Cluster Analysis to Find Sets of High-frequency Queries for Filtering in Similarity Join

Kamolwan Kunanusont[1] and **Jaruloj Chongstitvatana**[2], Non-members

ABSTRACT

Similarity search and similarity join are important operations in text databases. In some situations, some similar queries, called high-frequency queries, are repeated over a period of time. High-frequency-queries-based filter is used to facilitate this type of queries. However, the performance of this method depends mostly on the chosen high-frequency queries. This paper proposes methods, which are based on DBSCAN and agglomerative hierarchical-based clustering algorithm, to find high-frequency queries for the filter, called DBRAN and DBSM. For evaluation, both DBRAN and DBSM are applied on various sets of queries to find high-frequency queries for three datasets. It is found that DBSM performs better than DBRAN when the variation among high-frequency queries is high. However, when the variation among high-frequency queries is low, the performance of both DBRAN and DBSM are about the same.

Keywords: Similarity Join, Similarity Search, High-frequency Queries, Cluster Analysis

1. INTRODUCTION

Similarity join is used to retrieve all similar pairs of texts between two relations in a database. It is used in text databases in many situations, such as inexact queries and data cleaning. A pair of texts is considered to be similar if the similarity between the texts, which is measured by a similarity function, exceeds the specified threshold. Many similarity functions such as Cosine similarity [1] and Jaccard similarity [1] are widely used. However, calculating the similarity between every pair of texts in the relations takes a long time. Filter-and-verify framework is used to reduce the computation time for similarity join by first filtering out some dissimilar pairs of texts and then calculating the similarity of the remaining pairs, called candidates, for verification. Many filter methods, such as prefix filtering [2] and AdaptJoin [3], are proposed with the objective to obtain high filter percentage with low computation time.

Prefix filtering reduces the computation time by examining only a short prefix of a text to prune a dissimilar text. In [2], a text is further examined in the verification step when it is not possible to determine from its short prefix if it is dissimilar to the query. Both the filter percentage and the filter time for prefix filtering increase when longer prefix is used for filtering. AdaptJoin and AdaptSearch [3] dynamically choose the prefix length for each pair of texts so that the increase in filter percentage is worth the increase in the filter time. Suffix filtering [3] and positional filtering [3] are also used to improve the effectiveness of filtering.

The performance of the filtering methods such as [2, 3, 4] is measured by the filter percentage and the filter time, and it depends mostly on the tokens in the pair of texts. However, in real-world situations, many queries contain a common set of tokens, called a high-frequency query. These filter methods do not assume to have the knowledge of these patterns, and they repeat the same filter process for all queries which contain similar tokens. High-frequency-query based filtering [5] makes use of the high-frequency queries by organizing text data according to its similarity compared to each high-frequency query in a table called a similarity table. If a query is similar to one of the high-frequency queries, the similarity table of the closest high-frequency query is chosen and candidates of the query are obtained from the similarity table. However, if a query is not similar to any high-frequency queries, prefix filtering is used to effectively find the candidates. It is shown in [5] that the filter time depends on the number of chosen high-frequency queries. An additional inverted indices in [6] reduces the filter time by using prefix filter to reduce the time for finding the closest high-frequency query.

However, it is necessary to first find an appropriate set of high-frequency queries, which covers a large number of queries containing the repeated patterns. In order to reduce the space for similarity tables, fewer high-frequency queries should be used. This paper proposes to use cluster analysis in order to identify high-frequency queries for filtering. The density-based cluster analysis is an unsupervised method which groups data together based on their closeness [7]. Three main types of cluster analy-

Manuscript received on August 15, 2015 ; revised on December 30, 2015.

Final manuscript received on February 4, 2016.

[1,2] The authors are with Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University Bangkok 10330, Thailand, E-mail: kamolwan.k11@gmail.com and jaruloj@gmail.com

sis are center-based, hierarchical-based and density-based clustering. In this paper, methods to find an appropriate set of high-frequency queries are presented. Density-based spatial clustering of applications with noise (DBSCAN) [8] is used to find clusters in a query set, and a core point in each cluster is used as a high-frequency query. However, DBSCAN yields too many clusters, and many of them are overlapped. This paper presents a method of choosing representative points for a query set. DBSCAN with a RANdom core point (DBRAN) randomly picks one core point from each cluster obtained from DBSCAN as a high-frequency query. This method is used as a basis for the comparison. However, the random core points may not be good representatives for the query sets. DBSCAN with Merging (DBSM) which merges near-by clusters under some criteria is proposed.

In the rest of this paper, similarity join and cluster analysis are described in Section 2. The problem of finding high-frequency queries is defined and methods for finding high-frequency queries are proposed in Section 3. Experiments which are performed to evaluate the high-frequency queries obtained from the proposed method are presented in Section 4. In Section 5, the experimental results are discussed. The conclusion and future work are discussed in Section 6.

2. RELATED WORKS

This section describes similarity join and filtering methods used for similarity queries, especially high-frequency-queries-based filter. Then, cluster analysis is described.

