Received December 5, 2017, accepted December 30, 2017, date of publication January 23, 2018, date of current version March 12, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2797048

Finding Top-*k* Dominance on Incomplete Big Data Using MapReduce Framework

PAYAM EZATPOOR, JUSTIN ZHAN, JIMMY MING-TAI WU, AND CARTER CHIU
Department of Computer Science, University of Nevada, Las Vegas, NV 89154 USA

Corresponding author: Jimmy Ming-Tai Wu (ming-tai.wu@unlv.edu)

This work was supported in part by the United States Department of Defense under Grant W911NF-17-1-0088 and Grant W911NF-16-1-0416, in part by AEOP/REAP programs, in part by the National Science Foundation under Grant 1625677 and Grant 1710716, and in part by the United Healthcare Foundation under Grant UHF1592.

ABSTRACT Incomplete data is one major kind of multi-dimensional dataset that has random-distributed missing nodes in its dimensions. It is very difficult to retrieve information from this type of dataset when it becomes large. Finding top-*k* dominant values in this type of dataset is a challenging procedure. Some algorithms are present to enhance this process, but most are efficient only when dealing with small incomplete data. One of the algorithms that make the application of top-*k* dominating (TKD) query possible is the Bitmap Index Guided (BIG) algorithm. This algorithm greatly improves the performance for incomplete data, but it is not designed to find top-*k* dominant values in incomplete big data. Several other algorithms have been proposed to find the TKD query, such as Skyband Based and Upper Bound Based algorithms, but their performance is also questionable. Algorithms developed previously were among the first attempts to apply TKD query on incomplete data; however, these algorithms suffered from weak performance. This paper proposes MapReduced Enhanced Bitmap Index Guided Algorithm (MRBIG) for dealing with the aforementioned issues. MRBIG uses the MapReduce framework to enhance the performance of applying top-*k* dominance queries on large incomplete datasets. The proposed approach uses the MapReduce parallel computing approach involving multiple computing nodes. The framework separates the tasks between several computing nodes to independently and simultaneously work to find the result. This method has achieved up to two times faster processing time in finding the TKD query result when compared to previously proposed algorithms.

INDEX TERMS Top-*k* dominance, incomplete data, bigdata, mapreduce, hadoop, dominance relationship, query processing.

I. INTRODUCTION

In a given dataset R with multiple dimensions d, an in-depth analysis may be required to find the most powerful or influential values throughout the dataset. The most influential values can also be referred to as the dominant objects over the other objects present in the data, based on a specific predefined definition, referred to as dominance definition. A value can be evaluated as a dominant value based on the dominance definition. Discovering the dominant values in a dataset helps to fulfill different data mining purposes. Suppose the dataset R has n objects (= items) from d dimensions that can be imagined as a two-dimensional array with a range of objects and dimensions. Each item in R, accommodates the corresponding value of *(n,d)*. In the real-life application, a database of movies with different movie ratings from a range of users is a reasonable sample of a multi-dimensional dataset. The values are the ratings for each movie m from user *0* to *n-1* represented for every object m as:

$$\sum_{i=1}^{m-1} i = (\sum_{p=0}^{n-1} p, \sum_{q=0}^{d-1} q) \tag{1}$$

Top-*k* dominance query finds the most powerful values which dominate other values in the same dimension by using a predefined scoring function. Top-*k* is one of the main uncertain queries that returns the top-*k* objects with the highest scores according to a scoring function [1]. The dominant values are distinguished using the dominance definition which defines when and how a value can be dominant over the other.

Finding the answer for TKD queries can be accomplished by using different methods and algorithms. While thinking about finding dominant values in a dataset, the first and

2169-3536 © 2018 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

easiest approach that might come to mind is to compare each two items in the dataset individually. This naive approach implies the pair-wise comparison between the values of different objects in the same dimension. In this approach, a comparison is required for every two values existing in the dataset. The observed dominant value in each comparison can be used to find the final top-*k* dominant values later. Indicating the top-*k* dominant values can be challenging when facing big data; processing time can become astronomical even if the computational cost can be ignored. Therefore, this approach may not be the best way of solving this problem.

Several algorithms have been proposed to apply the TKD queries with performance more acceptable and efficient than the naive approach. Based on [2] and [3], applying the top-*k* dominance query is possible in the incomplete dataset using Skyline query processing. The Skyline algorithm separates the uniform values into different buckets. The buckets group the dataset by dividing it into chunks that have same missing-value dimension. The buckets are easier and faster to process. By having separate top-*k* dominant values for each bucket and combining them together, determining the top-*k* values of the whole dataset becomes possible. The Upper Bound Based algorithm is another method which works by finding top scores using the bit-wise comparison between values, covered later in the paper [4].

The Bitmap Index Guided algorithm is another approach proposed by [4] that can greatly enhance the performance of the TKD query, although, our analysis shows that this difference is not significant in comparison to other algorithms like k-Skyband Based or Upper Bound Based. As the size of data increases over time with big data becoming more common in this field, it is useful to consider applying the same logic and algorithms more realistically and prepare them to face real-world problems with exponentially larger datasets. Based on the results indicated in [4], the performance of the BIG algorithm for finding TKD results has been improved by using tiny subsets of an incomplete multi-dimensional data. However, the performance of the BIG approach is not clear in real datasets which are exponentially larger in size.

In all of the aforementioned algorithms, incompleteness is the nature of the datasets for applying the TKD queries. The incompleteness in this data is independent, meaning that both present and missing values in this type of dataset are not related to each other and there is no way to find values based on others using any probabilistic approaches. For incomplete data, neither prior knowledge nor calculation of data is required. Derivation of values is assured and not based on probabilities, but in the uncertain data, the missing values can be found based on experience or prior information. Also, the probabilistic concept of TKD approach has been reviewed by [5] for its efficacy on missing data.

The results of top-*k* dominance algorithms help us to enhance our ability to obtain information and knowledge from unprocessed raw data that contain numerous missing values. The incomplete dataset will soon be the ever-present data in every system, and finding the top-*k* dominant values throughout a dataset helps us to design smart and intelligent systems such as movie recommenders that have strong, accurate, efficient, and real-time recommendations.

This paper explains an attempt to enhance the performance of the Bitmap Index Guided algorithm while dealing with large datasets by getting help from not only one machine, but having multiple processing machines working simultaneously to find the TKD query result in a fast and accurate way. Using single computing nodes, even with powerful computing components, is still not enough for processing the large real-time data, and the process duration makes those systems completely unresponsive. The machine power resources are not always able to accommodate the algorithm's metadata and temporary files. In those cases, limited processing power and memory capabilities become a significant difficulty. Ideally, the MapReduce framework is one productive method this paper tries to focus on in implementing a new enhanced algorithm that can efficiently apply TKD queries in a faster way by using multiple processing machines working simultaneously to find the TKD query results.

To our knowledge, this paper is one of the first works of considering big data in the area of finding the top-*k* dominance values. Applying the MapReduce framework on this subject is an innovative approach which guarantees enhanced performance. This work also has several different aspects of innovation in comparison to previous works which are focused on uncertain data or complete data. Efforts in this context are not only focused on incomplete data, but also massive, incomplete Big Data. Our work provides new ways of thinking about a specific usage area that has not been considered thoroughly so far.

The rest of this paper is organized by describing related work in the different areas of top-*k* and top-*k* Dominance, Incomplete Data, Bitmap Indexing, and MapReduce in Section II. Preliminaries and the problem statement are presented in Section III. A complete overview of the Bitmap Index Guided (BIG) algorithm and the related modifications have been provided in Section IV. Also, in Section IV, other currently available algorithms for applying TKD queries have been considered such as Skyband Algorithms and Upper Bound Based Algorithm, as well as the Bitmap Index Guided algorithm and its details. Thorough consideration of the related algorithms leads to the reveal of our proposed MapReduce Enhance Bitmap Index Guided Algorithm (MRBIG) which uses the MapReduce framework as the infrastructure for applying the TKD query that has been described in Section V. Further experiments and analysis of the efficiency of the algorithm, in addition to analysis of the results, have been provided in Section VI. Finally, conclusions and future work ideas are given in Section VII to provide some notions and possibilities for future research.

II. RELATED WORK

In this section, the related works about finding top-k dominance, incomplete data, bitmap indexing, and MapReduce is provided. First, an overview of the previous works for applying top-*k* dominating (TKD) queries will be provided. Then we proceed by explaining related works about incomplete data. Next, we consider the bitmap indexing related works, which help us to distinguish different approaches and compare to our work. Finally we explore applications of MapReduce in the field.

A. TOP-*k* DOMINANCE

TKD query finds the best values in a dataset based on a pre-defined goodness criterion in each use case. Dominance indicates that at least one attribute or value makes an object better than the object it dominates. The dominance admittance is based on a predefined definition or relationship which reviews the best dominance relationship in terms of handling constraints of the system. In the approach described in [6], budget constrained optimization query helps to increase the profitability of products. Furthermore, Yiu and Mamoulis [7] reviewed the TKD queries on multi-dimensional datasets. They propose ITD, an enhanced algorithm applied to indexed multi-dimensional datasets without using Skyline-based algorithms. Furthermore, they also proposed the LCG algorithm that computes upper bound scores using a tree structure, giving a relaxed version of the top-*k* dominating query. Reference [8] suggests a new algorithm that refines the object accesses throughout top-*k* processing. Lian and Chen [9] consider Probabilistic Top-*k* Dominating (PTD) query in uncertain data. Reference [10] works on uncertain data by applying probabilistic top-*k* queries by assigning a probability threshold. Their approach reduces the PTD search space by pruning the improbable values in the uncertain dataset. It finds the PTD query for values that dynamically dominate all possible values in the dataset. Han *et al.* [11] work on TKD queries on massive data, accomplished by sorting and listing values and making the process faster than other methods. Their proposed TDEP algorithm sorts and prunes the data with specifically selected objects, which helps to make multi-criteria decisions for the data.

