Efficient and Effective Filtering of Duplication Detection in Large Database Applications

Ji Zhang
Department of Mathematics and Computing
University of Southern Queensland
Toowoomba, QLD 4350, Australia

Abstract— **In this paper, a robust filtering technique, called PC-Filter (PC stands for partition comparison), is proposed for effective and efficient duplicate record detection in large databases. PC-Filter distinguishes itself from all of existing methods by using record partitions in duplicate detection. PC-Filter operates in three steps. It first sorts the whole database and splits the sorted database into a number of record partitions. The Partition Comparison Graph (PCG) is then generated by performing fast partition pruning. Finally, duplicate records are effectively detected through internal and external partition comparison based on PCG. Four closure properties, used as heuristics, have been devised to achieve a remarkable efficiency of the filter based on triangle inequity of record similarity. The partition size is well specified such that the time complexity of PC-Filter can be optimized. By equipping existing detection methods with PC-Filter, we are able to well solve the major problems that the existing methods suffer.**

I. Introduction

Data cleaning is of crucial importance for many industries over a wide variety of applications [5]. Aiming to detect the duplicate or approximately duplicate records that refer to the same real-life entity, duplicate record elimination is a very important data cleaning task attempting to make the database more concrete and achieve higher data quality. The most naive method is to pair-wisely compare all record pairs in the database in order to detect the duplicate records. Obviously, this method is practically infeasible due to its intolerable complexity of $O(N^2)$, where N is the number of records in the database. To lower the time complexity, various techniques have been proposed. We can broadly classify the state-of-the-art methods into two major categories in terms of their high-level mechanism/framework used to bring potentially similar records together for comparison: window-based methods [7], [8], [13], [17] and clustering-based method [11]. The window-based methods typically sort the whole database based on a key and use an in-memory sliding window to delimit the scope of record comparison. Record comparison is only carried out within the scope of sliding window. The clustering-based methods group the records into clusters with unfixed sizes, and record comparison is performed within each of the clusters independently. Clustering-based methods normally do not require database sorting.

Though they can, to some different degree, improve the efficiency of the pari-wise comparison method, the exiting methods suffer three major problems: the "Key Selection" problem, the "Scope Specification" problem and the "Low Recall" problem. Firstly, the existing methods involving database sorting are very sensitive to the key chosen to sort the database. Two truly duplicate records may be far apart from each other in the sorting using an improperly chosen key. Thus they cannot be detected as duplicates because they simply cannot be contained in the same window for comparison. In addition, the optimal size of the window (for window-based methods) or the clusters (for clustering-based methods) is hard to determine, making it difficult to specify the optimal scope for record comparison for the given database. Large-sized windows or clusters generally improve effectiveness, but this will simultaneously increase main memory requirements and computational complexity. Small-sized windows or clusters, on the other hand, will produce a low recall for the detection result since many truly similar records may fail to be compared, resulting in loss of many true duplicate records. The third problem, "Low Recall" problem, is tightly related to the "Scope Specification" problem. Given the typically small size of the windows or clusters, the existing methods cannot produce results with a very satisfactory recall level as the limited scope of record comparison leads to the failure of detecting many true duplicates.

In addition, we have recently seen the application of explicitly computing the transitive closure by means of the discovered pair-wise "*is a duplicate of*" relationships in the detection methods to save expensive string computation involved [11]. It makes the assumption that if record R_1 is duplicate with record R_2, and record is R_2 is duplicate with record R_3, then by transitivity R_1 is duplicate with R_3. We term this as *Naive Transitive Closure*. Using Naive Transitive Closure in duplication detection suffers the following two major drawbacks. First, naive transitive closure may propagate error. Two record actually having very low similarity may be claimed a duplicate pair by a long chain of pair-wise transition. Second, two records having a high similarity to each other may be regarded as a non-duplicate pair simply because they are not quite similar to a third record the bridges the transition. This is called the "Transitive Closure" problem in our work.

In order to solve the aforementioned problems, we will propose PC-Filter, a novel filtering technique for

doi:10.4304/jsw.7.11.2424-2436

effective and efficient duplicate record detection. The preliminary version of PC-Filter was proposed in [20]. PC-Filter consists of three major steps in detecting duplicate records: database sorting, construction of Partition Comparison Graph (PCG) and record partition comparison. The database is first sorted and sequentially split into a number of partitions and the outer partitions needed to be compared for each partition are found using fast partition pruning. Such information is represented by Partition Comparison Graph (PCG). Based on PCG, partitions will be internally and externally compared in order to detect duplicate records. The major contribution of our proposed PC-Filter is that it is able to solve the four major problems (i.e "Key Selection" problem, "Scope Specification" problem, "Low Recall" problem and "Transitive Closure" problem) that the existing methods suffer. Specifically:

(1) PC-Filter uses the notion of partition instead of a window or cluster to detect duplicate records in the database. PC-Filter will not only compare records within each partition, but also compare records across some different partitions, if necessary. This strategy enables PC-Filter to detect duplicate records even when they are located far apart from each other under a sorting based on an improperly chosen sort key. Therefore, the result of PC-Filter is not only insensitive to the sorting order of the database but also able to achieve a very good recall that is even comparable to the pair-wise comparison method;

(2) The partition size has been optimized to give the best possible speed performance. This provides a good solution to the "Scope Specification" problem;

(3) Instead of using the naive transitive closure, PC-Filter utilizes the transitive closure properties of record similarity measures that satisfy the triangle inequity to save expensive record comparison and achieve better accuracy.

Besides being able to well solve the problems the existing methods suffer, PC-Filter is characterized by following advantages: (i) It is reasonably fast. Four properties, used as heuristics, have been devised based on the transitive closure of record similarity to substantially improve the efficiency of the filter by saving a huge amount of record comparison workload in various stages of the technique; (ii) PC-Filter also features good flexibility as it can be seamlessly incorporated into most existing detection methods: PC-Filter returns an initial set of duplicate result and a post-processing process can be performed to refine the result using another comparison method.

The reminder of this paper is organized as follows. Section 2 presets a survey of the state-of-the-art methods for duplicate record detection. We discuss the transitive closure of record similarity and the properties used for fast partition pruning in Section 3. In Section 4, we will discuss in details our filtering technique, PC-Filter. The complexity analysis of PC-Filter is presented in Section 5. Experiments are conducted to evaluate the robustness of PC-Filter under a wide spectrum of settings in Section 6. The last section concludes the whole paper.

II. Related Work

There are two major branches of research efforts in duplicate detection: the research work that proposes high-level duplication detection frameworks to achieve better performance of the whole system and the works that focus on how to measure the similarity of records more accurately. The major contribution of this paper is in the direction of the *first* research effort.

In terms of high-level duplication detection frameworks, the Sorted Neighborhood Method (SNM) is the first proposed method [8], serving as the basis of many existing duplicate detection methods. SNM involves three major steps, that are, create key, sort data, and merge records. Specifically, a key is firstly computed for each record in the database by extracting relevant fields or portions of fields for discriminating records. Then all the records are sorted by the chosen key. Finally in the step of record merging, a window of fixed size is moved through the sequential order of records, limiting the record comparison to be carried out only within the window. SNM only compares a newly entered record with all the records in the window. The first record in the window will slide out upon the entry of a new record into the window. SNM is able to speed up the duplication detection process by only examining neighboring records for a specific record. Among the variants of SNM are Duplicate Elimination SNM (DE-SNM) [7], Multi-pass-SNM [7], Clustering-SNM [8], SNM-IN/OUT [13] and RAR [17]. In DE-SNM, the records are first divided into a duplicate list and non-duplicate list. The SNM algorithm is then performed on the duplicated list to produce the lists of matched and unmatched records. The list of unmatched records is merged with the original non-duplicate list using SNM again. Multi-pass-SNM uses several different keys to sort the records and perform SNM algorithm several times, rather than only one pass of SNM based on a single key. Generally, combination of the results of several passes over the database with small window sizes will be better than the result of the single pass over the database. Clustering-based SNM clusters records into a few clusters and the record comparison/merging process is performed independently for every cluster using SNM. SNM-IN/OUT and RAR, which are probably the most related methods to our PC-Filter, use several properties based on the lower and upper bounds of the Longest Common Subsequence (LCS) Similarity and TI-Similarity, to save record comparisons without impairing accuracy under the framework of SNM.