2.1 Similarity Join

Similarity join finds all pairs of similar texts from two datasets, where the similarity between texts is measured by a similarity function. Given two text datasets D_1 and D_2, a similarity function f and a similarity threshold t, the similarity join of D_1 and D_2, denoted by $SJ(D_1, D_2)$, is $\{(d_1, d_2)|d_1 \in D_1, d_2 \in D_2$ and $f(d_1, d_2) \geq t\}$ [2].

The brute force method to find $SJ(D_1, D_2)$, which is to calculate the similarity between all pairs of texts (d_1, d_2) in $D_1 \times D_2$ and compare with t, consumes too much time. The filter-and-verify framework [4] reduces the time for similarity join by first filtering out non candidates, which are text data that cannot possibly be answers of the query, and then verifying the remaining candidates for the answers. In the verification step, the similarity between the query and each candidate is calculated, and only the candidate whose similarity exceeds the given threshold is returned as an answer. Consequently, the following two factors must be considered for the performance. First, the average filter time for each text data should be less than the verification time. Second, the filter step preferably produces a small number of candidates. Thus, a trade-off between the filter time and the filter power must be considered.

Many methods for filtering, such as [2, 3, 4], are proposed. Some filtering methods minimize the filter time by examining only some parts of the text. Prefix filtering [2], positional filtering [3] and suffix filtering [3] are based on this idea. For prefix filtering, inverted indices are created to group text data according to their prefixes, and to speed up filtering. AdaptSearch and AdaptJoin [4] aim to improve the filter power of prefix filtering by finding a good trade-off point for filter time and filter power. Delta inverted indices are proposed to support the use of varying prefix length in filtering.

However, when high-frequency queries arise, these filtering methods go through the indices and get similar results repeatedly. The filter time can be reduced by caching the result of a high-frequency query with varying query thresholds, and used this result for any queries which are similar to the high-frequency query. Based on this idea, another approach for filtering is to organize text data according to the similarity between the text data and a chosen text. High-frequency-queries-based filter [5] is an example of this approach, and a text which is similar to many expected queries, called a high-frequency query, is one of the chosen texts. A high-frequency query is used because the filter power is high when the query is similar to the chosen text. In practice, these high-frequency queries often appear. For example, after April 2015 Nepal earthquake [9], the search for words 'Nepal', 'earthquake' and 'help Nepal' dramatically increases compared with the previously recorded data by Google trends explorer [10]. This group of words is a high-frequency query, which can be defined as a set of words/tokens such that many queries are similar to it.

When high-frequency queries are chosen, a similarity table is created based on each of them. Each table stores the pointers to all records in the dataset sorted by the similarity between each record and the corresponding high-frequency query. Each row of a similarity table built for a high-frequency query F contains pointers to text data whose similarity with respect to F are in a specified range for the row. That is, for the similarity table ST_F created for the high-frequency query F and the dataset D, the row i, in s rows, of ST_F stores pointers to text data whose similarity compared to F is between i/s and $(i+1)/s$. Thus, $ST_F[i] = \{p|p$ is the pointer to $r \in D, i/s < f(r, F) \leq (i+1)/s\}$. The following example illustrates the similarity table.

Example 1: Suppose a similarity table ST_F of 10 rows, shown in Table 1, is created for a dataset D, using a high-frequency query $F = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\}$. The Jaccard similarity

function f, where $f(s_1, s) = |s_1|/|s_2|$, is used as the similarity function. Consider the following data records in D:

$d_1 = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_{10}\}$,
$d_2 = \{t_1, t_2, t_3, t_4, t_5, t_6, t_9, t_{11}\}$,
$d_3 = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_{10}, t_{11}, t_{12}\}$, *and*
$d_4 = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{11}\}$,

The similarity between the high-frequency query F and each data record is calculated, as shown below.

$f(d_1, F) = 8/10 = 0.800$
$f(d_2, F) = 7/10 = 0.700$
$f(d_3, F) = 7/12 = 0.583$
$f(d_4, F) = 9/10 = 0.900$

Thus, the data records d_1, d_2, d_3 and d_4 are stored in rows 7, 6, 5 and 8 of ST_F, respectively, as shown in Table 1.

To find the answers for a query q with the threshold t, the high-frequency query, say F, which is closest to the query q is chosen. Then, the similarity between q and F is calculated. It is found that a text data d is a possible answer of the query q with the threshold t if $f(q, F) - (1 - t) \leq f(d, F) \leq f(q, F) + (1 - t)$. From the similarity value between F and any possible answer of q, the lower bound $f(q, F) + (1 - t)$ and the upper bound $f(q, F) + (1 - t)$ are used to filter out data in some rows of the similarity table.

Table 1: *The system availability comparison between HADM-KRS and the proposed KRS with KRA failure (K = N-2) and KRS without KRA failure (K = 0).*

Row	Data records	Jaccard similarity (0.0-1.0]
0	...	(0.0, 0.1]
1	...	(0.1, 0.2]
2	...	(0.2, 0.3]
3	...	(0.3, 0.4]
4	...	(0.4, 0.5]
5	$\ldots, d_3, \ldots$	(0.5, 0.6]
6	$\ldots, d_2, \ldots$	(0.6, 0.7]
7	$\ldots, d_1, \ldots$	(0.7, 0.8]
8	$\ldots, d_2, \ldots$	(0.8, 0.9]
9	...	(0.9, 1.0]

The following example shows how to find the answers of a query from a similarity table created from a high-frequency query.

