International Journal of Applied Metaheuristic Computing
Volume 13 • Issue 1

Multi-Objective Big Data View Materialization Using MOGA

Akshay Kumar, Jawaharlal Nehru University, India
T. V. Vijay Kumar, Jawaharlal Nehru University, India*

ABSTRACT

The COVID-19 pandemic has resulted in large scale of generation of big data. This big data is heterogeneous and includes the data of people infected with corona virus, the people who were in contact with an infected person, demographics of infected persons, data on corona testing, a huge amount of GPS data of people location, and a large amount of unstructured data about prevention and treatment of COVID-19. Thus, the pandemic has resulted in producing several Zettabytes of structured, semi-structured, and unstructured data. The challenge is to process this big data, which has the characteristics of very large volume, brisk rate of generation and modification, and large data redundancy in a time-bound manner to take timely predictions and decisions. Materialization of views for Big data is one of the ways to enhance the efficiency of processing of the data. In this paper, Big data view selection problem is addressed as a bi-objective optimization problem using multi-objective genetic algorithm.

KEYWORDS

Big Data, Multi-Objective Genetic Algorithm, Multi-Objective Optimization, View Materialization

1. INTRODUCTION

Information is one of the important criterion for the survival of Businesses in the present world. Big data applications are required to process large amounts of data, which is cleaned, integrated, and presented in different forms, for making optimal business decisions. Big data has four basic characteristics defined as the 4 V's of Big data - volume i.e. a large size, velocity, i.e. a high rate of data generation, variety, i.e. heterogeneity in data, and veracity, i.e., the trustworthiness of data (Jacobs, 2009; Zikopoulos et al., 2011; Gupta et al., 2012, Kumar et al., 2015). Big data is generated from a variety of data sources, which generally produces inconsistent data at different rates, leading to complex and challenging data cleaning and integration processes. In addition, Big data in its raw form is not suitable for business decisions, rather it is processed to create useful information for the benefit of an organization. This is also referred to as the value of Big data. Big data visualization, validity, vulnerability and volatility are other important considerations of Big data processing (Khan et al., 2014; Gandomi et al., 2015; Firican, 2017).

Big data applications process data in real time to enable timely decisions for helping the organization or society, for whom the application has been designed. (Luo et al., 2016) suggested the development of Big data applications for health care systems, where large amounts of clinical and hospital data could be used to forecast the spread of infectious diseases. One such recent Big data application is concerned with the spread of the *COVID 19* pandemic (Chenghu et al., 2020). This

DOI: 10.4018/IJAMC.292499 *Corresponding Author

application models the spread and future healthcare infrastructure requirements for *COVID 19* patients. The application uses the Big data related to corona virus positive cases that includes the data of people who got infected with corona virus, people who came in contact with these infected people, the final outcome of the infected cases including recoveries, the number of tests conducted, demography of corona virus infected people, isolation data of corona virus infected people etc. This heterogeneous data is increasing at a massive rate with the global spread of the pandemic. Thus, tracking the spread of *COVID 19* disease, and assessing future healthcare infrastructure, are required to process Zettabyte of geographical, semi-structured and unstructured data. (Chenghu et al., 2020) identified the role of Geographical Information Systems (*GIS*) and Big data for mapping, tracking, and modeling the spread of corona infections using data visualization. Big data with its predictive power can play a major role for various support processes that will help in the efforts to control of the pandemic. (Chenghu et al., 2020) identified the major challenges in implementation of a *GIS* system for the development of an application for monitoring *COVID 19* spread. These challenges were concerned with integration of the redundant data generated from different data sources, the dynamic mapping of the sources of the epidemic and their contacts, analyzing the transmission of the disease to newer geographical areas, and assessing the risk of non-availability of resources in various geographical areas to deal with the pandemic. All these challenges required appropriate and efficient query processing on the Big data related to the pandemic. (Shneiderman, 2020) lists the data visualization applications, which model the spread of the pandemic and support policy decision making by the Government. One such effort listed in (Shneiderman, 2020) is a *COVID19* dashboard by Lauren Gardner with her team at John Hopkins University. It highlights the importance of visualization for such worldwide disasters. Thus, Big data applications, which impact social life, have to deal with very high volumes of data, high speed of data generation, heterogeneity of data generating sources, and large data redundancy. Big data application for *COVID 19* requires extensive data cleaning, integration and processing to generate accurate information, in different visual forms, having high information value. It should create reliable, dynamic and actionable knowledge in a timely manner, which can save many precious lives. View materialization is one of the techniques used for enhancing the speed of processing of Big data, which can result in faster decision making. This paper addresses the bi-objective Big data view materialization problem using the Multi-Objective Genetic Algorithm (*MOGA*).

This paper is organized as follows: Section 2 discusses view materialization followed by the Big data view selection problem in the context of Big data view materialization in section 3. Big data view selection using *MOGA* is discussed in section 4 and an example illustrating the use of the proposed *MOGA* based Big data view selection algorithm to select Big Data views for materialization is illustrated in section 5. Experimental results are given in section 6. Section 7 is the conclusion.

2. VIEW MATERIALIZATION

A database system can have a large number of views, which can be generated as a result of queries over a database schema. A sub-set of these views can be materialized to enhance the efficiency of processing of frequent queries. However, view materialization results in an increase in the maintenance cost of the database, as both, the database and the views need to be maintained. In addition, view materialization increases the overall size of the database system, as materialized views are also stored with the database. Therefore, the objective of view materialization is to minimize both the query response times and the maintenance cost, keeping in the constraint on the maximum cumulative size of the views. (Chirkova et al., 2001) formally defined the problem of selection of views for materialization in the context of relational database management and data warehouse as given below:

Given the relational schema (R), a set of queries (Q) on R and an available size of database storage (S), the view selection problem identifies a set of views for materialization that minimizes the estimated cost of query processing over R, with the constraint on the overall stored size of the database (S) and materialized views.

The materialization of views, however, results in materialized view maintenance cost, as updates on the database are also propagated to the data of the related views. (Kenneth et al., 1996) represented views using directed acyclic graphs, with the aim of identifying the common sub-graphs or sub-expressions of these graphs. These query sub-graphs can also be materialized to reduce the view maintenance cost, as these sub-views could be a part of several materialized views. (Mistry et al., 2001) presented a greedy algorithm for the selection of views for materialization having an optimal maintenance plan. (Gupta & Mumick, 1995) defined four aspects of the view maintenance problem viz. identification of integrity constraints and access rights on view data; expected data manipulations on the view data; the relational operators used to create the views; and the possibility of incremental updates on the view data. Thus, the problem of view materialization requires to select a set of views for materialization, which minimizes the cost of query processing for the given set of queries (Q) and the cost of maintenance of the materialized views, for a given size of the database (S) (Mami & Bellahsene, 2012).

Rapid growth in operational data of organizations resulted in the development of data warehouses. A data warehouse is an integration of operational data of an organization for answering analytical queries, results of which are used for making business decisions. A data warehouse is organized as multi-dimensional data cubes, with each cell of the cube representing a measured value. These data cubes of the data warehouse are represented using the dimensional tables and fact tables, where the dimension tables are used for grouping of the values stored in the fact tables. (Harinarayan et al., 1996) represented the data warehouse queries using a lattice structure consisting of a set of nodes of queries, with two nodes having a dependence relation if a query can be answered using the result of another query. Each of the node in the lattice is a candidate view for materialization.

The problem selection of view for materialization was extensively studied for data warehouses (Harinarayan, 1996; Gupta, 1996; Roussopoulos, 1998; Chirkova et al. 2001; Mami et al., 2012). The selection of views for materialization is an *NP-Hard* problem for data warehouse (Harinarayan et al., 1996), therefore, the problem cannot be solved by standard algorithms. The view selection problem has been addressed using the empirical approach (Agrawal et al. 2000) or heuristic approaches (Harinarayan, 1996; Gupta, 1996) or meta-heuristic approaches (Arun & Vijay Kumar, 2015a, 2015b, 2017a, 2017b; Vijay Kumar & Arun, 2016, 2017, Vijay Kumar & Kumar, 2014, 2015; Kumar & Vijay Kumar, 2018). This problem was also formulated as a bi-objective view selection problem and solved using multi-objective evolutionary algorithms *VEGA* (Prakash & Vijay Kumar, 2019a), *MOGA* (Prakash & Vijay Kumar, 2020a), *SPEA-2* (Prakash & Vijay Kumar, 2019b) and *NSGA-II* (Prakash & Vijay Kumar, 2020b).

With an increase in the complexity of data and their interrelationships, object oriented database management systems (*OODBMS*) were developed. View materialization in case of *OODBMS* requires dynamic classification of objects into view classes (Kuno et al., 1995). A materialized view class in *OODBMS* maintains links to the objects of that view class. The modification of data of the object may require re-classification of the object into another materialized view class. This change will require adjusting the object links of the materialized view classes (Kuno et al., 1995).

With the evolution of Web 2.0 and *NoSQL* databases, semi-structured database systems became popular. (Abiteboul et al., 1997) proposed that a view on semi-structured data should be created as a result of a query on semi-structured data and it should be treated as an independent entity. (Tang et al., 2009) modeled the views on the semi-structured *XML* data using a directed acyclic graph, with the nodes representing the views and the arcs represent the subset relationships between the views. The paper extends the model of view materialization for relational databases to *XML* database, and also presents a greedy algorithm to select views for materialization on the semi-structured data. (Abiteboul, 1999; El-Sayed M. et al., 2006) proposed an incremental model for maintenance of *XML* views using query sub-graphs. However, the view materialization for semi-structured data alone cannot be used to model view materialization for Big data, which is presented in subsequent sections.

With the popularity of social media and e-commerce applications, a large amount of unstructured data in the form of video, audio and large texts is being generated. Such data contains useful information, which needs to be extracted. Such data is drawn from many heterogeneous sources having variable data rates, referred to as variability of data (Gandomi & Haider, 2015). Unstructured data requires pre-processing and integration with structured and semi-structured data. (Gandomi & Haider, 2015) presents several methods that can be applied on unstructured data to extract useful information. (Yafooz et al., 2013) suggested three models of representing unstructured data, viz. linking relational entities and unstructured data; creating a separate model for unstructured data using an identifying relation; using queries to process unstructured data, which can be stored with the structured data or integrated with inverted indexes. Thus, in general, meaningful information is extracted from data repositories for making informed decisions. Materialization of selected views enhances the efficiency of this process. The view materialization for Big data is discussed in the next section.

