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
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 $Q_A$'s.

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.
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 QA Country 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 QAs 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 QA City. This query can be answered using a view that can be generated using semi-structured data of node 6 (QA 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 (QA 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 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 $2^{|QA|}$ 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
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) and the frequency of that query (f_i). MCQ_i of j^{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_{ik}), 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_{ik}), 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:

\[
MCQ_i = \text{Min} \left[ \sum_k \left( m_k \times CMV_k \times \left[ 1 + \frac{1}{2} \times ufv_{ik} \right] + \left( 1 - m_k \right) \times CV_k \times \left[ 1 + \frac{1}{2} \times uf_{ik} \right] \right) \right] ^{1/2} \tag{1}
\]

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:

\[
QEC_{BDV} = \sum_{i=1}^{n} (MCQ_i \times f_i) \tag{2}
\]

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:

\[
UPC_{BDV} = \sum_i \left( m_i \times UMV_i \div I_m \right) \tag{3}
\]

(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 $CV_{DB}$, the cost of using a candidate view in $CV_{BD}$, if it is materialized ($CMV$); or the cost, if the candidate view is not materialized ($CV$), the update factors ($uf$), the cost of updating a view ($UMV$), the integrity factor ($I_m$) for each view and the number of Big data views to materialize ($k$).
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 ($V_i$). The view vector ($V_{Vk}$) represents a set of the materialized views of size $k$. The objective of BDVSAMOGA 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 ($V_{V4}$). The view vector $V_{V4}$ consists of four distinct Big data views ($V_{id}$).

BDVSAMOGA
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 ($C_{BD}$)
- Query evaluation plans using candidate views ($Q_{Vs}$)
- Cost matrix of candidate Big data views ($C_{DB}$) (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 ($Im_i$)
- Number of Big data views to materialize ($k$)

Output
Good view vectors $VV_k$ 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 ($VV_k$) of size $k$
- Initialize generations = 1
Perform the following Steps WHILE generations <= $n_g$
Step 2: Compute Objective functions
- For each view vector $VV_k$ in population, $VV_k$ $\in$ 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 $j$th plan:

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

Compute $MCQ_i$ using the equation (1) given below:

$$MCQ_i = \min_j \left[ \sum_v \left[ m_v \times CMV_v \times \left( 1 + \frac{1}{2} \times u_f v \right) \right] + \left[ (1 - m_v) \times CV_i \times \left( 1 + \frac{1}{2} \times u_f v \right) \right] \right]$$

Compute QEC using equation (2) given below:

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

Compute UPC using equation (3) given below:

$$UPC_{BDV} = \sum_{i} \left( m_i \times UMV_i \div I_m \right) \quad \text{// Second Objective function}$$

// Compute the objective functions for every view vector

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

Initialize, dominated count, $DCV_i = 0$ for each $VV_k_i \in PP$

For each $VV_k_i \in PP$ and for $VV_k_j \in PP$ ($j \neq i$) if $VV_k_j$ is dominated by $VV_k_j$

Increment $DCV_i$ by 1;

Step 4: Compute Rank

For each $VV_k_i \in PP$

$$RV_i = DCV_i + 1 \quad \text{// equation to compute rank}$$

Step 5: Sort rank vector and assign raw fitness

For each $VV_k \in PP$

$$\mu(RV_i) = \mu(RV_j) + 1 \quad ; \quad \text{// count the view vectors with same rank}$$

Sort RV into sRV

// All the VVk have been assigned raw fitness

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

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

$$aFV_i = N - \left( \sum_{j=1}^{RV_i-1} (RV_j) \right) - \left( \frac{1}{2} \times \left( \sqrt[RV_i-1]{RV_i} - 1 \right) \right)$$
Step 7: Compute Normalized distance, shared distance and Niche Count

Assume minimum sharing distance ($\sigma_{Vsh}$) = 0.5; and $\alpha = 1$;

Repeat for each Rank

For each pair of view vectors $VV_{ki}$ and $VV_{kj}$ having the same rank ($RV_i = RV_j$)

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

$$d_{VV_{kij}} = \left( \frac{QEC_i - QEC_j}{QEC_{max} - QEC_{min}} \right)^2 + \left( \frac{UPC_i - UPC_j}{UPC_{max} - UPC_{min}} \right)^2$$