Example 2: Suppose the query $q = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ is queried on the dataset D, shown in Example 1, with threshold = 0.85. The similarity between q and the high-frequency query F of the similarity table ST_F, which is shown in Table 1, is computed; $f(q, F) = 9/10 = 0.900$. Considering any candidate for the query q, the lower bound and the upper bound of its similarity compared to the high-frequency query F are $0.9 + (1 - 0.85) = 0.75$ and $0.9 + (1 - 0.85) = 1.05$, respectively. Thus, the rows 7, 8 and 9 of ST_F contain possible answers of q, while the rest cannot possibly contain any answer. That is, d_2 and d_3 can be filtered out, but d_1 and d_4 must be further verified. For verification, the similarity between the query q and both d1 and d4 are calculated. Since $f(q, d_1) = 9/10 = 0.9$, which is higher than the threshold, d_1 is an answer of q. On the other hand, d_4 is not an answer of q because $f(q, d_4) = 9/11 = 0.818$, which is lower than the threshold.

When more high-frequency queries are used, the filter power can be increased but the filter cost is also increased because more similarity must be calculated. In [6], an inverted index of high-frequency queries is proposed, in order to reduce the computation time for choosing the high-frequency queries.

However, the performance of high-frequency-queries-based filter depends heavily on the set of chosen high-frequency queries. An appropriate set of high-frequency queries is necessary to create effective similarity tables for similarity join. This paper proposes an algorithm which finds the set of high-frequency queries that can be used efficiently for high-frequency-query-based filter based on cluster analysis, which is described next.

2.2 Cluster Analysis

Cluster analysis is an unsupervised learning technique which finds clusters of data in a dataset. It is used in many applications such as data mining, machine learning and pattern recognition [7]. Cluster analysis can be categorized as density-based clustering, hierarchical-based clustering and center-based clustering [7]. Density-based clustering creates data clusters in the high density areas. Hierarchical-based clustering groups small clusters of data to a larger cluster hierarchically. Center-based clustering groups data located near the same centers together. In this work, density-based clustering algorithm called DBSCAN is used as it is less susceptible to noise [8] along with agglomerative hierarchical based clustering algorithm.

DBSCAN is the density-based clustering algorithm that finds the clusters by eliminating points in the low density areas while grouping those in higher density areas together. A cluster is determined based on two parameters. One is *minpt*, which is the minimum number of points and the other is *eps*, which is the radius of clusters. A group of points is classified as a cluster if there are sufficient data points, i.e. at least minpt points, located near each other, i.e. within a radius *eps*. In DBSCAN, a data point is either a core point, a border point or a noise point. A core point is a data point with least minpt points in the radius *eps*. A border point is a data point, which is not a core point but located within eps distance from a core point. A noise point is a data point which is neither a core point nor a border point.

Although DBSCAN is initially proposed for spatial database [6], it can be applied in other applica-

tions efficiently. Similarity functions can also be used instead of distance function in order to define the relationship between data [11]. In this paper, the similarity function which is used for similarity join and search is also used for clustering, and the similarity is used as the distance between two points. Because a high-frequency query is a query which is similar to many other queries, high-frequency queries are center points of clusters created from a query set. Thus, our proposed method is based on DBSCAN, which discovers clusters of data and identifies core points of each cluster.

Another approach for clustering analysis is hierarchical-based clustering algorithm, which groups data in hierarchical relationship [7]. Data points which are close together have stronger relationship and are grouped into a cluster, and then clusters which are close together are further grouped together. For agglomerative methods [7], each single data point is initially considered as a cluster and clusters are grouped based on their distance. This procedure is repeated until all becomes only one cluster. The idea of agglomerative methods is applied, together with DBSCAN, to help identifying high-frequency queries.

Next, the algorithms for finding sets of high-frequency queries, based on DBSCAN and the idea of agglomerative clustering, are proposed.

3. PROPOSED METHOD

This paper proposes methods to find a set of high-frequency queries which can be used to build similarity tables as an index structure for high-frequency-queries-based filtering. High-frequency queries can be obtained by examining the query history. Given a set of queries, it is needed to find queries which appear frequently, or are similar to many queries from this set. First, high-frequency queries is defined in Section 3.1. Based on this definition, a high-frequency query is a representative of a group of similar queries. Clustering algorithms can be used to find such queries. DBSCAN is used to find high-frequency queries because it is less susceptible to noise [7]. From the definition, if queries are clustered densely, there are many high-frequency queries which are similar to each other. It is redundant to create two similarity tables based on two similar high-frequency queries. Two methods to choose an optimal set of high-frequency queries for high-frequency-queries-based filtering are proposed in Section 3.2. One method, called DBRAN, randomly chooses one of the high-frequency queries from each cluster. The other one, called DBSM, merges similar high-frequency queries together to create a representative of these queries. The performance evaluation of DBRAN and DBSM is described in Section 4.

