Abstract Recently, in the area of big data, some popular applications such as web search engines and recommendation systems, face the problem to diversify results during query processing. In this sense, it is both significant and essential to propose methods to deal with big data in order to increase the diversity of the result set. In this paper, we firstly define the diversity of a set and the ability of an element to improve the overall diversity. Based on these definitions, we propose a diversification framework which has good performance in terms of effectiveness and efficiency. Also, this framework has theoretical guarantee on probability of success. Secondly, we design implementation algorithms based on this framework for both numerical and string data. Thirdly, for numerical and string data respectively, we carry out extensive experiments on real data to verify the performance of our proposed framework, and also perform scalability experiments on synthetic data.

Keywords diversification, query processing, big data

1 Introduction

Nowadays, as the amount of information dramatically increases in several popular applications, such as recommendation systems and web search engines, people are not only satisfied with relevant search results but also require that more relevant yet diverse topics are covered by a limited number of search results. The diversity of a query result means the degree of how the records in the result are different from each other. Therefore, researchers pay considerable attention to the diversity of the search results. Many diversification methods specific to normal-scale data are proposed and applied in practice in order to improve the users’ experience by avoiding the retrieval of too homogeneous results [1].

In big data era, the diversification of query results on big data is extremely important and meaningful. The reason is that when dealing with big data, users can only access a small share of query results due to the huge amount. Thus, we have to return limited number of results with high quality. One of the proper indicators of results’ quality is the diversity of the final result set. Improving diversity of the query result is significant in many areas such as recommendation systems, web search and crowdsourcing [2]. Diversification methods can be applied on the query result of many kinds of queries including range queries, fuzzy queries and common conditional queries in these areas. By providing users with query results diverse enough, we can help them obtain various and sufficient information in reasonable time and therefore can improve the satisfaction of the users. We use the following example to show the importance of increasing diversity of the query result with limited size.

Example 1 Suppose a customer wants to book a hotel in Bali island, and he try to get useful information from a recommendation system with some queries. Information in Table 1 shows the Name, Price, Level, and Location (distance from center) of 10 hotels. Suppose these are all the hotels in Bali, and the customer want a recommendation list including only 3 hotels instead of all the query results. The customer may adopt different kind of queries with different known information.

(1) Assuming that the customer is in entire ignorance of the hotels, and he wants to get the whole picture of the hotels. He
may submit a query like:

\[ Q_1: \text{Select } * \text{ from hotel.} \]

The recommendation list \( L_1 = \{1, 8, 9\} \) is better than \( L_2 = \{1, 2, 3\} \). The reason is that the information of hotels in \( L_1 \) are quite different from each other, while the hotels in \( L_2 \) have similar Name, Price, Location and the same Level. The diversity of \( L_1 \) is higher than \( L_2 \).

(2) Assuming that the expected price of the customer is between 100 and 2500, he may submit a range query like:

\[ Q_2: \text{Select Price from hotel where } 100 \leq \text{Price} \leq 2500. \]

The recommendation list \( L_1 = \{2, 7, 9\} \) is better than \( L_2 = \{5, 7, 9\} \), since the gap between the Price of the hotel ID=2 and the Price of the other hotels in \( L_1 \) is larger than gap between hotel ID=5 and the other hotels in \( L_2 \).

(3) Assuming that the customer want to find a hotel whose name is like ‘manda’, however, he can not think of the whole name. He may submit a fuzzy query like:

\[ Q_3: \text{Select Name from hotel where Name like ‘%manda%’}. \]

The recommendation list \( L_1 = \{1, 4, 6\} \) is better than \( L_2 = \{1, 2, 3\} \), since the Name of the hotels in \( L_1 \) is quite similar to each other. If “Mandapa A Ritz-Carlton Reserve agoda” is the hotel that the customer is looking for, the information in \( L_2 \) is useless.

In Example 1, we can return the whole query result to the customer since there are only 10 tuples in the database. However, the query results of big data may contain a large number of tuples. The expectation of the users is limited information with high quality. Diversity is one of the metrics to measure the quality of query result. Since the number of records in the query result is limited, the more diverse the result is, the higher probability that the information in the result meets the demand of the user. Consequently, the higher diversity leads to higher degree of satisfaction.

Solving the diversification problems on big data faces several technical challenges. The memory is the first challenge. Due to the essential characteristic—huge amount of big data, we have difficulty in dealing with so much data in relatively small memory. Fast response is the second challenge. The required short response time is difficult to achieve due to the large amount of data. These challenges prevent us from fully analysing big data and extracting diverse results from the big data to generate the final result set. Additionally, due to the huge amount of big data, existing super-linear algorithms on small data sets are not acceptable on big data. There is a pressing need to find a method that can solve the diversification problem by only scanning the data once or even less.

Due to the significant importance of diversification on big data and the challenge to solve this problem, we attempt to find methods solving diversification problem efficiently within a small memory. In this sense, for big data, we should design linear or sub-linear space algorithms in order to retrieve diverse results in limited memory and also, we should design on-line algorithms for the restriction of scan times.

While in query processing, many diversification methods have been proposed and implemented specific for normal-scale data [3–6]. However, they are not applicable for big data. This is because existing algorithms fail to take into consideration the huge amount of big data, and most of them are super-linear or have to scan the input data many times. Let us take the classical algorithm-maximal marginal relevance (MMR) [4] as an example. MMR incrementally selects an element from the candidate set and inserts it into the final result set. As for each incremental iteration, we have to scan the candidates once. Therefore, the MMR algorithm has to scan the input data for many times and is not suitable for big data. Max-Sum and Max-Min in [7] maximize the average diversity and minimum diversity, respectively. They greedily find the element bringing the highest gain of diversity from the whole data set in each iteration. Therefore, they have to scan the input data many times. The greedy algorithms GMS and GNE are shown to spend more running time compared with MMR as reported in reference [4]. The diversity-weighted utility maximization (DUM) [6] scans the data set only once when calculating the diverse set, however, the input should be formed in decreasing order of utility. Consequently, the time complexity of DUM is \( O(L \cdot \log L) \), similar to the time complexity of sorting the \( L \) input numbers. This super-linear complexity is also unacceptable for big data.

The query result diversification specialized in big data search has not been systematically studied so far, even though query result diversification over big data is meaningful and helpful in query processing. In the era of big data, we have to

| Data in Bali island |
|---------------------|
| ID | Name | Price | Level | Location |
|-----|------|-------|-------|----------|
| 1   | Mandara Beach Villa Bali | 3,118 | ***** | 9.23 |
| 2   | Mandara Beach Hotel | 2,187 | ***** | 8.62 |
| 3   | Mandara Beach Resort | 2,789 | ***** | 1.06 |
| 4   | Alamanda Canggu Villa | 76 | *** | 34.12 |
| 5   | Mandia Bungalows agoda | 153 | * | 20.24 |
| 6   | Mandapa A Ritz-Carlton Reserve agoda | 5,548 | ***** | 22.96 |
| 7   | Villa Mandi | 344 | *** | 28.12 |
| 8   | Mandala Bungalows | 70 | ** | 20.51 |
| 9   | Mandari Hotel SINGARAJA agoda | 153 | *** | 62.29 |
| 10  | TS Suites Bali and Villas | 2,536 | ***** | 0.92 |
deal with a huge amount of on-line data, which can only be scanned no more than once. Therefore, it is both significant and essential to improve the diversity of results from big data within one scan.

This paper attempts to solve this problem. In this paper, we propose a stream-model-based diversification framework. As we know, this is the first one to diversify the results for big data within one scan. This framework is both effective and efficient in improving the diversity of the result set with low time complexity, low space cost and low computational overhead. Additionally, this framework processes the input data in on-line style and can be implemented for various data types concretely with its advantages.

In this paper, we make the following contributions:

- We propose a framework for diversification of query processing on big data. This framework allows us to improve the diversity of the final result set within one scan with guaranteed performance. In order to describe this framework thoroughly, we firstly present definitions of diversity, possible diversity gain (to describe the ability of an element to improve overall diversity). Secondly, based on these definitions, the framework to solve the diversification problem on big data is proposed, formulated and evaluated. We prove that by assigning proper parameters in this framework, the probability of success in a single run can be guaranteed. As we know, this is the first paper to study the diversification method for big data within one scan.

- On the basis of the proposed diversification framework, we design implementation algorithms for two common data types, numerical data and string data, to diversify the final returned results. We prove that the proposed framework can be implemented specific to different data types effectively and efficiently without degrading the performance of the framework.

- To verify the performance of the proposed algorithms, we perform extensive experiments. From experimental results, our algorithms are verified to be effective and efficient, and we also study the factors that influence the performance of our methods. Then, in order to test the scalability of our proposed framework, we conduct experiments on synthetic data with a tremendous amount and study the performance of the framework. Experimental results also demonstrate that the scalability and efficiency of our approach outperform existing approaches significantly.

The rest of this paper is structured as follows. A brief review of related work about existing diversification methods is presented in Section 2. Also, Section 2 demonstrates the intrinsic distinction between our proposed method and existing methods. Then in Section 3, we present a diversification framework which allows us to improve the diversity of the final result set within one scan with a guarantee of performance and effectiveness. In Section 4, as for numerical data and string data, we present the basic definitions and the corresponding detailed implementation algorithms. Next, Section 5 illustrates the experimental results of our proposed methods on both real data and synthetic data, and then show thorough evaluations of them. Finally, Section 6 concludes this paper and discusses some challenging yet interesting directions for further research.

2 Related work

Diversification in query processing has aroused many researchers’ attention and interest these years. It can help enhance users’ satisfaction in three aspects. Drosou and Pitsoura [1] categorize diversification methods into three parts, namely, content-based, intent-based and novelty-based diversification. Then they survey, compare and study the corresponding definitions, implementation algorithms of these three categories thoroughly and deeply. The first category is content-based diversification, or similarity-based diversification [8]. It aims to return objects which are dissimilar enough to each other. The second category is intent-based diversification, or coverage-based diversification, which cover different aspects of a query, by returning objects from different categories or satisfying various possible users’ intentions [5, 9–11]. The third category is novelty-based diversification, a method to return objects containing new information previously not seen before [12, 13].

