag{1}$$

Enhancements

There are four main enhancements in our proposal in relation to the basic Standard Blocking strategy. These improvements make it possible for our Merge solution to achieve high accuracy, while maintaining good performance, being able to process large-scale datasets.

The first improvement is in the way the Inverted Index is built. The main difficulty a Standard Blocking approach faces when building the Inverted Index structure is dealing with very large datasets since this structure can easily become too big to fit in the main memory of an off-the-shelf computer. Depending on the size of the source, the amount of memory required might overwhelm even very robust computers.

In order to deal with that problem, it is necessary to use the secondary storage. One approach to that use would be creating a different file for each blocking-key in order to save the necessary information to process that block of records. The problem with that approach is that, depending on the configuration, millions of files can be created. Experiments have shown that, in certain operating systems, the need to deal with that number of files can slow down the process considerably, or even lead it to freeze.

Another approach would be to save all the processed records with their blocking-key to one big file, and then sort the records in that file by the blocking-key values. After that, simply reading the file sequentially would allow the program to obtain all the information related to a given block, since all those records would be clustered together. The problem with that approach is that sorting one huge file can be very time consuming, due to the need to constantly load records to the main memory and then offload them to the disk again. This is true even for algorithms especially suited to deal with records in secondary storage.

The approach taken in the solution presented in this paper was to break the Inverted Index into *n* files, where *n* is configured by the user. Each record that is part of the Inverted Index structure is, after going through all preprocessing steps, saved to a file. The specific file is chosen through a hash of the record's blocking-key, which means that all records for a given blocking-key end up in the same file. After all records are processed, each file is sorted by the blocking-key values, and then all files are used as sources for the data comparison process.

The advantage of this approach is that, if an adequate number of files is chosen, most of them can be sorted in main memory, speeding up the process considerably. Moreover, when the files do not fit into the main memory, their size in disc is much smaller than keeping only one file with all records. This leads to fewer swaps between main and secondary memory, which also accelerates the process. With this in mind, the number of files should be set according to the size of the dataset and an estimate of the frequency distribution of the values in the fields chosen to build the blocking-key.

The parallelization of the comparison process is the second enhancement proposed in our approach. This allows the solution to make use not only of the parallelism already present in the multi-core machines (which is the standard in the PC market), but also of the processing power of several distributed machines, with the help of an internally developed middleware for the distribution of Java applications. The parallelization makes use of the fact that since comparisons are only made between records in the same data block, each one can be processed independently. This way, several data blocks can be processed at the same time in different threads without any effect upon the final results.

The third improvement is provided through the matching criteria building. Our approach provides means for defining relevance values that weight the similarity degrees resulting from each record pair comparison. This way, it becomes easier for the user to determine which information is the most reliable one (most important), and that information has greater impact on the similarity calculation for each pair of records. This advance contributes to a better accuracy result provided that higher weights are given to the proper information.

Although numerous deduplication solutions have been proposed based on the similarity between two objects, most of them are oriented to discover pairs of duplicates (pairs-oriented solution) rather than sets of similar instances (group-oriented solution). For the pairs-oriented solutions, an additional step is necessary (the grouping phase) in which the pairs extracted during the previous stage must be combined to form groups of entities' copies. This process is analogous to finding the connected components of a given graph, if one considers all the records in a block as vertices with connections between those that are found to be duplicates of each other. This is a problem that is well understood and for which a number of well performing solutions exist. However, these solutions generally assume the whole graph can be kept in memory. When processing large datasets this is often not the case, which makes the problem more complicated and the solutions more costly.

To deal with that complication, a fourth enhancement was made to the basic Standard Blocking strategy. First, the comparison process was more closely linked to the grouping process. Instead of first generating all record pairs and only then creating the groups, two records are already placed in the same group whenever they are found to be duplicates. This leads to a data structure that occupies less memory, since there is no need to maintain information about every pair of records that are considered duplicated. Only the group to which each record belongs to needs to be stored. As a side benefit, this also allows the number of necessary comparisons to be diminished, since if a given record is a duplicate of another one, it is not necessary to compare the first one to any other record that is already in the second record's group.

