Accelerating Data Mining on Incomplete Datasets by Bitmaps-based Missing Value Imputation

Sameh Shohdy, Yu Su, Gagan Agrawal
Computer Science Department, The Ohio State University
Columbus, Ohio, USA, 43210
Email: {ahmedsa,su1,agrawal}@cse.ohio-state.edu

***Abstract*—Among all 'big data' research issues, the veracity challenge, which refers to the precision and accuracy of the data, has not received as much attention. Traditionally, it has been well known that problems related to data quality, such as incomplete, redundant, inconsistent, and noisy data pose a major challenge to data mining and data analysis. Particularly, we note that existing methods for handling missing values cannot scale to larger datasets. In other words, this particular veracity challenge has been addressed, but not in context of also handling volume (and possibly the velocity) challenge of 'big data'. This paper focuses on speeding up the missing values imputation process using the bitmap indexing technique. The research takes two directions: first, the bitmap indexing is used to directly access the required records for the imputation method (i.e., Direct Access Imputation (DAI)). Second, the bitmap indexing technique is used for missing value estimation using the pre-generated bitmap indexing vectors without accessing the dataset itself (i.e., Bitmap-Based Imputation (BBI)). Both approaches have been evaluated using different real and synthetic datasets, and four common imputation algorithms. We show how our bitmap-based methods can accelerate data mining classification of incomplete data while also maintaining precision.**

***Keywords*—Missing Values; Bitmap Indexing; Indexing as a Service.**

I. Introduction

In recent years, 'big data' has become one of the most important challenges in computing research. One of the most popular definitions of 'big data' involves four challenging aspects of dealing with data - *volume*, *velocity*, *variety*, and *veracity*. There has been a lot of work in recent years on addressing these challenges, with the volume challenge, and to a less extent, the velocity challenge, receiving most interest. Work on MapReduce [1] and related framework for handling massive data, as well the work on handling streaming data and *in-situ* data analysis have been topics of much investigation.

The *veracity* challenge, which refers to the *precision* and *accuracy* of the data, has not received as much attention. Traditionally, it has been well known that problems related to data quality, such as incomplete, redundant, inconsistent, and noisy data [2] pose a major challenge to data mining and data analysis. In fact, one of the most important steps in data mining is considered to be the *data preparation* step, which is the process of ensuring the quality of data by changing the original data into a suitable format for the analysis process. About 60% of data mining time is consumed in the data preparation process compared to only 20% in the actual data mining process [3].

Unfortunately, recent work on MapReduce and other 'big data' frameworks has not emphasized the challenges of data preprocessing, and particularly, the difficulty of applying known techniques on large-scale data. Consider the problem associated with handling Missing Values (MVs). Incomplete dataset or missing values can impact data analysis tasks. To avoid their negative impact, a popular approach is to *fill* the missing values with estimated values that can be calculated from the complete records of the same dataset [4]. Several studies have illustrated different methods for dealing with missing values, i.e., using the complete set of records to *impute* the missing values in the real datasets. Clark and Niblett explained a simple algorithm (i.e., the CN2 algorithm) for imputing missing values using the most common value of the same attribute [5]. A modified version of this method is the most common value of the missing value attribute restricted to a label (or concept) [6]. Another study performed by Grzymala-Busse suggested replacing the missing value with all the possible values that appears at the same attribute in all the dataset records [7]. Another approach called Global Closest Fit has also been proposed [8], where missing value in a specific record is taken from the corresponding values from the record that is most similar to this record in the entire dataset. Moreover, different studies have surveyed and compared several approaches for missing values imputation [8]–[12].

A common theme in all of the works in this area is that algorithms for imputing missing value cannot scale to larger datasets. In other words, this particular *veracity* challenge has been addressed, but not in context of also handling volume (and possibly the velocity) challenge of 'big data'. This paper addresses this shortcoming of existing work, and focuses on improving the scalability of methods. We achieve this by using indexing to accelerate missing value imputation - particularly, we use bitmap indexing, which can serve as a summary of datasets. A key advantage of this technique that low-cost bit-wise operations can be exploited for processing. In this work, we show how various missing value imputation methods can be accelerated using indexing, at two levels. First, like any indexing technique, bitmap indexing can be used to look-up relevant records, and thus, complete scans over the entire dataset are avoided. Second, one can use bitmaps as a summary of the entire dataset, and not access the dataset at all. These two methods are referred to as Direct Access Imputation (DAI), and Bitmap-Based Imputation (BBI), respectively. We evaluate both approaches extensively using different real and synthetic datasets, and four different imputation algorithms.

The rest of paper is structured as follows: In Section 2, a brief background on existing imputation methods and bitmap indexing is introduced. In Section 3, we present our proposed bitmap-based algorithms in detail. In Section 4, we present the performance evaluation of the proposed algorithms. Finally, we draw our conclusions in Section 5.

II. Background

This section provides key background on two important topics. The first is the existing algorithms for dealing with missing values, i.e., specifically the imputation methods. The second is bitmaps-based indexing.

A. Imputation Methods

The goal of any imputation algorithm is to estimate the missing value in a specific record using others complete records in the same dataset. As we stated earlier, this has been an active area of research and much work has been done. We give an overview of four existing algorithms.

1) Global Closest Fit Method (GCF): The main idea in this method is to replace the missing value by the value of corresponding attribute from a single record that is the most similar record to the record with the missing value. The most similar record is the one with the smallest *distance* from the current record, with the distance calculated as follows:

 ISBN: 978-1-61208-408-4

$$Distance(X, Y) = \sum D(x_i, y_i) \quad (1)$$

$$D(x_i, y_i) = \begin{cases} 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \text{ is missing} \\ & \text{or } y_i \text{ is missing} \\ \frac{|x_i - y_i|}{r} & \text{if } x_i \neq y_i \end{cases} \quad (2)$$

Equation (1) is the general form for calculating the difference between any two records X and Y as a summation of the distance between values in each attribute for both records. Equation (2) can be used to calculate the distance between the two values for the same attribute in both X and Y records, where r is difference between maximum and minimum values of this attribute for the entire dataset – this corresponds to using what is referred to as the Manhattan distance, a commonly used metric [10].

2) Most Common Value (MCV): This is a simple method where the missing value can be replaced by the most common value in the missing value attribute. By scanning the entire dataset, we can find the single value of the attribute that occurs most frequently [2] [13].

3) Concept-based Most Common Value (CMC): The Concept-based Most Common Attribute Value method is a special case of the most common value method. This method assumes that records have labels associated with them. Using this information, the missing value is replaced by the most common value for the specific attribute, among the set of records with the same label [13].

