STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **7**, June 2019, pp 501–519.
Published online in International Academic Press (www.IAPress.org)

Improved View Selection Algorithm Using SOM and 0/1 Knapsack

Reyhaneh Sabbagh Gol, Negin Daneshpour*

Department of Computer Engineering, Shahid Rajaee Teacher Training University, Iran

Abstract Data warehouse is designed for answering analytical queries. Data warehouse saves historical data. In the data warehouse, the response time to analytical queries is long. So reducing the response time is a critical problem. There are a lot of algorithms to solve the problem. Some of them, materialize frequent views. The previously posed queries have important information that will be used in the future. This paper proposes an algorithm for view materialization. The proposed algorithm finds proper views using previous queries and materializes them. The views are able to answer future queries. The view selection algorithm has four steps. At first, it clusters previous queries by SOM method. Then frequent queries are found by Apriori algorithm. In the third step the problem is converted to 0/1 knapsack equations and finally, optimal queries are joined to create only one view for each cluster. This paper improves the first and third step. This paper uses the SOM algorithm for clustering previous queries in the first step and it solves the 0/1 knapsack equations according to shuffled frog leaping algorithm in the third step. Experimental results show that it improves the previous view selection algorithms according to response time and storage space factor.

Keywords Data warehouse, Optimal queries, View materialization, SOM algorithm, Zero-one knapsack.

DOI: 10.19139/soic.v7i2.561

1. Introduction

Nowadays a lot of data are produced in the world. The collection and storage of data is a big problem although the technology has many advances. Also, large dimensional problems are occurring as large amount of data are produced [1]. The data are generated by managers for decision making. The advances of computer technology facilitate to store large databases [2]. There are two methods to access the data. The first method is called on-demand and the other one is called in-advance. In the first method, data are collected from different databases after executing users queries. But in the second method, data are collected in the data warehouse and then analytical queries are answered [3]. A data warehouse is a subject-oriented, integrated, time-variant and nonvolatile source that is used for decision making [4]. Subject-oriented means that the data warehouse is created according to a specific subject and it concerns on modeling and analyzing the data, instead of daily operations. Integration means that the data warehouse is created by integrating data from different databases. Databases are usually non-uniform. Time-variant means that a data warehouses data are related to a period of time like 5 or 10 years. Nonvolatile means that data wont be destroyed by itself [4].

Data warehouses are very important for decision making. They integrate data from various sources and save in a data warehouse for business managers [5].

Data in data warehouses are integrated from different sources to help decision making. Thus it must be able to answer the user's analytical queries. The response time for answering OLAP queries is an important factor. The data warehouse uses materialized views to reduce the response time. It is impossible to materialize all views

*Correspondence to: Negin Daneshpour (Email: ndaneshpour@sru.ac.ir). Department of Computer Engineering, Shahid Rajaee Teacher Training University. Shabanlu, Lavizan, Tehran Province, Iran.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

because of limited storage space. So some views must be selected to materialize. There are a lot of algorithms for view selection. One of them materializes profitable views according to previous queries. Previous queries have main information; as they will happen in the future most probably. In the algorithm, previous queries are clustered and then frequent queries are found in each cluster. Therefore optimal queries are found in each cluster according to the limited storage space; and finally, optimal queries are joined to create one view for each cluster [6]. It has a long response time; because of using primary techniques for finding optimal queries in each cluster. On the other hand, the main factor in view selection algorithms is response time; so this paper proposes an algorithm that solves the problem and has higher speed than the other ones that were simulated in our paper [6, 7].

A data warehouse is subject-oriented. So most of the previous queries posed on a data warehouse are related to a subject. Thus, it is better to materialize views according to the previous queries posed on the data warehouse. Because previous queries contains beneficial information related to the data warehouses subject. But, all the previous queries dont have important information. Therefore, the queries that access the same data and were repeated more than the others are more significant. Because they are more probable to occur in other times in the future.

In this paper, an improved view selection algorithm is proposed. The algorithm uses previous queries; according to previous paragraph. This algorithm clusters the previous queries to find similar queries, then frequent queries are found in each cluster to find similar and frequent queries. But, all the views cant be materialized, because of storage limitation. So, in the next step optimal queries are found according to available space. Finally, the optimal queries are merged together to find one view in each query. The information helps to improve query performance, so it would simplifies decision making.

The main contribution of the proposed algorithm in this paper is in the first and third steps. In the first step, previous queries are clustered using SOM neural network [8, 9]. It has less response time than the previous one. In the third step, the problem is modeled as 0/1 knapsack based on storage space. The shuffled frog leaping algorithm [10] is used to solve the knapsack equations. This algorithm doesnt trap into a local optimal and also has less response time.

The algorithm explained in [6] and the proposed algorithm are simulated and the results show that the proposed algorithm reduces the response time and storage space. The proposed algorithm has 14.22% improvement in time factors and 45.78% in storage space factor.

The structure of the paper is as the following: section 2 explains related works. Section 3 describes the proposed algorithm and section 4 shows simulation results. The last section explains the conclusion.

2. Related works

Most of the view selection algorithms materialize views by maximizing the profit. One of the algorithms is a greedy algorithm [11, 12]. The algorithms materialize optimal views according to limited storage space. But these algorithms dont consider all the parameters like maintenance cost.

The other group of algorithms is genetic algorithms [13, 14, 15, 16]. These algorithms select more appropriate views than the others, but they have a long response time for huge data warehouses.

The input of some algorithms is workload. This means that they use previous queries. Because future queries will be very similar to the previous queries. Some of them find similar sub-expressions in the workload and materialize them [17, 18]. But finding sub-expressions requires a long time.

Some algorithms model a graph based on input queries, according to view selection problem. Some of these graphs are called AND-OR graph or MVPP . According to both cost function and modeled graph, views are materialized [19, 20, 21]. But the algorithm doesn't consider some parameters like the complexity of the query.

One of the algorithms materializes views by maximizing the profit and minimizing the cost [22]. Some factors that are important in profit function are as follows: access frequency of the query, query execution time, complexity of the query, query processing cost and view maintenance cost. After that, the DAG graph is constructed according to the cost function. Views are edges in the graph. Then the shortest path is found in the graph. Thus views with maximum cost are found to materialize. The algorithm involves many factors but it doesnt consider some factors

like the cost of dropping the materialized view. One of the view selection algorithms predicts the next query and materializes related views according to the prediction [23]. But always the prediction isnt confident. So much time will be wasted if the prediction will be wrong.

Some algorithms use mathematical modeling. These algorithms convert the view selection problem to mathematical equations and then materialize the views by solving the equations. Some of these algorithms model the problem by Constraint Satisfaction Problem [24, 25, 26] and integer programming [27, 28]. The method is very powerful because it uses mathematics. To model the problem, it had to be supposed some constraints, therefore it makes the problem unreal.

The algorithm proposed in [29] is an improvement of the greedy algorithm. The algorithm uses some factors in addition to greedy factors: size of the view, frequency of the view and decision making capability of views [29]. The algorithm uses the lattice of cuboids.

The algorithm introduced in [30] is another improvement of the greedy algorithm. The algorithm uses a table-like structure and a cost model. But the algorithm only considers query processing and view maintenance cost in the cost function.

The algorithm proposed in [31] uses a transactional database. In the algorithm, the input queries are transformed into a transaction table. A transaction corresponds to a query and its itemsets are the original queryś predicates.

The algorithm proposed in [32] uses backtracking search optimization algorithm for materializing views. This algorithm minimizes the cost of query processing within the storage constraint. But, when number of branches increases it needs long time and large space complexity because of multiple function calls. Query prioritization algorithm [33] considers the queries priority. Because each query has a priority value that means how immediately it must be answered. But, if the priority of a query assigns wrongly, it wastes response time of the algorithm.