3.1 Definition of High-frequency Queries

To make it possible to find useful high-frequency queries from a set of queries, a definition of a high-frequency query is formulated. First, the concept of *friends* is defined to allow the measure of frequency of a query from similar queries. Definition 1 states that two queries are friends if their similarity is not lower than a specified threshold t.

Let Q be a set of n text queries.
Let f be a similarity function and t be a specified similarity threshold between friends in range [0, 1].
Definition 1: A text q_1 is called a $\underline{friend}$ of another query q_2 in iff $f(q_1, q_2) \geq t$.
Lemma 1*: q_1 is a friend of q_2 iff q_2 is a friend of q_1.*

Then, a high-frequency query is defined based on the number of friends, as shown in Definition 2.

Let min_f be a positive integer $\leq n$.
Definition 2: A text query q is called a $\underline{high-frequency\ query}$ of the query set Q iff q has more than min_f friends in Q.

According to this definition, a high-frequency query is a query that has sufficient number of friends. There are several ways to find high-frequency queries from a query set Q. However, choosing an appropriate set of high-frequency is crucial for the performance of high-frequency-based filter.

3.2 Methods to Find Sets of High-frequency Queries

Based on the definition of a high-frequency query, DBSCAN method can be used to find high-frequency queries with a specified minimum number of friends and a similarity threshold. However, the distance function used in DBSCAN is different from the similarity function. The distance between two data points is high if they are different. On the other hand, the similarity between two data points is low if they are different. According to [11], a distance function can be mapped into similarity function using three conversion functions, i.e. linear, sigmoidal and inverted functions. In this section, the linear conversion function is used.

For both proposed methods, DBSCAN is used to find high-frequency queries. When high-frequency queries are closed together, the difference between the similarity tables created from these queries is so small that they are redundant. To avoid this redundancy, only some of these high-frequency queries must be chosen. The first approach is to randomly choose one high-frequency query for each cluster. DBSCAN with this randomly chosen high-frequency query for each cluster is called DBRAN. The other approach is to merge high-frequency queries obtained from DBSCAN, and this method is called DBSM.

For DBRAN, DBSCAN is used with the query set to find the clusters of queries along with their core points, and then one core point is randomly chosen from each cluster as one of the high-frequency queries. This resolves the redundant problem of the high-frequency queries. However, if the cluster is large, the random text might not cover all data in the cluster. DBSM is proposed to solve this problem.

DBSM also uses DBSCAN to find core points, which are high-frequency queries. Then, similar high-frequency queries are merged together to reduce the redundancy while preserving the query coverage. Therefore, we need a criteria to determine whether two core points should be both chosen as high-frequency queries or they should be merged into one. Furthermore, it must also consider how to merge two core points. We propose a merging scheme to deal with both of these issues.

To merge core points, the concept of hierarchical agglomerative clustering is adapted. For each cluster obtained from DBSCAN, the high-frequency queries are brought to find the pair with highest similarity. The decision to merge core points, or high-frequency queries, is based on the comparison between the similarity tables created from the original high-frequency queries and the merged high-frequency query. Suppose we have two high-frequency queries d_1 and d_2 which are most similar with each other, compared with other pairs of high-frequency queries in the same clusters. The merged high-frequency query created from $d1$ and d_2 contains the common tokens between d_1 and d_2. Specifically, the merged query $m = d_1 \cap d_2$. Then, the average number of candidates in the similarity table created with m as the high-frequency query and the average number of candidates in the similarity table created with d_1 and d_2 as the high-frequency queries are compared. The average numbers of candidates are obtained by doing high-frequency-queries-based filtering for queries which are the friends of the queries d_1 and d_2 compared to the friends of m, with the specified threshold t. If the latter one is smaller, the merged query is used as a high-frequency query, instead of the two original queries. Otherwise, we do not merge them and continue trying to merge the rest of the high-frequency queries. The remaining set is the set of high-frequency queries used for high-frequency-queries-based filtering. The following example shows how the merging strategy works.

Example 3: Suppose there are two text queries $q_1 = \{t_1, t_2, t_3, t_4, t_5\}$ *and* $q2 = \{t2, t3, t5, t6\}$. q_1 *has two friends* $b_1 = \{t_1, t_2, t_3, t_4\}$ *and* $b_2 = \{t_1, t_2, t_3, t_4, t_6\}$ *while* q_2 *has one friend* $b_3 = \{t_1, t_2, t_3, t_4, t_6\}$. *To determine whether* q_1 *and* q_2 *should be merged, the merged query* m *which is* $q_1 \cap q_2 = \{t_2, t_3, t_5\}$, *is created. Then, the similarity values between each query and its friends are computed, and the results are shown below.*

$f(q_1, b_1) = 0.89, f(q_1, b_2) = 0.8,$
$f(q_2, b_3) = 0.75,$
$f(m, b_1) = 0.57, f(m, b_2) = 0.52, f(m, b_3) = 0.86.$

Then, the similarity table for m is created from the dataset.

