

Department of Electrical and Computer Engineering

ESE 589: Learning Systems for Engineering Applications

Project 1: Star Cubing Algorithm

Submitted to Prof. Alex Doboli

Date: 10/14/2021

Team Members:  
Hamid Jalili

Mahan Agha Zahedi

[**Introduction**](#_wqra037su1vc) **2**

[**Star-Cubing Implementation**](#_v2xykqvytspf) **3**

[Pre-Processing the raw data](#_jpu7adqoxd7k) 4

[Generating the star-table](#_7mj5hhnm9bcq) 4

[Generating the star-tree](#_13f1pgbmy34h) 4

[Verifying the Implementation](#_ealcqsd0knl9) 4

[**Experimental Set-Up**](#_jusiexfv60by) **6**

[Datasets](#_ssg5gbse4a9q) 6

[Preprocessing Steps](#_sarh381tl4lk) 7

[Choice of Threshold](#_5n14d3fm2owq) 7

[Threshold Summary Table](#_gs02ek5l40f9) 9

[Metrics to be collected](#_wc0550rm3022) 10

[**Results & Discussion**](#_kyx5casn1on) **10**

[Execution Time VS Threshold](#_rfqoteql8sz3) 10

[Memory Resident Set Size VS Threshold](#_uoryirvbj37k) 12

[Virtual Memory & Shared Memory Size VS Threshold](#_ufsxhvefqc3c) 12

[**Conclusion**](#_m7zxbwrp72ty) **13**

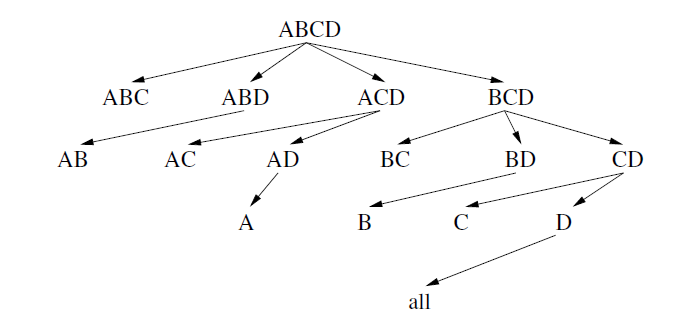
[**References**](#_8pmehxdf36mh) **15**

**Star-Cubing Algorithm**

# Introduction

Multidimensional data mining is an approach to data mining that integrates OLAP (online analytical processing) based data analysis with knowledge discovery techniques [1]. It searches for useful patterns in data warehouses by exploring data in multi-dimensional space. For fast access of summarized data, precomputation of a data cube is a crucial operation. Given the high dimensionality of most data, multidimensional analysis can run into performance bottlenecks; Therefore, it is important to study data cube computation techniques.

Earlier studies have developed two significant methods for data cube computation: top-down vs bottom-up. The top-down approach, used by the MultiWay algorithm, aggregates the multiple dimensions simultaneously starting at the apex cuboid (most general). The caveat with this method is that it cannot utilize Apriori pruning when computing iceberg cubes.

  
Figure 1: Top-Down Approach

The bottom-up approach, utilized by both the BUC (Bottom-Up Computation) and H-Cubing compute the iceberg cube starting from the base cuboid (least general) and employ Apriori pruning. This technique is used to reduce unnecessary computation using the anti-monotonic property. The Anti-monotonic property states that if the aggregate value on the shared dimension does not satisfy the Iceberg condition, then all those cells extending from that shared dimension, cannot satisfy that condition either. BUC explores fast sorting and partitioning techniques; while H-Cubing traverses a data structure, the H-tree, for shared computation. Although these two bottom-up approaches utilize pruning, they cannot make full use of multi-dimensional simultaneous aggregation.