Multi-dimensional databases are primarily used in applying the top-*k* dominating queries. Reference [12] study processing top-*k* dominating queries over dynamic attribute vectors where finding the distances depends on the defined metrics between objects. Their algorithms benefit from the applied metric space to solve the TKD query. Results of their work show that out of several reviewed approaches such as SBA and ABA, the pruning-based algorithms show the best performance. References [4] and [13] follow the combination of top-*k* and Skyline queries that led to top-*k* dominating query, and [13] process the more complex situations. Reference [14] provides the work on Skyline in Crowd-Enabled Databases with consideration of incomplete datasets. Their proposed query, continuous top-*k* dominating query (cTKDQ), can continuously generate answers to a query even after changes in the data. Reference [15] applies the top-*k* query on massive data without considering the Skyline queries which distinguishes it from top-*k* dominance queries, but establishes a relation due to the TKAP model which uses adaptive pruning processes on massive datasets to enhance performance. Reference [16] also follows the same structure as [15], but applied to monitoring purposes for tracking top-*k* queries. Also, [17] addresses the concurrency problems while applying TKD queries based on a service selection scheme to apply TKDs based on specific requests.

Papadias *et al.* [3] uses Skyline Computation to find dominated values. First, they introduce the branch-and-bound skyline (BBS) algorithm that accesses the skyline points using an R-tree and Nearest-Neighbor search, and then implements the TKD queries. Furthermore, top-*k* dominance has been further considered in [18], in which the top-*k* Dominance Range Query (TkDR) operator helps to find the most interesting objects in any uncertain datasets by taking advantage of probabilistic skyline queries, which try to enhance the TkDR performance of uncertain datasets.

Further details and definitions have been provided in [1] to give a better understanding of top-*k* and Skyline queries based on uncertain data. The presented concepts in [1] are helpful for grasping the underlying concepts of this paper. The mentioned efforts in this subsection provide different approaches for dealing with top-*k* dominating queries, and each may have different implementation processes. They consider the complete, incomplete, probabilistic and uncertain data and query types, with several performance outcomes. Our MRBIG approach adds a new and improved method for evaluating top-*k* dominating queries on a massive scale by improving the performance.

B. INCOMPLETE DATA

The main characteristic of incomplete data is the presence of missing values in its dimensions. Khalefa *et al.* [2] propose TKD query processing using the ISkyline algorithm that is especially suitable for incomplete data. Haghani *et al.* [19] study incomplete unsynchronized data streams and try to address the issue of continuously monitoring top-*k* queries through their efficient pruning approach. Soliman *et al.* [20] provides a probabilistic model and express types of ranking queries to apply the TKD query.

Razniewski and Nutt [21] assure the data integrity of query answers while dealing with incomplete data. Reference [22] considers the clustering of incomplete datasets in high-dimensional big data and improves performance for clustering. This approach also reduces the dimensions of the dataset using a hierarchical clustering structure. Reference [14] uses incomplete datasets and proposes an approach to overcome current defects while applying skyline queries, enhancing its effectiveness in crowd-enabled databases.

The incomplete data requires different processing methods than other types of data such as complete, probabilistic, or uncertain data. One must solve issues like the presence of missing values and how to manage the value absence without wasting available computing resources, and this requires innovative methods. The missing values in an incomplete structure require complex methods of computation which distinguish them from complete datasets [23]–[25]. This incompleteness can mean differences in approaches from processing information in a database to conducting relational operations [23], or searching and mining incomplete data [24]. Some works provide language to handle incompleteness and study algebraic methods to map them to completed datasets [25], [26]. Finding top-k dominance and relating it to the incomplete data is a major point that can differentiate the MRBIG approach from the previously mentioned works. This paper addresses further considerations in this field by considering big data.

C. BITMAP INDEXING

Bitmap Indexing is a way to ease the processing of non-uniform data. By using the bitmap indexing approach, the results discovery will be efficient and easier to handle in most cases. The bitwise operations in the Bitmap Indexing generate data patterns which are simpler for a machine to use and process, but in some cases might make the process over complicated. Big Data is one of the environments where bitmap indexing can be either useful or dangerous. Bitmap indexing can become an additional load for the system while dealing with multi-attribute data. Creating a bitmap index for a particular algorithm can be as complex as the algorithm itself in some cases. Therefore, compression of bitmap indexes has also emerged as a useful tool to make the bitmap indexes simpler to navigate and process.

Bitmap indexing compression speeds up the processes and increases the efficiency of the algorithms. The effects of compression on multi-component and multi-level compressed bitmap indexes has been considered in [27]. There are several methods for compressing bitmap indexes, such as BBC, CONCISE, and WAH methods that [28] has considered thoroughly and reviewed in the context of Big Data. For instance, to make the compression more stable, [29] has proposed their word-aligned hybrid code (WAH) as a compressing method for bitmap indexes that leads to improved performance. Each method has different CPU and GPU runtimes as well as varying segmentation, chunking, etc., configuration characteristics that are utilized for various use cases [28].

Miao *et al.* [4], review the TKD query on incomplete data as one of the first attempts to solve this issue using Bitmap Index Guided (BIG) algorithm. They use the Bitmap Indexing as the infrastructure for their algorithm and to perform TKD queries on incomplete datasets. After conducting compression on the bitmap indexing, they propose the IBIG algorithm, which uses an improved version of the bitmap index tables. There is no significant difference in the performance between the BIG and IBIG algorithms.

D. MapReduce

The MapReduce framework has been used to greatly reduce the runtime of parallelizable algorithms on big data by way of distributing the workload to run simultaneously on multiple machines. Recent significant research has been done in the area of developing algorithms for use in MapReduce. Manogaran and Lopez [30] propose a MapReduce disease surveillance system for analyzing correlation between climate data and Dengue fever transmission in real time. Kamal *et. al.* [31] suggest a k-nearest neighbor classifier for imbalance data reduction by employing MapReduce, applying the method to a big DNA dataset with 90 million base pairs. In a different paper, Kamal *et. al.* [32] apply MapReduce to de Bruijn graphs to more efficiently and accurately perform metagenomic gene classification. Research in improving the MapReduce framework and its encompassing Apache Hadoop architecture for use in big data has also been conducted; Matallah *et. al.* [33] propose enhancements to the storage of metadata in the Hadoop Distributed File System for improved scalability, demonstrating the continuing value of MapReduce in modern applications.

III. PROBLEM STATEMENT

In this section, we address the issue of finding top-k dominant values and illustrate the details of implementation of the MRBIG algorithm. TKD query on incomplete data starts with an incomplete dataset R with n dimensions and m items. Dimensions are indicated as

$$\sum_{i=1}^{n} d_i \tag{2}$$

By using the pairwise comparison method, the objective is to find the items that are dominant over the other values in the dataset. If item m_1 dominates over m_4, it is denoted as $m_1 \succ m_4$. Consider an item m_1 having four dimensions, the depiction of which is provided below:

$$m_1 = (d_1, d_2, d_3, d_4) = (2, -, 1, 0)$$

Each missing value is represented as a dash (-). We define a dominance definition which let us decide which value(s) can be dominant based on this definition. For instance, consider the following theorem which defines our dominance definition. The definition remains primarily the same throughout this context.

Dominance Definition: Given the two items m_1 and m_2, m_1 dominates over m_2 if $\forall d$ the values in the dimensions of m_1 are larger than the values in the dimensions of m_2, excluding the missing values for all dimensions. In other words, the condition for being dominant is as follows:

$$\forall\, m_1[\forall d_i] \text{ and } m_2[\forall d_j] : m_1[\forall d_i] > m_2[\forall d_j] \tag{3}$$

The domination of m_1 over m_2 is denoted as $m_1 \succ m_2$ while holding the above condition.

In the dominance definition, the basis of dominance is based on the larger value between two corresponding dimensions. In other words, the greater value is a better value based on the dominance definition. Dominance definitions define the strength of a value in a dataset and are a vital part of deciding what values are considered as dominant in any particular usage. Going back to the sample item m_1, by comparing $m_1 = (2, -, 1, 0)$ and item $m_5 = (-, -, 3, 2)$ as an example, the dominance is given to $m_5[d_3] = 3$ in comparison to $m_1[d_3] = 1$. Based on the Dominance Definition, item m_5 dominates m_5 due to the larger value it has. Missing values are not considered because they are not making any changes to the result. For example, users that have not submitted any rating for item m_5 cannot be a part of TKD query processing for item m_5.

IV. TKD QUERY ON INCOMPLETE DATA

In this section, we review the procedure to apply the TKD query to incomplete data as well as the problem statement for finding the top-*k* dominant values. Various algorithms have been proposed to handle Top-*k* dominance. Some of these algorithms handle incomplete data, for which an overview is provided later. Further down in the paper, the structure and functionality of the Bitmap Index Guided algorithm (BIG) will be considered.

A. SKYBAND BASED ALGORITHMS

In order to apply TKD query to a dataset, the easiest approach that first comes to mind is to compare the values of the whole dataset by doing a pairwise comparison. This method can be helpful for small datasets, but following this approach for larger datasets causes poor performance and potentially total failure because of the high volume of resource-exhaustive comparisons. There are many flaws to this approach. The pairwise comparisons require a prohibitively long runtime to examine every single value in the dataset and massive storage to keep track of the progress. The inefficiency gets worse while dealing with the larger datasets that comprise big data.