Instead of using a fixed window in scanning the sorted databases, the Priority Queue method [11] clusters the records and uses the structure of priority queue to store records belonging to the last few clusters already detected. The database is sequentially scanned and each record is tested as whether it belongs to one of the clusters residing in the priority queue. The information-theoretic metric and clustering technique have also been used to

identify groups of similar tuples, which will be considered duplicates [2].

In addition to the above window-based and clustering-based methods, an on-the-fly detection scheme for detecting duplicates when joining multiple tables [6]. This method, however, is not directly applicable for detecting duplicates in a single table where no join operations will be involved. A fuzzy task-driven duplicate match method is proposed to detect duplicates for online data [1], [4]. Technique for detecting duplicate objects in XML documents is also proposed [19].

Another important issue to address in duplicate record detection is the similarity metrics used to measure the similarity of records. So far, there have been two broad classes of similarity metrics applied in measuring similarity of records: *domain-dependent* and *domain-independent metrics*. As for the domain-dependent metrics, an approach that uses an equational theory consisting of a set of rules to decide whether two records are duplicates is adopted [9]. This set of rules typically involves human knowledge and therefore are highly application-dependent. The major disadvantages of domain-dependent metrics are: (i) the rules can only decide whether two records are duplicate or not, and cannot reflect the degree to which the two records are similar to each other; (ii) the creation of such rules is time-consuming and must be updated to allow for the data updates; and (iii) the rule-based comparison is normally slow and cannot well scale up for large databases. The domain-independent measures, such as Edit Distance, LCS-Similarity and TI-Similarity, are used to measure the similarity of two fields of records by considering each field of the records as an alphanumeric string [11], [12], [16]. N-Gram is used to measure record similarity in [18]. These metrics are domain-independent since they can be applied in a wide range of applications without any major modifications. The adaptive combination of different domain-independent metrics is also studied using machine learning technique in order to achieve a higher level of accuracy than using each of them alone [3].

We refer to [15] for a good survey and classification of various approaches used in duplicate detection.

III. Measurements of Record Similarity

A. Field and Record Similarity Measurements

In our work, the fields of records are treated as strings and two similarity metrics, LCS-Similarity [13] and TI-Similarity [17] are used. The reasons to use these two metrics are two-fold: (i) They can be computed efficiently: the complexity of computing the field similarity is $O(m+n)$ for both measures if the lengths of the two fields are m and n, which is significantly less than the Edit Distance's $O(m*n)$; (ii) They have transitive closure properties, which can potentially help save a huge amount of record comparison in the detection process. We will first give the definitions of the similarity metrics before discussing their similarity-based transitive closure properties.

Longest Common Subsequence (LCS) Similarity. A subsequence of a string s is any string, which can be created from s by deleting some of its elements. The longest common subsequence of two fixed strings is the subsequence that appears in both the two strings and has the maximum possible length. Let $LCS(A_F, B_F)$ denote the LCS of the field F of record A and B, the field similarity of the field F for A and B is computed as:

$$LCS\_Sim_F(A,B) = \frac{LCS(|A_F|,|B_F|)}{max(|A_F|,|B_F|)}$$

TI-Similarity. Suppose a field F in record A has the character set $A_F = x_1, x_2, \ldots, x_m$ and the corresponding field in record B has the character set $B_F = y_1, y_2, \ldots, y_n$, where x_i, $1 \le i \le m$, and y_j, $1 \le j \le n$, are characters in the strings. The field similarity of the field F for A and B is computed as:

$$TI\_Sim_F(A,B) = min(\frac{|A_F \cap B_F|}{|A_F|}, \frac{|A_F \cap B_F|}{|B_F|})$$

Weighted Record Similarity. Suppose a table consists of fields $F_1, F_2, \ldots, F_n$, and the field weights are $W_1, W_2, \ldots, W_n$, respectively, $\sum W_i = 1$. The similarity of two records, A and B, is computed as:

$LCS\_Sim(A,B) = \sum_{i=1}^{n} W_i * LCS\_Sim_F(A,B)$
$TI\_Sim(A,B) = \sum_{i=1}^{n} W_i * TI\_Sim_F(A,B)$

The field weights can be initially specified based on human users' understanding of the discriminating capability of the fields and can be adapted based on the detection results.

Duplicate Record Pair. Two records are treated as a duplicate pair by the filter if either their LCS-Similarity or TI-Similarity or both exceed a user-defined threshold, denoted as σ, and is treated as a non-duplicate pair otherwise. This relaxed definition of the duplicate record pair enables the filter to detect more potentially duplicate record pairs.

B. Transitive Closure Properties of Record Similarity

In this part, we will discuss the transitive closure properties of LCS-Similarity and TI-Similarity, which will be intensively used in various stages of PC-Filter to achieve a remarkable efficiency improvement. The Lower Bound (LB) and Upper Bound (UB) of the similarity between two records using the two metrics are unveiled in Lemma 1. The detailed proof can be referred to [13] and [17].

Lemma 1: When the distance measure satisfies the triangle inequity, the LB and UB of the similarity between two records A and C, denoted as $LB_B(A,C)$ and $UB_B(A,C)$, can be computed as follows using record B as an *Anchor Record*:

$$LB_B(A,C) = max(0, sim(A,B) + sim(B,C) - 1)$$

$$UB_B(A,C) = 1 - |sim(A,B) - sim(B,C)|$$

Note that, for sake of simplify, we will not distinguish which similarity metric is involved when the notion of

$sim()$ is used, but $sim(A, B)$ and $sim(B, C)$ should use the same similarity metric in Lemma 1.

Now, we will discuss the transitive closure properties utilized by PC-Filter, based on LB and UB of the similarity between two records using the two metrics. The properties are called Record-Record (R-R) Properties and Record-Partition (R-P) Properties. These properties provide heuristics for deciding the duplication or non-duplication between two records and between a record and a record partition, respectively.

Record-Record Properties (R-R Properties)

Record-Record Properties (R-R Properties) are the properties used for deciding duplication/non-duplication between two records using an Anchor Record (AR). Suppose we have three records, A, B and C. B is chosen as the Anchor Record. We have two R-R Properties regarding duplication/ non-duplication between records A and C.

Property 1: If $sim(A, B) + sim(B, C) - 1 \geq \sigma$, then A and C are duplicate records.

Property 2: If $1 - |sim(A, B) - sim(B, C)| < \sigma$, then A and C are non-duplicate records.

The correctness of Property 1 and 2 can be easily verified: if the lower bound of similarity between A and C exceeds the threshold, then A and C are duplicate and if the upper bound of similarity between A and C is less than the threshold, then A and C are not duplicate. Using record B as the AR, the duplication/non-duplication between records A and C can be decided immediately if they satisfy either Property 1 or 2, and the expensive record comparison using LCS-Similarity or TI-Similarity between A and C can thus be avoided.

Record-Partition Properties (R-P Properties)

Based on the two R-R Properties presented above, we devise the following R-P Properties (Properties 3-4) regarding the duplication/non-duplication decision for a single record and a partition of records. The notations used in R-P properties are given in Table I.

Notations	Meaning
AR	The Anchor Record of a record partition P
r_i	The record in the partition P
$MinSim$	Minimum similarity between AR and r_i
σ	The user-defined similarity threshold

TABLE I.
NOTATIONS USED IN R-P PROPERTIES

Property 3: If $sim(r, AR) + MinSim - 1 \geq \sigma$, then the record r is definitely duplicate with all the records residing in the partition P.

Proof: Since for any record r_i in the partition P, we have $sim(r, AR) + sim(r_i, AR) - 1 \geq sim(R, AR) + MinSim - 1$ then if $sim(r, AR) + MinSim - 1 \geq \sigma$, then we have $sim(r, AR) + sim(r_i, AR) - 1 \geq \sigma$. From Property 1, we can conclude that record r is duplicate with all the records residing in the partition P. Property 3 is proved. ■

Property 4: If $1 - min(|sim(r, AR) - sim(r_i, AR)|) < \sigma$, then r is definitely not duplicate with any records residing in the partition P.