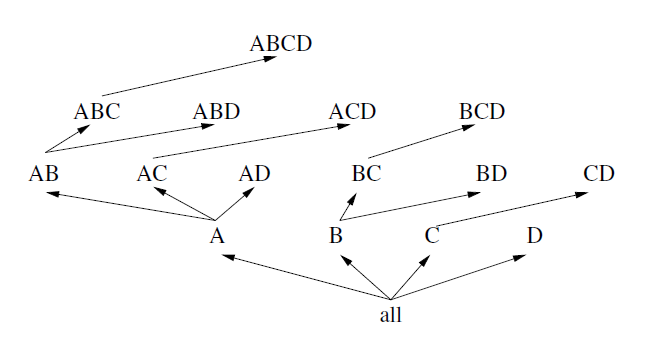


Figure 2: Bottom-Up approach

None of these previous algorithms incorporates the strength of both top-down and bottom-up cube computation to their full extent. Another iceberg cube algorithm is needed to accomplish this: the Star-cubing algorithm. This algorithm integrates the top-down and bottom-up cube computation with two important optimization techniques [2]. Globally, it shares aggregation by taking advantage of shared dimensions among the current cuboid and its descendent cuboids. Underneath, it has a sublayer that prunes as soon as possible the unpromising cells during the cube computation, exploiting the notion of shared dimension. The major computational properties of the four algorithms are listed in the table below.

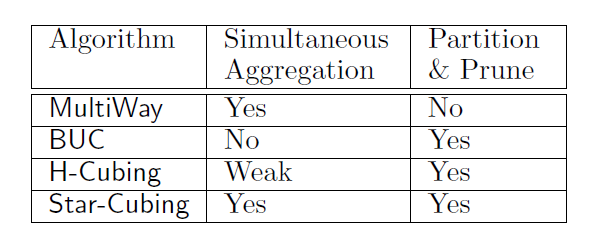


Table 1: Summary of the Four Algorithms

A new data structure, the star-tree, is introduced that explores lossless data compression by using an Apriori-like dynamic subset selection strategy. If a single dimensional aggregate on an attribute value does not satisfy the iceberg condition, the node can be replaced by \*. This node is now known as a star-node. The star-tree is a cuboid tree that consists of both these star-nodes and non-star nodes (attributes that passed the iceberg condition). To pick out which nodes are star nodes, a star-table is constructed for each star tree.

# Star-Cubing Implementation

Our implementation of the Star-cubing algorithm will be divided into the following parts: Pre-processing the raw data, generating the star-table, generating the star-tree, and running the Star-Cubing algorithm utilizing the previously generated star-tree and pre-processed data.

## Pre-Processing the raw data

In some of the tuples of our datasets, it is common for unknowns (marked as ?) to appear. To get rid of these, we search the array and replace all unknowns with star nodes from the beginning. For real numbered attributes, these values must be binned into partitions of equal length. To determine the bins, the attribute must be iterated through once to get the maximum and minimum value and then iterated through again to place the real numbers in their respective bin. The number of partitions are determined using a subset of the dataset.

## Generating the star-table

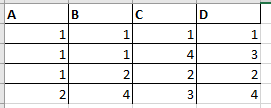
The next step is to create the star table. For an attribute the count of each unique value is determined. Any count that does not meet the iceberg condition/minimum support transformed into a star node, identified by a \*. The cuboids are then grouped according to their respective sample along with their count and appended to the bottom of the table. Each cuboid is assigned a single letter string to identify it for the sake of simplicity. If more than 26 unique values exist (the letters a through z), then double letter strings will be used for the subsequent cuboids.

## Generating the star-tree

Finally the star-tree needs to be constructed. Each tree has a root, head, and neighbor. The nodes in the tree have links to its surrounding nodes. A ‘left’ link to any child nodes, a ‘right’ link to sibling nodes, and a ‘parent’ link to the parent node. As the tree is being constructed, the previously created star-table is read in row by row.

## Verifying the Implementation

The *Python 3.9* implementation has been tested with the example from the textbook, dataset shown in figure3. The steps through which the Star Cubing is being carried out by the code, shown in figure 4, have been printed out for verification with textbook steps.