The authors in references [14] and [15] categorize the diversification problem into either implicit or explicit. The strategies in the implicit diversification assume no prior knowledge of the query aspects. Many typical diversification methods such as maximum marginal relevance (MMR) [16], MaxSum and MaxMin [5] are implicit methods. In the explicit diversification methods, query aspects are modeled explicitly. These methods employ query logs or exploit query reformulations to model the query aspects [14, 17]. In reference [6], the authors proposed an approach to diversify a list of recommended items by maximizing the utility of the items subject to the increase in their diversity.
Vieira et al. [4] evaluate six existing content-based diversification methods thoroughly, and then propose two new approaches, namely, the greedy with marginal contribution and the greedy with neighbourhood expansion. Angel and Koudas [3] focus on the problem of diversity-aware search which ranks results according to both relevance and dissimilarity to others. Then they propose a diversification method named DIVGEN which is qualified in terms of efficiency and effectiveness.

As for intent-based diversification, researchers propose various methods in order to cover as many user intentions and topics as possible. In [18], Ziegler et al. first propose a new metric, intra-list similarity, to represent the topical diversity of recommendation lists, and then a new method named topic diversification is proposed to decrease the intra-list similarity of lists. In [19], Radlinski and Dumais discuss three result diversification methods, namely, the most frequent method, the maximum result variety method and the most satisfied method so as to improve personalized web search. Additionally, Capannini et al. [17] present an original method, Opt-Select, in order to effectively and efficiently accomplish the diversification task.

With regard to novelty-based diversification, not so many algorithms or implementation techniques have been put forward. Clarke et al. make a thorough distinction between novelty—the need to avoid redundancy—and diversity—the need to resolve ambiguity in [12].

Greedy algorithms are widely adopted to diversify the query results [18, 20, 21], they find the $N$ elements first, and then test whether replacing an element in the origin set with a further element increases the diversity or not. Mostly, multiple passes of input data are needed in most of the methods.

Some approaches adopt tree index based structure to achieve diversity in database systems. In reference [21], a Dewey tree is employed. Diverse tuples are retrieved by traversing this tree. However, this approach can be used only with a specific diversity function on structured data that defines the diversity ordering [2]. The approach proposed in reference [22] is based on cover trees and supports dynamic item insertion and deletion.

Machine learning methods are included in result diversification searching in some researches. R-LTR [23] is a typical diverse learning model. The ranking function in R-LTR is defined as the combination of relevance score and diversity score, where the relevance score only depends on the content of the document, and the diversity score depends on the relationship between the current document and those previously selected. Some variations of R-LTR employ handcrafted relevance features and calculate scores of the linear combinations of the features [24, 25]. In [26], the authors proposed a method using deep learning model of neural tensor network to model the novelty score. Though the diversification results of these methods are better than traditional implicit algorithms, the training process usually costs hours of time.

The essential difference between our proposed method and existing ones is that in our method, the input data only need to be scanned once or less, and during the scanning procedure, we select the element which will diversify the result set and put it into memory. On the contrary, existing diversification methods mostly have to scan and process input data several times and also, cannot make effective use of available memory. In this sense, our presented method can solve the technical challenges which diversification tasks are faced with in big data, while existing methods fail to do so.

3 General framework of diversification

In this section, we present a general framework of diversification for big data. Such framework is suitable for various data types and can effectively improve the overall diversity of the final result set. Firstly, we introduce some basic definitions to better establish the framework. Next we describe the proposed framework and explain how it works. Then, we perform evaluation of this framework. After this, we study the probability of success in improving the diversity of the overall result set through this framework. Finally, we discuss the number samples to select during the implementation of this framework.

As far as we are concerned, our proposed framework is the first one which offers users a diversification method for big data within one scan. It also has the advantage of low time complexity as well as low computational overhead. This proposed framework can also be implemented concretely for various data types without degrading its performance. In implementation, we often carry out randomly sampling to make it suitable for big data.

As in our proposed method, we intend to scan the input big data no more than once and the data is scanned in order, we can logically consider the input big data as streaming data and therefore, can employ the data stream model, in which, the elements $a_1, a_2, \ldots, a_n$ have to be processed one after another. These elements in the stream model are not available for random access from disk or memory and can only be scanned once.
3.1 Basic definitions

3.1.1 Definition of diversity

We assume that the final result set with regard to a user query is limited and stored in memory. We denote the size of available memory as \( m \), the number of elements in the whole data set as \( n \). Our task is to increase the diversity of the elements in memory. The definition of diversity is formalized as follows.

As for several elements \( x_1, x_2, \ldots, x_k \), which make up a vector denoted as \( X \), the diversity of \( X \) is computed as \( \text{Div}(X) = \text{Div}(x_1, x_2, \ldots, x_k) \). Thus, the diversity of \( m \) different elements \( A = \{a_1, a_2, \ldots, a_m\} \) in memory is computed as \( \text{Div}(A) = \text{Div}(a_1, a_2, \ldots, a_m) \). As for different data types, we choose various indicators for \( \text{Div}(A) \).

With regard to numerical data, we use variance to describe the diversity while as for string data, we express diversity as the sum of edit distances. Our definition of diversity is compatible with existing diversity definitions. In [1], Drosou and Pitoura mention a definition of diversity which interprets diversity as an instance of the \( p \)-dispersion problem. The \( p \)-dispersion problem is to pick out \( p \) points from given \( n \) points, so that the minimum distance between any two points is maximized [27]. In this case, the definition of diversity is the minimum distance between any two points, while another existing definition is described as the average distance of any two points.

Since the variance of a numerical data set can describe the distances between numerical elements in the set and similarly, the edit distance can represent the distance between each two strings, the proposed definitions of diversity are reasonable and also compatible with existing ones. The concrete definitions will be discussed in detail in Section 4.

3.1.2 Definition of possible diversity gain

As the number of elements in available memory is fixed as a constant \( m \), they are denoted as \( A = \{a_1, a_2, \ldots, a_m\} \), where \( A \) represents the original result set. As for an input element \( \varphi \), we consider its contribution to the diversity of the result set. Although the contribution of \( \varphi \) to the overall diversity of the final result set could be defined in various ways, we decide to choose the change in diversity value caused when replacing \( \varphi \) with another element in memory. This is because the difference of the “new” diversity when putting \( \varphi \) in memory to take place of another element and the original diversity can both intuitively and clearly describes the ability of \( \varphi \) to diversify current result set. The detailed definition is formalized below.

Given an element \( \varphi \) in the input data, its possible diversity gain to the diversity of the available memory is defined as \( \text{PDG}(\varphi) \), which is computed as follows:

\[
\text{PDG}(\varphi) = \max_{a_j \in A} \{ \text{Div}(A \setminus \{a_j\} \cup \{\varphi\}) - \text{Div}(A) \}. 
\]

From this formula, the possible diversity gain of a certain element \( \varphi \) is the maximal difference of the diversity value, where the element replaces the \( j \)th element \( a_j \) in the memory, and the original diversity value.

Also, it is obvious that \( \text{PDG}(\varphi) \) is only related to the elements \( A = \{a_1, a_2, \ldots, a_m\} \) in main memory, and has nothing to do with other input elements. If we consider the available memory as fixed, \( \text{PDG}(\varphi) \) can be thought as an “attribute” of \( \varphi \). Such attribute of \( \varphi \) is helpful in our further discussion.

3.2 Framework

In this part, we will describe our proposed diversification framework thoroughly and then explain how it works. The framework of our method is shown in Fig. 1. As discussed before, we set the size of available memory as \( m \). We store the initial diversification set containing \( m \) items in the memory. Then, we calculate the \( \text{PDG} \) of the first \( k \) input items and find the maximum \( \text{PDG}_{\text{max}} \). At last, we calculate the \( \text{PDG} \) of remaining items in turn, and replace item whose \( \text{PDG} \) is larger than \( \text{PDG}_{\text{max}} \). Meanwhile, in order to restore the number of the result set, one item should be removed from the memory. More details will be mentioned later along with the introduction of pseudo-code.

![Fig. 1 Framework](image_url)

Our proposed framework is an on-line algorithm, which scans the input data no more than once with a time complexity \( O(n) \). Besides, the required space is far smaller than the input size. In this sense, the algorithm is a sub-linear-space algorithm. Also, this algorithm aims to select one element from input to replace it into available memory and therefore, can increase the diversity of the final result set stored in memory.

The pseudo-code illustrated in Algorithm 1 explains the procedure of our proposed framework. This framework increases the diversity of the results in main memory.
input as \( n \) and suppose that the time complexity of the procedure calculate-element-PDG(\( u_i \)), which is invoked in Line 5 and Line 11 of Algorithm 1, is \( h(m) \). This is a function of \( m \), since the PDG value can be considered as an attribute of an input element and thus, is only related to the elements in memory.

Firstly, during the scanning of first \( k \) elements, we calculate the PDG value of each element in order to initialize the PDG\(_{\text{max}}\) value. Therefore, it costs \( k \cdot h(m) \) time. Then in the following scanning procedure, in Line 10 to Line 16 of Algorithm 1, the number of elements to be considered is no more than \( n - k \). Therefore, the time complexity of this part is \( O((n - k) \cdot h(m)) \). In a nutshell, the time complexity of this proposed diversification framework is \( O(n \cdot h(m)) \). As the input size is \( n \), and the size of available memory \( m \) is a constant far smaller than \( n \) and can be considered irrelevant to the time complexity, the time complexity of the framework is \( O(n) \).

