EXTENDIBLE MULTIDIMENSIONAL SPARSE ARRAY REPRESENTATIONS FOR DATA WAREHOUSING

Catherine Ann Honegger

A masters proposal submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, September 2017

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

Declaration

I declare that this masters proposal is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination to any other University.

Signed this ____ day of _____________ 20__

____

Catherine A. Honegger

Abstract

Data warehousing is a method used to combine multiple varied datasets into one complete and easily manipulated database as a decision support system. Research over several years has concluded that multidimensional representation of data in data warehousing not only gives a good visual perspective of the data to the user, but also provides a storage scheme for efficient processing. Since data in a data warehouse grows dynamically the multidimensional representation should also be expanded dynamically. Efficient dynamic storage schemes for storing dense, extendible, multidimensional arrays by chunks have been developed in the last couple of years. However in data warehousing the corresponding multidimensional arrays are predominantly sparse arrays. Currently no storage or mapping techniques allow for the extendibility of multidimensional sparse arrays. This research proposes to improve the modelling and representation of extendible multidimensional sparse arrays that involve compression and storage efficiency techniques. Our work addresses extendibility of the sparse array only, and does not concern the shrinking of the sparse array and shows how the techniques are applied in data warehousing.

Acknowledgements

I would like to thank my research supervisor, Professor Ekow Otoo, for taking me on as a masters student. My special thanks also go to Professor Ken Nixon for his help and support in early stages of the proposal.

I am grateful to the University of the Witwatersrand for awarding me a Postgraduate Merit Award as financial assistance during my studies.

I am also grateful to the National Research Foundation (NRF) South Africa for their financial support for my study.

Contents

List of Figures

List of Tables

Nomenclature

BESS	Bit-Encoded Sparse Storage
BxCCS	Bit-Encoded Compression Column Storage
BxCCR	Bit-Encoded Compression Row Storage
CCS	Compressed Column Storage
CRS	Compressed Row Storage
GCC	GNU Compiler Collection
HDF	Hierarchical Data Format
HDFS	Hadoop Distributed File System
NetCDF	Network Common Data Form
OLAP	On-Line Analytical Processing
OS	Operating System
PTCS	PATRICIA Trie Compressed Storage
SQL	Structured Query Language
xCRS	Extended CRS
xCCS	Extended CCS
XSAS	Extendible Sparse Array Storage

Chapter 1

Introduction

Over recent years with the increase in smart-phone use, internet use, and virtually every activity leaving a digital trace, society has been able to store and collect more data, leading to the emergence of huge databases with quintillion bytes of data being produced everyday [1]. Data has been dubbed “The world’s most valuable resource” in 2017 as economists believe it is “the oil of the digital era” with the five most valuable listed firms in the world for 2017 being data titans[1].

By combining different data sets, useful information can be retrieved from the stored data. Information resources are incredibly important and valuable in business management and decision-making [2, 3]. By collecting lots of data, companies are able to improve their products and entice more clients with these improvements [1]. Due to their significance, these data assets must be stored properly and accessed easily [2]. Traditional data analysis tools are not able to handle such growing quantities of data. Thus various applications and technologies for managing big-data and high volume data-streams in data intensive computing is becoming essential. Further big-data, data-mining and machine learning are now of major concern in several scientific domains. Thus the field of Big Data research has materialised from the need to store such large quantities of information. Data analysis in all societal domains involves the storage, retrieval, processing and visualisation of huge databases [4].

Big data is a rapidly growing on a global scale. However there is a bottleneck of technology that is required to extend the capabilities of the field. Several big data technologies are required that currently don’t exist, including: architectures, algorithms, and techniques [5, 6].

1.1 Problem Motivation

A new phenomenon, known as data warehousing, was developed as a result of the large quantities of data that need to be stored in recent years [2]. Data warehousing is a method used to combine multiple varied datasets into one complete and easily manipulated database as a decision support system. On-Line Analytical Processing (OLAP) can then be performed on these databases by an organisation in order to analyse the data, complete data-mining and determine trends. The organisation is then able to build business intelligent decisions by utilising the analysed data. Data warehousing has many applications in medical informatics and public-health, smart cities, mining, energy, physics and financial systems to name a few.

The storage of the datasets influences the accuracy, speed and performance of the data analysis and thus it is crucial to assess how this information is stored and accessed. Research over several years has concluded that multidimensional representation of data in data warehousing not only gives a good visual perspective of the data to the user, but also provides a storage scheme for efficient processing. Several research advancements have been made in multidimensional arrays in order to ensure that the datasets are efficient and inexpensive. Multi-dimensional arrays when used for their selection, aggregation, summation and other range queries are processed more efficiently than their SQL counter part in huge database queries. Since data in a data warehouse grows dynamically the multidimensional representation should also be expanded dynamically. These multidimensional datasets are continuously increasing by appending new data to the dataset as new information is added [4].

1.2 Significance

Recent breakthroughs in extendible multidimensional arrays have led to a new area of study in data warehousing, which requires further research and development. Efficient dynamic storage schemes for storing dense, extendible, multidimensional arrays by chunks have been developed [7, 8]. However in data warehousing the corresponding multidimensional arrays are predominantly sparse arrays. It is thus important to analyse extendible multidimensional sparse arrays. There have been a number of advances made in the performance and efficiency of storage schemes for representing multidimensional sparse arrays [9, 10, 11]. However, currently no storage or mapping techniques allow for the extendibility of multidimensional sparse arrays [12]. Being able to utilize efficient extendible multidimensional sparse arrays will extend the capabilities and domains for data warehousing, data-mining and big-data analysis.