(6)

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

$$sh(d_{VV_{kij}}) = 1 - \left( \frac{d_{VV_{kij}}}{\sigma_{Vsh}} \right)^\pm; \text{ if } d_{VV_{kij}} \leq \sigma_{Vsh}$$

$$= 0; \text{ otherwise}$$

(7)

//Compute shared distance for each $VV_k$

For each $VV_{ki} \in PP$

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

$$NC_{V_i} = \sum_j (sh(d_{VV_{kij}})); \text{ where } j \in \{\text{set of } \{VV_{kj}\} \mid RV_i = RV_j\}$$

(8)

Step 8: Compute Shared fitness and Scaled shared fitness

For each $VV_{ki} \in PP$

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

$$sF_{V_i} = \frac{aF_{V_i}}{NC_{V_i}}$$

(9)

For each $VV_{kj} \in PP$

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

$$ssF_{V_i} = \frac{aF_{V_i} \times \frac{1}{4}(RV_i) \times sF_{V_i}}{\sum_j (sF_{V_j})}; \text{ where } j \in \{\text{set of } \{VV_{kj}\} \mid RV_i = RV_j\}$$

(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 \( V_{id} \) in a crossed over view vector \( V_{Vk} \)
Perform mutation with a probability \( p_m \) using random mutation operation (Goldberg, 1989) ensuring no duplicate view \( V_{id} \) in a mutated view vector \( V_{Vk} \)

Crossover and mutation result in a offspring population (OP) of view vectors \( V_{Vk} \)

Increment generations

WHILE generations \( \leq n_g \) repeat Step 2 to Step 9 using OP

Perform non-dominated count on the final offspring population (OP)

Collect the view vectors \( V_{Vk} \) with non-dominated count zero into \( \text{OutV}_{Vk} \)

Output \( \text{OutV}_{Vk} \);

END of Algorithm

\( \text{BDVSA}_{MOOGA} \) 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 (\( V_{Vk} \)) 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 (\( DCV \)) is computed.

4.1.1 Raw and Average Fitness Computation

To compute the fitness for each \( V_{Vk} \) in the population (\( V_{Vk} \in PP \)), first the rank (\( RV \)) of each \( V_{Vk} \) is computed using \( DCV \) at Step 4 using the equation (4) (Fonseca et al. 1993; Deb 2014):

\[
RV_i = DCV_i + 1
\]  

(4)

At Step 5, the \( \mu(RV) \), which represents the number of view vectors \( V_{Vks} \) 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 \( V_{Vk} \) down to 1 to the least rank \( V_{Vk} \) (Fonseca et al. 1993; Deb 2014).

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

\[
aFV_i = N - \left( \sum_{j=1}^{RV_i-1} \lfloor \frac{1}{2} \times \left( \frac{RV_i}{\mu(RV_i)} - 1 \right) \right) \right)
\]  

(5)

4.1.2 Niche Count Computation

The niche count \( NCV \) for a view vector \( V_{Vk} \) computes the number of view vectors \( V_{Vk} \) in the population (\( PP \)), which are inside the distance specified by \( \sigma_{V_{sh}} \) (Goldberg et al., 1987). (Goldberg et al., 1987) defined \( \sigma_{V_{sh}} \) as the maximum distance between two view vectors so that they can be in the same niche. The purpose of \( \sigma_{V_{sh}} \) is to obtain diverse view vectors in the multi-objective space.
The value of $\sigma_{Vsh}$ is chosen to be 0.5 in this implementation. To compute the Niche count of $VV_k \in PP$, the following steps are performed:

At Step 7 of the algorithm, a normalized distance between a pair of view vectors $VV_k$ 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:

\[
d_{VV_k} = \sqrt{\left[\frac{(QEC_i - QEC_j)}{(QEC_{max} - QEC_{min})}\right]^2 + \left[\frac{(UPC_i - UPC_j)}{(UPC_{max} - UPC_{min})}\right]^2}
\] (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).

\[
sh\left(d_{VV_k}\right) = 1 - \left(\frac{d_{VV_k}}{\sigma_{Vsh}}\right)^\alpha; \quad \text{if} \quad d_{VV_k} <= \sigma_{Vsh}
\]
\[= 0; \quad \text{otherwise}
\] (7)

The value $\sigma_{Vsh}$ specifies the minimum distance to which the sharing effect is to be considered and $\alpha$ denotes the scaling of the computation. In this implementation, $\alpha$ is considered to be 1.

