A Comparison and Generalization of Blocking and Windowing Algorithms for Duplicate Detection

Uwe Draisbach
FernUniversität in Hagen, Germany
uwe.draisbach@fernuni-hagen.de

Felix Naumann
Hasso Plattner Institut, Potsdam, Germany
naumann@hpi.uni-potsdam.de

ABSTRACT

Duplicate detection is the problem of identifying pairs of records that represent the same real world object, and could thus be merged into a single record. To avoid a prohibitively expensive comparison of all pairs of records, a common technique is to carefully partition the records into smaller subsets. If duplicate records appear in the same partition, only all pairs within each partition must be compared.

Two competing approaches are often cited: Blocking methods strictly partition records into disjoint subsets, for instance using zip-codes as partitioning key. Windowing methods, in particular the Sorted-Neighborhood method, sort the data according to some key, such as zip-code, and then slide a window of fixed size across the sorted data and compare pairs only within the window.

Herein we compare both approaches qualitatively and experimentally. Further, we present a new generalized algorithm, the Sorted Blocks method, with the competing methods as extreme cases. Experiments show that the windowing algorithm is better than blocking and that the generalized algorithm slightly improves upon it in terms of efficiency (detected duplicates vs. overall number of comparisons).

1. DUPLICATE DETECTION

Duplicate detection is the problem of determining that two different database entries in fact represent the same real-world object, and performing this detection for all objects represented in the database. "Duplicate detection" is also known as record linkage, object identification, record matching, and many other terms. It is a much researched problem with high relevance in the areas of master data management, data warehousing and ETL, customer relationship management, and data integration [4]. Duplicate detection must solve two inherent difficulties: Speedy discovery of all duplicates in large data sets (efficiency) and correct identification of duplicates and non-duplicates (effectiveness).

Efficiency. The first difficulty is the quadratic nature of the problem: Conceptually, each candidate must be compared with every other candidate. To approach this problem and thus improve efficiency of duplicate detection, many solutions have been proposed to reduce the number of comparisons. The general idea is to avoid comparisons of vastly different objects and to concentrate on comparisons of objects that have at least some quickly detectable similarity. Two of the most popular solutions are the focus of this paper.

Effectiveness. The second difficulty of duplicate detection is the definition of an appropriate similarity measure to decide whether a pair is in fact a duplicate. Typical approaches employ combinations of edit-distance, tf-idf weighting and other text-based similarity measures. In addition, a similarity-threshold must be chosen to classify pairs as duplicates (similarity greater than or equal to threshold) or as non-duplicates (similarity lower than threshold). This paper ignores this difficulty; we use only a simple similarity measure and threshold and assume that they yield correct results.

Contributions. We present a comparison and generalized model of the well-known blocking and windowing methods and propose a generalized algorithm, the *Sorted Blocks method*, with a parameter whose extreme settings yield the two methods, respectively. An experimental evaluation for precision and recall compares the two methods and find an optimal setting for the generalized method. These contributions roughly correspond to the structure of the following sections

2. PARTITIONING METHODS

In this paper we analyze two popular families of methods for duplicate detection. *Blocking methods* partition data into multiple blocks or partitions and compare only tuples within a partition. *Windowing methods* on the other hand sort the data, slide a window across the sorted data and compare only within the window. In the following sections we briefly introduce both methods and then compare them.

2.1 Blocking

Blocking methods pursue the simple idea of partitioning the set of tuples into disjoint partitions (blocks) and then comparing all pairs of tuples only within each block [1, 3, 2]. Thus, the overall number of comparisons is greatly reduced; see Tab. 1 for an overview of the computational complexity of the different methods compared to the exhaustive approach of comparing all pairs of tuples.

An important decision for the blocking method is the choice of a good partitioning predicate, which determines

International Workshop on Quality in Databases (QDB) August 24, 2009, Lyon, France

the number and size of the partitions. They should be chosen in a manner that potential duplicates appear in the same partition. E.g., for CRM applications a typical partitioning is by zip-code or by the first few digits of zip-codes. If two duplicate tuples have retained the same zip code, they appear in the same partition and thus can be recognized as duplicates. Other partitionings might be by last name or some fixed-sized prefix of them, by street name, by employer, etc. In general, partitions of roughly same size are preferable. For simplicity we assume in the following that partitions are of equal size. Finally, a transitive closure is formed over all detected duplicates, because duplicity is inherently a transitive relation and thus more correct duplicate pairs can be reported.