Even with that enhancement, sometimes data blocks are still too big to be kept whole in main memory. In this case, we use a strategy of breaking them into several smaller blocks which individually fit into the main memory. Each of these smaller blocks will be separately processed and the results of that processing saved to a file. After all blocks are processed the generated results are then joined iteratively until a final result is generated.

The algorithm used to break the large blocks and then join them is inspired by the external memory graph algorithms described in [1], but highly tuned to the specific case of a deduplication process. Its main goals were to diminish, as much as possible, the access to the secondary storage, and to make that access as much sequential as possible. This algorithm was the key to make it possible for the proposed solution to process very large datasets in a timely manner even on off-the-shelf machines.

Enhancements 1, 2 and 4 contribute to a better performance in the solution execution, while the third improvement enables achieving better accuracy results.

EXPERIMENTAL EVALUATION

The proposed solution was used in several experiments with both synthetic and real-world datasets. The experiments' evaluation was made in terms of accuracy and performance. The measures for evaluating accuracy were: Precision, Recall and F-measure; defined according to B-CUBED algorithm [2].

There are two different possibilities of weighting schemes. The one adopted here assigns the same weight to each group. The weight for each element in a group is computed by dividing the weight of the group by the number of elements in that group. Precision is calculated by assigning equal weights to each group generated in the output while Recall is computed with equal weights to each group existing in the template.

In order to evaluate performance, the proposed solution was executed with large real-world datasets as input. The total processing time was computed for each execution, and the system performance was analyzed given the computer configuration.

As stated in the Configuration section, the choice of which fields must be used for each dataset should be made based on well defined statistic information about the dataset. Before running our experiments, we analyzed each dataset using a data profiling tool. Based on that analysis, we selected the fields that should be used as blocking-keys and the ones that should be used in the matching criteria composition.

The main information computed for each field was: number of missing values, minimum and maximum number of tokens, minimum and maximum length, the values' frequency distribution along with a list with the fifty most frequent values.

Datasets

In order to evaluate the proposed solution, four datasets were used: datasets 1 and 2 for evaluating accuracy, and datasets 3 and 4 for evaluating performance. It should be pointed out that the accuracy evaluation demands a template for each input set. Since the available real data sources do not have any templates, an effort had to be made in order to manually generate the desired output for a chosen dataset. For that to be possible, a small sample had to be used, given that the cost of manually generating the template is very high. For this reason, a synthetic dataset was also employed in order to enrich the evaluation. In the performance evaluation, only real-world data sources were used.

A more detailed description on each dataset is given below. The real-world ones contain variations of client register information, such as: name, address, telephone, etc, from Brazilian companies. Due to confidentially concerns, more details on those datasets' attributes cannot be provided.

<u>*Dataset 1*</u>**:** Two real-world data sources, with different structures, were employed to provide samples in order to build the first dataset. The resulting sample, having 203 records (with 85% duplicate), was used for evaluating accuracy. The main errors presented in this dataset are: missing values (8% records in the field *Suburb*), abbreviations (20% records in *Name*, 89% in *Address*, 16% in *Suburb*, and 18% in *City*), misspellings (6.5% in *Name*, and 7% in *Postcode*), and some values containing more information than others (10% in *Name*, and 23% in *Address*).

<u>*Dataset 2*</u>**:** The second input was a synthetic dataset created from a freely available dataset generator [4] capable of creating data sets in English. It contains names, addresses and other personal information, based on real world error distributions. The generated dataset has the following characteristics: 5,000 records (3,750 original and 1,250 duplicate); up to nine duplicates for one original record; maximum three modifications per attribute; and maximum five modifications per record. All types of modifications provided by the generator were used, applied according to a uniform distribution.

<u>*Dataset 3*</u>**:** The third input source is a real-world dataset containing 45,762,365 records, each one with 10 fields. Besides using the complete dataset for evaluating performance, it was also split into smaller samples, with 10, 20, 30 and 40 million records, in order to track the complete duplicate detection process time in relation to the size of the given dataset.

