The Journal of Supercomputing (2022) 78:3976–3997
https://doi.org/10.1007/s11227-021-04005-x

Top-*k* dominating queries on incomplete large dataset

Jimmy Ming-Tai Wu[1] · Min Wei[1] · Mu-En Wu[2] · Shahab Tayeb[3]

Accepted: 17 July 2021 / Published online: 17 August 2021

Abstract

Top-*k* dominating (TKD) query is one of the methods to find the interesting objects by returning the *k* objects that dominate other objects in a given dataset. Incomplete datasets have missing values in uncertain dimensions, so it is difficult to obtain useful information with traditional data mining methods on complete data. BitMap Index Guided Algorithm (BIG) is a good choice for solving this problem. However, it is even harder to find top-*k* dominance objects on incomplete big data. When the dataset is too large, the requirements for the feasibility and performance of the algorithm will become very high. In this paper, we proposed an algorithm to apply MapReduce on the whole process with a pruning strategy, called Efficient Hadoop BitMap Index Guided Algorithm (EHBIG). This algorithm can realize TKD query on incomplete datasets through BitMap Index and use MapReduce architecture to make TKD query possible on large datasets. By using the pruning strategy, the runtime and memory usage are greatly reduced. What's more, we also proposed an improved version of EHBIG (denoted as IEHBIG) which optimizes the whole algorithm flow. Our in-depth work in this article culminates with some experimental results that clearly show that our proposed algorithm can perform well on TKD query in an incomplete large dataset and shows great performance in a Hadoop computing cluster.

Keywords Top-textitk dominating query · Incomplete data · Dominance relationship · Big data framework · MapReduce

✉ Mu-En Wu
mnasia1@gmail.com

[1] College of Computer Science and Engineering, Shandong University of Science and Technology, Qindao, China

[2] Department of Information and Finance Management, National Taipei University of Technology, Taipei, Taiwan

[3] Department of Electrical and Computer Engineering, California State University, Fresno, CA, USA

1 Introduction

In a given set S with d-dimensional objects, the most valuable and important elements can be found called top-k objects. Generally, we think an object o dominates other objects in S if it is one of the top-k objects. The domination relationship between two objects is defined as follows: An object o dominates another object o‘ , only if the values in all dimensions of o are no worse than those of o‘ and are better than o‘ in at least one dimension. For example, there are two objects $o = (1, 2, 1, 3)$ and $o\} = (2, 2, 3, 3)$ with four dimensions, it can be considered that o‘ is dominated by o, because the values of o are less than o‘ in the first and the third dimension, and the values of both are equal in the second and the fourth dimension. (In this study, we conclude that the smaller the value, the better.) According to the definition of dominance relationship, we can easily find the top-k dominating objects in a dataset. By analyzing and processing historical data to obtain top-k objects, we can use this method to build some recommendation systems. For example, if we want to establish a movie recommendation system, we can dig for top-k movies according to the audience's rating of the movie. If one movie dominates another movie, we can think that it is better. Unfortunately, data in real life is often incomplete especially the data with multiple dimensions, such as movie rating. Audiences have seen some movies and rated them, but no audience has seen all of them, and not all of them are rated by the same audience. Therefore, there are always missing values in some dimensions in the real data. For instance, a movie $o = (-, 2, 3, -)$, that means the second and the third audiences have seen and rated it, but the first and the fourth audiences have not given it a rate, hence, there are two missing values in the first and the fourth dimension. In a case of movie rating, take the typical dataset *Movielens* from a movie recommendation system (http://www.imdb.com/) as an example, some movie and the rate from audiences are shown in Table 1. Obviously, due to the presence of missing values, the definition of the dominant relationship in incomplete data is different from that in complete data. The presence of missing values makes it impossible for the two films to be compared in some dimensions, which also makes top-k dominating queries more difficult.

In this paper, based on the previous work on top-k dominating query, we followed the original definitions and the representation in the algorithm. In incomplete data, two objects may not be able to be compared in some dimensions because of missing value such as $a_1 = (-, 5, -, 3)$ and $a_2 = (-, 3, 2, 1)$. Thus, we only compare the dimensions in which both objects are observed. In general, an object o' is dominated by another object o, denoted as $o \prec o'$, if the two objects fit the definition of

Table 1 Example of a movie recommendation system

ID	Movie Name	a1	a2	a3	a4	a5
m1	Schindler's List (1993)	–	–	3	4	2
m2	The Godfather (1972)	5	2	1	–	–
m3	The Silence of Lambs (1991)	–	3	4	5	3
m4	Star Wars (1977)	3	1	5	3	4

domination relationship, as we have mentioned before. For example, we can observe the values in the second and the fourth dimension from a1 and observe the values in the second, third, and fourth dimensions from a_2, so we can only compare the value in the second and the fourth dimension. What's more, $a_1.[2] > a_2.[2]$, $a_1.[4] > a_2.[4]$, thus we consider $a_2 \prec a_1$. In TKD query, the basis to rank the objects is the score of each object, so it is important to formulate a score function. According to the previous work, the score of an object $score(o)$ is defined as the number of objects in S those are dominated by o, that is, $score(o) = \left\{o' \in S | o \prec o'\right\}$. Take the rating information in Table 1 as an example, we can get $m2 \prec m1$, $m2 \prec m3$, therefore, $score(m2) = |\left\{m_i \in S | m_2 \prec m_i\right\}| = |m_1, m_3| = 2$. So far, the score of an object is obtained.

Many previous works have provided feasible algorithms for TKD queries, such as Skyline algorithm, Skyband-Based algorithm, and Upper Bound Based algorithm. These algorithms take advantage of the definition of domination relationship and compare all objects in pairs to determine the dominant number of each object, thus calculating the score. However, the pairwise comparison is time-consuming, and the problem will get worse when the size of the dataset becomes larger. In order to solve this problem, Miao et al. proposed a new algorithm, BitMap Index Guided algorithm. It builds bitmap index for the original dataset, which supports new bitmap pruning and fast bit-wise manipulation to calculate scores more efficiently. Unfortunately, even though the BitMap Index Guided algorithm has greatly reduced the time complexity, it is still tough to process datasets on a single machine when the data volume is enormous. A novel algorithm, MapReduce Enhanced Bitmap Index Guided Algorithm, is proposed to deal with this issue which applies the MapReduce framework on top-k dominance queries on large incomplete datasets. Compared with the proposed algorithm, the processing time of this method in finding TKD query results is twice as fast. This proves that the MapReduce framework is very effective for dealing with this problem. We found that the MRBIG algorithm still has a lot to improve in terms of process. Theoretically, these improvements can make the algorithm more efficient and also be helpful for handling larger datasets. Therefore, we improved the algorithm on this issue, proposed the EHBIG algorithm. Furthermore, an improved version also is developed to optimize the process of the algorithm. Experiments show that our two versions of the algorithm are more efficient than the previous one. In brief, the main contributions of this paper are introduced as follows.

1. In this article, an efficient top-k query algorithm (EHBIG) is proposed to handle an extensive dataset. The designed algorithm applies the MapReduce framework to the whole TKD query process, and each time the original dataset is scanned, an object score is obtained. The number of scanning datasets is reduced to the greatest extent, thus improving the feasibility of the algorithm in practical application.
2. The designed EHBIG algorithm utilizes a new pruning strategy to prune the unpromising objects, thus reducing the unnecessary computations. Moreover, the developed EHBIG designs a parallel way to perform the pruning strategy, which is more efficient in handling large-scale datasets.