To detect duplicates that differ in the partitioning attribute, a multi-pass method is employed. Blocking methods perform multiple runs, each time with a different partitioning predicate.

2.2 Windowing

Windowing methods are slightly more elaborate than blocking methods. In [5, 6] the authors describe the Sorted Neighborhood Method (SNM), which is divided into three phases. In the first phase, a sorting key is assigned to each tuple. The key does not have be unique and can be generated by concatenating values (or substrings of values) from different attributes. In the second phase, all tuples are sorted according to that key. As in the blocking method, the assumption is that duplicates have similar keys and are thus close to each other after sorting. The first two phases are comparable to the selection of a partitioning predicate in the blocking method.

The final phase of SNM slides a window of fixed size across the sorted list of tuples. All pairs of tuples that appear in the same window are compared. The size of the window (typically between 10 and 20) represents the trade-off between efficiency and effectiveness; larger windows yield longer runtimes but detect more duplicates.

To avoid mis-sorts due to errors in the attributes that are used to generate the key, multi-pass variants of SNM produce multiple keys and and perform the sorting and windowing multiple times. As with the blocking method, the transitive closure is finally calculated. Research has produced many variants of SNM, including one that avoids the choice of keys [8] and a variant for nested XML data [9].

2.3 Comparison

The two presented methods have much in common. Both aim at reducing the number of comparisons by making intelligent guesses as to which pairs of tuples have a chance of being duplicates. Both rely on some intrinsic orderings of the data and the assumption that tuples that are close to each other with respect to that order have a higher chance of being duplicates than other pairs of tuples. Their closeness is maybe best characterized by the work of Yan et al. in which they present an "adaptive sorted neighborhood" method, which in fact (and inadvertently?) turns out to be a blocking method [10].

Figure 1 is a schematic representation of both methods under a common model. It shows a matrix in which each field represent a comparison of two tuples. A trivial method might compare each tuple with each other tuple, thus performing a comparison for each field of the matrix. An immediate improvement is to avoid comparing a pair of tuples twice, i.e., to only perform comparisons corresponding to fields above the center diagonal. For presentation purposes we ignore this improvement in the figure, but of course use it for our implementation. Without loss of generality, the model is further simplified, because it assumes all partitions to be of same size, which is typically not the case in real-world scenarios.

Figure 1(a) shows the windowing and blocking methods "as-is". Both methods perform approximately the same total number of comparisons but it is clearly shown that the sets of actual comparisons differ. For instance, Tuples 2 and 5 are compared only by the blocking method, because they lie in the same first partition. On the other hand Tuples 5 and 6 are compared only by the windowing method. Figure 1(b) shows a variant in which the window size of the windowing method was increased so that all comparisons within partitions are encompassed. Figure 1(c) adapts the definition of partitions of the blocking method so as to encompass all comparisons made by the windowing method. To this end, we allow partitions to overlap, which is the basis for the generalized method presented in the following section.

Table 1 shows the computational complexities of the different methods and compares them to the full enumeration of all pairs of tuples.

	Blocking	**Windowing**	**Full enum.**
Comparisons	$n\,\frac{n-b}{2b}$	$(w-1)(n-\frac{w}{2})$	$\frac{n^2-n}{2}$
Key gen.	$O(n)$	$O(n)$	–
Sorting	$O(n \log n)$	$O(n \log n)$	–
Detection	$O(\frac{n^2}{2b})$	$O(wn)$	$O(\frac{n^2}{2})$
Overall	$O(n\,(\frac{n}{2b} + \log n))$	$O(n(w + \log n))$	$O(\frac{n^2}{2})$

Table 1: Complexity analysis with number of partitions b, window size w and number of tuples n.