Even though this algorithm is in linear time, it indicates the worst case. Mostly we do not need to scan every element in \( n \) to increase the diversity. First, the elements in the memory will be scanned for \( k \) times to get the PDG\(_{\text{max}}\). After that, we only need to find an element in the following elements which can increase the diversity with a replacement, and the rest of the following elements would not be scanned after the replacement. Mostly, the elements in the memory will be scanned for only a few times much smaller than \( n \). The procedure of checking whether a certain element will be replaced into memory is performed online. Also, if the value of \( k \) is chosen properly, we can guarantee the probability to successfully select the element with the maximal PDG value. This will be discussed more clearly in Section 3.3.

Additionally, this is a sub-linear space algorithm since the space spent is the low-order function of the input size, and this algorithm can improve the diversity of elements stored in limited available memory. In this way, users can obtain more information in a fixed time period and thus, users’ information needs can be satisfied more efficiently. As a result, the users’ satisfaction is improved.

### 3.3 The probability of success

In this section, we provide mathematical foundation of our proposed diversification framework. From the analysis, our framework is established to increase the overall diversity of the final result set by replacing an element with a relatively large PDG value from input into memory. We regard finding the maximal PDG value with our algorithm as a event \( U \), and we will study the success probability \( Pr(U) \) of this event. In

---

**Algorithm 1** Diversify-final-result-set(\( k, n \))

1. **Input**: \( A_0 = \{a_1, a_2, \ldots, a_0\}, a_1, a_2, \ldots, a_0 \)
2. **Output**: final PDG value
3. \( PDG_{\text{max}} \leftarrow -\infty \)
4. for \( i = 1 \rightarrow k \) do
   5. \( PDG(u_i) = \text{Calculate-Element-PDG}(u_i) \)
   6. if \( PDG(u_i) > PDG_{\text{max}} \) then
      7. \( PDG_{\text{max}} \leftarrow PDG(u_i) \)
   8. end if
9. end for
10. for \( i = k + 1 \rightarrow n \) do
11. \( PDG(u_i) = \text{Calculate-Element-PDG}(u_i) \)
12. if \( PDG(u_i) > PDG_{\text{max}} \) then
      13. replace argmax \( \text{Div}(A_0 \setminus \{u_i\} \cup \{u_i\}) \) with \( u_i \)
14. return \( PDG(u_i) \)
15. end if
16. end for
17. replace argmax \( \text{Div}(A_0 \setminus \{u_i\} \cup \{u_i\}) \) with \( u_i \)
18. return \( PDG(u_i) \)

Firstly, when the scanning begins, \( m \) different values are selected and stored into the available memory. We use the algorithm of counting distinct elements in streaming data [28] to implement the procedure efficiently. Additionally, suppose the original maximal PDG value is \(-\infty\), and this can be seen in Line 3 of Algorithm 1.

Secondly, we apply the strategy of selecting a positive integer \( k < n \), scanning the first \( k \) elements of the input, then calculating their corresponding PDG values and storing the maximal PDG value \( PDG_{\text{max}} \) among them. Line 4 to Line 9 in Algorithm 1 clearly demonstrate this procedure. Then the scanning of data continues. Line 10 to Line 16 in Algorithm 1 describe the following condition when the algorithm is executed. If an element \( u \) with PDG value larger than \( PDG_{\text{max}} \) among the following elements is met, the element argmax \( \text{Div}(A_0 \setminus \{u_i\} \cup \{u_i\}) \) in the memory will then be replaced by \( u \) to maximize the difference of the diversity value.

Thirdly, however, if no following elements have a larger PDG value than \( PDG_{\text{max}} \), the last element in the data set, i.e., the \( n \)th element \( u_n \) is used to take place of the element argmax \( \text{Div}(A_0 \setminus \{u_i\} \cup \{u_i\}) \) in the memory. This condition is demonstrated in Line 17 and Line 18 of Algorithm 1.

Through the procedure of this diversification framework, we can make sure which element from input data will be replaced into memory and used to increase the diversity of the final result set in memory.

**Complexity analysis** Here we analyse the time complexity of the proposed diversification algorithm. Set the size of
this section, we separate the task of calculating $Pr(U)$ into three steps. In the first step, we prove that $Pr(U)$ can be computed by summing up the probability of a list of sub-event $Pr(U_i)$ (Theorem 1). In the second step, we prove that the probability of each sub-event $Pr(U_i)$ equals to the probability of two other events happening simultaneously (Theorem 2). At last, we get the boundary of $Pr(U)$ (Theorem 3).

As stated in Section 3.1, the proposed PDG value of a given element can be considered as an attribute of this element and thus, has nothing to do with the other input elements. In this sense, the PDG value here can be analogical to the score described in the on-line hiring problem [29]. Therefore, our mathematical foundation here uses the similar idea of the proof in [29]. During the scanning of input data described in the proposed diversification framework, we hope to select the element with the maximal PDG value and replace it into memory. We denote this event as $U$, and the success probability of this event as $Pr(U)$. Our task is to calculate $Pr(U)$ or obtain the range of $Pr(U)$.

Firstly, we separate the event $U$ into a list of sub-event $U_{k+1}, \ldots, U_n$. We denote $U_i$ as the event that when the element with maximal PDG value is the $i$th element, we successfully select it and put in into memory. Then we obtain Theorem 1 shown below.

**Theorem 1** $Pr(U) = \sum_{i=k+1}^n Pr(U_i)$.

**Proof** For various values of $i$, $U_i$ is non-intersect. Thus we can get $Pr(U) = \sum_{i=1}^n Pr(U_i)$.

According to the description of our algorithms, we will fail to select the best element, i.e., the element with the maximal PDG value, if this element appears in the first $k$ positions from input. Thus, $Pr(U_i) = 0$, where $i = 1, 2, \ldots, k$. In this way, we get Theorem 1.

Secondly, we aim to calculate $Pr(U_i)$. Assume the input data is denoted as $u_1, u_2, \ldots, u_n$. Set $M(u_j) = \max_{1 \leq i \leq j} \{PDG(u_j)\}$ as the highest PDG value of the element which is among $u_1, u_2, \ldots, u_j$. If the event $U_i$ is to take place, two other events have to happen simultaneously. One is that the element with the highest PDG value must be in the position of the element $u_i$, and we define this event as $R_i$. The other is that all the PDG values of elements among $u_{k+1}, u_{k+2}, \ldots, u_{i-1}$ must be smaller than $M(u_k)$. This event is denoted as $T_i$. Then we can obtain Theorem 2 demonstrated below.

**Theorem 2** $Pr(U_i) = Pr(R_i \cap T_i) = Pr(R_i) \cdot Pr(T_i)$.

**Proof** The reason why the event $R_i$ has to happen to make sure that $U_i$ is to take place is obvious. Then, only if $T_i$ happens, the proposed algorithm will not select the element from the $(k+1)$th element to the $(i-1)$th element. $R_i$ and $T_i$ are two independent events, so we obtain the expression of $Pr(U_i)$ according to the statistics property of independent events.

At last, the range of the probability of success $Pr(U)$ can be obtained and illustrated in Theorem 3.

**Theorem 3** $\frac{k}{n}(\ln n - \ln k) \leq Pr(U) \leq \frac{k}{n}((n-1) - \ln(k-1))$.

**Proof** We can easily figure out that $Pr(R_i) = 1/n$ as the maximal PDG value is equally likely to appear in $n$ positions of input. Then $Pr(T_i) = k/(i-1)$, because if $T_i$ happens, it means that the highest PDG value among $u_1, u_2, \ldots, u_{i-1}$ must appear in the first $k$ positions of input and also, this value is equally likely to appear in the $k$ positions. Therefore, $Pr(U_i) = k/(n(i-1))$. Then we can finally get the computational formula of $Pr(U)$:

$$Pr(U) = \sum_{i=k+1}^n Pr(U_i) = \sum_{i=k+1}^n \frac{k}{n(i-1)}.$$ (2)

After simplification, the formula above is expressed as:

$$Pr(U) = \frac{k}{n} \sum_{i=k+1}^n \frac{1}{i-1} = \frac{k}{n} \sum_{i=k}^{n-1} \frac{1}{i}.$$ (3)

Then we can use integration to make constraints on the upper bound and lower bound of $\sum_{i=k}^{n-1} (1/i)$ as follows:

$$\int_k^n \frac{1}{x} \, dx \leq \sum_{i=k}^{n-1} \frac{1}{i} \leq \int_{k-1}^{n-1} \frac{1}{x} \, dx.$$ (4)

After solving the definite integrations above, we get the final upper bound and lower bound of $Pr(U)$ demonstrated in Theorem 3.

In summary, the probability of successfully choosing the element with maximal PDG value in one experiment $Pr(U)$ will be no less than $(k/n) \cdot (\ln(n) - \ln(k))$ and no more than $(k/n) \cdot (\ln(n-1) - \ln(k-1))$. Both upper bound and lower bound are related to the value of $k$ and $n$. In a certain experiment, the value of $n$ is set. Then if $k$ is chosen properly, we can guarantee the probability of success in one single experiment is satisfactory.

### 3.4 Discussion

As described before, we know how the proposed framework works and why this framework has a guarantee on probability of success. In the implementation of this diversification framework, due to the large amount of big data, we have to
pick out representative samples from the input data instead of accessing the whole data set. In this section, we will discuss the relation between the number of samples and the reliability of the diversity result captured from the samples. We prove the limitation of the degree that the results of samples deviate from the mathematical expectation of the population in Theorem 4. In addition, we will discuss how to select a proper number of samples.

As the total amount of input data can be large, sampling is essential in experiments. Hence, we divide the input into a certain number of segments with the same size. These segments can be considered as the samples of the whole big data set. Concretely, we try to divide the data file into segments with a fixed size of \(a\), then the whole data file is partitioned into \([n/a]\) segments. We select \(s\) segments from them and use them as experimental samples. As we carry out such sampling experiments and obtain corresponding results based on the samples selected, it is apparent that variables \(s\) and \(a\) will exert an influence on the performance.