View materialization can be merged with cloud computing. In [34] a cloud based view materialization algorithm is presented to enhance the performance of the data warehousing. But, cloud computing is very expensive. In some papers clustering and data mining methods are used [6, 7, 35, 36, 37]. In these algorithms, previous queries are used because they will be used in the future most probably. These algorithms follow the MVCF method. This method has four steps:

a) First, queries are clustered using clustering methods. b) Then frequent queries are found in each cluster using data mining methods. c) Then optimal queries are found in each cluster. d) Finally, optimal queries are merged together to find only one view for each cluster.

In [6, 7, 35, 36, 37] primary techniques like Hierarchical [7] technique is used for clustering. 0/1 knapsack is used for finding optimal queries in each cluster and dynamic programming is used for solving knapsack problems. The solution isn't suitable for view selection problem because of the long response time. Thus it wont be able to use for huge data warehouses. In this paper SOM method [8, 9] is used for clustering previous queries and shuffled frog leaping algorithm [10] is used for solving 0/1 knapsack equations to improve the performance of the view selection algorithm.

3. The proposed view selection algorithm

In this section, the proposed view selection algorithm is explained. This algorithm uses the MVCF method that was explained in the related works. Figure 1 shows the steps in general.

In this algorithm previously posed queries are used, because the queries have essential information that will be needed in the future most probably. The proposed algorithm in this paper is an improvement of the view selection algorithm in [6, 7, 35, 36, 37]. The algorithm explained in [6, 7] is called KD2013 according to the editors name and the year of publication. The proposed view selection algorithm is called SRTTU-SF according to use SOM and shuffled frog leaping algorithm. The proposed view selection algorithm is explained as follow:

At first, previous queries are clustered by SOM [8, 9] neural network. SOM neural network has input and output nodes. The number of input nodes is equal to the number of dimensions in the data warehouse. So each node corresponds to one dimension. The number of Output nodes is equal to the number of clusters. An edge connects the input node to the output node. In the SOM neural network, each input node is connected to all of the output

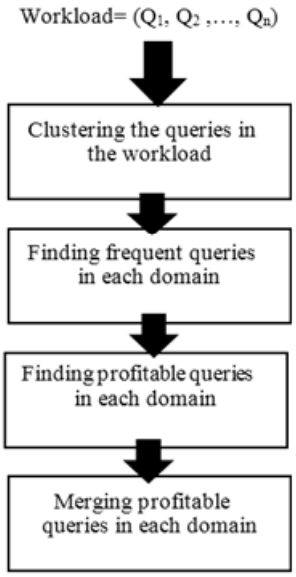

Figure 1. Steps of MVCF method.

nodes. Each edge has a weight that its value is between -1 and 1. Weights are produced randomly for the first run of the algorithm. Figure 2 shows an example of the SOM algorithm.

This figure has 4 input nodes and 2 output nodes. This means that the data warehouse has 4 dimensions and the number of clusters is 2. As shown in Figure 2 each edge has a weight. Equation 1 shows an example of weights.

$$w_t1 = \begin{bmatrix} W_t11 \\ W_{12} \\ W_{13} \\ W_{14} \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.7 \\ -0.2 \\ 0.5 \end{bmatrix}, \quad w_2 = \begin{bmatrix} W_{21} \\ W_{22} \\ W_{23} \\ W_{24} \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.5 \\ -0.9 \\ 0.3 \end{bmatrix} \tag{1}$$

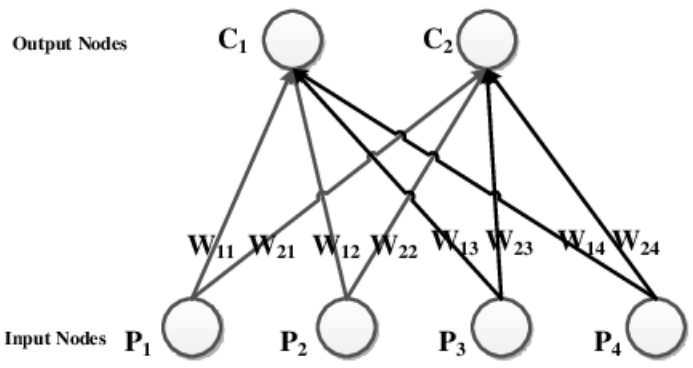

Figure 2. An example of SOM.

SOM has training and test data. The training data train the SOM neural network and the test data tests the SOM neural network. In SRTTU-SF algorithm the previous queries are training data and the test queries are test data.
After initializing input, output nodes and edges weights, the training queries are analyzed. At first, a training query is selected. The query is shown as a matrix: $x = [x_1, x_2, x_3 x_n]$. x_i could have two values. If x_i = 0 it means that training query x doesnt contain i dimensions in the query's from clause. If x_i = 1 it means that training query x contains i dimensions in its from clause. Then the distance between x and each weight matrix of output nodes will be calculated. In other words in Equation 1 the distance between x and W_1, W_2 will be calculated. The Euclidean distance between two matrixes $A = [a_1, a_2 a_n]$ and $B = [b_1, b_2 b_n]$ is calculated with Equation 2.

$$d = \sqrt{\sum_{i=1}^{i=n} (a_i - b_i)^2} \tag{2}$$

After calculating the distances, the minimum distance is selected to update the weight matrix of that output node. For example, the distance between x and W_1 is 1.67 and the distance between x and W_2 is 0.9. So the minimum value corresponds to W_2 and the weight matrix of W_2 should be updated. The update function is represented in Equation 3.

$$W_j(t+1) = W_j(t) + \eta(t)(x(t) - W_j(t)) \tag{3}$$

In this equation, W_j is a weight matrix corresponded to the output node j. The distance between x and W_j is minimum so W_j should be updated by Equation 3.η is the learning rate and its value is between 0 and 1. After updating the W_j matrix, the next training query is analyzed and the distances are calculated and then weights are updated. So the process will be repeated for all training queries. For Convergence, the algorithm will be repeated several times. The number of repetition is called epoch.
After executing the algorithm, weight matrixes will be calculated. To decide which query belongs to which cluster, we should do as following for each query: at first, each input query is showed as an $n \times 1$ matrix. Then the distance between the input query matrix and all of the weight matrixes (corresponded to this input query) is calculated. Therefore the minimum distance is found and the input query belongs to the cluster with minimum distance. The number of clusters is the same as the number of output nodes.
Algorithm 1 shows the pseudo code for finding the queries cluster of the proposed algorithm.