  
Figure 3: Dataset Example form Textbook

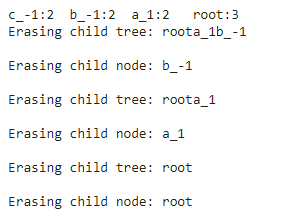
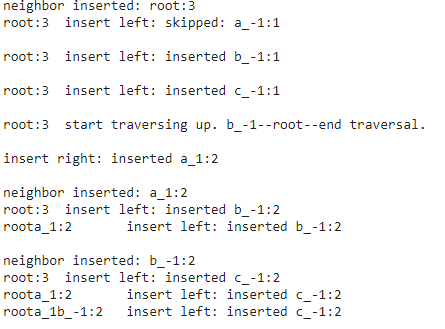


Figure 4: Star Cubing steps printed from implementation inputted with dataset in Figure 3  
Note: ‘-1’ has been used instead of ‘\*’ to indicate Star Cubes

# Experimental Set-Up

## Datasets

A total of 4 datasets have been used from the *UC Irvine Machine Learning Repository* [3], varying in size -Number of Instances-and dimension -Number of Attributes, however, they are all of an integer data type. As depicted in Figures 5 -6, “*Letter Recognition*” dataset has the largest number of instances, while “*Cargo 200*” has the largest number of attributes. “*Soybean Large*” and “*Divorce Predictors*” are both moderate in number of instances and attributes.

## 

Figure 5: Plot of Datasets VS Number of Instances

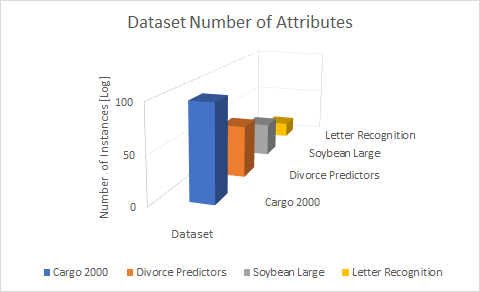


Figure 6: Plot of Datasets VS Number of Attributes

Table 2 summarizes the mentioned dataset metadata -Number of Instances, Number of Attributes, and whether it contains missing values- numerically.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Dataset Name | Number of Instances | Number of Attributes | Missing Values |
| 1 | Letter Recognition | 20,000 | 16 | No |
| 2 | Cargo 2000 Freight Tracking & Tracing | 3,942 | 98 | Yes |
| 3 | Soybean Large | 307 | 35 | Yes |
| 4 | Divorce Predictions | 170 | 54 | No |

Table 2: Dataset Details Summary Table

## Preprocessing Steps

The following steps have been carried out in order to get the data ready to be inputted into the data mining algorithm, Star Cubing:

1. Clean up: removing classification/clustering labels
2. Missing Value Treatment: Inserting ‘-1’ values for the missing data points suitable to the dataset
3. Initial Measurements of Value Range and Frequency:
   1. Small datasets: Using *Excel* to obtain the range of the frequency manually
   2. Big datasets: Using histogram samples as estimators
4. Minimum Support (Threshold) Selection: Using the information obtained from the initial measurements -part III, a couple of threshold values were selected.

## Choice of Threshold

In the case of small datasets, a manual measurement was done in *Excel* in order to gain necessary insight into the data for an educated guess of threshold candidates. First, the maximum and minimum values from data points were calculated to obtain range, then depending on the range either binning -in the case of “*Cargo 2000*” dataset- or discrete integer values - for the rest of datasets- were used for frequency calculations. The results of these frequency calculations are summarized in tables 3-6.

Since “*Cargo 2000*” dataset is comparatively large in size and broad in datapoint value range which need further discretization and binning,it is double treated, namely both measurement approaches have been used; obtaining frequencies manually and using histogram samples. The histogram samples for the “*Cargo 2000*” dataset are shown in figure 7. From table 5, it can be seen that the manual frequency range obtained based on descriptization is not a good representation of the frequency distribution, so in order to choose the threshold candidates, we relied on the values given by the histogram samples.