Now suppose we select \(n\) samples to represent the whole data set, and we set \(X_i\) to represent the result of the \(i\)th sample. \(X_i = 1\) represents that we manage to find the element with the maximal PDG value and replace it into memory in order to increase the diversity. Then, \(X_i = 0\) means that we fail to do so. It is obvious that \(X_1, X_2, \ldots, X_n\), the \(n\) samples, can be treated as a sequence of independent Poisson tests. \(X_1, X_2, \ldots, X_n\) all satisfy that \(Pr(X_i = 1) = P_i\), where \(i = 1, 2, \ldots, n\). Then we suppose that \(X = \sum_{i=1}^{n} X_i\) and then \(\mu = E(X)\), using \(\mu\) to denote the mathematical expectation of \(X\). As a result of the properties of the expectation and the summation, we obtain the following equation.

\[
\mu = E(X) = E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} P_i. \tag{5}
\]

Now, we would like to prove the limitation of the degree that the results of samples deviate from the mathematical expectation of the population.

**Theorem 4** \(Pr(|X - \mu| \geq \delta \mu) \leq 2e^{-\frac{\delta^2 \mu}{3}}.\)

**Proof** We can get the following inequality according to Chernoff bound [30].

\[
Pr(|X - \mu| \geq \delta \mu) \leq 2e^{-\frac{\delta^2 \mu}{3}}. \tag{6}
\]

Thus, we define \(f(\mu) = 2e^{-\frac{\mu^3}{3}}\) to describe it. The 1st derivative of \(f(\mu)\) can be easily calculated as \(f'(\mu) = 2e^{-\frac{\mu^3}{3}}(-\delta^2/3)\). Clearly, \(f'(\mu) \leq 0\). It means that \(f(\mu)\) decreases gradually with \(\mu\) increasing. Thus, suppose the maximal value of \(\mu\) is \(\mu_{\text{max}}\) and the minimal value of \(\mu\) is \(\mu_{\text{min}}\). Then as a result of the properties of \(f(\mu)\), we make sure that \(f(\mu_{\text{max}}) \leq f(\mu) \leq f(\mu_{\text{min}})\). From Eq. (6), we can get

\[
Pr(|X - \mu| \geq \delta \mu) \leq f(\mu), \tag{7}
\]

\[
Pr(|X - \mu| \geq \delta \mu) \leq f(\mu_{\text{min}}). \tag{8}
\]

Now our task is to compute \(f(\mu_{\text{min}})\). As \(\mu = \sum_{i=1}^{n} P_i\), the minimal value of \(\mu\) is presented as \(\mu_{\text{min}} = n \cdot \min P_i\). Back in the inequality in Theorem 3 in Section 3, here \(Pr(U) = k/n \sum_{i=k}^{n} 1/i\) represents the probability that one single experiment is successful. In this sense, \(Pr(U)\) is equivalent to \(P_i\). Thus, it is true that

\[
\frac{k}{n}(\ln(n - \ln k) \leq P_i \leq \frac{k}{n}(\ln(n - 1) - \ln(k - 1)). \tag{9}
\]

If we try to calculate \(\mu_{\text{min}},\) we have to compute the minimal value of \(P_i\). Set \(h(k) = \frac{k}{n}(\ln(n - \ln k), \) as \(P_i > h(k),\) then it is apparent that \(P_i > h(k)_{\text{max}},\) and the minimal value of \(P_i\) is \(h(k)_{\text{max}}\). After calculation and analysis, when \(k = n/e,\) the value of \(h(k)\) is the largest and \(h(k)_{\text{max}} = 1/e.\)

Therefore, \(\mu_{\text{min}} = n \cdot \frac{1}{e} = \frac{n}{e}.\)

Consequently,

\[
Pr(|X - \mu| \geq \delta \mu) \leq 2e^{-\frac{\delta^2 \mu}{3}}. \tag{10}
\]

The inequality in Theorem 4 measures the degree that the results of samples deviate from the mathematical expectation of the population. Theorem 4 describes the probability that the deviation of this sampling result from the expectation of the population is no less than \(\delta\) times of \(\mu.\) We denote its right section as \(p_0.\) Therefore, we can get \(Pr(|X - \mu| \geq \delta \mu) \leq p_0.\)

As \(\delta\) is an indicator of the deviation degree of sampling results from the expectation of the population, different values of \(\delta\) have various mathematical meanings. \(n\) describes the number of picked out samples. By choosing various values of \(\delta\) and \(n,\) we can get different \(p_0.\) That is, diverse probability values.

After testing the change of \(p_0\) values with \(\delta\) and \(n,\) two conclusions are drawn. The first is that with a given \(n,\) if \(\delta\) increases, then the value of \(p_0\) correspondingly decreases. The second conclusion is that when the value of \(\delta\) stands constant, with \(n\) increasing, the value of \(p_0\) is on the decrease.

Therefore, it is learned that given a certain \(\delta,\) which represents the deviation degree, we should correspondingly choose a proper value of \(n, i.e., the number of samples, in order to make sure that the value of \(p_0\) is small enough. Only with this method, can we ensure the effectiveness and reliability.
of our diversification framework using samples to represent the whole population.

4 Implementation algorithms

In Section 3, we thoroughly describe the diversification framework and then explain how it works. In this section, we develop implementation algorithms based on our proposed framework for two kinds of data types, numerical data and string data. Note that these two kinds covers most data types. For instance, both integer and double values can be processed as numerical data while both text and category attributes can be processed as string data.

4.1 Algorithms for numerical data

4.1.1 Expression of diversity and PDG

In this section, we assume the input data is in the form of numerical values. Then our task is to pick out a portion from the query results on a massive numerical data set and then improve the diversity of this portion.

As discussed in Section 3, we assume that the final result set with regard to a user query is stored in memory. Then we denote the size of available memory as $m$, the number of elements in the whole data set as $n$, and our task is to increase the diversity of the elements in memory.

As for several numerical values $X = \{x_1, x_2, \ldots, x_k\}$, we describe the diversity $Div(X)$ of $X$ as the variance, represented as $Div(X) = Var(X) = E[(X - \mu)^2]$, where $\mu$ is the average value of $X$. Thus, the diversity of $m$ different values $A = \{a_1, a_2, \ldots, a_m\}$ in memory can be computed as $Div(A) = Div(a_1, a_2, \ldots, a_m) = \frac{1}{m} \cdot \sum_{i=1}^{m} (a_i - \mu_0)^2$, where $\mu_0$ is the average of $a_1, a_2, \ldots, a_m$.

Then, given an element $\varphi$ in the input, we describe its possible diversity gain to the diversity of the available memory as $PDG(\varphi)$. In this case, $PDG(\varphi)$ is computed as follows.

$$PDG(\varphi) = \max_{a_j \in A} \{Var(A \setminus \{a_j\} \cup \{\varphi\}) - Var(A)\}. \quad (11)$$

Variance measures how far the elements are spread out from the mean value and from each other. Therefore, if the variance value of several elements is large, then it means that these elements are pretty different from each other. It correspondingly means that the diversity of this element set is large. In this sense, variance is a proper indicator of diversity among numerical elements. Additionally, with our simple but useful definition of diversity, it can really decrease the computational overhead of our proposed method.

4.1.2 Implementation algorithm

As demonstrated in Algorithm 1 in Section 3, the algorithm invokes the function $PDG(u_i) = \text{calculate-element-PDG}(u_i)$ during its execution. This algorithm specific to numerical data is shown in Algorithm 2.

| Algorithm 2 Calculate-numerical-element-PDG($\varphi$) |
|------------------------------------------------------|
| 1: **Input:** values in main memory $A_0 = \{a_1, a_2, \ldots, a_m\}$, double element $\varphi$ |
| 2: **Output:** PDG value of $\varphi$ |
| 3: **result** = 0 |
| 4: for $i = 1 \rightarrow m$ do |
| 5: $distance_i = \text{Improve}(\varphi, a_i)$ |
| 6: if $variance_i > result$ then |
| 7: **result** $\leftarrow$ $variance_i$ |
| 8: end if |
| 9: end for |
| 10: $PDG(\varphi) \leftarrow$ **result** |
| 11: return $PDG(\varphi)$ |

Here we analyse the procedure of Algorithm 2 in detail. The elements in memory are described as a vector $A_0 = \{a_1, a_2, \ldots, a_m\}$. As for an input element $\varphi$, our intention is to output $PDG(\varphi)$. As illustrated in Line 3, we first initialize $result$ as 0. Then in the loop presented in Line 4 to Line 9, we take into account the diversity improvement $\text{Improve}(\varphi, a_i)$ after replacing $a_i$ with $\varphi$. Within the loop, if this improvement is larger than $result$, then we update the value of $result$ with the improvement. When the loop is terminated, in Line 10, the value of $result$ is assigned to $PDG(\varphi)$ and thus, this algorithm halts.

**Time complexity analysis** In order to prevent calculating the variances frequently, we store the mean $\mu$ and the variance $Var$ of the elements in the original set $A$. We define $\mu_{old}(i)$ and $Var_{old}(i)$ as the mean and variance of the elements in $A$ except $a_i$, respectively. These two variations can be calculated easily in the following way as $\mu_{old}(i) = (\mu m - a_i)/(m - 1)$, and $Var_{old}(i) = (Var \cdot m - (a_i - \mu)^2)/(m - 1)$. The time cost of computing these two variations is $O(1)$.

When a new element $\varphi$ arrives, we can regard the task of calculating the new variance after replacing one element in $A$ as the combination of two sets. The old set is composed of $m - 1$ elements, and the mean and variance can be calculated as we discussed above. The new set is the new element $\varphi$. Then, the new mean $\mu_{new} = \varphi$, and the variance $Var_{new} = 0$.