3. BIG DATA VIEW MATERIALIZATION

Big data architectures process large volumes of heterogeneous data using distributed file system (*DFS*) and map-reduce frameworks (Hadoop 2008; Hadoop 2012; Dezyre 2015). Therefore, Big data view materialization should be defined in the context of the Big data store, which uses (*DFS*) and the Map-Reduce framework. (Kumar &Vijay Kumar, 2021a) defined the Big data view materialization problem involving Big data characteristics viz. large volume, heterogeneous data, high data generation rate and low veracity. It also presented a model for cost computation and presented a greedy algorithm for the selection of Big data views for materialization. Query attributes (Q_A), which are the attributes of structured or semi-structured or unstructured data, are derived from the most frequent queries posed on Big data. Q_A and their inter-relationships form the basis of selection of candidate views for Big data view materialization (Kumar & Vijay Kumar, 2021a). A Big data application can be designed using an extended semantic model, which includes Big data entities and their relationships. A Big data entity can consist of structured, semi-structured and unstructured data. This semantic model and the workload queries on a Big data application can be used to identify Q_A and their interrelationships. Fig.1 illustrates the semantic model of an application for tracking a *COVID 19* infected person and the people who came in their contact. The data, which may be stored for this application, includes the patient details, their past history of diseases, their location, and the details of people who came in contact of such patients. Fig. 1 shows the semantic model of this data. It consists of the Big data entities *Patient*, *Location* and *Contact History* and a relationship *Tested*. The Big data entity *Patient* can be modeled as a two structured relation name *Patient(Id, Name, DOB)* and *DiseaseHistory*(*Id, Disease, dateofillness*); or a semi-structured document named *Patient* having the attribute *Id* and *XML* tags (*Name, DOB, DiseaseHistory*(*Disease, dateofillness*)). The *Location* entity can also be modeled in a similar way. In addition, the *Location* entity includes the data dimensions *Country* and *City*, which have a dependence relationship *Country* → *City*. The *Contact History* is a Big data entity having a large amount of semi-structured and unstructured data. This entity has been represented as a big rectangle in Fig. 1. It stores large amounts of unstructured and semi-structured data of those people who came in contact with an infected person. The details of level 1 contacts (Fig. 1) is semi-structured data whereas Level 2 contacts, and the various activities performed by these contacts, is unstructured data (Fig. 1). The relationship named *Tested* can be modeled as a <date: result> pair, which can store a sequence of test results of a patient as semi-structured data. It may be noted that the relationship between the *Location* and the *Patient* is many-to-many as a patient can travel to many different locations. However, no cardinality has been shown for *Contact History* as it is primarily unstructured data, which needs to be processed and related with the Q_As.

Fig. 1 can also be used for the identification of Q_A and their interrelationships from the model, which can then be used to create a view structure as shown in Fig 2. Such directed view structure graph can be used to identify candidate views and the dependence between the views, and thus it can be used to create alternative query evaluation plans using materialized views.

Figure 1. An extended semantic model

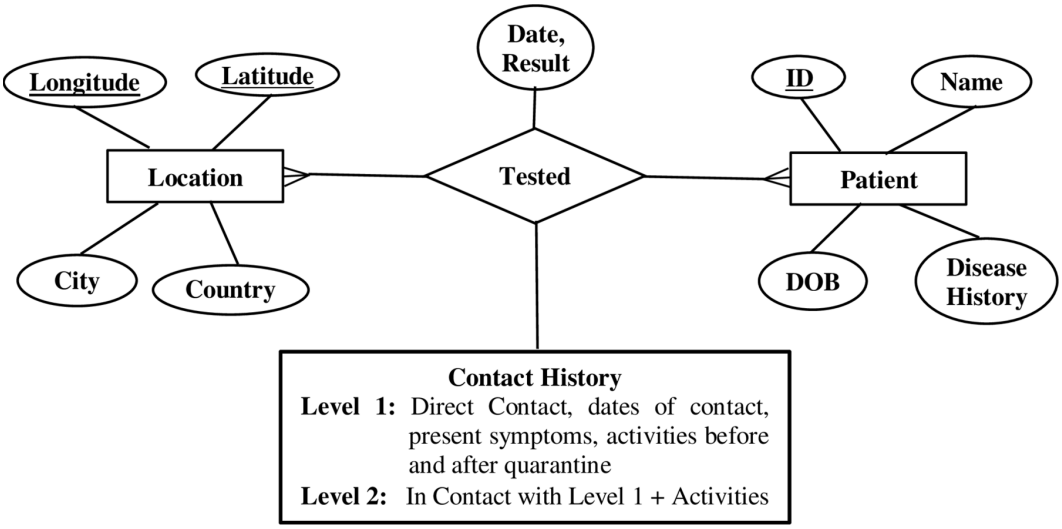
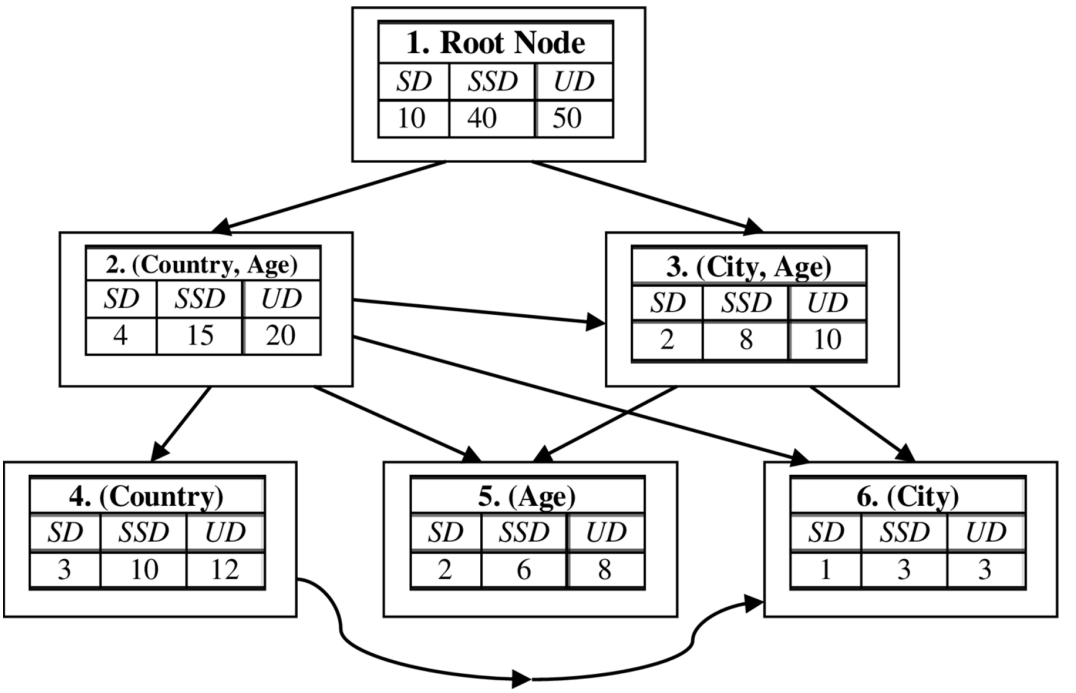


Fig.2 shows the structure of Big data views and their relationships. A node in Fig.2 represents Big data views related to specific query attributes. For example, node 2 represents the structured, semi-structured and unstructured data of *COVID 19* patients and their contacts selected/organized using the Q_A C*ountry* and *Age*. The estimated size of each type of data is also represented in the node. It may be noted that the size is represented as the number of stored blocks of Big data. Any possible combination of structured, semi-structured and unstructured data of a node can be considered to represent a Big data view.

A view can be identified as one of the candidate views for materialization, if the related Q_As are part of any one of the frequent queries. For example, a query which finds the city wise list of persons who came in contact of an infected person, involves Q_A *City*. This query can be answered using a view that can be generated using semi-structured data of node 6 (Q_A *City*) of Fig. 2. On the other hand a query that seeks the city wise details of activities performed by people who are quarantined can be answered using view created using unstructured data of node 6 (Q_A *City*) of Fig. 2. In addition, it can be observed from Fig.2, that a query which can be answered by a view created from node 6, can also be answered by views created from node 2 or node 3 or node 4, provided they are on similar type of data. Thus, a large number of candidate views of the order of $2^{|Q_A|}$ can be generated for Big data view materialization, as suggested in (Kumar & Vijay Kumar, 2021a; Kumar & Vijay Kumar, 2021b). It may be noted that the root node can be used to create three views, viz. one each for structured, semi-structured and unstructured data respectively, which are assumed to be materialized.

The selection of Big data views for materialization has been defined as a multi-objective problem in the context of the Big data warehousing tool - Hive (Goswami et al., 2017). This paper proposed three objectives for Big data view materialization. These objectives were - to minimize the query processing cost using a set of materialized views; to minimize the materialized view maintenance cost; and to minimize the number of materialized views, while maximizing the storage space (Goswami et al., 2017). The paper also suggested to use two multi-objective evolutionary algorithms, viz. the multi-objective differential evolution algorithm (*MOEA*) and the non-dominated sorting evolutionary algorithm (*NSGA-II*), to solve the multi-objective view materialization problem. However, their proposed Big data view materialization did not incorporate the Big data characteristics as suggested in (Kumar & Vijay Kumar, 2021a). Also, (Goswami et al., 2017) used map-reduce time in seconds over a single node to estimate the cost of query processing and materialized view maintenance. These

Figure 2. Structure of Big Data Views for query attributes Country, City and Age of a patient with dependency Country→City (S-Structured, SS-Semi-structured, U-Unstructured, D-Data)



cost estimates are dependent on the state as well as the number of map-reduce nodes. In addition, the paper did not suggest any mechanism for the identification of candidate views.

