Reverse engineering the ovine gene network under Psoroptes Ovis infestation.

Supervisor: Dr. PierLuigi Frisco

Student: Ioannis Efstathiou
ID: H00017853

August 2012

Computer Science
School of Mathematical and Computer Sciences
Dissertation submitted as part of the requirements for the award of the degree of MSc in Artificial Intelligence

Table of Contents
Section 1: Introduction...........................................................................................................................................5
  1.1 Statement of non-plagiarism.................................................................................................................................5
  1.2 Acknowledgements..................................................................................................................................................5
  1.3 Abstract.....................................................................................................................................................................5
  1.4 Introduction............................................................................................................................................................6
  1.5 The Aim..................................................................................................................................................................7
  1.6 The Objectives......................................................................................................................................................8
  1.7 What Follows..........................................................................................................................................................9
Section 2: Literature Review .....................................................................................................................................10
  2.1 Introduction and the importance of the problem..................................................................................................10
  2.2 Reverse Engineering the Gene Regulatory Networks (GRNs)............................................................................12
    2.2.1 Complexity of Boolean Algorithms..............................................................................................................16
    2.2.2 Potential conclusions and the final decision..................................................................................................20
  2.3 Fuzzy logic and Fuzzy C-Means data clustering..................................................................................................22
    Fuzzy c-means data clustering algorithm..............................................................................................................23
    Fuzzy c-means algorithm's steps............................................................................................................................23
    Description and analysis of the parameters and equations....................................................................................24
  2.4 The Bayesian Information Criterion (BIC) formula...............................................................................................25
Section 3: Requirements analysis.............................................................................................................................27
  3.1 Research Questions................................................................................................................................................27
  3.2 Requirements from this project............................................................................................................................28
Section 4: Design........................................................................................................................................................29
  4.1 Clustering by Distance Algorithm........................................................................................................................29
    4.1.1 Activity Diagram of the Clustering by Distance algorithm............................................................................31
    4.1.2 Class diagram of the Clustering by Distance algorithm....................................................................................32
  4.2 Fuzzy C-Means Clustering Algorithm..................................................................................................................33
    4.2.1 Activity Diagram of the Fuzzy C-Means clustering algorithm........................................................................33
    4.2.2 Class diagram of the Fuzzy C-Means clustering algorithm............................................................................34
  4.3 REVEAL algorithm................................................................................................................................................35
    4.3.1 Activity Diagram of the REVEAL algorithm..................................................................................................35
    4.3.2 Class diagram of the REVEAL algorithm........................................................................................................36
Section 5: Implementation.........................................................................................................................................37
  5.1 Clustering by Distance Algorithm........................................................................................................................37
  5.2 Fuzzy C-Means Algorithm...................................................................................................................................37
    5.2.1 The core mechanism.........................................................................................................................................46
    5.2.2 The errors calculation part................................................................................................................................51
    5.2.3 Fuzzy C-Means clustering program for single numbers of clusters...............................................................55
    5.2.4 Typographical mistakes in the main resource of Fuzzy C-Means Clustering Algorithm...............................56
  5.3 REVEAL Algorithm...............................................................................................................................................56
    5.3.1 Comments on entropies and the simple 1552 genes pairs calculation...........................................................74
    5.3.2 Observations on logarithms and calculations between doubles in Java.......................................................75
    5.3.3 Java heap space issue – out of memory error.................................................................................................76
Section 6 : Experiments

6.1 Introduction ............................................................................................................ 77
6.2 Experiments and Results using Fuzzy C-Means Clustering .................................... 78
6.3 Clustering by Distance results ............................................................................... 94
6.4 Experiments and Results using the REVEAL algorithm ...................................... 95
Section 7 : Evaluation .................................................................................................. 97
7.1 Fuzzy C-Means and K-Means ++ clustering results .............................................. 97
7.2 Clustering by Distance algorithm's discussion ....................................................... 98
7.3 REVEAL's evaluation ............................................................................................. 98
Section 8 : Conclusion ................................................................................................ 99
  8.1 Future Suggestions .............................................................................................. 99
Section 9 : References ............................................................................................... 101
Section 10 : Appendices ............................................................................................ 103
  10.1 Screenshots from Clustering by Distance algorithm ......................................... 103
  10.2 Screenshots from Fuzzy C-Means Clustering algorithm .................................... 104
  10.3 Screenshots from REVEAL algorithm ................................................................ 107