Then, the combination of the $m - 1$ elements in $A$ and the new element $\varphi$ can be regraded as the set after replacing $a_i$ with $\varphi$. The mean of the combination $\mu_{total} = (\mu m - a_i + \varphi)/m$. The variance of the combination set can be
calculated as $\text{Var}_{\text{total}} = (m - 1)[\text{Var}_{\text{old}}(i) + (\mu_{\text{old}}(i) - \mu_{\text{total}})^2] + [\text{Var}_{\text{new}} + (\mu_{\text{new}} - \mu_{\text{total}})^2]/m$. The time cost of computing $\text{Var}_{\text{total}}$ is $O(1)$. Since we have to choose the maximum diversity improvement from the $m$ replacements, the time complexity of processing one new element is $O(m)$. The space cost is $O(m + 2)$.

The time complexity of processing the whole data set including $n$ elements is $O(mn)$, and the space complexity is still $O(m)$.

We then use an example below to illustrate the algorithm.

**Example 2** Consider the following scenario. A user wants to select a house to purchase from a tremendous amount of information, but the concrete intention for the spot or the size is not clear. Then we can really do him a favour by providing him with a certain number of houses with diverse sizes and layouts, which of course, are in the form of numerical values. Let us just take this situation as an example.

Suppose there are 25 numerical elements in the input, which successively are 711.56, 121.65, 7498.12, 2866.83, 794.47, 7638.57, 9561.95, 6819.74, 8324.07, 2753.54, –272.60, 3396.49, 3857.34, 5266.30, 2888.52, 4681.03, 6.34, 5494.43, –8914.71, 7603.40, 1428.25, 591.98, 3332.02, 9255.67, 7133.70. Set the size of available memory as $m = 5$, then the parameter $n$ which means the number of input elements described in the framework is correspondingly 20. We assume the number of elements to scan is $k = 10$. Thus, for 25 numerical elements, 5 will be stored in memory and the following $n = 20$ will be used as input data.

From Algorithm 2, firstly, we choose $m$ different elements into memory and they are 121.65, 711.56, 7498.12, 2866.83, 794.47 in order. Secondly, we try to scan the $n$ elements in order. We scan the next $k$ elements and find out the $PDG_{\text{max}}$ value is from the element 9561.95, which is in the second position of $n$ elements. Then the scanning continues. When we get to the 14th element –8914.71, it is found that its PDG value is larger than the $PDG_{\text{max}}$ value and it needs to replace the second element in memory. Thirdly, we carry out the replacement and improve the diversity of elements stored in memory.

However, if we change the element –8914.71 in input data to 8914.71 and then also carry out the procedure of Algorithm 2, we may find the implementation results are quite different. The $PDG_{\text{max}}$ value stays the same, but when the scanning continues, no following elements have a larger PDG value than $PDG_{\text{max}}$. Thus, we select the last element 7,133.70 to replace into memory and it needs to take place of the 4th element in it.

There are two possible execution results of Algorithm 2. With the implementation of this algorithm, the consumer access enough measurements and thus, can have more options. In this sense, our work is quite meaningful.

### 4.2 Implementation on string data

#### 4.2.1 Expression of diversity and PDG

To study how to describe the diversity of a string set, we first lay emphasis on how to represent the dissimilarity between two strings. Here, we use edit distance, which is the minimum operations from insertion, deletion or substitution to change one string to another, so as to measure the difference between two strings. We choose edit distance because it is often used to measure the difference between strings [31]. Then given two strings $x, y$, the dissimilarity between them is denoted as $\text{Dis}(x, y) = \text{EditDis}(x, y)$.

Thus, we represent the diversity of a string set $S = \{s_1, s_2, \ldots, s_m\}$, whose cardinality is $m$, as follows:

$$\text{Dis}(S) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \text{Dis}(s_i, s_j) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \text{EditDis}(s_i, s_j).$$

(12)

Note that $\text{Dis}(S)$ is equivalent to $\text{Div}(S)$ when dealing with string data.

Then, we describe the PDG of a string element quantitatively. Given a certain string element $\varphi$ in the input data, we define its possible diversity gain as follows.

$$PDG(\varphi) = \max_{s_i \in \varphi} [\text{Dis}(S \setminus \{s_j\} \cup \{\varphi\}) - \text{Dis}(S)].$$

(13)

Edit distance can well measure the difference between two strings. In this sense, we sum up the edit distances between all pairs of strings in a string set together, and then use the result to describe how various the string elements in this set are. Therefore, this sum result denoted as $\text{Dis}(S)$ can reasonably represent the diversity of this string set $S$.

#### 4.2.2 Implementation algorithm

We have proposed the implementation of the algorithm named $PDG(u_i) = \text{calculate-numerical-element-PDG}(u_i)$ specific to numerical data in Algorithm 2. Here we focus on the implementation of our proposed diversification framework specific to string data. Its pseudo-code is demonstrated in Algorithm 3.

Now we explain the procedure of Algorithm 3 in detail. At first, strings $a_1, a_2, \ldots, a_m$ are stored in memory and form a vector denoted as $A_0$. The input of this algorithm is a string
element $\varphi$ and the output is $PDG(\varphi)$. In Line 3, we firstly initialise $result$ as 0. Secondly, in the loop in Line 4 to Line 9, we consider the conditions where $\varphi$ takes place of each element in memory, and compute $\text{Improve}(\varphi, a_i)$ the diversity improvement after replacing $a_i$ with $\varphi$. With this value, we then check whether it is larger than current $result$ value. If so, we assign this edit distance sum value to $result$. Finally, in Line 10, $PDG(\varphi)$ is set as $result$ and then the algorithm is terminated.

**Algorithm 3** Calculate-string-element-PDG($\varphi$)
1: Input: available memory $A_0 = \{a_1, a_2, \ldots, a_n\}$, string element $\varphi$
2: Output: PDG value of $\varphi$
3: $result = 0$
4: for $i = 1 \rightarrow m$
5: $distance_i = \text{Improve}(\varphi, a_i)$
6: if $distance_i > result$ then
7: $result \leftarrow distance_i$
8: end if
9: end for
10: $PDG(\varphi) \leftarrow result$
11: return $PDG(\varphi)$

**Time complexity analysis** Suppose the length of strings considered is $a$. Then it takes $\theta(a^2)$ time to calculate the edit distance between two strings. In implementation, we employ a method that only takes into account the calculation of the edit distances between the strings which are influenced by the input string $\varphi$. That is, the improvement of diversity after replace element $S[i]$ with $\varphi$ is $\text{Improve}(\varphi, S[i]) = \sum_{j \neq i \neq j \neq j}(\text{EditDist}(\varphi, S[j]) - \text{EditDist}(S[i], S[j]))$. In order to avoid calculating the edit distance so many times and further improve the efficiency, we store some intermediate results.

We define $D[i] = \sum_{j \neq i \neq j}(\text{EditDist}(S[i], S[j]))$. $D[i]$ is the sum of the edit distance from the ith element to other elements in Memory $S$. We store the list of $D[1], D[2], \ldots, D[m]$, since these values will be used frequently. The space cost of this list is $\theta(m)$, and the time cost is $\theta(m^2 + a^2)$.

When a new element $\varphi$ arrives, we store the edit distance $\text{EditDist}(\varphi, S[i])$ between $\varphi$ and each element $S[i]$. We define $\text{NewDistance}[i] = \text{EditDist}(\varphi, S[i])$ and store the list $\text{NewDistance}$ composed of $m$ elements. The space cost of this list is $\theta(m)$, and the time cost is $\theta(m + a^2)$. We also store the $SUM$ of elements in $\text{NewDistance}$.

In this way, the $\text{Improve}(\varphi, S[i]) = (SUM - \text{NewDistance}[i]) - D[i]$ can be calculated in $\theta(1)$. Since we have to process $m$ elements in $S$, the time cost is $O(m)$. In conclusion, the total space cost is $O(m)$ and the time cost of processing one new element is $\theta(m^2 + a^2 + m)$. The complexity of calculating the list $D$ is added in, since we will calculate $D$ only once no matter how many new elements to be processed.

We discuss the complexity of calculating $PDG$ of a string element $\varphi$ above. When processing the whole data set with $n$ new elements, the time cost is $\theta(m^2 a^2 + n(m \cdot a^2 + m))$. Since $a$ is a constant, and $m$ is much smaller compared with $n$, the time complexity can be regraded as linear. And the total space cost is still $O(m)$.

We now take a practical situation as an example and explains the procedure of the implementation method more thoroughly.

**Example 3** Here we collect the input data from Amazon. We choose the category “Books” and then focus on the book list called “100 Books to Read in a Lifetime: Readers’ Picks”. We randomly choose several books and use their names as input in this example.

Suppose the input has 15 strings. That is $s_1$: “A Brief History of Time”, $s_2$: “Alice Munro: Selected Stories”, $s_3$: “Bel Canto”, $s_4$: “Charlie and the Chocolate Factory”, $s_5$: “Darling Greatly: How the Courage to Be Vulnerable Transforms the Way We Live, Love, Parent, and Lead”, $s_6$: “Great Expectations”, $s_7$: “Harry Potter and the Sorcerer’s Stone”, $s_8$: “Invisible Man”, $s_9$: “In Cold Blood”, $s_{10}$: “Jimmy Corrigan: Smartest Kid on Earth”, $s_{11}$: “Kitchen Confidential”, $s_{12}$: “The Devil in the White City: Murder, Magic, and Madness at the Fair that Changed America”, $s_{13}$: “Love in the Time of Cholera”, $s_{14}$: “Man’s Search for Meaning”, $s_{15}$: “The Lion, the Witch and the Wardrobe”.