3. The developed EHBIG can also reduce the number of database scans since it could catch all of the information of an object to calculate its score, which is very efficient in the large-scale environment by reducing the seeking time of the database.
4. An improved version of the algorithm (IEHBIG) is developed to optimize the algorithm flow, which applies the MapReduce framework to the whole algorithm process. The algorithm can score all objects by scanning the dataset once, which minimizes the impact of the limitations of Hadoop's MapReduce architecture on the algorithm results.

The rest of this paper is organized as follows. Section 2 reviews related work. The problem statement is described in Sect. 3. Section 4 introduces our algorithm idea and algorithm flow in detail. The introduction and results of the experiment are shown in Sect. 5. Finally, Sect. 6 includes the summary of this paper and the prospect of future work.

2 Related work

In this section, we review the related work of some related concepts, such as top-*k* dominance, incomplete data, bitmap indexing, and MapReduce framework.

2.1 Top-*k* dominance

The TKD query finds the best value in the dataset according to the predefined good criteria. The original TKD algorithm was designed to process complete datasets, so this aspect of the work is more focused on complete datasets. Papadias et al. [1] first introduce the concept of top-*k* dominating query which is considered as a variation of skyline queries, and they proposed an algorithm for processing TKD queries based on skyline to process complete dataset indexed by an R-tree. Yiu and Mamoulis reviewed the works and the challenge of TKD queries on complete multidimensional datasets [2]. Ge et al. [3] proposed a budget constrained optimization query algorithm which can help to increase the profitability of products. Mamoulis et al. [4] developed a new algorithm that refines the object accesses throughout top-*k* processing. E. Tiakas and G. Valkanas studies processing top-*k* dominating queries over dynamic attribute vectors where finding the distances depends on the defined metrics between objects [5]. Miao et al. follow the combination of top-*k* and Skyline queries that led to top-*k* dominating query [6]. Furthermore, H Zhu et al. provide a systematic study of TKD queries on skyline groups and validate our algorithms with extensive empirical results on synthetic and real-world data [7]. Tiwari et al. [8] apply four ensemble regressor models Gradient-Boosting Regressor, Extra-Trees Regressor, Ada-Boost Regressor, and Random-Forest Regressor to analyze the association between dependent and independent variables of a COVID-19 prediction in India. There's a lot of other research in this field [9–12].

2.2 Incomplete data

The TKD query on the incomplete dataset is completely different from that on the complete dataset. Due to the existence of missing data, the traditional algorithm cannot carry out calculation. Therefore, we must consider the impact of missing values on the dominant relationship on some dimensions. Wang et al. [13] provided more details and definitions to give a better understanding of top-k and Skyline queries based on uncertain data. Khalefa et al. [14] propose TKD query processing using the ISkyline algorithm that is especially suitable for incomplete data. Lian and Chen [15] also consider the uncertain data, so they proposed a new algorithm, called Probabilistic Top-k Dominating (PTD) to realize the top-k dominating query in uncertain data. Then, Lian and Chen [16] propose an effective pruning approach to reduce the PTD search space and present an efficient query procedure to answer PTD queries. Han et al. [17] worked on TKD queries on massive data, accomplished by sorting and listing values and making the process faster than other methods. Zhang et al. [18] proposed incomplete models and estimate probability density functions of missing values on independent, correlated, and anti-correlated distributions, respectively. Chen et al. [19] proposed a novel algorithm, called High-Utility-Occupancy Pattern Mining in Uncertain databases (UHUOPM) to process the missing values in utility mining. Some studies use feature engineering and feature extraction methods to deal with missing data in datasets. Sefidian et al. proposed a novel technique to impute missing data, which employs a new version of Fuzzy c-Means clustering algorithm which benefits from advantages of Grey Relational Grade over Minkowski-like similarity measures [20]. Biessmann et al. release DataWig [21], a robust and scalable approach for missing value imputation that can be applied to tables with heterogeneous data types, including unstructured text.

2.3 BitMap indexing

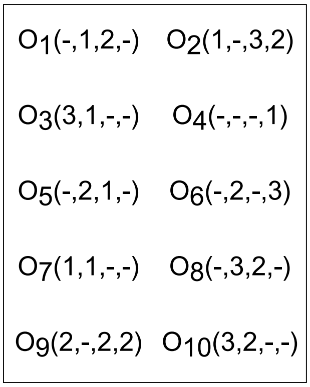
Bitmap index refers to bitmap index technology, which is a special database index technology. Bitmap index USES bitmap array (or bitmap, bit-set, bitString, bit-Vector) for storage and calculation. A bitmap index is a special database index that uses bitmaps and is created primarily for a large number of columns with the same value. Therefore, bitmap indexing technology is suitable for solving TKD query, because it can use bit-vector operation to replace the traditional algorithm pair comparison, thus improving the efficiency of the algorithm. Miao et al. [6] proposed a new algorithm, called Bitmap Index Guided (BIG) algorithm as one of the first attempts to solve this issue. Wu et al. [22] considered the effects of compression on multi-component and multi-level compressed bitmap indexes. There are several methods for compressing bitmap indexes, such as BBC, CONCISE, and WAH methods that is proposed by Chen et al. [23], and it is the first attempt to apply traditional algorithms to big data environment. For instance, to make the compression more stable, Wu et al. [24] has proposed their word-aligned hybrid code (WAH) as a compressing method for bitmap indexes that leads to improved performance.

2.4 MapReduce framework

MapReduce is a programming model and an associated implementation for processing and generating large datasets that are amenable to a broad variety of real-world tasks. The MapReduce framework can divide the input data into several parts and deliver them to different nodes for processing, thus solving the large amount of data that cannot be handled by the single machine algorithm. Manogaran and Lopez [25] proposed a MapReduce disease surveillance system for analyzing the correlation between climate data and Dengue fever transmission in real-time. Kamal et al. [26] suggested a k-nearest neighbor classifier for imbalanced 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. [27] applied 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. [28] proposed enhancements to the storage of metadata in the Hadoop Distributed File System for improved scalability, demonstrating the continuing value of MapReduce in modern applications. Ezatpoor et al. [29] proposed MapReduce Enhanced Bitmap Index Guided Algorithm (MRBIG), which uses the MapReduce framework to enhance the performance of applying top-*k* dominance queries on large incomplete datasets.

3 Problem statement

This section will explain the dominance relationship and formalize the issue of TKD query on incomplete data. First, Fig. 1 provides a sample incomplete dataset D with d dimensions. We are going to represent the value of each object in terms of vectors with d bits. The "-" represents the missing value of the object in this dimension. For example, $o_1 = (-, 1, 2, -)$, that means o_1 has two missing values in d_1 and d_4. Note that, the smaller evaluating value of one movie means the better ranking from a person. According to the dataset shown in Fig. 1, we can format the data into Table 2.