As mentioned in algorithm 1, queries are clustered by SOM neural network. In this algorithm epoch is the number of repetition. In the next stage, frequent queries in each cluster are found. To find the frequent queries Apriori algorithm is used [7]. Then optimal queries should be found in each cluster [6]. The optimal queries are selected according to limited storage space. The problem likes a knapsack. It is supposed that the limited storage space likes the space of a knapsack. So all the queries cant be put in the knapsack, and some of them should be selected. In our problem, optimal queries should be selected according to the storage space limitation [6]. Suppose that $Q_1, Q_2, , Q_n$ are frequent queries in cluster k. There are two variables P_i and S_i for each query Q_i. The result of executing query Q_i is a table that is called T_i. The number of T_is records is saved in P_i variable. Q_i contains some tables in from clause. The record number of the tables is calculated and saved in the S_i variable. Therefore for each Q_i, both P_i and S_i are calculated. 0/1 Knapsacks equation is constructed according to equation 4.

$$Maximize\ P_1Q_1 + P_2Q_2 + ... + P_nQ_n. \quad Subject\ to\ S_1Q_1 + S_2Q_2 + ... + S_nQ_n \leq S \tag{4}$$

S_i is defined according to equation 5.

$$S_i = \sum_{R=Q_i} t(R) \tag{5}$$

$t(R)$ is the number of a table's records. R is a table that is in from clause of Q_i. The value of Q_i can be zero or one; if the i-th query is selected then its value is one; otherwise, it's zero [37]. Equation 6 shows an example of

Algorithm 1 Clustering Queries

procedure FINDING CLUSTERS
  Input: Previously queries set $Q = Q_1, Q_2, , Q_m$, Dimensions of data warehouse n,ηLearning rate ,number of epochs $epoch$, number of clusters c.
  Output: clusters with their members
  Initialize all weights randomly;
  for each $epoch$ **do**
    for each previously query x **do**
      for each output node j **do**
        Find Euclideans distance between x and W_j;
      end for
      min = Find the index of the minimum Euclidean's distance;
      Update W_{min} by using Eq. 3;
    end for
  end for
  for each query x **do**
    for each output node j **do**
      Find Euclideans distance between x and W_j;
    end for
    s = Find the index of the minimum Euclidean's distance;
    Assign x to cluster s;
  end for
end procedure

knapsacks equation.

$$\begin{aligned} &Maximize\ 13Q_1 + 15Q_7 + 28Q_{16} + 35Q_{17} + 32Q_{20}. \\ &Subject\ to 120Q_1 + 250Q_7 + 430Q_{16} + 310Q_{17} + 360Q_{20} \leq 1000. \\ &and\ Q_i = 0\ or 1\ where\ i = 1, 7, 16, 17, 20. \end{aligned} \tag{6}$$

The knapsacks equations are solved through shuffled frog leaping algorithm to get the Q_is value.
First, shuffled frog leaping algorithm will be explained. Then the changed shuffled frog leaping algorithm for solving 0/1 knapsack problem is explained, and finally, our proposed algorithm will be described.

3.1. Shuffled frog leaping algorithm

Shuffled frog leaping algorithm [10] is based on group behavior of frogs to find a location with maximum food. The algorithm is as follows:
In this algorithm, the population includes a set of frogs (solutions). The population participates into some subsets. Each subset is called memeplex. Each memeplex has a different culture to the others. Each frog has an idea in each memeplex. Its idea can be affected from other frogs in its memeplex. Their idea will be improved by memeplex evolution procedure. After executing the evolution procedure for special steps, the ideas will be shared between memeplexes. Local search procedure that is executed inside the memeplex and sharing ideas between memeplexes are continued until the termination condition happens. The algorithm is explained in figure 2.

An initial population of frogs is produced randomly. For a problem with S dimensions (with S variables) the frog i is shown as $i = (X_{i1}, X_{i2}, , X_{iS})$. Then frogs are sorted according to their fitness in descending order. Their population is divided into m parts (memeplex) that each part has n frogs (their population is $P = m * n$). Then the first frog is allocated to the first memeplex, the second frog is allocated to the second memeplex, the m-th frog is allocated to the m-th memeplex and m+1-frog is allocated to the first memeplex. In each memeplex, the frog with the best fitness is shown with X_b and the frog with the worst fitness is shown with X_w. In the whole memeplexes,

Algorithm 2 Shuffled frog leaping algorithm [10]

procedure SHUFFLED FROG LEAPING ALGORITHM
Input: the number of frogs P; the number of memeplexes m; the number of generation for each memeplex before shuffling n; the number of shuffling iterations it; and the maximum number of iterations $iMax$.
Output: best solution
Generate random population of P solutions (frogs)
for each individual $i \in P$ **do**
Calculate fitness(i);
end for
Sort the population P in descending order of their fitness;
Divide P into m memeplexes;
for each memeplex **do**
Determine the best and worst frogs;
Improve the worst frog position;
Repeat for a specific number of iterations;
end for
Combine the evolved memeplexes;
Sort the population P in descending order of their fitness;
if termination = true **then**
Return the best solution;
end if
end procedure

the frog that has the best fitness is defined with X_g. Then X_w is improved in a loop. Equations 7 and 8 shows the improvement function of X_w:

$$D_i = Rand() \times (X_b - X_w) \tag{7}$$

$$\begin{aligned} X_w(new) &= X_w + D_i \\ -D_{max} &\leq D_i \leq D_{max} \end{aligned} \tag{8}$$

D_i shows the change in the ith frog position. Equation 8 shows a new position of Xw. $Rand()$ generated a random number between zero and one. D_{max} is the possible maximum changes in frogs position. If the procedure produces better position, $X_w(new)$ will be replaced with X_w; otherwise, X_g will be replaced with X_h in equation 7 and 8 and the equations will be repeated. But if $X_w(new)$ isnt better than X_w, then a new solution is produced randomly. The procedure is repeated for specific cycles. In section (b) modified shuffled frog leaping algorithm that is used for solving 0/1 knapsack problem is explained.

3.2. Modified shuffled frog leaping algorithm for solving 0/1 knapsack

Shuffled frog leaping algorithm cant solve the knapsack problem directly. So it is modified to solve the knapsack problem [10]. The shuffled frog leaping algorithm converges but sometimes it drops into a local optimal. To solve the problem we use a function to shuffle the frogs population. The changes of shuffled frog leaping algorithm are explained as follow:

3.2.1. **Producing initial population** Consider each frog as an n-bit number; n is the number of knapsacks dimensions.

3.2.2. **Variables discretization** Equation 8 should be discrete because 0/1 knapsack problem is a discrete problem. Equation 9 shows the discretization of $X_w(new)$ position.

$$t = {}^{1}/_{(1+exp(-D))}$$
$$X_w(new) = \begin{cases} 0, & \text{if } t \leq \alpha \\ X_w, & \text{if } \alpha < t \leq \frac{1}{2}(1+\alpha) \\ 1, & \text{if} t \geq \frac{1}{2}(1+\alpha). \end{cases} \tag{9}$$

The D variable is defined in equation 4 and is a constant variable.

3.2.3. **Constrained optimization** Constrained problems are more difficult to solve than unconstrained ones. In constrained ones, we should find a balance between finding the optimal solutions and satisfying constraints. One approach is using repair methods. The repair method that was used in this paper is as follow: first, all items are sorted in descending order of the ratio of their profit to weight. Then the repair method deletes the last one. The repair method is called when the sum of the solution's weight reaches more than the knapsack capacity.

3.2.4. **Genetic mutation** Sometimes the shuffled frog leaping algorithm traps in a local optimal. To solve the problem, a genetic mutation is used. The function changes the initial population a bit. This cause that it visits other optimal too.

3.2.5. **Termination condition** As mentioned in algorithm2, $iMax$ is the maximum value of the algorithm's iterations. If the termination condition is a constant number, it may be satisfied before converging. So the termination condition should be like $\lceil \frac{iMax}{20} \rceil \leq \triangle \leq \lceil \frac{iMax}{10} \rceil$.

3.3. Solving knapsack equations in MVCF method using shuffled frog leaping algorithm

According to mentioned items, the 0/1 knapsack problem is solved by shuffled frog leaping algorithm. So the equation 4 (knapsack equations) could be solved by modified shuffled frog leaping algorithm to find optimal queries in each cluster. The algorithm is explained in algorithm 3.

Algorithm 3 shows the algorithm for finding optimal queries.