Proof: We have $min(|sim(r, AR) - sim(r_i, AR)|) \leq |sim(r, AR) - sim(r_i, AR)|$. So $1 - min(|sim(r, AR) - sim(r_i, AR)|) \geq 1 - |sim(r, AR) - sim(r_i, AR)|$. If $1 - min(|sim(r, AR) - sim(r_i, AR)|) < \sigma$, then $1 - |sim(r, AR) - sim(r_i, AR)| < \sigma$, which means that every record in the partition satisfies Property 2. Thus we conclude that r is definitely not duplicate with any records in the partition P. Property 4 is proved. ■

The advantage of R-P Properties (Properties 3-4) is that they provide heuristics greatly facilitating the decision of duplication/non-duplication between a single record and a whole partition of records, without directly comparing this particular record with each record in the partition using LCS-Similarity or TI-Similarity.

The algorithms for deciding the duplication or non-duplication between two records (called RR_Duplicate) and between a record and a record partition (called RP_Duplicate) using R-R Properties and R-P Properties are presented in Figure 1 and 2, respectively. Note that in the algorithm of RR_Duplicate, *true* or *false* are returned when the two records are duplicate or non-duplicate. While in the algorithm of RP_Duplicate, "1" is returned when the record is duplicate with all the records in the partition, and "2" is returned when the record is not duplicate with any records in the partition and "3" is returned when the duplication/non-duplication can not be decided using Property 3 or 4.

Algorithm RR_Duplicate (r_1, r_2, σ)
1. IF $(sim(r_1, AR) + sim(r_2, AR) - 1 \geq \sigma)$ THEN
2. RETURN(true);
3. IF $(1 - |sim(r_1, AR) - sim(r_2, AR)| < \sigma)$ THEN
4. RETURN(false);
5. IF $(sim(r_1, r_2) \geq \sigma)$ THEN
6. RETURN(true);
7. ELSE RETURN(false);

Figure 1. Algorithm for deciding duplication/non-duplication of a record pair

Algorithm RP_Duplicate (r, P, σ)
1. IF $(sim(r, AR(P)) + MinSim(P) - 1 \geq \sigma)$ THEN
2. RETURN(1);
3. $NearestSim \leftarrow NNSearch(r, P)$;
4. IF $(1 - NearestSim < \sigma)$ THEN
5. RETURN(2);
6. RETURN(3);

Figure 2. Algorithm for deciding duplication/non-duplication between a record and a partition

Remarks. The naive transitive closure stipulates that two records are deemed to be duplicate when they are simultaneously duplicate with the third record. This does not necessarily hold in our transitive closure of similarity measures. Consider an example in which the similarity value between A and B and that between B and C are 0.8 and 0.9, respectively. The threshold σ is 0.75. In the naive transitive closure, A and C are considered to be duplicate. However, their duplication/non-duplication cannot be instantly decided as their similarity fall into the range of [0.7, 0.9] and, without actual comparison, it is not clear whether their similarity can really exceed the threshold.

C. Speed up the Evaluation of R-P Properties

The evaluation of R-P properties is non-trivial. Evaluating Property 3 requires the retrieval of the minimum similarity value between the AR and records in the partition, and evaluation of Property 4 involves searching the similarity value between AR and records in the partition that is closest to the similarity between AR and the given record r, which in fact is a Nearest Neighbor (NN) searching problem.

Since Property 3 and 4 need to be evaluated frequently in PC-Filter, it is worthwhile indexing the similarity list of each partition to perform the evaluation of Property 3 and 4 efficiently. In PC-Filter, the middle record (i.e. the record that locates in the middle of the partition) in each partition is chosen as the Anchor Record. After the database is sorted, the middle record in each partition can be reasonably thought of as the representative of the records in the partition. The similarity between each record in the partition and the AR is pre-computed and maintained in a sorted similarity list. The minimum similarity value between the AR and records in the partition is the first similarity value in the sorted list if the similarity values are organized in an ascending order and the NN searching problem can be performed efficiently using binary search on the sorted list with only a logarithmic cost.

IV. PC-FILTER

In this section, we will discuss in details our filtering framework, PC-Filter. PC-Filter performs duplicate record detection in 3 major steps as below: database sorting and partitioning, construction of Partition Comparison Graph (PCG) and partition comparison.

A. Database Sorting and Partitioning

Like most window-based methods, a key is first computed for each record in the database by extracting relevant fields or portions of fields for discriminating records. The whole database is sorted based on the chosen key. The selection of sort key is quite domain-specific and human subjective, and is carried out based on human's understanding of the discriminating power of the record fields. The sort key are expected to well, if not uniquely, discriminate records in the database. Due to the insensitivity of PC-Filter to the key chosen to sort the database, this step only has to to be performed once to sufficiently make PC-Filter to achieve its best possible effectiveness.

We then divide the sorted database into a number of sequential partitions. All partitions are set to have an equal size in our work. This can not only make database partitioning very fast but also help avoid skewing the computation workload of record comparison in each partition, which may seriously degrade the efficiency performance otherwise. We understand that setting all the partitions to be an equal size may lead to dividing the similar records, which are close with each other in the sorted database, into two or more neighboring partitions. However, thanks to the process of Inter-PC (will be discussed in the sequel), which detects duplicate record across some different partitions, we do not need to worry that the duplicate records possibly existing in these similar records cannot be detected. Specifically, we set the size of each partition to be s (a user-defined constant), thus we have $N = (p-1) * s + q$, where p is the number of partitions obtained and q is the number of records in the last partition, $0 \leq q \leq s$. The partitions to which a record r does not belong are called its *outer partitions*. There are a total of $(p-1)$ outer partitions for any record in the database.

Finally, the pre-processing work involves choosing the middle record in each partition as the Anchor Record of the partition and computing the similarity between each record in the partition and the AR. The similarity list is then sorted in order to provide efficient support to the evaluation of R-P Properties (Property 3 and 4) in PC-Filter.

B. Construction of Partition Comparison Graph (PCG)

In most cases, the outer partitions that needed to compare for records within the same partition actually falls into a relatively small range. Based on this observation, we will construct the *Partition Comparison Graph (PCG)* for the whole database such that the records in a particular partition will only be compared with the records of its immediate neighboring outer partitions in this graph rather than with all the partitions in the database.

We will first construct the *Partition Comparison Set (PCS)* for each partition, the set containing the sequence numbers of the outer partitions that each partition needs to be compared in Inter-PC and then convert them to PCG. To specify PCS, a few records, termed *Delimiting Records (DRs)*, will be randomly selected from each partition. A quantitative measurement, called *Partition Duplicate Degree (PDD)*, is computed based on the Delimiting Records selected to reflect the degree to which the records of two partitions are similar. Two outer partition comparison schemes, called the top-k outer partition comparison method and PDD+ outer partition comparison method, are also proposed. We will first introduce the definition of Partition Duplicate Degree as follows.

Definition 1. (Partition Duplicate Degree) Partition Duplicate Degree between two partitions p_1 and p_2 is defined as the ratio of the duplicate Delimiting Record pairs detected against the total number of Delimiting Record pairs of p_1 and p_2, i.e.

$$PDD(p_1, p_2) = \frac{|\{< dr_i, dr_j > | sim(dr_i, dr_j) \geq \sigma, dr_i \in p_1, dr_j \in p_2\}|}{N_{dr}(N_{dr} - 1)/2}$$

1) Top-k Outer Partition Comparison Method: The top-k outer partition comparison method involves comparing each partition with its outer partitions that have k highest PDD values with it. If a tie of PDD value occurs, we will select one or more partitions (with the tied PDD value) that are as close to the current partition as possible based on the sorting order to break the tie. This is because that the closer two partitions are from each other, the higher chance that they contain duplicate records. These k partitions are the k elements in the PCS of the current partition, which are also its k neighboring partitions in the corresponding PCG.

2) PDD+ Outer Partition Comparison Method: In the PDD+ outer partition comparison method, any partition with a positive PDD value with the current partition will be compared with it, meaning that an outer partition will be compared with the current partition as long as there exits at least one pair of duplicate Delimiting Records between them as we have the following mathematical equivalence: $PDD(p_1, p_2) > 0 \Leftrightarrow \exists dr_i \in p_1, \exists dr_j \in p_1, sim(dr_i, dr_j) \geq 0$.