Given the number of data records in each row of the similarity tables created from q_1, q_2 *and* m *as shown in Table 2. The candidate number of* q_1 *using* b_1 *and* b_2 *is 1+1 = 2 while the candidate number of* q_2 *using* b_3 *is 1. Therefore, the candidate number before merge is 2+1 = 3. Then, the candidate number for m using b1, b2 and b3 is calculated from STm, which is 0+4+4 = 8. Since 8 > 4, which means the candidate after merge is more than the candidate before merge,* q_1 *and* q_2 *will not be merged. Otherwise they are merged into m and all of their friends are added to the friend list of* m.

Table 2: *Candidate numbers from similarity tables of* q_1, q_2 *and* m.

Table / Row	ST_{q1}	ST_{q2}	ST_m
1	1	1	0
2	2	1	2
3	4	2	4
4	15	13	28
5	28	33	16

In this paper, two methods are proposed to find an appropriate set of high-frequency queries because of the low performance of brute force methods. Therefore, only DBRAN and DBSM are implemented to compare the performance and the experiments are described in Section 4.

4. EXPERIMENTS

In this section, we describe experiments which are performed in order to evaluate the proposed method. The two methods to find sets of high-frequency queries - DBRAN and DBSM - are implemented in Java and compile with Netbeans IDE 7.1.2 on Windows 7 professional machine with 8 GB memory.

Three text datasets used for high-frequency-queries-based filter are:

- Enron [12], which contains email from Enrons senior management,
- NYTimes [12], which contains news article, and
- DBLP [13], which contains computer science bibliography.

Table 3 shows the sizes of these datasets.

However, the query sets which contain high-frequency queries must be generated. For each dataset, a certain number of data records are randomly chosen. These chosen records are the original high-frequency queries. Each of these high-frequency queries is mutated by changing some percentage of

Table 3: *Details of each dataset used in the experiments.*

Dataset	Number of possible words	Number of records	Average length per record
Enron	28,099	39,861	160
NYTimes	101,636	299,749	232
DBLP	467,446	1,385,952	14

the tokens, and added in a query set. Each query set used in the experiments is composed of a number of mutated high-frequency queries and a number of other queries randomly chosen from the dataset. From each dataset, 8, 16, 32, 64 and 128 records are randomly chosen as the original high-frequency queries. The original sets of chosen high-frequency queries cannot be known in real-world situation. But, for our synthetic query sets, they are used in the experiments for a baseline measurement to compare with the results from both proposed methods.

For each mutated query, mutation can be done by inserting, deleting or substituting a token in a query. A certain percent of tokens in each query are mutated, and the mutation percentage is varied from 20% to 50%. Each query set contains mutated high-frequency queries and randomly chosen records in the dataset. The percentage of the high-frequency queries are 50, 60 and 80. However, since the percentage of queries related to high-frequency queries does not affect the difference of the result between each method, only the results from 60% related sets are shown here.

Half of each query set is randomly chosen as a training set that DBRAN and DBSM use to find high-frequency queries. The other half of the set is used as a test set, which is used as query sets for high-frequency-queries based filter to measure the performance.

The set of high-frequency queries obtained from each method is used to measure the performance of the two methods. The method performs well if high-frequency-queries based filtering works well, using the set of high-frequency queries obtained from the method. Two factors coverage and the number of candidates are observed in the experiments. The coverage of a set of high-frequency queries is the number of queries in a query set that are similar to at least one high-frequency query. These queries are called in-coverage queries and others are called *out-of-coverage queries*. Large coverage indicates that the similarity tables can be used for a large number of queries. For an in-coverage query, a similarity table can be used for filtering and the candidates are obtained from the table. These candidates are called *in-coverage candidates*. A smaller number of in-coverage candidates indicates that the filter works well. On the other hand, for an out-of-coverage query, high-frequency-queries based filter does not use similarity tables and switches to AdaptSearch. In this case, the candidates obtained from similarity tables, called *out-of-coverage candidates*, are too numerous to be of use. It is preferable for a set of high-frequency queries to have large coverage and a small number of in-coverage candidates. Thus, the number of in-coverage candidates per in-coverage query is also an important indicator. However, if both the number of in-coverage queries and the number of in-coverage candidates are large, the average number of in-coverage candidates must be considered. These three values are measured in the experiments, and they are analyzed next.

5. PERFORMANCE MEASUREMENT

5.1 Coverage Percentage

First, the coverage percentage, which is the percent of the in-coverage queries in a query set, is examined. High coverage percentage indicates that many queries are similar to at least one of the chosen high-frequency queries, and one of the similarity tables can be used for filtering. Thus, it is desirable that a set of high-frequency queries gives high coverage percentage.

From the experiments with the query sets with 60% of the queries created by mutation on the high-frequency queries, Fig. 1-3 show the coverage percentage for the three datasets. It is shown from all three datasets that, when the mutation is low, i.e. at 20%, the coverage percentages of high-frequency queries obtained from DBRAN, DBSM and original sets are nearly the same. This means the sets of high-frequency queries retrieved from these two methods similar with the original sets. With at least 40% mutation, the coverage percentages from DBSM sets are better than DBRAN. They are both lower than original sets. For the original set of high-frequency queries, it remains the same when the queries are mutated at lower level. On the other hand, if the queries are mutated more than 40%, the coverage percentage decreases when the mutation level increases.