<u>*Dataset 4*</u>**:** The fourth and last dataset is another source with real-world information which was used in the performance evaluation. It contains 156,471,190 records, each one with 6 attributes.

Evaluating Accuracy

In order to evaluate the accuracy of our solution and how its configuration may influence in the quality of the detected duplicates, the application was executed with more than one configuration for each sample. Several configurations were tested, modifying blocking-key and match code settings. Due to space restriction, only the experiments that achieved the best results are described here.

Experiment 1

Dataset 1 was employed in the first experiment in which three configurations were evaluated (Configurations 1, 2 and 3). The field *State* was used as the blocking-key value for all of them. The difference among them lies in the matching criteria setting. The result obtained with each configuration is given in Tabe1.

<u>*Configuration 1*</u>: The first configuration applies one matching criterion with six match codes (see Figure 2), employing the same relevance value (Relevance = 1).

Client (Threshold = 0.8)
– NAME [Jaro; Threshold = 0.75; Relevance = 1]
– ID [HammingDistance;Threshold = 0.85; Relevance = 1]
– POSTCODE [SubString (0-4); HammingDistance; Threshold = 0.85; Relevance = 1]
– CITY [Jaro;Threshold = 0.8; Relevance = 1]
– SUBURB [Jaro; Threshold = 0.75; Relevance = 1]
– ADDRESS [Levenshtein; Threshold = 0.7; Relevance = 1]

Figure 2: Matching criterion in Configuration 1

Configuration 2: Modifying the relevance values in order to give different importance to the match codes, as illustrated in Figure 3, we get an improvement of 4.60% in the result in terms of Recall (Table 1).

Client (Threshold = 0.8)
– NAME [Jaro; Threshold = 0.75; **Relevance = 2**]
– ID [HammingDistance;Threshold = 0.85; **Relevance = 2**]
– POSTCODE [SubString (0-4); HammingDistance; Threshold = 0.85; **Relevance = 3**]
– CITY [Jaro;Threshold = 0.8; **Relevance = 3**]
– SUBURB [Jaro; Threshold = 0.75; Relevance = 1]
– ADDRESS [Levenshtein; Threshold = 0.7; Relevance = 1]

Figure 3: Matching criterion in Configuration 2

Configuration 3: Employing the same relevance value to the fields *Name*, *ID*, *Postcode*, and *City* greater than the value applied for the attributes *Suburb* and *Address*, as shown in Figure 4, we get the best result for dataset 1. From Table 1 it can be seen that the Precision value remained 100%, while Recall increased from 97.65% to 99.04%, reaching an F-measure of 99.52%.

Client (Threshold = 0.8)
– NAME [Jaro; Threshold = 0.75; **Relevance = 2**]
– ID [HammingDistance;Threshold = 0.85; **Relevance = 2**]
– POSTCODE [SubString (0-4); HammingDistance; Threshold = 0.85; **Relevance = 2**]
– CITY [Jaro;Threshold = 0.8; **Relevance = 2**]
– SUBURB [Jaro; Threshold = 0.75; Relevance = 1]
– ADDRESS [Levenshtein; Threshold = 0.7; Relevance = 1]

Figure 4: Matching criterion in Configuration 3

Configuration	**Precision**	**Recall**	**F-Measure**
1	100%	93.05%	96.40%
2	100%	97.65%	98.81%
3	100%	99.04%	99.52%

Table 1: Accuracy results for Experiment 1

Experiment 2

The second experiment was conducted using dataset 2. A substring with four characters from the field *Date_Of_Birth* was used as the blocking-key value in the three configurations evaluated in this experiment. The results for the experiments applying Configurations 4, 5 and 6 are given in Table 2.
Configuration 4: Six match codes were organized in one matching criterion and the same relevance value was applied to each one of them (as presented in Figure 5).
Configuration 5: Keeping the same matching criterion structure but employing a higher importance weight (Relevance = 4) to the field *Suburb*, and an intermediate value (Relevance = 2) to the fields

Soc_Sec_Id and *Postcode* (see Figure 6); we get an improvement in the F-measure score. As presented in Table 2, there was a slight decrease in the Precision result. However, the result for Recall increased from 99.18% to 99.55%.

