Fall 2015

Improving the Accuracy and Robustness of Self-Tuning Histograms by Subspace Clustering

Sai Kiran Padooru
San Jose State University

Improving the Accuracy and Robustness of Self-Tuning Histograms by Subspace Clustering

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sai Kiran Padooru

December-2015

The Designated Project Committee Approves the Project Titled

Improving the Accuracy and Robustness of Self-Tuning Histograms

by Subspace Clustering

by

Sai Kiran Padooru

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

Dr. Tsau Young Lin	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Prasad Kopanati	Manyship,Inc.

ABSTRACT

Improving Accuracy and Robustness of Self-Tuning Histograms by Subspace Clustering

by Sai Kiran Padooru

Self-tuning histograms are a type of histograms very popular these days, as they allow the usage of multidimensional datasets. The main advantage of them is that they have a low computational cost due to their capacity to understand the dataset. Also, they proposed a better approach as they stay up-to-date and have adaptability to query patterns. According to the above, many researchers have worked on improving the accuracy of these type of histograms, which has led to the use of subspace clustering methods as initialization values. Following this approach in this study, a self-tuning histogram code was developed with the objective of comparing two different subclustering methods (Proclus and Mineclus) for the initialization values. The script was tested with two different datasets (2-D and 4-D). It was found that the Proclus algorithm performed better than the Mineclus. Also, it was proved that the size of the bucket was crucial to achieve more accuracy (Khachatryan, Clustering-initialized adaptive histograms and probabilistic cost estimation for query optimization, 2012).

ACKNOWLEDGEMENTS

I would like to thank my Advisor Dr. Tsau Young Lin for his continuous guidance and for believing in me. I would also like to thank the members of the committee Dr. Robert Chun and Mr. Prasad Kopanati for their valuable input and to monitor the progress of the project closely. In addition, I would like to thank my parents and friends for being there with me at every step along my Masters program.

TABLE OF CONTENTS

APPENDIX

LIST OF TABLES

LIST OF FIGURES

Introduction

Histograms are a representation of the distribution of numerical data. An estimation of the probability distribution of a continuous variables, and have become more important tool for data summarization techniques, as they don't assume any type of distribution. Also, histograms are widely used for commercial as for scientific purposes in several fields, firstly regarding databases approaches due to its flexibility. For the present paper, we will be referring to query optimization [1].

Query optimizers rely on query predicate selectivities to estimate the cost of different plans in an accurate way. The importance of the selectivities lies on its use when comparing physical access method, as well as when choosing the method and the order of joints. This is why most of the times histograms are used for that, due to its potential to obtain selectivity estimates. Hence, the objective of this paper would to improve the use of histograms in query optimizers [1].

In order to do that, we should consider the two different approaches to histogram construction: Static and self-tuning histograms, to further on select the best suited paradigm for our purpose.

As mentioned above, on the first hand, we have static histograms. They are built by analyzing the whole set of data, and required to be built again over time to be updated. Also, the construction of a multi-dimensional static histogram it is really expensive, both

in terms of time and space consumed, and that is just for non-optimal histograms.

As a way to solve that issue, a new approach was taken to create static histogram by escalating against number of tuples, instead of escalating against number of dimensions. This way it was possible to remove less relevant attributes. Another methodology to do so, was using the SASH framework, which skipped irrelevant attributes for the whole dataset. Still, dimensionality reduction techniques, were not completely solving static histogram problems, as (even for non-extensive dimensional datasets) it was not taking into account local correlations, which could affect the accuracy of the histogram. And even though there are techniques that take advantage of local correlations, they required the system to know about them previous to its use, which is not possible when talking about large, multi-dimensional datasets [1].

Moving on, on the other hand, we have traditional self- tuning histograms. This approach, opposed to the above paradigm, use feedback from the query to study the dataset, which reduces built cost significantly. Also, they have shown impressive advantages against static histograms in terms of estimation accuracy, as well as in their adaptability to query patterns and capability to stay up-to-date to the changes of the dataset [1].

The data structure method used for the paper was STHoles, which works by finding rectangular regions within the dataspace with a very uniform density. Unfortunately, as for the static histograms, traditional self-tuning histograms also failed when talking about data spaces of high dimensionality, mainly because of the bad selection of initial top level buckets. This was a major issue, because irrespective of the training efforts, it was not going to be accurate [1].

In order to solve the issue above mentioned, subspace clustering for self tuning histograms were introduced. Meaning that it uses the self-tuning capability to understand the dataset from scratch, but combining it with an initial configuration obtained using subspace clustering. This way, the self-tuning procedures would only be used for refining the histogram, but not creating it, fixing the issues regarding the bad initial top level buckets. Taking this into account, the main objective of this paper was to create a script able to run a self-tuning histogram, using both the Mineclus and the Proclus algorithms to create a sub-cluster initialization [1].

Background

Histograms are a abridged portrayal of data. They are used in many databases and alike applications. A histogram represents the distribution of the values of an attribute in a cluster. Histograms are used in databases for the estimation of selectivity.

Clustering:

Analysis of clustering of data found objects that are similar in a sense to one another. The members of a cluster are more similar to each other that are like members of other groups. The aim of the clustering analysis is to find clusters of high quality that the inter-cluster similarity is low and the similarity of intra-cluster is high [5].

Clustering, such as classification, are used to segment the data. Unlike the models of classification, clustering segment the data into groups that were not previously defined. Models for classifying data segment of assigning classes defined previously, that are specified in a destination. The models of clustering do not use a destination [5].

Clustering is useful to explore the data. If there are many cases and non-obvious groupings, clustering algorithms can be used to find natural groupings. Clustering can also serve as a preprocessing step of useful data to identify homogeneous groups on which to build models supervised [5].

Clustering can also be used for the detection of abnormalities. Once the data has been segmented into clusters, you might find that some cases do not fit well in any clusters. These cases are anomalies or outliers [5].

Cons of Static Multi-Dimensional Histograms

We cannot change static histograms after their building is over. This implies that the constructing of histograms needed to be done periodically to see modifications in the data. The building costs of histograms which are high dimensional amplify quickly with different data set dimensions. For datasets of high dimensions, the very grand cost of building and way to build histograms again and again, is a greater hurdle to the implementation of approaches multi-dimensional [1].

Self Tuning Histograms

These type of histograms use the final outcomes of the consultations that have already implemented to improve their own. This type of learning is superintended. These are adept to retaliate the construction costs, acclimate to the assigned tasks and consultation are commonly treated as a formative substitute to the ways which are static, which builds them and allow it to be consistent [1].