However, (Kumar & Vijay Kumar, 2021a) proposed Big data view materialization that incorporated Big data characteristics like volume, heterogeneity, veracity and the rate of data generation. In addition, query frequency was used as one of the parameters for Big data view materialization. The paper also derived a model for computation of total query evaluation cost, which was the sum of the query evaluation cost of the queries and the update processing cost of the materialized views. The paper suggested to use the stored size of the Big data blocks, which is a more robust measure, for computing the query evaluation cost and materialized view update cost. The paper proposed a greedy algorithm, which minimizes the total cost of query evaluation, to select the Big data views for materialization. This single objective Big data view materialization problem, was designed as a bi-objective Big data view materialization problem (Kumar & Vijay Kumar, 2021b). This bi-objective problem, which was solved using the vector enabled genetic algorithm (*VEGA*) in (Kumar & Vijay Kumar, 2021b), is discussed next.

3.1 Big Data View Selection Problem

Big data views are selected in order to optimize the processing of frequent Big data queries. This view selection problem has two basic objectives (Kumar & Vijay Kumar, 2021b), viz. the minimization of query evaluation cost of the most frequent queries and the minimization of the update processing cost of the views. This minimization is subject to the constraint on the cumulative size of the materialized views (Kumar & Vijay Kumar, 2021b). These two objectives as proposed in (Kumar & Vijay Kumar, 2021b) are given below:

3.1.1 Minimization of Query Evaluation Cost (QEC_{BDV})

The QEC_{BDV} is the cost of evaluating the workload queries, for a given set of materialized views. QEC_{BDV} is computed using equation (2) from the value of minimum cost of evaluating a query (MCQ_i) and the frequency of that query (f_i). MCQ_i of i^{th} query is computed using equation (1) for all possible j query evaluation plans, where each of these plans may involve k views. The computation of MCQ_i involves - update factors of the materialized views (ufv_k), which represent the ratio of increase in size of the view during a window of time, to the initial size of view; update factors of non-materialized view (uf_k), in case a view is not materialized, which is defined as the ratio of the increase in the size of the data that is required to compute a view during a specific window of time, to the initial size of data that is required to compute the view; the cost of materialized views (CMV); which is the cost of processing of the materialized view, in terms of stored blocks of Big data, to answer a query on the view; cost of data (CV), which is the cost of processing of the data, in terms of stored blocks of Big data, that is required to be processed to answer a query on the view, if the view is not materialized. The variable m_k in equation (1) is a binary variable, which represents the materialization status of a view. Thus, m_k=1, if the k^{th} view is materialized, else m_k=0. The equation (1) as given in (Kumar &Vijay Kumar, 2021b), to computes MCQ_i is given below:

$$MCQ_i = Min_j\left[\sum_k\left\{m_k \times CMV_k \times\left(1+\frac{1}{2}\times ufv_k\right)\right\}+\left\{\left(1-m_k\right)\times CV_k \times\left(1+\frac{1}{2}\times uf_k\right)\right\}\right] \tag{1}$$

After computing the MCQ_i for every query, the QEC_{BDV} is arrived at by computing the product of MCQ_i with the frequency of the queries (f_i) using equation (2) as given in (Kumar & Vijay Kumar, 2021b), which is given below:

$$QEC_{BDV} = \sum_{i=1}^{n}\left(MCQ_i \times f_i\right) \tag{2}$$

In equation (2) n represents the number of workload queries.

3.1.2 Minimization of update cost (UPC_{BDV})

In most Big data applications, inserting new data is more frequent than modifying old data. In addition, rather than correcting such old semi-structured or unstructured data in a data store, additional data is added to a Big data store with the corrected information. Thus, the semi-structured and unstructured data, which has low data integrity, results in generation of additional Big data. This additional data is referred to as UMV, and is processed to update materialized views. Integrity factor of view data (I_m) is the measure of integrity of Big data, which is also used to compute the update processing cost of Big data views (UPC_{BDV}) (Kumar & Vijay Kumar, 2021b). The UPC_{BDV}, as given in equation (3) (Kumar & Vijay Kumar, 2021b), is given below:

$$UPC_{BDV} = \sum_i\left(m_i \times UMV_i \div I_{m_i}\right) \tag{3}$$

(Kumar & Vijay Kumar, 2021b) have shown that the two objectives conflict with each other, as the larger Big data views can process a larger number of queries, which can result in the minimization of QEC_{BDV}, however, it results in a higher UPC_{BDV}. These two objectives are subject to a constraint on the cumulative size of the Big data views. Thus, the problem of selection of Big data views for materialization is a bi-objective optimization problem and can be solved using the multi-objective

variant of the genetic algorithm. The costs, viz. *QEC* and *UPC,* are computed in terms of the number of stored blocks of Big data (Kumar & Vijay Kumar, 2021b). In this paper, a Multi-objective Genetic Algorithm (*MOGA*) has been used to solve the bi-objective Big data view selection problem discussed above. Big data view selection using *MOGA* is discussed next.

4. BIG DATA VIEW SELECTION USING *MOGA*

Multi-Objective Genetic Algorithm (*MOGA*) is used for the simultaneous optimization of conflicting objectives. *MOGA* produces a number of non-dominated solutions for an optimization problem. In *MOGA*, a random population of solutions, represented by a chromosome, are created. In this paper, the permutation encoding of the chromosome is used. Next, the rank of every solution in the initial population, is computed by identifying the solutions that dominate it. A solution vector SV_i dominates a vector SV_j if one of the following holds:

(1) SV_i dominates SV_j for all objectives; or
(2) SV_i is equivalent to SV_j for n-m objectives and SV_i dominates SV_j in the remaining m objectives (where, m can be any value from 1 to n-1)

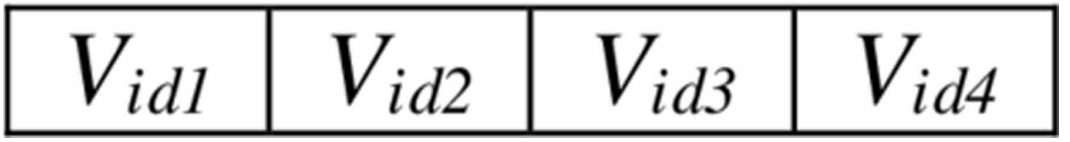
The dominated count of a solution is the count of the solutions that dominate it. After the dominated count for each solution is computed, it is assigned a rank, which is computed by adding 1 in the dominated count, i.e., if n is the number of solutions which dominate the given solution vector, then its rank will be n+1. Thus, the solution vectors which are not dominated by any other solution vector are assigned the highest rank. Rank forms the basis for assigning a raw fitness to a solution vector by using the linear mapping function. However, the raw fitness of the solution vectors that have the same rank is not the same, therefore, the raw fitness is averaged on the number of solution vectors, which have the same rank. The average fitness, in general, for the same rank solutions are the same and the highest rank (rank 1) solution vectors have the highest average fitness. The same rank solutions are, then compared for diversity using the Niche count. The diverse solutions are extracted using the shared fitness and the scaled shared fitness. The scaled shared fitness is used as the basis for the proportionate selection operator (Goldberg, 1989). The proportionate selection, in general, favors the elitist solutions. Thus, the solution, which has the higher shared scaled fitness, has a higher probability of selection to the mating pool. The basic principle here is to find diverse solutions, while maintaining the elitist bias. Next, the crossover and mutation are performed on the solutions in the mating pool to generate the population for the next generation. In this paper, a single point modified crossover operator (Davis, 1985) with a probability of crossover (p_c); and a random mutation operator (Goldberg, 1989) with a probability of mutation (p_m) are used. *MOGA* is run for predetermined number of generations, thus, creating a set of diverse, higher ranked solutions.

A *MOGA* based Big data view selection algorithm ($BDVSA_{MOGA}$) that selects Big data views for materialization is proposed in this paper and is discussed next.

4.1 $BDVSA_{MOGA}$

$BDVSA_{MOGA}$ is an evolutionary algorithm designed to solve the bi-objective view materialization problem. The inputs to the algorithm are - the list of frequent queries (Q), the query frequency vector (f) over a specific period of time (T), the candidate views for materialization (CV_{DB}), the query evaluation plans for each query ($QtVs$) in the context of CV_{BD}, the cost of using a candidate view in CV_{BD}, if it is materialized (CMV_i); or the cost, if the candidate view is not materialized (CV_i), the update factors (uf), the cost of updating a view (UMV_i), the integrity factor (I_m) for each view and the number of Big data views to materialize (k).

Figure 3. Structure of View Vector of size 4 (VV4)



This algorithm selects a set of Big data views for materialization from a large set of candidate views. Therefore, permutation encoding has been used to define the chromosome or view vector for the algorithm. Each candidate view is identified by a unique view identifier (V_{id}). The view vector (VVk) represents a set of the materialized views of size k. The objective of $BDVSA_{MOGA}$ is to find the good view vectors, which optimize the query processing and query update processing costs. The algorithm is run for a specific size of view vector (k) and collects the non-dominated view vectors as the output. Fig. 3 represents a view vector of size 4 ($VV4$). The view vector $VV4$ consists of four distinct Big data views (V_{id}).

$BDVSA_{MOGA}$

Find a list of good view vectors (of materialized views) for a given Query Workload over a time window (T).