In the previous step, frequent queries were found in each cluster. In this step, optimal queries are found according to storage space limitation. First knapsack equations should be written according to storage space, P_i and S_i. If the number of a specific clusters queries is d and the number of frogs is P, so P numbers with d-bits should be produced to create an initial population. Then the fitness function is calculated for each frog and frogs are sorted in descending order. Therefore frogs are divided into m memeplexes. X_b and X_w are calculated for each memeplex. Next, X_w is updated according to equation 9. The cycle is repeated for each time. Then genetic mutation is executed and the frogs are sorted in descending order. If the termination condition is satisfied it leaves the loop. X_g is the result. After Converting the number into the binary format, bits with the value equal to 1, means that the corresponding query should be materialized. For example, suppose that the final result is a number in binary format like 10100111. It means the bits number 0,2,5,6 and 7 are one and others are zero. So queries number 0,2,5,6 and 7 should be materialized. Since S_i is defined for each query, if the sum of S_i is more than storage space limitation, then repair method will be used.

In the last step, optimal queries are merged to result in a view for each cluster [35]. In this step, related tables will be natural outer join to find a view for the cluster.

Figure 3 and 4 show the flowcharts of the proposed algorithm. Figure 3 shows the first step of the algorithm that uses the SOM method for clustering.

Figure 4 shows the flowchart of the third step that uses shuffled frog leaping algorithm for finding optimal queries.