These utilize the consultation assessment to modify themselves. The main aim in the back of these histograms is once the execution of queries is over, the outcomes are familiar, and these outcomes can be utilized to filter the histogram. This is substantially a way of schooling histograms superintendedly [1].

Figure 1 shows how the user provides a query and is analyzed by the database management system and are polished [1].Histograms are assessed by the optimizer during the process of optimization . Then, the query is run by the system. The user gets the results of current consultation; the same outcomes are also seen in the histogram and are used for making its structure well filtered.

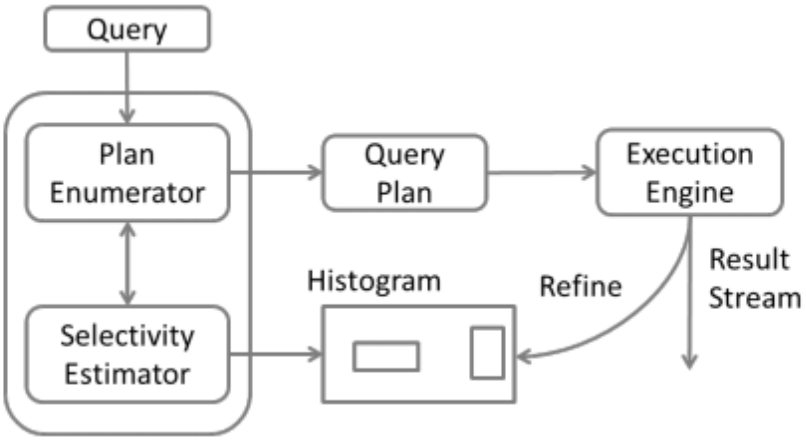

Figure 1. The STHoles Histogram [1].

STHoles generally has a root bucket in which it includes the entire space of values of attribute. Figure 2 is the bucket r [1]. In buckets next to each segment is the number of tuples.

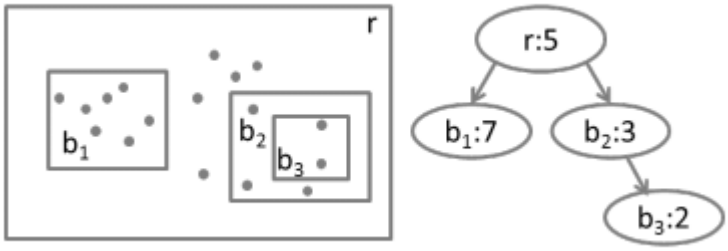

Figure 2. Left shows STHoles histogram and right shows bucket tree [1].

We will begin with cardinalities as estimates and then we will display how new buckets are added by utilizing the consultation comments, and finally, as compact the histogram, space is freed up to cover space of the budgetary rules [1].

Algorithm 1: STHoles "Estimate, Refine, Compact" cycle [1].

Input: H: STHoles, q: Query
Output: H: Refined Histogram
{ Estimation}
for all $b \in H$ **do**
if $b \cap q \neq \emptyset$ **then**
$estimate \leftarrow estimate + n(b) \cdot (vol(b \cap q)/vol(b))$
$Intersections \leftarrow Intersections + (b \cap q)$
end if
end for
$results \leftarrow q.Execute()$
{Histogram refinement}
for all intersection $(b \cap q) \in Intersections$ **do**
Compute Tuples in $b \cap q$ using $results$
Add new bucket(s) to H
end for
{Compacting the histogram}
Remove buckets from H to meet the space budget

Cardinality Estimation

To proximate tuple cardinalities, STHoles makes use of continuous value assumptions.

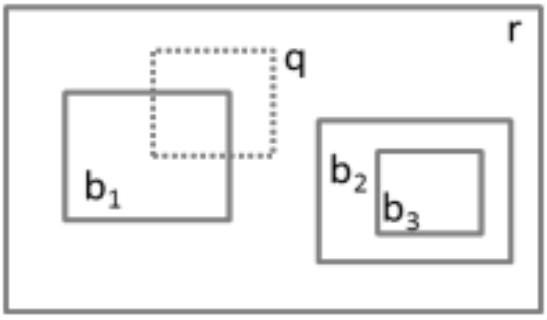

Figure 3: histogram(STHoles) with query

Here dashed rectangle is the query q. It intersects with r and b1, histogram buckets. By applying the Continuous Value Assumption, STHoles estimates the number of tuples in intersection of a bucket and a query to be proportional to the volume of the intersection.

METHODOLOGY

Following is the methodology used to achieve the objectives of this study. First, a brief explanation of how the subclusters methods (Mineclus and Proclus) work will be made. Second, the algorithm to convert the cluster results to buckets will be explained. Third, the STHoles data structure will be shown as it is the structure used to manage the histogram. Lastly, the self-tuning algorithms will be shown.

In order to completely understand the methodology, some important definitions are required.

- **Histograms**: It is a data structure that allows to represent a dataset by discretizing the domain into spaces (buckets).
- **Bucket**: geometrical space that represents a number of elements (tuples), uniformly distributed within it. Buckets do not contain the information from each tuple, but only the total count of them.
- **Tuple**: every entry in a dataset. It has a geometrical position in every dimension.
- **Frequency**: is the number of tuples within a bucket.
- **Dimension**: represents every attribute of the dataset (should be a numeric value).
- **Cluster**: is a representation of a section of a dataset. Each cluster contains one or more tuples. Contrary to bucket, they don't have geometric boundaries.
- **Query**: is a geometrical spaced restriction that allows to evaluate the performance of the buckets, in order to represent the histogram.

Both Mineclus and Proclus are subspace clustering methodologies used to set up the initialization of the buckets. The scripts used to run both methodologies were available online from the Open subspace framework of Weka. As the script making process was not part of the approach of this paper, no further explanations will be given.

Converting clusters to buckets

When creating clusters with either one of the approaches above mentioned, there are not geometric boundaries, but just a group of tuples assigned to a cluster id. This is an issue because the histogram needs specific boundaries as the idea is not to use the position of the tuples, but to compact the dataset into different frequencies within the dataspace. In order to fix that the following algorithm was used.

First of all, the median of each dimension of the tuples was set as the center of the cluster, to then be able to create the specific boundaries of the bucket at a distance of 5% from the median to all directions.

Once this first buckets was created, an iterative process begins were the dimensions of the bucket are increased by 5% each time, to compare both buckets and select the most suitable one. This process ends once the new bucket is less suitable than the one before.

To evaluate the above comparison, the following equation was used:

$$\lambda(RR, RR') = |RR' \cap C| - |RR \cap C| + |RR - C| - |RR' - C| \quad (1)$$

Where,

RR: Old bucket volume

RR': New bucket volume

C: Cluster volume

In order to calculate the intersections, the following equation was used:

$$|RR \cap C| = |RR| * \frac{E[m]}{M} \quad (2)$$

Where,

M: Number of tuple within RR (or RR')

E[m]: Number of tuples within $RR \cap C$

Whenever λ from equation (1) for a new bucket is greater than the old λ, it is said that the new bucket is suitable, hence a new iteration begins [1].