Input Vectors:

List of frequent queries (Q) for a specific window of time (T),

Query Frequency vector (f) for a specific window of time (T)

Candidate Big data views for materialization (CV_{BD})

Query evaluation plans using candidate views ($QtVs$)

Cost matrix of candidate Big data views (CV_{DB}) (in terms of stored blocks of Big data)

Stored blocks of view data, if Big data view is materialized (CMV), and

Stored blocks of data required for computing the view, if view is not materialized(CV)

Update factor of Big data for i^{th} View(uf_i)

Cost of updating a view (UMV_i)

View Integrity Factor (I_{mi})

Number of Big data views to materialize (k)

Output

Good view vectors VVk of size k

Procedure:

Initialize:

Population Size (N); Number of Generations (n_g);

Probability of crossover (p_c); Probability of mutation (p_m);

MOGA:

Step 1: Generate Initial Population:

Create random population (PP) of size N of view vectors (VVks of size k

Initialize generations= 1

Perform the following Steps WHILE generations<= n_g

Step 2: Compute Objective functions

For each view vector VVk_t in population, $VVk_t \in PP$

Compute the MCQ_i for every Query $Q_i \in Q$

Compute MCQ_i over every alternative evaluation plan j consisting of v views for every v in jth plan

If v in VVK_t then m_v=1 else m_v = 0

Compute MCQ_i using the equation (1) given below:

$$MCQ_i = Min_j\left[\sum_v\left\{m_v \times CMV_v \times\left(1+\frac{1}{2}\times ufv_v\right)\right\}+\left\{\left(1-m_v\right)\times CV_k \times\left(1+\frac{1}{2}\times uf_v\right)\right\}\right]$$

Compute QEC using equation (2) given below:

$$QEC_{BDV} = \sum_{i=1}^{n}\left(MCQ_i \times f_i\right) \text{// First Objective function}$$

Compute UPC using equation (3) given below:

$$UPC_{BDV} = \sum_i\left(m_i \times UMV_i \div I_{m_i}\right) \text{// Second Objective function}$$

// Compute the objective functions for every view vector

Step 3: Compute Non-dominated count for each view vector VVk

initialize, dominated count, DCV_i= 0 for each $VVk_i \in$ PP

For each $VVk_i \in$ PP and for $VVk_j \in$ PP (j ≠ i)

if VVk_i is dominated by VVk_j

increment DCV_i by 1;

Step 4: Compute Rank

For each $VVk_i \in$ PP

$$RV_i = DCV_i + 1$$ //equation to compute rank (4)

Step 5: Sort rank vector and assign raw fitness

For each $VVk_i \in$ PP

$\mu(RV_i)=\mu(RV_i)$ + 1 ; // count the view vectors with same rank

Sort RV into sRV ; sort the rank view vector

for j = 1 to N

assign raw fitness (rFV) using linear function:

rFV (sRV)= N+1-j;

// All the *VVk* have been assigned raw fitness

Step 6: Compute average fitness (aFV) for each $VVk_i \in$ PP

Compute aFVi for each $VVk_i \in$ PP use equation (5) (Fonseca et al. 1993; Deb 2014):

$$aFV_i = N-\left(\sum_{j=1}^{(RV_i-1)}¼\left(j\right)\right)-\left\{\frac{1}{2}\times\left(¼\left(RV_i\right)-1\right)\right\} \quad (5)$$

Step 7: Compute Normalized distance, shared distance and Niche Count

Assume minimum sharing distance (σVsh) = 0.5; and α = 1;

Repeat for each Rank

For each pair of view vectors VVk_i and VVk_j having the same rank (RVi= RVj)

Compute distance $dVVk_{ij}$ using the equation (6) (Fonseca et al. 1993; Deb 2014):

$$dVVk_{ij} = \sqrt{\left[\left\{\frac{\left(QEC_i - QEC_j\right)}{\left(QEC_{max} - QEC_{min}\right)}\right\}^2 + \left\{\frac{\left(UPC_i - UPC_j\right)}{\left(UPC_{max} - UPC_{min}\right)}\right\}^2\right]} \tag{6}$$

Compute Shared distance using equation (7) (Fonseca et al. 1993; Deb 2014):

$$\begin{aligned} \mathrm{sh}\left(dVVk_{ij}\right) &= 1 - \left(\frac{dVVk_{ij}}{\sigma Vsh}\right)^{\pm} \quad ; \quad \text{if} \quad dVVk_{ij} <= \sigma Vsh \\ &= 0 \qquad\qquad\qquad ; \qquad \text{otherwise} \end{aligned} \tag{7}$$

//Compute shared distance for each *VVk*

For each $VVk_i \in$ PP

Compute Niche count using equation (8) (Fonseca et al. 1993; Deb 2014)

$$NCV_i = \sum_{\mathrm{j}} \left(\mathrm{sh}\left(dVVk_{ij}\right)\right) \ ; \text{ where } j \in \{\text{set of}\left\{VVk_j\right\} \mid RV_i = RV_j\} \tag{8}$$

Step 8: Compute Shared fitness and Scaled shared fitness

For each VVki∈ PP

Compute shared fitness equation (9) (Fonseca et al. 1993; Deb 2014)

$$sFV_i = \frac{aFV_i}{NCV_i} \tag{9}$$

For each $VVk_i \in$ PP

Compute scaled shared fitness using equation (10) (Fonseca et al. 1993; Deb 2014)

$$ssFV_i = \frac{aFV_i \times ¼\left(RV_i\right) \times sFV_i}{\sum_{\mathrm{j}}\left(sFV_j\right)}; \text{ where } j \in \{\text{set of}\left\{VVk_j\right\} \mid RV_i = RV_j\} \tag{10}$$

```
      Step 9: Create mating pool and perform Crossover and
Mutation
Create mating pool by applying proportionate selection
(Goldberg,1989) using scaled shared fitness ssFV
Perform crossover with a probability (p_c) using single point
modified crossover operation (Davis, 1985) ensuring no duplicate
view Vid in a crossed over view vector VVk
Perform mutation with a probability (p_m) using random mutation
operation (Goldberg, 1989) ensuring no duplicate view Vid in a
mutated view vector VVk
Crossover and mutation result in a offspring population (OP) of
view vectorsVVk
     Increment generations
WHILE generations <= n_g repeat Step 2 to Step 9 using OP
     Perform non-dominated count on the final offspring population
(OP)
     Collect the view vectors (VVk) with non-dominated count zero
into OutVVk
Output OutVVk;
END of Algorithm
```

$BDVSA_{MOGA}$ is a bi-objective optimization algorithm to solve the Big data view materialization problem. This algorithm has 9 steps. Step 1 generates the initial random population (*PP*) of size *N* of the view vectors (*VVk*) of size *k*. At Step 2, the objective function values are computed for every view vector of the population using equations (1), (2) and (3). At Step 3, the dominated count of each view vector (DCV_i) is computed.

4.1.1 Raw and Average Fitness Computation

To compute the fitness for each VVk_i in the population ($VVk_i \in PP$), first the rank (RV_i) of each VVk_i is computed using DCV_i at Step 4 using the equation (4) (Fonseca et al. 1993; Deb 2014):

$$RV_i = DCV_i + 1 \tag{4}$$

At Step 5, the $\mu(RV_i)$, which represents the number of view vectors *VVk*s having the same rank, is computed for each rank assignment. In addition, the population (*PP*) is sorted in the ascending order of the rank vector (*RV*), and a linear mapping function is applied to assign the raw fitness (*rFV*) starting from *N* to the highest rank *VVk* down to 1 to the least rank *VVk* (Fonseca et al. 1993; Deb 2014).

At Step 6, the average fitness values (*aFV*) are computed for each $VVk_i \in PP$ using the equation (5) (Fonseca et al. 1993; Deb 2014):

$$aFV_i = N - \left(\sum_{j=1}^{(RV_i-1)} ¼\left(j\right)\right) - \left\{\frac{1}{2} \times \left(¼\left(RV_i\right) - 1\right)\right\} \tag{5}$$

4.1.2 Niche Count Computation

The niche count NCV_i for a view vector VVk_i computes the number of view vectors *VVk* in the population (*PP*), which are inside the distance specified by σV_{sh} (Goldberg et al., 1987). (Goldberg et al., 1987) defined σV_{sh} as the maximum distance between two view vectors so that they can be in the same niche. The purpose of σV_{sh} is to obtain diverse view vectors in the multi-objective space.

The value of σV_{sh} is chosen to be 0.5 in this implementation. To compute the Niche count of $VVk_i \in PP$, the following steps are performed:

At Step 7 of the algorithm, a normalized distance between a pair of view vectors *VVk* with the same rank, is computed using the objective function values. Equation (6) (Fonseca et al. 1993; Deb 2014) is used to compute this distance:

$$dVVk_{ij} = \sqrt{\left[\left\{\frac{\left(QEC_i - QEC_j\right)}{\left(QEC_{max} - QEC_{min}\right)}\right\}^2 + \left\{\frac{\left(UPC_i - UPC_j\right)}{\left(UPC_{max} - UPC_{min}\right)}\right\}^2\right]} \tag{6}$$

In the equation (6), QEC_{max} and QEC_{min} represent the maximum and minimum *QEC* values respectively in the population; likewise, UPC_{max} and UPC_{min} represent the maximum and minimum *UPC* values respectively in the population.

In addition, at Step 7, a sharing function is computed, which identifies the sharing effect between each pair of view vectors with the same rank. Sharing distance is computed using equation (7) (Fonseca et al. 1993; Deb 2014).

$$\begin{aligned} \mathrm{sh}\left(dVVk_{ij}\right) &= 1 - \left(\frac{dVVk_{ij}}{\sigma Vsh}\right)^{\pm} \quad ; \quad \text{if} \quad dVVk_{ij} <= \sigma Vsh \\ &= 0 \qquad\qquad\qquad ; \qquad otherwise \end{aligned} \tag{7}$$

The value σV_{sh} specifies the minimum distance to which the sharing effect is to be considered and α denotes the scaling of the computation. In this implementation, α is considered to be 1.

Finally, at step 7, Niche count (NCV_i) for i^{th} view vector VVk_i is computed by summing the sharing effect of all the view vectors having the same rank using equation (8) (Fonseca et al. 1993; Deb 2014).

$$NCV_i = \sum_{\mathrm{j}} \left(\mathrm{sh}\left(dVVk_{ij}\right)\right) \;\; ; \text{ where } j \in \{\text{set of} \left\{VVk_j\right\} \mid RV_i = RV_j\} \tag{8}$$

The Niche count for the i^{th} view vector is obtained by summing the sharing effect of the i^{th} view vector to all those j^{th} view vectors which have the same rank as the i^{th} view vector.