Therefore, in order to know whether an outer partition is in the PCS of the current partition, we only have to check whether there is at least one duplicate Delimiting Record pair between them. Unlike the top-k outer partition comparison method where we have to evaluate all the Delimiting Record pairs of two partitions, the evaluation process in PDD+ Partition Comparison method can be early-stopping the moment a duplicate Delimiting Record pair has been found.

3) Conversion of PCS to PCG: Using PCS directly for partition comparison have some problems. First, PCS is not space efficient. PCS is by nature redundant: two partitions may appear in the PCSs of each other. Second, PCS is not a convenient way used to keep track of previous computation in order to prevent duplicate partition comparison. To solve these problems, we convert PCS to Partition Comparison Graph (PCG), an undirected graph which is more space economic and able to well keep track of previous computation that has been done in the comparison process.

Lemma 2: In PDD+ outer partition comparison method, $SequNo(p_1) \in PCS(p_2)$ iff $SequNo(p_2) \in PCS(p_1)$, where $SequNo(p)$ denotes the sequence number of partition p.

Proof: If $SequNo(p_1) \in PCS(p_2)$, then $PDD(p_1, p_2) > 0$, then $SequNo(p_2) \in PCS(p_1)$. Vice versa. ■

Note that Lemma 2 may not hold for top-k outer partition comparison method. For two partitions p_1 and p_2, it might be that $SequNo(p_1) \in PCS(p_2)$ but $SequNo(p_2) \notin PCS(p_1)$ simply because p_2 might no be in the top-k PDD list of p_1 even though p_1 is in the top-k PDD list of p_2.

Definition 2. (Partition Comparison Graph (PCG)) Partition Comparison Graph (PCG) is an undirected graph $G =< V, E >$, where V denotes the non-empty finite node set, representing all the partitions in the database, and E denotes the edge set. Two nodes (partitions) p_1 and p_2 are directly connected (i.e. adjacent) if

1) For top-k outer partition comparison method, we have $SequNo(p_1) \in PCS(p_2)$ or $SequNo(p_2) \in PCS(p_1)$ or both;
2) For PDD+ outer partition comparison method, we have $SequNo(p_1) \in PCS(p_2)$.

The order of any node in PCG in the top-k outer partition comparison method is k, so the PCG G is regular, with a degree of k as $degree(G) = max_{x \in V}\delta(x) = k$. In contrast, the order of any node in PCG will be in the range of $[0, p-1]$ in the PDD+ outer partition comparison method. A node x is called a singleton if $\delta(x) = 0$. A singleton in PCG is the node that is not adjacent with any other nodes in the graph, representing the partition whose PCS is $\emptyset$.

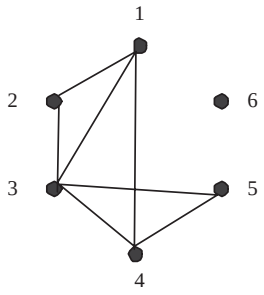

Figure 3. The corresponding Partition Comparison Graph (PCG)

Example. Let's suppose that we use PDD+ outer partition comparison method and there are 6 partitions in a database and their PCSs, from Partition 1 to 6, are {2, 3, 4}, {1, 3}, {1, 2, 4, 5}, {1, 3, 5}, {3, 4} and $\emptyset$. Their corresponding PCG is presented in Figure 3. The 6^{th} partition is an example of singleton node that are not adjacent with any other nodes in the PCG.

C. Selection Between the top-k and PDD+ outer partition comparison method

PDD+ outer partition comparison method features better effectiveness than the top-k outer partition comparison method because it can compare a partitions with all the outer partitions that potentially contain duplicates. However, it is possible that the resulting PCG of PDD+ outer partition comparison method is a complete graph, in which case the complexity will become $O(N^2)$. To solve this problem, we adopt the following strategy to select the suitable comparison method: First, k is set to be the logarithmic order of the number of partitions (i.e.

Algorithm Intra-PC (D, σ)
1. FOR each partition P in database D DO
2. FOR each pair of records r_1 and r_2 in partition P DO
3. IF RR_Duplicate $(r_1, r_2, \sigma) = true$ THEN
4. DuplicatesDetected(r_1, r_2);

Figure 4. Algorithm of Intra-PC

$k = log(\frac{N}{s})$). Second, if the average number of adjacent nodes in the PCG for PDD+ outer partition comparison method exceeds the value of k, we choose top-k outer partition comparison method, otherwise we use PDD+ outer partition comparison method. This strategy gives a speed performance guarantee while not compromising the effectiveness in most cases.

D. Partition Comparison

After we obtain the PCG of each partition in the database, partition comparison will be performed. The record comparison in PC-Filter involves an Intra-Partition Comparison process (Intra-PC) and an Inter-Partition Comparison process (Inter-PC).

1) Intra-PC: Intra-PC involves the record comparison in each of the partitions. R-R Properties (Property 1 and 2) are used in Intra-PC to avoid the record comparison of two records if they can satisfy either Property 1 or Property 2. Intra-PC is performed in 3 steps as follows (see Figure 4 for the detailed algorithm):

(1) Using Property 1 and 2, pair-wise similarity evaluations are performed among all the records within each partition. A record pair is detected duplicate (or not duplicate) if Property 1 (or Property 2) is satisfied;

(2) All the records of the partition, whose duplication or non-duplication decision cannot be made using Property 1 and 2, are compared using LCS-Similarity and TI-Similarity. Also, duplicate pairs of records are detected if their similarities exceed the similarity threshold;

(3) Repeat Step (1) and (2) until all partitions of the database have been compared internally.

One of the salient advantages of Intra-PC is that it is performed within each partition independently, therefore it inherently features very good parallelism. The equal-sized partitions can well balance the computation workload of each processor and ensure the best speed performance.

2) Inter-PC: Instead of finding duplicate records only within partitions, Inter-PC globalizes the record comparison to find duplicate records across some different partitions, if necessary. Based on the PCG constructed, Inter-PC will compare the records of the partitions that are directly connected in PCG. Given a record r, R-P Properties are used in this phase to instantly detect the partitions whose records are duplicate with r (using Property 3), or prune away the partitions that do not contain any duplicate records with r (using Property 4). For those neighboring partitions that satisfy neither Property 3 nor 4, we will evaluate r with each record of these partitions. Similar to the record comparison in Intra-PC, R-R Properties (Property 1 and 2) are used now to save the record comparison of r and the records in the neighboring partitions if the two records can satisfy either Property 1 or 2.

Algorithm Inter-PC(D, σ)
1. FOR each partition P in database D DO {
2. FOR each record r in partition P DO {
3. $Set\_of\_Neighbors \leftarrow PCG\_Neighbors(P)$;
4. For each partition Q in $Set\_of\_Neighbors$ DO
5. IF RP_Duplicate $(r, Q, \sigma) = 1$ THEN {
6. DuplicatesDetected (r, Q);
7. $Set\_of\_Neighbors- = Q$; }
8. ELSE IF RP_Duplicate $(r, Q, \sigma) = 2$ THEN
9. $Set\_of\_Neighbors- = Q$;
10. IF $(Set\_of\_Neighbors \neq \emptyset)$ THEN
11. FOR each partition Q in $Set\_of\_Neighbors$ DO
12. FOR each record $r^{'}$ in partition Q DO
13. IF RR_Duplicate $(r, r^{'}, \sigma) = true$ THEN
14. DuplicatesDetected$(r, r^{'})$; }
15. Delete P and its associated edges from the PCG;}

Figure 5. Algorithm of Inter-PC

Inter-PC performs 4 steps for the whole database (see Figure 5 for the detailed algorithm):

(1) For each record r, the similarities are computed between r and ARs of r's immediate neighboring outer partitions in the PCG. r is detected duplicate with all the records in an outer partition if Property 3 is satisfied for r and the outer partition. If Property 4 is satisfied for r and an outer partition, the outer partition, which does not contain any duplicate records with r, can be safely pruned;

(2) For those neighboring partitions that do not satisfy either Property 3 or 4, we will evaluate r with each record in these partitions. r is detected duplicate (or not duplicate) with a record in an outer partition if Property 1 (or Property 2) is satisfied for this pair of records. For those records that cannot be evaluated using Property 3 or 4, Inter-PC will perform detailed comparisons between r and these using LCS-Similarity and TI-Similarity;

(3) Repeat Step (1) and (2) until all the records in a partition have been evaluated. When a whole record partition has been evaluated, this partition will be deleted from the PCG, together with all the edges associated with this partition in the PCG.