1.3 Problem Statement

The question raised is whether multidimensional sparse arrays can be stored in such a way that new elements and hyper-plane dimensions can be added. The problem that we are focused on requires the use of new storage techniques to investigate the extendibility of multidimensional sparse arrays with regards to computer performance and storage efficiencies.

An illustrative example of where data warehousing is used is given in Table 1.1. Here we will have a fact table displaying the number of items sold on a particular day. This can be further extended into a data repository consisting of items sold per day per store as a relational table. In order to analyse the data to improve a company's sales, they might want to know the number of items sold, or which item is sold more on which day, i.e. more carrots are sold on Wednesdays. In order to summarise these values and find patterns in the data these fact tables are converted into multidimensional arrays as shown in Figure 1.1. Using these arrays, drill-down (or roll-down) and roll-out queries can be conducted to summarize the number of sales per week.

Table 1.1: A Typical Data Warehousing Fact Table.

Item	**Time**	**Amount**
Apple	Monday	12
Orange	Monday	10
Carrot	Monday	9
Banana	Monday	9
Grapefruit	Monday	10
Apple	Tuesday	7
Orange	Tuesday	8
Banana	Tuesday	18
Apple	Wednesday	5
Banana	Wednesday	3
Carrot	Wednesday	10

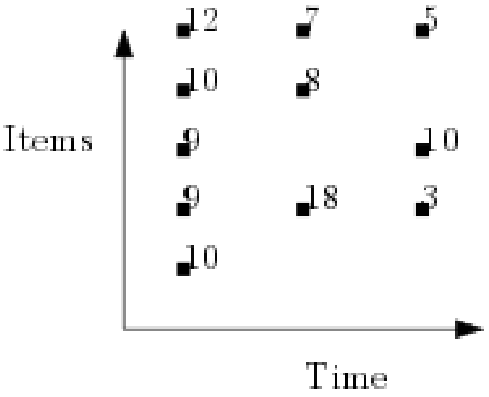

Figure 1.1: Fact Table Converted to a 2D Array

Due to the importance of data warehouses, several methods for representing multidimensional sparse arrays have been developed over recent years. These methods include, triplicate, linked list, compressed row storage, compressed column storage, bit-encoded sparse storage and PATRICIA trie compressed storage. Some of these techniques are only suitable for 2-dimensional sparse arrays and do not cater for higher dimensionality. These methods are discussed in Chapters 2 and 3.

As data warehouses are increasing dynamically research has also been made on the extendibility of multidimensional dense arrays. However as data warehouses primarily generate sparse arrays, using a method for dense arrays will waste a considerable amount of storage space. Thus it is proposed to investigate the extendibility of multidimensional sparse arrays with regards to computer performance and storage efficiencies.

1.4 Other Applications

There are several problems that currently exist in the world that would utilize Big Data and data warehousing. A major example would be the frequency and quality of demographic statistics produced by countries as well as demographic statistics deficits. Such applications are particularly of major demand in developing countries.

- There are 46 African countries operating without a complete birth and death registration system.

- Population censuses have not been conducted since 2010 in several African countries.
- Several African countries have not conducted an agricultural census in the last ten years. South Africa in particular held their last agricultural census in 2007.

Having up-to-date, reliable demographic statistics enables people within the population to partake in highly important activities. These include gaining quality education, formal employment, voting in elections, access to financial services, obtaining passports and obtaining IDs to name but a few. Reliable data also enables governments to budget better and improve the delivery of public goods and services [13]. A further application of data warehousing technologies used in the economic sector include tracking taxes, risk analysis and fraud detection.

The field of application of data warehouse systems are also used in the trade sector to monitor inventory control, check up on customer care, and analyse company sales [2].

1.5 Contribution to Field

As data is constantly growing, there needs to be a method to provide for this extendibility. The aim of the proposed study is to improve the modelling and representation of extendible multidimensional sparse arrays that also involves useful compression and storage efficiency. The representation of the extendible multidimensional sparse arrays must include the characteristics of an array format, such that the array can take on any designed multiple hyper-plane dimensions. The characteristics of the array allow for easy data access. In addition, this allows for both drill-down and roll-out queries. Where drill-down queries allow for detailed data access and roll-out queries allow for summary data access.

The modelling of the extendible multidimensional sparse arrays will be able to contribute to developing algorithms that can process data at higher speeds, use less physical computational resources and improve the usage of computational power. While there are numerous works published on the storage of sparse arrays, these have been limited predominantly on 2D arrays and are not extendible [3]. In the few cases where the storage of sparse arrays have been done, the techniques are not extendible. These include the method of TileDB [14].

Our work addresses extendibility of the sparse array only, and does not concern the shrinking of the sparse array as data is never deleted from data warehouses [2]. We will refer to this as

XSAS which stands for Extendible Sparse Array Storage.

1.6 Organisation of the Proposal

Chapter 2 provides a background on data warehousing and OLAP, extendible multidimensional dense arrays and multidimensional sparse arrays. Chapter 3 provides the proposed methodology for developing extendible multidimensional sparse array representations for data warehousing. The experimental set-up is provided in Chapter 4. Chapter 5 details the preliminary results of the research. Validation of the results is discussed in Chapter 6. A schedule for the completion of the research is outlined in Chapter 7. A summary of the proposal is given in Chapter 8.