Illustration Index

Figure 1. Diagram of the main common procedures during the reverse engineering of GRNs [18]. 12
Figure 2. State transition tables showing the input and output data of the Boolean methods[19] .... 15
Figure 3. Example of a fuzzy clustering algorithm's result [20] ....................................... 22
Figure 4. Segmented line showing the centroids of the Clustering by Distance algorithm .... 30
Figure 5. Activity diagram of the Clustering by Distance algorithm ........................... 31
Figure 6. Class diagram of the Clustering by Distance algorithm ............................... 32
Figure 7. Activity diagram of the Fuzzy C-means clustering algorithm ..................... 33
Figure 8. Class diagram of the Fuzzy C-Means clustering algorithm ........................... 34
Figure 9. Activity diagram of the REVEAL algorithm .............................................. 35
Figure 10. Class diagram of the REVEAL algorithm ............................................... 36
Figure 11. Properties file showing the parameters for the clustering program ............. 44
Figure 12. State transition table of REVEAL ............................................................... 56
Figure 13. Diagram showing the stored entropy pair calculations in our “triangular” two-
dimensional array[21] .................................................................................................. 75
Figure 14. Changing the heap size in NetBeans IDE .................................................... 76
Figure 15. Plot of Fuzzy C-Means. First Results on Animal 1 ...................................... 78
Figure 16. Plot of Fuzzy C-Means. Results for clusters 1-140 on Animal 1 and high threshold ........................................................................................................... 80
Figure 17. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 2 and high threshold ........................................................................................................... 81
Figure 18. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 3 and high threshold ........................................................................................................... 81
Figure 19. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 4 and high threshold ........................................................................................................... 82
Figure 20. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 5 and high threshold ........................................................................................................... 82
Figure 21. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 6 and high threshold ........................................................................................................... 83
Figure 22. Table of Fuzzy C-Means. Results for clusters 12-17 on Animals 1-6 and high threshold .......................................................... 84
Figure 23. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 1 with low threshold ........................................................................................................... 85
Figure 24. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 2 with low threshold ........................................................................................................... 86
Figure 25. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 3 with low threshold ........................................................................................................... 86
Figure 26. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 4 with low threshold ........................................................................................................... 87
Figure 27. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 5 with low threshold ........................................................................................................... 87

3
Figure 28. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 6 with low threshold......88
Figure 29. Table of Fuzzy C-Means. Results for clusters 12-17 on Animals 1-6 and low threshold.89
Figure 30. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 1 with normalised from biologists' data and high threshold.................................................................89
Figure 31. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 2 with normalised from biologists' data and high threshold.................................................................90
Figure 32. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 3 with normalised from biologists' data and high threshold.................................................................90
Figure 33. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 4 with normalised from biologists' data and high threshold.................................................................91
Figure 34. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 5 with normalised from biologists' data and high threshold.................................................................91
Figure 35. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 6 with normalised from biologists' data and high threshold.................................................................92
Figure 36. Table of Fuzzy C-Means. Results for clusters 12-17 on Animals 1-6 with normalised data from biologists' and high threshold.................................................................93
Section 1: Introduction

1.1 Statement of non-plagiarism

I, Ioannis Efstathiou

hereby declare that the composition of this report submitted for examination has been made by myself and the words are personally expressed. Any exceptions, including works taken from any other authors, are all stated in my text and are also included in my references list.

Signed:

Date:

1.2 Acknowledgements

Firstly I would like to thank my supervisor, Dr. Pierluigi Frisco, who inspired me to work in the intriguing field of Biology and motivated me to study the crucial sheep scab disease. Without his valuable guidance and experienced suggestions this report would lack content and aim. The Heriot-Watt University and its Library department, for providing me all the essential knowledge and the flexible means respectively to effectively seek resources and produce this work. My family, partner and friends for their support and understanding of my effort.

1.3 Abstract

The sheep scab disease caused by the ectoparasitic mite psoroptes ovis is considered to be of
critical importance in many countries (including the UK) having countless victims. With the implementation and use of the Fuzzy C-Means data clustering -as a first step in reverse engineering the ovine gene network- along with the REVEAL reverse engineering algorithm, we initially concluded on 17 clusters of similar gene expressions and then extracted the core anatomy of the network consisting of 500 genes respectively. These insightful results were produced from 6 different gene expression profiles which were based on 5 distinct time data points. Their further comparisons with previous biological studies reinforced their validity and set new promising directions for any future pharmaceutical support.

1.4 Introduction

One of the most dangerous and highly epidemic ectoparasitic diseases, affecting the ovine skin in the UK and resulting in economic losses, is the sheep scab incurred by the mite psoroptes ovis. This prompt cutaneous inflammation is characterized by damaged skin, yellowish scales, excessive skin irritation with pain and itching due to the rapid response of the immune system. This, still limited in knowledge, host-parasite contact has been investigated deeper. Thus, the Fuzzy C-Means clustering algorithm along with the REVEAL reverse engineering algorithm have been studied, implemented and applied on ovine gene expression data.