Finally, at step 7, Niche count ($NCV_i$) for $i$th view vector $VV_k$ 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).

\[
NCV_i = \sum_j (sh\left(d_{VV_k}\right)) \quad \text{where } j \in \{ \text{set of } \{VV_k\} \mid RV_i = RV_j \}
\] (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):

\[
sFV_i = \frac{aFV_i}{NCV_i}
\] (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):

\[
sFV_i\cdot\alpha = \frac{aFV_i}{NCV_i\cdot\alpha}
\]
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 $BDVSAMOGA$ to select Big data views is discussed next.

5. AN EXAMPLE

The $BDVSAMOGA$ 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 $BDVSAMOGA$ 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 $MCQ_j$ is computed using equation (1):

$$MCQ_j = \min \left[ \sum_k \left( m_k \times CV_k \times \left( 1 + \frac{1}{2} \times u_{fv_k} \right) \right) + \left( 1 - m_k \right) \times CV_k \times \left( 1 + \frac{1}{2} \times u_{v_k} \right) \right]$$  (1)
For example, consider the MCQi computation of view vector VV41 (5, 8, 6, 3) for Q2 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 VV41 and, therefore, is considered for materialization. Thus, MCQi for Q2 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 m9, m10, m11 and m3 is 1, as the related views are materialized, and m1 = 0, as the view V1 is not materialized.

\[
MCQ_2 = \text{Minimum of } \{(1 \times \text{CMV}_9 \times (1 + 0.5 \times \text{ufv}_9) + (1 \times \text{CMV}_10 \times (1 + 0.5 \times \text{ufv}_{10})
\}
\]

\[
= \text{Minimum of } \{(1 - 0) \times \text{CV}_1 \times (1 + 0.5 \times \text{ufv}_1) \};
\]

\[
MCQ_2 = \text{Minimum of } \{(1 \times \text{CMV}_9 \times (1 + 0.5 \times \text{ufv}_9) \};
\]

\[
MCQ_2 = \text{Minimum of } [70; 70; 13.5] = 13.5
\]

The values of MCQi for VV41 (5, 8, 6, 3) are shown in the Figure 7:
Next, the value of $QEC_{BDV}$ is computed using equation (2):

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

(2)

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

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

= 1924

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

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

(3)

For $VV4_i (5, 8, 6, 3)$, $UPC_{BDV}$ will be computed using the values of $UMV_i$ and $I_{mi}$ for the views of $VV4_i$ viz. $V5, V8, V6, V3$ and already materialized views $V9, V10, V11$.

Figure 6. Initial Population of Top-4 View Vectors

<table>
<thead>
<tr>
<th>VV4</th>
<th>Initial Population</th>
</tr>
</thead>
<tbody>
<tr>
<td>VV4_1</td>
<td>5 8 6 3</td>
</tr>
<tr>
<td>VV4_2</td>
<td>5 7 3 2</td>
</tr>
<tr>
<td>VV4_3</td>
<td>5 2 7 8</td>
</tr>
<tr>
<td>VV4_4</td>
<td>7 4 1 3</td>
</tr>
<tr>
<td>VV4_5</td>
<td>5 8 4 1</td>
</tr>
<tr>
<td>VV4_6</td>
<td>2 7 4 3</td>
</tr>
<tr>
<td>VV4_7</td>
<td>8 7 3 1</td>
</tr>
<tr>
<td>VV4_8</td>
<td>2 6 3 8</td>
</tr>
<tr>
<td>VV4_9</td>
<td>4 7 5 6</td>
</tr>
<tr>
<td>VV4_10</td>
<td>6 3 5 7</td>
</tr>
</tbody>
</table>

Figure 7. MCQ’s for VV4_1 (5, 8, 6, 3)