To further compare blocking and windowing we analyze the relationship between number of blocks b and window size w. To achieve the same number of comparisons for both methods we calculate:

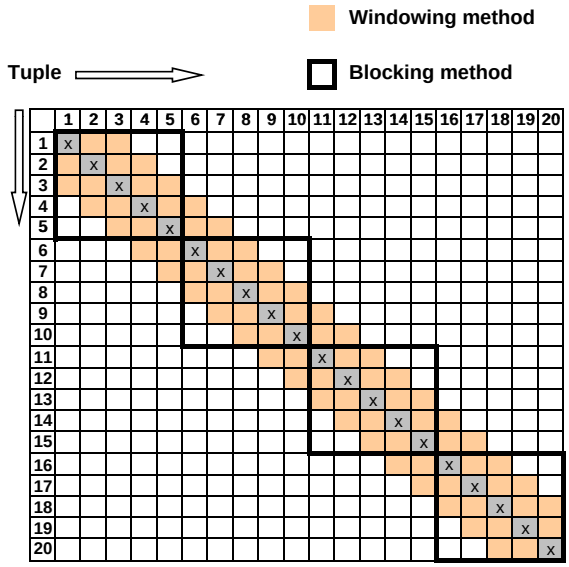
$$n(\frac{n-b}{2b}) = (w-1)(n-\frac{w}{2}) \Leftrightarrow b = \frac{n^2}{2wn - n - w^2 + w}$$

For the example of Fig. 1 with $n = 20$ and $w = 3$ we confirm that the partitioning into 4 partitions approximately achieves the same number of comparisons as the windowing method with window size 3:

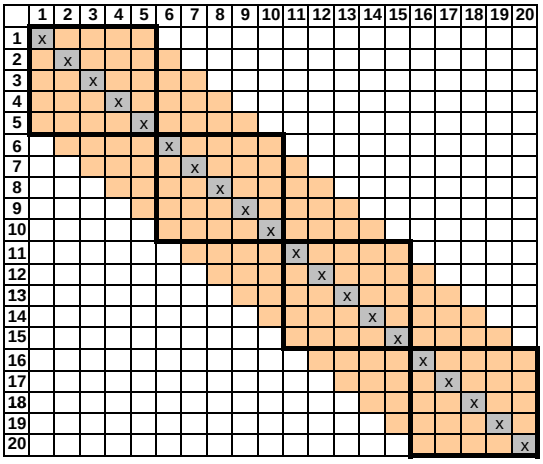
$$b = \frac{20^2}{2 * 3 * 20 - 20 - 3^2 + 3} = \frac{400}{94} \approx 4,26$$

3. GENERALIZATION: THE SORTED BLOCKS METHOD

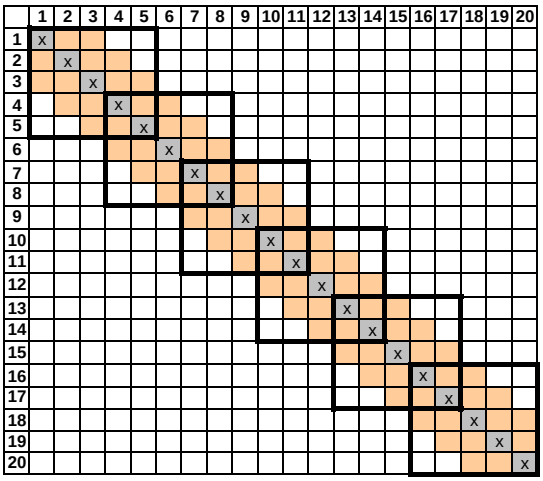
Windowing methods and blocking are two extreme examples concerning the overlap of the partitions. Let U be the intersection between two partitions P_1 and P_2, which we call



(a) Comparing blocking and windowing



(b) Increasing window size to approximate blocking



(c) Overlapping blocks to approximate windowing

Figure 1: Schematic comparison between blocking and windowing methods. Each field in matrix corresponds to a comparison of two tuples.

overlap in this paper:

$$U_{P_1,P_2} = P_1 \cap P_2 \qquad \text{and} \qquad u = |U_{P_1,P_2}|$$

Then we have $u = 0$ for Blocking and $u = w-1$ for Sorted Neighborhood with window size w. Figure 2 shows the effect of an increasing overlap. Parameter u is the number of tuples from one partition that are additionally compared with tuples from the adjacent partition. If the partition size is fixed, an increase of the overlap results in additional partitions and hence additional record comparisons.