The above algorithms have never been used before on ovine data under the infestation of psoroptes ovis. Six expression profiles, originating from six different animals and based on 5 distinct time data points (initial pre-infestation time, 1 hour after the infestation, 3 hours, 6 hours and 24 hours) were obtained from Dr. Stuart Burgess of Moredun Research Institute. Mapping the gene network of the sheep and working thoroughly on these expression data profiles, the Fuzzy C-Means clustering algorithm resulted in specific numbers of clusters (17 being the most popular) of
similar values (mRNA levels) of gene expressions revealing interesting biological similarities between them.

The REVEAL algorithm managed to extract successfully 500 gene interactions between the ovine gene expressions, uncovering the core anatomy of the biological network. Furthermore, the above results have been compared with known gene expression groups from other biological studies and they provided us with promising explanations about the sheep scab mechanisms, verifying and bringing forth some more important biological effects that might be insightful for the making of novel, efficient vaccines.

1.5 The Aim

Alternatively efficient approaches, inspired by the field of computer science will be implemented for the sake of clarity. Thus, in this dissertation the Fuzzy C-Means clustering algorithm along with the REVEAL reverse engineering algorithm will be developed and used in a Java programming environment, in order to group the expressions of 1,552 genes over a segmented 24-hour time course and result in the revelation of the biological network's gene relationships and pathways respectively. The approximate clustering nature of the Fuzzy C-Means algorithm combined with the exhaustive uncovering behavior of the REVEAL, are expected to assist radically in successfully locating clusters where genes will perform similar expressions in specific time units. In addition, the detailed answers from the reverse engineering nature of REVEAL will validate them by outlining the anatomy of the core biological network. These results will be then further compared with previous insightful works. Thus, we will hopefully be able to provide better directions in producing suitable drug targets for the sheep scab disease.
1.6 The Objectives

We conclude on the below critical objectives:

1. Development of the Fuzzy C-Means clustering program

2. Development of the REVEAL program for comparisons including at least 2 input arguments

3. Efficiently run the REVEAL program in parallel if needed to take the advantage of time and computational resources

4. Comparison of the Fuzzy C-Means clustering results with those of the K-Means++ (implemented by Dr. Frisco) clustering to evaluate the performance of both

5. Comparison of the Fuzzy C-Means clustering results with those of previous studies in order to uncover significant biological effects

6. Comparison of the Fuzzy C-Means clustering results with those of REVEAL to validate the current results

7. Document the performance results of the two applied algorithms on ovine gene expression data

8. Document possible important biological results from the comparisons of the two algorithms with the results of the previous studies

9. Stay in contact with any newer similar works until the end of the project

10. Document any findings on future works
1.7 What Follows

In this report, along with the studies that have been already made on the sheep scab disease so far in the Research Report, including that of the DNA genetic expression and the concepts of the two popular clustering algorithms, K-Means++ and Fuzzy C-Means, the REVEAL reverse engineering algorithm will be also examined, implemented and applied on ovine gene expression data to uncover hidden important biological patterns. In addition, an attempt will be made to validate the results of Fuzzy C-Means clustering mechanism through their comparisons with those of the REVEAL algorithm. Finally, further comparisons with biological works that have been already made will take place and therefore results will be also directed back to the biologists. Any unfinished experiments, suggestions, recommendations and future works will be thoroughly reported at the end of this document.
Section 2 : Literature Review

2.1 Introduction and the importance of the problem

Based on the Research Report and according to the Central Dogma of Molecular Biology, each gene of the DNA is transcribed to messenger RNA (mRNA) and then follows the translation of the latter to a protein. This is known as the gene expression. Each RNA can be considered as a transportable storage mean containing the programming code which is required for the synthesis of a particular protein. The proteins are used to either promote (increase the rate, also known as activation) or repress (decrease the rate) the expression of the DNA sequence segments, the genes. The genotype (genes) defines and develops the phenotype (proteins) at a specific rate, according to the cell's needs. This mechanism is used by all of the known living creatures and it's significance is fundamental to sciences such as the genetics. In this dissertation it is considered to be the “key” of the current applied clustering and reverse engineering processes as we are interested in uncovering specific gene expression similarities and interactions during the psoroptes ovis infestation.