(4) Repeat Step (1)-(3) until there are not any nodes left in the PCG.

V. Complexity Analysis

A. Notations

Let N be the number of records in the database, s be size of each partition, p be the number of partitions, N_{nei}

be the average number of neighboring outer partitions in PCG of each partition, a, b and c be the percentage of total record pairs that have to be compared using LCS-Similarity or TI-Similarity in PCG construction, Intra-PC and Inter-PC, respectively, $0 \leq a, b, c \leq 1$, C_{sim} be the average cost of comparison of a pair of records using LCS-Similarity or TI-Similarity and C_{ari} be the cost other than C_{sim} for arithmetic operations such as additions, deletions, multiplications, divisions and comparisons.

B. Complexity of the Major Steps

In database sorting, the cost will be $N * logN * C_{ari}$ for a database of N records.

In the pre-processing step, similarities between AR and other records within each partition will be computed, so the cost will be $N * C_{sim}$. These similarity values will be sorted within each of the partitions with a cost of $p * s * logs * C_{ari} = N * logs * C_{ari}$. The total cost for this step is thus approximately $N * C_{sim}$ given $s << N$ and $C_{ari} << C_{sim}$.

To construct PCS of each partition, similarities between each DR of the partition and ARs of its outer partitions will be computed, with a cost of $N_{dr} * p^2 * C_{sim}$. Then, pair-wise similarity evaluation among Delimiting Records of two partition pairs is performed to construct PCS of each partition, with a cost of $a * N_{dr}^2 * p^2 * C_{sim}$. Finally, one scan of the PCSs of all partitions is required in converting PCS to PCG, which requires $p * k * C_{ari}$ for the top-k outer partition comparison method and requires $p^2 * C_{ari}$ for the PDD+ outer partition comparison method at most as the maximum value of N_{nei} is p. In sum, the cost of constructing PCG for both methods can be simplified as $a * N_{dr}^2 * p^2 * C_{sim}$ given $C_{ari} << C_{sim}$.

In Intra-PC, the similarities of all record pairs will be examined in terms of Property 1 and 2, which requires a cost of $s^2 * C_{ari}$. The cost of comparing record pairs using LCS-Similarity or TI-Similarity will be $b * s^2 * C_{sim}$. In summary, the total cost to perform p such processes will be $p * (s^2 * C_{ari} + b * s^2 * C_{sim})$. Given $C_{ari} << C_{sim}$ and $N = p * s$, the cost of Intra-PC can be simplified as $p * b * s^2 * C_{sim} = b * (p * s^2) * C_{sim} = b * s * N * C_{sim}$.

In Inter-PC, if the top-k outer partition comparison method is used, then each record r will compare with the ARs in its k neighboring outer partitions in PCG, requiring a cost of $k * C_{sim}$. Inter-PC will then draw on R-P and R-R Properties to evaluate pairs of records, with a cost of $k * C_{ari}$ and $k * s * C_{ari}$, respectively. Finally, r will compare with the records in the partitions that have not been pruned using the similarity measure, with a cost of $c * k * s * C_{sim}$. In sum, the total cost for examining N records in inter-PC will be $N * (k * C_{sim} + k * C_{ari} + k * s * C_{ari} + c * k * s * C_{sim})$. Given $C_{ari} << C_{sim}$, the cost of Inter-PC using top-k outer partition comparison method can be simplified to $c * k * s * N * C_{sim}$. Similarly, the cost of Inter-PC using PDD+ outer partition comparison method can be simplified to $c * N_{nei} * s * N * C_{sim}$.

From the above analysis, we can see that (i) the cost of the two outer partition comparison methods only differs in the last step, and (ii) the value of s does not affect the complexity of the first two steps, thus we will only consider the effect of s on the total cost of the *last three steps*.

The total cost C for the last three steps, when $a = b = c = 1$ in the worst case, will be

$$C = (N_{dr}^2 * p^2 + N * s + N * k * s) * C_{sim}$$

for top-k outer partition comparison method, and

$$C = (N_{dr}^2 * p^2 + N * s + N * N_{nei} * s) * C_{sim}$$

for PDD+ outer partition comparison method.

The partition size s is chosen such that the number of record similarity computations using the similarity measurement can be minimized, i.e. $s^* = min_s C$. Plugging s^* into the cost function, we can obtain the minimum value of the total cost as:

$$C_{min} = N * (N_{dr}^2 * \frac{N}{s^{*2}} + s^*(k+1)) * C_{sim}$$

for top-k outer partition comparison method, and

$$C_{min} = N * (N_{dr}^2 * \frac{N}{s^{*2}} + s^*(N_{nei}+1)) * C_{sim}$$

for PDD+ outer partition comparison method.

C. Cost of PC-Filter Using Top-k Outer Partition Comparison Method

In the top-k outer partition comparison method, s is a constant specified by human users, thus the cost of the last three steps can be minimized to $1.89 * N_{dr}^{\frac{2}{3}} * (1+k)^{\frac{2}{3}} * N^{\frac{4}{3}}$ when $s = 1.26 * N_{dr}^{\frac{2}{3}} (1+k)^{-\frac{1}{3}} * N^{\frac{1}{3}}$. This analysis shows that the complexity of PC-Filter can be reduced to the order of $O(N^{\frac{4}{3}})$ by picking an optimized value of s for the top-k outer partition comparison method.

D. Cost of PC-Filter Using PDD+ Outer Partition Comparison Method

Unlike the top-k outer partition comparison method that the number of outer partitions needs to be compared for each node in PCG is of a fixed value k, N_{nei} in PDD+ outer partition comparison method may change from 0 to $\frac{N}{s} - 1$. In our work, we devise the following statistical model to estimate N_{nei}, which is important in analyzing the total cost of the algorithm using PDD+ outer partition comparison method.

Let φ denote the duplicate ratio of the database and ρ denote the probability that a give record pair is duplicate. Therefore we have the following approximation $\rho = \varphi^2$. The probability that a record pair is not duplicate is $1 - \rho$, then the probability that none of the Delimiting Records of two partitions are not duplicate is $(1-\rho)^{N_{dr}^2}$, then the probability that two partitions are considered to contain duplicate records (i.e. immediately connected in PCG) is $1 - (1-\rho)^{N_{dr}^2}$.

Let $t = 1-(1-\rho)^{N_{dr}^2}$, thus the expected value of neighbors of each partition in PCG can be computed as:

$$E(N_{nei}) = \frac{t+2t^2+3t^3+\cdots+(\frac{N}{s}-1)t^{\frac{N}{s}-1}}{t+t^2+t^3+\cdots+t^{\frac{N}{s}-1}}$$

$$= \frac{1}{1-t} - \frac{(\frac{N}{s}-1)t^{\frac{N}{s}-1}}{1-t^{\frac{N}{s}-1}},\ t\neq 0, t\neq 1, N\neq s$$

If $t=1$, then $E(N_{nei}) = \frac{N}{s}-1$. If $t=0$, then $E(N_{nei}) = 0$. If $N=s$, then $E(N_{nei}) = 0$.

Plugging $E(N_{nei}) = \frac{1}{1-t} - \frac{(\frac{N}{s}-1)t^{\frac{N}{s}-1}}{1-t^{\frac{N}{s}-1}}$ into the cost function of the last three steps, we can obtain

$$C = (N_{dr}^2 * p^2 + N*s + N*s*(\frac{1}{1-t} - \frac{(\frac{N}{s}-1)t^{\frac{N}{s}-1}}{1-t^{\frac{N}{s}-1}}))*C_{sim}$$

E. Parameter Specification

In this subsection, we will elaborate on the specification of the partition size s and k in top-k outer partition comparison method, the partition size s in PDD+ outer partition comparison method, and the number of Delimiting Records, N_{dr}, for both top-k and PDD+ outer partition comparison methods.