<table>
<thead>
<tr>
<th>MCQ1</th>
<th>MCQ2</th>
<th>MCQ3</th>
<th>MCQ4</th>
<th>MCQ5</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>13.5</td>
<td>5.8</td>
<td>29.7</td>
<td>72.5</td>
</tr>
</tbody>
</table>
$$(UPC_{adv})_{VV41} = \left( 1 \times UMV_5 / I_{m5} + 1 \times UMV_8 / I_{m8} + 1 \times UMV_6 / I_{m6} + 1 \times UMV_3 / I_{m3} + 1 \times UMV_9 / I_{m9} + 1 \times UMV_{10} / I_{m10} + 1 \times UMV_{11} / I_{m11} \right)$$

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

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

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

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 $VV4_i$ is dominated by view vector $VV4_{10}$, which has better $QEC$ and $UPC$ values ($QEC_{VV4_i} > QEC_{VV4_{10}}$ and $UPC_{VV4_i} > UPC_{VV4_{10}}$). It may be noted that no other view vector in the population dominates $VV4_i$. Thus, the count of view vectors dominating view vector $VV4_i$, $DCV$, 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):

$$RV_i = DCV_i + 1$$  \hspace{1cm} (4)

For example, rank of $VV4_i$ is computed as:

$$RV_i = DCV_i + 1 = 0 + 1 = 1$$
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: $\mu(1)=5$; $\mu(2)=2$; $\mu(3)=1$; $\mu(4)=0$; $\mu(5)=1$; $\mu(6)=1$; and $\mu(7)=\mu(8)=\mu(9)=\mu(10)=0$.

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):

$$aFV_i = N - \left( \sum_{j=1}^{(RV_i-1)\frac{1}{4}j} \right) - \left[ \frac{1}{2} \times \left( \frac{1}{4}RV_i - 1 \right) \right]$$

$$aFV_i = N - \left( \sum_{j=1}^{(RV_i-1)\frac{1}{4}j} \right) - \left[ \frac{1}{2} \times \left( \frac{1}{4}RV_i - 1 \right) \right]$$

$$aFV_i = N - \left( \sum_{j=1}^{(i-1)\frac{1}{4}j} \right) - \left[ \frac{1}{2} \times \left( \frac{1}{4}RV_i - 1 \right) \right]$$

$$aFV_i = 10 - \left( 0 \right) - \left[ \frac{1}{2} \times \left( 5 - 1 \right) \right] = 8 \quad \text{(5)}$$

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):

$$dVV_{k_1} = \sqrt{\left( \frac{QEC_1 - QEC_2}{QEC_{max} - QEC_{min}} \right)^2 + \left( \frac{UPC_1 - UPC_2}{UPC_{max} - UPC_{min}} \right)^2}$$

$$dVV_{k_2} = \sqrt{\left( \frac{1924 - 3707}{4771 - 1731} \right)^2 + \left( \frac{299.36 - 263.11}{351.28 - 263.11} \right)^2} = 0.71626$$

Similarly, the normalized distance for other pairs having rank 1:
Assuming minimum sharing distance ($\sigma_{Vsh}$) = 0.5 and $\alpha=1$; and using equation (7), the following values of shared distance is computed:

$$sh(d_{VVk_{25}}) = 1 - \left( \frac{d_{VVk_{25}}}{\sigma_{Vsh}} \right) \leq \sigma_{Vsh}$$

$$sh(d_{VVk_{25}}) = 0$$

$$sh(d_{VVk_{25}}) = 1 - \left( \frac{0.14031}{0.5} \right) = 0.71939$$

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

$$sh(d_{VVk_{11}}) = 1; \; sh(d_{VVk_{12}}) = 0; \; sh(d_{VVk_{15}}) = 0; \; sh(d_{VVk_{18}}) = 0; \; and \; sh(d_{VVk_{110}}) = 0.25681$$

Next Niche count is computed using equation (8):

$$NCV_1 = \sum_j(sh(d_{VVk_{ij}})) \; \text{where} \; j \in \{VVk_1, VVk_2, VVk_5, VVk_8, VVk_{10}\}$$
\[ NCV_1 = \text{sh}(dVVk_{11}) + \text{sh}(dVVk_{12}) + \text{sh}(dVVk_{15}) + \text{sh}(dVVk_{18}) + \text{sh}(dVVk_{10}) \]
\[ NCV_1 = 1 + 0 + 0 + 0 + 0.25681 = 1.25681 \]