Client (Threshold = 0.7)
– SURNAME [Jaro; Threshold = 0.8; Relevance = 1]
– GIVEN_NAME [Jaro; Threshold = 0.8; Relevance = 1]
– SOC_SEC_ID [Numeric; HammingDistance; Threshold = 0.8; Relevance = 1]
– SUBURB [Jaro; Threshold = 0.7; Relevance = 1]
– POSTCODE [Numeric; HammingDistance; Threshold = 0.8; Relevance = 1]
– ADDRESS_1 [Alpha; Levenshtein; Threshold = 0.8; Relevance = 1]

Figure 5: Matching criterion in Configuration 4

Client (Threshold = 0.7)
– SURNAME [Jaro; Threshold = 0.8; Relevance = 1]
– GIVEN_NAME [Jaro; Threshold = 0.8; Relevance = 1]
– SOC_SEC_ID [Numeric; HammingDistance; Threshold = 0.8; **Relevance = 2**]
– SUBURB [Jaro; Threshold = 0.7; **Relevance = 4**]
– POSTCODE [Numeric; HammingDistance; Threshold = 0.8; **Relevance = 2**]
– ADDRESS_1 [Alpha; Levenshtein; Threshold = 0.8; Relevance = 1]

Figure 6: Matching criterion in Configuration 5

Configuration 6: The solution proposed in the present work provides means for defining more than one matching criterion, arranged in multiple levels. For some cases, it may contribute to a better accuracy result. Figure 7 displays the matching criteria for the sixth configuration in which the same six match codes used before are separated into three matching criteria organized in two levels. With this configuration, a slight improvement is achieved in the overall result, reaching an F-measure of 99.78% (Table 2).

Client (Threshold = 0.7)
– SURNAME [Jaro; Threshold = 0.8; Relevance = 1]
– GIVEN_NAME [Jaro; Threshold = 0.8; Relevance = 1]
– SOC_SEC_ID [Numeric; HammingDistance; Threshold = 0.8; Relevance = 1]
  Address1 (Threshold = 0.7; Relevance = 1)
    – POSTCODE [Numeric; HammingDistance; Threshold = 0.8; Relevance = 1]
    – ADDRESS_1 [Alpha; Levenshtein; Threshold = 0.8; Relevance = 1]
  Address2 (Threshold = 0.7; **Relevance =3**)
    – SUBURB [Jaro; Threshold = 0.7; Relevance = 1]

Figure 7: Matching criteria in Configuration 6

Configuration	**Precision**	**Recall**	**F-Measure**
4	100%	99.18%	99.59%
5	99.98%	99.55%	99.77%
6	100%	99.56%	99.78%

Table 2: Accuracy results for Experiment 2

Evaluating Performance

The experiments described here were performed in order to evaluate the application's scalability for large-scale real-world datasets.

Experiment 3
Dataset 3 was used to evaluate the performance of the proposed solution when executing in a "simple" machine: a PC with 2GB main memory and an Intel Core 2 CPU with 1.67GHz. The complete dataset, containing 45,762,365 records, was deduplicated by our approach in 3 hours, 14 minutes and 37 seconds. The results obtained with the samples generated from the same dataset, with 10, 20, 30, and 40 million records each, are displayed in Table 3. The same blocking-key was used in all executions: the seven first characters from the field *ID* concatenated with fifteen alpha-numeric characters from the field *Address*.

Records	**Files**	**Heap Size**	**Threads**	**Tasks**	**Blocks Generated**	**Total Time**
10 million	100	1 GB	2	100	6,429,733	00:41:29
20 million	100	1 GB	2	100	12,845,229	01:25:54
30 million	100	1 GB	2	100	20,313,839	02:08:59
40 million	100	1 GB	2	100	27,669,962	03:03:05

Table 3: Performance evaluation on dataset 3

Experiment 4
The second performance experiment was executed using dataset 4 in a more robust machine. The blocking-key used was: a substring with five characters from the field *Postcode* concatenated with ten characters from the field *Name*. More than one configuration regarding the use of cores was applied. The main objective of this evaluation was to compare the complete execution time with and without the use of multiple cores. The tests were executed in a PC with 16GB main memory and two processors Intel Quad Core with 2.0GHz. Table 4 demonstrates the obtained results.