To make the process of finding top-*k* dominant values practical and efficient, we need to have more sophisticated algorithms. One subset of algorithms designed for this purpose are the Skyband Based algorithms [34], which have been a valid solution for incomplete data. Extended Skyband Based [4] and Expired Skyline algorithms [35] are among the methods that utilize the same concept.

These processes uses normalization methods by categorizing the data into different parts based on their missing values. Each item in the dataset redirects to its corresponding bucket based on the pattern in their missing values. Each bucket contains similar items and their values. For instance, (-, 2, 4, 6) and (-, 8, 3, 2) both go into the same bucket due to having the missing values in the same dimensions. Following the same logic, each bucket populates its members, and eventually, dataset would be completely divided into different groups.

Among each group, the dominant candidate sets will be calculated. The candidate sets are created for each bucket. Candidate sets can have multiple possible values and depict all of the values in a bucket which can be a dominant value. By looking at the candidate set which is smaller in size and contains less data, the TKD query can be applied easier. By intersecting the whole candidate sets from the buckets, the TKD query answer can reveal the final top-*k* result of the dataset.

TABLE 1. Sample incomplete dataset.

	d_1	d_2	d_3	d_4		d_1	d_2	d_3	d_4
m_1	–	1	2	–	m_6	–	–	–	3
m_2	1	–	3	2	m_7	1	1	–	–
m_3	3	1	–	–	m_8	–	3	2	–
m_4	–	–	–	1	m_9	2	–	2	2
m_5	–	2	1	–	m_{10}	3	2	–	–

To clarify the concept, take Table 1 as an example incomplete dataset. As can be seen in Table 2, m_1,m_5 and m_8 have been bucketed into b_1 since they have the same missing-data dimensions. Starting from the second dimension, can be seen that $m_8 \succ m_5$ on both dimensions d_2 and d_3 as well as the $m_5 \succ m_1$ on the dimension d_2. The dominance on the dimension d_3 is not strictly distinguishable as both values are the same, further processing passes to next steps and comparison to all candidate sets from the whole dataset. The candidate set (S_c) can be formed as illustrated by Table 3.

The Skyband bucketing method cannot always be efficient. The worst case scenario to this approach is when the size of S_c equals the size of the dataset. In Table 3, every bucket has more than one candidate, and the worst occurs in b_4 when all of the values goes into S_c. In general, the performance of Skyband bucketing method decreases as the size of S_c increases.

This approach gives us the ability to eliminate multiple dimensions at once and constructs a temporary complete dataset for the Skyband algorithm which helps to process the remaining values much faster. Observing real datasets with thousands of items and dimensions can suggest a significant difference in speeding up the process. However, as shown above, there are still important imperfections to this approach, which in some cases can make the entire process inefficient.

After obtaining all of the candidate values from each bucket, it is possible to make final pair-wise comparisons and reach the final TKD result. Having the S_F set as the combined version of the candidate sets, the following summation gives the final input for performing the TKD query based on the Table 3:

$$S_F = \sum_{n=1}^{4} S_c[b_n] \tag{4}$$

By obtaining Table 4, final results for the top-*k* dominant values can be found by comparing the values at each dimension separately. It can be inferred that this process involves pairwise comparison, and consequently inherits the weaknesses of that methods as well.

As mentioned earlier, the performance of Skyband based algorithms is strongly dependent on the size of S_F. The size of this set can directly increase the complexity of the TKD process and affect the performance of the whole system.

By considering Skyband Based algorithms for the MapReduce framework, using a small S_F can exhaust the MapReduce cluster with simple calculations that waste runtime with massive communications cost in synchronizing

TABLE 2. Skyband based data bucketing method.

b_1			
—	1	2	—
—	3	2	—
—	2	1	—

b_2			
—	—	—	1
—	—	—	3

b_3			
3	1	—	—
1	1	—	—
3	2	—	—

b_4			
1	—	3	2
2	—	2	2

TABLE 3. Candidate set (S_C) formation for potential TKD items.

b_1			
—	1	2	—
—	3	2	—
—	2	1	—

b_2			
—	—	—	3
—	—	—	1

b_3			
3	2	—	—
3	1	—	—
1	1	—	—

b_4			
1	—	3	2
2	—	2	2

TABLE 4. Final candidate set (S_F) for skyband based algorithm.

S_F			
3	2	—	—
2	1	—	—
2	—	2	2
1	—	3	2
—	3	2	—
—	1	2	—
—	—	—	3

the computing nodes. Alternatively, by considering a single machine procedure, a large S_F can overload the machine with large, inoperable values. It is also possible that S_F equals the dataset size as mentioned before. This exceptional situation motivates the use of another algorithm that can address these issues.

B. UPPER BOUND BASED ALGORITHMS

We described the defects of Skyband algorithms and the processing overload problems when facing massive data. Having a new approach that can address the issue can hugely improve the performance of finding top-*k* dominance queries. Another approach for TKD is to use an integrated evaluation of items that is no longer based on the pair-wise comparison. UBB is one of the available algorithms to apply TKD queries on incomplete data. It relies on the upper bound values in a dataset for retrieving the answer of TKD query [4].

In any given dataset, there is always one criterion to evaluate the values to empower the decision to choose top-*k* dominant values based on the dominance definition. So for each dominance definition, the first goal is to define criteria that work uniquely to distinguish between different values. After finding the criteria, a unique approach is to score values based on their power according to dominance definition. A score is a number that is dedicated to each dataset to evaluate the values and make the decision of final TKD value possible. Scoring helps us by resulting in fewer calculations and eliminating pair-wise comparison.

The UBB algorithm is mainly based on finding the frequency of the items that our picked value dominates. For complete data, the challenge is to compare each value with the other ones in the same dimension. For our desired multi-dimensional incomplete data, as can be seen in the sample dataset from Table 1, UBB considers each dimension independently and compares the selected value to other corresponding values and finds the frequency of values it dominates.

The Upper Bound Based algorithm alters the dataset to make it easier to deal with by separating each dimension. This characteristic can be imagined as looking at the columns of a two-dimensional array and processing each column separately. Dataset R, having m items and d dimensions, contains d separated columns to process. The dominance definition can tell which value is dominant over a chosen item and assigns it to the proper B_i set. B_i is a list of all values that i dominates. For example, B_2 for m_5 shows all values that m_5 dominates on the second dimension. Obviously, the B_i for missing values are not considered as it is the same as the whole dataset for each dimension. By having B_i sets and obtaining the size of each B_i, UBB stores all the sizes into a single structure. Depending on the dominance definition, the answer is retrieved based on the scores.

The UBB algorithm suffers from the generic high volume exhausting pairwise comparison problem. This algorithm tries to separate the dimensions and then compare the whole dimension with the value it has. Having massive data requires a tremendous processing procedure to apply the Upper Bound method on the real world problems.

C. BITMAP INDEX GUIDED ALGORITHM

As reviewed in the previous sections, present methods enhance the performance in different ways, but they still have various problems which cause inefficiencies in some cases. Using the previously discussed naive approach involving pairwise comparison requires a significant amount of time and inefficiency, even with smaller data. Using the Skyband algorithms are also substantially dependent on the size of the dataset, and having large datasets creates huge buckets of data so that applying the TKD query on each would still be inefficient. UBB algorithms cannot be sufficiently efficient as they require numerous comparisons between values. The Bitmap

TABLE 5. Bitmap index table for the sample dataset.

Items	d_1	–	1	2	3	d_2	–	1	2	3	d_3	–	1	2	3	d_4	–	1	2	3
m_1	–	0	0	0	0	1	0	1	1	1	2	0	0	1	1	–	0	0	0	0
m_2	1	0	1	1	1	–	0	0	0	0	3	0	0	0	1	2	0	0	1	1
m_3	3	0	0	0	1	1	0	1	1	1	–	0	0	0	0	–	0	0	0	0
m_4	–	0	0	0	0	–	0	0	0	0	–	0	0	0	0	1	0	1	1	1
m_5	–	0	0	0	0	2	0	0	1	1	1	0	1	1	1	–	0	0	0	0
m_6	–	0	0	0	0	–	0	0	0	0	–	0	0	0	0	3	0	0	0	1
m_7	1	0	1	1	1	1	0	1	1	1	–	0	0	0	0	–	0	0	0	0
m_8	–	0	0	0	0	3	0	0	0	1	2	0	0	1	1	–	0	0	0	0
m_9	2	0	0	1	1	–	0	0	0	0	2	0	0	1	1	2	0	0	1	1
m_{10}	3	0	0	0	1	2	0	0	1	1	–	0	0	0	0	–	0	0	0	0

Indexed Guided is another algorithm that helps us to find the top-*k* dominant values by initially generating a bitmap index table, and then accommodating values based on the same format. This method includes the bitmap indexing method to solve the scoring problem and accelerate the process.

First, an overview of the Single Machine algorithm (the conventional way of using a machine to apply the TKD query on incomplete big data) will be provided. This is followed by the proposed MRBIG algorithm, which will be introduced and discussed at length.

1) SINGLE MACHINE ALGORITHM

For a given dataset R containing numbers of items in different dimensions, each dimension is represented as a group of values v. The value v_i represents the range of all numbers for dimension i in R. For example, in Table 1, v_1 is $= (-, 1, 2, 3)$ which shows all of the present values in dimension d_1. In the provided example in Table 1, all v_i values are the same because all dimensions have the same range of present values. In real-world problems, the benefit of having v_i is to give the power of creating dynamic bitmap index tables for each dimension d to save storage and improve performance.