Chapter 2

Background

2.1 Big Data Representations

In order to understand data warehouses one also needs to look at “Big Data”. Big Data can be characterized in the following ways. Big Data has big volume. When it is stored it needs to be scalable by increasing the storage allowing for the data to grow. Big Data has high velocity. Big Data databases need to cope with high speeds, ensuring that the rate at which data comes in is catered for. Big Data has a wide variety of data types. Data comes in any different formats, a clear example of databases that need to cater for variety are multimedia databases for geographic location. These databases need to ensure that spacial data, moving data, structured data, unstructured data and legal documents are captured. Big Data needs to be valid. Data is only useful if it is accurate and reliable. In order to ensure that data is veracity one must ensure that the data is stored correctly and not manipulated accidentally. Big Data has big value. As discussed in Chapter 1 data is a crucial component in improving intelligent business decisions.

There are several universal formats for representing Big Data including Hierarchical Data Format (HDF5), Network Common Data Form (NetCDF), Flexible Image Transport System (FITS), NumPy, PyTable, TileDB. Platforms for data analysis include Hadoop, Weka, RapidMiner, Pentaho, Scikit-learn, Apache Spark and R [14].

2.2 Data Warehousing

In Chapter 1 we mentioned how important information is in the current world [2]. It was also discussed that due to the vast amounts of data being created and stored every day and the inability for traditional database methods (e.g. SQL database systems) to deal with such large volumes of data, data warehousing was created for decision making.

Bill Inmon defines data warehousing as an architecture that is a subject-oriented, non-volatile, integrated, time variant collection of data created for the purpose of managements decision making. Ralph Kimball defines a data warehouse as a copy of transaction data specifically structured for query and analysis.

The modern advanced data warehousing processes allow for OLAP by creating a new data repository that integrates basic data from various sources, arranges the data formats and makes the data available for analysis and decision-making [2]. OLAP queries require dynamic, multidimensional analyses of huge quantities of data to process records for decision-making. OLAP queries make use of multidimensional representations of data warehouses as the data is viewed as points in space whose dimensions correspond to many possible analysis dimensions [2]. A 3D data warehouse cube is given in Figure 2.1 to explicitly show why multidimensional representations are important for OLAP.

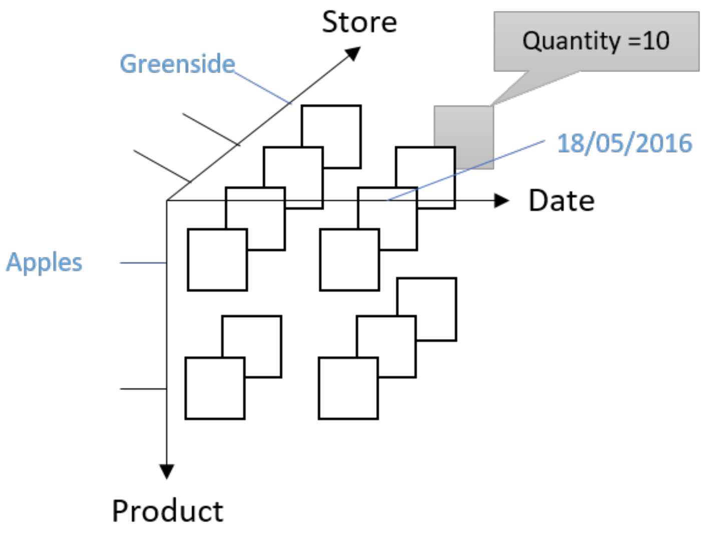

Figure 2.1: An Example of a Multidimensional Data Warehouse

The data warehousing and OLAP operations that are most commonly used include: Get, Insert, Addition, Multiplication, Subtraction, Division, roll-up, drill-down, slice-and-dice, pivot, drill-across, drill-through, and range or box access. We will show how these operations are executed in our XSAS representation.

2.3 Methodologies

In order to develop a model for extendible multi-dimensional sparse arrays, the current representations of extendible multi-dimensional arrays must be analysed. The study of the current representations will give a clear indication of how to integrate the representation of extendible multi-dimensional sparse arrays, so that the model can easily be integrated into the current system, preventing unnecessary additional costs. Once a model has been developed, compression of data as well as computing speed will be assessed.

2.4 Sparse Array Representations

There are numerous studies that have been conducted on the efficiency of different sparse array representation schemes [3, 10]. In order to accurately understand these models, we must first understand exactly what is meant by a sparse array. If an array is sparse, it has very few non-zero elements relative to the product of the cardinalities [3].

There are three main methods used for 2-dimensional sparse array representations, namely:

1. Matrix Market (known as a relational table or a Triplicate)
2. Linked list
3. Compressed Row (or alternatively Column) Storage (CRS/CCS)

These three methods are discussed in more detail in Chapter 3.

There are six other methods that are used for multidimensional representations, namely:

1. Bit-Encoded Sparse Storage (BESS)
2. Extended CRS/CCS known as xCRS and xCCS
3. PATRICIA Trie Compressed Storage (PTCS)
4. Bit-Encoded Compressed Row (or Column) storage (BxCRS/BxCCS)
5. Hybrid approach (Hybrid) that combines BESS with xCRS

BESS is the simplest low level storage scheme that chops every point into bit block dimensions. BESS has decent storage, but is traditionally static with the maximum number of chunks being determined algorithmically.