Configuration	**Files**	**Heap Size**	**Threads**	**Tasks**	**Total Time**
1	200	10 GB	1	5000	09:49:24
2	200	10 GB	8	5000	05:43:25

Table 4: Performance evaluation on dataset 4

Real Case Application
In addition to the experiments previously described, an informal evaluation was performed in which a dataset containing 8 million records with information about patients from a hospital was processed as part of a project for the Brazilian government. Due to the lack of a template, the result provided by our application was manually evaluated by the members of our company's Service team and was accepted as meeting the government's needs. The complete process time for this dataset was 01:43:21, in the same simple machine previously described.

Discussion
The effectiveness and performance of our approach is highly dependent on its configuration which, in its turn, is highly data-dependent. The results reported in the previous sections show that the Merge solution presented in this paper is able to deal with large-scale datasets, while achieving high values for F-measure, provided that the proper configuration is defined for each input source.
In the performance assessment, the complete duplicate detection process time in relation to the size of dataset 3 (in Experiment 3) increased close to linearly, as illustrated by Figure 8, which is expected since the number of blocks generated grew almost linearly (see Table 3) and no huge block was created (due to the configuration employed and the fact that the dataset does not have a large number of duplicates).
Figure 9, in its turn, demonstrates the gain obtained with the use of multiple cores in Experiment 4. From Figure 9 (a) we can see the improvement achieved in the total execution time. Figure 9 (b) considers only

the second step (the records comparison phase), which is the one benefited from the parallelization. It shows that it lasted 5 hours and 38 minutes when executed on a single core, and only 1 hour and 52 minutes when executed on 8 cores, an improvement of about 301%.

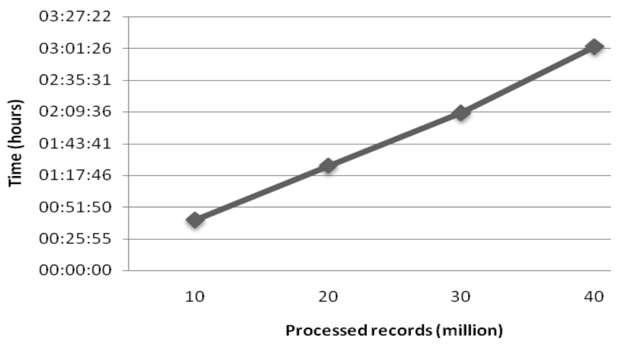
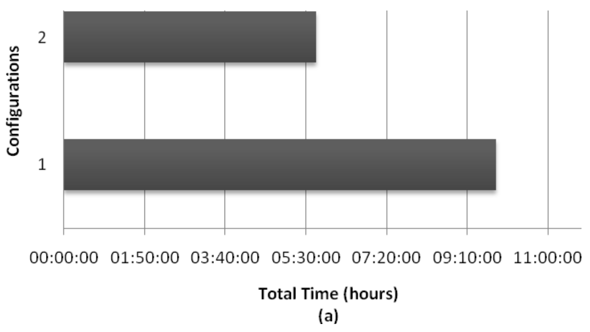