4) Concept-based Mean Attribute Value (CMAV): The missing attribute value is replaced with the mean of values of the same attribute, among the records having the same concept. A further generalization of this method is the mean attribute value method, which is used to impute the missing value as the mean of all values in the same attribute, i.e., ignoring any possible labels [8] [11].

B. Bitmap Indexing Method

The implementations of each of the above techniques in the literature assume a small dataset that fits in memory, and involves simple scan(s) on all records to find values to substitute missing values. These implementations clearly have limitations while dealing with large datasets. Thus, we examine the use of indexing as a mechanism to accelerate imputation methods we have described above. There are several reasons for considering this approach. First, indexing is already implemented in most data stores that are handling large-scale data. Second, intuitively, it is easy to see that all or most of the methods described above can be accelerated using indexing support.

The particular indexing method we have chosen is bitmap indexing [14]–[16]. This section gives a brief overview of bitmap indexing and the reasons behind choosing it specifically to improve the performance of existing missing values imputations techniques.

In the simplest form, a single bit-vector is generated for each distinct value *v* in each attribute. The length of the bit-vector is equal to the number of data records in the dataset. If the value of the particular attribute in a record matches the value *v* this bit-vector corresponds to, the bit is set to 1, and is 0 otherwise. However, large number of distinct values for an attribute negatively impacts the bitmap indexing performance, because large number of bit-vectors need to be generated to cover all these distinct values. A typical solution for this challenge is to use the binning process. In the binning process, the attribute values are binned so that each bin represents a range of values [15] [17]. Table I illustrates an example of using bitmap indexing technique with/without binning process.

Bit-vectors can become extremely space consuming in the original form. Thus, several compression methods have been proposed to solve the space problem, which includes Byte-aligned Bitmap Code [18], Word-Aligned Hybrid code [19], Partitioned Word-Aligned Hybrid (PWAH) [20], and Position List Word-Aligned hybrid [21]. The main idea in these methods is to encode a possible large series of contiguous 0s (or 1s, though they are less likely) before storing them.

Using the bitmap indexing, a typically subsetting query can be processed by extracting a set of bit-vectors depending on the query conditions, and then the result can be exported by performing bit-wise AND and OR operations over these bit-vectors, which are normally supported very efficiently in the hardware [22].

TABLE I. AN EXAMPLE OF BITMAP INDEXING

Temp.	**Bitmap Indices**								
	Without Binning						With binning		
	23	25	28	30	33	35	23-25	28-30	33-35
23	1	0	0	0	0	0	1	0	0
25	0	1	0	0	0	0	1	0	0
28	0	0	1	0	0	0	0	1	0
28	0	0	1	0	0	0	0	1	0
30	0	0	0	1	0	0	0	1	0
25	0	1	0	0	0	0	1	0	0
23	1	0	0	0	0	0	1	0	0
33	0	0	0	0	1	0	0	0	1
33	0	0	0	0	1	0	0	0	1
35	0	0	0	0	0	1	0	0	1

There are several reasons for choosing bitmap to improve the imputation process. First, bitmap treats each attribute separately, and thus, is likely to support efficiently index for incomplete datasets [23]. In comparison, other popular indexing methods such as B-tree, B+-tree, and R-tree are designed to support queries on one or two attributes. Second, bitmap indices can be built on top of an existing data store, i.e, they do not require that data be reorganized under the index. Such data reorganization is extremely time consuming and may not be justified if the only or the major goal of indexing could be to support imputation of missing values. Third, missing values imputation operation is an approximate process. Thus, bitmap-based missing values imputation algorithms can be accelerated using bitmap approximation techniques, such as binning without loss of precision. Finally, during data mining, we typically target datasets that are not updated, and bitmaps are well suited for such datasets [24].

III. Proposed Imputation Methods

Our goal is to accelerate different imputation algorithms using bitmap indices. It turns out that there are two distinct ways in which bitmaps can be used for the imputation process, with trade-off between accuracy and processing speed. First, search for records that are similar to the record with missing value, or records with a particular label can be accelerated using bitmaps. Thus, a scan on the entire dataset can be replaced by look-up on specific parts of the dataset. Second, it is possible to use bit-vectors as an approximate summary of the entire dataset, and provide approximate answers using this information. We refer to the first strategy as the *Direct Access Imputation* method or DAI, whereas the second method is called *Bitmap-Based Imputation* method or BBI. Before discussing the approaches, we note that bitmaps have been used to support query processing in several systems [25]–[27]. Thus, if bitmaps are already being built to support query processing, implementing imputation method using them will not even involve additional overhead of index generation.

A. Direct Access Imputation (DAI)

Several imputation algorithms can be accelerated if we can directly access the required records to impute each missing value without full database scan. This is the idea of the DAI approach. We show how

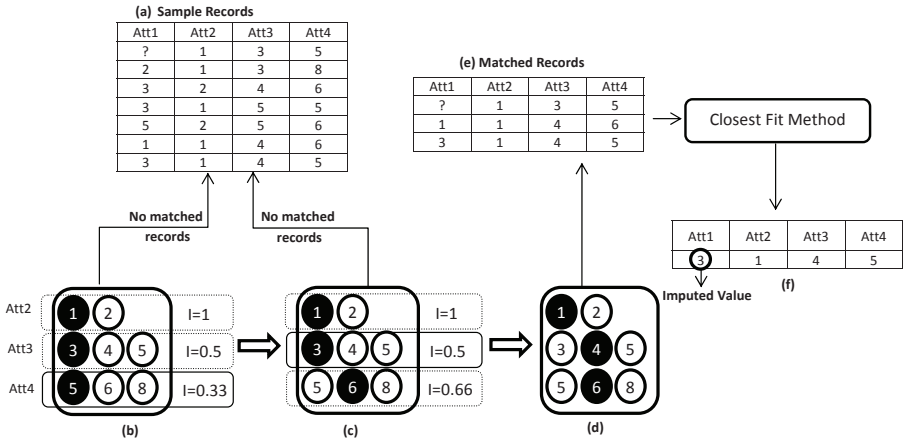


Fig. 1. Global Closest Fit Method using Bitmap Indexing - Impact Value of Each Attribute is Used to Select Records

to apply the DAI method in the case of three imputation algorithms: Global Closest Fit, Concept-based Most Common Attribute Value and Concept-based Mean Attribute Value , each of which was introduced in Section II.

1) Global Closest Fit Method: Recall that this method imputes the missing value from the record(s) that are closest to the record with the missing value, in the sense of the distance between the two records. The brute-force approach to implement this method involves retrieving the missing record and computing the distance between this record and all other records (i.e., Full Dataset Scanning method). By using bitmaps, we can retrieve only the set of records that have attributes values close to the target record values.