*“Letter Recognition*”

|  |  |  |  |
| --- | --- | --- | --- |
|  | Frequency Range |  |  |
|  | Max | Min | AVG |
| Max | 8,047 | 2702 | 3,574 |
| Min | 198 | 1 | 99.5 |

Table 3: “*Letter Recognition”* Datapoint Frequency Range Summary

“*Soybean Large*”

|  |  |  |  |
| --- | --- | --- | --- |
|  | Frequency Range |  |  |
|  | Max | Min | AVG |
| Max | 294 | 65 | 179.5 |
| Min | 12 | 0 | 6 |

Table 4: “*Soybean Large”* Datapoint Frequency Range Summary

*“Divorce Predictions”*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Frequency Range |  |  |
|  | Max | Min | AVG |
| Max | 115 | 48 | 81.5 |
| Min | 26 | 0 | 13 |

Table 5: “*Divorce Predictions”* Datapoint Frequency Range Summary

*“Cargo 2000”*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Frequency Range |  |  |
|  | Max | Min | AVG |
| Max | 3,942 | 4 | 1,973 |
| Min | 0 | 0 | 2 |

Table 6: “*Divorce Predictions”* Datapoint Frequency Range Summary

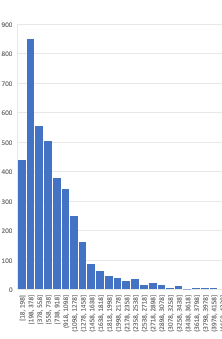
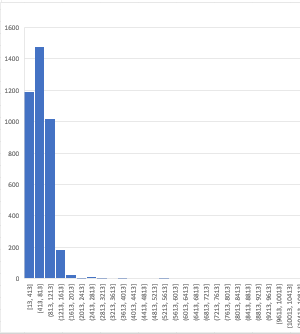
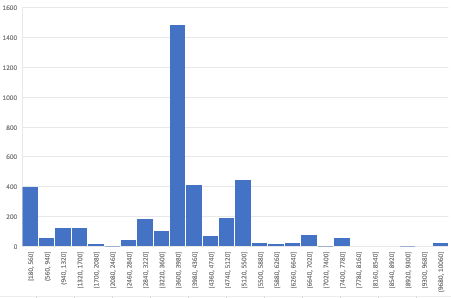
    


Figure 7: *“Cargo 2000”* Histogram Samples

## Threshold Summary Table

The threshold candidates for each dataset are listed in table 7.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dataset Name | Number of Instances | Threshold Values |
| 1 | Letter Recognition | 20,000 | 99, 198, 1450, 2702, 3574 |
| 2 | Cargo 2000 Freight Tracking & Tracing | 3,942 | 250, 400, 800, 1000, 1200 |
| 3 | Soybean Large | 307 | 12,65,123, 179 |
| 4 | Divorce Predictions | 170 | 13, 26, 37, 48, 81 |

Table 7: Dataset Threshold Values

## Metrics to be collected

For evaluation purposes the following utility metrics were collected:

* Execution Time
* Memory Resident Set Size (RSS) : the amount of memory being allocated to the process from RAM
* Virtual Memory Size (VMS): the entire memory space that can be accessed by the process including swap and shared memory
* Shared Memory Size

This has been achieved using *psutil 5.8.1* [4] library, a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in *Python*.

# Results & Discussion

## Execution Time VS Threshold

The execution time or wall time, have been measured for all the datasets with their corresponding threshold candidates. Figure 8 shows the plot of Execution Time Vs Threshold, a logarithmic scale has been used to assist representation of all 4 datasets that have sizes from 170 to 20,000. From figure 8, it can be seen that execution time will increase with size of the dataset,

Execution Time:

*“Divorce Predictions” <*  “*Soybean Large*” *< “Cargo 2000”* < *“Letter Recognition*”

Dataset Size:

*“Divorce Predictions” <*  “*Soybean Large*” *< “Cargo 2000”* < *“Letter Recognition*”

And decreases with the increase of the threshold in each dataset,

Execution Time based on thresholds:

*“Divorce Predictions”*

Threshold: 13 < 26 < 37 < 48 < 81 Slope: -0.642

“*Soybean Large*”

Threshold: 12 < 65 < 123 < 179 Slope: -0.337

*“Cargo 2000”*

Threshold: 250 < 400 < 800 < 1,000 < 1,200 Slope: -2.133

*“Letter Recognition*”

Threshold: 198 < 1450 < 2702 < 3574 Slope: -0.307

Observation: The dataset with the largest dimension or number of attributes has the biggest slope. This could be interpreted as the effect of threshold on number of attributes in a dataset, implying that the Star Cubing algorithm has the biggest effect on datasets with higher dimensions or larger number of attributes.