1) Specification of Partition Size s and k in top-k Outer Partition Comparison Method: If we let $k = log\frac{N}{s}$, then based on $s = 1.26*N_{dr}^{\frac{2}{3}}(1+k)^{-\frac{1}{3}}*N^{\frac{1}{3}}$, we can get $k = log(0.79*N_{dr}^{-\frac{2}{3}}*N^{\frac{2}{3}}*(1+k)^{\frac{1}{3}})$. We can approximate k as $k \approx log(0.79*N_{dr}^{-\frac{2}{3}}*N^{\frac{2}{3}})$. Plugging this into $s = 1.26*N_{dr}^{\frac{2}{3}}(1+k)^{-\frac{1}{3}}*N^{\frac{1}{3}}$, we can get the optimal value of s as $s = 1.26*N_{dr}^{\frac{2}{3}}(1+log(0.79*N_{dr}^{-\frac{2}{3}}*N^{\frac{2}{3}}))^{-\frac{1}{3}}*N^{\frac{1}{3}}$.

2) Specification of Partition Size s in PDD+ Outer Partition Comparison Method: It is impossible to explicitly compute the value of s by letting the first derivative of cost function w.r.t s to be equal to 0. In addition, the cost function does not only have one local minima. Therefore, using the first derivative alone may not be sufficiently to find the global minimum of the cost function. We alternatively choose to adopt a search method to find the optimal value of s such that the cost function can be minimized.

Since the lower bound for searching the optimal partition size s^* is N_{dr} (each partition should at least contains N_{dr} records), thus the searching space for optimal value of s will be $[N_{dr}, N]$. The exhaustive search for the optimal value of s involves evaluating each integer from N_{dr} to N against the cost function, requiring a complexity of $O(N)$. This search method is obviously not scalable for large datasets. Therefore, we will explore the upper bound for searching the optimal value of s so that the search cost can be remarkably reduced. We will first prove the following two lemmas.

Lemma 3. Given a database with a fixed size of N, the optimal value of s, s^*, will increase as ρ decreases.

Proof: Intuitively, when the probability that two records are duplicate decreases, the expected value of immediately connected neighbors of each partition in PCG will decrease (or will remain unchanged). That is, N_{nei} will be non-increasing when ρ decreases. Given the fixed values of N and N_{dr}, we can consider the cost function $C = (N_{dr}^2 * k^2 + N*s + N*N_{nei}*s)*C_{sim}$ as the function of s and N_{nei}, thus the cost can be minimized to $1.89*N_{dr}^{\frac{2}{3}}*(1+k)^{\frac{2}{3}}*N^{\frac{4}{3}}$ when $s^* = 1.26*N_{dr}^{\frac{2}{3}}(1+k)^{-\frac{1}{3}}*N^{\frac{1}{3}}$. We can see that when N_{nei} decreases, the optimal value of s, s^*, will increase. Thus, s^* will increase as ρ decreases. ■

Lemma 4. The upper bound of searching the optimal value of s is $(2*N_{dr}^2*N)^{\frac{1}{3}}$.

Proof: Based on Lemma 1, we can see that the maximum integer required to be evaluated in the search is the asymptotic value of s when ρ is approaching to 0. When is ρ approaching to 0, we have

$lim_{\rho\to 0}C = lim_{t\to 0}C = lim_{N_{nei}\to 0}C = N*(N_{dr}^2*\frac{N}{s^2}+s)*C_{sim}$, which can be minimized to $(N_{dr}^2+1)*N^{\frac{1}{3}}*C_{sim}$ when $s = (2*N_{dr}^2*N)^{\frac{1}{3}}$. Therefore $(2*N_{dr}^2*N)^{\frac{1}{3}}$ is the upper bound of s^*.

Furthermore, we will show that $(2*N_{dr}^2*N)^{\frac{1}{3}} > N_{dr}$, which ensures that the possible values for optimal partition size will exceed the number of Delimiting Record in the partition. Since $N_{dr}\leq s$ and $s\leq N$, thus $N_{dr}\leq N$. Therefore, we have $(2*N_{dr}^2*N)^{\frac{1}{3}} \geq (2*N_{dr}^2*N_{dr})^{\frac{1}{3}} = 1.26*N_{dr} > N_{dr}$. ■

Based on Lemma 3 and 4, we establish that it is only necessary to evaluate the integers from N_{dr} to $(2*N_{dr}^2*N)^{\frac{1}{3}}$, thus the complexity of searching the optimal partition size can be reduced from $O(N)$ to $O(N^{\frac{1}{3}})$.

We further verify the correctness of our lemma by calculating the optimal partition size under 28 different dataset sizes (ranging from 10^3 to 10^6) and three varied values of ρ and the corresponding upper bounds (i.e. $(2*N_{dr}^2*N)^{\frac{1}{3}}$) in our model. The results are shown in Figure 6. Figure 6 graphically shows that the upper bound we establish theoretically provides sufficient searching scope for the optimal partition size for different dataset sizes and ρ values.

3) Specification of the Number of Delimiting Records N_{dr}: The value of N_{dr} should be as small as possible to lower the cost of specifying the PCS of the partitions. However, when the value of N_{dr} is too small, the calculation of PDD of two partitions will not be accurate enough. To seek a good trade-off, we utilize the hypothesis testing method to find a good value for N_{dr}. We assume that the similarities between the Delimiting Records and the Anchor Record in each partition satisfy normal distribution and devise a technique to pick up Delimiting Records based on the construction of confidence interval of normal mean. A specific number of Delimiting Records are randomly selected from each partition such that the average similarity value between the Delimiting Records and the Anchor Record in each partition will be falling into the confidence interval with a certain offset from the actual midvalue of the similarities between the records and Anchor Record in the partition.

Specifically, to obtain a $(1-\alpha)$-confidence interval, the

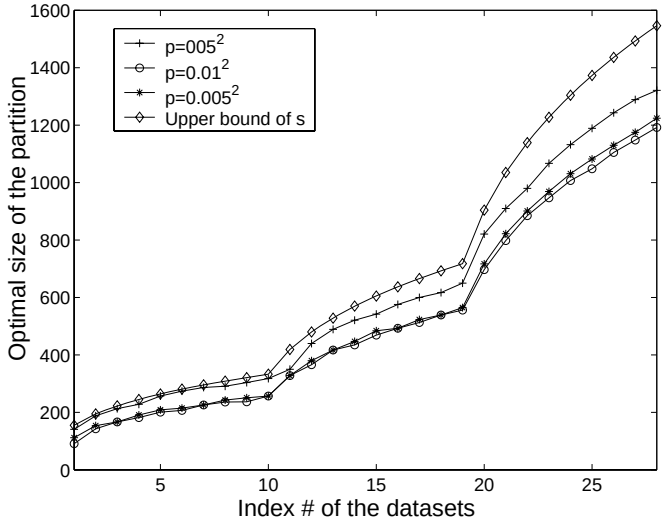

Figure 6. Optimal size of partition under varying dataset sizes

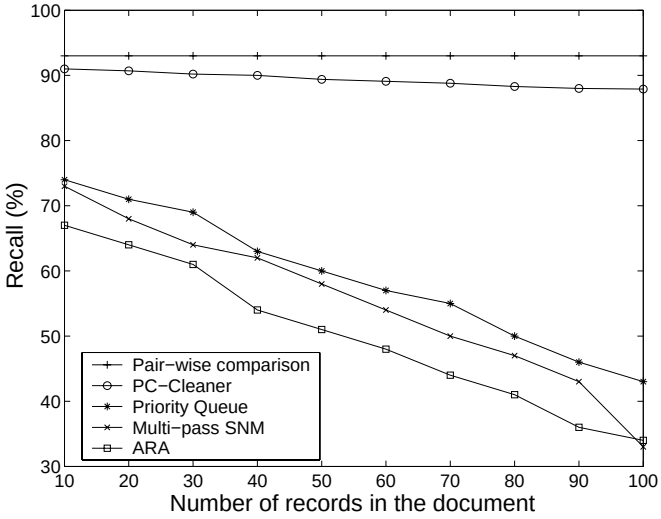

Figure 8. Recall results when varying size of database

minimum size of a random sample from each partition, used as DRs, is given as follows [10]:

$$N_{dr}^{*} = [\frac{t_{\alpha/2} * \sigma^{'}}{\delta^{*}}]^{2}$$

where $\sigma^{'}$ denotes the estimated standard deviation of the similarity values between the reocrds and AR in the partition and δ^{*} denotes the half-width of the confidence interval.

Using the above theorem, we select 43 DRs from each partition in PC-Filter so that one can expect that a confidence interval for the mean similarity value between the records and AR in the partition would be established at the 90% probability level, which would have limits about $\pm 1\%$ from the midvalue of the confidence interval.