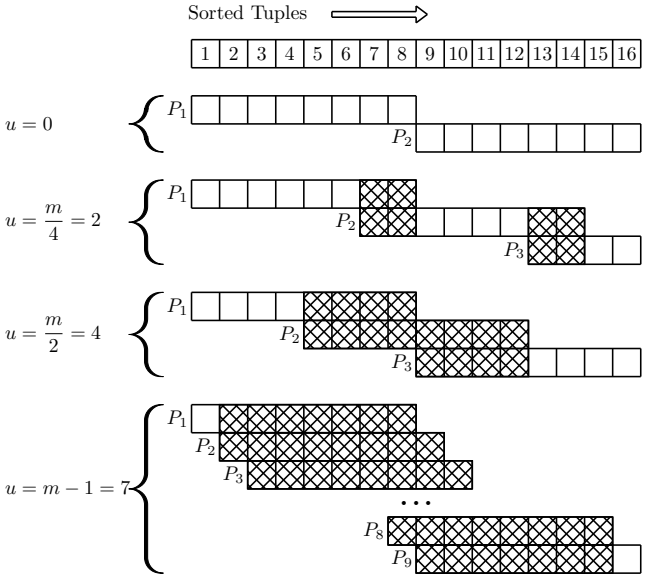


Figure 2: Partition overlap for partition size $m = 8$

In the best case when using partitioning methods, tuples that are indeed true duplicates are assigned to the same partition. The ideal overlap between two partitions should be big enough, so that real duplicates that for any reason are not in the same partition, are identified. On the other hand, the overlap should not be too high, thus resulting in a large increase of record comparisons. The ideal overlap depends on the data set and has to be determined manually for each use case.

The basic idea of the new Sorted Blocks method is to first sort all tuples so that duplicates are close in the sort sequence, then partition the records into disjoint sorted subsets, and finally to overlap the partitions. The size of the overlap can be defined using u, e.g., $u = 3$ means that three tuples of each neighboring partition are part of the overlap, which hence has a total size of $2u$. Within the overlap, a fixed size window with size $u + 1$ is slid across the sorted data and all records within the window are compared. In this way, the additional complexity of the overlap is linear. Note that this windowing technique is used only in the overlapping part; within a partition all pairs of tuples are compared.

Figure 3 shows the Sorted Blocks method for partitions with variable size and an overlap $u = 2$. Within a partition, each record is compared with all other records. The overlap between the partitions results in several windows, which have to be checked for duplicates. For instance, overlap between partitions P_1 and P_2 has windows $F_{(P_1,P_2).1}$ and

$F_{(P_1,P_2).2}$. A special case arises, if a partition is smaller than the overlap. In this case, the windows can comprise more than two partitions, as illustrated for partition P_3. The only impact on the Sorted Blocks method is, that in this case there are identical windows (e.g., $F_{(P_2,P_3).2}$ and $F_{(P_3,P_4).1}$), which are folded in the implementation.

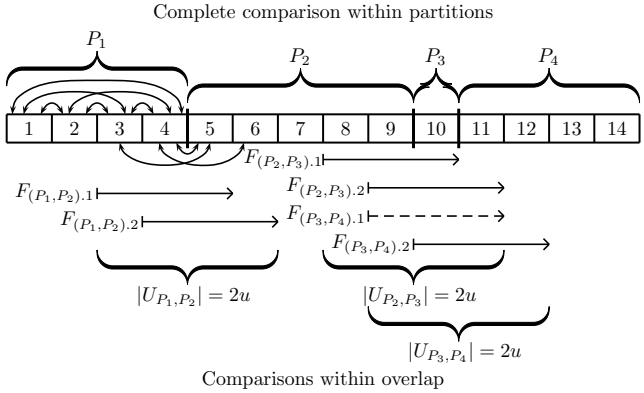


Figure 3: Conceptual illustration of the Sorted Blocks method

Figure 4 shows the program flow chart for the Sorted Blocks method. The tuples are read in a sorted sequence. All tuples that are required for duplicate detection are stored in a linked list. If the current tuple is the first element of a new partition, all elements that are not used anymore are removed from the list. Otherwise, the algorithm checks if it is between two partitions in a window that has to be moved with step size one. Finally, the current tuple is compared with all tuples in the list and then also appended to the list.