O_1(-,1,2,-)	O_2(1,-,3,2)
O_3(3,1,-,-)	O_4(-,-,-,1)
O_5(-,2,1,-)	O_6(-,2,-,3)
O_7(1,1,-,-)	O_8(-,3,2,-)
O_9(2,-,2,2)	O_{10}(3,2,-,-)

Fig. 1 An sample incomplete dataset

Table 2 A sample incomplete dataset table

object	d_1	d_2	d_3	d_4
o_1	–	1	2	–
o_2	1	–	3	2
o_3	3	1	–	–
o_4	–	-	–	1
o_5	–	2	1	–
o_6	–	2	–	3
o_7	1	1	–	–
o_8	–	3	2	–
o_9	2	–	2	2
o_{10}	3	3	–	–

As we have introduced before, the TKD query is implemented based on pairings of objects and determination of the dominance relationship. Given two objects o_1 and o_2 , if o_1 dominates o_2, it is denoted as $o_1 \prec o_2$.The conditions for determining the dominance relationship between two objects are as follows: In each dimension where values are observed both at o_1 and o_2, (1) the value of o_1 cannot be larger than o_2; (2) in at least one dimension, the value of o_1 is strictly less than the value of o_2. For example, for $o_1 = (-, 1, 2, -)$ and $o_2 = (1, -, 3, 2)$ in Fig. 1, we can say $o_1 \prec o_2$ because the values are observed on d_3 both at the two objects and the value of o_1 is less than o_2.

After determining the dominance relationship between two objects, we need a method to compare all objects, so as to get the top-k dominant objects. Therefore, we need a scoring mechanism to calculate the score of each object. For an object o in a given incomplete dataset D, the score of o ,denoted as $score(o)$, is the number of objects it dominates in that dataset. The set of objects dominated by o is $R(o)$, $R(o) = \{o\} \in D|o \prec o\}\}$, then the score of o is expressed as $score(o) = |R(o)|$. For instance, $R(o_1) = \left\{o_2, o_8, o_{10}\right\}$, so $score(o_1) = |R(o_1)| = 3$.

4 TKD query on large incomplete dataset

As we have reviewed in the previous sections, traditional data mining algorithms based on complete datasets are not suitable for incomplete datasets with missing values. Bitmap indexing provides the possibility of TKD queries on incomplete datasets. It is better than other methods in terms of feasibility and performance, such as Skyband-Based and UpperBound-Based Algorithm. In fact, we can get more information from bitmap index table. And the application of MapReduce framework to the whole algorithm process will greatly improve the performance of the algorithm. More importantly, we need a more effective pruning strategy to reduce the computation and improve the performance of the algorithm. In this section, we will introduce the previous algorithm and our algorithm with a new pruning strategy. First, let's review the single machine algorithm, i.e., BitMap Index Guided Algorithm.

4.1 Single machine algorithm (BIG)

In order to visually observe the value of each object in each dimension, we can get a table as shown in Table 2 according to Fig. 1. Our bitmap index guided algorithm is based on a bitmap indexing table, so we need to build a bitmap indexing table. Unlike traditional bitmap index guided algorithms, we need a bitmap index table with missing values.This bitmap index table is created as follows: (1)For a dataset, we can first obtain a dimension table, such as Table 2. Then, we analyze the values of all objects that can be observed on each dimension. We use a vector v_i to represent the values of all objects that can be observed in the i-th dimension. Take the given dataset as an example, i.e., Table 2. On the first dimension, 1,2,3 can be observed with missing value. So $v_1 = \{-, 1, 2, 3\}$. Next, we will insert $|v_i|$ columns after the i-th dimension, each of which represents a value in v_i. (2)Initialize all the values in the table to 1. The bitmap index table is then updated based on the value of each object on each dimension. Starting from the left, if the value of an object is not missing on a dimension, change the value of the column corresponding to v_i and all subsequent columns to 0. If the value of an object on a dimension is a missing value, the columns corresponding to that dimension are not modified. According to the above steps, we can get the bitmap indexing table corresponding to Table 2 as shown in Table 3.

After creating the bitmap index table, we can get some information from the table. We previously described the dominance relationship between two objects and the scoring mechanism for each object in the TKD query on an incomplete dataset. From this, we can get that the score of an object is the number of objects it dominates. However, based on the conditions for judging the dominant relationship between the two objects, we can conclude that the set of objects dominated by an object o can be divided into two parts. The first subset contains all objects whose values on any dimension are greater than o, called $\Gamma(o)$. And the other one collects the objects whose value on one or more dimensions is equal to o, called $\Lambda(o)$. Bitmap indexing table clearly reflects the values of different objects in the same dimension, so we can easily analyze the dominant relationship between objects.

Table 3 A sample BitMap indexing table

Items	d_1	–	1	2	3	d_2	–	1	2	3	d_3	–	1	2	3	d_4	–	1	2	3
o_1	–	1	1	1	1	1	1	0	0	0	2	1	1	0	0	–	1	1	1	1
o_2	1	1	0	0	0	–	1	1	1	1	3	1	1	1	0	2	1	1	0	0
o_3	3	1	1	1	0	1	1	0	0	0	–	1	1	1	1	–	1	1	1	1
o_4	–	1	1	1	1	–	1	1	1	1	–	1	1	1	1	1	1	0	0	0
o_5	–	1	1	1	1	2	1	1	0	0	1	1	0	0	0	–	1	1	1	1
o_6	–	1	1	1	1	2	1	1	0	0	–	1	1	1	1	3	1	1	1	0
o_7	1	1	0	0	0	1	1	0	0	0	–	1	1	1	1	–	1	1	1	1
o_8	–	1	1	1	1	3	1	1	1	0	2	1	1	0	0	–	1	1	1	1
o_9	2	1	1	0	0	–	1	1	1	1	2	1	1	0	0	2	1	1	0	0
o_{10}	3	1	1	1	0	3	1	1	1	0	–	1	1	1	1	–	1	1	1	1

First, we introduce the definition of four sets: Q , P , $\Phi(o)$ and $nonD(o)$. Set Q of an object o contains the objects whose values on all dimensions are not less than o or are missing values. Set P of an object contains the objects whose value on all dimensions are larger than o or are missing values. All objects in set $\Phi(o)$ cannot be compared with o. Set $nonD(o)$ of an object o represents the set of all objects that are not dominated by o. The containment relationships for these four sets are shown in Fig. 2. By analyzing the bitmap indexing table, we can conclude that for an object o, the column corresponding to the first 0 appearing from the left in each dimension can be represented as a vector.This vector represents the set P of o in each dimension. A 1 in a vector represents an object whose value in that dimension is less than o or a missing value. For example, for o_1, $[P^2] = 0101110111$, $[P^3] = 0111011001$, and$[P^1]$ and $[P^4]$ are 1111111111 because the missing values. Similarly, the vector represented by the left column of the column for which the first 0 appears in each dimension represents the set Q of o. A 1 in a vector represents an object whose value in that dimension is no larger than o or a missing value. As in the previous example, for o_1, $[Q_2] = 1111111111$, $[Q_3] = 1111011111$, and $[Q_1]$ and $[Q_4]$ are 1111111111 because the missing values. So the set Q and P of o are the intersection of all the vectors in all the dimensions, expressed as:

$$Q = \bigcap_{i=1}^{d} Q_i$$

$$P = \bigcap_{i=1}^{d} P_i$$

According to this formula, we can get the P and Q for o_1. $[P] = 0101010001$ and $[Q] = 1111011111$. That is to say, the $P = \left\{o_2, o_4, o_6, o_{10}\right\}$ and the $Q = \left\{o_1, o_2, o_3, o_4, o_6, o_7, o_8, o_9, o_{10}\right\}$. After we get Q and P, we can rescan the dataset to look for $\Phi(o)$ and $nond(o)$ .According to the definition of each set, we can get the following relation:

$$\Gamma(o) = P - \Phi(o)$$

$$\Lambda(o) = Q - P - nonD(o)$$

From this, we can calculate the score of each object according to the bitmap indexing table. The BIG algorithm loops through all objects, getting the Q and P of one object at a time, then rescanns the dataset, finds the $\Phi(o)$ and $nonD(o)$ of the object, and calculates the score of the object. The pseudo-code of the whole algorithm flow is shown in Algorithm 1.