Data clustering is the procedure where similar objects comprising a data set are being grouped into the same classes, or clusters, allowing us to efficiently locate significant relations. Depending on the specific task's nature, some of them seem to have better results than others. For example, the fuzzy data clustering technique that will be used in the dissertation is the process that divides a data set of objects into clusters where the objects are as much similar as possible. It is inspired by fuzzy logic, where instead of considering that something can be either 0 or 1, up or down, it treats it as it belongs partially to both, having a specific degree of belonging for each. On the other hand, the k-means clustering method is strict and exact as we have seen in the Research Report. Each of the data objects belongs to a specific cluster with an absolute degree of belonging.
The sheep scab disease leads to significant economic losses in many countries, including the UK. The accelerated cutaneous damage that is caused once the mite establishes contact with the ovine skin results in excessive itching, unbearable pain and eventually death to its victims. Over a 24-hour period of time the skin's infestation with psoroptes ovis leads to a differential expression of 1,552 genes. Various attempts have been made in order to classify these genes by their peak expressions, clustering them temporarily during the sheep scab infestation and based on different specific time zones. Unique psoroptes ovis sequences have been synthesized, followed by ontology analysis predicting pathways involved in the host response and protein structure domains.

All the above gene expression sequences have been widely compared with others and have been analyzed with tools that locate potential translations in unknown nucleotide sequences. But most important, predictive conclusions have been reached about the gene network of the sheep scab extending any previous knowledge of the host response to infestation. And hierarchical classification so far, using Pearson's and Spearman's correlation methods with the assistance of micro arrays and scatter plots for quality assurance was the key of the ovine gene expression clustering as we examined in the Research Report.
2.2 Reverse Engineering the Gene Regulatory Networks (GRNs)

Over the last sixty years the complexity of the biological systems has been studied widely as the molecular biology has thoroughly provided many answers to their structures and inner mechanisms. Reverse engineering is the method that uncovers hidden parts of genetic networks\[6\] like the pieces from a puzzle, relationships and pathways between gene expressions, in an attempt to outline the core anatomy of the biological systems in living organisms. However, confusion in research has been observed as sometimes the term of reverse engineering is mixed with that of the simulation where the main focus is given to the representation of the genetic network that is typically required a priori as it is presented in the work of R. Khanin, V. Vinciotti and E. Wit[8].

The main purpose of simulation in computational biology is to symbolize features, traits and behaviors of biological systems through the use of computer applications that are programmed
Ioannis Efstathiou – MSc in AI

exclusively for this specific reason. Information about the structures of these networks with details of the nodes and their edges need to be defined first in order to achieve the appropriate simulation. On the other hand, reverse engineering has an inverse concept and can be considered to be the step before that of the simulation. Here we attempt to reveal the biological network's anatomy performing an inverted analysis that is based on our observations. In other words, we gradually extract interactions between the biological system's components by observing, sequentially processing biological data while synthesizing the predicted relationships and pathways.

There are different methods which infer significant gene interactions[6] in reverse engineering genetic regulatory networks (GRNs) that require different input data according to Figure 1. This is very important to keep in mind as these methods produce specific results only from particularly designed input which also resulted from distinguished biological experiments. The input can be either expression data (i.e. DNA microarray data) or biological information data (i.e. DNA sequence data) or both. As it is stated by Cho, K.-H. et al.[6]: “there is a limitation in inferring a GRN using only the expression data and hence, it has been proposed to make further use of diverse biological information.” In our case, sheep scab gene expression data will be used as input for our reverse engineering inference method and therefore further biological information will be used on our results to produce deeper gene predictions by directing them to the biologists.

According to the work of Cho, K.-H. et al.[6], the two main types of inference methods include:

**Inference methods based on expression data**

- Boolean methods
- Bayesian methods
- Regulation matrix methods
Inference methods based on biological information (as well as expression data)

- Module construction using gene expression and sequence motif methods (MODEM)
- Genetic regulatory modules methods (GRAM)

Furthermore, every different inference method provides specific inferred GRN output. In some cases this output includes only gene correlations, in others detailed information about the related regulations of the genes such as promotions and repressions, other times probabilities of the related regulations etc. Therefore, it is very important for the researchers to understand the exact outcome that can be produced from each of the above inference methods in order to select them wisely along with the correct input for potential specific experiment. In our case, we received expression data from the biologists (mRNA intensity values) and decided to proceed with Boolean methods.

The inference of genetic regulation networks (GRNs) with the use of Boolean methods is achieved with the application of Boolean logic (0 or 1) to the time states of the genes[6]. These binary values indicate either the genes promotion (increase in the mRNA level) with 1 or the repression (decrease in the mRNA level) with 0. The construction of the state transition pair tables follows, as we can see in Figure 2., by representing all of the gene expressions binary and then comparing all of the genes (nodes) of our genetic network with a number of input arguments sequentially (first with 1, then with 2, 3.. etc.) in an attempt to reveal their interactions.