The complexity of key generation and sorting is the same as for blocking or windowing methods (see Tab. 1). The number of record comparisons is the sum of comparisons within the partitions and those within the windows. Let p be the number of partitions, then $p-1$ is the number of overlaps, and P_i the set of records within partition i:

$$\text{number comparisons} = (p-1)\frac{u^2+u}{2} + \sum_{i=1}^{p}\frac{|P_i|^2-|P_i|}{2}$$

This calculation does not include the special case of identical overlapping windows, which yields a slight reduction of the number of comparisons. For partitions of fixed size m ($m = \frac{n}{p}$ with n as the number of tuples), the number of record comparisons is:

$$(p-1)\frac{u^2+u}{2} + p\frac{m^2-m}{2}$$

Because u is a constant, the complexity for comparisons is in $O(\frac{pm^2}{2}) = O(\frac{nm}{2})$, which results in an overall complexity for the Sorted Blocks method of $O(n(\frac{m}{2}+\log n))$.

4. EVALUATION AND COMPARISON

In this section we evaluate the presented methods by determining precision (proportion of correctly identified duplicates), recall (proportion of identified real-world duplicates), and F-Measure (harmonic mean of precision and recall), based on a real-world data set.

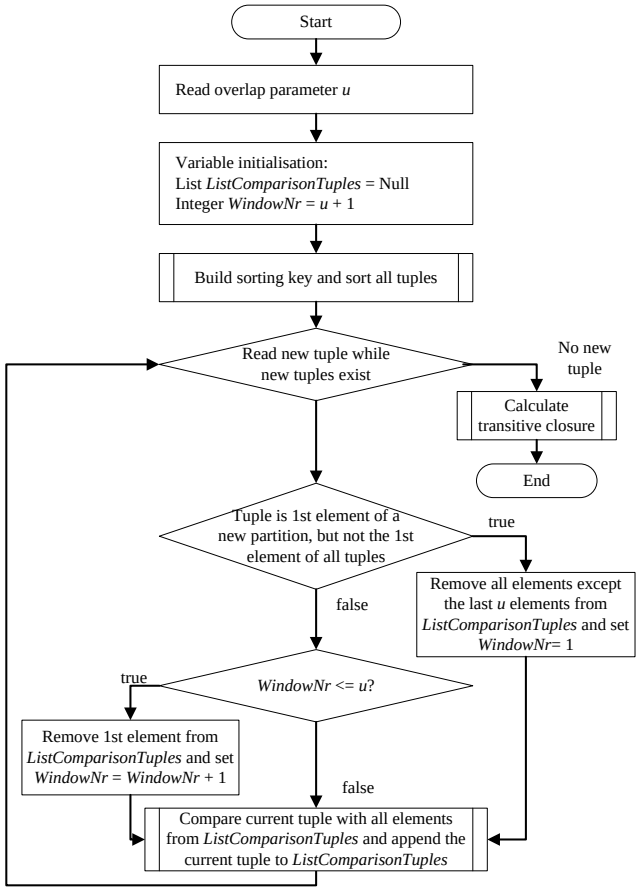


Figure 4: Program flow chart of the Sorted Blocks method

4.1 Data sets and similarity measure

The test data set comprises 9,763 records with audio CD information, such as artist, title, and tracks, which were selected randomly from freeDB[1] [7]. In an arduous manual process a list of 298 real duplicates was generated[2]. Each record has 111 attributes due to multiple occurrences of tracks and artists in the XML sources. Obviously, only a few of the attributes were useful.