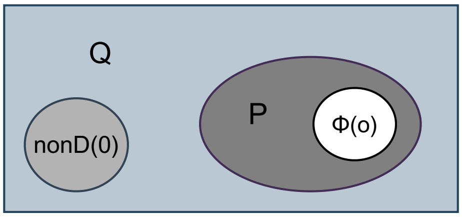


Fig. 2 The containment relationships

Algorithm 1 Single Machine (BIG) Algorithm Pseudo Code

```
Require: the original dataset and the parameter $k$.
Ensure: the top-$k$ objects.
 1: Score Calculation for item $m_i$
 2: Create $[P]$ and $[Q]$
 3: for each item in $m_i$ do
 4:     for each column in $v_i$ do
 5:         temp$\leftarrow$ Index(temp)
 6:         $[P^i] \leftarrow$ append $(\sum_{j=1}^{n}[m_j, ind])$
 7:         $[Q^i] \leftarrow$ append $(\sum_{j=1}^{n}[m_j, ind+1])$
 8:         $P^* \leftarrow \bigcap_{i=1}^{d} P^i$
 9:         $Q^* \leftarrow \bigcap_{i=1}^{d} Q^i$
10:         $nonD(m_i)$
11:         $\Phi(m_i)$
12:     end for
13:     $\alpha \leftarrow Q^* - P^* - nonD$
14:     $\beta \leftarrow P^* - \Phi$
15:     $score \leftarrow \alpha + \beta$
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-$k$
```

4.2 MRBIG

In this section, we will briefly introduce the idea and flow of MRBIG algorithm. This algorithm is to realize the TKD query in an incomplete large dataset. The first problem to be solved is that the dataset is too large for the single machine algorithm to handle. As far as we know, the MapReduce architecture of Hadoop enables distributed processing of input data in distributed file system HDFS, thereby reducing the stress of storing and processing data internally.

In the BIG algorithm, the process of calculating the score of an object is divided into three steps. The first step is to scan the bitmap indexing table to get the Q and P of an object, the second step is to scan the dataset to find the $\Phi(o)$ and $nonD(o)$ of the object, and the last step is to calculate the score of the object according to the four sets. MRBIG algorithm applies the MapReduce framework to the first step of the BIG algorithm for distributed reading of data and calculation of Q and P. The MapReduce framework splits the input data by dimensions and submits them to different Mapper nodes for processing. Each node will handle several dimensions and mine the set Q and P of an object. The algorithm framework is shown in Fig. 3. And the pseudo-code is shown in Algorithm 2.

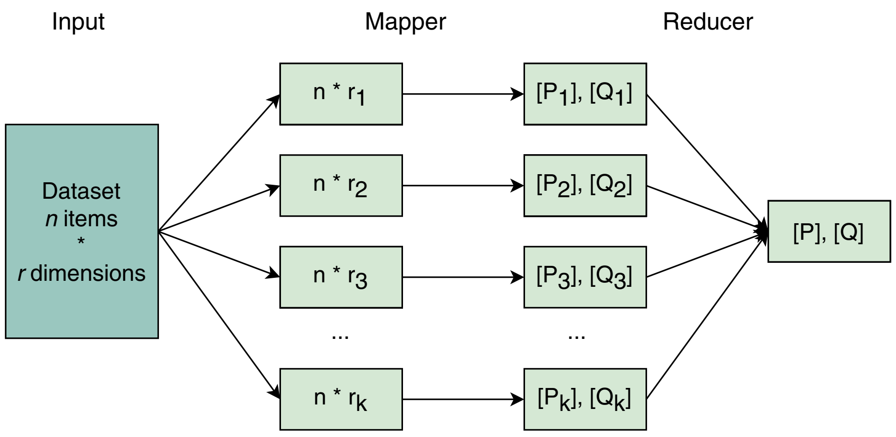


Fig. 3 The algorithm framework of MRBIG

Algorithm 2 MRBIG Algorithm Pseudo Code

```
Require: the original dataset and the parameter $k$.
Ensure: the top-$k$ objects.
1: Score Calculation for item $m_i$
2: Create $[P]$ and $[Q]$
3: for each item in $m_i$ do
4:     Mapper:
5:     Map(split$\sum_{j=1}^{r} d_j$)
6:     Bitmap($d_j$)
7:     for each dimension in $d_i$ do
8:         Create $[P_j]$ and $[Q_j], nonD_j$
9:         for each $v_{d_j}$ in $_i - throw$ do
10:            Candidate$\leftarrow$ index($if(m_i, v_{d_j} == 0)$)
11:            $[P_j] \leftarrow$ append $(\sum_{i=1}^{n}[m_i, Candidate])$
12:            $[Q_j] \leftarrow$ append $(\sum_{i=1}^{n}[m_i, Candidate + 1])$
13:        end for
14:    end for
15:    Reducer:
16:    for $\sum_{1}^{r} i$ do
17:        $P_i \cap P^*$
18:        $Q_i \cap Q^*$
19:    end for
20:    $\lambda \leftarrow Q^* - P^*$
21:    for each $i$ in $\lambda$ do
22:        $\Phi \leftarrow count(if!(\lambda_i \succ m_i))$
23:        $nonD \leftarrow \lambda_i$
24:    end for
25:    $\alpha \leftarrow \lambda - nonD$
26:    $\beta \leftarrow P^* - \Phi$
27:    $score \leftarrow \alpha + \beta$
28:    $maxscore[i] \leftarrow score$
29: end for
30: finish when length($maxscore$) = n
31: Finding top-$k$
32: sort(descending($maxscore$))
33: return top-$k$
```

4.3 EHBIG

As we mentioned before, the BIG algorithm can only get Q and P of one object in one loop, and then, it needs to rescan the dataset to get the score of one object. MRBIG algorithm utilizes the flow of the BIG algorithm and applies MapReduce to the first step. The disadvantage is also obvious; when the dataset is huge, the scanning process will be very time-consuming. After analyzing the definition of bitmap index table and each set, we improve the flow of the algorithm and propose a new pruning strategy to reduce the computation time and improve the performance of the algorithm. This section mainly introduces the process and details of our new algorithm and the working mechanism of the pruning strategy.

First of all, EHBIG improved the calculation method of $\Phi(o)$ and $nonD(o)$ sets. Instead of rescanning the dataset after obtaining Q and P, we obtained them in the iteration bitmap indexing table. As mentioned in the previous Q and P, we can also obtain these two sets by analyzing the bitmap index table and the already obtained sets. When we look at the last column of the bitmap index table, we can find that on the dimensions of an object$o$ where the value can be observed, if the value of the last column of $o'$is 1, then o' is missing on that dimension. If o' is missing on all dimensions where the value can be observed of o, then o' is one of the elements of $\Phi(o)$. So the $\Phi(o)$ can be expressed as:

$$\Phi(o) = \bigcap_{i=1}^{d} \Phi_i(o)$$

According to this formula, we can get $[\Phi_2(o1)] = 0101010010 and [\Phi_3(o1)] = 0011011001$. So the $[\Phi(o1)] = 0001010000$ That is to say, the $\Phi(o1) = \left\{o_4, o_6\right\}$.