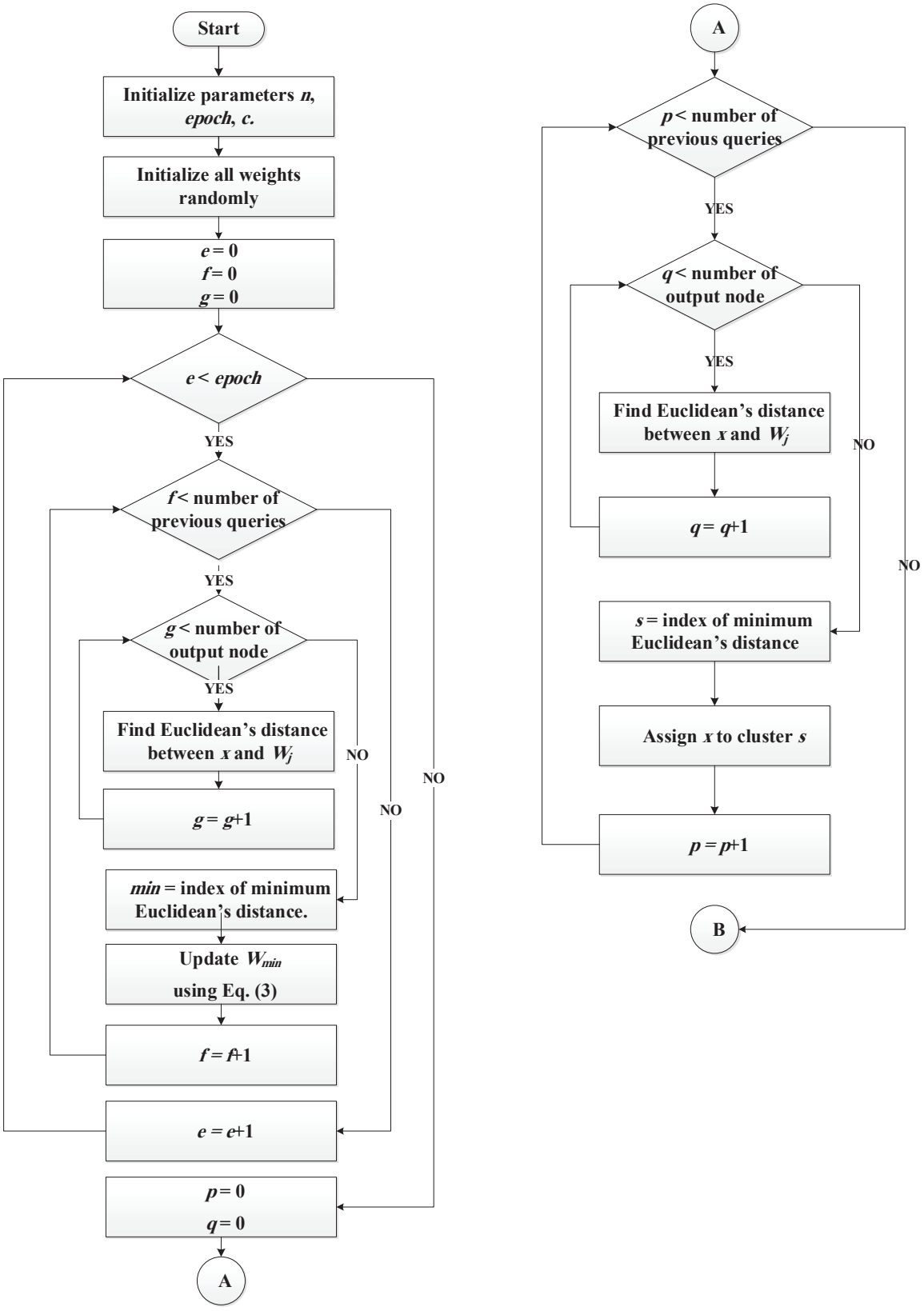

Figure 3. The first step of the flowchart for SRTTU-SF

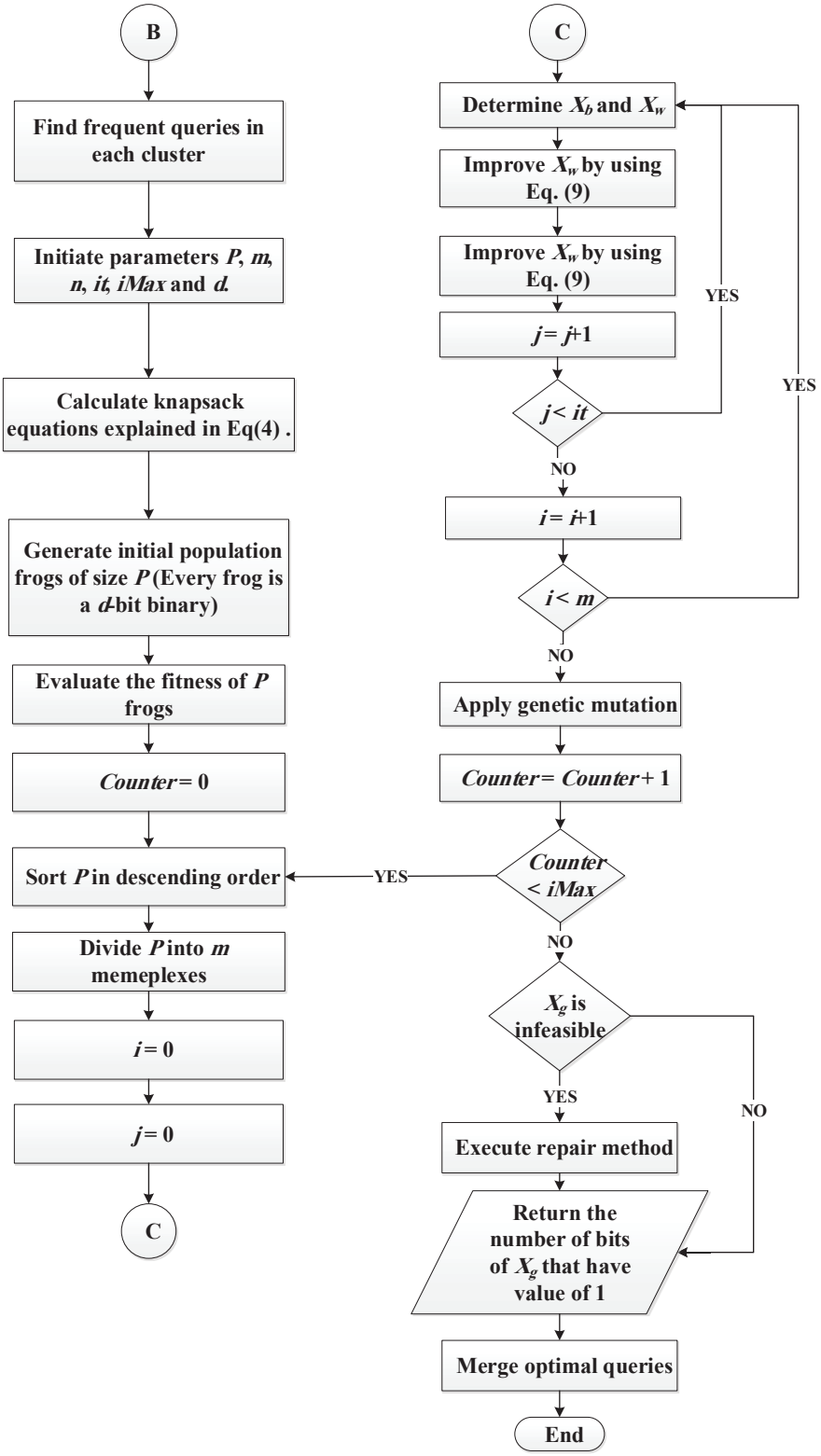

Figure 4. The third step of the flowchart for SRTTU-SF

Algorithm 3 Finding optimal queries

procedure FINDING OPTIMAL QUERIES
Input: the number of frogs P; the number of memeplexes m; the number of generation for each memeplex before shuffling n; the number of shuffling iterations it, the maximum number of iterations $iMax$; and Dimensions of the 0-1 Knapsack problem d
Output: best solution
for each cluster **do**
Calculate the equations explained in Eq. 4 like Eq. 6; //Solve the 0-1 knapsack problem equations:
Generate random population of P=n*m solutions (frogs), so that each solution is a d-bit binary number;
for each individual $i \in P$ **do**
Calculate fitness(i);
end for
Do
if termination = true **then**
Return the best solution;
end if
Sort the population P in descending order of their fitness;
Divide P into m memeplexes;
for each memeplex **do**
Determine the best and worst frogs;
Improve the worst frog position by using Eq. 9;
Repeat for a specific number of iterations;
end for
Combine the evolved memeplexes;
Apply genetic mutation on population;
Sort the population P in descending order of their fitness;
while (termination = False) **do**
Find the optimal queries according to best solution;
end while
if the best solution is an infeasible solution **then**
Execute repair methods to find optimal queries;
end if
end for
end procedure

4. Simulation results

The results and experiments are discussed in this section. The algorithms KD2013 and SRTTU-SF are simulated by Microsoft Visual Studio. We use Microsoft SQL Server database. Simulation data are generated by a loop of queries in the programming language. We produce some dimensional queries. The queries are produced randomly on our dimensions. The results arent sensitive to a special database because data was produced randomly. The data produced logically using dimension tables in C#.Net programming language with Microsoft Visual Studio. Our system has 4GB RAM, CPU 2.2 GHz Corei3. Our data warehouse has 10 dimension tables and one fact table.

To compare the algorithms, the KD2013 and SRTTU-SF algorithm are executed. There are two kinds of query; input and test queries. Both of them are produced randomly and logically using dimension tables. Input queries are the previous queries. After executing the proposed algorithm, test queries are used. The response time and storage space are important factors for view selection algorithms [35, 37]. The factors are studied in our experiments; so we use three factors which are explained:

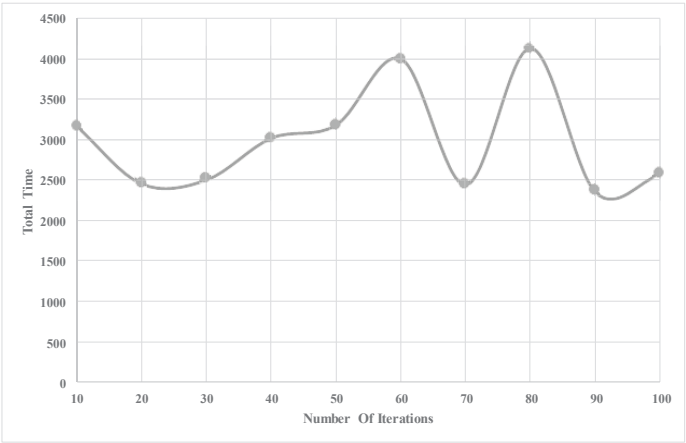

Figure 5. Comparison of total time according to increasing iteration value.

1. The number of materialized views' rows: After executing the view selection algorithm, the results view are materialized. This factor counts the number of materialized views' rows. In other words, this factor studies storage space.
2. Total time: This factor considers two parameters; first the time of executing the view selection algorithm and second, the response time of answering test queries.
3. Test time: The response time for answering test queries.

Each experiment was repeated five times and the result is the average of five numbers and then the diagrams were drawn.
First, the optimal value for the number of iteration,α parameter, learning rate (η) and $epoch$ parameter are found.
In figure 5 the number of test queries is constant (for example 60), the number of memeplexes is 4, max changes of D is 10, the value of α is 0.5 and maximum number of Xw is 5.
In figure 5 by increasing number of iterations and comparison of test time and total time in SRTTU-SF, the optimal value of iterations is calculated. According to figure 5 optimal value of iteration is 70.
Suppose that the number of test queries is constant (for example 60), the number of iteration is 70, the number of memeplexes is 4, max changes of D is 10 and maximum number of X_w is 5, so the optimal value of α is found. In figure 6 the value of α is increasing to find the optimal value of α. So the optimal value of α is 0.6.
The optimal value of the $epoch$ parameter is found in figure 7. In this figure, the number of test queries is constant (for example 60), the number of iteration is 70, the number of memeplexes is 4, max changes of D is 10, the maximum number of X_w is 5 and the value of α is 0.6.

According to figure 7, the optimal value of $epoch$ is 10.
Figure 8 founds the optimal value of learning rate (η). In this figure, suppose that the number of test queries is constant (for example 60), the number of iteration is 70, the number of memeplexes is 4, max changes of D is 10, the maximum number of X_w is 5, the value of is 0.6 and the value of α the $epoch$ parameter is 10.

According to Figure 8, the optimal value of learning rate (η) parameter is 0.01.
Now, the effect of improvement in the first step is analyzed. In figures 9, 10 and 11 the optimal values of parameters are supposed; the number of epochs is 10, and the learning rate is 0.01. Figure 9 shows the comparison between KD2013 and improvement of the first step of SRTTU-SF according to first factor (number of materialized records). In this figure, the horizontal axis represents the number of test queries and the vertical axis represents the number of materialized views' records.

According to Figure 9, it is obvious that the number of materialized views rows in the SRTTU-SF algorithm is less than the other. So the proposed algorithm needs less storage space than KD2013.

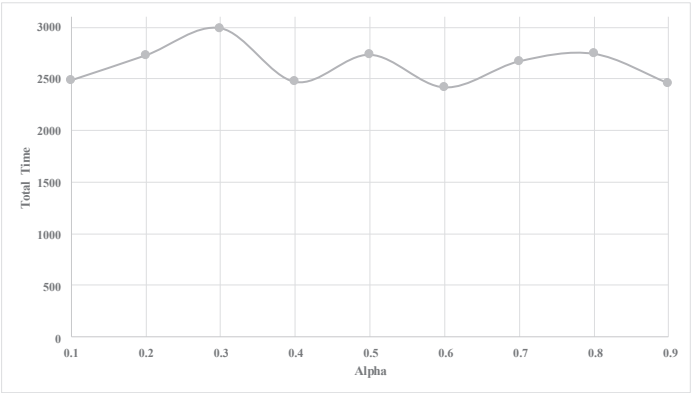

Figure 6. Comparison of test time and total total time according to increasing α.