The sorting key was built from the three attributes artist1, title1, and track01, using the concatenation of the first three letters of each attribute (using upper-case letters without spaces). This sorting key was used for all three partitioning methods (Blocking, Sorted Neighborhood, Sorted Blocks).

To define our similarity measure we used the same three attributes artist1, title1, and track01. Let t_1 and t_2 be two records, then is their similarity defined as:

$$\begin{aligned} f(t_1,t_2) \quad = \tfrac{1}{3}\times \quad & (s(t_1[\mathsf{Artist1}], t_2[\mathsf{Artist1}]) \\ & + s(t_1[\mathsf{Title1}], t_2[\mathsf{Title1}]) \\ & + s(t_1[\mathsf{Track01}], t_2[\mathsf{Track01}])) \end{aligned}$$

[1] `http://www.freedb.org`

[2] Data sets available at `http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets/`.

with:

$$s(x,y) = \begin{cases} = 1, \text{ if } x\text{=}SubstringOf(y) \text{ or } y\text{=}SubstringOf(x) \\ = \text{threshold, if } IsNull(x) \text{ or } IsNull(y) \\ = 1 - \dfrac{edit\_distance(x,y)}{max\{|x|,|y|\}} \text{ otherwise} \end{cases}$$

The similarity measure calculates the average similarity of all participating attributes. The result is then compared with a threshold, which results in a classification as duplicate or non-duplicate. The similarity measure is not very complex, but it delivers satisfactory results and is sufficient for a comparison of the partition methods.

4.2 Experiments

To compare Sorted Blocks with windowing and blocking, all three methods have been implemented in a Java and MySQL environment. The similarity threshold was set to 0.78, which delivers good results for both precision and recall. The three different methods were executed several times with different partition/window sizes. To obtain comparable results, the partition sizes for blocking were selected in such a way that there was always a corresponding window size with nearly the same total number of comparisons. This was achieved by sorting the tuples and cutting them in fixed size partitions. Additionally, an exhaustive comparison of all tuples was processed without any partitioning. This is especially interesting to see the impact of partitioning methods on the recall.

The most efficient overlap-setting of $u = 2$ between the partitions was selected experimentally. Figure 5 compares the results of different overlaps with $u = 2$ as a baseline. For each F-Measure-value the graphs show the minimum number of additional comparisons necessary to achieve that value.

A first observation is that the graphs are always above zero, so Sorted Blocks with overlap $u > 2$ needs more record comparisons than Sorted Blocks with $u = 2$ to achieve the same F-Measure and is hence less efficient. Another effect we observe is that with an increasing F-Measure, the number of additional comparisons generally decreases. Because the F-Measure depends on the recall and therefore on the number of record comparisons, it can be increased with an increasing partition size, but this reduces the effect of the overlap between the partitions.

An analysis of the real duplicates showed that 86,58% of the tuples have a distance of 5 or less for our sorting order, which means that they can be covered with a partition/window size of 6. Another 10,74% have a distance > 10, so they can only be detected when using large partitions/windows; however, this would result in a large increase of record comparisons. So when we compare the three different partition methods, the range up to a partition/window size of 10 is especially significant.

Figure 6(a) compares the precision of the different methods. The results of the exhaustive comparison are split into a value including the calculation of the transitive closure and one without. They are plotted as a constant, even though in reality they yield just a single point in the diagram. The precision of the three methods is nearly the same, which shows that the precision does not depend on the partition method. Due to partitioning the precision value is better than for an exhaustive comparison. The reason is that in an exhaustive comparison many more non-duplicates are compared with each other and due to an inaccuracy in the similarity function classified as duplicates. This effect is even amplified by calculating the transitive closure.