Furthermore, according to the definitions of Q and P, we can find that for o, if $Q - P = 1$ on a dimension of o', it can prove that the value of the two objects on this dimension is equaled, so o cannot dominate o' on this dimension. This shows the $nonD_i(o) = Q_i - P_i$, so can be expressed as:

$$nonD(o) = \bigcap_{i=1}^{d} nonD_i(o)$$

According to this formula, we can get $[nonD_2(o_1)] = 1010001000$ and $[nonD_3(o_1)] = 1000000110$. So the $[nonD(o_1)] = 1000000000$. That is to say, the $nonD(o1) = \phi$. Now, we can get the score of o_1, $score(o_1) = |0110001111| = 6$.

According to the above findings, the EHBIG algorithm can get four sets of an object in a loop to calculate the score of the object. In this way, MapReduce can be applied to the whole algorithm process, eliminating the process of rescanning the dataset, thus greatly reducing the computation time. The whole framework of EHBIG is shown in Fig. 4.

In this algorithm, each object runs the MapReduce architecture once as it loops through all the objects. However, as far as we know, MapReduce architecture of Hadoop tends to have some loss of startup time and communication time. Therefore, reducing the running times of MapReduce can greatly reduce the running time of the algorithm. Next, we will introduce our efforts in pruning strategies.

We know that by the definition of the dominant relationship, given two objects o_1 and o_2, if o_1 dominates o_2, then o_2 will not dominate o_1.

So we start with the $Max_Dominance$ to represent the objects dominated by an object, which is the maximum score that the object can get. If the size of the given dataset is n, the $Max_Dominance$ of all objects is initialized to $n - 1$. The other thing is we have an array to represent the top-k dominating objects whose size is k. Set a *threshold* initialized to 0 to represent the minimum score of the current top-k dominating objects.

Then, start looping the bitmap index table. If a object o is dominated by another, the $Max_Dominance$ corresponding to o is reduced by 1, i.e.,

$$Max_Dominance(o) = n - 1 - 1$$

Taking Table 2 as an example, suppose we want to query the top-4 objects. According to the above process, in the first and second iterations, we can get that: $score(o_1) = |o_2, o_7, o_8, o_{10}| = 4$, $score(o_2) = |o_3, o_6, o_{10}| = 3$, $score(o_3) = |o_4, o_5, o_7, o_8, o_{10}| = 5$, $score(o_4) = |o_2, o_6, o_9| = 3$, $score(o_5) = |o_2, o_8, o_9, o_{10}| = 4$, $score(o_6) = |o_8, o_{10}| = 2$, $score(o_7) = |o_2, o_3, o_5, o_8, o_9, o_{10}| = 4$. By the end of this iteration, the top-4 objects are o_7, o_3, o_1, o_4 , and the *threshold* is updated as 4. In these iterations, we found that o_{10} is dominated by $o_1, o_2, o_3, o_5, o_6, o_7$. According to our proposed pruning strategy, $Max_Dominance(o_{10}) = 10 - 1 - 6 = 3 < 4$. So the score of o_{10} does not need to be calculated, thus reducing one iteration of MapReduce.

Update the *threshold* when the number of loops is larger than or equal to k. If the$Max_Dominance$ of an object is less than the *threshold*, it means that the score of the object will not exceed the minimum score of the first k objects, so the object does not need to loop. This pruning strategy can greatly reduce the running times of MapReduce, thus greatly reducing the running time of the algorithm. The running

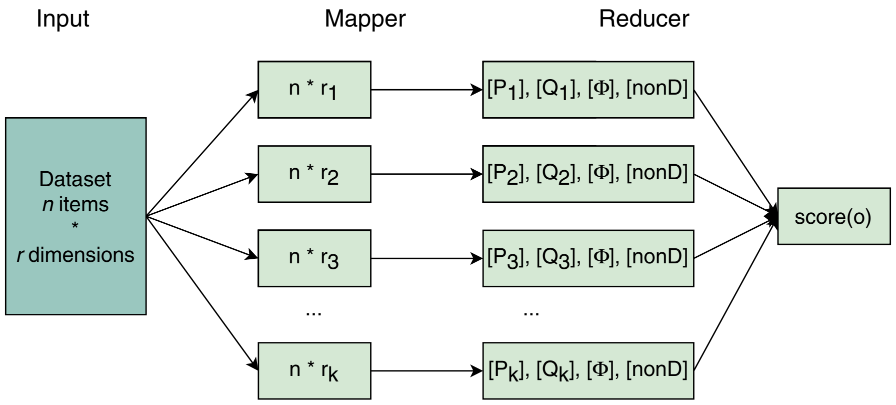


Fig. 4 The algorithm framework of EHBIG

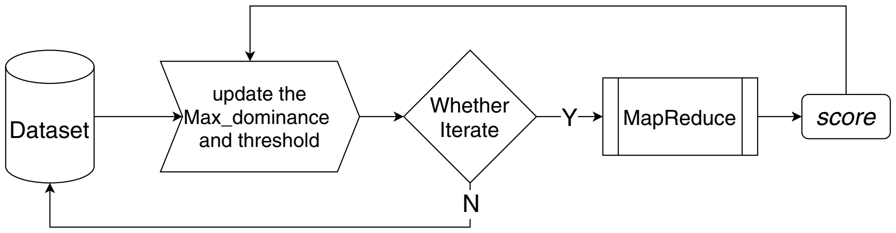


Fig. 5 The algorithm flow of EHBIG

process of the entire algorithm is shown in Fig. 5. And the pseudo-code is shown in Algorithm 3.

Algorithm 3 EHBIG Algorithm Pseudo Code

```
Require: the original dataset and the parameter $k$.
Ensure: the top-$k$ objects.
 1: Score Calculation for item $m_i$
 2: Create $[P]$ and $[Q]$
 3: for each item in $m_i$ do
 4:     $threshold \leftarrow 0$
 5:     $MaxDominance \leftarrow n-1$
 6:     Mapper:
 7:     Map(split$\sum_{j=1}^{r} d_j$)
 8:     Bitmap($d_j$)
 9:     for each dimension in $d_i$ do
10:         Create $[P_j]$ and $[Q_j], nonD_j$
11:         for each $v_{d_j}$ in $_i - throw$ do
12:             Candidate$\leftarrow$ index($if(m_i, v_{d_j} == 0)$)
13:             index $\leftarrow$ the last 0
14:             $[P_j] \leftarrow$ append $(\sum_{i=1}^{n}[m_i, Candidate])$
15:             $[Q_j] \leftarrow$ append $(\sum_{i=1}^{n}[m_i, Candidate+1])$
16:             $[\Phi_j] \leftarrow$ append $(\sum_{i=1}^{n}[m_i, index])$
17:             $[nonD_j] \leftarrow$ append $(Q_j - P_j)$
18:         end for
19:     end for
20:     Reducer:
21:     for $\sum_1^r i$ do
22:         $P_i \cap P^*$
23:         $Q_i \cap Q^*$
24:         $\Phi_i \cap \Phi^*$
25:         $nonD_i \cap nonD^*$
26:     end for
27:     $score \leftarrow |Q - nonD - \Phi$
28:     $maxscore[i] \leftarrow score$
29:     update $treshold$ and $MaxDominance$
30:     finish when $length(maxscore) = n$
31: end for
32: Finding top-$k$
33: sort(descending($maxscore$))
34: return top-$k$
```

