PC-Filter: A Robust Filtering Technique for Duplicate Record Detection in Large Databases

Ji Zhang[1] Tok Wang Ling[2] Robert. M. Bruckner[3] Han Liu[1]

[1]Department of Computer Science, University of Toronto,
Toronto, Ontario, M5S 3G4, Canada
{*jzhang, hanliu*}*@cs.toronto.edu*
[2]School of Computing, National University of Singapore, Singapore 117543
lingtw@comp.nus.edu.sg
[3]Microsoft Research, Redmond, WA, USA
robruc@microsoft.com

Abstract: In this paper, we will propose PC-Filter (PC stands for Partition Comparison), a robust data filter for approximately duplicate record detection in large databases. PC-Filter distinguishes itself from all of existing methods by using the notion of partition in duplicate detection. It first sorts the whole database and splits the sorted database into a number of record partitions. The Partition Comparison Graph (PCG) is then constructed by performing fast partition pruning. Finally, duplicate records are effectively detected by using internal and external partition comparison based on PCG. Four properties, used as heuristics, have been devised to achieve a remarkable efficiency of the filter based on triangle inequity of record similarity. PC-Filter is insensitive to the key used to sort the database, and can achieve a very good recall level that is comparable to that of the pair-wise record comparison method but only with a complexity of O($N^{4/3}$). Equipping existing detection methods with PC-Filter, we are able to well solve the "Key Selection" problem, the "Scope Specification" problem and the "Low Recall" problem that the existing methods suffer from.

1 Introduction

Data cleaning is of crucial importance for many industries over a wide variety of applications. 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. The naïve 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 complexity, various techniques have been proposed. We can broadly classify the state-of-the-art methods into two major categories: *window-based methods* [4, 5, 9, 10] and *clustering-based methods*[7]. The window-based methods typically sorts the whole database based on a key and use the notion of *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.

Though they can, to some different degree, improve the efficiency of detecting duplicate records, the existing methods suffer from three major problems: the *"Key Selection"* problem, the *"Scope Specification"* problem and the "*Low Recall*" problem. Firstly, methods involving database sorting are very sensitive to the key chosen to sort the database. 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. Given the typically small size of the window or clusters, the existing methods cannot produce results with very satisfactory recall level.

In order to solve the abovementioned problems, we will propose PC-Filter, a novel filtering technique for effective and efficient duplicate record detection. The major contribution of our proposed PC-Filter is that it is able to solve the three major that the existing methods suffer from. Specifically:

(1) PC-Filter uses the notion of partition instead of a window or cluster to detect duplicate records in a database. PC-Filter will not only compare records within each partition, but also compare records across different partitions. This strategy makes it possible for PC-Filter to detect duplicate records even when they are located far apart from each other under a sorting based on an improperly chosen key. Therefore, the result of PC-Filter is insensitive to the sorting order of the database under different keys and a single sorting of the whole database will suffice in our work.

(2) By using a process called Inter-Partition Comparison, PC-Filter is able to globalize the scope of record comparison in the whole database. Therefore, even the duplicate record pair that is far apart from each other in the database can be detected. PC-Filter is able to achieve a very good recall that is even comparable to the pair-wise comparison method.

(3) The size of the partition has been optimized in our work to give the best possible speed performance.

2 Related Work

[5] has proposed the *Sorted Neighborhood Method (SNM)*, serving as the basis of many existing duplicate detection methods. SNM is able to speed up the data cleaning process by only examining neighboring records that is in the sliding window for a relevant record. Among the variants of SNM are *Duplicate Elimination SNM (DE-SNM)* [4], *Multi-pass-SNM* [4], *Clustering-SNM* [5] and *SNM-IN/OUT* [9]. 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 the 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 merging process is performed independently for every cluster using SNM. *SNM-IN/OUT*, which is probably the most related method to our PC-Filter, uses several properties, namely the lower and upper bounds of the Longest Common Subsequence (LCS) and Anchor Record (AR), to save record comparisons in SNM without impairing accuracy. The *priority Queue method* [7] clusters the records and uses a priority of set of 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.