Set $m = 5$, $k = 5$ and $n = 10$. Firstly, Algorithm 3 scans the input data and store $m$ different elements in memory, and the string elements are $s_1, s_2, s_3, s_4, s_5$ in order. Then we scan the next $k$ string elements and pick out the $PDG_{max}$ value which is from $s_{10}$. Then the scanning continues. When we reach $s_{12}$, this element has a larger PDG value than $PDG_{max}$ and it needs to take place of the second element, that is, $s_5$ in memory. Then the replacement is carried out and the overall diversity of available memory is increased.

Nevertheless, if we change $s_{12}$ in input file to “Life After Life” and still execute the algorithm, we will find that $PDG_{max}$ will not vary, but no following elements have a larger PDG than $PDG_{max}$. Then the scanning continues and the last element, $s_{15}$, is met. The last element should replace $s_1$ in memory to obtain its PDG. Therefore, we switch $s_{15}$ with $s_1$ in memory and manage to diversify the results.

There are two possible conditions of this method and they are both illustrated in the example above. Additionally, in this scenario, to choose a book from the given book list, our pro-
posed method allows diversifying the returned book titles and thus, can improve users’ satisfaction and experience.

5 Experimental evaluation

In this section, we evaluate our proposed methods with extensive experiments.

Test datasets We choose various data sets according to the features of two different data types: as for numerical data, we use double data extracted from TPCH (see TPC-H Homepage). The scale of double data is 150,000.

With regard to string data, we carry out our experimental evaluation on DBLP dataset (see Index of xml in DBLP), and the scale of string data utilized is 500,000. The titles of various papers are used as the input of experimental evaluation of string data.

The TPCH dataset and DBLP dataset are used to test the impact of parameters on the performance, and the experimental results are shown in Section 5.2 and Section 5.3. In order to test the scalability of our algorithm on big data, we generate two synthetic datasets in 5.0GB. The synthetic numerical dataset is generated uniformly in [−1000, 1000]. Each string data in the synthetic string dataset is composed of 1 to 10 characters generated randomly from \{a, b, . . . , z\} ∪ \{A, B, . . . , Z\}. The experimental results of the scalability and comparisons are shown Section 5.4.

In order to test the performance of our diversification algorithm, we just regard the datasets as the query results, and implement our algorithm directly on these datasets. However, our diversification algorithm can be applied to the query processing of many kinds of queries including range queries, fuzzy queries and common conditional queries. Each of the query result satisfying the conditions in a query can be regarded as the input data of our diversification algorithm. Since our algorithm is a one-pass algorithm, the diversification set can be computed along with the query processing.

5.1 Experimental methods

Independent variables In our algorithms, three parameters affect the effectiveness and efficiency of our methods. The first one is the size of available memory \(m\). The second parameter is the total number of input elements \(n\) and the third parameter is the number of elements \(k\) to be firstly scanned in the procedure.

As discussed in Section 4, the cardinality of selected samples \(a\) and the number of selected samples \(s\) both exert an influence on our methods’ performance. Hence, these two parameters have to be taken into consideration as well.

As discussed above, in the global experiments, we consider five parameters, \(m, n, k, a\) and \(s\). We study and analyse these their influence on the efficiency and effectiveness of proposed methods, respectively. These five parameters may affect each other to some extent, so we employ the control variate method to study the impact of each parameter on the proposed methods’ performance.

Dependent variable In order to better measure the impact of these variables on returned results’ diversity, we present a measurement to describe the degree that the diversity is increased. With the definition of PDG value proposed in Section 3 and Section 4, suppose the original diversity value computed is \(div_0\), and the PDG value after diversification is denoted as \(PDG\), we define diversity increasing rate, i.e., \(DIR\) value as \(DIR = PDG/div_0\).

In the next two subsections, we study and analyse the impact of five parameters, \(m, n, k, a,\) and \(s\) on \(DIR\) value and discuss about whether and how they affect the performance of our diversification methods.

About \(\delta\) When we try to study the influence of five control parameters on the diversity increasing rate, we base our experimental results on a fixed number of randomly selected samples. Note that we employ the simple random sampling scheme without replacement. In most of our presented figures (except for the experimental study on the impact of selected sample number \(s\)), three separate lines are illustrated with the legends named respectively as \(\delta = 0.2, \delta = 0.5,\) and \(\delta = 0.8\), respectively. \(\delta\) describes the degree that sampling results deviate from population results.

About \(p_0\) Considering the meaning of the formula discussed in Section 4, \(Pr(|X - \mu| > \delta \mu) \leq p_0\), these three lines are drawn using various \(\delta\) and different \(s\) in order to void large \(p_0\). As for \(\delta = 0.2\), we set \(s = 500\), and then we get \(p_0 = 1.72151 \times 10^{-1}\), i.e., \(Pr(|X - \mu| > \delta \mu) \leq 1.72151 \times 10^{-1}\). It means that the probability our sampling results deviate from the theoretical results of all data, to a degree of more than \(\delta\) times of expectation value, is less that \(1.72151 \times 10^{-1}\). As the value of \(p_0\) is small enough, we can guarantee that the results of our sampling experiments are credible.

Similarly, with regard to \(\delta = 0.5\) and \(\delta = 0.8\), we respectively set the value of \(s\) as 300 and 100, and then the value of \(p_0\) is correspondingly \(2.02689 \times 10^{-4}\) and \(7.80991 \times 10^{-4}\). As the values of \(p_0\) are small enough, we ensure that the experimental results of these two scenarios are both trustable.

5.2 Experiments on numerical data

In this section, we analyse the impact of the five parameters
on the DIR value, when dealing with numerical data and aiming to diversify returned results.

5.2.1 Impact of memory size

First, we consider the impact that $m$, i.e., the size of memory, exerts on the value of DIR. We present a scatter plot of DIR value vs. the value of $m$ in Fig. 2(a). As illustrated in the figure, it is obvious that with the increasing of $m$, DIR value tends to decline. It is all the same for various $\delta$ values. The experimental results are consistent with the analysis. Because a larger $m$ means that the size of available memory gets larger, then only replacing one element into memory to increase the overall diversity, i.e., the variance of returned results stored in memory, will be harder. Therefore, the diversity increasing rate will decline significantly.

Then as observed from Fig. 2(b), which describes the influence of $m$ on the running time of a single experiment, when $m$ becomes larger, the running time gets larger correspondingly, and it holds for various $\delta$ values. This is because with the increasing of $m$, to calculate the original diversity of the results stored in memory takes more time. Also, as for each input element $\varphi$, when considering replacing it into memory, it takes more time to compute the “new” diversity, i.e., variance, of the “new” result set. Therefore, the running time will also increase.

5.2.2 Impact of number of elements to be scanned

In this part, the impact of the number of elements to be scanned, that is, $k$ is studied. A scatter plot of DIR value vs. the value of $k$ is illustrated in Fig. 3(a). As observed in the figure, with $k$ increasing, the DIR value is mainly decreasing, except for a little rise in some local areas. This is true according to three different values of $\delta$, which means this experimental result holds as for various frequencies in sampling. The reason is that when $k$ gets larger, more elements are scanned to find the initial maximal PDG value. Then if the initial maximal PDG value becomes larger, the following elements will not have a larger PDG value. Thus, the last element which contributes less is selected and the DIR value gets smaller.

Now we analyse the influence of $k$ on the running time of a single experiment. From Fig. 3(b), we observe that the running time increases accordingly with the growth of $k$, regardless of $\delta$ values. This is because when $k$ becomes larger, during the procedure to find the $\text{PDG}_{\text{max}}$ value, we have to scan more elements and thus, process more elements by computing their PDG values. Therefore, the computational overhead gets larger and then the running time of a single experiment increases.

5.2.3 Impact of number of input elements

As for three $\delta$ values 0.2, 0.5 and 0.8, with the input scale $n$ increasing, the corresponding DIR values are all going through several rises and falls. The plot with regard to the relationship between DIR value and $n$ is Fig. 4(a). This is because when the scale of input data gets larger, the elements to be selected will be different and, meanwhile, the ability of the selected element to increase the variance of the result set will also be diverse.

Also, the running time of a single experiment varies and fluctuates for various $\delta$ values when $n$ increases. This is illustrated in Fig. 4(b). The running time is relevant to the number
of elements processed in each experiment and also related to the data distribution in the selected samples. Here only the total number of input elements gets larger, but we have no idea how the data distribution is. Hence the running time may go through several rises and falls.

![Image of Fig. 4](image)

**Fig. 4** Impact of input size $n$ (numerical). (a) On DIR; (b) on running time

5.2.4 Impact of cardinality of selected samples

As shown in Fig. 5(a), no matter what $\delta$ is, when the sample size $a$ gets larger, the DIR becomes larger on the whole. Such experimental results are understandable in that a larger $a$ indicates a larger size of samples in one test. Then the larger the sample size is, the more possible that it is to find an element with a large enough PDG. Hence, DIR gets larger accordingly.

![Image of Fig. 5](image)

**Fig. 5** Impact of sample size $a$ (numerical). (a) On DIR; (b) on running time

If we consider the running time of a single experiment, we obtain the results illustrated in Fig. 5(b). As for three different $\delta$ values, when the value of $a$ increases, the running time will get larger accordingly. This phenomenon is understandable in that if $a$ is larger, we have to deal with more data in each sampling experiment. When the processing time of an element does not change too much, the running time of an experiment will accordingly become longer.

5.2.5 Impact of number of selected samples

When we choose different numbers of samples, denoted as $s$, the DIR value varies in a fluctuating way correspondingly. The scatter plot is illustrated in Fig. 6(a). This result is caused by the different values of $s$ and also, diverse selected samples.

Additionally, the running time varies when $s$ changes. The scatter plot of the relationship between $n$ and the running time can be observed in Fig. 6(b). The running time is also related to the data distribution of samples and which samples we choose.