STHoles data structure

STHoles is the data structure used to represent the histogram. It has a tree structure, meaning that there is a parent bucket that has childs, and those childs may or may not be parents to another childs. In Figure 4, it is shown the above explained structure [1].

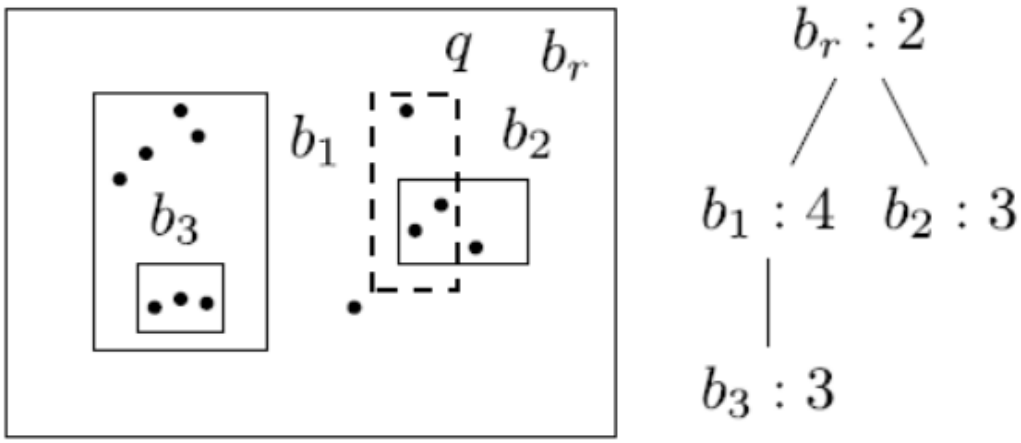

Figure 4. STHoles structure.

Self-Tuning Algorithm

The self-tuning algorithm is composed of three different main steps:

1. Query: In this step a query is created. To be able to create it, a random point is formed for each dimension, and the longitude of the query in that dimension is

obtained as 1% from that point to each direction.

2. Shrink: The objective of this step is to divide the buckets intercepting with the query. In order to explain this algorithm a little better, a graphic example is shown in Figure 5.

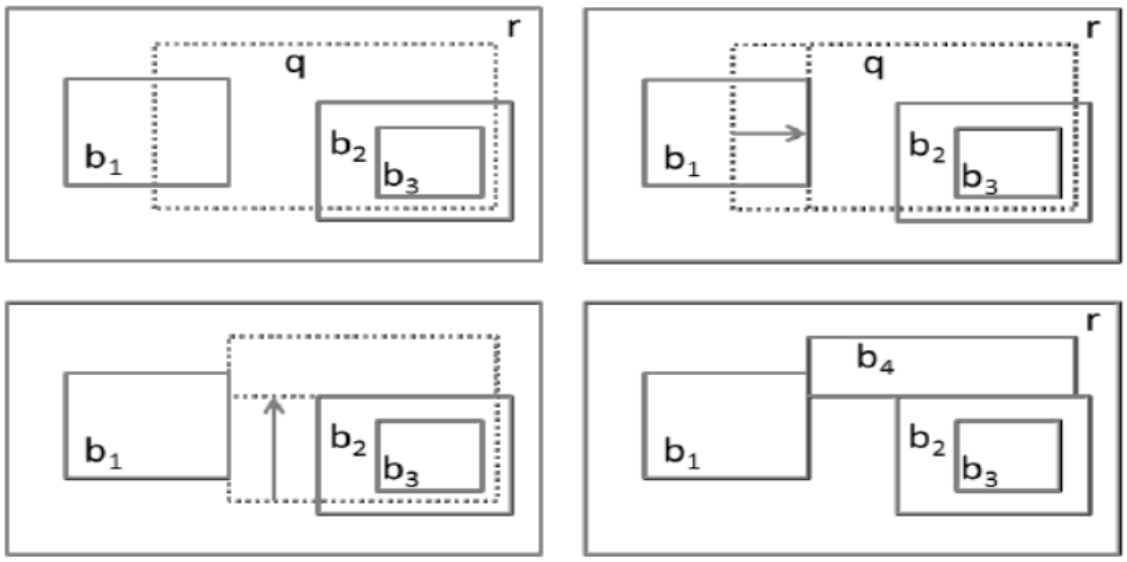

Figure 5. Shrink Algorithm Example

The above figure shows the shrinking method between bucket r and query q, where b1 and b2 are children of r, and b3 is the child of b2.

The process is as follows:

- All children of r that intersect q are determined. For this case: b1 and b2.
- For every intersected child, the boundaries of q are changed, until there is no

intersection. They could be adjust by any dimension. For the case: q was resized to the right, to avoid intersection with b1. Then, q is resized upwards to elude intersecting b2, and hence b3 [1].

- For each child, every resizing dimension is considered. The selected resized query will be the one with highest volume within all the possibilities.
- Once the above is done, the process is repeated for the intersection of original q with buckets b1 and b2 [1].

3. Some buckets are merged, so the total number of buckets remains constant. To further explain this, please see Figure 6:

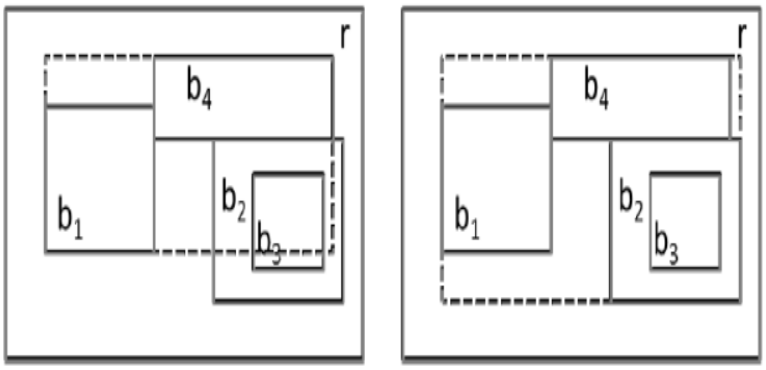

Figure 6. Merge algorithm example.