In addition to the above window-based and clustering-based methods, [3] presents an on-the-fly detection scheme for detecting duplicates when joining multiple tables. This method, however, is not directly applicable for detecting duplicates in a single table where no join operation will be involved. A fuzzy duplicate match method is also proposed to detect duplicate for online data in [1] and [2]. A knowledge-based method is proposed in [6] based on a number of rules. However, using rules to detect duplicates in large database is rather expensive.

3 Measurements of Record Similarity

The fields of records are treated as strings and we utilize the similarity measure used in [10] in our work.

Field Similarity: Suppose a field F in record A has the character set $A_F=\{x_1, x_2, ..., x_m\}$ and the corresponding field in record B has the character set $B_F=\{y_1, y_2, ..., y_n\}$, where x_i, $1 \le i \le m$, and y_j, $1 \le j \le n$, are characters with association numbers. For example, for a string "abcabd", it is transferred to the character set {a1, b1, c1, a2, b2, d1}. The field similarity of the field F for A and B is computed as:

$$Sim_F(A,B) = \min\left(\frac{|A_F \cap B_F|}{|A_F|}, \frac{|A_F \cap B_F|}{|B_F|}\right)$$

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

$$Sim(A,B) = \sum_{i=1}^{n} (W_i * Sim_{F_i}(A,B))$$

Two records are treated as duplicate pair if their similarity value exceeds a user-defined threshold, denoted as σ.

Next, we will discuss the transitive closure properties of TI-Similarity, which will be intensively used in generation of Partition Comparison Graph and partition comparison in PC-Filter to achieve a remarkable efficiency improvement. The *Upper Bound* (*UB*) and *Lower Bound* (*LB*) of the similarity between two records using TI-Similarity is unveiled in Lemma 1 by [10].

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:

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

The properties utilized by PC-Filter, based on *LB* and *UB* of the similarity between two records using TI-Similarity 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.

(1) 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 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* $sim(A,B)+sim(B,C)-1 \ge \sigma$, *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.*

Using record *B* as the AR, the duplication/non-duplication between records *A* and *C* can be decided immediately if they satisfy either Rule 1 or 2, and the expensive record comparison between *A* and *C* can thus be avoided.

(2) 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.

***Property 3**: If $sim(R, AR)+MinSim-1 \geq \sigma$, then R is definitely duplicate with all the records residing in the partition, where MinSim is the minimum similarity value between AR and records in the partition.*

Proof: Since for any record in the partition R_i, 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 record residing in the partition. Thus Property 3 is proved. ■

***Property 4:** For every record R_i in the partition, if $(1-min\{| sim(R, AR)- sim(R_i, AR)|\}) < \sigma$), then R is definitely not duplicate with any records residing in the partition.*

Proof: For each record R_i in the partition, 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 *R* is definitely not duplicate with any records residing in the partition. 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 comparing this particular record with each record in the partition using TI-Similarity.

4 PC-Filter

PC-Filter performs duplicate record detection in 3 steps described below: database sorting and partitioning, construction of Partition Comparison Graph (PCG) and partition comparison. The overview of PC-Filter is given in Figure 1.

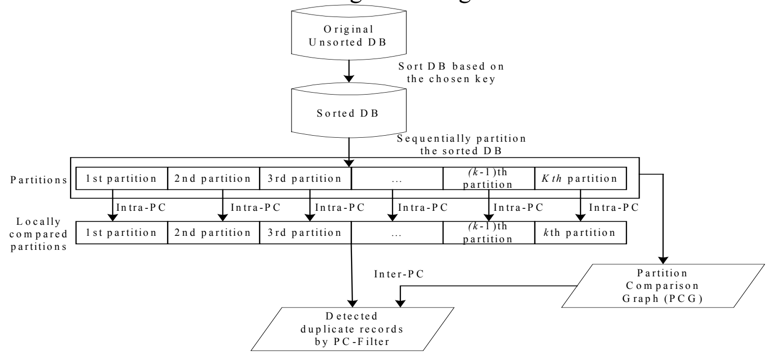
Original Unsorted DB
Sort DB based on the chosen key
Sorted DB
Sequentially partition the sorted DB
Partitions
1st partition
2nd partition
3rd partition
...
(k-1)th partition
Kth partition
Intra-PC
Locally compared partitions
kth partition
Inter-PC
Partition Comparison Graph (PCG)
Detected duplicate records by PC-Filter