The bitmap index creates separate columns for each value in v_i. Having v_i gives a full representation of present numbers in each dimension among all items. The bitmap index table would be initialized to 0, and then based on each item we would modify the values in the table as described below.

- For each missing value, leave the fields in the corresponding row without any changes. So each v_i with a missing value remains as all 0s.
- For each number we have in v_i, insert number 1 in the corresponding row, and all of the following right rows.

For a better illustration of what the values would be after modifying the bitmap, suppose we consider all v_i values for the items in the sample dataset in Table 1. As can be seen, the items have four different values overall. Missing values and also the numbers 1, 2 and 3 are among the members of v_i for this dataset. These three are among all of the possible values in the dimensions of items in R. The generated bitmap index table can be seen in Table 5.

Using a single computing machine to execute the process of Algorithm 1 and creating bitmap index tables for incomplete data is not possible while dealing with large datasets with hundreds of thousands of columns and rows. The described approach and Algorithm 1 works with best performance when the data does not exceed certain thresholds. Otherwise, there are massive run-time and storage requirements to run the TKD query using the mentioned algorithm.

Algorithm 1 Single Machine(BIG) Pseudo Code

```
1:  Score calculation for item $m_i$
2:  Create [P] and [Q]
3:  for each item $m_i$ do
4:      for each column in $v_i$ do
5:          temp ← the first 0 in the $m_i$-th row
6:          ind ← Index(temp)
7:          [P] ← append($\sum_{j=1}^{n}$ [$m_j$, ind])
8:          [Q] ← append($\sum_{j=1}^{n}$ [$m_j$, ind+1])
9:          nonD($m_i$)
10:         $P \cap P^*$
11:         $Q \cap Q^*$
12:     end for
13:     $\alpha = P^* - Q^* - nonD$
14:     $\beta = count(P^* -)$
15:     $score = \alpha + \beta$
16:     $maxscore[i] \leftarrow score$
17: end for
18: finish when length( $maxscore$ ) = n
19: Finding Top-$k$
20: sort( descending ( $maxscore$ ) )
21: return top-$k$
```

By keeping the mentioned problems in mind as an incentive, the proposed algorithm is an effort to make the process of finding top-*k* dominant values in incomplete data efficient in both time and storage while dealing with large files. To avoid redundancy, P and Q sets are defined in later sections.

2) MapReduce MODIFIED ALGORITHM

As the single machine procedure is not capable of dealing with large datasets, another method is required to bolster this approach. The single machine procedure (referred as the BIG algorithm) provides good performance for small

TABLE 6. Simple example for the word count problem.

Apple	Orange	Mango
Orange	Grapes	Plum
Apple	Plum	Mango
Apple	Apple	Plum

datasets, but as the size of dataset increases, the performance weakens. Thus, another strategy is necessary to make the BIG algorithm practical on larger files.

Incomplete big data is ubiquitous, and TKD query processing on this data class needs to be addressed. In this paper, we propose the MRBIG algorithm, which enables us to apply TKD queries on incomplete big data using the MapReduce framework. The complete explanation of our proposed algorithm is detailed below.

V. MRBIG: MapReduce ENHANCED BITMAP INDEX GUIDED ALGORITHM

The MapReduce framework evolved when data became too large for machines to process. The processing time of certain tasks can take months using one machine, and efforts to significantly increase a single machine's computer power was impractical and expensive. As mentioned before, MapReduce framework has two fundamental functions called Mapper and Reducer, which are used to separate huge tasks between multiple nodes to make them faster. Each task, when applied to the MapReduce framework, is split into different chunks based on internal patterns and gets distributed between nodes. After assigning these data fragments to the Mapper, each piece is processed and the output returned. Later, the Mapper results are aggregated to calculate the final result using the Reducer.

In the MapReduce framework, the communication time is one issue that has to be considered. Synchronizing the nodes and making them informed of the status of tasks, in addition to sending and receiving data amongst them, are major factors that can impact computation cost in MapReduce. There are vast domains that MapReduce can be applied to enhance performance. Time inefficiency in top-*k* dominance is one major issue that this paper addresses by applying the MapReduce framework. A majority of complex problems such as text processing systems and data mining technologies can be handled using the MapReduce framework, making systems operable in real-time with outputs computed in a minute fraction of the time needed in a single machine approach.

One generic MapReduce example is counting the frequency of words in large text files (Table 6). The dataset first divides into different slices, each of which contain a smaller portion of the original dataset. Each piece of the dataset goes to a different Mapper, and each Mapper performs the same process of word counting. In this manner, Mapper generates a simple line by line output wherein each line contains the word itself and the number of appearances, as shown in the Table 7.

TABLE 7. Mapper result.

Mapper1	Mapper2	Mapper3	Mapper4
Apple,1	Orange,1	Apple,1	Apple,1
Orange,1	Grapes,1	Plum,1	Apple,1
Mango,1	Plum,1	Mango,1	Plum,1

TABLE 8. Top: Results after sort-and-shuffle; Bottom: Reducer appended the results.

Apple,1	Orange,1	Mango,1	Grapes,1	Plum,1
Apple,1	Orange,1	Mango,1		Plum,1
Apple,1				Plum,1
Apple,1				
		↓		
Apple,4	Orange,2	Mango,2	Grapes,1	Plum,3

The process starts with the first word of the document and creates a line that contains the word and the number of appearances. The output is sorted and cleaned, which is done by the framework automatically using Sort-and-Shuffle. Sort-and-Shuffle aligns the words in separate places then together. The internal function of MapReduce framework prepares the results for the Reducer (Table 8). After obtaining the results of Sort-and-Shuffle, the Reducer appends all the results and combines the Mapper results to assemble the final result. Table 8 displays the sorted and shuffled result and the resulting Reducer output.

MapReduce framework does not require a high volume of processing and the results can be calculated rapidly while dealing with big files and datasets. There is no significant change between using the Hadoop MapReduce framework or traditional single machine code in the small cases. In such cases, using the single machine can be more effective by avoiding synchronization and metadata exchange costs. Not having that overhead make the processes faster in smaller datasets. However, the single machine procedure is not helpful with big datasets as the number of calculations and comparisons can and do increase exponentially.

MapReduce framework is a fast procedure for dealing with big datasets by using simple programming methods, but it still has some flaws. The amount of time which is required to synchronize the data in different nodes is a factor to weigh. Network congestion and delay are among the important factors which cannot be left unconsidered. Also, recovering from error is another aspect of MapReduce which can worsen performance. If a node gets disconnected or some error occurs that interrupts the node, data recovery or node suspension for continuing the process can inflate runtimes. However, MapReduce framework now can handle most of the issues autonomously.

Based on the characteristics mentioned above, MapReduce shows promise as a reliable method to implement our proposed algorithm for finding top-*k* dominant values in incomplete big data. Hadoop clustering method used for implementation and two mathematically proven lemmas have been provided to evaluate the effectiveness of the MRBIG algorithm.

To start implementing the MRBIG algorithm, preparation and pre-formatting of the dataset are vital steps to ensure the accuracy of the input. Bitmap indexing is another aspect of implementation that has to be defined. A description of the construction of the Bitmap Index table has been provided in Section IV-C1, but in this section, we provide more detailed explanation to illustrate the concept.

A. BITMAP INDEXING

Dataset R is a multi-dimensional incomplete big dataset that has m items and d dimensions. The size of the dataset is millions of times larger than the previous sample datasets used by the single machine approach. Throughout the dataset R, a two-dimensional matrix can accommodate rows and columns in itself. As discussed in Section IV-C1 and shown in Table 5, v_i provides a range of all present values in dimension i. If each column is considered separately, a range of all present values in that specific column can be found. Having the v_i helps us to construct the bitmap index table for the algorithm and gives the power to create dynamic bitmap index tables based on the dataset values. For instance, $v.[6] = [-, 1, 2, 5]$ shows that in the 6_{th} dimension of data all present values are missing values and 1, 2, 5. This helps us to smartly construct the bitmap index table and tells us the number of columns required for the bitmap index table for each dimension.

By having v_i for each dimension, the Bitmap Index table can be constructed. By having d as dimensions and v_i there exists the following condition and bitmap index table can be initialized after this step:

$$\sum_{i=1}^{d} v_i \tag{5}$$

To generate the Bitmap Index Table to continue the steps of the MRBIG algorithm, as can be seen in Table 5, the table would be initialized to 0. By following the appropriate rules, the proper values are inserted into the table.

As mentioned earlier, there are two rules to follow for filling up the values in the Bitmap Index Table. First, for each missing value, we leave the fields of that row and the range of columns that spans without any changes. Second, for each non-missing value in v_i, the corresponding field and the following rightmost fields changes to 1.

By repeating the described process for every object and dimension, the Bitmap Index Table will be generated. In the Single Machine Algorithm (Section IV-C1), a bitmap index table representation has been provided to clarify the concept. The bitmap index table will be used to run the algorithm and find the top-k dominant value(s) in any incomplete dataset.

B. MRBIG STRUCTURE

To evaluate the values based on a power which is defined in the dominance definition, a scoring method is vital to examine values in a comparable way. Having a scoring method enables us to compare values by the score they acquire throughout the dataset. Higher scores are better according to the dominance definition. The scoring method has been modified to match and take advantage of the MapReduce framework structure. The pre-defined scoring function is embedded into the MRBIG algorithm.