In performing a full-scan of the dataset, it is relatively easy to select the closest match with respect to other attributes. However, using an indexing method to find the closest match requires certain challenges. We need to keep identifying neighboring values of each attribute, so as to find the closest match or a good approximation. To enable the selection process, we use the notion of *impact* of a neighboring value of an attribute to the distance. The *impact* I of the next value v for each attribute A can be computed as: $I(A,v) = \frac{v-c}{r}$, where c is the current used value for attribute A and r is the difference between the maximum and minimum value for attribute A. The idea is that neighboring values of each attribute can contribute differently to the impact with respect to the distance. Records that have a neighboring value with the smallest *impact* should be searched first, because they will have a smaller distance.

An example of our approach is given by Figure 1 to solve this challenge. Subfigure (a) represents a set of records from any given dataset that has a missing value marked by *"?"*. Suppose that there are no records that have the same value for all other attributes as the record with the missing value. A second search process is conducted by selecting the next closest value (as recorded in a metadata file) for each attribute, and choosing the attribute and its value with the least *impact* – for example, $att4$ has the smallest *impact* as shown in sub-figure (b). In subfigure (c), we now need to retrieve records where the corresponding value is less than or equal to this particular value for the chosen attribute $att4 = 6$ and larger that the old value $att4 = 5$, while having the same value for the remaining attributes that were used for the previous search. If there are no matching records identified, the process of picking a new attribute using the *impact* value is repeated (see sub-figures (c) and (d)). Once the result bit-vector does retrieve a set of records, e.g., as shown in (e), these records should be examined, with the goal of obtaining the most closest fit record(s) to the record with the missing value. This last step is shown through the sub-figure (f).

The implementation of our approach has been in the context of a particular bitmap-based data management system [27]. This system supports database-like query subsetting over data in scientific data formats like NetCDF. The query processing module converts the user query request into a set of query conditions where each condition can be represented by a single bit-vector (i.e., the *condition bit-vector*).

Returning to the example shown in Figure 1, the challenge here is how to build a query that retrieves the records that are most closest to the record with the missing value in each step. For example, in sub-figure (a), we need to retrieve all records that exactly match the record with missing value. In this case, each distinct attribute query condition will be built to cover all the records in a range with the maximum and minimum values equalling the corresponding attribute value in the missing value record. Further, in sub-figure (b), we need to increase the internal range in $att4$ to cover the records that can be used in the imputation process. So, the query should be built to have a maximum and minimum values related to $att4$, i.e., $att4 > 5$ AND $att4 \leq 6$.

Algorithm 1 BuildQueryforGCF($record$)

```
1: for att= 1 → sizeof(record) do
2:     vals[att]=ReadValues(MetaDataFile,att)
3:     query_vals[att]= record[att];
4: end for
5: resultbitvector:= null
6: while resultbitvector is null do
7:     for att= 1 → sizeof(record) do
8:         min_diff:= ∞
9:         for i= 1 → sizeof(vals[att]) do
10:            diff=abs(vals[att].get(i)-query_vals[i])
11:            if diff < min_diff then
12:                min_diff=diff
13:                r=Max(att) − Min(att)
14:                impact[att]=min_diff/r
15:                candidate_vals[att]=vals[i]
16:            end if
17:        end for
18:    end for
19:    selected_att=PickMin(Impact)
20:    query_vals[selected_att]=candidate_vals[selected_att]
21:    vals[selected_att].remove(selected_value)
22:    query= GenerateQuery(record,att,query_vals)
23:    resultbitvector=RetrieveIndex(record,query);
24: end while
```

Fig. 2. Algorithm 1: Query Building Operation

The query building operation algorithm is formally described by Figure 2. The algorithm takes as input the record with missing value. In the line 2, distinct values for each attribute are read from its $metadata$ file into $vals$ vector. Line 3 initiates the $query\_vals$ array with the values of missing value record. Through the algorithm, for each attribute, all records that have a value between both $query\_vals$ value and $record$ value should be retrieved. The idea is to adapt the $query\_vals$ array values until the $resultbitvector$ retrieved any number of records. In lines 9-17 , for each attribute's distinct value, the impact of changing the old value in $query\_vals$ with this value is computed to determine the value of each attribute that makes the smallest change in the $impact$ value. In the line 19, $PickMin$ method is used to determine the attribute that with the smallest $impact$ value. In the line 20, The value of this attribute replaces the old value in the $query\_vals$ array. In line 21, this value is removed from $vals$ vector to guarantee a new value is selected the next time. In line 23, the query was generated using the missing value's $record$ and the $query\_vals$ array. If the $bitmapvector$ has no records, a next iteration is required.

The same strategy can be applied if binning process is used. The operation of changing a proper attribute value depending on the impact degree on the distance can be applied to the binned values by selecting a proper binned range of values rather than a single value. In this case, the impact degree can be computed by using the average value of the current range c and the average value of the candidate range v.

2) Concept-based Most Common Value Method: The method can be implemented using bitmaps as follows. After retrieving all missing values, a single query request is built for each record R that contains a missing value. The goal of the query will be to retrieve all the records that contain the same label as the record R. These retrieved records can be then be checked to determine the most common value for the attribute where the record R has a missing value. These operations are trivial to perform if binning is not used.

However, in the case of binning, for each missing value, the retrieved records will be the records that contain a label in the same binned range, as the label associated with the record R. For the last step, we can determine the most common bin, and then proceed to search for the most common value. Note that this may result in a different value, as compared to the value obtained by a brute-force algorithm. This is because the most popular bin may not contain the most commonly occurring value.

Note that implementation of most common value is also very similar - we simply focus on finding the most common value of the attribute involved across all records.

3) Concept-based Mean Attribute Value Method: Similar to the previous method, a single query request is built to retrieve all the records that have the same label as the record R that has a missing value. These records can be accessed directly and all the values of the attribute where R has a missing value can be retrieved into the memory. The estimated value is the mean of all retrieved values.

In the case of binning, certain calculations are required. The label attribute's metadata file should be checked to determine the binned range that contains the target record's label (i.e., the record with the missing value). Then, a simple query request is built to retrieve all the records within this binned range. These records are examined again to determine the records with the same label as the missing value record. The imputed value will be considered as the mean of all the values of the missing value attribute.

B. Bitmap-Based Imputation (BBI)

This approach involves imputing the missing values directly using the bitmap vectors, i.e., without accessing the dataset itself. We will explain the use of BBI approach in three imputation algorithms: Most Common Value, Concept-based Most Common Value, and the Concept-based Mean Attribute Value methods. An example of using the BBI methods in different imputation algorithms is given by Figure 3. The example uses a dataset with three attributes and a label.