Figure 8: Performance evaluation on Dataset 3



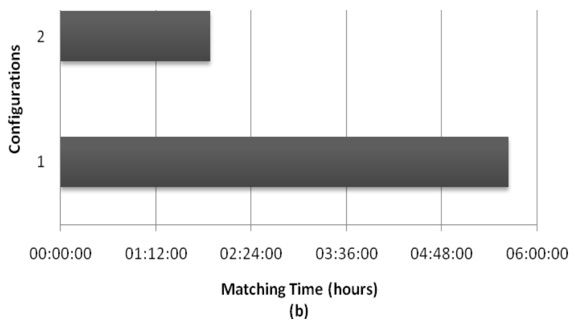


Figure 9: Performance evaluation on Dataset 4

With regard to accuracy, two experiments were carried out. In Experiment 1, due to the lack of templates for real-world datasets, only a small sample from that type of data (dataset 1) could be formally evaluated in terms of accuracy. The field *State* was chosen as the blocking-key value for having no records with missing values, besides having the same values for the records that must be matched. All configurations for this experiment reached 100% Precision, as shown in Figure 10 (a). The score for Recall, in its turn, got lower values since some records incorrectly fell outside their output groups because of the high level of noise in the fields' values (especially *Address* and *Suburb*). This result was improved over the three configurations by applying relevance weights to the appropriate fields in the matching criterion, enhancing the F-measure score.

When all fields in a dataset are noisy, it is possible to apply transformation operations over their values before using them as blocking-keys. Such a choice was made in Experiment 2 in which a substring with four characters from the field *Date_Of_Birth* was used as key. As illustrated by Figure 10 (b), only Configuration 2 got a Precision value lower than 100% (99.98%). That happened since a higher relevance weight was given to the field *Suburb* in that configuration, leading to wrong pairs being classified as matched because of the high similarity score computed for that field.

The result for Recall in Experiment 2 does not reach 100% for two reasons: (1) some records being incorrectly separated into different data blocks due to the chosen blocking-key; and (2) the high occurrence of noise and missing values in all fields. In order to deal with the records that fell into the latter situation, the matching criterion was refined through the use of relevance weights and multiple levels (Configurations 5 and 6), improving the results for Recall and F-Measure.

As for the similarity measures, it was verified (from our experiments) that *Jaro*, for example, was more suitable for comparing names, *Levenshtein* worked better for addresses, while *Hamming Distance* and *Smith Waterman* were more appropriate for numerical fields like *Postcode*. However, there is the need for a more in-depth evaluation on those measures in order to provide a more precise guideline for choosing the best option.

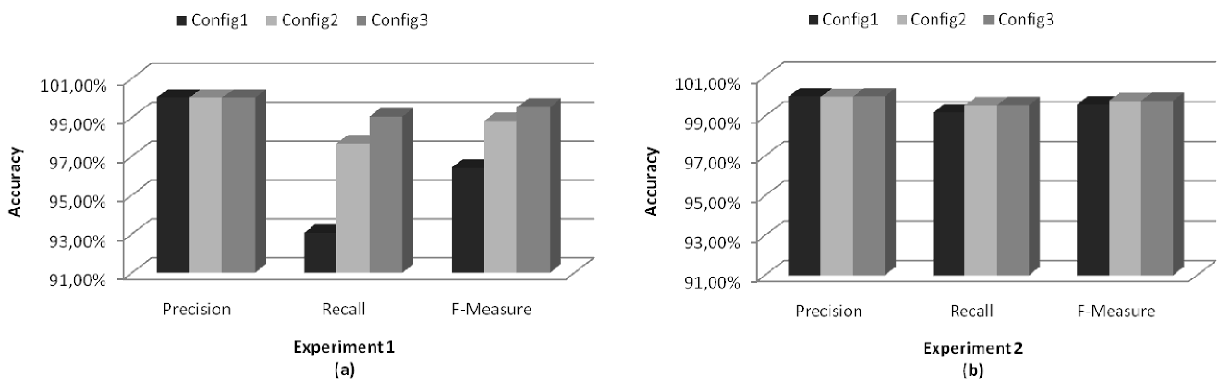


Figure 10: Accuracy evaluation in Experiments 1 and 2

Comparison with Other Techniques

As previously mentioned, the main objective of the present work is to meet the needs of a real-world application, being able to process large-scale datasets without significant loss in the accuracy of the provided results. There are several pieces of research available regarding duplicate detection. However, there are few publications on real-world applications of the proposed solutions. Moreover, most publications in the field of data deduplication present experimental results based on only small to medium sized datasets [3, 6, 13, 14]. Besides, as mentioned in [8], there is a lack of standardized, large-scale benchmarking datasets for comparing new merge techniques with existing ones. With this in mind, two approaches are described here: an academic system and an industry application.