Figure 1. Overview of PC-Filter

(1) Database Sorting and Partitioning

A key is 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. We then divide the sorted database into k sequential partitions. To perform fast database partition and balance the computation load of record comparison in each of the partitions, all partitions are set to have the same size in our work. The partitions to which a record R does not belong are called the *outer partitions* of R.

Other pre-processing work involves choosing the median record in each partition as the Anchor Record of the partition and computing the similarity between each record in the partition and the AR that will be maintained in a similarity list. The similarity list is sorted to provide efficient support to the evaluation of Property 3 and 4 in PC-Filter.

(2) Construction of Partition Comparison Graph (PCG)

It is observed that, 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.

Partition Comparison Range (PCR) for each partition is first constructed and then converted to Partition Comparison Graph (PCG). To construct PCR, the first and last $N_{dr}/2$ records in a partition will be selected as the *Delimiting Records (DRs)*, the records used to delimit the PCR of this partition. N_{dr} is the number of DRs specified by users and should be an even number. To construct the PCR of a partition, pruning of outer partitions using Property 4 is performed based on each of the DRs, whereby the lower and upper bounds of the range can be specified. More precisely, let $F_1, F_2, \ldots, F_{Ndr/2}$ be the first $N_{dr}/2$ records in a partition P and $L_1, L_2, \ldots, L_{Ndr/2}$ be the last $N_{dr}/2$ records in this partition. These N_{dr} records constitute the *DRs* of this partition. Each *DR* is associated with a set of outer partitions that are left after pruning using Property 3, denoted as $PartList(F_1), \ldots, PartList(F_{Ndr/2})$, and $PartList(L_1), \ldots, PartList(L_{Ndr/2})$. For a Delimiting Record *dr,* we have

$PartList(dr)=I- \{P_{dr}\} - \{P \mid P$ is an outer partition of dr, and P and dr satisfy Property 3$\}$

where I denotes the complete set of partitions in the database and P_{dr} denotes the partition to which *dr* belongs. The lower and upper bounds of the PCR of a partition are determined by the non-empty post-pruning partition sets of the first and last $N_{dr}/2$ DRs in the partition, respectively. Specifically, let k be the number of DRs that have non-empty post-pruning partition sets, where $0 \leq k \leq N_{dr}$. There are 3 cases in constructing the PCR of a partition:

(1) If $k=N_{dr}$, then the lower and upper bounds of the PCR of the partition P, denoted as $Range_{LB}(P)$ and $Range_{UB}(P)$, are defined as

$$PCR_{LB}(P)=min\{min\{PartList\,(F_1)\}, \ldots, min\{PartList(F_{Ndr/2})\}\}$$
$$PCR_{UB}(P) = max\{max\{\,PartList\,(L_1)\}, \ldots, max\{PartList\,(L_{Ndr/2})\}\}$$

The *max*() and the *min*() functions return the maximum and minimum *sequence number* of partitions in the sorted order;

(2) If $0<k<N_{dr}$, the lower and upper bounds of the PCR of the partition P are specified in a similar way to (1), but they will only use the non-empty partition sets of the DRs;

(3) If k=0, then all the partition sets of all DRs are empty, thus we write the PCR of this partition as [\$, \$], where "\$" is a special symbol.

Definition 1: *Partition Comparison Graph (PCG)* is an undirected graph G=<V, E>, where V denotes the node set, representing all the partitions in the database, and E denotes the edge set. Two nodes (partitions) are directly connected if the sequence number of at least one partition is in the PCR of the other partition. More precisely, for two nodes p_1 and p_2, connect (p_1, p_2)=true if SequNo(p_1) $\in$ PCR(p_2) or SequNo(p_2)$\in$ PCR(p_1) or both.