1) Most Common Value Method: In this method, the key is to determine the number of records for each distinct value in the target attribute (i.e., attribute with missing value). The value with the largest number of records is the most common value. The bitmap bit-vectors can be used to determine the most common value in the following manner. First, distinct attribute values can be retrieved from the missing value attribute's metadata file. Second, for each distinct value, the related bitmap vector is retrieved from the index file. It is easy to count the number of records for each distinct value by counting the number of ones on each retrieved bit-vector. The bit-vector with the largest count of ones represent the bit-vector of the most common value in the target attribute.

Figure 3 (c) gives an example of retrieving each distinct value's bit-vector and the bit-vector with the largest number of ones reflect the most common value. However, the BBI approach cannot be used without approximation in the case of binning, because each bit-vector represents a set of values rather than a single value.

2) Concept-based Most Common Value Method: First, a query condition bit-vector is constructed to retrieve all the records with the same label as the missing value record's label. Second, for each distinct value in the target attribute's metadata file, a query request is built and passed to the bitmap query system to retrieve a bit-vector representing all the records equal to this value. A logic AND operation will be performed between all distinct values' bit-vectors and the label bit-vector. The number of ones per each distinct value's bit-vector is counted which represents the number of records with the same value and label. The value's bit-vector with the largest number of ones can be used as the imputed value for the missing value.

Similar to the most common value method, this strategy becomes less accurate in the case of using binning when constructing the bitmap index. Figure 3 (d) gives an example of using BBI approach with Concept-based Most Common Value algorithm.

3) Concept-based Mean Attribute Value Method: Similar to the previous method, a single condition bit-vector will be constructed to retrieve all the records that have a label matches with the label of the missing value record. For the missing value attribute, the imputed value is the mean of all the target records values of this attribute. In the case of binning the technique is changed. First, the label attribute's metadata file needs to be checked to determine the binned set that contains the missing value record's label. Second, we construct a query condition bit-vector to retrieve all the records within this set. These records can be checked to determine the records that have the same label of the missing value record. The imputed value will be the mean value for all the records with the same missing value record label. Figure 3 (e) also gives an example of this method. In the case of binning, the technique is changed. First, the label attribute's metadata file needs to be checked to determine the binned set that contains the missing value record's label. Second, we construct a query condition bit-vector to retrieve all the records within this set. These records can be checked to determine the records that have the same label of the missing value record. The imputed value will be the mean value for all the records with the same missing value record label.

IV. Experimental Evaluation

This section presents results from a series of experiments done to evaluate the proposed bitmap-based imputation approaches. Specifically, our proposed DAI and BBI imputation approaches are compared with the brute-force FSI imputation approach in two ways: execution speedup and imputed values' accuracy. In accuracy comparison, to focus on the impact of different approaches to missing value imputation on the data mining process, we used a popular classification algorithm (i.e., the CPAR algorithm [28]) to classify the dataset records after the missing values imputation process is applied using different approaches (i.e., FSI, DAI, and BBI), then the dataset noise ratio is evaluated in each case.

 ISBN: 978-1-61208-408-4

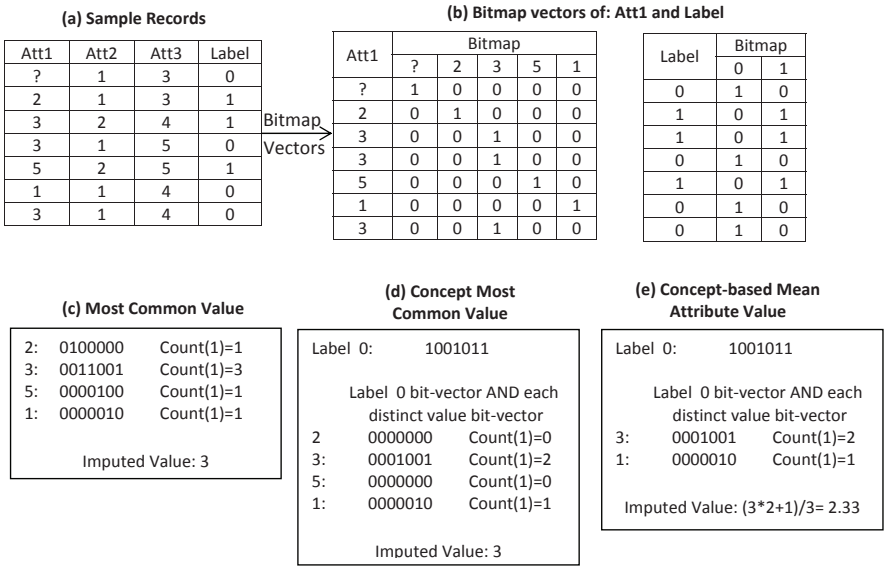


Fig. 3. Bitmap-based Imputation Method Example

All approaches were implemented in C++. The experiments were performed on Linux Red Hat 4.4.7-3 machine equipped with 2 quad-core Intel CPUs running at 2.53 GHz and with a 4 GB RAM.

A. Execution Speed Comparison

Execution speed experiments have been performed using two different datasets. The first dataset is what we refer to as the Berkeley dataset [29] (∼1GB), which is an earth surface temperature dataset. This dataset comprises seven attributes and nearly $3 \cdot 10^7$ records, and can fit into memory, and thus helping us evaluate the performance of the proposed approaches when datasets fit in memory. Second, we used a synthetic dataset (∼10GB), which was generated with nine numerical attributes and a label, and nearly $20 \cdot 10^7$ records. This dataset is used as an example of a large dataset, helping us evaluate the performance of proposed imputation methods on disk-resident datasets.

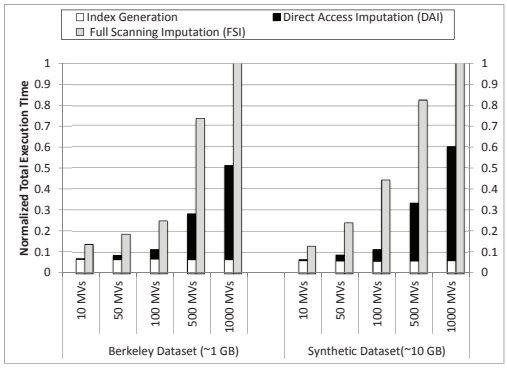


Fig. 4. Time Comparison between DAI and FSI Based Global Closest Fit Methods Using Different Numbers of Missing Values (MVs)

1) Global Closest Fit (GCF): Only DAI and FSI methods have been used to implement the Global Closest Fit imputation algorithm, as BBI cannot be applied. This experiment targets both datasets (i.e., Berkeley and the Synthetic dataset).A comparison between the execution speed in handling datasets' missing values on the GCF method with different number of missing values is shown in Figure 4. The number of missing values in the dataset varies from 10 to 1000, and they are distributed across different attributes. Note that our comparison is assuming that an index is generated specifically for the imputation process. However, in practice, an index may be generated in advance for a different reason, like need for supporting subsetting (or other types of) queries, and index generation time may be amortized. Returning to the Figure, the x-axis shows different number of missing values in both datasets while the y-axis shows the *normalized execution time*, which is computed as the fraction of the longest execution time. We can draw from the figure the following three observations. First, the total execution time of DAI method, even including the required time to generate index files for the target dataset, is less than the total execution time of the brute-force FSI method on both datasets. Second, with increasing number of missing values, the DAI's advantage becomes even more significant, when compared to the FSI method. Finally, the figure shows that the DAI method's relative performance is even better in the case of disk-resident datasets. The reason is as follows: the imputation time for a single value is the sum of the times for finding the most closest records to the missing value's record and the required time to retrieve these records, whether from the memory or the disk. In the case of the Berkeley dataset, the entire dataset fits into the memory, which means that the total execution time depends primarily on the time to find the most similar records. However, for a large dataset, disk operations are required to retrieve records from disk to memory.