The merging process consists on merging together two buckets to create a new bigger bucket. As there is no knowledge of what would be the best merging option, all possible merge options between parent-child and sibling-sibling are evaluated, and the combination with less penalty value will be the chosen one to be merged [1]. The next steps are followed:

- For Figure 6 the merge combination that is going to be evaluated is between sibling b1 and b4.
- First of all, the smallest possible bucket, that contains both buckets, is generated. This new bucket will be called bnew for this example.
- The intersection between bnew and any other bucket is determined. For this case, it is intersecting b2 and b3.
- bnew boundaries are increased until it contains all buckets that were being intersected. This process is repeated until no intersection remains. For this case: b1 and b4 buckets are replaced by bnew bucket, which will contain b2 bucket as a child.
- The penalty for this merging option is calculated, according to equation 3.

$$\text{Penalty}(b1, b4) = \left| n(b1) - n(bnew) * \frac{vol(b1)}{vol(bnew)} \right| + \left| n(b4) - n(bnew) * \frac{vol(b4)}{vol(bnew)} \right| \quad (3)$$

- After determining the penalty for all possible merging options, the one with lower penalty will be merged.

4. Lastly, all the frequency and volumes are updated according to the steps below. The algorithm proceeds to evaluate a new query [1].

Case Study

The methodology explained above, was implemented with two different datasets: cross dataset and 4D dataset.

The first one, was a 2-Dimensional dataset, with 21000 instances. The graphical representation of it can be seen in Figure 7.

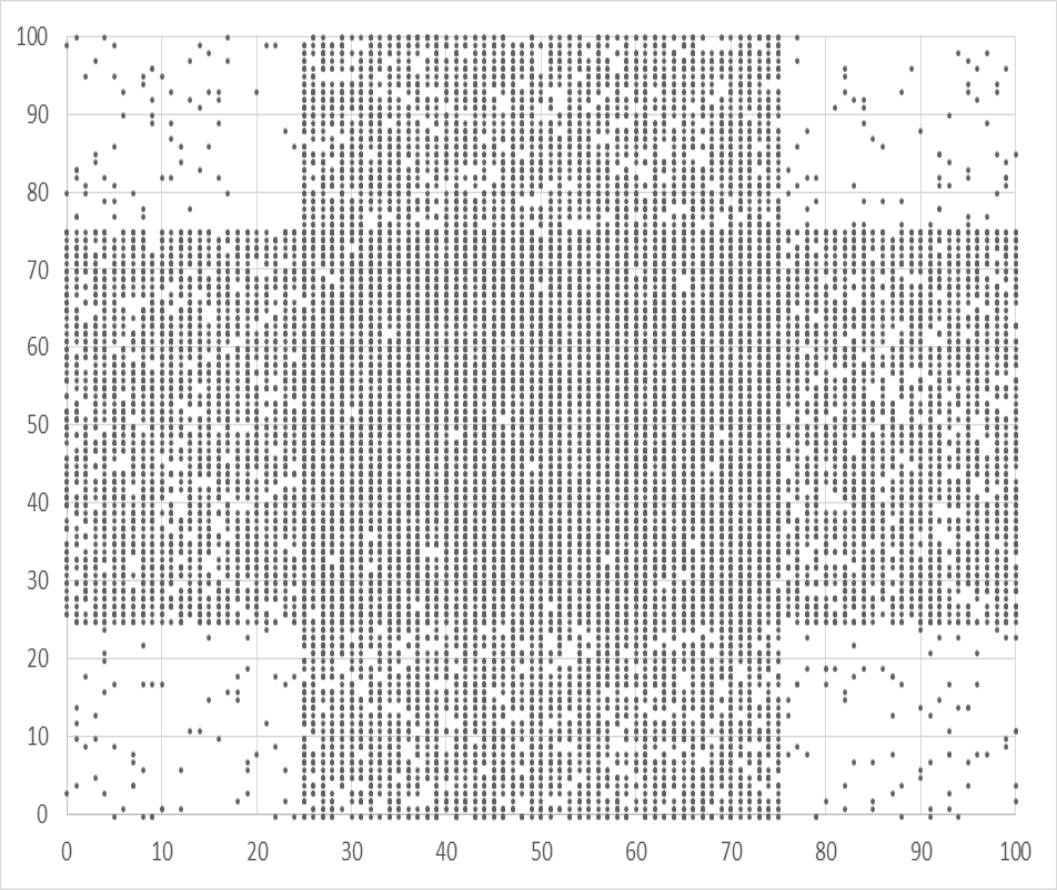

Figure 7. 2-Dimensional dataset representation.

The second one, a 4-Dimensional dataset, it is a real dataset for the Road 4D of North Jutland, in Denmark. It has approximately 400000 instances.

For each dataset evaluation, both Proclus and Mineclus subclustering algorithms were used. Additionally, three different maximum bucket limits were stablished, of 50, 100 and 150.

Results

Following there are the results for both datasets, after running a 1000 queries for training, and another 1000 to compare it with the original dataset.

For both datasets the parameters used to implement the Proclus and Mineclus algorithms are find in Table 1 and Table 2, respectively.

Table 1. Proclus parameters used for both datasets.

PROCLUS PARAMETERS

PARAMETER	VALUE
K	50
D	4

Table 2. Mineclus parameters used for both datasets.

MINECLUS PARAMETERS

PARAMETER	VALUE
A	0.01
B	0.1
K	50

M	-1
N	1
W	10

Cross dataset results

In Figure 14, the results for Cross dataset are shown. There, a normalized error, for both subclustering algorithms, is reported for three different bucket limits (50, 100 and 150).