4.1.3 Computation of Shared Fitness and Scaled Shared Fitness

At Step 8, the shared fitness of i^{th} view vector (sFV_i) is computed from the average fitness and niche counts using equation (9) (Fonseca et al. 1993; Deb 2014):

$$sFV_i = \frac{aFV_i}{NCV_i} \tag{9}$$

However, the shared fitness is scaled to preserve the average fitness of the view vectors. The shared fitness is computed using equation (10) (Fonseca et al. 1993; Deb 2014):

$$ssFV_i = \frac{aFV_i \times ¼\left(RV_i\right) \times \ sFV_i}{\sum_j \left(sFV_j\right)} \ ; \text{ where } j \in \{\text{set of} \left\{VVk_j\right\} \mid \ RV_i = RV_j\} \tag{10}$$

4.1.4 Crossover and Mutation (Step 9)

Using the $ssFV_i$ for all the view vectors, proportionate selection (Goldberg, 1989) is performed to create a mating pool. A single point modified crossover (Davis, 1985), with the probability of crossover (p_c), is then performed on the mating pool. Finally, a random mutation (Goldberg, 1989) with the probability of mutation (p_m) is performed on the view vectors of the mating pool. The population, so produced, is called the offspring population (*OP*), and is considered as the population for the next generation.

Steps 2 to 9 are repeated for the specified number of generations for the view vectors of size *k*. After completing the above steps, a non-dominated count is performed on the final offspring population and the view vectors with non-dominated count zero are produced as the output of the algorithm.

An example illustrating the use of $BDVSA_{MOGA}$ to select Big data views is discussed next.

5. AN EXAMPLE

The $BDVSA_{MOGA}$ was run on a data set of 100 Big data Blocks (assuming each Big data Block = 128 MB). The data set assumes an initial size of 10 Big data blocks for structured data, 40 Big data blocks for semi-structured data and 50 Big data blocks for unstructured data. This data is shown as the Views *V9, V10* and *V11* respectively in Figure 4, which are assumed to be materialized. Figure 4 also shows the structure of the eight views which are at different levels in the view structure of Fig.2. For example, the Big data view *V1(2)*, as shown in Figure 4, means that view *V1* relates to node *2(Country, Age)* in the view structure of Fig. 2. The *S+SS* in the next row of Figure 4, indicates that this view involves only structured and semi-structured data. The view cost data has been computed using the sizes of different types of data, as shown in Fig.2. The view integrity factors are assumed to be lowest for unstructured data. Size increase factor also have chosen on the basis of data characteristics. View update data is approximated using the size of the data required to compute the view and size increase factor. Directed arcs of Fig. 2 have been considered while identifying the query evaluation plans for the five random queries in Figure 5, which also shows the random query frequencies.

The $BDVSA_{MOGA}$ steps are demonstrated below using the view vector of size 4 (k=4) and a single generation of *MOGA*:

Step 1: Generate Initial population

Assuming a population size 10 and view vector size as four (*Top-4* view vector), a random population of size 10 was generated. The initial population is shown in Figure 6.

Step 2: Compute Objective functions

The objective function values are computed using the equations (1), (2) and (3). First the value of MCQ_i is computed using equation (1):

$$MCQ_i = Min_j \left[\sum_k \left\{m_k \times CMV_k \times \left(1 + \frac{1}{2} \times ufv_k\right)\right\} + \left\{\left(1 - m_k\right) \times CV_k \times \left(1 + \frac{1}{2} \times uf_k\right)\right\}\right] \tag{1}$$

Figure 4. Big Data Views Information (S-Structured, SS-Semi-structured, U-Unstructured, D-Data)

V1(2)	*V2(2)*	*V3(3)*	*V4(3)*	*V5(4)*	*V6(4)*	*V7(5)*	*V8(6)*	*V9(1)*	*V10(1)*	*V11(1)*
S+SS	*S+U*	*S+SS*	*S+U*	*S*	*SS+U*	*S+SS*	*S+U*	*SD*	*SSD*	*UD*
Views Cost Data:										
Size of data that will be used for query evaluation if view is not materialized(**CV_i**):										
50	60	50	60	10	90	50	60	10	40	50
Size of materialized view (CMV_i):										
19	24	10	12	3	22	8	4	10	40	50
View Integrity Factor (I_{mi})										
0.9	0.8	0.9	0.8	0.99	0.8	0.9	0.8	0.99	0.9	0.8
Size Increase Factor (ufv_i)										
0.8	0.8	0.7	0.8	0.8	0.7	0.9	0.9	0.8	0.8	1
View Update Data (UMV_i)										
30	48	35	48	8	63	45	54	8	32	50

Figure 5. Query evaluation plans and assumed query frequency

Query Data: Assuming 5 Queries Various plans of Query Evaluation					**Assumed Query Frequency**
Q_1	*V9*	*V5*	-		20
Q_2	*V9, V10*	*V1*	*V3*		30
Q_3	*V9, V11*	*V2*	*V4*	*V8*	20
Q_4	*V10, V11*	*V6*			20
Q_5	*V9, V11*	*V2*	*V4*	*V7*	10

For example, consider the MCQ_i computation of view vector $VV4_1$ *(5, 8, 6, 3)* for Q_2 which has three alternative paths using views *(V9, V10)* or *(V1)* or *(V3)*. Out of all these views *V9* and *V10* are assumed to be materialized, and *V3* is part of the view vector $VV4_1$ and, therefore, is considered for materialization. Thus, MCQ_i for Q_2 will be computed using the materialized view sizes (*CMV*) for views *V9, V10* and *V3* and the size of the data to compute the view (*CV*) for *V1*, which is not materialized. Also, the values of m_9, m_{10}, m_{11} and m_3 is 1, as the related views are materialized, and m_1=0, as the view *V1* is not materialized.

MCQ_2 = Minimum of[{(1×CMV_9× (1+ 0.5×ufv_9) + (1×CMV_{10} × (1+ 0.5×ufv_{10}) };
{(1-0)×CV_1× (1+ 0.5×ufv_1) } ;
{(1×CMV_3× (1+ 0.5×ufv_3) }]

MCQ_2 = Minimum of[{(1×10× (1+ 0.5×0.8) + (1×40 × (1+ 0.5×0.8) };
{(1)×50× (1+ 0.5×0.8) } ;
{(1×10× (1+ 0.5×0.7) }]

MCQ_2 = Minimum of [70; 70; 13.5] = **13.5**

The values of MCQ_i for $VV4_1$ *(5, 8, 6, 3)* are shown in the Figure 7:

Figure 6. Initial Population of Top-4 View Vectors

VV4	Initial Population			
$VV4_1$	5	8	6	3
$VV4_2$	5	7	3	2
$VV4_3$	5	2	7	8
$VV4_4$	7	4	1	3
$VV4_5$	5	8	4	1
$VV4_6$	2	7	4	3
$VV4_7$	8	7	3	1
$VV4_8$	2	6	3	8
$VV4_9$	4	7	5	6
$VV4_{10}$	6	3	5	7

Next, the value of QEC_{BDV} is computed using equation (2):

$$QEC_{BDV} = \sum_{i=1}^{n} \left(MCQ_i \times f_i \right) \tag{2}$$

// the values of f_i are given in Figure 4)

$(QEC_{BDV})_{VV41}$ = (4.2× 20 + 13.5 × 30 + 5.8 × 20 + 29.7 ×20 + 72.5 × 10)

= **1924**

The second objective function value will be computed using equation (3):

$$UPC_{BDV} = \sum_i \left(m_i \times UMV_i \div I_{m_i} \right) \tag{3}$$

For $VV4_1$ *(5, 8, 6, 3)*, UPC_{BDV} will be computed using the values of UMV_i and I_{mi} for the views of $VV4_1$ viz. *V5, V8, V6, V3* and already materialized views *V9, V10, V11.*

Figure 7. MCQ_i's for $VV4_1$ (5, 8, 6, 3)

MCQ_1	MCQ_2	MCQ_3	MCQ_4	MCQ_5
4.2	13.5	5.8	29.7	72.5

$$(UPC_{BDV})_{VV41} = (1\times UMV_5/I_{m5}+1\times UMV_8/I_{m8}+1\times UMV_6/I_{m6}+1\times UMV_3/I_{m3}+ 1\times UMV_9/I_{m9}+1\times UMV_{10}/I_{m10}+1\times UMV_{11}/I_{m11})$$

$$(UPC_{BDV})_{VV41} = (8/0.99+54/0.9+63/0.8+35/0.9+8/0.99+32/0.9+50/0.8) = \mathbf{299.36}$$

Similarly, the objective function values for each of view vectors *VV4* in the population will be computed. These values are shown in Figure 8.

Figure 8. QEC, UPC, DCV, RV of Top-4 view vectors

VVk_4	*QEC*	*UPC*	*DCV*	*RV*
$VV4_1$	1924	299.36	0	1
$VV4_2$	3707	263.11	0	1
$VV4_3$	4771	291.72	4	5
$VV4_4$	3567	288.36	1	2
$VV4_5$	3596	275.05	0	1
$VV4_6$	3567	315.03	5	6
$VV4_7$	3347	295.86	1	2
$VV4_8$	1731	351.28	0	1
$VV4_9$	3155	302.97	2	3
$VV4_{10}$	2879	281.86	0	1
	Step 2	**Step 2**	**Step 3**	**Step 4**

Step 3: Compute Non-dominated count for each *VVk*

In this step, the i^{th} view vector is compared to all other view vectors to determine the count of view vectors, which dominate this i^{th} view vector. For example, view vector $VV4_4$ is dominated by view vector $VV4_{10}$, which has better *QEC* and *UPC* values ($QEC_{VV44} > QEC_{VV410}$ and $UPC_{VV44} > UPC_{VV410}$). It may be noted that no other view vector in the population dominates $VV4_4$. Thus, the count of view vectors dominating view vector $VV4_4$, DCV_4 is one. Similarly, non-dominated count *DCV* is computed for all view vectors. Figure 8 shows the computed values of *DCV*.

Step 4: Compute Rank

In this step, the rank of each view vector is computed using equation (14) (Fonseca et al. 1993; Deb 2014):

$$RV_i = DCV_i + 1 \tag{4}$$

For example, rank of $VV4_1$ is computed as:

$$RV_1 = DCV_1 + 1 = 0+1 = 1$$

Similarly, the rank of all the view vectors *VV4* is computed. These ranks are shown in Figure 8.