Algorithm 2 MRBIG Algorithm (Having n Items and r Dimensions)

```
1:  Create [$P^*$] and [$Q^*$]
2:  for item $m_i$ in $\{m_1, m_2, \ldots, m_n\}$ do

3:      Mapper:
4:      Map(split($\sum_{j=1}^{r} d_j$))  // split by dimensions
5:      Bitmap($d_j$)
6:      for each dimension $d_j$ do
7:          Create [$P_j$], [$Q_j$] and $nonD_j$
8:          for each $v_{d_j}$ in $m_i$-th row do
9:              Candidate ← index( if $(m_i, v_{d_j}) == 0$ )
10:             [$P_j$] ← append($\sum_{i=1}^{n}$ [$m_i$, Candidate])
11:             [$Q_j$] ← append($\sum_{i=1}^{n}$ [$m_i$, Candidate+1])
12:         end for
13:     end for

14:     Reducer:
15:     for $\sum_{1}^{r} i$ do
16:         [$P_i$] ∩ [$P^*$]
17:         [$Q_i$] ∩ [$Q^*$]
18:     end for

19:     $\lambda = [Q^*] - [P^*]$
20:     for each $i$ in $\lambda$ do
21:         $\phi$ = count ( if $\lambda_i \not\succ m_i$)
22:         $nonD \leftarrow \lambda_i$
23:     end for
24:     $\alpha = |\lambda - nonD|$
25:     $\beta = count(P^* - \emptyset)$
26:     $score = \alpha + \beta$
27:     $maxscore[i] \leftarrow score$
28:     finish when length($maxscore$) = $n$

29: end for
30: Finding Top-$k$
31: sort ( descending ($MaxScore$))
32: return top-$k$
```

Based on the inherent characteristics of MapReduce, the most logical approach would be to calculate the score for each particular dimension. The mapper component in the MapReduce calculates the score as it always accommodates a fixed amount of dimensions.

There are also three internal sets that form the MRBIG algorithm and helps us to acquire the top-k dominant values in a dataset for each object m.

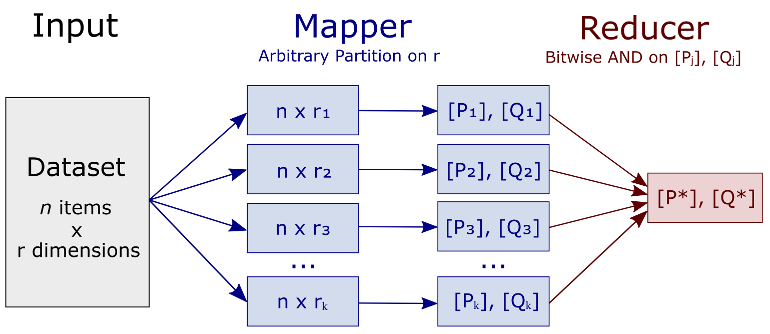


FIGURE 1. **High-level representation of the application of MapReduce in Algorithm 2 (MRBIG).**

One of the important sets that have a significant role in the algorithm is called [Q]. [Q] is defined as a set of objects which is not better than m or the values missing excluding m based on the dominance definition in that particular dimension. The other set [P] is a subset of [Q] and can be defined as a set of objects that are strictly worse than object m or missing from that specific dimension. The last and one of the most important components of the MRBIG algorithm is the [$nonD$] set, as it can be implied from its name, demonstrates the set of objects that are not dominated by the considered object m.

To provide a better understanding, suppose we have a movie recommender system that calculates the most popular movies among users' ratings. Suppose that the approach is to calculate the score of each dimension in the Bitmap Index Table (sample shown in Table 5) by using one Mapper for processing each dimension. The Mapper calculates the candidate sets [Q] and [P] for each processed dimension which is useful to find the top-rated movies of a particular user. This process is described in detail in lines 4-13 of Algorithm 2. After the Mapper calculations, the Reducer intersects the resultant sets from the Mapper, leading to [Q^*] and [P^*] respectively, as detailed in lines 15-18. The operations performed by MapReduce are displayed diagrammatically in Figure 1. Furthermore, the [$nonD$] calculation takes place after finding the [Q^*] and [P^*] values. Finding the [Q^*] and [P^*] would be the main objective of each MapReduce process, and the score calculation of top-k values from one specific dimension takes place in the remainder of the MRBIG algorithm.

One of the main requirements for applying the TKD query is to have the score for each item in the dataset. The main objective of the MRBIG algorithm is to help to make the score calculation of each item as fast as possible by taking advantage of MapReduce framework. The scoring method is a key feature that can affect the algorithms performance significantly. In this context, the score is a criterion that shows how powerful an item must be to be a top-k dominant value. This value is calculated based on how many items an item is dominating and the ratings for that item. The highest scores would be considered as the answer to the TKD query.

Finding the score of each item requires a look at the whole set of dimensions and finding the [P], [Q], and [$nonD$] sets for each dimension in a Mapper, later appending all of them to find the final score of the item. Therefore, for each item, one complete review of all the dimensions is required. Later, an outer loop is used to follow this procedure and find scores for each object. The scores are stored in *maxscore*, which is a data structure containing the scores of all items. By looking at the top values, the TKD final answer of the large incomplete input dataset is found.

To clarify the MRBIG procedure and the whole TKD process concept so far, we review an example by looking to find the score of the item m_4 based on the Table 5. The MRBIG algorithm starts to send each dimension to one Mapper and calculate the [P] and [Q] sets from each dimension. $\sum_{i=0}^{4} = P_i$ and $\sum_{i=0}^{4} = Q_i$ are the results from all of the Mappers which are available for the Reducer to process. It intersects the sets together and calculates the [P^*], excluding the zero values ($\emptyset$) which primarily help us eliminate any values that are incomparable, and [Q^*]. Then, the Reducer calculates the α and β values as can be seen in Algorithm 2 and appends the score of m_4 to the *maxscore* data structure to its corresponding index that represents m_4 score. By repeating this process for each item, we obtain a complete *maxscore* data structure containing all of the scores, and at this point we sort *maxscore* to identify the top-k dominant values.

Our approach in the MRBIG algorithm is based on numerous mathematical assumptions. It is worth mentioning the mathematical proofs for each of the processes take place during the transition to the MapReduce framework. The method of candidate set calculations for [P] and [Q] is the major difference between the Single Machine and MapReduce approaches. MRBIG Algorithm divides the sets to make the process faster and send each portion to different computing nodes, while the single machine uses one calculated [P] and [Q] and proceed to the rest of the algorithm. These two ways of calculating [P] and [Q] are proven mathematically to result in the same output by the following lemmas.

Lemma 1: Let $[P] = \bigcap_{i=1}^{d}[P^i]$ *where each* $[P^i]$ *is a binary string of length n. Let* $P_1, P_2, \ldots, P_k$ *be an arbitrary partitioning of the set of all* $[P^i]$, $k \geq 1$, *and where for any partition j,* $|j| \geq 1$. *If* $[P^*] = \bigcap_{j=1}^{k}(\bigcap_{P^m \in P_j}[P^m])$, *then* $[P] = [P^*]$.

Proof: By definition, intersection and its bitwise equivalent AND is commutative and associative. Therefore the order of intersection does not matter, and $[P] = [P^*]$ regardless of partitioning. □

It is clear that $[P^*]$ and $[Q^*]$ are sets that have been aggregated from multiple, variably-sized smaller ones which come from computing nodes and are ultimately intersected in a final reducing step. The same naming convention is followed for sets $[P]$ and $[Q]$ for Single Machine resultant sets and $[P^*]$ and $[Q^*]$ for MapReduce resultant sets. Lemma 1 shows a proof that for $[P^*]$, based on the definition for $[P]$, it is possible to claim that the resulting $[P^*]$ from MRBIG algorithm (Algorithm 2) is mathematically equivalent to the resulting $[P]$ of Single Machine Algorithm (Algorithm 1).

Lemma 2: Let $[Q] = \bigcap_{i=1}^{d}[Q^i] - \{o\}$ *where each* $[Q^i]$ *is a binary string of length n and o is the current object. Let* $Q_1, Q_2, \ldots, Q_k$ *be an arbitrary partitioning of the set of all* $[Q^i]$, $k \geq 1$, *and where for any partition j,* $|j| \geq 1$. *If* $[Q^*] = \bigcap_{j=1}^{k}(\bigcap_{Q^m \in Q_j}[Q^m]) - \{o\}$, *then* $[Q] = [Q^*]$.

Proof: By Lemma 1, we know $\bigcap_{j=1}^{k}(\bigcap_{Q^m \in Q_j}[Q^m]) = \bigcap_{i=1}^{d}[Q^i]$, since each $[Q^i]$ is just a binary string. Subtracting o from both sides gives the desired result. □

Following the same explanation for $[Q]$ candidate set, each $[Q]$ which has been calculated by Single Machine Algorithm is similar to $[Q^*]$ candidate set coming from MRBIG Algorithm. In other words, there are several $[Q]$ sets that have been calculated in different computing nodes for MRBIG Algorithm.

By having three computing nodes, each node calculated its own $[Q_i]$ set based on the in-hand fraction of the input file, with the resulting sets after mapper processing being $[Q_1]$, $[Q_2]$ and $[Q_3]$ respectively. The MRBIG algorithm intersects the sets by $\bigcap_{i=1}^{3}[Q^i]$ to arrive at $[Q^*]$. Lemma 2 shows that $[Q^*]$ equals $[Q]$, where $[Q^*]$ is the resulting set from the MRBIG algorithm and $[Q]$ is the resulting set from the Single Machine approach.

By using Lemma 1 and Lemma 2, it is proven that the resulting sets are mathematically correct and the approach in MRBIG algorithm can be genuinely referenced and followed.