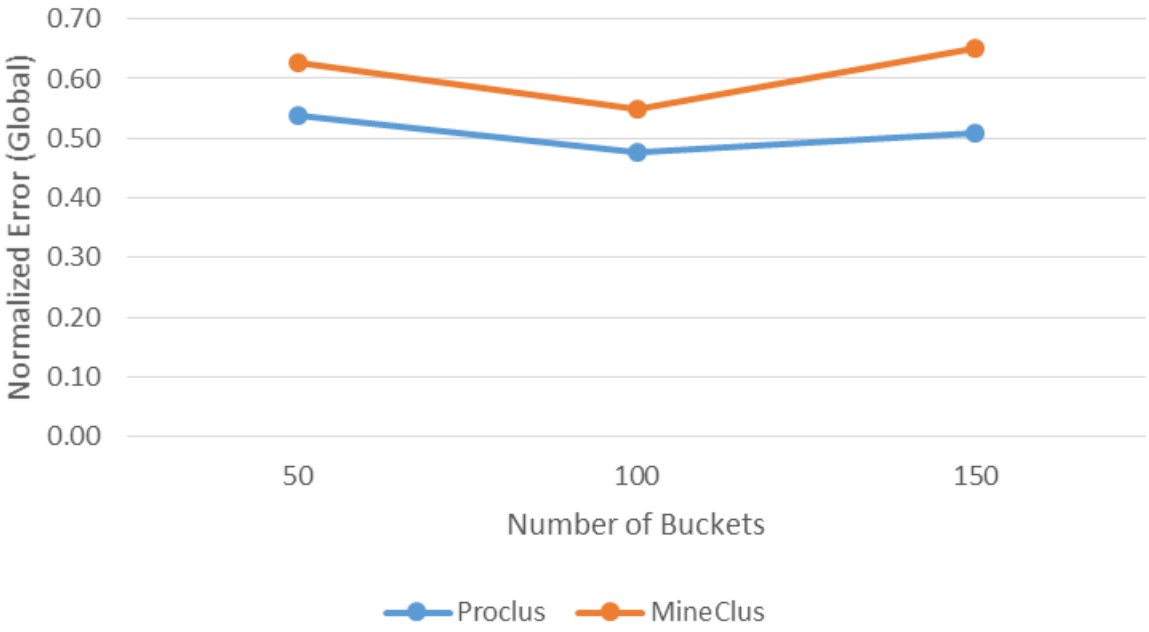

Figure 8. Normalized error for Cross dataset.

The normalized errors represent a percentage difference between the self-tuning histogram and the original dataset, for 1000 different queries. In this case, it can be seen, as a general observation, that the Proclus algorithm has a lower error margin than the Mineclus algorithm, which means it approaches better to the original dataset.

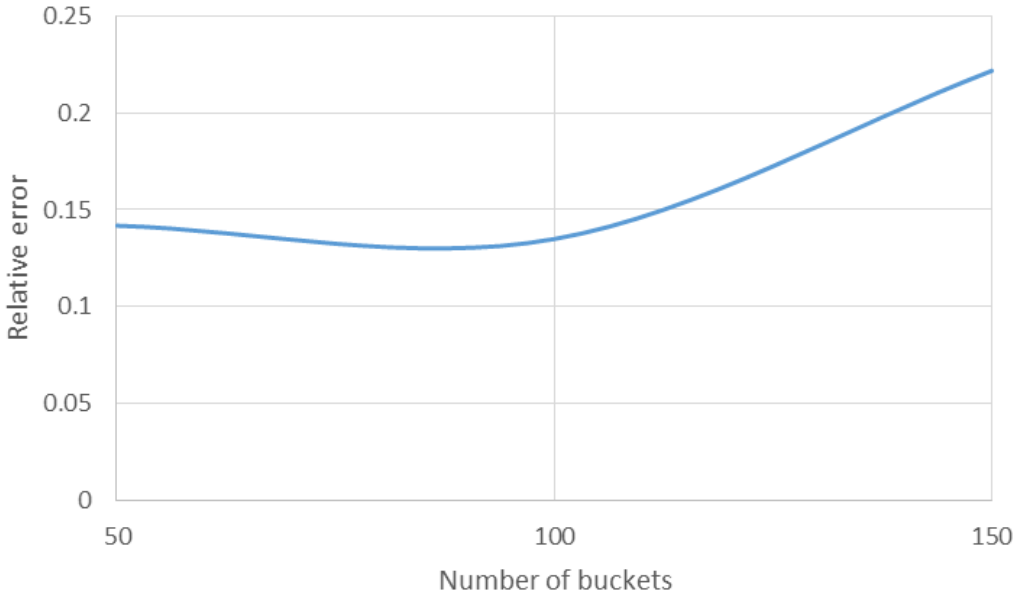

Figure 9. Relative error between Proclus and Mineclus algorithms.

Also, in Figure 9, it is shown the relative error between both subclustering algorithms. From this it can be determined that Proclus algorithm is, in average, a 16.6% more accurate than the Mineclus algorithm.

Lastly, from the above figures, it is possible to see that 100 buckets is the case with best results, with a lower error rate in both cases. This is mainly due to bucket size, where when there are 50 buckets, they group together a wider dataspace; and on the other hand, when there are 150 buckets the size is much smaller making many compatibles groups be separated [1].

4D Dataset results

In Figure 10, the results for the 4D dataset are shown. There, it is seen a normalized error, for both subclustering algorithms, for three different bucket limits (50, 100 and 150). As mentioned above, this error represent the percentage change between the representation of the dataset by the histogram, and the original dataset itself.

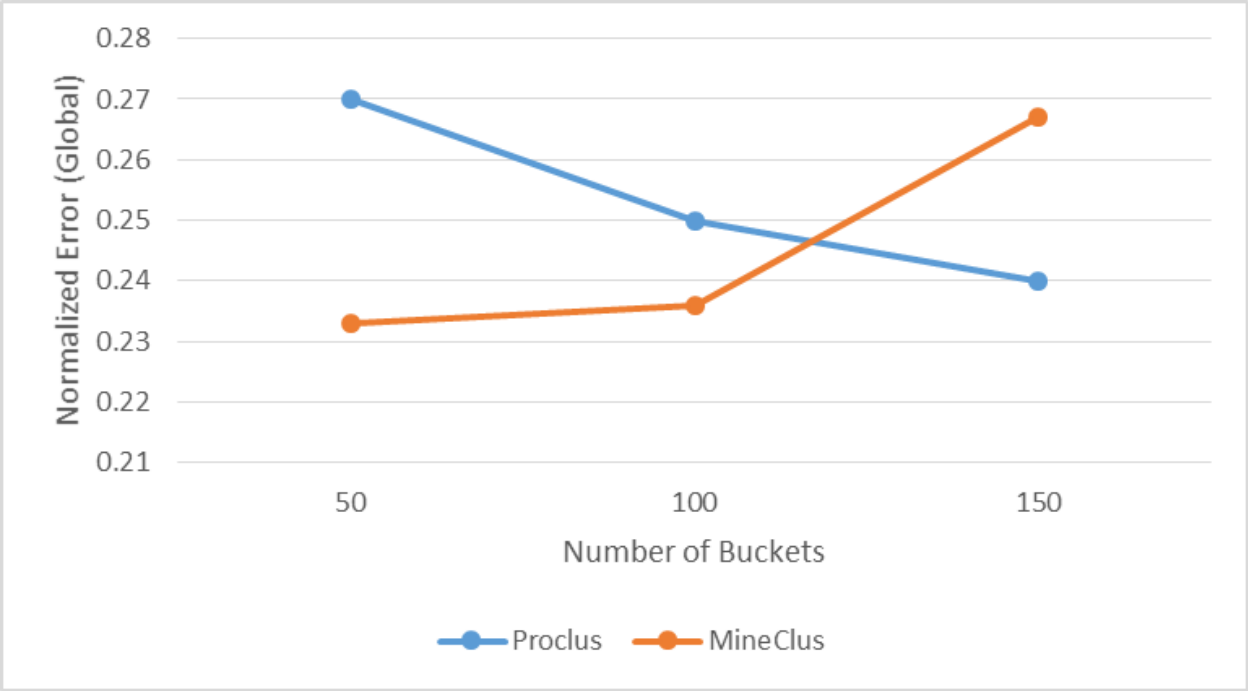

Figure 10. Normalized error for 4D dataset.