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).

\[ ssFV = \frac{aFV_1 \times \frac{1}{4} (RV_{V1}) \times sFV_1 \sum_j (sFV_j)}{sFV_1 + sFV_2 + sFV_5 + sFV_8 + sFV_{10}} \text{; where } j \in \{VVk_1, VVk_2, VVk_5, VVk_8, VVk_{10}\} \]
\[ ssFV_1 = \frac{8 \times 5 \times 6.3653}{(6.3653 + 3.9442 + 3.5986 + 8 + 3.8659)} = 9.8786 = 9.8786 \]

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.
6. EXPERIMENTAL RESULTS

The BDVSAMOGA 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 BDVSAMOGA 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 ($ng$) = 25; Probability of crossover ($pc$) = 0.8; Probability of mutation ($pm$) = 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 BDVSAMOGA. 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.
Some of the typical view vectors obtained from BDVSA\textsubscript{MOGA} are compared with the view vectors obtained from BDVSA\textsubscript{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\textsubscript{MOGA} that dominate view vectors produced by BDVSA\textsubscript{VEGA}. The objective of Big data view materialization is to minimize the QEC and the UPC. The BDVSA\textsubscript{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
In addition, it may be noted that BDVSAMOGA has produced more diverse view vectors than BDVSAVEGA. Thus, BDVSAMOGA produce better quality view vectors in comparison of BDVSAVEGA.

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

<table>
<thead>
<tr>
<th>QEC</th>
<th>UPC</th>
<th>No of Views Materialized (k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1859</td>
<td>223</td>
<td>9</td>
</tr>
<tr>
<td>1951</td>
<td>197</td>
<td>8</td>
</tr>
<tr>
<td>2138</td>
<td>195</td>
<td>7</td>
</tr>
<tr>
<td>2341</td>
<td>176</td>
<td>7</td>
</tr>
<tr>
<td>2476</td>
<td>172</td>
<td>7</td>
</tr>
<tr>
<td>2663</td>
<td>170</td>
<td>6</td>
</tr>
<tr>
<td>2866</td>
<td>151</td>
<td>6</td>
</tr>
<tr>
<td>3646</td>
<td>140</td>
<td>5</td>
</tr>
<tr>
<td>3833</td>
<td>138</td>
<td>4</td>
</tr>
<tr>
<td>4743</td>
<td>133</td>
<td>3</td>
</tr>
<tr>
<td>5188</td>
<td>125</td>
<td>5</td>
</tr>
<tr>
<td>5375</td>
<td>123</td>
<td>4</td>
</tr>
<tr>
<td>5968</td>
<td>115</td>
<td>4</td>
</tr>
<tr>
<td>6225</td>
<td>112</td>
<td>3</td>
</tr>
<tr>
<td>6878</td>
<td>109</td>
<td>3</td>
</tr>
<tr>
<td>7580</td>
<td>101</td>
<td>5</td>
</tr>
<tr>
<td>7767</td>
<td>97</td>
<td>3</td>
</tr>
<tr>
<td>8360</td>
<td>89</td>
<td>3</td>
</tr>
<tr>
<td>9270</td>
<td>86</td>
<td>3</td>
</tr>
<tr>
<td>13437</td>
<td>80</td>
<td>4</td>
</tr>
<tr>
<td>14217</td>
<td>70</td>
<td>3</td>
</tr>
<tr>
<td>15127</td>
<td>67</td>
<td>3</td>
</tr>
</tbody>
</table>
Figure 16. Non-dominated Top-k view vectors

![Diagram showing non-dominated view vectors](image)

Figure 17. BDVSA\textsubscript{MOGA} Vs. BDVSA\textsubscript{VEGA}; QEC and UPC of Top-7 view vectors

<table>
<thead>
<tr>
<th>View Vectors of BDVSA\textsubscript{MOGA}</th>
<th>View Vectors of BDVSA\textsubscript{VEGA}</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEC</td>
<td>UPC</td>
</tr>
<tr>
<td>2138</td>
<td>195</td>
</tr>
<tr>
<td>2341</td>
<td>176</td>
</tr>
<tr>
<td>2476</td>
<td>172</td>
</tr>
</tbody>
</table>
REFERENCES


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.