VI. EXPERIMENT AND ANALYSIS

This section examines the effectiveness of the BIG and MRBIG algorithms and a comparison of their performance using different real and synthetic datasets. The experiment starts with performance evaluation of Single Machine algorithm and follows with the assessment of the proposed MRBIG algorithm based on metrics such as CPU time, item frequency, dimension fluctuations, and incompleteness rate.

The real dataset originates from an authentic user-initiated source that can help us evaluate the MRBIG performance in a real-world problem. The *MovieLens* dataset contains different sizes for testing and analysis purposes. The number of present values in the dataset is what distinguishes the various real datasets, since using different sizes give the most detailed investigation of the MRBIG algorithm's effectiveness.

TABLE 9. Real *MovieLens* dataset information.

Name	No. of Users	No. of Movies
$100k$	$1,000$	$1,700$
$1M$	$6,040$	$3,706$
$10M$	$71,000$	$11,000$
$20M$	$138,000$	$26,000$

The experiments were conducted in computing nodes equipped with the *Intel Xeon* E5-2695 v4 @ 2.10GHz CPU and 32GB assigned RAM, running Linux *Ubuntu* 16.04 LTS. For our parallel computing purposes, we have used four nodes each equipped with the same configurations.

Based on volume, the number of computing clusters can be easily adjusted. In this context, the number of computing cluster is believed to be optimal. Having big data requires the computing nodes to have a significant amount of RAM available for the algorithm. This will help to be able to initialize and process the necessary components such as the bitmap index table.

The MRBIG parallel computing approach runs through three slave computation nodes in addition to a master node as a hub point for the others. The single machine code (Algorithm 1) runs on the master node with the configurations as mentioned earlier, and the MRBIG processing (Algorithm 2) takes place on *Apache Spark* configured clusters pre-built for *Hadoop* 2.7.

A. DATA INFORMATION AND PREPARATION

The *MovieLens* datasets include a range of movies with submitted ratings from a set of users. This dataset can be used to build a movie recommender system that can dynamically evaluate impactful movies. Sizes of the dataset can be seen in Table 9 in which the numbers depict the frequency of present values in each dataset. The smallest dataset contains $100k$ values and the $20M$ version, which is one of the largest real datasets, contains over $138,000$ users and $27,000$ movies.

To provide more in-depth analysis of MRBIG performance, diverse groups of synthetic datasets have been used. Generation of the synthetic datasets' involved various mathematical and statistical parameters to thoroughly test the MRBIG algorithm.

1) EVALUATION AND CREATION

Other than the size of the data, there are several other parameters of data that have been considered for analyzing the algorithms, including mean and standard deviation of the ratings, incompleteness rate, etc. The artificially generated datasets have been crafted carefully involving six different manually-defined parameters to preserve the accuracy of the experiments. These components include the missing rate, average

TABLE 10. Artificial dataset information.

No. of Users	No. of Movies	No. of Users	No. of Movies
500	3,500	6,000	500
1,000	3,500	6,000	1,500
2,000	3,500	6,000	2,500
3,000	3,500	6,000	3,500
4,000	3,500	6,000	4,500
5,000	3,500	6,000	5,500
7,000	3,500	6,000	6,500
8,000	3,500		

value for inserted numbers, and standard deviation threshold. Additionally, missing rate standard deviation, the standard deviation for each dimension which depicts the values discrepancy for each dimension are among the other parameters for the synthetic data.

Lastly, the standard deviation for the standard deviation for each dimension is the last parameter that assigns the difference of deviation between different items for dimensions deviation. For the real datasets, not all of the mentioned parameters are provided as it is published as a real-world data and such information is not available. By defining the parameters mentioned above, we can procure the most productive information to observe the efficiency of the MRBIG algorithm and provide a way to elicit the most accurate experiments.

2) PRE-FORMATTING AND PREPARATION

To make the different datasets uniform and compatible with our desired input for the experiments, we have developed a formatting process. As discussed, the real dataset is from the publicly available *MovieLens* dataset. Each value in this dataset has three different characteristics including movie identification, user identification, and the rating submitted. A small sample of the raw input is depicted in Figure 2. A formatting process needed to empower the experiments accuracy and making the data suitable for the Hadoop MapReduce framework. As can be seen in Figure 2, user 101 has submitted four ratings for movies 01, 02, 04, 05, but not any rating for movie 03. Not having rating submission for movie 03 indicates a missing value and no row exists for this particular value. After pattern change occurrence, instead of having multiple lines for each object, one line represents user 101 and the ratings of this user as seen in Figure 2.

Further formatting process involves operations to make the data compatible with the algorithm such as removing special characters and separators, readying the data for use by the MRBIG algorithm.

There are multiple reasons for having a pre-processing method. Comparing uniform datasets with the same format to make the data analysis accurate is one the reasons. Not tracking the spent times for arranging or formatting the dataset is another reason which helps to provide valid comparisons. This process helps us to make implementation of the algorithm easier and ensure our comparison and analysis

UserID	MovieID	Rating
101	01	3
101	02	2
101	04	1
101	05	3
236	03	4
236	05	1
87	01	4
512	01	2

	01	02	03	04	05
101	3	2	-	1	3
236	-	-	4	-	1
87	4	-	-	-	-
512	2	-	-	-	-

FIGURE 2. Sample input formatting conversion for the MRBIG algorithm. Top: Raw input; Bottom: Processed input.

are logical and correct. The experiments on MRBIG and BIG algorithm using real data in Table 9 and synthetic data in Table 10 all follow the same formatting style.

The synthetic data contains different missing rates, and the experiment has been done using various combinations of objects in several dimensions. These datasets use a standard deviation for creating numbers for each object. Each object is created based on the missing rate, the deviation function and the pre-defined mean value which has been defined during the generation of datasets. Synthetic datasets are created to cover all experimental angles and completely evaluate the performance of MRBIG.

B. ALGORZITHM DEVELOPMENT AND EVALUATION

The MRBIG algorithm implementation methods can be logically and fundamentally different based on the needs. Hence, different results may not indicate consistent approaches for applying the TKD queries on incomplete data.

Our implementation process goes more in-depth to analyze the entirety of available information in each dataset. MRBIG algorithm calculates the final score for every single object in the dataset. Furthermore, MRBIG algorithm stores all objects' score values in a permanent structure which can be utilized even after the termination of the algorithm. The final scores are available to retrieve k objects after running the algorithm. This helps to acquire the TKD application result faster for later use.

By considering alternative ways of reducing the run-time, it is also possible to predefine the k value as some objects which have to be returned by MRBIG as top-k dominant values by performing some modifications to the algorithm. The approach that MRBIG follows helps to recall the results faster for later needs.

Using a predefined k value helps to execute the algorithm much faster when finding the top-k dominant values, but not in all cases. Choosing between using either a pre-defined k value for running the algorithm or implementing the algorithm using the entire scoring procedure is not clear. Therefore, there is no perfect answer or approach for using

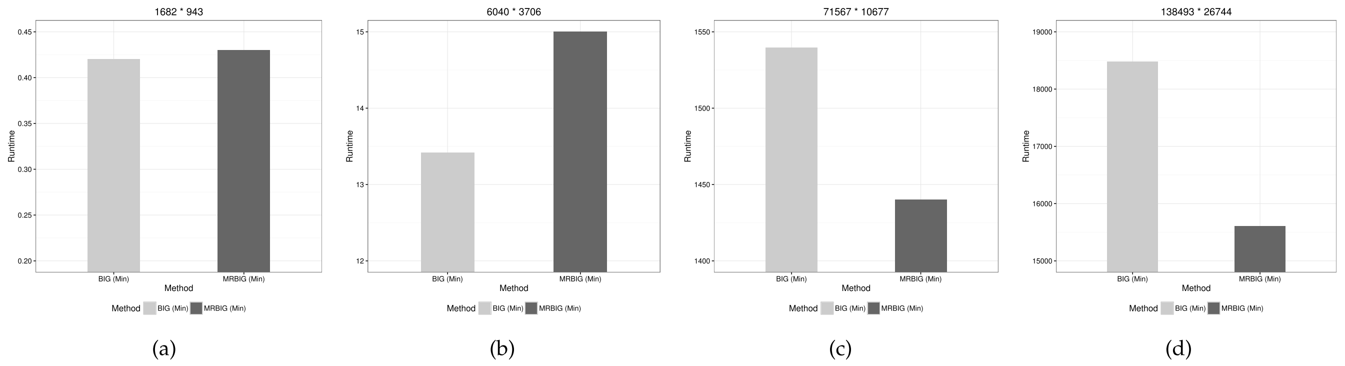


FIGURE 3. Real *MovieLens* simulation results comparing BIG and MRBIG algorithm in the uniform configured clusters and computing machines described in Section VI - *Black bars shows MRBIG, Grey bars shows BIG.* (a) 100k. (b) 1M. (c) 10M. (d) 20M.

which k calculation method. There are, however, several advantages and disadvantages associated with each method.

By predefining the k, multiple executions of MRBIG are required for every desired top-k value. In some cases, the dataset changes often. This method may be helpful as running the algorithm would be performed intermittently to find new answers for the TKD query after the data is changed. On the other hand, if the score calculation covers the whole dataset like the MRBIG approach, after the initial implementation of the algorithm, the scores are permanently present and can be recalled whenever required. This approach would help enterprise systems to retrieve answers for different top-k values in an instant just by looking up the scores and retrieving the appropriate dominant values based on the top-k results.

As shown in Figure 3, by using the real dataset, the performance of the MRBIG algorithm is dependent on the size of the dataset. When using small data sets in the MapReduce framework, the communication cost incurred for synchronizing nodes and sending and receiving data leads to the slower run-time for the algorithm. On the contrary, by increasing the size of the dataset, employing Algorithm 1 exhausts the resources and make the process exponentially slow while MRBIG shows a high performance in handling a large flow of data and speeding up the process.