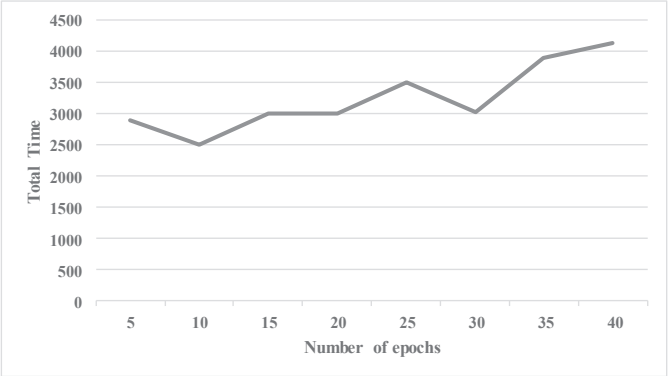

Figure 7. Comparison of total time according to increasing number of epochs.

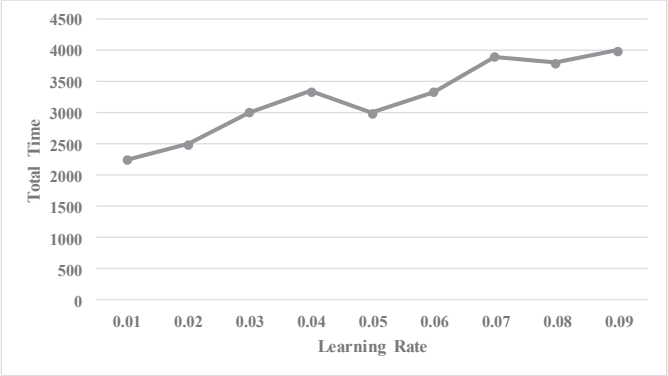

Figure 8. Comparison of total time according to increasing learning rate.

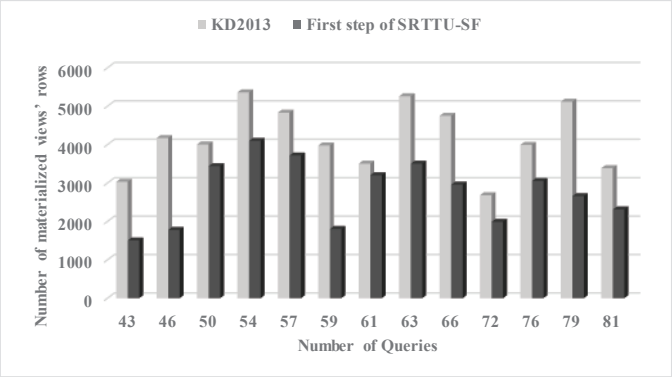

Figure 9. Comparison of materialized views records vs. increase of test queries according to the improvement of the first step of SRTTU-SF.

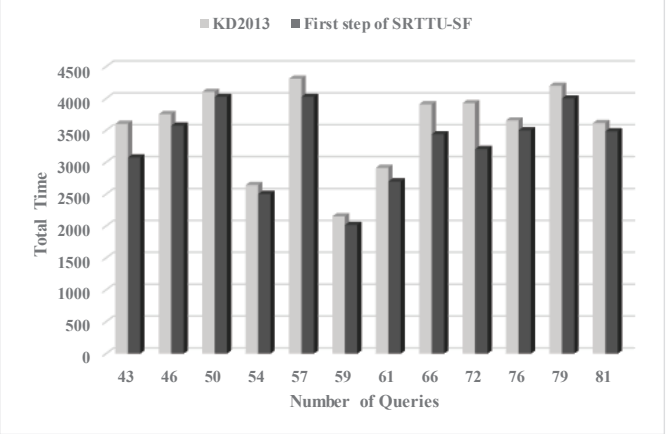

Figure 10. Comparison of total time vs. increase of test queries according to the improvement of the first step of SRTTU-SF.

Figure 10, shows the comparison of two algorithms according to factor 2 (total time). The horizontal axis shows the number of test queries and the vertical one shows total time; total time contains both executing time of the algorithm and response time for answering OLAP queries.

Figure 10 shows that the total time of the proposed algorithm is less than KD2013. Since time is very important in view selection algorithms, the proposed algorithm is better than KD2013 in time factor too.

Figure 11, shows the comparison of two algorithms according to test time. The horizontal axis shows the number of test queries and the vertical one shows test time.

Figure 11 shows that the test time of the proposed algorithm is less than KD2013.

According to our experiments, it is observed that the improvement of the first step is more efficient than KD2013 according to the mentioned factors.

Now, the effect of the improvement in the third step is probed. In figures 12, 13 and 14 the optimal values of parameters are supposed; the value of iteration (repetition of the algorithm) is 70, the number of memeplexes is 4, max changes of D is 10, the value of α is 0.6 and maximum number of X_w is 5.

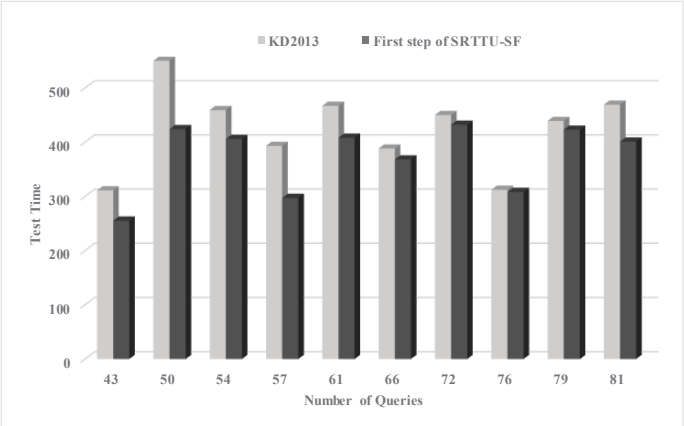

Figure 11. Comparison of test time vs. increase of test queries according to the improvement of the first step of SRTTU-SF.

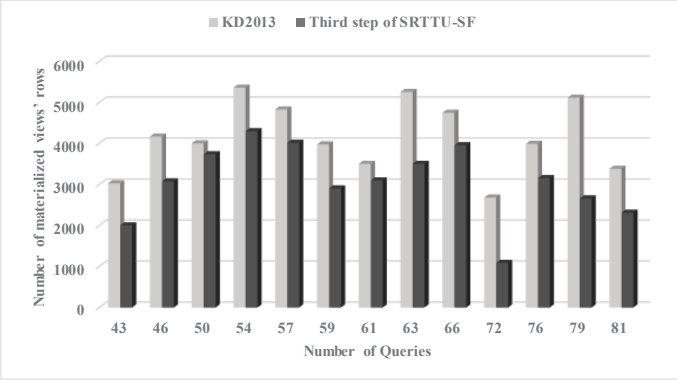

Figure 12. Comparison of the number of materialized views' rows vs. increase of queries according to the improvement of the third step of SRTTU-SF.

Figures 12, 13 and 14 show the comparison of the improvement of the third step of SRTTU-SF vs. KD2013 according to the number of materialized views' rows, total time and test time factors respectively.

In the figures, the horizontal axis shows the number of test queries and the vertical axis shows the experimental factors. They show that the improvement of the third step of the proposed algorithm is better than KD2013 according to our experimental factors.

Finally, we compare SRTTU-SF algorithm (which has two improvements in first and third steps) vs. KD2013. In figures 15, 16 and 17 optimal values are supposed. In these figures the value of iteration (repetition of the algorithm) is 70, the number of memeplexes is 4, max changes of D is 10, the value of α is 0.6, the maximum number of X_w is 5, the value of $epoch$ is 10 and the value of learning rate (η) is 0.01.