Another point to consider is the effects of the number of original high-frequency queries. If queries are mutated for 20%-30%, the number of high-frequency queries does not affect the coverage percentage. On the other hand, for the sets of high-frequency queries obtained from DBRAN and DBSM, their coverage percentages moderately decrease if the train sets contain more high-frequency queries. However, the original sets of high-frequency queries are not affected by this parameter.

From the experiments, it is shown that the sets of high-frequency queries obtained from both DBRAN and DBSM have similar coverage as the original set of high-frequency queries when the percent of mutation is low. However, when mutation is increased, both DBRAN and DBSM yield sets of high-frequency queries with lower coverage percentage compared to the original sets. However, DBSM finds sets of high-frequency queries with higher coverage percentage

than DBRAN, especially when the number of high-frequency queries is low.

The higher number of in-coverage queries leads to more in-coverage candidates which is shown next.

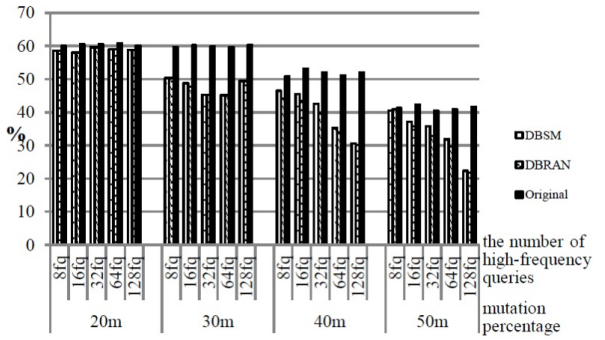


Fig.1: *Coverage percentage of DBLP dataset with 60% high-frequency queries.*

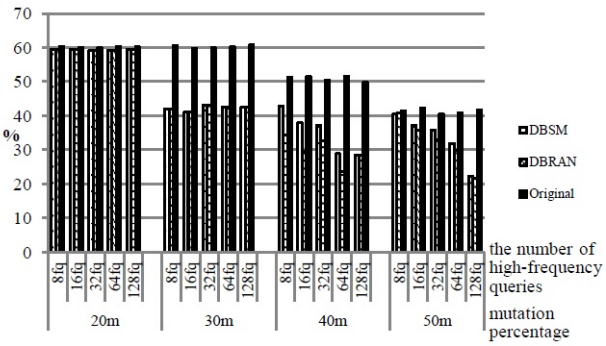


Fig.2: *Coverage percentage of NYTimes dataset with 60% high-frequency queries.*

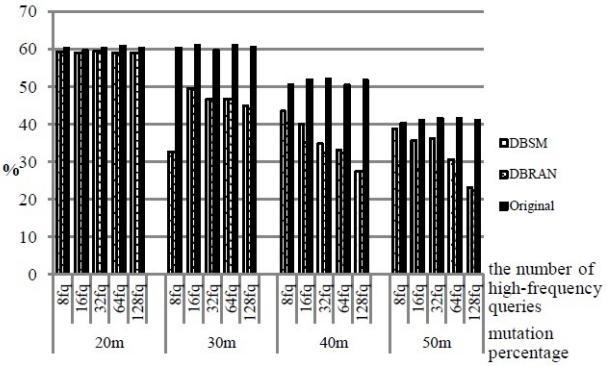


Fig.3: *Coverage percentage of Enron dataset with 60% original high-frequency queries.*

5.2 Numbers of In-coverage Candidates

For an in-coverage query, its candidates are retrieved from the similarity table of the closest high-frequency query. Thus, the number of in-coverage candidates indicates how much verification is required for similarity join. The number of in-coverage candidates depends on the similarity between each query and the chosen high-frequency query, together with the similarity threshold, which is 0.5 in this experiment. From the experiments on query sets, in which 60% of the queries created by mutation on the high-frequency queries, Fig. 4-6 show the number of in-coverage candidates for the three datasets.

For all three datasets, when the queries are mutated for 20%, the numbers of in-coverage candidates are low. Using high-frequency queries obtained from both DBSM and DBRAN, the numbers of in-coverage candidates are highest when the queries are 30% mutated and then decrease when queries are mutated more.