2) Most Common Value (MCV): For the Most Common Value algorithm, Figure 5 shows the normalized total execution time of the algorithm using both BBI and FSI methods. Because all the missing values of the same attribute will be replaced by the most common value for this attribute, the performance with this algorithm depends on the number of attributes containing missing values, rather than the number of missing values itself. As we can see, BBI speeds up the imputation process compared to the brute-force FSI method. The BBI method depends only on the pre-generated bitmap vectors, so the increasing of dataset size does not affect the performance of the BBI method, unlike the FSI or the DAI methods. In other words, the performance gap between the total execution time in the FSI method and the BBI method increases as we consider a bigger dataset. However, BBI method becomes worse when the number of distinct values in the target attribute increase. In this case, the number of generated bitmap vectors increase which negatively affect the performance of the BBI method.

3) Concept-based Most Common Value (CMC): The Concept-based Most Common Attribute Value method can be implemented using both DAI and BBI methods. The normalized total execution

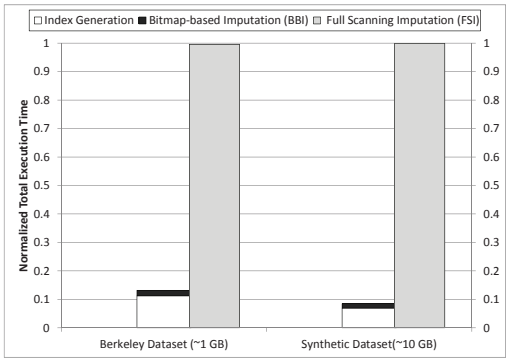


Fig. 5. Time Comparison between BBI and FSI Based Most Common Value Imputation

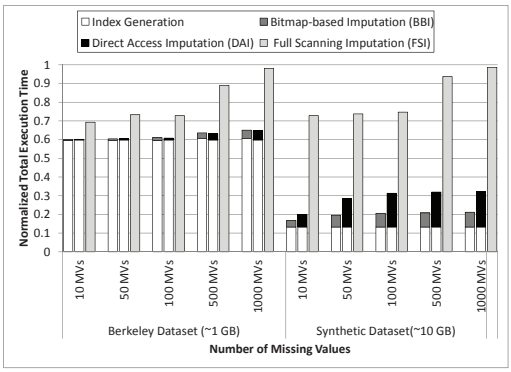


Fig. 7. Time Comparison between DAI, BBI, and FSI Based Concept-based Most Common Value Methods Using Different Numbers of Missing Values (MVs)

time for DAI, BBI, and FSI methods is shown in Figure 6 with different number of missing values in both target datasets. The figure shows that DAI and BBI methods speeds up the imputation process compared to the FSI method. The overall time is dominated by the index generation time. If the index can be pre-generated, the method can provide very large speedups. The figure also shows an important observation. In the Berkeley dataset, DAI satisfies a higher performance compared to BBI method with larger number of missing values. However, in the synthetic dataset, the BBI gives a higher performance. The reason of this behavior that BBI depends on the number of distinct values in the target attributes (i.e., in this case the missing value's attribute and the label attribute) while the DAI depends on the number of matched records for each missing value. In the Berkeley dataset, the number of distinct values in the target attributes is large compared to the synthetic dataset which negatively impact the BBI performance.

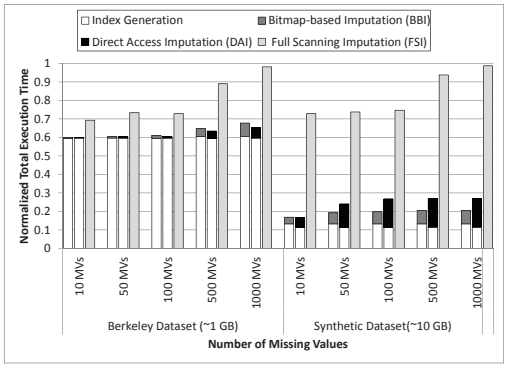


Fig. 6. Time Comparison between DAI, BBI, and FSI Based Concept-based Mean Attribute Value Methods Using Different Numbers of Missing Values (MVs)

4) Concept-based Mean Attribute Value (CMAV): The efficiency of implementing the Concept-based Mean Attribute Value algorithm with both DAI and BBI methods is shown by Figure 7. On both target datasets, the DAI and BBI methods are more faster than the FSI method with different number of missing values. Just like the case of the Concept-based Most Common Attribute Value method, in the Berkeley dataset the DAI is faster than BBI because of larger distinct values in the target attributes. However, in the synthetic dataset the performance gains from the BBI method is more significant, because the BBI method does not require any I/O processing compared to DAI method.

B. Impact of the Number of Distinct Values

A single bitmap vector is built for each distinct value in each attribute, unless binning is used. Increasing the number of distinct values in each attribute impacts both the index generation time and the proposed imputation methods based on bitmap vectors. To see this impact, we evaluate the performance of each imputation algorithm using both DAI and BBI methods with different number of distinct values in each attribute. Our experiment assumes that binning is not used. In this experiment, we generate five synthetic datasets with the same number of records (i.e., $20 \cdot 10^7$) and attributes (i.e., nine attributes and one label), but with different number of distinct values in each attribute (i.e., 100, 500, 1000, 5000, and 10000).

The performance of both CMC and CMAV methods with different number of distinct values is shown by Figure 8. It is not surprising that execution time increase with increasing number of distinct values. A more significant observation is that the BBI method becomes worse with very large number of distinct values. The BBI method depends on scanning the bitmap vectors to estimate the missing value, and increasing the number of distinct values increases the number of bitmap vectors generated, which slowdown the imputation process. Overall, it will be better to use binning when the number of distinct values is large.