The PTCS is unique and allows for extendibility with regards to data warehousing operations. BESS and PTCS are discussed further in Chapter 3.

2.5 Impact of Current Environment

Multidimensional representation of data in data warehousing not only gives a good visual perspective of the data to the user, but also provides a storage scheme for efficient execution of data warehouse operations [7]. Data warehousing is a highly utilized platform for gaining insight into the ever growing mass of data - Big Data in both business and scientific, being accumulated [10]. However due to the dynamic nature of data expansion of data in data warehousing, the corresponding multidimensional array representation must naturally be extendible [7] [8]. Typically, the multidimensional array representations in data warehousing are sparse arrays. This requires that we address the problem of extendible sparse arrays. Efficient dynamic storage schemes for storing dense, extendible, multidimensional arrays by chunks have been developed in the last couple of years [7] [8]. Currently no storage or mapping techniques allow for the extendibility of multidimensional sparse arrays [12].

Chapter 3

Methodology

Based on the literature review in Chapter 2, it is evident that the extendibility of multidimensional sparse arrays is an incredibly important topic that has been overlooked, specifically with regards to data warehousing and OLAP. In order to make a contribution to this field, an extendible multidimensional sparse array model must be developed, implemented and evaluated. This chapter presents the proposed model design and experimental procedures for the research study.

Consider a 2D array $A[N_i][N_j]$ with bounds $N_i N_j$. An element denoted by $a < i, j >$ or equivalently a_{ij} gives the value stored in location $< i, j >$ of the array. The locations $< i, j >$ are mapped into a set of linear locations $I[0, ..., M - 1]$ where $M = N_i X N_j$. A mapping function $f(i, j)$, gives the location $I[l]$ where $a < i, j >$ is stored. An example of such a mapping function is the conventional row-major order allocation where $l = f(i, j)$ and $f(i, j)$ is defined as $f(i, j) = iN_j + j$.

3.1 Extendible Multi-Dimensional Sparse Arrays

In Chapters 1 and 2 it was noted that a multidimensional representation of data in data warehousing provides a good conceptual view of the data to the user, as well as providing a storage scheme for efficient processing. It was also noted that these multidimensional representations need to expand dynamically as the data in a data warehouse grows dynamically. Data warehouses predominantly generate sparse arrays. Indexing a 2D dense array using its column and row indexes (i.e. $< i, j > =$ value) is fine for dense arrays, however when it comes to sparse arrays this technique is cumbersome and wastes storage space. In sparse arrays some elements are missing thus we need to store them in a compressed format to ensure that they can be use efficiently.

In order to represent a data warehouse in a multidimensional space a formula needs to be defined to increase the dimensionality of the storage scheme without changing the mapping function $f(i, j)$. As sparse data warehouses consist of large quantities of information with few non-zero elements, an efficient storage scheme that represents these non-zero elements must be chosen. Furthermore a method for extendibility of the multi-dimensional sparse array representation must be chosen.

3.2 Multi-Dimensional Sparse Array Representation

In order to improve the storage efficiency of multi-dimensional sparse arrays, a good storage scheme must be chosen. A comparison of the different sparse array storage techniques detailed in Section 2 is carried out.

3.2.1 Matrix Market Format

Given the example data warehouse in Figure 3.1 one can use a simple matrix market format (otherwise known as a relational table) to represent the information as shown in Table 3.1. When using matrix market format the row index and the column index of each non-zero element is stored along with its value in a 2D array.

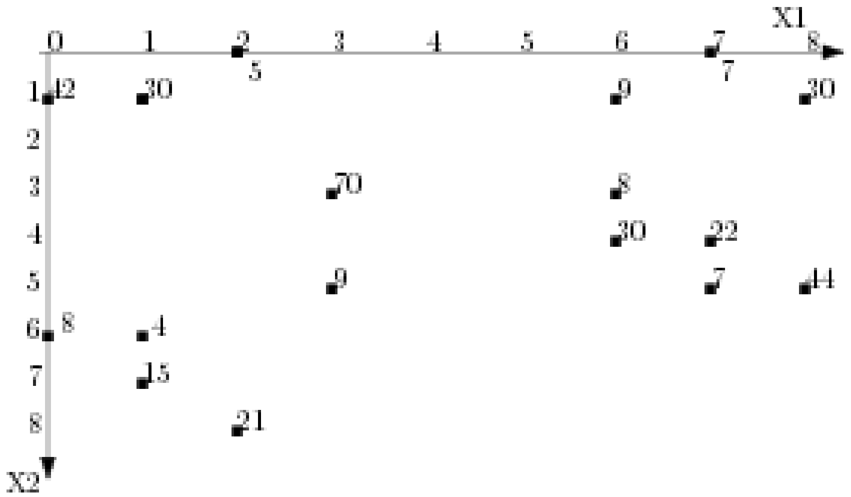

Figure 3.1: A Sparse 2D Array

Table 3.1: Matrix Market Format

Row Index	**Column Index**	**Value**
1	0	42
1	1	30
0	2	5
3	3	70
5	3	9
6	0	8
6	1	4
7	1	15
8	2	21
5	8	44
1	8	30
0	7	7
1	6	9
3	6	8
4	7	22
4	6	30
5	7	7

3.2.2 Linked List

Linked lists consist of nodes that link the non-zero elements of sparse arrays together using pointers. Each node has four fields. A **row** field that stores the row index of the non-zero element, a **column** field that stores the column index of the non-zero element, a **value** field that holds the value of the non-zero element located at index ¡row,column¿, and a **next node** field that indicates the address of the next node. Figure 3.2 provides the basic architecture for a linked list representation.