Step 5: Sort rank vector and assign raw fitness

The algorithm computes the count of same ranked *VVk*. These values for the example are: μ(1)=5; μ(2)=2; μ(3)=1; μ(4)=0; μ(5)=1; μ(6)=1; and μ(7)=μ(8)=μ(9)=μ(10)=0.

Next, a sorted sequence of view vectors (*sRV*) is created using the rank (*RV*):

```
{1, 2, 5, 8, 10}, {4, 7}, {9}, {}, {3}, {6}
```

Accordingly, the view vectors are assigned the raw fitness value (*rFV)* using the linear function, as shown in Figure 9. It may be noted that view vector $VV4_1$ is assigned a raw fitness 10, $VV4_2$ is assigned a raw fitness 9 and $VV4_5$ is assigned a raw fitness 8, and so on.

Step 6: Compute average fitness (*aFV)* for each view vector *VVk*

Compute the value of average fitness using equation (5) (Fonseca et al. 1993; Deb 2014):

$$aFV_i = N - \left(\sum_{j=1}^{(RV_i-1)} ¼\left(j\right)\right) - \left\{\frac{1}{2} \times \left(¼\left(RV_i\right) - 1\right)\right\}$$

$$aFV_1 = N - \left(\sum_{j=1}^{(RV_1-1)} ¼\left(j\right)\right) - \left\{\frac{1}{2} \times \left(¼\left(RV_1\right) - 1\right)\right\}$$

$$aFV_1 = N - \left(\sum_{j=1}^{(1-1)} ¼\left(j\right)\right) - \left\{\frac{1}{2} \times \left(¼\left(RV_1\right) - 1\right)\right\}$$

$$aFV_1 = 10 - \left(0\right) - \left\{\frac{1}{2} \times \left(5 - 1\right)\right\} = 8 \tag{5}$$

Similarly, the average fitness of all the view vectors is computed and is shown in Figure 9.

Step 7: Compute Normalized distance, shared distance and Niche Count

In this step the normal distance for the same ranked *VVk*s are computed using the values of the objective function using equation (6):

$$dVVk_{12} = \sqrt{\left[\left\{\frac{\left(QEC_1 - QEC_2\right)}{\left(QEC_{max} - QEC_{min}\right)}\right\}^2 + \left\{\frac{\left(UPC_1 - UPC_2\right)}{\left(UPC_{max} - UPC_{min}\right)}\right\}^2\right]}$$

$$dVVk_{12} = \sqrt{\left[\left\{\frac{\left(1924 - 3707\right)}{\left(4771 - 1731\right)}\right\}^2 + \left\{\frac{\left(299.36 - 263.11\right)}{\left(351.28 - 263.11\right)}\right\}^2\right]} = 0.71626$$

Similarly, the normalized distance for other pairs having rank 1:

Figure 9. RV, rFV, aFV, NCV, sFV, ssFV of Top-4 view vectors

VVk_4	RV	rFV	aFV	NCV	sFV	ssFV
$VV4_1$	1	10	8	1.2568	6.3653	9.8786
$VV4_2$	1	9	8	2.0283	3.9442	6.1212
$VV4_3$	5	2	2	1	2	2
$VV4_4$	2	5	4.5	1.7766	2.5329	4.5
$VV4_5$	1	8	8	2.2231	3.5986	5.5849
$VV4_6$	6	1	1	1	1	1
$VV4_7$	2	4	4.5	1.7766	2.5329	4.5
$VV4_8$	1	7	8	1	8	12.416
$VV4_9$	3	3	3	1	3	3
$VV4_{10}$	1	6	8	2.0694	3.8659	5.9997
		Step 5	**Step 6**	**Step 7**	**Step 8**	**Step 8**

$dVVk_{15}$= 0.61522, $dVVk_{18}$=0.59227, $dVVk_{110}$= 0.37159
$dVVk_{25}$= 0.14031, $dVVk_{28}$=1.1927, $dVVk_{210}$= 0.34556
$dVVk_{58}$= 1.0601, $dVVk_{510}$= 0.24816
$dVVk_{11}$=$dVVk_{22}$ = $dVVk_{55}$= $dVVk_{88}$= $dVVk_{1010}$=0

Assuming minimum sharing distance (σV_{sh}) = 0.5 and $\alpha = 1$; and using equation (7), the following values of shared distance is computed:

$$\begin{aligned} \mathrm{sh}\left(dVVk_{25}\right) &= 1 - \left(\frac{dVVk_{25}}{\sigma Vsh}\right)^{\pm} \quad ; \quad \text{if } dVVk_{ij} <= \sigma Vsh \\ &= 0 \qquad\qquad\qquad ; \qquad \text{otherwise} \end{aligned} \tag{7}$$

$$\mathrm{sh}\left(dVVk_{25}\right) = 1 - \left(\frac{0.14031}{0.5}\right)^{1} = 0.71939$$

All the shared distances will be computed accordingly. Computation of shared distance for $VV4_1$ is given below:

sh($dVVk_{11}$) = 1; sh($dVVk_{12}$)=0 sh($dVVk_{15}$)= 0, sh($dVVk_{18}$)=0,and sh($dVVk_{110}$)=0.25681

Next Niche count is computed using equation (8):

$$NCV_1 = \sum_{\mathrm{j}} (\mathrm{sh}\left(dVVk_{1j}\right)) \;\; ; \text{ where } j \in \left\{VVk_1, VVk_2, VVk_5, VVk_8, VVk_{10}\right\}$$

$$NCV_1 = \text{sh}\left(dVVk_{11}\right) + \text{sh}\left(dVVk_{12}\right) + \text{sh}\left(dVVk_{15}\right) + \text{sh}\left(dVVk_{18}\right) + \text{sh}\left(dVVk_{110}\right)$$

$$NCV_1 = 1 + 0 + 0 + 0 + 0.25681 = 1.25681$$

Similarly, the Niche count for all the view vectors are computed and are shown in Figure 9.

Step 8: Compute Shared fitness and Scaled shared fitness

The shared fitness is computed using equation (9) as follows:

$$sFV_1 = \frac{aFV_1}{NCV_1}$$

$$sFV_1 = \frac{8}{1.2568} = 6.3653$$

Similarly, shared fitness for all the view vectors is computed and are shown in Figure 9.
The shared fitness values are scaled to preserve the original ranks by using equation (10).

$$ssFV_1 = \frac{aFV_1 \times ¼\left(RV_1\right) \times sFV_1}{\sum_j \left(sFV_j\right))}; \text{where } j \in \left\{VVk_1, VVk_2, VVk_5, VVk_8, VVk_{10}\right\}$$

$$ssFV_1 = \frac{aFV_1 \times ¼\left(RV_1\right) \times sFV_1}{\left(sFV_1 + sFV_2 + sFV_5 + sFV_8 + sFV_{10}\right)}$$

$$ssFV_1 = \frac{8 \times 5 \times 6.3653}{\left(6.3653 + 3.9442 + 3.5986 + 8 + 3.8659\right)} = 9.8786 = 9.8786$$

Similarly, all the shared fitness values are scaled and are shown in Figure 9.

Step 9: Create mating pool and perform Crossover and Mutation

The mating pool is created using proportionate selection (Goldberg, 1989) on the scaled shared fitness values (*ssFV*) in Figure 9. Figure 10 shows the mating pool. The crossover is performed with the probability of crossover (p_c). The value of p_c is taken as 0.8. The crossover is performed on the mating pool using a modified single point crossover (Davis, 1985). The modified crossover ensures that no view is duplicated in a view vector. Next, a random mutation, as suggested in (Goldberg 1989; Kazarlis et al. 1996), is performed with mutation probability (p_m) as 0.1. Figure 10 illustrates the result of crossover and mutation.

Crossover and mutation result in an offspring population (*OP*) of *VVk,* which is shown in Figure 10. The steps 2 to 9 are repeated for the predefined number of generations with *OP* as the population for the next generation. A non-dominated count is performed on the *OP* of the final generation to produce the non-dominated view vectors as output view vectors of the algorithm. Next, the experimental results are discussed.

Figure 10. Crossover and Mutation on Top-4 view vectors

Mating Pool				Point for Crossover	Mating Pool after Crossover				Point for Mutation	Offspring Population			
8	7	3	1	3	8	7	1	3		8	7	1	3
7	4	1	3		7	4	3	1		7	4	3	1
2	7	4	3	3	2	7	4	3		2	7	4	3
7	4	1	3		7	4	1	3		7	4	1	3
8	7	3	1	2	8	7	5	1		8	7	5	1
6	3	5	7		6	3	4	1		6	3	4	1
5	8	6	3	2	5	8	6	7		5	8	6	7
6	3	5	7		6	3	7	4	1	8	3	7	4
5	8	6	3	1	5	2	7	8		5	2	7	8
5	2	7	8		5	8	6	3		5	8	6	3

6. EXPERIMENTAL RESULTS

The $BDVSA_{MOGA}$ is implemented using *GNU* Octave 4.4.1 on an Intel dual core *I5*, 2.5 *GHz*, 64 bit processor, with 6 *GB RAM*. The data set of $BDVSA_{MOGA}$ has a set of 14 queries with 4 query attributes and 16 views, same as was in (Kumar & Vijay Kumar, 2021b). The query statistics has been kept identical to (Kumar & Vijay Kumar, 2021b) so as to validate the results, as there are no existing benchmarks in case of Big data view materialization. The algorithm was run for a population (N) =100; Number of generations (n_g) = 25; Probability of crossover (p_c) = 0.8; Probability of mutation (p_m) = 0.1; and for different sizes of the view vectors.

Figure 11 shows the non-dominated view vectors that are generated for different sized view vectors *VV3* (with k=3) to *VV10* (with k = 10). These view vectors show different characteristics of the non-dominated view vectors, e.g., for a materialized view vector of size 3, the view update cost is low but they have high query evaluation cost, whereas for a materialized view vector of size 10, the view update cost is high while having low query evaluation cost. In addition, there is an overlapping of view vectors of different sizes, indicating that an overall dominated count may be used to obtain the best view vectors, which may be of different sizes. The overlap between the view vectors of different sizes is also evident in Figure 5, which is a three-dimensional representation of Figure 4.