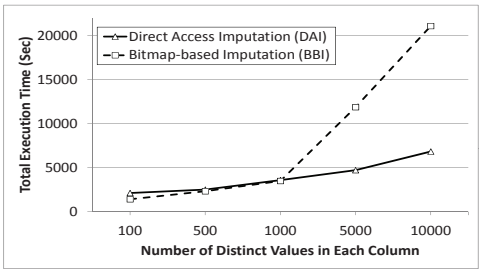
C. Scalability

The goal of this experiment is to evaluate the scalability of our proposed methods when imputing missing values in datasets with increasing sizes. Five synthetic datasets are used in this experiment, i.e., 5, 10, 15, 20, and 30 GB. These datasets were randomly generated in two different manners: by increasing the number of attributes and by increasing the number of records. Each dataset was also generated to include 100 missing values distributed across different attributes. Because our goal is to evaluate the runtime scalability of proposed methods, we only recorded the total execution time of each imputation algorithm, without including the index generation time.

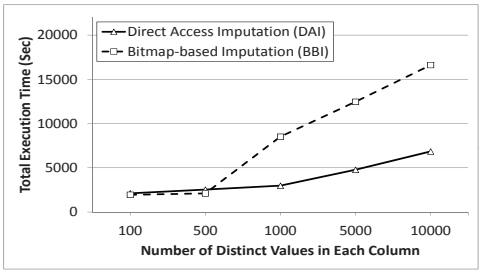
The results are shown in Figures 9 and 10. As shown in the figures, the execution times only increase linearly as the dataset size increases – either by increasing the number of records or by increasing the number of attributes in each record. This experiment also shows another observation, as illustrated particularly by Figures 10(b) and 10(c). Because the BBI method needs to analyse the indices files to impute the missing values, increasing the number of records negatively impacts the efficiency of BBI method compared to DAI method in both Concept-based Most Common value and Concept-based Mean Attribute imputation algorithms.

D. Imputation Method Impact on Data Mining

Since accuracy comparisons on synthetic datasets are not very meaningful, and because the size of the dataset was not very impor-

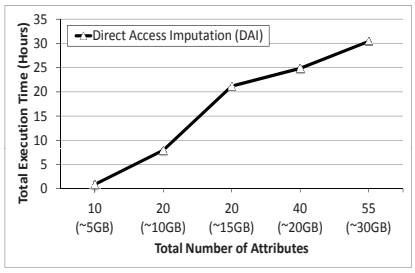


(a) Concept-based Most Common Value (CMC)

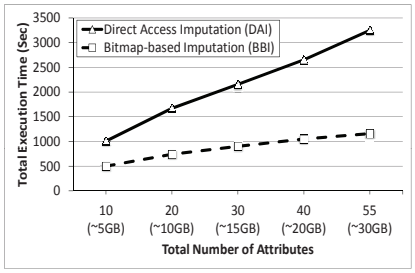


(b) Concept-based Mean Attribute Value (CMAV)

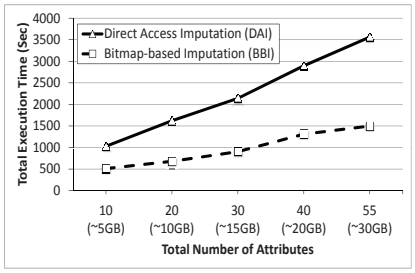
Fig. 8. Performance of Different Imputation Algorithms with Different Numbers of Distinct Values



(a) Global Closest Fit

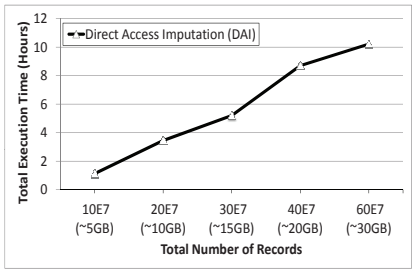


(b) Concept-based Most Common Value

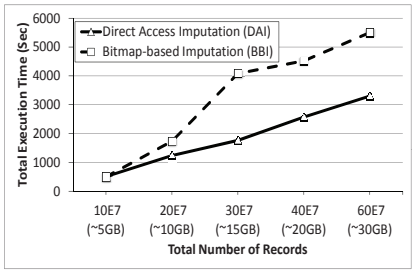


(c) Concept-based Mean Attribute Value

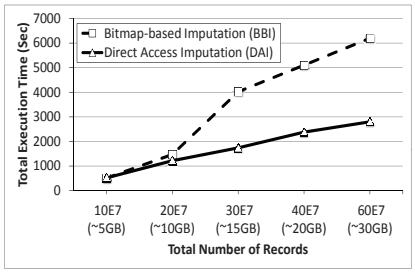
Fig. 9. Scalability: Influence of Increasing the Number of Attributes on DAI and BBI based Imputation Algorithms



(a) Global Closest Fit



(b) Concept-based Most Common Value



(c) Concept-based Mean Attribute Value

Fig. 10. Scalability: Influence of Increasing the Number of Records on DAI and BBI based Imputation Algorithms

tant, we used seven benchmarks datasets from the UCI repository [30] for accuracy comparisons. The latter are summarized in Table II; specifically, we show different characteristics of each dataset such as the number of records, attributes, and classes. We consider all the attributes as numerical attributes. In the case of symbolic attributes, we convert then into numeric format during the preprocessing stage.

TABLE II. UCI DATASETS DESCRIPTION

Dataset	No. records	No. attributes	No. classes
Glass Identification	214	10	7
German Credit Data	1000	20	2
Indian Liver Patient (ILP)	583	10	2
Mammographics	961	6	2
Pima Indians Diabetes	768	8	2
User Knowledge Modeling (UKM)	403	5	4
Wisconsin	699	10	2

As we stated above, all imputation methods are heuristics. Thus, the end goal of DAI and BBI methods is not to produce the same results as the full-scan methods, but to support the data mining process. With this observation, we evaluate different imputation algorithms with respect to the final results from a specific classification method. Particularly, a rule-based classification algorithm (i.e., the CPAR classifier) is used [11]. This algorithm depends on generating a set of predicative association rules from labeled records in a given dataset to classify unlabeled records in the same dataset. Comparing to other association rules-based classifiers, CPAR algorithm has several advantages such as: avoiding redundant rule generation by using dynamic programming, generating a small set of association rules, and achieving higher accuracy [28]. Because our main goal is the imputation process, we only use a single classification algorithm to compare the accuracy between different proposed imputation process.