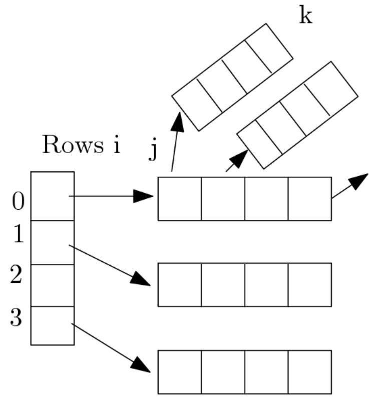

Figure 3.2: Linked List Representation

Pros

- Linked lists are fine if all of the data and processing is done in memory.
- Linked lists are extendible.

Cons

- Linked lists are inappropriate for large files on the disk memory as linked lists can't be organized on the disk. As data warehousing works with big data the storage schema needs to be stored on the disk.

3.2.3 Compressed Row/Column Storage

CRS representation makes use of three one-dimensional arrays to represent sparse 2D arrays. The **Value** array holds the non-zero elements from the 2D array, the **Column index** array holds the column indexes of each non-zero element, and the **Row pointer** array holds the offset value that starts a row [3].

Using the example data given in Table 3.1 and Figure 3.1 we give the CRS representation as shown in Table 3.2.

Table 3.2: Compressed Row Storage Representation

Offset	0	1	2	3	4	5	6	...	14	15	16	17
Value	5	7	42	30	9	30	70	...	4	15	21	17
Column index	2	7	0	1	6	8	3	...	1	1	2	17
Row Pointer	0	2	6	8	10	12	13	15	16	17		

Pros

- CRS and CCS have a wide usage - mainly with 2D arrays and image processing.

Cons

- CRS and CCS are not appropriate representations of data stored on disk.
- CRS and CCS are not extendible - they require reorganising the entire array representation with the new additions by extending the bounds. Thus CRS and CCS are not appropriate in a dynamic environment when changes are happening rapidly.
- CRS and CCS representations are not appropriate for large arrays.

3.2.4 Bit Encoded Sparse Storage

BESS uses a concatenation of the bit encoded representations of the indexes to generate a Bit Encoded Index (BEI). To improve storage for this method, a level of partitioning known as chunking is used. If $h_i, 0 \leq i < k$ is the size of the chunk in each dimension, the number of chunks is $\prod_{i=0}^{k-1}[\frac{n_i}{h_i}]$. The storage required is $(\prod_{i=0}^{k-1}[\frac{n_i}{h_i}]) * ((c + [\frac{\sum_{i=0}^{k-1}[\log h_i]}{wordsize}])N_{chunk})$. Only chunks that contain non-zero elements are stored.

For the example provided in Figure 3.1 if we were to divide the 2D array into 3x3 chunks we would get Figure 3.3 where the chunks breaks and discarded chunks are shown in blue.

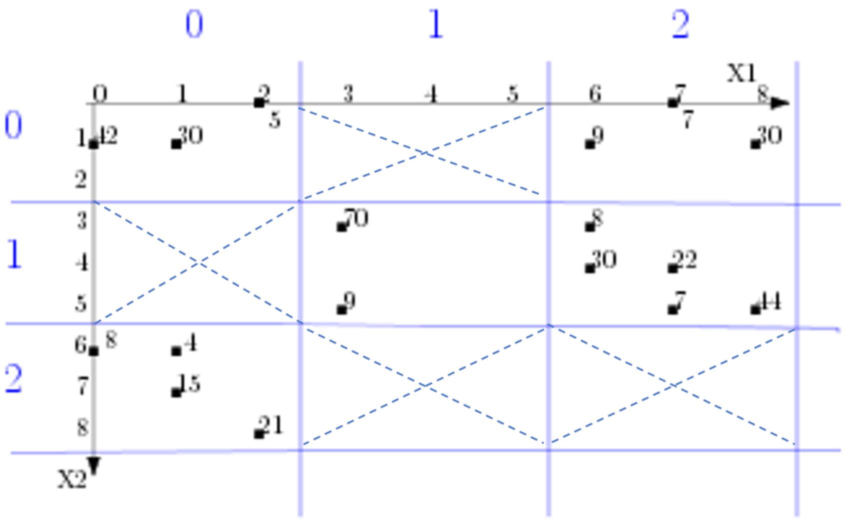

Figure 3.3: BESS Chunking

The content from a non-empty chunk is stored as either a dense array or by using a sparse array method depending on the sparsity of the chunk.

3.2.5 Comparisons of the Different Representations

CRS and CCS are both only used for 2D arrays. In the work of Goil et al. [10] it is clear that BESS out performs Offset-Value pair. In the work of Wang [3] it is evident that although PTCS has a better storage ratio and xCRS/xCCS and BxCRS have faster element retrieval times, BESS has a faster construction time and a faster multi-dimensional aggregation time than all of the other storage schemes.