Note that it is possible that there exits one or more singleton nodes in the PCG, the nodes that are not connected with any other nodes in the graph. These singletons are those partition whose PCR is [\$, \$].

Sequence No. of partitions	**PCR**
1	[2, 4]
2	[1, 3]
3	[1, 4]
4	[3, 3]
5	[3, 4]
6	[\$, \$]

Table 1. The Partition Comparison Ranges (PCR) of the partitions

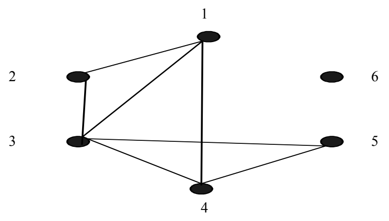

Figure 2. The corresponding Partition Comparison Graph (PCG)

E.g. Let's suppose that there are 6 partitions in a database and their PCRs are given in Table 1. Their corresponding PCG is presented in Figure 2. The 6th partition is an example of singleton node that is not directly connected with any other nodes in the PCG.

The general steps of constructing PCG are as follows:

(1) The set of outer partitions that are left after pruning using Property 4 for each Delimiting Record of a partition is computed;

(2) The PCR of the partition is obtained by using all the post-pruning list of Delimiting Records in the partition in Step (1);

(3) Step (2) and (3) are repeated until the PCRs of all partitions in the database have been found.

(4)PCRs of all partitions are converted to PCG.

(3) 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).

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:

(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 a partition, whose duplication or non-duplication decision cannot be made using Property 1 and 2, are compared using 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.

Inter-PC
Instead of finding duplicate records only within each partition, Inter-PC globalizes the record comparison to find duplicate records across different partitions. After the PCG has been constructed, Inter-PC will compare the records of the partitions that are directly connected in PCG. Inter-PC performs 4 steps for the whole database:
(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 impossibly contains 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 1 or 2, Inter-PC will perform detailed comparisons between *R* and these using TI-Similarity.
(3) Repeat Step (1) and (2) until all the record 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.

5. Complexity Analysis
Let N be the number of records in the database, s be size of each partition, k be the number of partitions, p be the average number of neighboring outer partitions in PCG of each partition, a be the percentage of total record pairs that have to be compared using TI-Similarity in Intra-PC, b be the percentage of records in neighboring outer partitions that each examined record has to compare with using TI-Similarity in Inter-PC, C_{sim} be the average cost of comparison of a pair of records using TI-Similarity and C_{flop} be the cost other than C_{sim} for float point operations such as additions, deletions, multiplications and judgments.

In database sorting, the cost will be $N*\log N*C_{flop}$ 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 $k*s*\log s*\ C_{flop} = N*\log s\ *C_{flop}$. The total cost for this step is thus approximately $N*C_{sim}$ given $s<<N$ and $C_{flop}<< C_{sim}$.

To construct PCR of each partition, similarities between each DR of the partition and ARs of its outer partitions will be computed, with a cost of $N_{dr}*k^2*C_{sim}$. Then, one scan of the PCRs of all partitions is required in the conversion from PCR to PCG, which requires a cost of $k* C_{flop}$. In sum, the cost of constructing PCG is $N_{dr}*k^2*C_{sim}+ k* C_{flop}$. Given $C_{flop} << C_{sim}$, the cost of constructing PCG can be simplified as $N_{dr}*k^2*C_{sim}$.

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

In Inter-PC, each record R will compare with all the ARs in its neighboring outer partitions in PCG, requiring a cost of $p* C_{sim}$. Inter-PC will then draw on R-P and R-R Properties to evaluate pairs of records, with a cost of $p* C_{flop}$ *and* $p*s*C_{flop}$, respectively. Finally, R will compare with the records in the partitions that have not been pruned using TI-Similarity, with a cost of $b*p*s*C_{sim}$. In sum, the total cost for examining N records in inter-PC will be $N*(p* C_{sim}+ p* C_{flop} +p*s*C_{flop}+b*p*s*C_{sim})$. Given $C_{flop}<<C_{sim}$, the cost of Inter-PC can be simplified to $b*p*s *N*C_{sim}$.