VI. Experimental Results

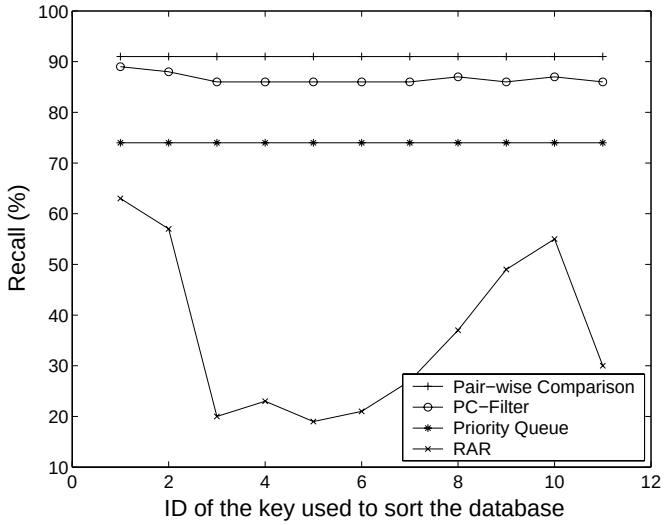

Figure 7. Recall results when varying the keys

Extensive experiments have been conducted to evaluate the effectiveness and efficiency of PC-Filter in duplicate record detection/filtering. We investigate the performance of PC-Filter on a large real-world company database and large synthetic customer databases. The real-life company dataset has 100,000 records and each of the records has 7 fields. The large synthetic dataset is generated by a record generator, through which we can specify the dimension of the record, the number of clean records and duplicate records.

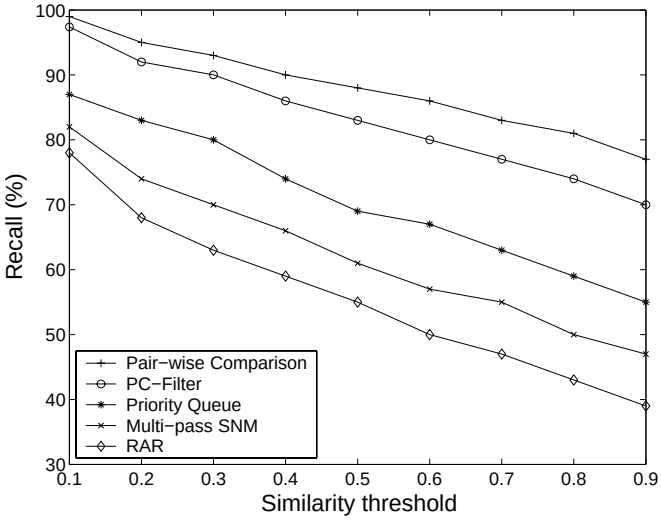

Figure 9. Recall results when varying threshold value

In our experiments, the pair-wise comparison method, Multi-pass SNM, Priority Queue method and RAR are used for comparative study on the performance in duplicate record detection. Even though practically infeasible, the pair-wise comparison is the most naive yet the most effective method. SNM is the basic but probably the most widely used method. Multi-pass SNM improves SNM by performing multiple sorting and scanning of the database. Priority Queue method, a typical clustering-based method, clusters each record into the clusters that are organized in a priority queue and merge records in each cluster. RAR explicitly uses the transitive closure

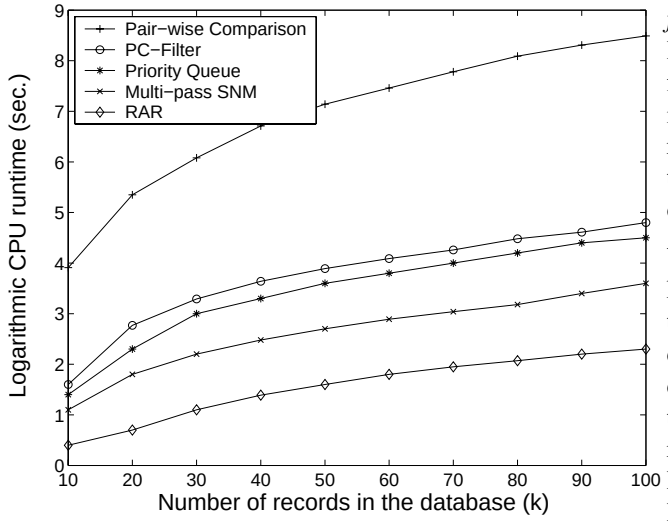

Figure 10. Logarithmic CPU runtime

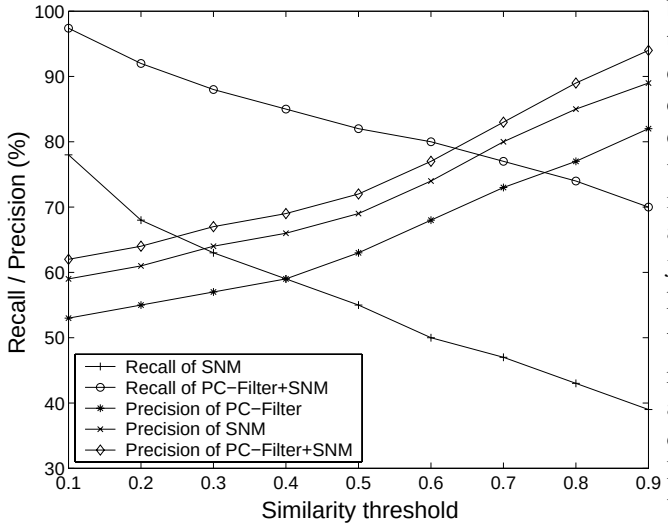

Figure 11. Recall-Precision graph of SNM and PC-Filter+SNM

property of TI-Similarity to realize fast duplicate record detection under the framework of SNM. Since PC-Filter uses two similarity metrics, thus for a fair comparison, all the methods involved in the comparative study use LCS-Similarity and TI-similarity to measure the similarity of records and averge the performance. ω, the size of the window used in multi-pass SNM and RAR is set to be 10.

A. *Effectiveness of PC-Filter*

The first part will investigate the effectiveness of PC-Filter. We will evaluate various methods against the criteria of *recall* that measures the completeness of the detection result. In the context of duplicate record detection, recall is mainly decided by the scope of record comparison/matching while precision is largely depended on the robustness of the similarity metric in accurately measuring the true similarity betwwen two records. *The precisions of these methods will not remarkably differ from each other as long as the same metric is used.* However, the recalls of these methods, as you will see from the experiments in this section, may differ by a large margin since different methods specify varied scopes for record comparison. Therefore, we will mainly focus on the effectiveness analysis of different methods in terms of recall.

Varying Key for Sorting the Database. We vary the key to sort the database in order to evaluate its effect on the recalls of different methods. Specifically, we sort the database based on each of the 11 fields in the synthetic dataset and compute the corresponding recall levels. All the keys are shown in Figure 7. We can see that the recalls of PC-Filter, pair-wise comparison method and Priority Queue method have very stable recalls while RAR is very sensitive to the sort key. The pair-wise comparison method and Priority Queue method do not require database sorting, thus their recalls are not affected by the sort key at all. Using a fix-sized window in RAR, the recall will be relatively higher when using more discriminating keys to let the truly duplicate records locate close to each other, but will be lower when using less discriminating ones in which the truly duplicate records will probably stay far apart from each other. PC-Filter is able to compare two records even when they are far apart from each other in the sorting list. This experiment justifies that PC-Filter is able to solve the "Key Selection" problem the existing methods suffer.

Varying Number of Records in the Database. The recall results when varying the number of records in database are shown in Figure 8. As these results show, PC-Filter outperforms Multi-pass SNM, Priority Queue method and RAR in terms of recall by a large margin for databases with different sizes. The recall level of PC-Filter is very close to that of the pair-wise comparison method. One of the salient advantages of PC-Filter is its ability to reliably achieve high recall levels even when the size of the dataset varies, while Multi-pass SNM, Priority Queue method and RAR feature a decline in recall when the size of the dataset increases. As database size increases, the distance of duplicate record pairs in the sorted dataset stands a higher chance to exceed the window size, in which case Multi-pass SNM and RAR will fail to detect these duplicate pairs due to the limited size of the window. For the Priority Queue method, the accuracy of the cluster center in representing all the points in the cluter will also be compromised when the database size increases, leading to a decline in recall. This problem is obviated in PC-Filter by comparing records globally among necessary outer partitions.