BESS is a very simple and efficient multi-dimensional sparse array representation as it maps a multi-dimensional sparse array space to a one-dimensional array space, resulting in a bit encoded key index [3]. A multi-dimensional sparse array data model using a BESS array representation scheme as well as a model using a CRS representation scheme will be analysed.

3.3 Extendible Multi-Dimensional Sparse Array Representation

By making use of chunking techniques on a BESS array representation scheme as well as a CRS array representation scheme. The proposed extendibility model can increase the dimensionality of the storage scheme by either increasing the density of the array or increasing the dimension of the indices or bounds of the array. When increasing a sparse array by density, the bounds and structure remain the same, new elements are slotted into a chunk where there was a free space. Extendibility of the bounds of the array is done by appending new chunks to the array. The two different extendibility techniques are displayed in red in Figure 3.4.

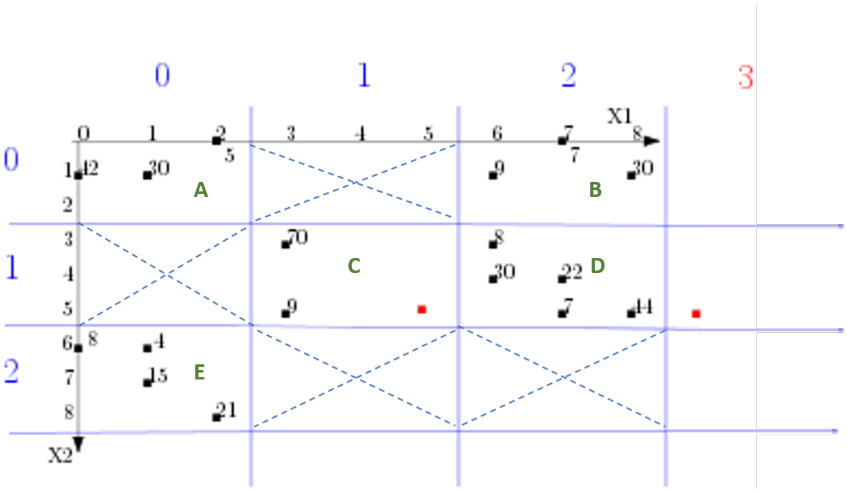

Figure 3.4: A Sparse 2D Array

3.3.1 Multidimensional Sparse Array Blocks

In order to organise the chunks easily for reading and writing, a linear index of the array needs to be generated. Mapping for conventional arrays is done using row major order or column major order [15]. Using column major order we have

$$\begin{aligned} < i, j, k > &= C_0 C_1 C_2 \\ < i_0, j_0, k_0 >= I &= i_0 C_0 + j_0 C_1 + k_0 C_2 \end{aligned} \tag{3.1}$$

where:

$$C_0 = X_1 \cdot X_2,$$
$$C_1 = X_2,$$
$$C_2 = 1$$

An in memory index needs to be built into the blocks. As the indexing needs to be done in memory, multi-branching is not optimal, thus we focus on two way branching. Two important two way branches are the binary tree, and the PATRICIA Trie. Binary search trees are not balanced and can be skewed wasting storage space. PATRICIA stands for Practical Algorithm to Retrieve Information Coded In Alphanumeric [16]. In PATRICIA Tries the skewness is compressed as it only compares bits that change the path and skips levels that are the same. If the chunk index values are stored and organised in a PATRICIA Trie we will avoid heavy sided trees and allow easy access to reading and writing of the array.

PATRICIA Trie representation of Blocks

Using the example in Figure 3.4 by labelling each chunk as A, B, C, D and E respectively and assigning their 2-bit row and 2-bit column indexing we obtain the bit indexing of the BESS chunks as shown in Table 3.3. From this representation the PATRICIA Trie in Knuth representation is given in Figure 3.5.

Table 3.3: Bit Indexing of BESS Chunks

Linear Position	Bits	Chunk
0	0000	A
2	0010	B
4	0100	C
5	0101	D
6	0110	E

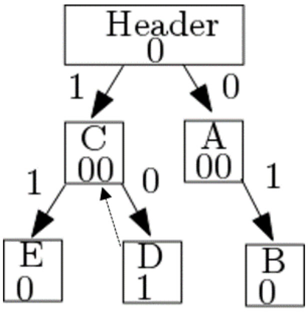

Figure 3.5: Knuth representation of PATRICIA Trie indexing for the example in Figure 3.4

3.4 Methodology for Accessing an Element from a Compressed Sparse Array

Suppose we have the compressed sparse array given in Figure 3.4 and we wish to find element 20 at position $< 4, 6 >$. We will need to take the floor of the row and column indexes divided by the bounds of the matrix. I.e. we have $\lfloor \frac{4}{3} \rfloor = 1$ and $\lfloor \frac{6}{3} \rfloor = 2$. Thus the block index is located at $< 1, 2 >$ which has a linear address of 5. We then use the PATRICIA Trie to locate the block. Once the block has been located, we use row major order to locate the element in the linear matrix within the block.

3.5 Project methodology

A sparse Big Data database will need to be sourced and stored. A BESS compression method as well as a CRS compression method will be applied to the database.

The two algorithms for extension by appending and by inserting will need to be developed and implemented in software in order to determine which algorithm performs better. To assess the performance of the extendible multi-dimensional models, a measure of the storage efficiency and order of retrieval of queries shall be conducted using basic construction, low level retrieval and partial match query array operations.