Combining the cost of all the above steps, the total cost of PC-Filter will approximately be $N*\log N*C_{flop}+(N+ N_{dr}*k^2)*C_{sim}+ a*s*N* C_{sim} +b*p*s* *N*C_{sim}$. Given $k=N/s$, the cost will be $N*\log N*C_{flop}+(1+(a+b*p)*s+ N_{dr}*N/s^2)*N*C_{sim}$. This cost can be minimized to $N*\log N*C_{flop}+(1+1.89(a+b*p)^{2/3}N^{1/3}+N_{dr}^{1/3}N^{1/3})*N*C_{sim}\sim O(N^{4/3})$ when $s=2^{1/3}*N_{dr}^{1/3}*N^{1/3}*(a+b*p)^{-1/3}$. This analysis shows that the complexity of PC-Filter can be ideally reduced to the order of $O(N^{4/3})$ by picking an optimized value of s.

6 Experimental Results

In our experiments, the pair-wise comparison method, Multi-pass SNM, Priority Queue method [7] and RAR [10] are used for comparative study on the performance in duplicate record detection.

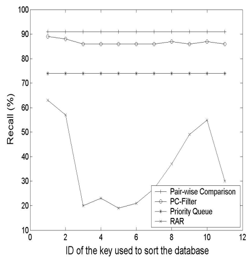

Figure 3. Recall results when varying the keys

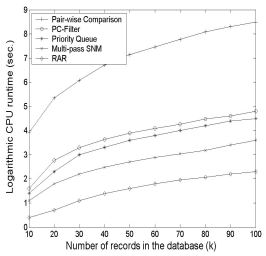

Figure 4. Logarithmic CPU runtime

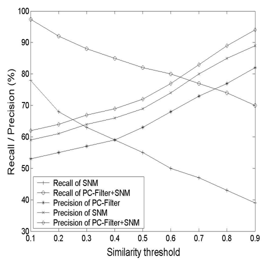

Figure 5. Recall-Precision graph of SNM and PC-Filter+SNM

At first, we will vary the key to sort the database in order to evaluate its effect on the recall level of PC-Filter. Specifically, we sort the database based on each of the 11 fields of the records in the synthetic dataset and compute the recall of duplicate record detection using each of these 11 sorted databases. The keys are 1. Social Security No. 2. Name 3. Gender 4. Martial status 5. Race 6. Nationality 7. Education 8. Office phone number 9 Home phone No. 10. Mailing address 11. Position. The result is shown in Figure 3. 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 key selected to sort the database. Pair-wise comparison method and Priority Queue method do not require database sorting, thus their recalls are not affected by the key chosen to sort the database. 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. Without using a window of fixed size, 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 from.

We are also interested in exploring the efficiency of PC-Filter against other four methods. Figure 4 shows the logarithmic CPU runtime of different methods. The CPU time of PC-Filter is higher than the time of Multi-pass SNM (O($k*w*N$)) and RAR (O($w*N$)), but 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)), where w is the window size and N is the number of records in the database. 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 due to its computational complexity of O($N^{4/3}$).

Finally, we will experimentally show that the framework of PC-Filter+X (incorporating PC-Filter into a compare method X with a different similarity measure, such as edit distance), will enable us to achieve better recall and precision than only using X. 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 5. 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 SNM. This is because using two similarity measures, 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 better recall and precision performance than performing SNM alone.

7 Conclusions

A robust filtering technique, called PC-Filter, is proposed in this paper for duplicate record detection in large database. In PC-Filter, the database is first sorted and sequentially split into a number of partitions. These 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. Experimental results verify PC-Filter is able to well solve a number of critical problems the existing methods suffer from.

References

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

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

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

[4] M. Hernandez. A Generation of Band Joins and the Merge/Purge Problem. *Technical Report CUCS-005-1995, Columbia University*, Feb 1996.

[5] M. A. Hernandez and S. J. Stolfo. The Merge/Purge Problem for Large Documents. In *Proceedings of the 1995 ACM-SIGMOD*, pages 127-138, 1995.

[6] 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.

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

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

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

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