Febrl[1] (Freely Extensible Biomedical Record Linkage) is an open-source linkage system with a graphical user interface [5]. It contains many advanced techniques for data cleaning and standardization, blocking, field comparison, and record pair classification. Its main advantage is the possibility of combining several techniques in order to conduct experiments for evaluating the suitability of each approach for a specific dataset. However, Febrl is an academic tool and only provides means for conducting small to medium sized experimental deduplications, with up to several hundred thousand records.

Some experiments performed with our solution were reproduced using the basic Standard Blocking strategy provided by Febrl. The result for the accuracy evaluation on dataset 1 was: 91.92%, 99.19% and 95.42% for Precision, Recall and F-measure, respectively. For dataset 2, the result was: 84.28%, 100% and 91.74%. Although achieving lower values for Recall, our proposal gets higher values for Precision, reaching greater F-measure with both datasets. As for performance, we tried to use dataset 3. However, only a sample with 300,000 records could be processed due to memory requirements. It took 23 minutes and 45 seconds to deduplicate this dataset. Besides that, it should be pointed out that Febrl's output is generated in the form of record pairs. It does not provide means for obtaining groups of records.

A proposal for a real-world application is given in [16]. This work presents a solution specifically developed for a batch duplicate detection process at Schufa Holding AG. The approach adapts DogmatiX, initially developed to detect duplicates in XML, to deal with relational databases and extends it to include domain knowledge in the form of rules. It applies the multi-pass Sorted Neighborhood Method (SNM), with the improvement of a key definition, to detect duplicates. Experimental results are

[1] http://sourceforge.net/projects/febrl/

given for both accuracy and efficiency.
The proposal reached 90% Precision for 53% Recall, when time is critical, and 85% Recall using more time. For performance evaluation, a 10 million persons dataset was processed after 15 hours on an off-the-shelf PC. The solution was only applied for detecting duplicate persons, due to the extensive use of domain-knowledge. This restricts the use of the application, demanding an adaptation of the solution whenever it needs to be applied in another context. Besides being domain-independent, our application is able to achieve high values for Recall (higher than 90%) without significant loss regarding performance.
Several commercial tools [10, 11, 15] can be found regarding duplicate detection on customer data. However, as in [16], it is not possible to perform a comparison with them as there are no publicly available experimental results in terms of accuracy and/or performance on their approaches.

CONCLUSION

The efficiency of an information processing infrastructure may be greatly affected by the quality of the data present in its data sources. As already mentioned, the (lack of) quality of the data may be strongly determined by the duplication level in that dataset. In order to solve this problem, a domain-independent approach to detect duplicate records was proposed as the Merge solution in a real-world scenario. The solution, based on Standard Blocking strategy, provides enhancements over that strategy in order to provide better performance while achieving high accuracy.
The performance improvement is given by the use of the file system to store the data blocks indexing structure (an Inverted Index), the parallelization of the records comparison process, and the generation of record groups, instead of record pairs, as part of the matching process, avoiding the need for a demanding post-processing step in the solution. The enhancement proposed for achieving higher accuracy is the provision of relevance weights in the matching criteria definition. This way, fields that are more reliable in the dataset can contribute more to the similarity degree calculation between each record pair.
The proposed solution was evaluated in terms of accuracy and performance. The accuracy evaluation was performed with experiments on both real-world and synthetic samples. The results obtained show that our approach is able to reach high values for both Precision and Recall evaluations, reaching an F-measure of 99.78% in one experiment. The performance evaluation, in its turn, was performed with two large-scale real-world datasets. The proposed strategy successfully processed a dataset with 45 million records in about 3 hours in a simple PC, and a dataset containing 156 million records in about 5 hours and 40 minutes in a more robust machine. It is important to observe that this dataset could also be processed in an off-the-shelf machine but, evidently, taking a little longer. Besides the proposed experiments, when compared to the basic Standard Blocking strategy through Febrl, and to a real-world application, our approach achieved better results. Both evaluations provide means for supporting the use of the proposed solution as a real-world application in the context of large-scale companies.