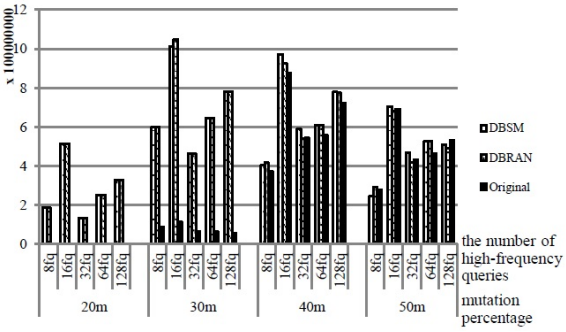


Fig.4: *In-coverage candidates of DBLP dataset with 60% high-frequency queries.*

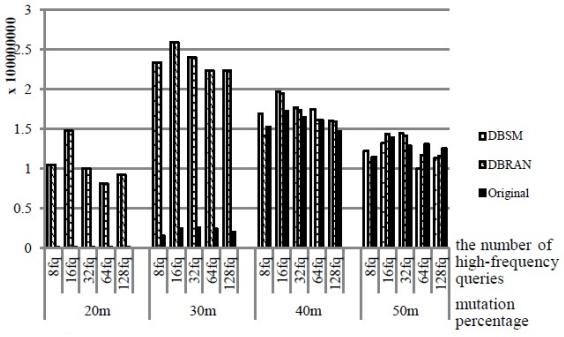


Fig.5: *In-coverage candidates of NYTimes dataset with 60% high-frequency queries.*

In contrast, in-coverage queries of the original sets generate very few candidates when the queries are mutated 20%-30%. However, the numbers of in-coverage candidates become almost the same for both DBSM and DBRAN when the queries are 40% mutated. With 50% query mutation level, the in-coverage candidate number decreases as well as the result from other methods. This means out-of-coverage candidates increase as the queries are more mutated.

Next, the average number of candidates per in-

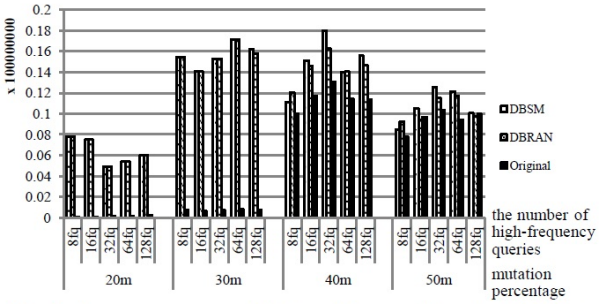


Fig.6: *In-coverage candidates of Enron dataset with 60% high-frequency queries.*

coverage query is examined.

5.3 Average Number of Candidates per In-coverage Query

The total number of in-coverage candidates increases with more in-coverage queries. Therefore, the number of candidates alone is not sufficient to evaluate the resulting sets of high-frequency queries. The average number of candidates generated from each in-covered query must also be considered. This is computed from the number of all in-coverage candidates dividing by the number of in-coverage queries. As the numbers of data records are different for all datasets, the percentage of average number of candidates, compared to the whole dataset, is used.

Fig. 7-9 show the mentioned percentage of the three datasets when 60% of the test sets related to the original high-frequency queries with various mutation percentage. For every dataset, when the mutation percentage is low, i.e. 20%, the in-coverage queries using original high-frequency queries generate very few in-coverage candidates, i.e. less than 0.01% of the whole dataset. However, both DBSM and DBRAN generate more in-coverage candidates, i.e. 2%-6%. When the percent of mutation is increased, high-frequency queries generated from DBSM and DBRAN yield roughly the same percentage of in-coverage candidates. In contrast with the higher percent of mutation, say 40%, each of the in-coverage queries using the original high-frequency query sets generate dramatically higher candidates.

The results from these three measurements lead to this conclusion. If the queries are mutated at low level, high-frequency-queries-based filter using the high-frequency queries found from DBRAN and DBSM give the similar results. Query sets with low mutation contain many similar queries which lead to denser clusters. Therefore, the merging strategy is not necessary as the random strategy performs well. On the other hand, if queries are highly mutated, DBRAN and DBSM give slightly different sets of high-frequency queries which lead to the difference in coverage percentage, candidate numbers and the percentage of candidate numbers per query in the datasets. The sets obtained from DBSM provide better results in most of the experiment. This means that the merging strategy is useful for the highly mutated query set.

6. CONCLUSION

This paper proposes a method to find a set of high-frequency queries, which is used in high-frequency-queries-based filtering, from a query set. The set of high-frequency queries is used to create similarity tables for high-frequency-queries-based filter for similarity join. DBSCAN clustering algorithm is applied to find the clusters of queries. DBRAN, which is DBSCAN with random core points, finds high-frequency queries and removes redundant core points by randomly selecting one core point from each cluster. However, one core point may not cover every query in a cluster. DBSM, which is DBSCAN with merging strategy, removes redundant core points and also preserves the coverage of each core point. This method merges two core points if they are similar. Then, the remaining core points are used as high-frequency queries. Experiment results show that DBSM and DBRAN are nearly the same when the high-frequency queries are similar, or the clusters are compact. On the other hand, if the high-frequency queries are highly varied, DBSM outperforms DBRAN as the resulting sets of high-frequency queries provide better performance for high-frequency-queries-based filter.

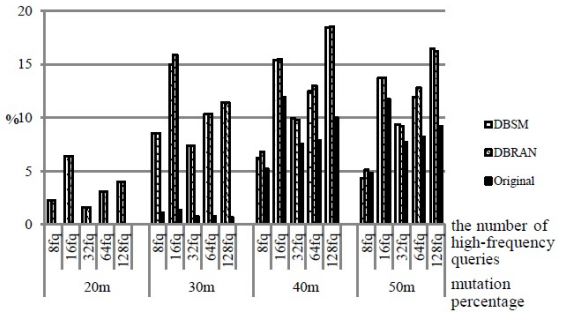


Fig.7: *In-coverage candidates per query percentage of DBLP with 60% related to original high-frequency queries.*

Although DBSM found the sets that cover more queries in the test sets than DBRAN, it takes much longer to compute if there are many core points in each cluster. Therefore, the strategy to determine whether the merging strategy is necessary for the set of core points should be studied further.