It can be seen that, for both Mineclus and Proclus algorithm, there is a low error. And, even though the Mineclus algorithm presents lower errors for 50 and 100 buckets, it

retrieves a bigger error for 150 buckets. From there, it can also be concluded that Proclus algorithm's error decreases when more buckets are generated, for this case, so it would be expected a much lower error for 200 and more [1].

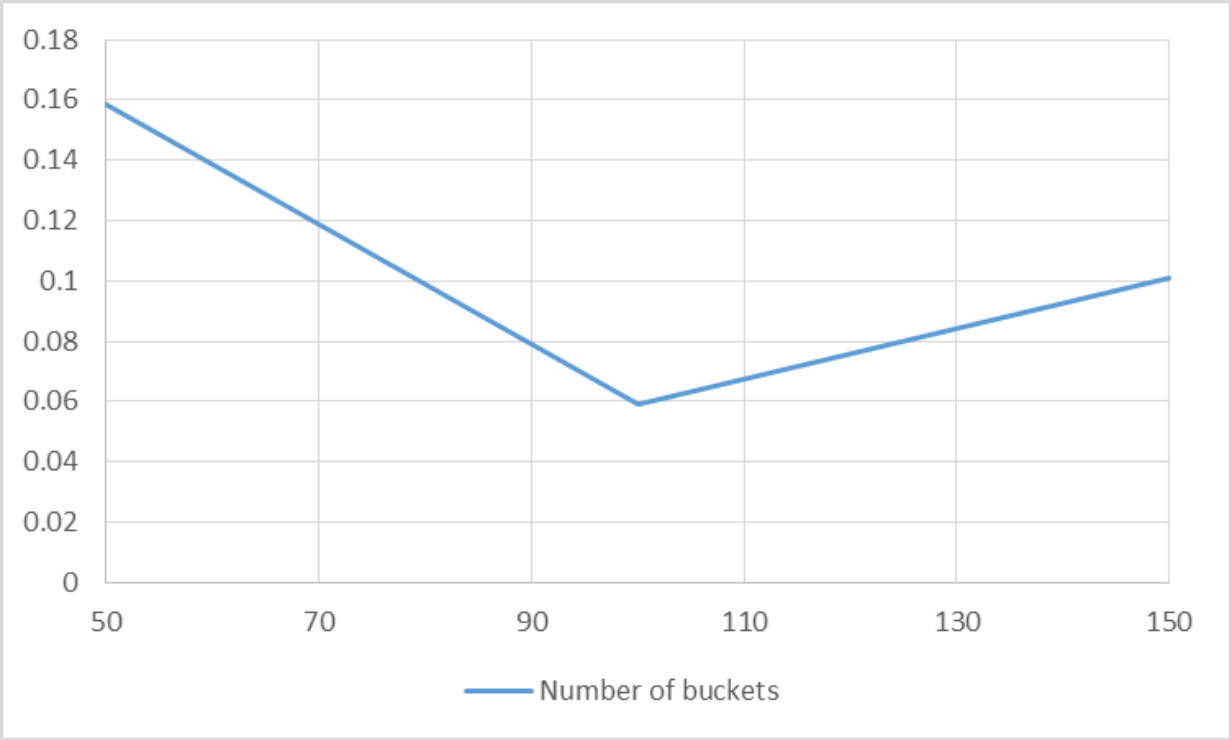

Figure 11. Relative error between Proclus and Mineclus algorithms.

Figure 11 shows the relative error between the Proclus and Mineclus algorithms. From there, it can be said that the difference between the results of each algorithm is lower when 100 buckets are created.

Future work

As future work, it is proposed to consider the quality of the results of the sub clustering methods. The objective of this would be to improve the initialization values of the self-tuning algorithm, as the results of the sub clustering methods vary highly according to its own initialization, so it is important to understand how good they are before proceeding with the methods. This way, it will be possible to determine the best values to retrieve results as close to the original dataset.

Also, it is important to work on the time when running the scripts, as at the moment the computational cost is high when running big histograms.

Conclusions

- A self-tuning algorithm was implemented, using two different subclustering approaches as the initialization values.
- The script was evaluated in two different datasets, of two and four dimensions, in order to prove the scope of it.
- For the Cross dataset, it was possible to determine that the Proclus algorithm is 16.6% more accurate than the Mineclus one, meaning that it represents better the original dataset.
- Also, for the same dataset, it was concluded that 100 buckets gives more accurate results, due to the bucket size. However, the difference in accuracy, with 50 buckets is not as big (with a difference of 0.06), but the second one runs much faster.
- For the 4D dataset it was determined that the Mineclus algorithm was a better approach than the Proclus algorithm until 100 buckets where created. However, the Proclus algorithm performs better every time when creating more buckets, achieving a better error than the Mineclus algorithm for the case of 200 buckets. This meaning that the Mineclus was more accurate than the Proclus, at representing the original dataset, when lower number of buckets were created, and vice versa when more buckets were created [1].

References

[1] Khachatryan, A. (2012). *Clustering-initialized adaptive histograms and probabilistic cost estimation for query optimization.*

[2] Khachatryan, A., Müller, E., Stier, C., & Böhm, K. (2015). Improving accuracy and robustness os self-tuning histograms by subspace clustering. *IEEE transactions on knowledge and data engineering, 27*, 2377-2389.

[3] E. M€uller, S. G€unnemann, I. Assent, and T. Seidl, "Evaluating clustering in subspace projections of high dimensional data," Proc. VLDB Endowment, vol. 2, no. 1, pp. 1270–1281, 2009

[4] Guojun Gan, Chaoqun Ma, Jianhong Wu. Data clustering : theory, algorithms, and applications

[5] Data Mining Concepts. Retrieved from

http://docs.oracle.com/cd/B28359_01/datamine.111/b28129/clustering.htm.