4.4 IEHBIG

This version of the algorithm improves the algorithm flow, IEHBIG only runs MapReduce once and calculates the score of all objects at once, as shown in Fig. 6. As we all know, the startup costs of Hadoop are very large, so this version avoids the iterative operation of MapReduce, thus saving the start-up and shutdown time of Hadoop, thus saving time cost and improving algorithm efficiency to a large extent. However, the pruning strategy we proposed cannot be used in this version because it is based on the MapReduce iterative scenario and therefore cannot minimize the running time. In addition, in this version, because each node needs to scan all objects, the computing power for each node is high, that is, the memory for each node needs to be large enough. Combining time consumption and memory consumption, we could not compare the advantages of the two versions, so we included both versions in the paper so that use cases could choose which version to use according to their own requirements.

5 Experimental evaluation

This section contains the necessary experiments to verify the performance gap among the algorithms. These experiments are mainly carried out to compare the running time and required memory of each algorithm. Based on the analysis of these experimental results, we evaluated the performance of each algorithm, so as to intuitively show the advantages and disadvantages of each algorithm. In order to obtain more objective and accurate results, we considered several factors that might influence the results, including the size of the datasets, the number of records and dimensions, and the incompleteness rate of the datasets. Based on these factors, we conducted extensive research using both real datasets and synthetic datasets to obtain the following experimental results. As MapReduce architecture is made to work in a distributed environment, our experiments are also done in a distributed cluster of different machines. 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. To achieve parallelization, the experiment was run under the Hadoop 2.8.5 cluster with one master node and five data nodes. The number of nodes can be adjusted according to the size of the input data.

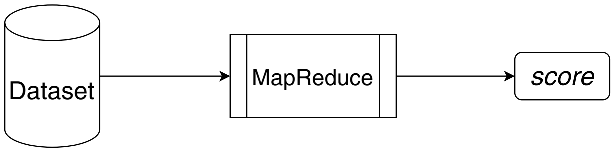


Fig. 6 The algorithm flow of IEHBIG

5.1 Data information and preparation

The real dataset we used references previous work, including *MovieLens*, *NBA* and *Zillows*, which contained different records, number of dimensions, and incompleteness rates, which made our experiment more comprehensive and objective. Besides, a synthetic dataset was also used to visually show the differences among the algorithms because its parameters can be artificially adjusted to meet our needs for various datasets. The main purpose of this paper is to apply a single machine algorithm to the big data environment. Therefore, to verify the algorithm's performance, it is essential to select datasets with the appropriate size. The storage mode of the original data is shown in Table 4. Before the algorithm starts, we first need to preprocess the data formatting, which can make the data more standardized and facilitate the algorithm to process the data so as to get more reasonable and correct results. This part of the processing will not count against the running time of the algorithm. To improve Hadoop's performance, we use HBase, a distributed, column-oriented, open-source database, to store data. HBase is a sub-project of Apache's Hadoop project, which provides bigtable-like capabilities on top of Hadoop. Using HBase, we can access the data more efficiently, thus improving the performance of the algorithm.

5.2 Algorithm development and evaluation

First of all, in order to test the advantages of EHBIG and IEHBIG algorithms in the big data environment, the first experiment is to compare the performance of each algorithm in processing different datasets. We selected four datasets of different sizes in three real datasets for the experiment, which contained 100KB, 1MB, 10MB, and 100MB, respectively. In this experiment, in order to guarantee the premise of unique variables, we try to keep the consistency of other parameters, including the number of objects, the number of dimensions, and the incompleteness rate. The result is shown in Fig. 7. By analyzing the experimental results, we can come to the following conclusion: When the dataset is small, single machine algorithms obviously have a shorter running time than other algorithms because Hadoop tends to take a long time to start. The distributed concept of MapReduce cannot be implemented because the dataset is too small, so the MapReduce architecture is not applicable when the dataset is too small to be segmented. The running time of EHBIG

Table 4 A sample original dataset

User	Movie	Rating
1	01	3
1	02	4
1	04	2
2	02	2
2	03	3
3	04	3
4	01	1
4	04	2

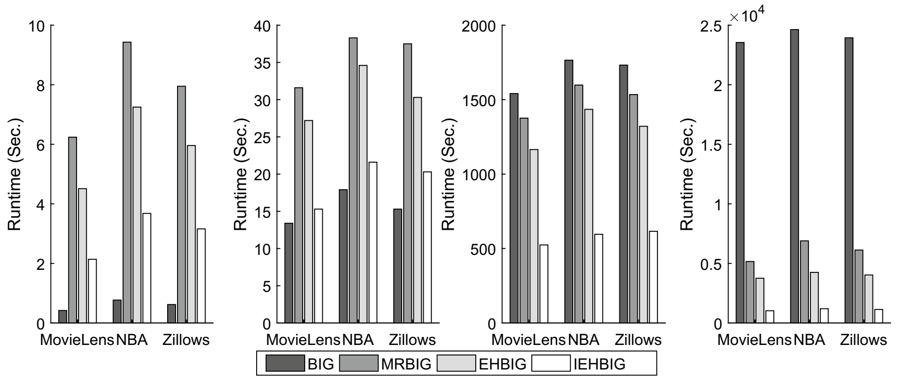


Fig. 7 The runtime of algorithms processing datasets with different sizes

algorithm is less than that of MRBIG algorithm, which indicates that the improvement of MRBIG algorithm is successful, and the use of MapReduce framework to process big data is very effective. The runtime of IEHBIG is much lower than that of EHBIG because IEHBIG only starts Hadoop once. As we mentioned earlier, the Hadoop startup process is a waste of time, and the communication cost among nodes is inevitable. However, this does not mean that IEHBIG is superior to EHBIG because the former requires much more memory to run than the latter, a conclusion we will discuss later in the experiment.

The number of objects and the number of dimensions in a dataset also have an impact on the running time of the algorithm. Theoretically, the number of objects has an even greater impact on the result because it directly determines the number of iterations of the algorithm and the size of the dataset. So we did two experiments to verify this statement. In these two experiments, we changed the number of objects and the number of dimensions, respectively, while keeping the other parameter unchanged. In this round of experiments, we use the synthetic dataset as the input because its parameters are easy to change. The experimental results are shown in Fig. 8. By observing the experimental results, we can intuitively see that these two parameters significantly impact the running time of the algorithm. On the one hand, as the number of objects or dimensions increases, the dataset gets larger and larger because another parameter remains the same; on the other hand, each object corresponds to one iteration, so for both MRBIG and EHBIG algorithms, Hadoop will be started once more for each additional object, so the increase in the number of objects will lead to a sharp increase in the running time.

In this study, the TKD algorithm is implemented based on incomplete datasets, and the incomplete data will directly affect the results of the algorithm. In order to verify the impact of data incompleteness rate on algorithm performance, we used synthetic datasets and obtained three datasets with different incompleteness by modification, 0.05, 0.2, and 0.4. The result is shown in Fig. 9. Experimental results show that the data incompleteness does affect the performance of the algorithm.