The two algorithms will be tested on a 2D array (both CRS and BESS), a 3D array(only BESS) and a 4D array(Only BESS) to asses their limitations on different dimensionalities. Figure 3.6 shows the main steps in the proposed methodology of the project.

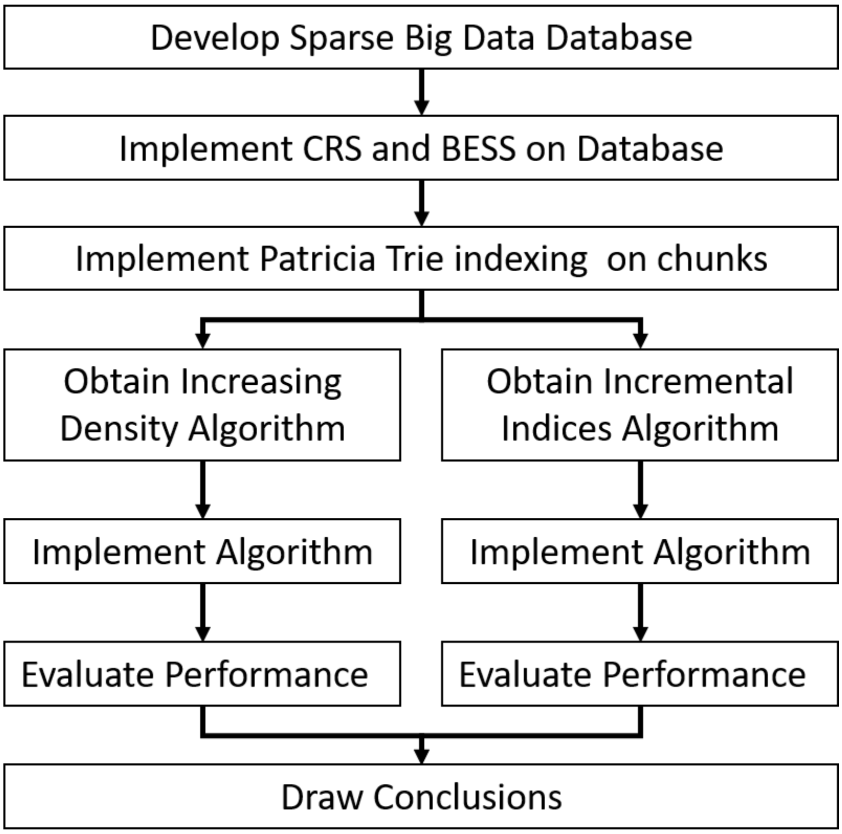
Develop Sparse Big Data Database
Implement CRS and BESS on Database
Implement Patricia Trie indexing on chunks
Obtain Increasing Density Algorithm
Obtain Incremental Indices Algorithm
Implement Algorithm
Implement Algorithm
Evaluate Performance
Evaluate Performance
Draw Conclusions

Figure 3.6: Proposed Methodology

This representation can then be used in data warehousing where items are increased, stores can be acquired in new locations and new time frames can be added.

Chapter 4

Experimental Setup

4.1 Computational Environment

The experiments will be run on a laptop with an Intel(R) Core(TM) i5-2450 multi-core processor at 2.50 GHz and 4 GB of memory running Ubuntu 64-bit Linux 16.04.3 LTS. We have the capability of using a computer with 24 GB of memory for medium sized data and machines with up to 128 GB for extremely large datasets.

The experiments will be implemented in C using GNU Compiler Collection (GCC) 7.2. The code development process will make use of software engineering techniques and best practice with regards to ensuring code is efficient and hardware resources are utilized correctly. This involves making use of tests, ensuing the code is non-repetitive, using standard libraries where possible, and good commenting and naming conventions. The source code will be composed in a revision control environment using Git.

Chapter 5

Preliminary Results

5.1 Investigations on Pre-existing Technologies

Some preliminary investigations have been conducted on pre-existing Big Data formats and pre-existing data analysis platforms. A quick overview of the TileDB storage manager shows that the storage manager only caters for 2D sparse arrays and does not cater for higher dimensions. TileDB inserts new data into sparse arrays by storing the new data in previously empty cells [14]. A look at two data analysis platforms, Weka and RapidMiner, show how pre-existing technologies cater for MOLAP operations, however the platforms do not allow for any extendibility of the actual datasets they are operating on.

5.2 XSAS Implementation in C

Some coding has been conducted in C making use of the GNU Scientific Library. The example matrix given in Figure 3.1 was stored using CRS. The program is currently being extended to store the matrix using BESS. Once the BESS algorithm has been implemented the code will be extended to incorporate PATRICIA Trie indexing. Upon completion of the PATRICIA Trie indexing the density extendibility and the bounds extendibility algorithms will be implemented and their performances analysed.

Chapter 6

Validation of Results

In order to test the performance and efficiency of the proposed solution, several tests will be run. Experiments similar to the ones done in [15], [8], [9] and [10] and will be done where three datasets of different sizes, three datasets of different dimensions (2D,3D and 4D) and three datasets of different sparsity levels (i.e. 80%, 90% and 95%) will be stored in the XSAS representation (i.e. six different datasets). Three different queries will be run and timed on the datasets, these queries will be run three times each in order to ensure that an accurate average measurement has been taken. An assessment of the storage utilization (in percentage) will be done on the arrays. A further investigation will be conducted on 2D arrays comparing the XSAS representation with BESS and the XSAS representation with CRS.