APPENDIX

Additional Screen-shots

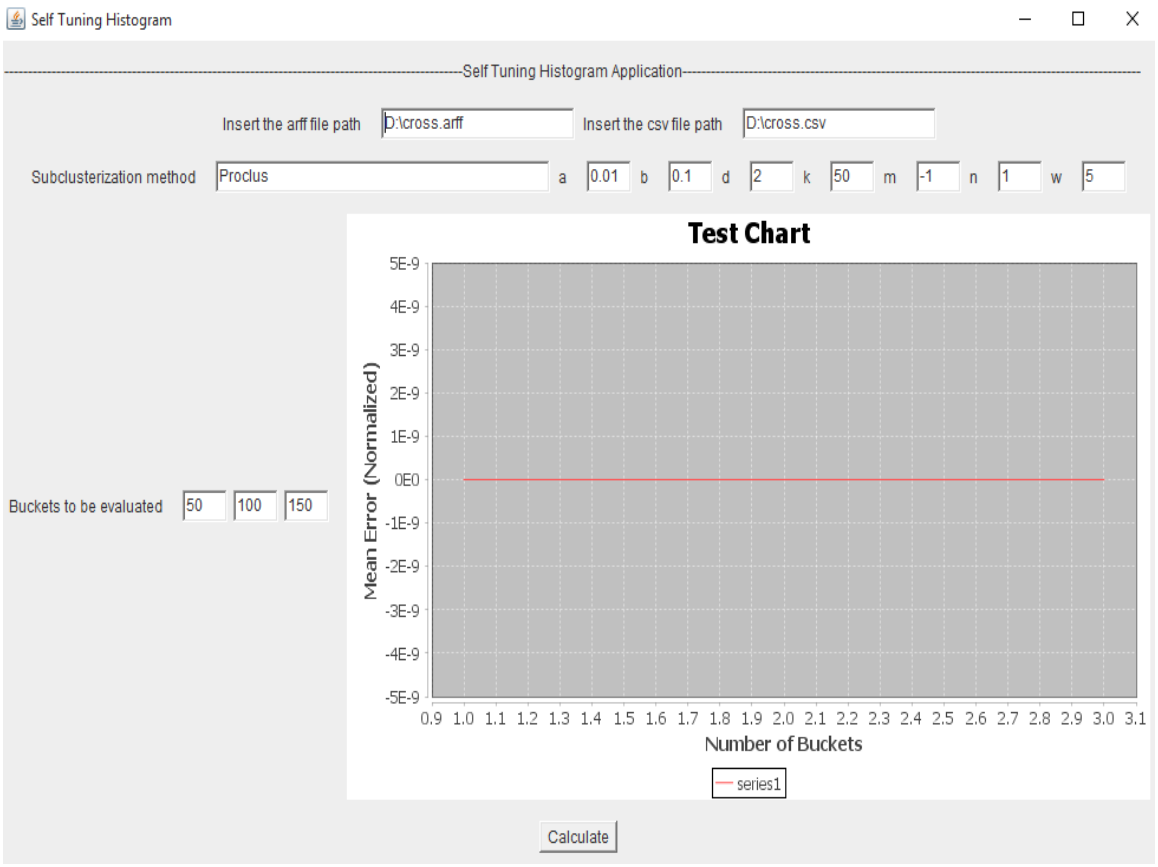

Figure A.12. Calculation of Results for Proclus Method

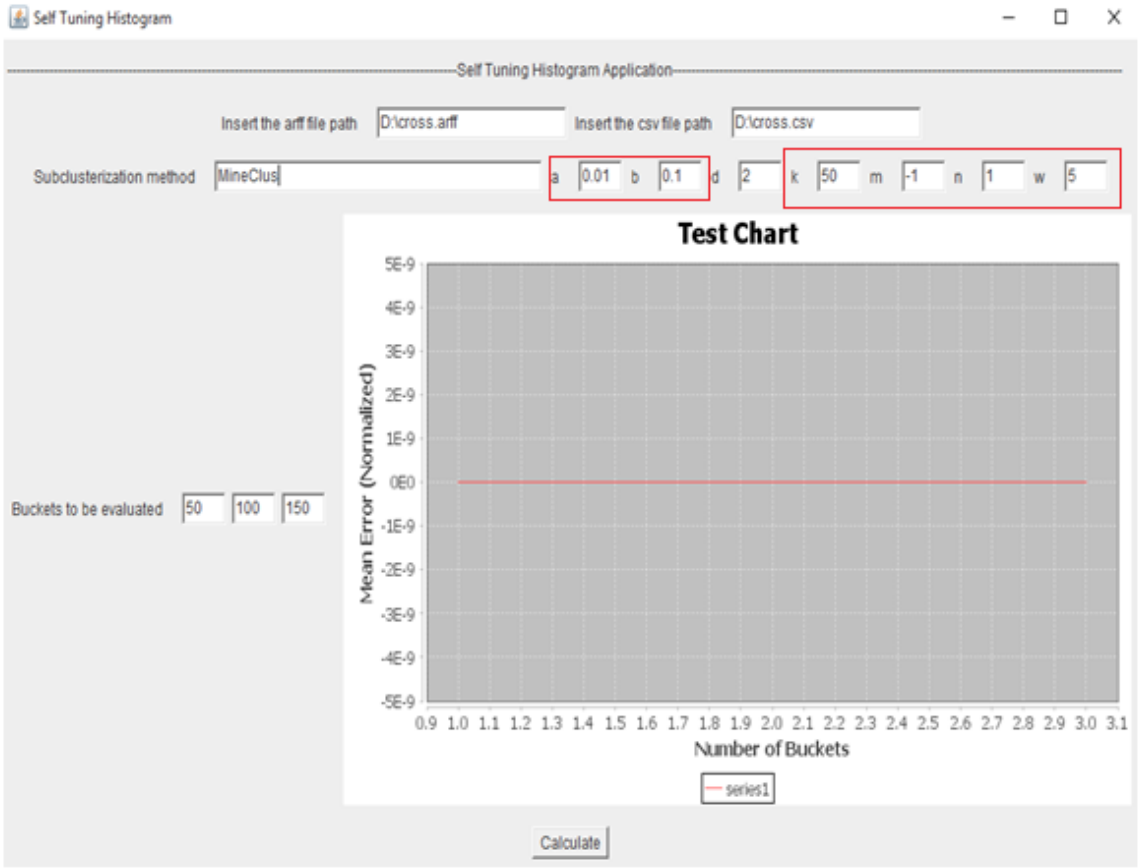

Figure A.13. Parameters used for Mineclus Method