As we mentioned before, we put both versions of the algorithm in the paper because both algorithms have advantages in performance, which is caused by the

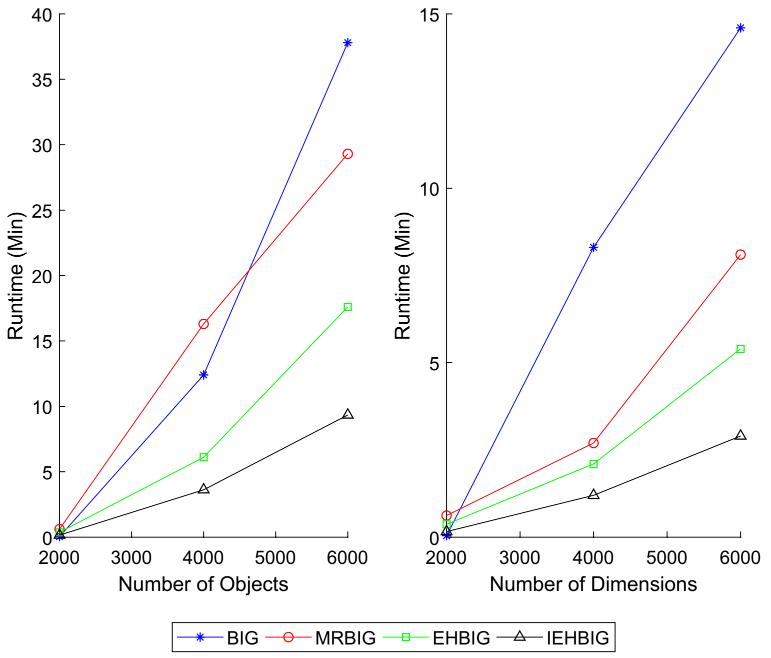


Fig. 8 The runtime of algorithms testing different parameter combinations

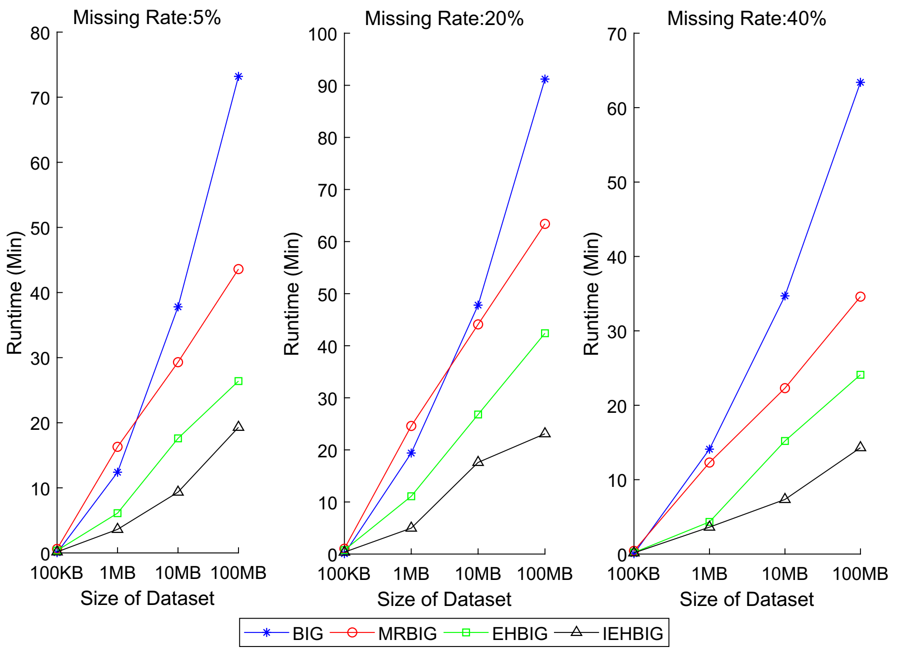


Fig. 9 The runtime of algorithms using different missing rates

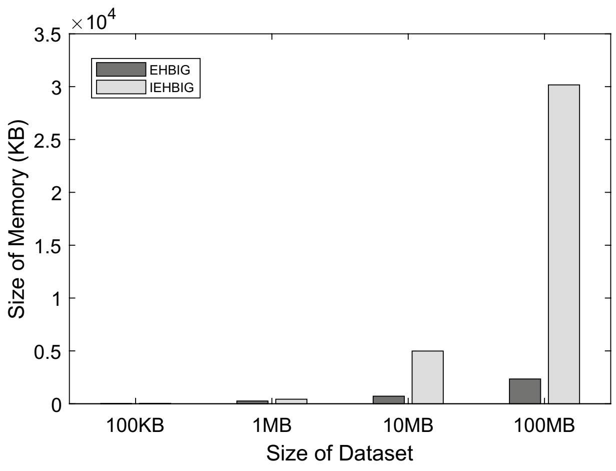


Fig. 10 The memory needed in algorithms

algorithm flow. In addition to the elapsed time verified in previous experiments, another important factor in evaluating algorithm performance is memory. Next, we will introduce new experiments, because this experiment is to verify EHBIG and IEHBIG both the performance of the algorithm, so only of these two algorithms, an experiment is carried out in the experiment, we consider the algorithm implementation of memory, you need to get the results more objective and comprehensive, mentioned in the first experiment we use four different sizes of datasets, and the four datasets from the real dataset, *MovieLens*. And the result is shown in Fig. 10. Obviously, IEHBIG requires a lot more memory than EHBIG because IEHBIG only runs Mapreduce once and calculates the score for all objects at this run time, which is bound to produce many intermediate data, so sufficient memory is required. We can choose these two versions of the algorithm as needed.

Also, we experimented with verifying the effectiveness of our proposed pruning strategy. Since our pruning strategy is based on a given k value, the size of k value will also affect the algorithm performance. In this experiment, we used the synthetic dataset as the input. We carried out three experiments, respectively, changing the k value. The experimental results are shown in Fig. 11. As we see, the change of the k value for MRBIG algorithm has no effect because MRBIG algorithm to calculate the score for each object, first, get top - k sorted, and pruning strategy is used in EHBIG algorithm, which sets threshold according to the scanned k-th value, thus, the smaller the k value, the setting of the threshold value, the greater the off this object will be more, so the algorithm, and even fewer iteration times and running time of the algorithm is less.

To sum up, the performance of EHBIG algorithm is better than that of MRBIG algorithm on large datasets. As can be seen from the above experimental results, firstly, the running time of EHBIG improves by 20%-30% compared with that of MRBIG; second, EHBIG also requires less memory to run. The larger the dataset,

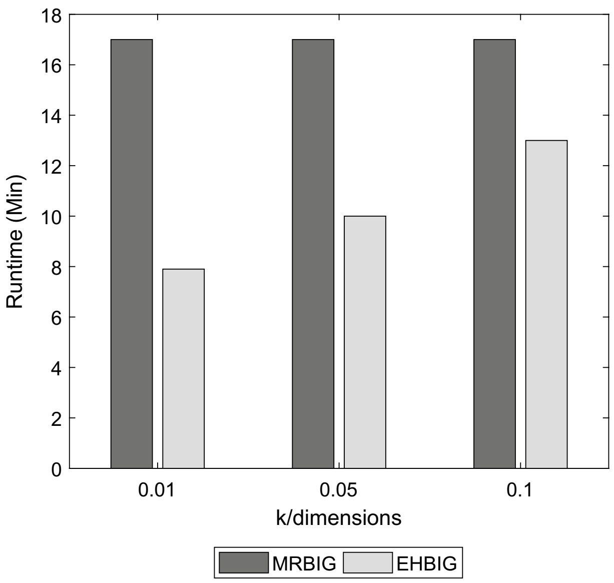


Fig. 11 The runtime of algorithms with different *k*

the more obvious the advantage is. Therefore, EHBIG is more suitable for TKD queries in large datasets and improves its feasibility in practical applications.