Imputation using different methods is applied on datasets prior to classification, and subsequently, the classifier obtained is evaluated using a popular metric, which is the *Wilson's Noise Ratio* [31]. This metric identifies the noise rate in a given dataset, i.e., the ratio of records identified as *noisy* to the total number of records. In turn, a record is considered noisy if labelling it using the KNN algorithm (assigning it the most common label among the k-nearest records) leads to an incorrect label. An effective Imputation algorithm should

impute values that reduce the *Wilson's Noise Ratio*. In this experiment, we compare the *Wilson's Noise Ratio* for imputed datasets, after applying the FSI, DAI, and BBI methods, respectively. We used the standard K-Fold Cross-Validation technique with the CPAR algorithm to classify the given dataset records, i.e., the complete dataset after the missing value imputation operation is divided into K parts. Each part is used to train the CPAR model to predict the label for the other $K-1$ parts records. Each record is labeled with the common label predicated using the other nine parts.

We set up different parameters that are required by the CPAR algorithm based on the default values given in an earlier study [28] (i.e., totalweight=0.05, decayfactor=2/3, and min_gain=0.7). For the Wilson's Noise Ratio, we used $k = 5$. Comparing using Wilson's Noise Ratio, we can estimate the impact of our proposed approaches on the results of the CPAR classification algorithm.

A comparison between the brute-force approach and proposed imputation approaches (i.e., DAI and BBI) are shown by Table III with and without the binning strategy. We used seven datasets from the UCI repository for this purpose. For the seven datasets considered in our study, the noise ratio values for FSI, DAI, and BBI are almost the same for different imputation algorithms. This observation shows that using bitmaps, the imputed values lead to the same quality classifier as the brute-force FSI approach. We have also reported the noise ratio when imputation is not applied. As the Table III shows, for six of the seven datasets, very significant reduction in noise ratio is achieved by applying imputation methods. Overall, we can see that DAI and BBI methods not only improve the efficiency of imputation, but help improve the quality of the data mining process after imputation.

TABLE III. COMPARING IMPUTATION METHODS BASED ON WILSON'S NOISE RATIO

Dataset	Before Imput. (%)	Imput. Alg.	FSI (%)	DAI (%)		BBI (%)	
				W/O Bin.	W Bin.	W/O Bin.	W Bin.
Glass Identification	67.28	GCF	11.76	11.76	-	-	-
		MCV	11.76	-	-	11.76	-
		CMC	11.76	11.76	11.76	11.76	-
		CMAV	11.76	11.76	11.76	11.76	11.76
German Credit Data	70.0	GCF	10.88	10.88	-	-	-
		MCV	11.17	-	-	11.17	-
		CMC	10.88	10.88	10.88	10.88	-
		CMAV	10.88	10.88	10.88	10.88	10.88
Indian Liver Patient	28.64	GCF	37.69	37.69	-	-	-
		MCV	19.89	-	-	19.89	-
		CMC	19.89	19.89	19.89	19.89	-
		CMAV	19.89	19.89	19.89	19.89	19.89
Mammographics	53.69	GCF	51.20	51.20	-	-	-
		MCV	52.36	-	-	52.36	-
		CMC	52.26	52.26	49.63	52.78	-
		CMAV	52.78	52.78	52.78	52.78	52.78
Pima Indians Diabetes	34.89	GCF	9.23	9.23	-	-	-
		MCV	9.23	-	-	9.23	-
		CMC	9.23	9.23	9.23	9.23	-
		CMAV	9.23	9.23	9.23	9.23	9.23
User Knowledge Modeling	67.7	GCF	10.48	10.48	-	-	-
		MCV	7.26	-	-	7.26	-
		CMC	7.26	7.26	8.87	7.26	-
		CMAV	7.26	7.26	7.26	7.26	7.26
Wisconsin	34.48	GCF	11.76	11.76	-	-	-
		MCV	12.19	-	-	12.19	-
		CMC	11.76	11.76	11.76	11.76	-
		CMAV	11.76	11.76	11.76	11.76	11.76

V. Related Work

This section gives a brief overview of related work in missing values imputation and using bitmap to index incomplete datasets.

Several approaches have been proposed to avoid the negative impact of missing values in a dataset while performing data mining. The treatment involves estimating missing values, or building a different data view that can be used instead of the incomplete dataset [32]. Our work falls in the categoy of missing value imputation. Many existing approaches involve a direct imputation operation from the complete rows of the same dataset to estimate the missing values. Besides the methods our work builds on, i.e, the Most Common Attribute Value [13] and Global Closest Fit [10], other approaches are based on k-nearest neighbors [33] and k-means clustering [34]. Maximum likelihood approach is also a popular approach to estimate the missing values. In this approach, a statistical model is built on a given incomplete dataset, using a machine learning approach, and model parameters are estimated. For example, the Expectation-Maximization (EM) algorithm has been used to estimate the missing values [35]–[37]. Similarly, neural networks also can be used to build a data model to fill the missing values given incomplete dataset [36].

A somewhat different problem, improving query execution on incomplete datasets by modifying indexing methods, has also been studied [23] [38] [39]. One of these efforts is based on bitmap indexing [23]. The goal of our work is quite different, as we estimate missing values, in preparation for further analysis and mining of the dataset.

VI. Conclusion

Datasets arising in real applications tend to have many uncertainty or incompleteness issues, one of which is the problem of missing values. There are many existing methods for choosing a likely value for a missing value, to be able to support other queries or analysis on the dataset. However, a common theme in all of the work in this area is that algorithms for imputing missing value cannot scale to large datasets. This paper has addressed this shortcoming of the existing work. Particularly, we have shown how bitmaps can be used to accelerate missing value imputation. We have presented two approaches, including one in which bitmap indices can help reduce the number of records to be retrieved, and the second where only bitmaps are used to select a value to replace the missing value.

Through extensive evaluation, we have shown large performance improvements. Even after including index generation time, our methods are faster. However, if an index has already been built to support other functionality (such as supporting query processing), the improvements from our methods are very large. On other hand, the result of the data mining process stays approximately the same as with the original methods.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Communications of the ACM, vol. 51, no. 1, 2008, pp. 107–113.

[2] L. L. Liu Peng, "A review of missing data treatment methods," International journal of intelligent information systems and Tech, 2005, pp. 412–419.

[3] K. J. Cios and L. A. Kurgan, "Trends in data mining and knowledge discovery," in Advanced techniques in knowledge discovery and data mining. Springer, 2005, pp. 1–26.

[4] D. F. Heitjan and S. Basu, "Distinguishing missing at random and missing completely at random," The American Statistician, vol. 50, no. 3, 1996, pp. 207–213.

[5] P. Clark and T. Niblett, "The cn2 induction algorithm," Machine learning, vol. 3, no. 4, 1989, pp. 261–283.

[6] I. Kononenko, I. Bratko, and E. Roskar, "Experiments in automatic learning of medical diagnostic rules," in International School for the Synthesis of Experts Knowledge Workshop, Bled, Slovenia, 1984.

[7] J. W. Grzymala-Busse, "On the unknown attribute values in learning from examples," in Methodologies for intelligent systems. Springer, 1991, pp. 368–377.