The obtained results match our theoretical expectations as it is expected for larger datasets to have larger frequent sets, consequently larger thresholds and lengthier processing time, hence, the data summarization will have a bigger effect.

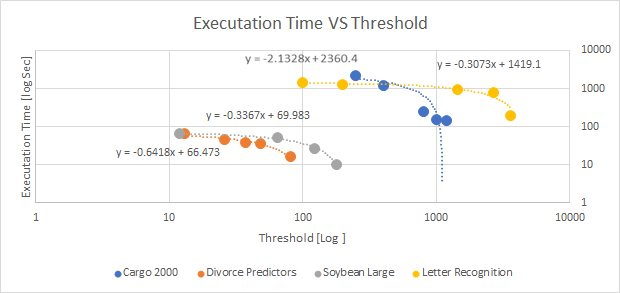
****

Figure 8: Execution Time VS Threshold for all Datasets in Logarithmic Scale

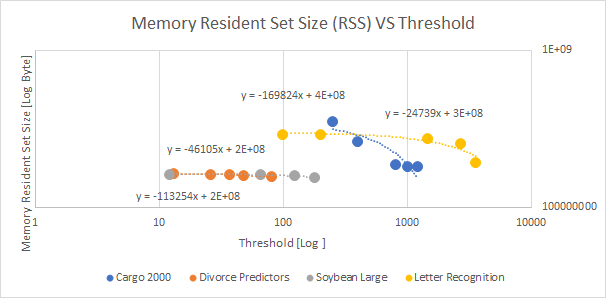
****

Figure 9: Memory Resident Set Size VS Threshold for all Datasets in Logarithmic Scale

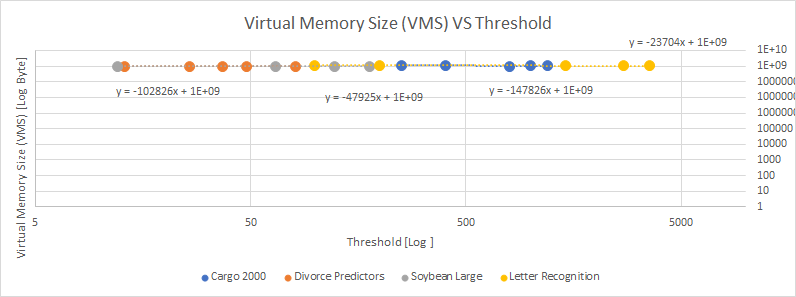
## Memory Resident Set Size VS Threshold

Similar trend is being observed for the relationship between Memory Resident Set Size and Threshold across all datasets, shown in figure 9. However, the effect in small datasets-*“Divorce Predictions”* and “*Soybean Large*”- is negligibly smaller than big datasets- *“Cargo 2000”* and *“Letter Recognition*”.

## Virtual Memory & Shared Memory Size VS Threshold

Since the implementation has been done through a single process, it is expected that the virtual and shared memory do not play a role or be affected by the threshold changes in a single dataset. On the bigger scope, however, they still increase with the size of the dataset, as bigger computing and storage resources are needed for their processes.

This is depicted by a horizontal line connecting points corresponding to different threshold values within each dataset, figure 10-11.