6 Conclusion and future work

In this paper, we proposed an algorithm(EHBIG) to apply top-*k* dominating queries using MapReduce framework on incomplete big data, and we developed a new pruning strategy for the algorithm. What‘s more, an improved version of the algorithm (IEHBIG) is developed to optimize the algorithm flow, which applies the MapReduce framework to the whole algorithm process. The two algorithms have their own advantages in performance and improves the performance of the original algorithm to some extent. In the above sections, we have introduced the whole algorithm process, detailed algorithm details, comparisons, and experiments. The experiments show that a stand-alone algorithm is not the optimal TKD query method for large files. When searching for the top-*k* values in a large-scale incomplete dataset, low resource efficiency, process failure caused by insufficient resources, and exponential processing time are the main defects, which are precisely solved by the distributed concept of MapReduce framework. By dividing the input data into several parts and handing them to different nodes for processing, the pressure on a single machine is alleviated, and the computing speed is also improved. This research has great significance for the practical application of TKD query.

Our algorithm has some shortcomings because the Hadoop MapReduce architecture has some disadvantages, such as the inevitable startup time and communication time consumption. Besides, you can apply this algorithm to Spark to overcome some of Hadoop's shortcomings. It is worth mentioning that the search for top-K advantages for incomplete data is an area that has not been fully addressed. With the emergence of big data, EHBIG can be well used in the design of a recommendation system to provide an approximate real-time solution for TKD query processing. TKD query plays an important role in real life, and this study is of great significance. With the development of big data, it is precious to conduct deeper research in this field.

References

1. Papadias D, Tao Y, Fu G, Seeger B (2005) Progressive skyline computation in database systems. ACM Trans Database Syst 30(1):41–82
2. Yiu ML, Mamoulis N (2009) Multi-dimensional top-k dominating queries. VLDB J 18(3):695–718
3. Ge S, Mamoulis N, Cheung DW et al (2015) Dominance relationship analysis with budget constraints. Knowl Inf Syst 42(2):409–440
4. Mamoulis N, Cheng KH, Yiu ML, Cheung DW (2006) Efficient aggregation of ranked inputs, In: 22nd International Conference on Data Engineering (ICDE'06), pp 72–72, IEEE
5. Tiakas E, Valkanas G, Papadopoulos AN, Manolopoulos Y, Gunopulos D (2014) Metric-based top-k dominating queries. In: EDBT, pp 415–426
6. Miao X, Gao Y, Zheng B, Chen G, Cui H (2015) Top-k dominating queries on incomplete data. IEEE Trans Knowl Data Eng 28(1):252–266
7. Zhu H, Li X, Liu Q, Xu Z (2020) Top-k dominating queries on skyline groups. IEEE Trans Knowl Data Eng 32(7):1431–1444
8. Tiwari D, Bhati BS (2021) A deep analysis and prediction of covid-19 in India: using ensemble regression approach. In: Artificial Intelligence and Machine Learning for COVID-19, pp 97–109
9. Tiwari D, Nagpal B (2020) Ensemble methodsof sentiment analysis: a survey. In: 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom), pp 150–155, IEEE
10. Zhang X, Fan M, Wang D, Zhou P, Tao D (2020) Top-k feature selection framework using robust 0-1 integer programming. IEEE Trans Neural Netw Learn Syst
11. Schibler T, Suri S (2020) K-dominance in multidimensional data: theory and applications. Comput Geom 87:101594
12. Xie M, Wong RC-W, Lall A (2020) An experimental survey of regret minimization query and variants: bridging the best worlds between top-k query and skyline query. VLDB J 29(1):147–175
13. Wang Y, Li X, Li X, Wang Y (2013) A survey of queries over uncertain data. Knowl Inf Syst 37(3):485–530
14. Khalefa ME, Mokbel MF, Levandoski JJ (2008) Skyline query processing for incomplete data, In: 2008 IEEE 24th International Conference on Data Engineering, pp 556–565, IEEE
15. Lian X, Chen L (2009) Top-k dominating queries in uncertain databases, In: Proceedings of the 12th international conference on extending database technology: advances in database technology, pp 660–671
16. Lian X, Chen L (2013) Probabilistic top-k dominating queries in uncertain databases. Inf Sci 226:23–46
17. Han X, Li J, Gao H (2015) Tdep: efficiently processing top-k dominating query on massive data. Knowl Inf Syst 43(3):689–718
18. Zhang K, Gao H, Han X, Cai Z, Li J (2020) Modeling and computing probabilistic skyline on incomplete data. IEEE Trans Knowl Data Eng 32(7):1405–1418
19. Chen C-M, Chen L, Gan W, Qiu L, Ding W (2021) Discovering high utility-occupancy patterns from uncertain data. Inf Sci 546:1208–1229
20. Sefidian AM, Daneshpour N (2019) Missing value imputation using a novel grey based fuzzy c-means, mutual information based feature selection, and regression model. Expert Syst Appl 115:68–94
21. Biessmann F, Rukat T, Schmidt P, Naidu P, Schelter S, Taptunov A, Lange D, Salinas D (2019) Datawig: missing value imputation for tables. J Mach Learn Res 20(175):1–6

22. Wu K, Shoshani A, Stockinger K (2008) Analyses of multi-level and multi-component compressed bitmap indexes. ACM Trans Database Syst 35(1):1–52
23. Chen Z, Wen Y, Cao J, Zheng W, Chang J, Wu Y, Ma G, Hakmaoui M, Peng G (2015) A survey of bitmap index compression algorithms for big data. Tsinghua Sci Technol 20(1):100–115
24. Wu K, Otoo EJ, Shoshani A (2002) Compressing bitmap indexes for faster search operations, In: Proceedings 14th International Conference on Scientific and Statistical Database Management, pp 99–108, IEEE
25. Manogaran G, Lopez D (2018) Disease surveillance system for big climate data processing and dengue transmission, In: Climate Change and Environmental Concerns: Breakthroughs in Research and Practice, pp 427–446, IGI Global
26. Kamal S, Ripon SH, Dey N, Ashour AS, Santhi V (2016) A mapreduce approach to diminish imbalance parameters for big deoxyribonucleic acid dataset. Comput Methods Programs Biomed 131:191–206
27. Kamal MS, Parvin S, Ashour AS, Shi F, Dey N (2017) De-bruijn graph with mapreduce framework towards metagenomic data classification. Int J Inf Technol 9(1):59–75
28. Matallah H, Belalem G, Bouamrane K (2017) Towards a new model of storage and access to data in big data and cloud computing. Int J Ambient Comput Intell 8(4):31–44
29. Ezatpoor P, Zhan J, Wu JM-T, Chiu C (2018) Finding top-*k* dominance on incomplete big data using mapreduce framework. IEEE Access 6:7872–7887