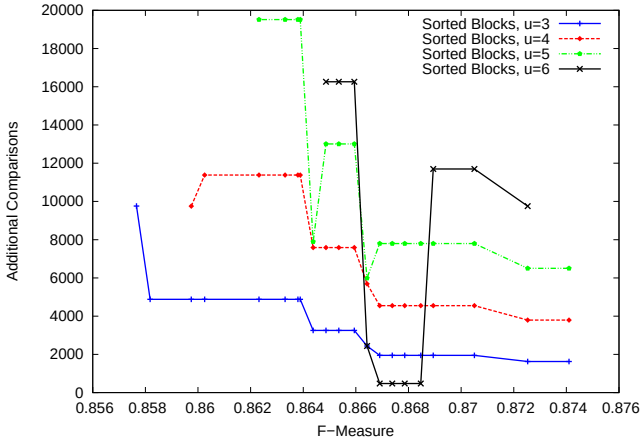
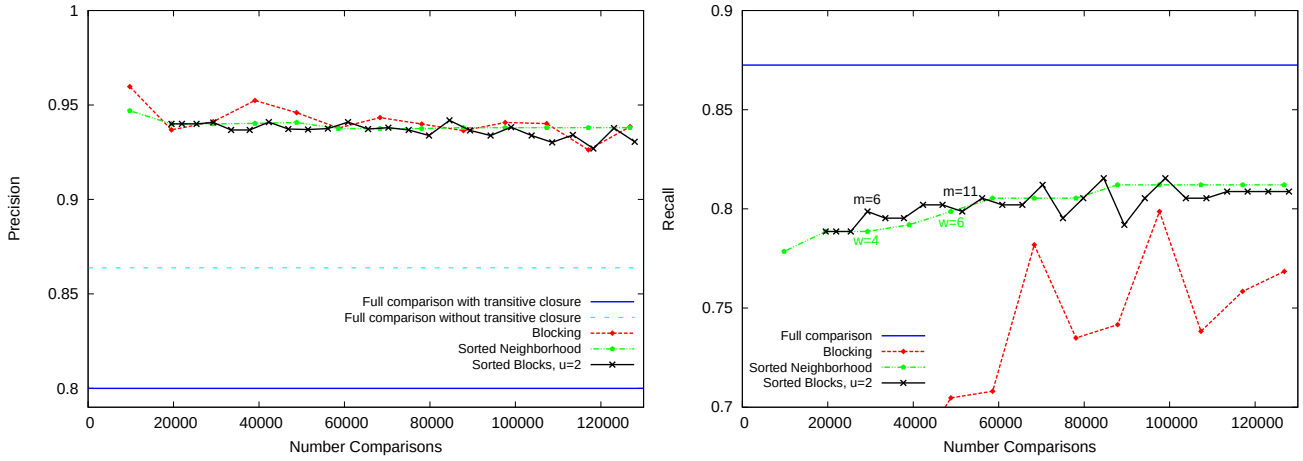


Figure 5: Additional record comparisons for Sorted Blocks method with different overlaps in comparison to Sorted Blocks with $u = 2$

In Figure 6(b) the recall values are compared for which the exhaustive comparison is the upper bound. For the recall of the exhaustive comparison the transitive closure has no effect, because for the used data set no additional real-duplicates could be found. A main result of the comparison is that windowing performs better than blocking, especially for small partitions. While the recall value for windowing is continuously at a high level, it is very low for small blocking partitions and has additionally higher amplitudes.

The recall of the Sorted Blocks method is at first the same as for the Sorted Neighborhood method. From a partition size of $m = 6$ up to $m = 11$ the recall is slightly higher. The Sorted Neighborhood method has its next intersection with Sorted Blocks at window size $w = 6$, which is the same partition size as Sorted Blocks has at its first increase ($m = 6$). This shows the importance of selecting the right partition sizes for the data sets. The difference between the Sorted Blocks and the Sorted Neighborhood method is that Sorted Blocks has less comparisons with $m = 6$ than the Sorted Neighborhood method with $w = 6$ and hence has a higher efficiency. Sorted Blocks achieves recall values earlier than Sorted Neighborhood. In the further course of the graph, Sorted Blocks has several local maxima and minima while the Sorted Neighborhood method is monotonically increasing. The recall value for Blocking is always less than the value for the two other partition methods.