LIMITATIONS AND FUTURE WORK

The present research includes the following items as future work directions:

- Giving support to the use of multiple blocking-keys. This feature can increase the accuracy of the application at the cost of a performance loss. However, if the need to effectively detect correct duplicates is more relevant than the performance issue, providing this feature can lead to even better results.
- Adopting a different strategy for processing large data blocks, as they generate a huge number of record comparisons and end up dominating the execution time of the comparison step. An idea of

a solution to minimize this problem would be the parallelization of the execution inside each large block.

- Providing the automatic generation of datasets with the corresponding templates in order to evaluate accuracy on larger datasets.
- Providing means for evaluating the configuration impact on the solution performance. The result from such an evaluation can help the setting of specific parameters, such as the number of files in which the Inverted Index structure must be stored into.

The main limitation of the presented approach is its dependence on the provided configuration regarding blocking-keys and matching criteria. In an effort to overcome this limitation, we intend to apply an approach, such as an optimization strategy, in order to automatically learn how to configure the application, making the task of configuring the application less complex.

Acknowledgements: The present work is supported by FINEP[2] (project number: 01.07.0178.00) and Acxiom Brazil[3].

REFERENCES

[1] Abello, J., Buchsbaum, A. L., Westbrook, J. "A functional approach to external graph algorithms". *Algorithmica*, 32(3). 2002. pp. 437–458.

[2] Bagga A., Baldwin, B. "Algorithms for scoring coreference chains". In *The First International Conference on Language Resources and Evaluation Workshop on Linguistics Coreference*. 1998. pp. 563-566.

[3] Baxter, R., Christen, P., Churches, T. "A comparison of fast blocking methods for record linkage". In *The First Workshop on Data Cleaning, Record Linkage and Object Consolidation at The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2003.

[4] Christen, P. "Probabilistic data generation for deduplication and data linkage", In *Intelligent Data Engineering and Automated Learning* (IDEAL 2005). 2005. pp. 109-116.

[5] Christen, P. "Febrl: a freely available record linkage system with a graphical user interface". In *HDKM '08: The second Australasian workshop on Health data and knowledge management*. 2008. pp. 17-25.

[6] Christen P., Gayler, R. "Towards scalable real-time entity resolution using a similarity-aware inverted index approach". In *The Seventh Australasian Data Mining Conference*, AusDM 2008 (87). 2008. pp. 51-60.

[7] Da Silva, R., Stasiu, R. K., Orengo, V. M., Heuser C. A. "Measuring quality of similarity functions in approximate data matching". *J. Informetrics*, 1(1). 2007. pp. 35-46.

[8] Elmagarmid, A. K., Ipeirotis, P. G., Verykios, V. S. "Duplicate record detection: A survey". *IEEE Transactions On Knowledge and Data Engineering*, 19(1). 2007. pp. 1-16.

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

[10] IBM InfoSphere QualityStage. Details at: http://www-01.ibm.com/software/data/infosphere/qualitystage/.

[11] Informatica Identity Systems. Details at: http://www.informatica.com/products_services/identity_resolution/Pages/index.aspx

[12] Jaro, M. A. "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida". *Journal of the American Statistical Association*, 84(406). 1989. pp. 414-420.

[13] Rendle, S., Schmidt-Thieme, L. "Scaling Record Linkage to Non-Uniform Distributed Class Sizes", In *The 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (PAKDD 2008). 2008.

[14] Santos, W., et al. "A scalable parallel deduplication algorithm". In *The 19th International Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD'07). 2007. pp. 79-86.

[15] Trillium Software System. Details at: http://trilliumsoftware.com/home/products/TrilliumSoftware.aspx.

[16] Weis, M., Naumann, F., Jehle, U., Lufter, J., Schuster, H. "Industry-scale duplicate detection". *VLDB Endowment*, 1(2). 2008. pp. 1253-1264.

[2] http://www.finep.gov.br/

[3] http://www.acxiom.com.br/