Chapter 7

Schedule and Time-Line for Completion

7.1 Proposed Task Completion Time-Line

The research is expected to be completed by April 2018. Regular meetings are held with the supervisor once a week. There is an open door policy enabling casual meetings to be arranged ensuring that there is constant feedback throughout the lifespan of the proposed research.

Table 7.1 presents a set of major tasks along with their preliminary completion dates for the proposed work.

Table 7.1: Proposed Task Completion Times.

Task Description	**Proposed Completion Date**
Begin Research Project	January 2017
Literature Review of Data Warehousing	March 2017
Literature Review of Sparse Array Representation	May 2017
Literature Review of Dynamic Dense Array Representation	July 2017
Documentation of the Methodology	September 2017
Submit Research Proposal for Approval	October 2017
Visual Presentation of the Research Proposal	October 2017
Develop and Implement Storage and Chunking Techniques	December 2017
Develop Dynamic Expansion Algorithms	February 2018
Evaluate Performance and Analyse Results	April 2018
Submit Research Dissertation for Approval	April 2018

Chapter 8

Summary

Currently no storage or mapping techniques allow for the extendibility of multidimensional sparse arrays. This research proposes to improve the modelling and representation of extendible multidimensional sparse arrays. The proposed solution incorporates compression and storage efficiency techniques by using BESS to divide the sparse array into chunks. Storing the elements within these chunks in a linear matrix using row major order. As well as using PATRICIA Trie indexing to index the BESS chunks. The proposed solution shows how the new XSAS techniques are applied in data warehousing.

Bibliography

[1] T. Economist. "The World's Most Valuable Resource." *The Economist*, vol. 423, no. 9039, p. 7, May 2017.

[2] M. Golfarelli and S. Rizzi. *Data Warehouse Design: Modern Principles and Methodologies*, chap. 1, pp. 1–42. India: McGraw-Hill Professional, first ed., Jan. 2009.

[3] H. Wang. *Sparse Array Representations And Some Selected Array Operations On GPUs*. MSc Dissertation, University of the Witwatersrand, Johannesburg, South Africa, Jul. 2014.

[4] E. J. Otoo and D. Rotem. "Efcient Storage Allocation of Large-Scale Extendible Multidimensional Scientic Datasets." In *18th International Conference on Scientific and Statistical Database Management*, pp. 179–183. IEEE, 2006.

[5] B. Twala. "Big Data for Africa." presentation, Feb. 2017.

[6] N. E. Moukhi, I. E. Azami, and A. Mouloudi. "Data Warehouse State of the art and future challenges." In *2015 International Conference on Cloud Technologies and Applications (CloudTech)*. Marrakech, Morocco: IEEE, Jun. 2015. URL `http://ieeexplore.ieee.org/abstract/document/7337004/`.

[7] G. Nimako, E. J. Otoo, and D. Ohene-Kwofie. "Chunked Extendible Dense Arrays for Scientic Data Storage." In *41st International Conference on Parallel Processing Workshops*, pp. 38–47. 2012. URL `http://ieeexplore.ieee.org/document/6337461/`.

[8] P. Pedereira. "Cubrick: A Scalable Distributed MOLAP Database for Fast Analytics." In *VLDB 2015 PhD Workshop*. Very Large Data Base Endowment Inc., Hawaii: Springer-Verlag, Jul. 2015. URL `http://www.vldb.org/2015/wp-content/uploads/2015/07/pedreira.pdf`.

[9] E. J. Otoo, H. Wang, and G. Nimako. "Multidimensional Sparse Array Storage for Data Analytics." In *IEEE 18th International Conference on High Performance Computing*

and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems, pp. 1520–1529. 2016.

[10] S. Goil and A. Choudhary. "BESS: A Sparse Storage Structure of Multi-dimensional Data for OLAP and Data Mining." Tech. Rep. CPDC-TR-9801-005, Centre for Parallel and Distributed Computing, Northwestern University, 1997.

[11] E. Otoo, G. Nimako, and H. Wang. "New Approaches to Storing and Manipulating Multi-Dimensional Sparse Arrays." In *SSDBM 2014*, 41. ACM, 2014. URL `http://dl.acm.org/citation.cfm?id=2618281`.

[12] G. Nimako. *Chunked Extendible Arrays and its Integration with the Global Array Toolkit for Parallel Image Processing*. PhD Dissertation, University of the Witwatersrand, Johannesburg, South Africa, Oct. 2016.

[13] M. I. Foundation. "Strength in Numbers: Africa's Data Revolution." online, 2015.

[14] TileDB. "TileDB Mechanics 101." online.

[15] E. J. Otoo, G. Nimako, and D. Ohene-Kwofie. "Chunked Extendible Dense Arrays for Scientic Data Storage." *Journal of Parallel Computing*, vol. 39, no. 12, pp. 802–818, Sep. 2013. URL `http://dx.doi.org/10.1016/j.parco.2013.08.006`.

[16] D. Morrison. "PATRICIA - Practical algorithm to retrieve information coded in alphanumeric." *Journal of the ACM*, vol. 15, no. 4, pp. 514–534, Oct. 1968.