Figure 15 shows the differences between KD2013 and SRTTU-SF according to the first factor. In this figure, the horizontal axis shows the number of test queries and the vertical axis shows the number of materialized views' rows.

According to figure 15, it is clear that the size of materialized views in SRTTU-SF is less than KD2013. It means that the proposed algorithm needs less storage space, so it is better than the KD2013 according to storage space.

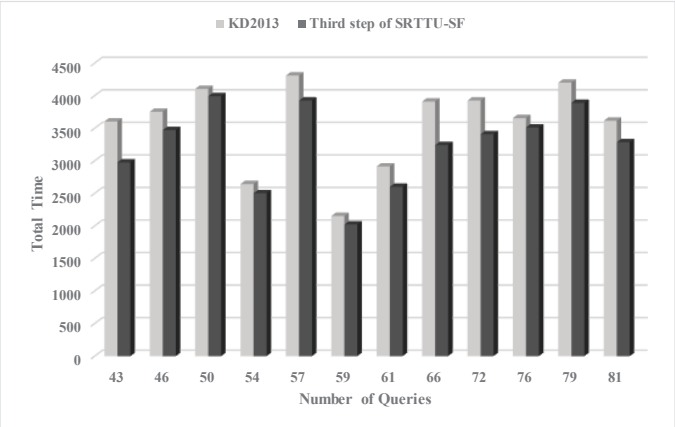

Figure 13. Comparison of total time vs. increasing queries according to the improvement of the third step of SRTTU-SF.

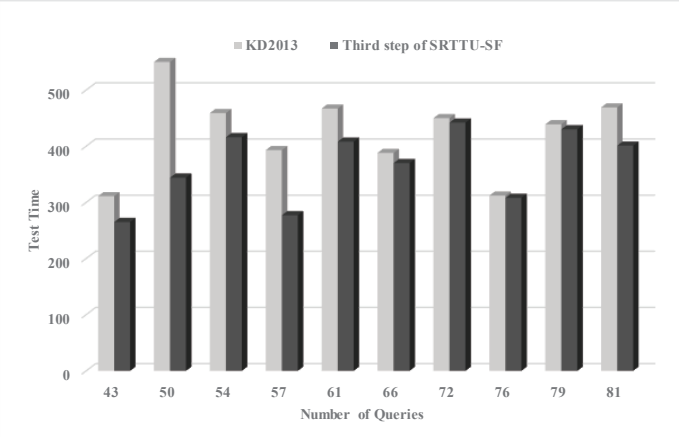

Figure 14. Comparison of response time vs. increasing queries according to the improvement of the third step of SRTTU-SF.

Figure 16 shows the comparison of two algorithms according to factor 2. The horizontal axis shows the number of test queries and the vertical axis shows total time; the time is the sum of executing the view selection algorithm and response time for answering test queries.

Figure 16 shows that the total time of the SRTTU-SF algorithm is less than KD2013.On the other hand, as time factor is one of the most important factors in view selection algorithms, figure 16 shows that the proposed algorithm is better than KD2013 according to total time.

Figure 17 shows the response time for answering to test queries. The horizontal axis shows the number of test queries and the vertical axis shows the response time for answering test queries.

According to figure 17, it is shown that the response time of the proposed algorithm is less than KD2013. So the proposed algorithm is better than KD2013 according to factor 3.

According to all experiments, it is shown that the proposed algorithm is better than KD2013 in three factors.

The improvement percentage for each step is described in table 1. According to table 1, it is obvious that the proposed algorithm has improvement in experimental factors.

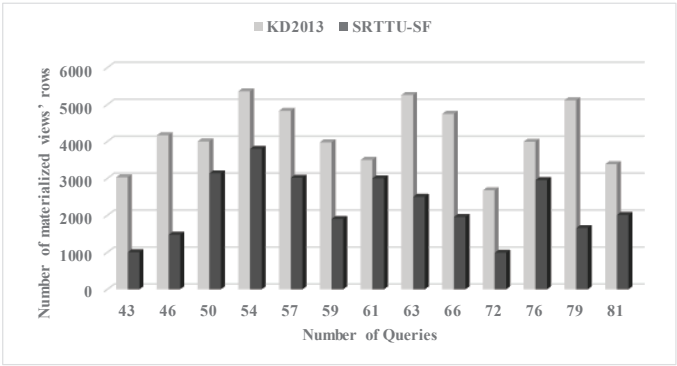

Figure 15. Comparison of the number of materialized views' rows according to the increase of test queries.

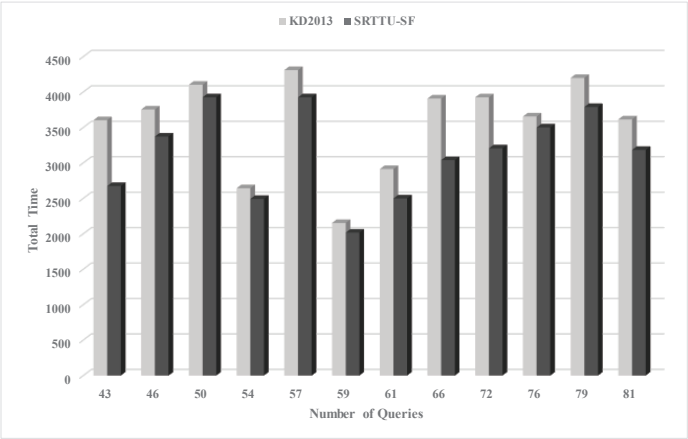

Figure 16. Comparison of total time according to increasing test queries.

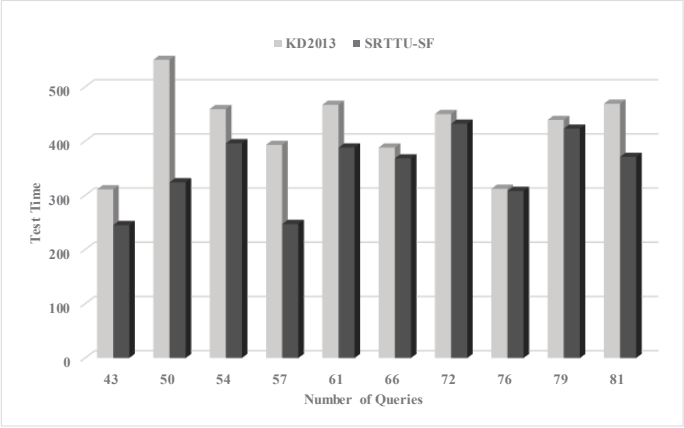

Figure 17. comparison of response time according to increasing test queries.

Table 1. The percentage of improvement

Algorithm	Factor1 (number of materialized views rows)	Factor2 (total time)	Factor 3 (test time)
Improvement of the first step of SRTTU-SF	33.25%	7.48%	11.96%
Improvement of the first step of SRTTU-SF	27.1%	9.14%	12.92%
SRTTU-SF	45.78%	11.82%	16.62%

5. Conclusion

In this paper, an algorithm for view selection was introduced. The algorithm is called SRTTU-SF. The proposed algorithm uses previous queries. First, previous queries are clustered and then frequent queries are found in each cluster. Therefore according to storage space, optimal queries are found in each cluster. In this stage, optimal queries are found using 0/1 knapsack algorithm. The knapsack algorithm is solved by shuffled frog leaping algorithm. Finally, optimal queries are joined to find only one view for each cluster. The proposed algorithm is an improvement of the KD2013 algorithm. SRTTU-SF algorithm improved the stages of clustering and finding optimal queries. Experimental results show that SRTTU-SF is more efficient than KD2013. The proposed algorithm has 14.22% improvement according to time factor (total and test time) and it has 45.78% improvement according to factor 3 (number of materialized views rows). We use the SOM algorithm to find clusters and shuffled frog algorithm to find optimal queries which improve KD2013 according to time and space factors.