References

[1] M. Hadjieleftheriou, and D. Srivastava, "Approximate String Processing," *Foundations and Trends in Databases*, Vol. 2, No. 4, pp. 267-402, 2011.

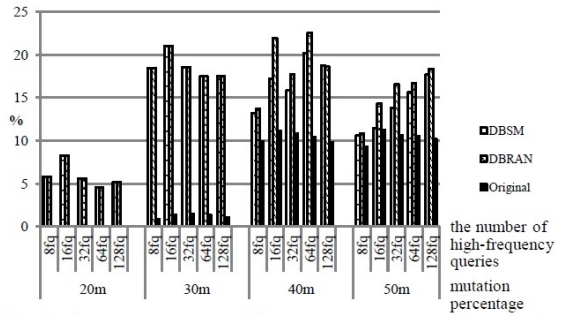


Fig.8: *In-coverage candidates per query percentage of NYTimes with 60% related to original high-frequency queries.*

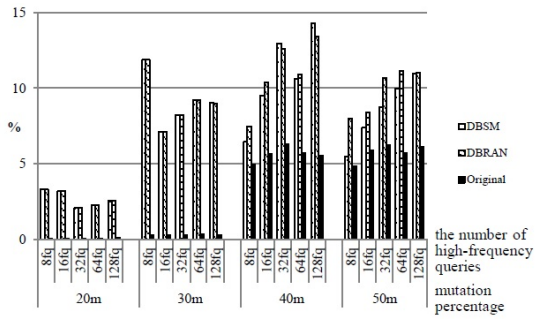


Fig.9: *In-coverage candidates per query percentage of Enron with 60% related to original high-frequency queries.*

[2] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," Proceedings of International Conference on Data Engineering (ICDE), 2006, pp. 5-16.

[3] C. Xiao, W. Wang, X. Lin, J. Xu Yu and G. Wang, "Efficient Similarity Joins for Near Duplicate Detection," *Proceedings of international conference on World Wide Web (WWW 08)*, 2011, pp. 131-140.

[4] J. Wang, G. Li and J. Feng, "Can we beat the prefix filtering?: An adaptive framework for similarity join and search," *Proceedings of ACM Management of Data (SIGMOD)*, 2012, pp. 85-96.

[5] K. Kunanusont and J. Chongstitvatana, "An Index Structure for Similarity Join Based on High-frequency queries," *Proceedings of International Computer Science and Engineering Conference (ICSEC)*, 2014, pp. 415 420.

[6] J. Chongstitvatana and N. Thitinanrungkit, "Refining High-frequency-queries-based Filter for Similarity Join," Proceedings of International Computer Science and Engineering Conference (ICSEC), 2015.

[7] J. Han, M. Kamber and J. Pei, *Data Mining: Concepts and Techniques*, Morgan Kaufman Publishers Inc., CA, 2011, ch. 10.

[8] M. Ester, H. Kriegel, J. Sander and X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proceedings of the second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996, pp. 226-231.

[9] "April 2015 Nepal Earthquake" `http://earthquake.usgs.gov/earthquakes/event-\\page/us20002926\#general\_summary.`

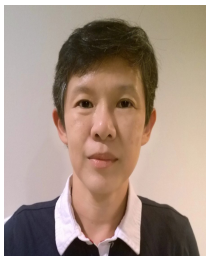
[10] "Google trends explorers" `http://www.google.com/trends/explore\#cmpt=q`

[11] M. Setten, M. Veenstra, A. Nijholt and B. Dijk, "Cased-Based Reasoning as a Prediction Strategy for Hybrid Recommender Systems," *Proceeding of Atlantic Web Intelligence Conference (AWIC)*, 2004, pp. 13-22.

[12] "UCI Machine Learning Repositary" `https://archive.ics.uci.edu/ml/datasets.html` Obtained on 8th January 2015.

[13] *The DBLP bibliography of published researchers in computer science* obtained on 25 Nov. 2013 from `www.cs.berkeley.edu/$\sim$jnwang/codes/adapt.tar.gz`

Kamolwan Kunanusont received the B. Sc. degree in computer science in 2014 and M. Sc. in computer science and information technology in 2015, both from Chulalongkorn University, Bangkok, Thailand. Now she is a master student in artificial intelligence at University of Essex, United Kingdom. Her research interests include similarity join, data mining and game artificial intelligence.



Jaruloj Chongstitvatana received the B.Eng. degree in computer engineering from Chulalongkorn University, Bangkok, Thailand, in 1986 and the M.Sc. and Ph.D. degrees in computer science from Michigan State University in 1989 and 1999, respectively. She is a lecturer in the Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University. Her research interests include index structures and data management.