Figure 13 lists the number of non-dominated view vectors that were obtained using the $BDVSA_{MOGA}$. These non-dominated solutions have a range of *QEC* and *UPC* values, which are also shown in the Figure 13. It was observed in the data that there were several non-dominated view vectors having the same value of both *QEC* and *UPC*, though they had a different set of materialized views (refer to Figure 14, which shows a pair of such view vectors). This is due to a specific data set used for the experimental results, which has several smaller views of similar size and usage frequency. However, for a large Big data set, such possibilities would be fewer.

Thus, a large number non-dominated view vectors are obtained for each value of (k). It can be observed from Figure 11 and Figure 12 that some of these view vectors of different sizes may have overlapping *QEC* and *UPC* values. Therefore, a set of non-dominated view vectors were extracted from these view vectors of different sizes. *QEC* and *UPC* values of these non-dominated view vectors are shown in Figure 15. Figure 16 shows the *QEC* and *UPC* values of these non-dominated view vectors for different values of k. It can be observed from Figure 15 and Figure 16 that the non-dominated view vectors are of sizes k=3 to k=9. Thus, the view vectors of size 10, for the given set of data, are being dominated by the view vectors of other sizes.

Figure 11. Query Evaluation Cost and Update Processing Cost for different number of Materialized views

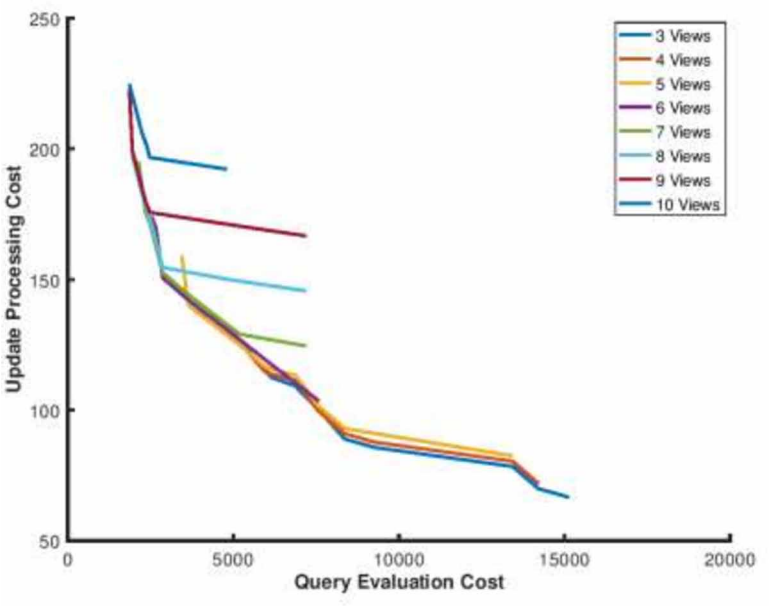
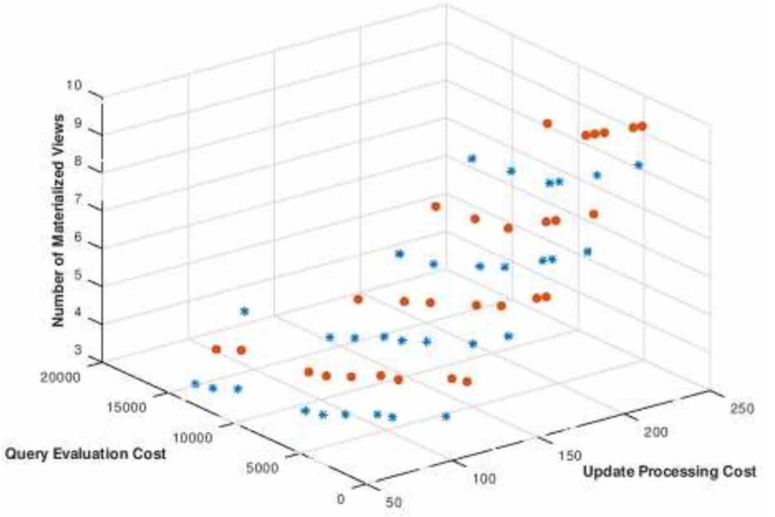


Figure 12. Query Evaluating Cost Vs. Update Processing Cost Vs. Number of Materialized Views



Some of the typical view vectors obtained from $BDVSA_{MOGA}$ are compared with the view vectors obtained from $BDVSA_{VEGA}$ (Kumar & Vijay Kumar, 2021b) after 25 generations. Figure 17 shows a comparison of the vectors obtained by the two algorithms for view vectors of size 7.

Figure 17 shows only few such view vectors produced by $BDVSA_{MOGA}$ that dominate view vectors produced by $BDVSA_{VEGA}$. The objective of Big data view materialization is to minimize the *QEC* and the *UPC*. The $BDVSA_{MOGA}$ has identified view vectors of size 7, which will result in lower query evaluation costs and lower update processing costs in comparison to the view vectors obtained by

Figure 13. The Non-dominated Top-k view vectors obtained using $BDVSA_{MOGA}$

Number of Materialized views (*k*)	Minimum *QEC*	*UPC* for Minimum *QEC*	Minimum *UPC*	*QEC* for minimum *UPC*	Number of non-dominated *VVk*
3	4743	133	67	15127	15
4	3833	138	72	14217	11
5	3308	164	82	13437	15
6	2663	170	104	7580	13
7	2138	195	125	7190	12
8	1951	197	150	4798	14
9	1859	223	167	7190	13
10	1859	225	192	4798	9

Figure 14. Top-3 view vectors having same QEC and UPC

QEC	*UPC*	*k*	View vector of size 3		
6878	109	3	1	9	10
6878	109	3	3	9	10

$BDVSA_{VEGA}$. In addition, it may be noted that $BDVSA_{MOGA}$ has produced more diverse view vectors than $BDVSA_{VEGA}$. Thus, $BDVSA_{MOGA}$ produce better quality view vectors in comparison of $BDVSA_{VEGA}$.

7. CONCLUSION

Big data view materialization is a complex problem due to very large semi-structured and unstructured data, which is generated at a fast rate and is low in integrity. An extended semantic model can be used to represent the structure of Big data, which can help in identifying query attributes and their interrelations for materialization. The view structure is a complex directed graph and can be used to generate a list of candidate views for materialization. The view structure also guides the process of creating query evaluation plans for a given query using Big data views. The Big data view materialization being a bi-objective problem having objectives minimization of the query evaluation cost and minimization of the view update costs, with constraints on the cumulative size of views, is addressed using *MOGA*. Accordingly, an algorithm $BDVSA_{MOGA}$ is proposed to select Big data views. Experimental based comparison of $BDVSA_{MOGA}$ with $BDVSA_{VEGA}$ shows that the former produces diverse non-dominated *Top-k* view vectors that dominate *Top-k* view vectors produced by *the* latter. Thus, $BDVSA_{MOGA}$ is capable of selecting better quality views, which when materialized would result in efficient query processing.

Figure 15. Non-dominated Top-k view vectors

QEC	*UPC*	No of Views Materialized (*k*)
1859	223	9
1951	197	8
2138	195	7
2341	176	7
2476	172	7
2663	170	6
2866	151	6
3646	140	5
3833	138	4
4743	133	3
5188	125	5
5375	123	4
5968	115	4
6225	112	3
6878	109	3
7580	101	5
7767	97	3
8360	89	3
9270	86	3
13437	80	4
14217	70	3
15127	67	3

Figure 16. Non-dominated Top-k view vectors

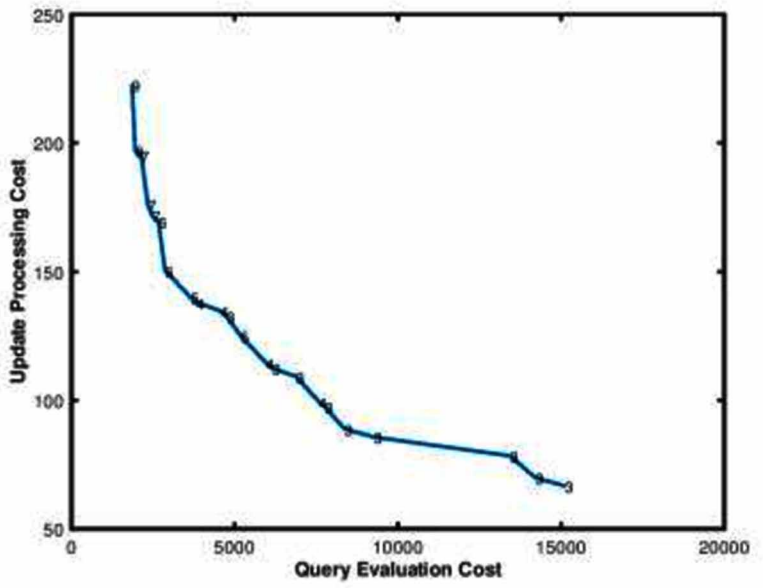


Figure 17. $BDVSA_{MOGA}$ Vs. $BDVSA_{VEGA}$: QEC and UPC of Top-7 view vectors

View Vectors of $BDVSA_{MOGA}$			View Vectors of $BDVSA_{VEGA}$	
QEC	*UPC*		*QEC*	*UPC*
2138	195	Dominates view vectors of $BDVSA_{VEGA}$	2826	210
2341	176	Dominates view vectors of $BDVSA_{VEGA}$	2956	215
2476	172	Dominates view vectors of $BDVSA_{VEGA}$	3159	196

REFERENCES

Abiteboul, S. (1999, December). On Views and XML. *SIGMOD Record*, *28*(4), 30–38. doi:10.1145/344816.344853

Abiteboul, S., Goldman, R., McHugh, J., Vassalos, V., & Zhuge, Y. (1997). *Views for Semi-structured Data.* Technical Report. Stanford InfoLab, Workshop on Management of Semi-structured Data, Tucson, AZ.

Agrawal, S., Chaudhari, S., & Narasayya, V. (2000). Automated Selection of Materialized Views and Indexes in SQL databases. *26th International Conference on Very Large Data Bases (VLDB 2000),* 486-505.