MRBIG algorithm does not show a sizable difference in comparison to the Bitmap Index Guided algorithm when processing smaller sets of synthetic data. The improved performance of MRBIG algorithm has been examined by both item size and dimension size for the incomplete synthetic datasets, shown in Figure 4a and Figure 4b. However, as soon large dataset sizes are considered, the MRBIG performance becomes more useful and time efficient. Our experiment shows that this difference can be more than two times faster in speed.

As can be seen in Algorithm 2, the overall execution of the algorithm can be understood as two major parts that handle the process of finding TKD query on incomplete data. The first part includes operations that take place in the MapReduce clusters. This part includes those calculations that have

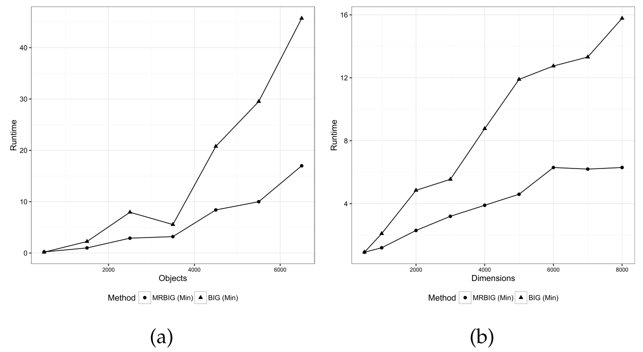


FIGURE 4. Artificial data simulation results comparing BIG and MRBIG algorithm in the uniform configured clusters and computing machines described in Section VI by testing different parameter combinations. (a) Objects. (b) Dimensions.

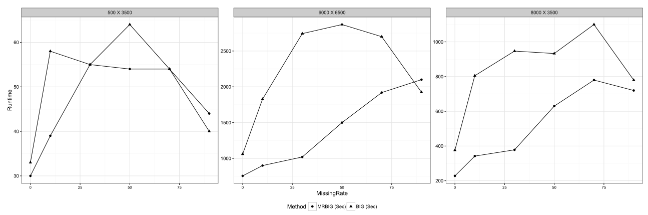


FIGURE 5. Compaing MRBIG and BIG algorithms using different missing rates.

been transferred to clusters to accelerate the process by distributing the workload among different computing nodes.

The second part, which has a minimal role throughout the algorithm, includes those operations which remain in the single machine procedure. These are not processed by either Mapper or Reducer, and the calculations are based on the conventional single-machine method.

The first component of the algorithm is where the efficiency is improved and enhanced. Based on different workloads and desired outcomes, the number of clusters can be adjusted. Adjusting the cluster by increasing or decreasing the number of computing nodes can have different effects. Depending on the volume of data, having more compute nodes might increase the runtime and worsen the

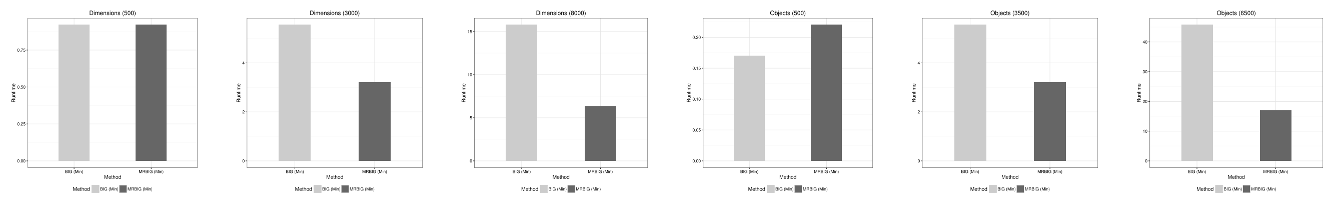


FIGURE 6. **Artificial data simulation results comparing BIG and MRBIG algorithm in the uniform configured clusters and computing machines described in Section VI -** ***Black bars shows MRBIG, Grey bars shows BIG.***

performance. Based on the synthetic data and real datasets in this context, four nodes was determined to be optimal.

Parallel computing approaches always have considerations that can affect the results negatively. The network structure is a vital part of the process. Based on the network status, queues and congestion, drops in performance are always expected. As can be observed in Figure 4b and Figure 4b, the MRBIG algorithm provides stable performance throughout the different data sizes.

In the conducted experiments, garbage collection is also included in the runtime, accounting for about 20 to 25 percent of the elapsed time. Each segment of data is assigned dynamically to different nodes by controlling the specific number of *dimensions* assigned to each mapper. The results of each mapper, including the $[P]$, $[Q]$ sets for the assigned dimension(s) would be aggregated (as depicted in Algorithm 2) and finalized in the reducer. The final steps are mostly performed on the master node.

The missing rate is another factor that can affect the performance of the MRBIG algorithm. Having a lower incompleteness rate causes a dataset to behave closer to as if it were complete. The experiments show that varying missing rates can influence the behavior of the MRBIG algorithm as well. The range of tested missing value rates varied from 0 to 90 percent as shown in Figure 5. Different sizes of synthetic data help to get a better insight of the MRBIG algorithm that has been represented in Figure 5. By keeping the deviation components to similarly evaluate the performance, multiple synthetic datasets were generated to help identify the MRBIG performance in response to differing missing rates.

The first zero percent missing rate represents complete data. In those datasets, the MRBIG algorithm still goes through the same procedure although there are no missing values. MRBIG is not designed specifically to handle complete datasets. Therefore, the performance improvement is not significant in that case as experiments show. For higher incompleteness rates that are more suitable for MRBIG evaluation, the performance enhances and the runtime decreases. The notable difference in performance helps the recommender systems retrieve desired answers more efficiently.

After reviewing the performance of MRBIG, it can be seen that applying TKD query on small incomplete datasets using MRBIG can result in large variations in performance. However, as the size of the dataset increases, the performance of the algorithm becomes more stable, and deviations in the processing runtime are much less than smaller datasets. For further evaluating the MRBIG algorithm, various synthetic datasets were generated as displayed in Table 10. The results of synthetic incomplete datasets on MRBIG algorithm can be seen in Figure 6. The synthetic data shows that the performance of applying TKD on synthetic data is highly dependent on the size of the dataset. By increasing the number of dimensions, MRBIG becomes more efficient, and this pattern is also demonstrated when increasing the number of objects (or items).

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed an algorithm to apply top-k dominating queries using MapReduce framework on incomplete big data. MapReduced Enhanced Bitmap Indexed Guided algorithm (MRBIG) is the basis of the work that develops a new way to handle large incomplete data and uses the MapReduce framework to enable parallel computing to manage the problem faster.

Throughout the paper, the single machine algorithm has been detailed, compared, and contrasted with the MRBIG algorithm. Based on the experiments, the single machine algorithm cannot be an optimal way for applying TKD queries on big files. Not being resource-efficient, process failure due to resource insufficiency, and having exponential processing time are among the major defects when it comes to finding top-k dominant values in massive incomplete data.

MRBIG algorithm is a faster way to process large incomplete datasets while carefully managing the machine resources and maintaining time efficiency simultaneously. The MRBIG algorithm provides excellent performance, and in most of the cases, it is two or more times faster than the single machine procedure. The results and experiments have also been considered in Experiment and Analysis Section to provide an overview of the efficacy and consistency of the approach.

BIG and MRBIG are both considered a resulting method from Skyband based algorithm and Upper Bound Based algorithms that use their key advantages to design better algorithms. Skyband based and Upper Bound based algorithms are reviewed in previous sections. Skyband based algorithms promote a notion of data bucketing to normalize the data, mimicked in MRBIG with the bitmap indexing method. Using bitmap indexing helps to deal with big data to reduce and speed up the pairwise comparison by representing binary strings for each value in the bitmap index table.

The contribution from Upper Bound Based algorithms consists of scoring methods, borrowed by MRBIG and BIG algorithm with a scoring evaluation of the fields independently. Having a scoring method can be considered as continued work on UBB algorithms. Scores help to retrieve desired top-*k* items in a significantly faster manner.

The MapReduce framework employs logical approaches to handle TKD query processing. The assumptions for those logical approaches promoted for creating $[P^*]$ and $[Q^*]$ can be mathematically proven using lemmas provided in the paper. Furthermore, having different structural implementations such as using a predefined top-*k* value and generating the proper score disregarding the other items scores can lead to differing performances of MRBIG. These notion and modifications can later be addressed to either improve the performance more or to provide better recommender systems based on needs.

It is worth mentioning some logical approaches that can be followed by future works using the same notion behind MRBIG:

- Partitioning the dataset using the mappers that use specific chunks of *objects* for performing TKD. In this case, we split a range of objects. Then we apply the TKD query to them. To be clear, this means splitting the data by rows by taking a particular collection of rows, with every dimension included.
- Separating the dataset using mappers that split the dataset by a different range of *dimensions*. This method uses specific dimensions from every object and tries to obtain candidate sets while applying the TKD query. As seen on Algorithm 2 (MRBIG), it can be said that the dataset would be separated by different ranges of columns, which contains some dimensions of all objects, and the TKD query performed on each of them.
- Separating larger datasets than what we have considered in this paper, by both *objects* and *dimensions*. In other words, we separate the data by rows and columns and obtain a specific range limited by objects and dimension for which we perform the TKD query and return the candidate sets to the reducer(s).