REFERENCES

1. C. Parpoula, C. Koukouvinos, D. Simos , and S. Stylianou, *Supersaturated plans for variable selection in large databases*, Statistics, Optimization & Information Computing, vol. 2, no. 2, pp. 161–175, 2014.
2. E. Macedo, *Two-Step Semidefinite Programming approach to clustering and dimensionality reduction*, Statistics, Optimization & Information Computing, vol.3 no. 3 pp. 294–311, 2015.
3. T. Kumar, and S. Kumar, *Materialized view selection using iterative improvement*, advances in computing & inf. technology, vol. 3, pp. 205–213, 2013.
4. J. Han and M. Kamber, *Data mining Concepts and Techniques*, 3rd ed., Newyork, 2012.
5. M. Golfarelli, and S. Rizzi, *From Star Schemas to Big Data: 20+ Years of Data Warehouse Research*, A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years, Springer, vol. 31, pp. 93–107, 2018.
6. T. V. V. Kumar, G. Dubey, and A. singh, *Frequent Queries Selection for View Materialization*, Advances in Computing and Information Technology, vol. 177, pp. 521–530, 2013.
7. T. V. Kumar, and K. Devi, *Frequent Queries Identification for Constructing Materialized Views*, Electronics Computer Technology (ICECT), Kanyakumari, 2011.
8. M. Bakhshi, M.-R. Feizi-Derakhshi and E. Zafarani, *Review and Comparison between Clustering Algorithms with Duplicate Entities Detection Purpose*, International Journal of Computer Science & Emerging Technologies, vol. 3, no. 3, pp. 108–114, 2012.
9. O. Abu Abbas, *Comparisons Between Data Clustering Algorithms*, The International Arab Journal of Information Technology, vol. 5, no. 3, pp. 320–325, 2008.
10. K. K. Bhattacharjee, and S. P. Sarmah, *Shuffled frog leaping algorithm and its application to 0/1 knapsack problem*, Applied Soft Computing, vol. 19, pp. 252–263, 2014.
11. J. Yang, K. Karlapalem, and Q. Li, *Algorithms for materialized view design in data warehousing environment*, VLDB, vol. 97, 1997.
12. P. Kalnis, N. Mamoulis, and D. Papadias, *view selection using randomized search*, data and knowledge engineering, vol. 42, pp. 89–111, 2002.
13. C. Zhang, X. Yao, and J. Yang, *"Evolving materialized views in a data warehouse*, evolitionary computation, vol. 2, pp. 823–829, 1999.
14. C. Zhang, X. Yao, and J. Yang, *An evolutionary approach to materialized view selection in a data warehouse environment*, IEEE Transaction on systems, Man, and Cybernerics, Part C: Applications and Reviews, vol. 31, pp. 282–294, 2001.
15. J. Horng, B. Chang, B. Lui, and B. Kao *Materialized view selection using genetic algorithms in a data warehouse system*, Evolutionary Computation, vol. 3, 1999.
16. J. Horng, B. Chang, and B. Liu, *applying evolutionary algorithms to materialized view selection in a data warehouse*, soft computing, vol. 7, pp. 574–581, 2003.

17. S. Rizzi and, E. Saltarelli, *View materialization vs, indexing: balancing space constraints in data warehouse design*, Advanced information systems engineering, Klagenfurt, Austria, 2003.
18. D. Theodoratos, and W. Xu, *constructing search spaces for materialized view selection*, ACM International workshop on data warehousing and OLAP, washington, USA, 2004.
19. I. Mami, and Z. Bellahsene, *A survey of view selection methods*, ACM SIGMOD, vol. 41, no. 1, pp. 20–29, 2012.
20. C. A. Dhote and M. S. Ali, *Materialized view selection in data warehousing: a survey*, Journal of Applied sciences, vol. 9, no. 3, pp. 401–414, 2009.
21. J. S. Sohn, J. H. Yang, and I. J. Chung, *Improved view selection algorithm in data warehouse*, IT Convergence and Security, pp. 921–928, 2013.
22. W. Xu, D. Theodoratos, C. Zuzarte, X. Wu, and V. Oria, *A dynamic view materialization scheme for sequences of query and update statements*, Data Warehousing and Knowledge Discovery, pp. 55–56, 2007.
23. N. Daneshpour, and A. Abdollahzadeh Barfourosh, *Dynamic view Management System for Query Prediction to view materialization*, International Journal of Data Warehousing and Mining, vol. 7, no. 2, pp. 67–96, 2011.
24. I. Mami, R. Coletta, and Z. Bellahsene, *Modeling view selection as a constraint satisfaction problem*, in International Conference on Database and Expert Systems Applications, France, 2011.
25. I. Mami, Z. Bellahsene, and R. Coletta, *View selection under multiple resource constraints in a distributed context*, in International Conference on Database and Expert Systems Applications, Vienne, 2012.
26. I. Mami, Z. Bellahsene, and R. Coletta, *A Declarative Approach to View Selection Modeling*, Transactions on Large-Scale Data-and Knowledge-Centered Systems, pp. 115–145, 2013.
27. Z. Asgharzadeh, R. Chirkova, and Y. Fathi, *Exact and inexact methods for selecting views and indexes for olap performance improvement*, International conference on Extending database technology: Advances in database technology, France, 2008.
28. R. Huang, R. Chirkova, and Y. Fathi, *Advances in Databases and Information Systems*, in Deterministic view selection for data analysis queries: Properties and algorithms, Berlin, Springer Berlin Heidelberg, pp. 195–208, 2012.
29. T. V. Kumar, and M. Haider, *Query answering-based view selection*, International Journal of Business Information Systems, vol. 18, no. 3, pp. 338–353, 2015.
30. M. K. Sohrabi, and H. Azgomi, *TSGV: a table-like structure-based greedy method for materialized view selection in data warehouses*, Turkish Journal of Electrical Engineering & Computer Sciences , vol. 25, no. 4, pp. 3175–3187, 2017.
31. M. K. Sohrabi, and V. Ghods, *Materialized View Selection for a Data Warehouse Using Frequent Itemset Mining*, Journal of Computers, vol. 11, no. 2, pp. 140–148, 2016.
32. A. Gosain, and K. Sachdeva, *Materialized View Selection Using Backtracking Search Optimization Algorithm*, Intelligent Engineering Informatics. Springer, vol. 695, pp. 241–251, 2018.
33. A. Gosain, and H. Madaan, *Query Prioritization for View Selection*, Advances in Intelligent Systems and Computing. Springer, vol. 518, pp. 403–410, 2018.
34. M. Megahed, R. M. Ismail, N. L. Badr, and M. Fahmy Tolba, *An Enhanced Cloud Based View Materialization Approach for Peer-to-Peer Architecture*, Multimedia Forensics and Security. Springer, vol. 115, pp. 77–95, 2017.
35. T. V. Kumar, and K. Devi, *Materialised view construction in data warehouse for decision making*, International Journal of Business Information Systems, vol. 11, no. 4, pp. 379–396, 2012.
36. T. V. V. Kumar, A. Singh and G. Dubey, *Mining Queries for Constructing Materialized Views in a Data Warehouse*, Advances in Computer Science, Engineering & Applications, pp. 149–159, 2012.
37. T. V. V. Kumar, A. Goel, and N. Jain, *Mining information for constructing materialised views*, Int. J. Information and Communication Technology, vol. 2, no. 4, pp. 386–405, 2010.