The F-Measure plotted in Fig. 7 is particularly interesting for the assessment of partitioning methods, because it includes both correctness and completeness of the results. As for the recall, Sorted Neighborhood and Sorted Blocks are superior to blocking. The curve shapes are similar to those for recall. We can see again that for the interval $m = 6$ up to $m = 10$ Sorted Blocks performs better than the Sorted Neighborhood. As for the recall value, the curve of Sorted Blocks is not monotonically increasing. The reason is, that the set of record comparisons for an increasing partition size is not necessarily a superset of the record comparisons for smaller partitions.



(a) Precision comparison

(b) Recall comparison

Figure 6: Comparison of Precision and Recall with window size w for SNM and partition size m for the Sorted Blocks method

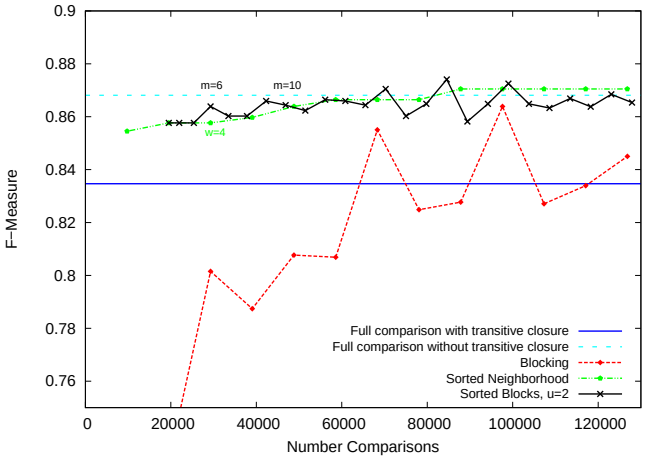


Figure 7: F-Measure comparison with window size w for SNM and partition size m for the Sorted Blocks method

5. CONCLUSIONS

This paper presents two main contributions. First a thorough comparison of the common blocking and sorted neighborhood methods. We clearly show that the latter outperforms the former. Second, led by the intuition that both perform some type of partitioning, we present a generalized algorithm, Sorted Blocks, with the two methods as extreme cases. This method parameterizes the degree of overlap from none (Blocking method) to $w - 1$ (Sorted Neighborhood method). This method in turn outperforms the Sorted Neighborhood method slightly.

Clearly, the results and observation should and shall be confirmed by experiments with more and diverse data sets. As usual, the problem is to obtain pre-classified data. Also, the Sorted Blocks method can yet be refined, for instance by allowing variable block sizes and by allowing dynamic adaptation of the overlap. The former refinement would also remove the inconvenient behavior of the Sorted Blocks that increasing overlap does not necessarily result in a superset of the previous comparisons. Thus, recall-graphs would be smoothed and we expect results to be more clear-cut.

6. REFERENCES

[1] Rohit Ananthakrishna, Surajit Chaudhuri, and Venkatesh Ganti. Eliminating fuzzy duplicates in data warehouses. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Hong Kong, China, 2002.

[2] Rohan Baxter, Peter Christen, and Tim Churches. A comparison of fast blocking methods for record linkage. In *SIGKDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*, Washington DC, 2003.

[3] Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. Adaptive blocking: Learning to scale up record linkage. In *Industrial Conference on Data Mining (ICDM)*, pages 87–96, 2006.

[4] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

[5] Mauricio A. Hernández and Salvatore J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD)*, pages 127–138, New York, NY, USA, 1995. ACM.

[6] Mauricio A. Hernández and Salvatore J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[7] Luis Leitao, Pavel Calado, and Melanie Weis. Structure-based inference of XML similarity for fuzzy duplicate detection. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, Lisbon, Portugal, 2007.

[8] Alvaro E. Monge and Charles P. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD)*, pages 23–29, Tuscon, AZ, 1997.

[9] Sven Puhlmann, Melanie Weis, and Felix Naumann. XML duplicate detection using sorted neigborhoods. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, Munich, Germany, 2006.

[10] Su Yan, Dongwon Lee, Min-Yen Kan, and C. Lee Giles. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the Joint Conference on Digital Libraries (JCDL)*, pages 185–194, Vancouver, Canada, 2007.