The analysis for MRBIG performance shows a significant improvement as described in Section VI. But there are a few areas that can be addressed to enhance the performance even more. These areas includes cases where the incomplete data has an extremely high incompleteness rate that resembles an empty dataset or low incompleteness rates that resemble complete datasets. It has been experimentally determined that having large incomplete data with moderate missing rates can enjoy stable performance. However, in the two mentioned cases, the MRBIG algorithm can potentially be altered to provide better performance.

It is worth mentioning that finding top-*k* dominance for incomplete data is a field that has not been fully addressed. With emerging big data, MRBIG can be utilized well to design recommender systems and provide an approximately real-time solution to TKD query processing.

ACKNOWLEDGMENTS

The authors would like to thank the UNLV Howard R. Hughes College of Engineering, and especially the Department of Computer Science, for providing the opportunity for this research. We also give a big thanks to the Big Data Hub - ILAB for equipment and assistance to make this research possible.

REFERENCES

[1] Y. Wang, X. Li, X. Li, and Y. Wang, "A survey of queries over uncertain data," *Knowl. Inf. Syst.*, vol. 37, no. 3, pp. 485–530, 2013.

[2] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, "Skyline query processing for incomplete data," in *Proc. IEEE 24th Int. Conf. Data Eng.*, Washington, DC, USA, Apr. 2008, pp. 556–565.

[3] D. Papadias, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.

[4] X. Miao, Y. Gao, B. Zheng, G. Chen, and H. Cui, "Top-*k* dominating queries on incomplete data," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, vol. 28. no. 1, pp. 1500–1501.

[5] X. Lian and L. Chen, "Probabilistic top-*k* dominating queries in uncertain databases," *Inf. Sci.*, vol. 226, pp. 23–46, Mar. 2013.

[6] S. Ge, L. H. U, N. Mamoulis, and D. W. L. Cheung, "Dominance relationship analysis with budget constraints," *Knowl. Inf. Syst.*, vol. 42, no. 2, pp. 409–440, 2015.

[7] M. L. Yiu and N. Mamoulis, "Multi-dimensional top-k dominating queries," *VLDB J.*, vol. 18, no. 3, pp. 695–718, 2009.

[8] N. Mamoulis, K. H. Cheng, M. L. Yiu, and D. W. Cheung, "Efficient aggregation of ranked inputs," in *Proc.-Int. Conf. Data Eng.*, 2006, p. 72.

[9] X. Lian and L. Chen, "Top-k dominating queries in uncertain databases," in *Proc. 12th Int. Conf. Extending Database Technol., Adv. Database Technol.*, New York, NY, USA, 2009, pp. 660–671.

[10] M. Hua, J. Pei, W. Zhang, and X. Lin, "Efficiently answering probabilistic threshold top-k queries on uncertain data," *Proc.-Int. Conf. Data Eng.*, Apr. 2008, pp. 1403–1405.

[11] X. Han, J. Li, and H. Gao, "TDEP: Efficiently processing top-k dominating query on massive data," *Knowl. Inf. Syst.*, vol. 43, no. 3, pp. 689–718, 2015.

[12] E. Tiakas and G. Valkanas, "Metric-based top-k dominating queries," in *Proc. 17th Int.-Nat. Conf. Extending Database Technol. (EDBT)*, 2016, pp. 415–426.

[13] B. J. Santoso and G.-M. Chiu, "Close dominance graph: An efficient framework for answering continuous top-*k* dominating queries," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 8, pp. 1853–1865, Aug. 2014.

[14] C. Lofi, K. El Maarry, and W.-T. Balke, "Skyline queries in crowd-enabled databases," in *Proc. 16th Int. Conf. Extending Database Technol.*, New York, NY, USA, 2013, pp. 465–476.

[15] X. Han, X. Liu, J. Li, and H. Gao, "TKAP: Efficiently processing top-k query on massive data by adaptive pruning," *Knowl. Inf. Syst.*, vol. 47, no. 2, pp. 301–328, 2016.

[16] K. Mouratidis, S. Bakiras, and D. Papadias, "Continuous monitoring of top-k queries over sliding windows," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2006, p. 635.

[17] J. Zhang, F. Zhong, and Z. Yang, "Efficient approach to top-k dominating queries on service selection," in *Proc. 6th Joint IFIP Wireless Mobile Netw. Conf. (WMNC)*, Apr. 2013, pp. 1–8, 2013.

[18] H. T. H. Nguyen and J. Cao, *Top-k Dominance Range-Based Uncertain Queries*. Cham, Switzerland: Springer, 2016, pp. 191–203.

[19] P. Haghani, S. Michel, and K. Aberer, "Evaluating top-k queries over incomplete data streams," in *Proc. Cikm*, 2009, pp. 877–886.

[20] M. A. Soliman, I. F. Ilyas, and S. Ben-David, "Supporting ranking queries on uncertain and incomplete data," *VLDB J.*, vol. 19, no. 4, pp. 477–501, Aug. 2010.

[21] S. Razniewski and W. Nutt, "Completeness of queries over incomplete databases," *Proc. VLDB Endow*, vol. 4, no. 11, pp. 749–760, 2011.

[22] F. Bu, Z. Chen, Q. Zhang, and X. Wang, "Incomplete big data clustering algorithm using feature selection and partial distance," in *Proc. 5th Int. Conf. Digit. Home*, 2014, pp. 263–266.

[23] T. Imieliński and W. Lipski, Jr., "Incomplete information in relational databases," *J. ACM*, vol. 31, no. 4, pp. 761–791, Sep. 1984.

[24] W. Cheng, X. Jin, J. T. Sun, X. Lin, X. Zhang, and W. Wang, "Searching dimension incomplete databases," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 3, pp. 725–738, Mar. 2014.

[25] L. Antova, C. Koch, and D. Olteanu, "From complete to incomplete information and back," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA, 2007, pp. 713–724.

[26] L. Libkin, "Incomplete data: What went wrong, and how to fix it," in *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, New York, NY, USA, 2014, pp. 1–13.

[27] K. Wu, A. Shoshani, and K. Stockinger, "Analyses of multi-level and multi-component compressed bitmap indexes," *ACM Trans. Database Syst.*, vol. 35, no. 1, pp. 2:1–2:52, Feb. 2008.

[28] Z. Chen, Y. Wen, W. Zheng, J. Chang, G. Peng, and Y. Wu, "A survey of bitmap index-compression algorithms for big data," *Tsinghua Sci. Technol.*, vol. 20, no. 1, pp. 100–115, 2015.

[29] K. Wu, E. J. Otoo, and A. Shoshani, "Compressing bitmap indexes for faster search operations," in *Proc. 14th Int. Conf. Sci. Statist. Database Manage.*, 2002, pp. 99–108.

[30] G. Manogaran and D. Lopez, "Disease surveillance system for big climate data processing and dengue transmission," *Int. J. Ambient Comput. Intell.*, vol. 8, no. 2, pp. 88–105, Apr. 2017. [Online]. Available: https://doi.org/10.4018/IJACI.2017040106

[31] S. Kamal, S. H. Ripon, N. Dey, A. S. Ashour, and V. Santhi, "A mapreduce approach to diminish imbalance parameters for big deoxyribonucleic acid dataset," *Comput. Methods Prog. Biomed.*, vol. 131, pp. 191–206, Jul. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.cmpb.2016.04.005

[32] M. S. Kamal, S. Parvin, A. S. Ashour, F. Shi, and N. Dey, "De-bruijn graph with mapreduce framework towards metagenomic data classification," *Int. J. Inf. Technol.*, vol. 9, no. 1, pp. 59–75, Mar. 2017. [Online]. Available: https://doi.org/10.1007/s41870-017-0005-z

[33] H. Matallah, G. Belalem, and K. Bouamrane, "Towards a new model of storage and access to data in big data and cloud computing," *Int. J. Ambient Comput. Intell.*, vol. 8, pp. 31–44, Oct. 2017.

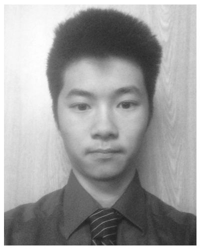
[34] M. S. Kamal, S. F. Nimmy, M. I. Hossain, N. Dey, A. S. Ashour, and V. Santhi, "Exsep: An exon separation process using neural skyline filter," in *Proc. Int. Conf. Electr., Electron., Optim. Technol. (ICEEOT)*, Mar. 2016, pp. 48–53.

[35] Y. Gao, X. Miao, H. Cui, G. Chen, and Q. Li, "Processing k-skyband, constrained skyline, and group-by skyline queries on incomplete data," *Expert Syst. Appl.*, vol. 41, no. 10, pp. 4959–4974, 2014.

PAYAM EZATPOOR is currently pursuing the master's degree in computer science with the Department of Computer Science, Howard R. Hughes College of Engineering, University of Nevada, Las Vegas. His research interests include big data analystics and data processing.

JUSTIN ZHAN was a Faculty Member with North Carolina A&T State University, Carnegie Mellon University, and the National Center for the Protection of Financial Infrastructure, South Dakota State. He is currently a Professor with the Department of Computer Science, College of Engineering, University of Nevada, Las Vegas. His research interests include big data, information assurance, social computing, and health science.

JIMMY MING-TAI WU received the Ph.D. degree from the Department of Computer Science and Engineering, National Sun Yat-sen University, Taiwan. He is currently a Post-Doctoral Research Fellow with the Department of Computer Science, University Nevada, Las Vegas, NV, USA. His research interests include data mining, fuzzy theory, evolutionary computation, and cloud computing.



CARTER CHIU is currently pursuing the Ph.D. degree with the Department of Computer Science, University of Nevada, Las Vegas. He is a member of the Big Data Hub, University of Nevada, Las Vegas. His research interests include deep learning and big data analytics.