Varying Similarity Threshold σ. Next, we will study the recall of various methods under different similarity thresholds. The threshold is varied from 0.1 to 0.9, with 0.1 increments each time. The result of recall is shown in Figure 9. As shown in Figure 9, the recall of all the five methods will be decreased as σ increases. However, for a given σ, the corresponding recall of PC-Filter is very close

to that of the pair-wise comparison method and is much higher than those of other three methods. This is because PC-Filter uses the Inter-PC process to compare records across some partitions in the database, so the detection result of PC-Filter contains more truly duplicate records than the results of the other three methods.

Using Different Transitive Closure Properties. We evaluate the robustness of the transitive closure properties of record similarity used in PC-Filter. We compare the effectiveness of the naive and similarity-based transitive closure. On the real-life dataset, the recall and precision of PC-Filter using the similarity-based transitive closure properties are approximately 90% and 85%, while the recall and precision of PC-Filter using the naive transitive closure are only 79% and 74%. The relatively low recall and precision when using the naive transitive closure is because that the detector not only includes many false-positives into the final result and but also excludes many true-negatives from the final result. Clearly, the transitive closure properties of record similarity utilized in PC-Filter are more robust.

B. Efficiency of PC-Filter

We also explore the efficiency of PC-Filter against other four methods. Figure 10 shows the logarithmic CPU runtime of different methods. The CPU time of PC-Filter is higher than the time of Multi-pass SNM $(O(k*\omega*N))$ and RAR $(O(\omega * N))$, but somewhat comparable to the time of Priority Queue method $(O(NlogN)$ and significantly less than the time of the pair-wise comparison method $(O(N^2))$. By taking advantage of the transitive closure properties of record similarity, PC-Filter saves a noticeable amount of expensive record comparisons and therefore is able to well scale to large databases.

C. Effectiveness of PC-Filter+X

In this part, we will experimentally show that the framework of PC-Filter+X, i.e. incorporating PC-Filter into another detection method X with a different similarity measure, such as Edit Distance, will enable us to achieve better recall and precision than only using X. The result is shown in Figure 11. Recall that the major role of PC-Filter is to return a relatively small set of duplicate records efficiently with a high recall level. However, there may be many false-positives in this result, thus the method X is used to refine the result by pruning away these false-positives from the result. In our experiment, we choose X as SNM using Edit Distance and compare the effectiveness of PC-Filter+SNM and SNM. The recall-precision graph is presented in Figure 13. We can see, from the figure, that (i) The recall of PC-Filter+SNM is much higher than SNM, this is because PC-Filter outperforms SNM in terms of recall; (ii) The precision of the result of PC-Filter is slightly lower than that of SNM. This is understandable since when more record are needed to compare, there will be a higher chance for some false-positives to be included in the result. (iii) The precision of the result of PC-Filter+SNM is, however, higher than that of SNNM. This is because using three similarity measures, LCS-Similarity and TI-Similarity in PC-Filter and Edit Distance in SNM, PC-Filter+SNM can be more effective in detecting duplicate records than only using one kind of similarity measure. Put simply, by incorporating PC-Filter to SNM, we can achieve both better recall and precision performance than performing SNM alone.

VII. Conclusions

To solve a number of critical problems the existing duplication detection methods suffer, we propose an effective and efficient filtering technique, called PC-Filter, in this paper for duplicate record detection in large databases. In PC-Filter, data partitions will be internally and externally compared in order to detect duplicate records. We utilize the transitive closure of record similarity and devise four properties, used as heuristics, based on such transitive closure to achieve a remarked efficiency of the filter. The partition size is well specified such that the time complexity of PC-Filter can be minimized. Extensive experimental results verify the better effectiveness and efficiency of PC-Filter than the existing methods.

Acknowledgement

We would like to acknowledge Prof. Ken Sevcik from University of Toronto, Prof. Tok Wang Ling from National University of Singapore and Robert M. Bruckner from Microsoft, Seattle, USA and Mr. Han Liu from John Hopkins University, USA for their useful thoughts and comments on this paper.

References

[1] R. Ananthakrishna, S. Chaudhuri and V. Ganti. Eliminating Fuzzy Duplicates in Data Warehouses. In Proceedings of *VLDB'02*, pages 586-597, Hong Kong, China, 2002.

[2] P. Andritsos, R. J. Miller and P. Tsaparas. Information-Theoretic Tools for Mining Database Structure from Large Data Sets. In Proceedings of *ACM SIGMOD'04*, pages 731-742, Paris, France 2004.

[3] M. Bilenko and R. J. Mooney. Adaptive Duplicate Detection Using Learnable String Similarity Measures. In Proceedings of *SIGKDD'03*, pages 39-48, Washington, DC, USA, 2003.

[4] S. Chaudhuri, K. Ganjam, V. Ganti and R. Motwani. Robust and Efficient Fuzzy Match for Online Data Cleaning. In Proceedings of *ACM SIGMOD'03*, pages 313-324, San Diego, USA, 2003.

[5] L. P. English: Improving Data Warehouse and Business Information Quality. J. Wiley and Sons, New York, 1999.

[6] L. Gravano, P. G. Ipeirotis, N. Koudas, D. Srivastava: Text Joins for Data Cleansing and Integration in an RDBMS. In Proceedings of *ICDE'03*, pages 729-731, Bangalore, India, 2003.

[7] M. Hernandez. A Generation of Band Joins and the Merge/Purge Problem. Technical Report CUCS-005-1995, Columbia University, Feb 1995.

[8] M. A. Hernandez and S. J. Stolfo. The Merge/Purge Problem for Large Databases. In Proceedings of the *ACM SIGMOD'95*, pages 127-138, San Jose, California, 1995.

[9] W. L. Low, M. L. Lee and T. W. Ling. A Knowledge-Based Framework for Duplicates Elimination. In *Information Systems: Special Issue on Data Extraction, Cleaning and Reconciliation*, Volume 26, Issue 8, Elsevier Science, 2001.

[10] A. E. Mace. *Sample-size Determination*. Reinhold Publishing Corporation, New York, 1964.

[11] A. E. Monge and C. P. Elkan. An Efficient Domain-independent Algorithm for detecting Approximately Duplicate Database Records. In Proceedings of *SIGMOD Workshop on Research issues and Data Mining and Knowledge Discovery*, Tucson, Arizona, 1997.

[12] A. E. Monge and C. P. Elkan. The Field Matching Problem: Algorithms and Application. In Proceedings of *SIGKDD'96*, pages 267-270, Portland, USA, 1996.

[13] Z. Li, S. Y. Sung, P. Sun and T. W. Ling, A New Efficient Data Cleansing Method. In Proceedings of *DEXA'02*, pages 484-493, Aix-en-Provence, France, 2002.

[14] E.J Otoo, D, Rotem, and S. Seshadri. Efficient Algorithm for Multi-file Caching. In Proceedings of *DEXA'04,*, pages 707-719, Zaragoza, Spain, 2004.

[15] E. Rahm and H. H. Do. Data Cleaning: Problems and Current Approaches. In *IEEE Bulletin on Data Engineering*, 2000.

[16] T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. In *Journal of Molecular Biology*, 147, pages 195-197, 1981.

[17] S. Y. Sung, Z. Li and S. Peng. A Fast Filtering Scheme for Large Database Cleansing. In Proceedings of *CIKM'02*, pages 76-83, 2002.

[18] Z. Tian, H. Lu, W. Ji, A. Zhou and Z. Tian: An N-gram-based Approach for Detecting Approximately Duplicate Database Records. In *International Journal on Digital Libraries*, 3(4): 325-331, 2002.

[19] M. Weis and F. Naumann: Detecting Duplicate Objects in XML Documents. In Proceedings of *IQIS'04*, pages 10-19, Paris, France, 2004.

[20] Ji Zhang, Tok Wang Ling, Robert. M. Bruckner, Han Liu. PC-Filter: A Robust Filtering Technique for Duplicate Record Detection in Large Databases. In *DEXA'04*, Zaragoza, Spain, 2004.