****Figure 10: Virtual Memory Size VS Threshold for all Datasets in Logarithmic Scale

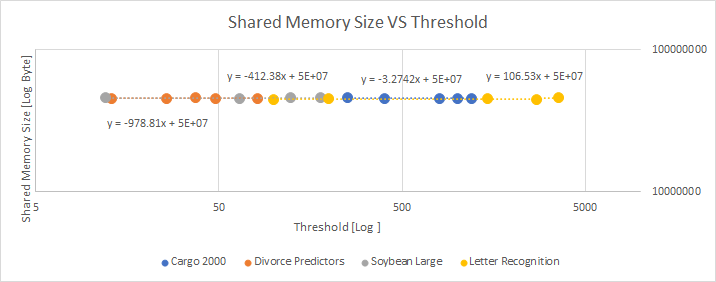
****

Figure 11: Shared Memory Size VS Threshold for all Datasets in Logarithmic Scale

**Algorithm Modification for Embedded Systems**

In the paper, **“Linear Programming-Based Optimization for Robust Data Modeling in a Distributed Sensing Platform”** [5]**,** a procedure to construct robust data models using samples acquired from a network of embedded sensory devices with limited resources, such as bandwidth and buffer memory, is presented. In it, local models are created at each node, using state variables, and transmitted to a specified node labelled TP (target point) where the streamed data is saved. The data from the sensor nodes is sent along multiple DCPs (Data Communication Paths) and the selected DCPs influence the modeling error as they help determine the experienced data loss and delays to the TPs. Data loss can occur when data stored in local buffers is overwritten before being forwarded along to the TP. Therefore it is important that the time delays be reduced. Having multiple DCPs is crucial as each one can yield different levels of performance for considered mobile energy source trajectories. To reduce the errors from data loss and time delays in a given set of traffic conditions, lumping is introduced.

Lumping eliminates the less significant state variables and simplifies the structure of the discrete representation of the state equations. Lumping might eliminate one or several consecutive variables and the level of lumping is determined by a certain condition/equation that includes the estimation of the minimum total error of the current and previous nodes, the temperature, and the bandwidth. The lumped data will be aggregated along the DCP, and for each node the lump level must be updated.

Considering the aforementioned description from the paper, a fundamental change that needs to be made to the star-cubing algorithm is to the minimum support. Rather than it being a static hyperparameter chosen by the user beforehand, it will be a dynamic value determined at every node along the DCP to the TP given the current traffic conditions, the errors, and the allotted power consumption. Another noteworthy change is that the star-trees (DCPs in this case) are not a depth-first traversal but a traversal based on which path will give the least error at each iteration.

# Conclusion

In this assignment, the Star Cubing algorithm was implemented in python and benchmarked across four different datasets, each of varying instance counts and dimensionality. This method is an efficient data cube computation algorithm that uses both data aggregation along dimensions and Apriori pruning given a specified iceberg condition. The iceberg condition of each dataset was determined by examining their histogram samples respectively. The experimental results were in agreement with the theoretical expectation that summarization through Star Cubing Algorithm has a bigger Execution Time and Memory Consumption impact on larger-either in dimension and size- datasets. Observation shows that the Execution Time undergoes a drastic change in the case of a dataset with higher dimension rather than the case of a dataset with larger size. Utility Analysis shows that the majority of summarization effects occur in Resident Set Memory and Virtual and shared Memory are not majorly affected due to the algorithm implementation using a single process. Similarly it is observed that the dataset with higher dimension undergoes the most drastic change in memory consumption. In the future if we were to implement the MultiWay, BUC, or H-Cube algorithms we would see that Star-cubing would outperform them all with the same datasets in terms of execution time and memory usage.

# References

**[1]**J. Han, M. Kamber, J. Pei. (2012). Data Cube Technology, *Data Mining: Concepts and Techniques* (3rd ed., pp. 187-242). Waltham, MA: Morgan Kaufmann Publishers.

**[2]**Xin, Dong, Jiawei Han, Xiaolei Li and Benjamin W. Wah. “Star-Cubing: Computing Iceberg Cubes by Top-Down and Bottom-Up Integration.” VLDB (2003).

**[3]**"UCI Machine Learning Repository", *Archive.ics.uci.edu*. [Online]. Available: https://archive.ics.uci.edu/ml/index.php. [Accessed: 14- Oct- 2021]

**[4]**"psutil documentation — psutil 5.8.1 documentation", *Psutil.readthedocs.io*, 2021. [Online]. Available: https://psutil.readthedocs.io/en/latest/#. [Accessed: 14- Oct- 2021]

**[5]**A. Umbarkar, V. Subramanian and A. Doboli, "Linear Programming-Based Optimization for Robust Data Modeling in a Distributed Sensing Platform," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 33, no. 10, pp. 1531-1544, Oct. 2014, doi: 10.1109/TCAD.2014.2334295.