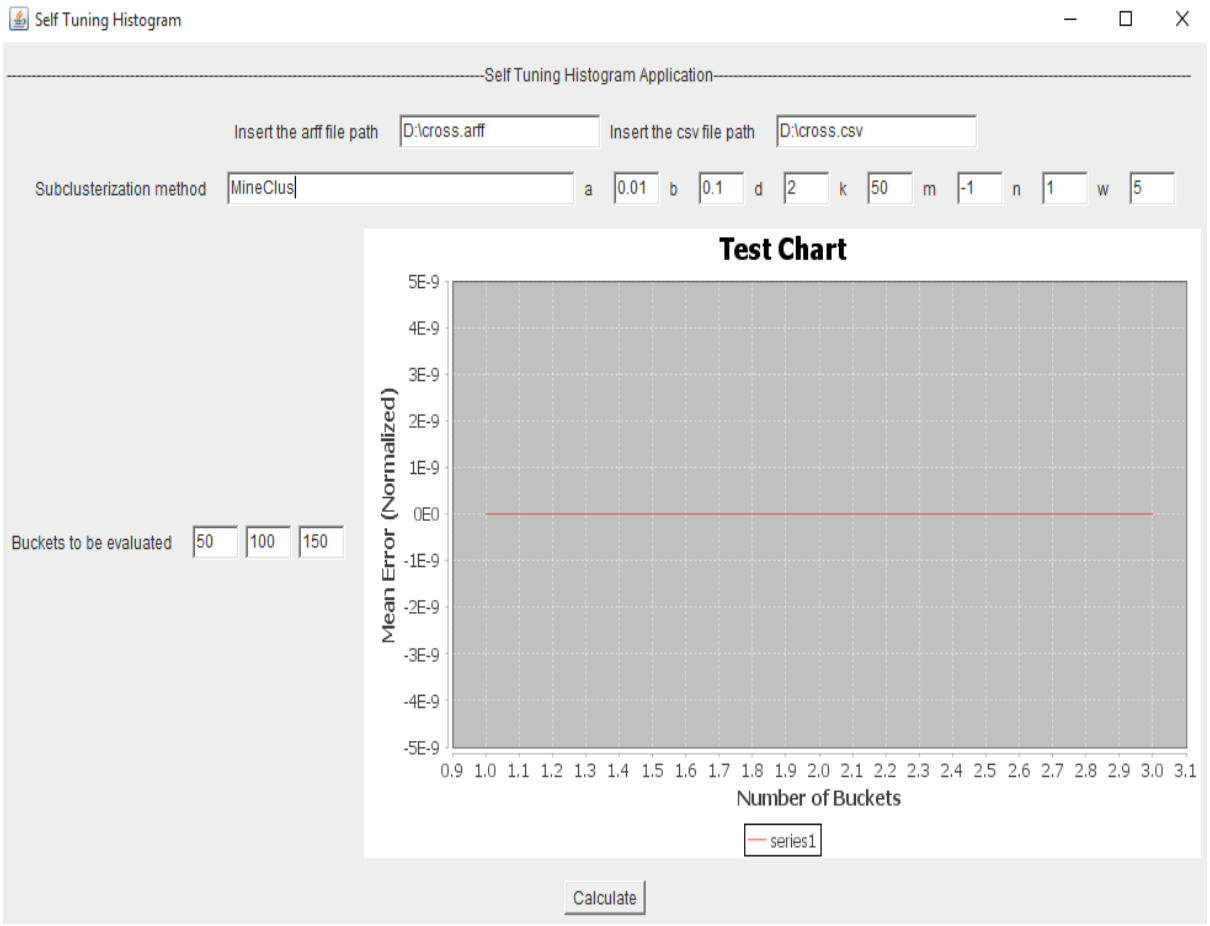

Figure A.14. Calculation of Results for Mineclus Method

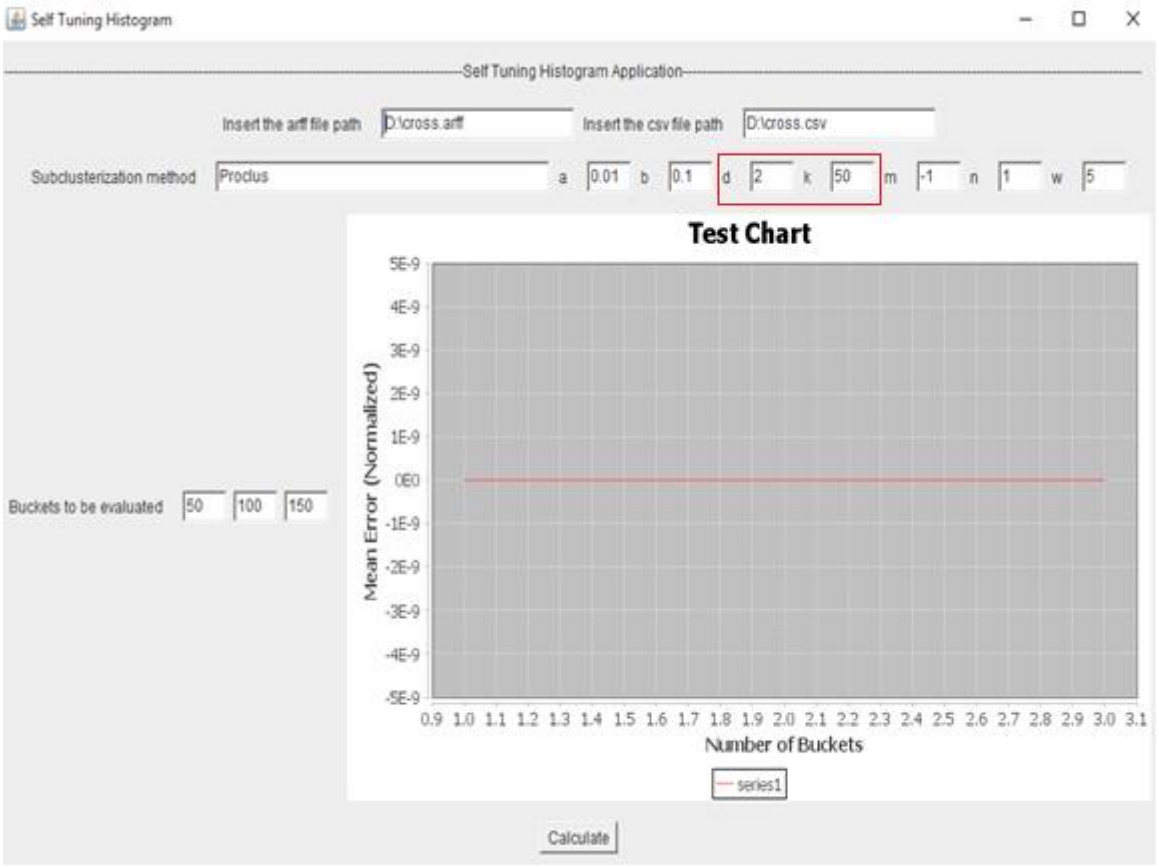

Figure A.15. Parameters used for Proclus Method