Arun, B., & Vijay Kumar, T. V. (2015a). Materialized View Selection using Marriage in Honey Bees Optimization. *International Journal of Natural Computing Research*, *5*(3), 1–25. doi:10.4018/IJNCR.2015070101

Arun, B., & Vijay Kumar, T. V. (2015b). Materialized View Selection using Improvement Based Bee Colony Optimization. *International Journal of Software Science and Computational Intelligence*, *7*(4), 35–61. doi:10.4018/IJSSCI.2015100103

Arun, B., & Vijay Kumar, T. V. (2017a). Materialized View Selection using Artificial Bee Colony Optimization. *International Journal of Intelligent Information Technologies*, *13*(1), 26–49. doi:10.4018/IJIIT.2017010102

Arun, B., & Vijay Kumar, T. V. (2017b). Materialized View Selection using Bumble Bee Mating Optimization. *International Journal of Decision Support System Technology*, *9*(3), 1–27. doi:10.4018/IJDSST.2017070101

Chirkova, R., Halevy, A. Y., & Suciu, D. (2001). A Formal Perspective on the View Selection Problem. *Proceedings of the 27th VLDB conference.*

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. *Proceedings of the international joint conference on artificial intelligence*, 162–164.

Dean, J., & Ghemawat, S. (2012, January). MapReduce: A Flexible data processing tool. *Communications of the ACM*, *53*(1), 72–77. doi:10.1145/1629175.1629198

Deb, K. (2014). Multi-objective Optimization. In E. Burke & G. Kendall (Eds.), *Search Methodologies*. Springer. doi:10.1007/978-1-4614-6940-7_15

Dezyre. (2015). *Hadoop Ecosystem Components and Its Architecture*. https://www.dezyre.com/article/hadoop-ecosystem-components-and-its architecture/114

DocumentationH. (2008). http://hadoop.apache.org/docs/r0.17.0/mapred_tutorial.html

El-Sayed, M., Rundensteiner, E. A., & Mani, M. (2006). Incremental Maintenance of Materialized XQuery Views. *Proceedings of 22nd International Conference on Data Engineering (ICDE'06).* doi:10.1109/ICDE.2006.80

Fonseca, C. M., & Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. *Proceedings of the 5th international conference on genetic algorithms*, 416–423.

Gandomi, A., & Haider, M. (2015). Beyondthe hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, *35*(2), 137–144. doi:10.1016/j.ijinfomgt.2014.10.007

George, F. (2017). *10 Vs of Big data*. https://tdwi.org/articles/2017/02/08/10-vs-of-big-data.aspx

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning* (Vol. 1). Addison Wesley. doi:10.1007/s10589-009-9261-6

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proceedings of the 2nd international conference on genetic algorithms on genetic algorithms and their application*, 41–49.

Goswami, R., Bhattacharyya, D. K., & Dutta, M. (2017, December). Materialized view selection using evolutionary algorithm for speeding up big data query processing. *Journal of Intelligent Information Systems*, *49*(3), 407–433. doi:10.1007/s10844-017-0455-6

Gupta, A., & Mumick, I. S. (1995). Maintenance of Materialized Views: Problems, Techniques, and Applications. Data Eng. Bulletin, 18(2).

Gupta, H. (1996). Selection of views to materialize in a data warehouse. In F. Afrati & P. Kolaitis (Eds.), Lecture Notes in Computer Science: Vol. 1186. *Database Theory — ICDT '97. ICDT 1997*. Springer.

Gupta, R., Gupta, H., & Mohania, M. (2012). Cloud Computing and Big Data Analytics: What is new from Database Perspective? In *Proceedings of Big Data Analytics-First International Conference*. Springer. doi:10.1007/978-3-642-35542-4_5

Hadoop. (2012). http://hadoop.apache.org/

Harinarayan, V., Rajaraman, A., & Ullman, J. D. (1996). Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD '96)*. ACM. doi:10.1145/233269.233333

Jacobs, A. (2009, August). The Pathologies of Big Data. *Communications of the ACM*, *52*(8), 36–44. doi:10.1145/1536616.1536632

Kazarlis, S. A., Bakirtzis, A. G., & Petridis, V. (1996). A genetic algorithm solution to the unit commentment problem. *IEEE Transactions on Power Systems*, *11*(1), 83–92. doi:10.1109/59.485989

Khan, M. A., Uddin, M. F., & Gupta, N. (2014). Seven V's of Big Data understanding Big Data to extract value. *Proceedings of the 2014 Zone 1 Conference of the American Society for Engineering Education*, 1-5. doi:10.1109/ASEEZone1.2014.6820689

Kumar, A., & Vijay Kumar, T. V. (2015). Big data and analytics: Issues, challenges, and opportunities. *International Journal of Data Science*, *1*(2), 118-138. doi:10.1504/IJDS.2015.072412

Kumar, A., & Vijay Kumar, T.V. (2021a). View Materialization over Big Data. *International Journal of Data Analytics, 2(1), 61-85.*.

Kumar, A., & Vijay Kumar, T.V. (2021b). *A Multi Objective Approach to Big Data View Materialization*, International Journal of Knowledge and Systems Science, 12(2), 17-37..

Kumar, S., & Vijay Kumar, T.V. (2018). A Novel Quantum Inspired Evolutionary View Selection Algorithm. *Journal Sadhana, 43*(10).

Kuno, H. A., & Rundensteiner, E. A. (1995). Materialized Object-Oriented Views in MultiView. *ACM Research Issues Data Eng. Workshop*, 78-85.

Luo, J., Wu, M., Gopukumar, D., & Zhao, Y. (2016). Big Data Application in Biomedical Research and Health Care: A Literature Review. *Biomedical Informatics Insights*, *8*. Advance online publication. doi:10.4137/BII.S31559 PMID:26843812

Mami, I., & Bellahsene, Z. (2012). A Survey of View Selection Methods. SIGMOD Record, 41(1).

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Hung Byers, A. (2011). *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute.

Mistry, H., Roy, P., Sudarshan, S., & Ramamritham, K. (2001). Materialized view selection and maintenance using multi-query optimization. *Proceedings of the ACM (SIGMOD) Conference on the Management of Data*, 307-318. doi:10.1145/375663.375703

Prakash, J., & Vijay Kumar, T. V. (2019a). A Multi-objective Approach for Materialized View Selection. *International Journal of Operations Research and Information Systems*, *10*(2), 1–19. doi:10.4018/IJORIS.2019040101

Prakash, J., & Vijay Kumar, T. V. (2019b). Multi-Objective Materialized View Selection using Improved Strength Pareto Evolutionary Algorithm. *International Journal of Artificial Intelligence and Machine Learning*, *9*(2), 1–21. doi:10.4018/IJAIML.2019070101

Prakash, J., & Vijay Kumar, T. V. (2020a). Multi-Objective Materialized View Selection using MOGA. *International Journal of Systems Assurance Engineering and Management*, *11*(2), 220–231. doi:10.1007/s13198-020-00947-2

Prakash, J., & Vijay Kumar, T. V. (2020b). Multi-Objective Materialized View Selection using NSGA-II. *International Journal of Systems Assurance Engineering and Management*, *11*(5), 972–984. doi:10.1007/s13198-020-01030-6

Ross, K., Srivastava, D., & Sudarshan, S. (1996). Materialized view maintenance and integrity constraint checking: Trading space for time. *SIGMOD Intl. Conf. on Management of Data*. doi:10.1145/233269.233361

Roussopoulos, N. (1998). Materialized Views and Data Warehouses. *SIGMOD Record*, *27*(1), 21–26. doi:10.1145/273244.273253

Shneiderman, B. (2020). *Data Visualization's Breakthrough Moment in the COVID-19 Crisis*. Nightingale. https://medium.com/nightingale/data-visualizations-breakthrough-moment-in-the-covid-19-crisis-ce46627c7db5

Tang, N., Xu Yu, J., & Tang, H. (2009). Materialized View Selection in XML Databases. Database Systems for Advanced Applications, 5463.

Vijay Kumar, T. V., & Haider, M. (2010). Materialized Views Selection for Answering Queries. *ICDEM 2010: Data Engineering and Management*, 44-51.

Vijay Kumar, T. V., & Arun, B. (2016). Materialized View Selection using BCO. *International Journal of Business Information Systems*, *22*(3), 280–301.

Vijay Kumar, T. V., & Arun, B. (2017). Materialized View Selection using HBMO. *International Journal of Systems Assurance Engineering and Management, 8*(1), 379-392.

Vijay Kumar, T. V., & Kumar, S. (2014). Materialized View Selection using Differential Evolution. *International Journal of Innovative Computing and Applications*, *6*(2), 102–113. doi:10.1504/IJICA.2014.066499

Vijay Kumar, T. V., & Kumar, S. (2015). Materialized View Selection using Randomized Algorithms. *International Journal of Business Information Systems*, *19*(2), 224–240. doi:10.1504/IJBIS.2015.069432

Yafooz, Abidin, Zaleha, Omar, & Idrus. (2013). Managing unstructured data in relational databases. *Proceedings - 2013 IEEE Conference on Systems, Process and Control, ICSPC 2013*, 198-203.

Zhou, C., Su, F., & Pei, T. (2020). COVID-19: Challenges to GIS with Big Data. *Geography and Sustainability, 1*(1), 77-87. https://www.sciencedirect.com/science/article/pii/S2666683920300092

Zikopoulos, P. C. E. (2011). Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data (1st ed.). McGraw-Hill Osborne Media.

Akshay Kumar completed his PhD at Jawharlal Nehru University, New Delhi. He did Master of Technology (M.Tech.) in Computer Science from IIT Delhi in 1988. He is employed as a faculty member at School of Computer and Information Science, IGNOU, New Delhi.

T. V. Vijay Kumar has completed his PhD in the area of databases from Jawaharlal Nehru University, New Delhi, India, after completing his MPhil and MSc in Operational Research, and BSc (Hons) in mathematics, from the University of Delhi, Delhi, India. His research interests are databases, data warehousing, data mining, machine learning, nature inspired algorithms, disaster management, Big Data, and analytics.