![Image of Fig. 6](image)

**Fig. 6** Impact of sample number $s$ (numerical). (a) On DIR; (b) on running time

5.2.6 Discussion

Well, we cannot overlook the important experimental result that in Figs. 2–6, the DIR value is all positive and in some cases, the DIR value can be quite large. This means that the result set is always more diverse after implementing the corresponding procedure of our proposed method and sometimes, the diversity is increased to a great extent. Therefore, the performance of our proposed methods to increase the diversity of results is pretty satisfactory, and this proves the ef-
fectiveness of our proposed diversification method specific to numerical data.

5.3 Experiments on string data

In this section, we study how the five parameters affect the results of the proposed diversification method with regard to string data experimentally.

5.3.1 Impact of memory size

First, when the memory size $m$ becomes larger, the diversity increasing rate, i.e., DIR value, gets smaller and the velocity of this decline procedure gets lower. When the values of $\delta$ are various, which means that we select and use different number of samples, the results are consistent. Such results can be intuitively observed in Fig. 7(a). The experimental results are in this way, since if the size of memory increases, the increase scale in diversity when a single element is replaced into memory becomes smaller. In detail, we describe the diversity of a string set as the sum of edit distances between each two elements. Then, when $m$ increases, exchanging only one element with another in memory will not contribute much to the overall diversity.

Fig. 7 Impact of memory size $m$ (string). (a) On DIR; (b) on running time

Well, when $m$ gets larger, the running time of a single experiment tends to increase. This is the same no matter what the value of $\delta$ is. The relationship between the running time and the value of $m$ is intuitively illustrated in Fig. 7(b). This is because when more elements are stored in memory, it takes more time to compute the original diversity, i.e., the sum of edit distances here, of the element set. Also, as for each element $\varphi$, it takes more time to compute its PDG value (demonstrated in Algorithm 3). Therefore, the running time correspondingly increases with a larger $m$.

5.3.2 Impact of number of elements to be scanned

In Fig. 8(a), we can easily find that when the number of elements to be scanned $k$ keeps on increasing, the DIR value first rises a little and then falls down continuously except for some fluctuations. Here whether DIR will increase or decrease is determined by the data distribution in each sample and the features of string data. If $k$ rises, and then more elements are scanned, we either can access the elements with larger PDG values in the rear part of input data or cannot find any elements with a larger PDG value than the original $PDG_{\max}$. Thus, accordingly, the DIR value will either go up or fall down. Also, note that this variation tendency is the same with $\delta = 0.2$, $\delta = 0.5$ and $\delta = 0.8$.

As to the running time of a single experiment, we find that with $k$ getting larger, the running time will increase on the whole regardless of $\delta$ values, and this result can be obtained from Fig. 8(b). The reason for this phenomenon is that during the procedure of our proposed diversification framework, we have to first scan the first $k$ elements to find the $PDG_{\max}$ value. When $k$ increases, we spend more time calculating the PDG value of each element and pick out the maximal value among them. This is why the running time tends to become longer.

Fig. 8 Impact of scanned elements $k$ (string). (a) On DIR; (b) on running time

5.3.3 Impact of input elements number

In this section, we study the impact of input data scale $n$. As shown in Fig. 9(a), similar to the experimental results specific to numerical data, here as to strings, when $n$ increases, the DIR value has several rises and falls regardless of the val-
ues of $\delta$. This is because when $n$ changes, the element that we finally choose will change accordingly. The ability of various elements to diversify results is different and hence, DIR values will fluctuate.

![Figure 9](image-url) Impact of input size $n$ (string). (a) On DIR; (b) on running time

Similarly, when the value of $n$ increases, the running time of a single experiment tends to fluctuate. The detailed change of running time is illustrated in Fig. 9(b), with three different $\delta$ values. This is because the running time is determined by the exact number of elements to process and the data distribution in each selected sample. When $n$ changes, we may select various samples and thus, process different number of elements. All these factors contribute to the fluctuation of the running time.

5.3.4 Impact of cardinality of selected samples

Apart from all the factors discussed above, the sample size $a$ also has a significant impact on the effectiveness of our method, which is described as DIR values. Overall, the DIR value increases with the growing of $a$ except for some local fluctuations, whatever the value of $\delta$ is. This experimental result can be clearly observed in Fig. 10(a). The reason is that when the sample size gets larger, the number of elements that we can deal with in a single test will become larger. Then the probability of meeting an element with a larger PDG value and promoting the growth of DIR value is larger. Therefore, when we carry out a fixed number of sampling experiments, the DIR value becomes larger.

![Figure 10](image-url) Impact of sample size $a$ (string). (a) On DIR; (b) on running time

5.3.5 Impact of number of selected samples

When we vary the number of sampling experiments executed, i.e., $s$, the DIR values will vary. This is because the number of samples and the concrete content in the samples determines which element to be selected at last and how much diverse it can increase the result set by. Figure 11(a) intuitively demonstrates such experimental results.

Then when $s$ tends to increase, the running time of a single experiment will fluctuate. This is also because the running time is relevant to the data distribution in the selected samples and which samples we have selected. Figure 11(b) clearly illustrates the change of the running time with $s$ varying.

5.3.6 Discussion

Note that in all of our sampling experiments, the diversity increasing rate is greater than zero and large enough. As the DIR value describes the degree that the diversity of a set is increased, this result shows that our proposed method with regard to string data have satisfactory performance in diversifying the final result set.

As thoroughly discussed before, using a large number of experiments and corresponding experimental results, we can learn that the definitions, algorithms and methods presented are meaningful and effective in the diversification procedure.
of both numerical data and string data. Therefore, our methods contribute much to providing users with more information and improving users’ satisfaction.

### 5.4 Scalability experiments

In this part, we focus on the scalability of our proposed diversification framework, and study its ability to deal with big data. We respectively carry out the scalability tests of both numerical data and string data, and then study and analyse them thoroughly. We also compare our approach with MaxMin [7], MaxSum [7] and Swap [20]. MaxMin and MaxSum greedily insert the element maximizing the minimum diversity and average diversity in each iteration, respectively. For the sake of fairness, we use the MaxMin and MaxSum to find an element maximizing the minimum diversity and average diversity respectively in the following experiments. The Swap iteratively insert a new element into the result set and remove the element with minimum sum of distance to other elements. In our experiment, the algorithm Swap stops when the removed element is different from the inserted element. We compare our method with these methods since they are relatively recent algorithms without requiring prior knowledge of the query aspects. There are some more recent algorithms, however, we do not compare the performance of our method with them for the following reasons. The GNE [4] is reported to have low efficiency compared with Swap and MaxSum. Some explicit algorithms [14, 17] require prior knowledge of query logs. Some machine-learning algorithms like R-LTR and its variations [23–25] cost hours of running time.

In the following experiments, we use the running time as the metric of the efficiency and regard the diversity increasing rate (DIR) defined in Section 5.1 as the metric of effectiveness.

#### 5.4.1 Scalability experiments of numerical data

When we try to carry out the scalability test of numerical data, which is in the form of double values here, we use randomly generated data which is uniformly distributed in \([-1000, 1000]\). Additionally, the utilized data amount varies from 1.0 GB to 5.0 GB in order to test how our method works when dealing with big numerical data. Note that only \(n\) varies here, other factors’ values are set as constant values: \(m = 10, k = 20, a = 150\). Figure 12 indicates the performance of the four algorithms on numerical data. The “Proposed” in these figures means the method we proposed in this paper.

Figure 12(a) illustrates the relationship between the data amount and the running time. It is intuitive from Fig. 12(a) that the running time of method we proposed is irrelevant with the data amount and the running time is less than 0.045 seconds. This means that we can always obtain the diversification results within a reasonable period of time and the performance of our proposed framework is quite satisfactory in terms of processing time when dealing with big numerical data. Therefore, our framework has a good ability to diversify big numerical data. Figure 12(b) indicates the relationship between the data amount and the diversity increasing rate DIR. We can learn from Fig. 12(b) that the DIR of our method is comparable to MaxMin and MaxSum and much higher than that of Swap.

From comparison, the efficiency of our method outperforms MaxMin and MaxSum. It is because we use effective sampling strategy and reduce the computation cost. The running time of our method is similar to that of Swap, however, we can learn from Fig. 12(b) that the DIR of Swap is much lower than our method. That is, our proposed method on numerical data is efficient, meanwhile, it provides effectiveness comparable to other methods.

#### 5.4.2 Scalability experiments of string data

In this part, we analyse the scalability of our proposed framework when dealing with big data in string type. The data set is randomly generated string data which can be composed of 1 to 10 characters from \{a, b, ..., z\} \cup \{A, B, ..., Z\} and follows uniform distribution. The amount of the data used ranges from 1.0 GB to 5.0 GB. In this way, we can analyse
the scalability of our proposed method when processing big data in string type. Note that only $n$ changes here, so other parameters are set as constant values: $m = 10$, $k = 20$, and $a = 200$. Figure 13 indicates the performance of the four algorithms on string data.

![Figure 12 Scalability experiments on numerical data. (a) Efficiency (numerical data); (b) effectiveness (numerical data)](image1)

![Figure 13 Scalability experiments on string data. (a) Efficiency (string data); (b) effectiveness (string data)](image2)

Figure 13(a) illustrates the relationship between the data amount and the running time. It can be seen from the figure that the running time has nothing to do with the data amount and is no more than 0.065 seconds. This scalability test shows that our proposed method is qualified in effectiveness and efficiency when processing big data in string type. That is to say, our proposed diversification framework is capable of increasing the diversity of resulted string results when faced with big data. Figure 13(b) indicates the relationship between the data amount and the diversity increasing rate $DIR$. We can learn from Fig. 13(b) that the $DIR$ of our method is comparable to $MaxMin$ and $MaxSum$ and much higher than that of $Swap$.