[8] J. W. Grzymala-Busse and M. Hu, "A comparison of several approaches to missing attribute values in data mining," in Rough sets and current trends in computing. Springer, 2001, pp. 378–385.

[9] S. Aslan, C. Yozgatlıgil, C. İyigün, İ. Batmaz, M. Türkeş, and H. Tatlı, "Comparison of missing value imputation methods for turkish monthly total precipitation data," in 9th International Conference on Computer Data Analysis and Modeling: Complex Stochastic Data and Systems, Minsk, Belarus, 2010, pp. 7–11.

 ISBN: 978-1-61208-408-4

[10] S. Gaur and M. Dulawat, “A closest fit approach to missing attribute values in data mining,” International Journal of advances in Science and Technology, vol. 2, no. 4, 2011, pp. 18–24.
[11] J. Luengo, S. García, and F. Herrera, “On the choice of the best imputation methods for missing values considering three groups of classification methods,” Knowledge and information systems, vol. 32, no. 1, 2012, pp. 77–108.
[12] L. Wohlrab and J. Fürnkranz, “A review and comparison of strategies for handling missing values in separate-and-conquer rule learning,” Journal of Intelligent Information Systems, vol. 36, no. 1, 2011, pp. 73–98.
[13] J. W. Grzymala-Busse, L. K. Goodwin, W. J. Grzymala-Busse, and X. Zheng, “Handling missing attribute values in preterm birth data sets,” in Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing. Springer, 2005, pp. 342–351.
[14] P. O’Neil and D. Quass, “Improved query performance with variant indexes,” in ACM Sigmod Record, vol. 26. ACM, 1997, pp. 38–49.
[15] K. Wu, W. Koegler, J. Chen, and A. Shoshani, “Using bitmap index for interactive exploration of large datasets,” in Scientific and Statistical Database Management, 2003. 15th International Conference on. IEEE, 2003, pp. 65–74.
[16] Y. Wang, Y. Su, A. Gagan, and T. Liu, “SciSD: Novel Subgroup Discovery Over Scientific Datasets Using Bitmap Indices,” Technical Report OSU-CISRC-3/15-TR03, Ohio State University, Tech. Rep., 2015.
[17] N. Koudas, “Space efficient bitmap indexing,” in Proceedings of the ninth international conference on Information and knowledge management. ACM, 2000, pp. 194–201.
[18] G. Antoshenkov, “Byte-aligned bitmap compression,” in Data Compression Conference, 1995. DCC’95. Proceedings. IEEE, 1995, p. 476.
[19] K. S. Kumar, M. Laxmaiah, and C. S. Kumar, “A compacted bitmap vector technique to evaluate iceberg queries efficiently,” International Journal, vol. 3, no. 6, 2013, pp. 412–418.
[20] S. J. van Schaik and O. de Moor, “A memory efficient reachability data structure through bit vector compression,” in Proceedings of the 2011 international conference on Management of data. ACM, 2011, pp. 913–924.
[21] F. Deliège and T. B. Pedersen, “Position list word aligned hybrid: optimizing space and performance for compressed bitmaps,” in Proceedings of the 13th International Conference on Extending Database Technology. ACM, 2010, pp. 228–239.
[22] Y. Su, Y. Wang, and G. Agrawal, “In-situ bitmaps generation and efficient data analysis based on bitmaps,” in HPDC. ACM, 2015.
[23] G. Canahuate, M. Gibas, and H. Ferhatosmanoglu, “Indexing incomplete databases,” in Advances in Database Technology-EDBT 2006. Springer, 2006, pp. 884–901.
[24] “Bitmap index vs. b-tree index: Which and when?” http://www.oracle.com/technetwork/articles/sharma-indexes-093638.html, [Online; accessed 1-January-2015].
[25] J. Chou, K. Wu, O. Rubel, M. Howison, J. Qiang, B. Austin, E. W. Bethel, R. D. Ryne, A. Shoshani et al., “Parallel index and query for large scale data analysis,” in High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for. IEEE, 2011, pp. 1–11.
[26] B. He, H.-I. Hsiao, Z. Liu, Y. Huang, and Y. Chen, “Efficient iceberg query evaluation using compressed bitmap index,” Knowledge and Data Engineering, IEEE Transactions on, vol. 24, no. 9, 2012, pp. 1570–1583.
[27] Y. Su, G. Agrawal, and J. Woodring, “Indexing and parallel query processing support for visualizing climate datasets,” in Parallel Processing (ICPP), 2012 41st International Conference on. IEEE, 2012, pp. 249–258.
[28] X. Yin and J. Han, “Cpar: Classification based on predictive association rules.” in SDM, vol. 3. SIAM, 2003, pp. 331–335.
[29] “Berkeley earth,” http://www.berkeleyearth.org, [Online; accessed 12-January-2015].
[30] C. Blake and C. J. Merz, “{UCI} repository of machine learning databases,” http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998, [Online; accessed 12-January-2015].
[31] D. L. Wilson, “Asymptotic properties of nearest neighbor rules using edited data,” Systems, Man and Cybernetics, IEEE Transactions on, no. 3, 1972, pp. 408–421.
[32] S. Parthasarathy and C. C. Aggarwal, “On the use of conceptual reconstruction for mining massively incomplete data sets,” Knowledge and Data Engineering, IEEE Transactions on, vol. 15, no. 6, 2003, pp. 1512–1521.
[33] G. E. Batista and M. C. Monard, “An analysis of four missing data treatment methods for supervised learning,” Applied Artificial Intelligence, vol. 17, no. 5-6, 2003, pp. 519–533.
[34] D. Li, J. Deogun, W. Spaulding, and B. Shuart, “Towards missing data imputation: A study of fuzzy k-means clustering method,” in Rough Sets and Current Trends in Computing. Springer, 2004, pp. 573–579.
[35] A. P. Dempster, N. M. Laird, D. B. Rubin et al., “Maximum likelihood from incomplete data via the em algorithm,” Journal of the Royal Statistical Society, vol. 39, no. 1, 1977, pp. 1–38.
[36] Z. Ghahramani and M. I. Jordan, “Learning from incomplete data,” Technical Report AI Lab Memo No. 1509, CBCL Paper No. 108, MIT AI Lab, August 1995.
[37] W. R. Harvey, User’s guide for LSML76: Mixed model least-squares and maximum likelihood computer program. Ohio State University, 1977.
[38] B. C. Ooi, C. H. Goh, and K.-L. Tan, “Fast high-dimensional data search in incomplete databases.” in VLDB, vol. 98, 1998, pp. 357–367.
[39] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, “Skyline query processing for incomplete data,” in Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. IEEE, 2008, pp. 556–565.

 ISBN: 978-1-61208-408-4