The efficiency of our method on string data still outperforms $MaxMin$ and $MaxSum$. It is because we use effective sampling strategy and store intermediate result to avoid calculating too many edit distances. The running time of our method is similar to that of $Swap$, however, we can learn from Fig. 13(b) that the $DIR$ of $Swap$ is much lower than our method. That is, our proposed method on string data is efficient, meanwhile, it provides effectiveness comparable to other methods.

5.5 Cache experiments

Our diversification algorithm can only replace one element into memory to diversify the final result set. In this experiment, we consider to store the elements that can increase the diversity in a cache and replace these elements into the memory together at last. In this way, the diversity of the final result set increases to a larger extent.

We use the numerical and string data sets same with those in the scalability experiments, and adopt the LRU Cache in our experiment. We regard the index of the elements in the memory and the index of the candidate replacing element in the data set as the $Key$ and $Value$ of the cache, respectively. The parameters are set as constant values: $m = 20$, $k = 20$, and $a = 200$. The size of cache varies from 1 to 5. We can learn from Fig. 14 that the diversity increasing rate $DIR$ increases with the size of cache on both numerical and string data. That is, involving a cache in our algorithm can increase the diversity of the result set.

6 Conclusions

In this work, we firstly propose a diversification framework which can solve the challenges that existing diversification methods are faced with when dealing with big data. This framework processes input data online and the computational overhead as well as space overhead are low. Next, we concretely implement this framework in the area of two
commonly-used data types: numerical data and string data and then design corresponding implementation algorithms. Finally, we carry out extensive experiments on real data to evaluate our proposed framework. Additionally, scalability experiments are conducted on synthetic data to evaluate the ability of the framework to process big data and demonstrate that the proposed approach outperforms the existing linear approach due to the randomized strategy.

Fig. 14 Cache experiments. (a) Impact of cache (numerical data); (b) impact of cache (string data)

Our proposed diversification framework can only replace one element into memory to diversify the final result set. However, if we can use a cache to store elements which can be replaced into memory together, we can increase the diversity of the final result set to a larger extent. This is an interesting yet challenging direction for further research.

Acknowledgements This paper was partially supported by NSFC (Grant Nos. U1509216, U1866002, 61602129) and Microsoft Research Asia.

References
1. Drosou M, Pitoura E. Search result diversification. Special Interest Group on Management of Data Record, 2010, 39(1): 41–47
2. Drosou M, Jagadish H V, Pitoura E, Stoyanovich J. Diversity in big data: a review. Big Data, 2017, 5(2): 73
3. Angel A, Koudas N. Efficient diversity-aware search. In: Proceedings of the ACM SIGMOD International Conference on Management of Data. 2011, 781–792
4. Vieira M R, Razente H L, Barioni M C, Hadjieleftheriou M, Srivastava D, Jr C T, Tsotras V J. On query result diversification. In: Proceedings of International Conference on Data Engineering. 2011, 1163–1174
5. Agrawal R, Gollapudi S, Halverson A, Ieong S. Diversifying search results. In: Proceedings of the 2nd International Conference on Web Search and Web Data Mining. 2009, 5–14
6. Ashkan A, Kveton B, Berkovsky S, Wen Z. Optimal greedy diversity for recommendation. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence. 2015, 1742–1748
7. Gollapudi S, Sharma A. An axiomatic approach for result diversification. In: Proceedings of the 18th International Conference on World Wide Web. 2009, 381–390
8. Zhang M, Hurley N. Avoiding monotony: improving the diversity of recommendation lists. In: Proceedings of ACM Conference on Recommender Systems. 2008, 123–130
9. Liu K, Terzi E, Grandt T. Highlighting diverse concepts in documents. In: Proceedings of the SIAM International Conference on Data Mining. 2009, 545–556
10. Sarma A D, Gollapudi S, Ieong S. Bypass rates: reducing query abandonment using negative inferences. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2008, 177–185
11. Wu T, Chen L, Hui P, Zhang C J, Li W. Hear the whole story: towards the diversity of opinion in crowdsourcing markets. Proceedings of the VLDB Endowment, 2015, 8(5): 485–496
12. Clarke C L, Kolla M, Cormack G V, Veitch D, Ashkan A, Butcher S, MacKinnon I. Novelty and diversity in information retrieval evaluation. In: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 2008, 659–666
13. Zhang Y, Callan J P, Minka T P. Novelty and redundancy detection in adaptive filtering. In: Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 2002, 81–88
14. Santos R L, Macdonald C, Ounis I. Exploiting query reformulations for web search result diversification. In: Proceedings of the 19th International Conference on World Wide Web. 2010, 881–890
15. Ozdemiray A M, Altingovde I S. Explicit search result diversification using score and rank aggregation methods. Journal of the Association for Information Science and Technology, 2015, 66(6): 1212–1228
16. Carbinell J, Goldstein J. The use of MMR, diversity-based reranking for reordering documents and producing summaries. Special Interest Group on Information Retrieval Forum, 2017, 51(2): 209–210
17. Capannini G, Nardini F M, Perego R, Silvestri F. Efficient diversification of web search results. Proceedings of the VLDB Endowment, 2011, 4(7): 451–459
18. Ziegler C, Mcnee S M, Konstan J A, Lausen G. Improving recommendation lists through topic diversification. In: Proceedings of the 14th International Conference on World Wide Web. 2005, 22–32
19. Radlinski F, Dumais S T. Improving personalized web search using result diversification. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information
20. Yu C, Lakshmanan L V, Ameryahia S. It takes variety to make a world: diversification in recommender systems. In: Proceedings of the 12th International Conference on Extending Database Technology. 2009, 368–378

21. Vee E, Srivastava U, Shanmugasundaram J, Bhat P, Yahia S A. Efficient computation of diverse query results. In: Proceedings of the 24th International Conference on Data Engineering. 2008, 228–236

22. Drosou M, Pitoura E. Diverse set selection over dynamic data. IEEE Transactions on Knowledge and Data Engineering, 2014, 26(5): 1102–1116

23. Zhu Y, Lan Y, Guo J, Cheng X, Niu S. Learning for search result diversification. In: Proceedings of the 12th International Conference on Extending Database Technology. 2009, 368–378

24. Vee E, Srivastava U, Shanmugasundaram J, Bhat P, Yahia S A. Efficient computation of diverse query results. In: Proceedings of the 24th International Conference on Data Engineering. 2008, 228–236

25. Xia L, Xu J, Lan Y, Guo J, Cheng X. Learning maximal marginal relevance model via directly optimizing diversity evaluation measures. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2015, 113–122

26. Erkut E, Ülküsal Y, Yeniçerioglu O. A comparison of $p$-dispersion heuristics. Computers & Operations Research, 1994, 21(10): 1103–1113

27. Baryossef Z, Jayram T S, Kumar R, Sivakumar D, Trevisan L. Counting distinct elements in a data stream. In: Proceedings of International Workshop on Randomization and Approximation Techniques in Computer Science. 2002, 1–10

28. Cormen T H, Leiserson C E, Rivest R L, Stein C. Introduction to Algorithms. 2nd ed. Cambridge: The MIT Press and McGraw-Hill Book Company, 2001

29. Mitzenmacher M, Upfal E. Probability and Computing - Randomized Algorithms and Probabilistic Analysis. Cambridge: Cambridge University Press, 2005

30. Hadjieleftheriou M, Li C. Efficient approximate search on string collections. Proceedings of the VLDB Endowment, 2009, 2(2): 1660–1661

Meifan Zhang received the bachelor’s degree in computer science from Harbin Institute of Technology, China in 2014, where she is currently pursuing the PhD degree. Her research interest includes big data analytics, data quality and machine learning.

Hong Gao is a professor and doctoral supervisor of Harbin Institute of Technology, China. She received her PhD degree from Harbin Institute of Technology, China. She has long engaged in research work of massive data computation and quality management, wireless sensor networks and graphic data management and computation. She was a recipient of awards and honors, including the Assistant Director of the China Computer Federation Technical Committee on Databases, a member of the China Computer Federation Technical Committee on Sensor Network, and the Deputy Director of the Massive Data Computing Lab.

Hong Gao is a professor and doctoral supervisor of Harbin Institute of Technology, China. She received her PhD degree from Harbin Institute of Technology, China. She has long engaged in research work of massive data computation and quality management, wireless sensor networks and graphic data management and computation. She was a recipient of awards and honors, including the Assistant Director of the China Computer Federation Technical Committee on Databases, a member of the China Computer Federation Technical Committee on Sensor Network, and the Deputy Director of the Massive Data Computing Lab.

Hong Gao is a professor and doctoral supervisor of Harbin Institute of Technology, China. She received her PhD degree from Harbin Institute of Technology, China. She has long engaged in research work of massive data computation and quality management, wireless sensor networks and graphic data management and computation. She was a recipient of awards and honors, including the Assistant Director of the China Computer Federation Technical Committee on Databases, a member of the China Computer Federation Technical Committee on Sensor Network, and the Deputy Director of the Massive Data Computing Lab.

Jianzhong Li received the BS degree from Heilongjiang University, China in 1975. He worked in the University of California at Berkeley as a visiting scholar in 1985. He has also been a visiting professor at the University of Minnesota at Minneapolis, USA, from 1991 to 1992 and from 1998 to 1999. Since 1998, he has been a professor of Department of Computer Science and Technology, Harbin Institute of Technology, China. His current research interests include database management systems, data warehousing and data mining, sensor network, and data intensive super computing. Prof. Li was a recipient of awards and honors, including the Chairman of the ACM SIGMOD China and the Director of the China Computer Federation.

Hong Gao is a professor and doctoral supervisor of Harbin Institute of Technology, China. She received her PhD degree from Harbin Institute of Technology, China. She has long engaged in research work of massive data computation and quality management, wireless sensor networks and graphic data management and computation. She was a recipient of awards and honors, including the Assistant Director of the China Computer Federation Technical Committee on Databases, a member of the China Computer Federation Technical Committee on Sensor Network, and the Deputy Director of the Massive Data Computing Lab.