However, according to the work of Cho, K.-H. et al.[6] all of the inference methods have the risks mainly of the computational complexity problem along with the dimensionality problem (the number of genes is very high compared to that of the time points which is very low) and the
experimental measurement problem (no specific instructions on how to accurately design the experiment in order to separate the direct and indirect relationships among the genes). We indeed met these problems, we confronted them and they will be explained later. Thus, we focused our research on locating the most efficient reverse engineering boolean method (algorithm) for our sheep scab problem while having mainly in mind our limited time frame. Therefore the computational complexity factor played the protagonist role from now on.

Input data

<table>
<thead>
<tr>
<th>Input state</th>
<th>Output state</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>$x_2$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>State transition pair 1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>State transition pair 2</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>State transition pair 3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>State transition pair 4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>State transition pair 5</td>
<td></td>
</tr>
</tbody>
</table>

Output data

<table>
<thead>
<tr>
<th>Argument $x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument $x_2$</th>
<th>$x_1$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument $x_3$</th>
<th>$x_1$</th>
<th>$x_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument $x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument $x_2$</th>
<th>$x_1$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Argument $x_3$</th>
<th>$x_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2. State transition tables showing the input and output data of the Boolean methods[19].

**a** State transition pair table showing 5 pairs of the input and the output states. The networks consists of 3 nodes (arguments) which have 2 states (0 or 1)

**b** 3 state transition pair tables of the outputs $x_1$, $x_2$, and $x_3$ compared with the input arguments $x_2$, $x_1$ and $x_3$, $x_1$ and $x_2$ and $x_3$ respectively as it is performed in the REVEAL algorithm.
2.2.1 Complexity of Boolean Algorithms

i) REVEAL:

- S. Liang, S. Fuhrman, and R. Somogyi[10] implemented the REVEAL algorithm in 1998 for only 50 elements having 3 inputs each where they suggested that the problem was "tractable withing the conditions tested so far". Incomplete transition state tables were analyzed (100 pairs out of 1015 possible) and according to their sayings provided "the original rule and wiring sets".

- On the other hand, according to the work of S.B. Wilson et al.[15] this algorithm was applied to small training sets (100-300 seizures and therefore the complexity was reduced) of a large overall data set containing 670 seizures and performed well. Their approach included a NN algorithm that was applied to different parts of their training sets. However it didn't examine the whole data.

- T. Akutsu, S. Miyano and S. Kuhara[1] stated that "We proved that $O(\log n)$ expression patterns are necessary and sufficient to identify the underlying Boolean network of n genes correctly with high probability if the maximum indegree is bounded. Since the Boolean network is not realistic, other models have been proposed."

So lets have a look at the other models.

- According to K.-H. Cho et al.[6] REVEAL's main disadvantage is the fact that it performs exhaustive search in order to find all of the required mutual information of the pairs of genes by increasing its arguments gradually. Thus, in our data set which contains 1,552 genes this will require a lot of time and computational power and especially in the cases where the
input arguments are more than two. One way to handle this issue is mentioned in the Discrete Function Learning algorithm below though where the search space is reduced.

ii) BOOL-2:

- T. Akutsu, S. Miyano and S. Kuhara[1] suggested that their boolean network (based on logical operations and not on the mutual information like the above) included 160 nodes where 140 expression patterns were required to provide them with correct identification (where \( k=2 \), \( k \) is the number of input nodes of each node). A Sun Ultra Enterprise 10000 with 64 CPUs was used for their experiments. High computational power was required and used for a small dataset (considering ours) mainly because of the logical operations that raised the complexity.

iii) Temporal Boolean:

- According to A. Silvescu and V. Honavar[13], a Boolean function is allowed to control the expression of at most \( k \) genes not only at times \( t \) (as before) predicting the expression of any gene at \( t+1 \) but also \( t-1 \) and \( t-(T-1) \) where the \( T \) is the given time point. This increases the complexity of our problem again but it is useful in small subsets of search spaces where detail is needed.

iv) Discrete Function Learning:

- Z. Yun and K.C. Keong[17] have stated that the exhaustive search disadvantage of REVEAL is handled by reducing the search space into the set \( S < \{x_j\} \) where \( S \) includes \( k \) nodes
Ioannis Efstathiou – MSc in AI

which have the highest mutual information with \( x_i \) when we change the arguments' number
of \( x_i \) from \( k \) to \( k + 1 \). It uses Karnaugh maps for noise detection and removal and it was
applied to 11 nodes (yeast cell cycle regulators). Low number of nodes in combination with
the application of the Karnaugh maps indicate its high complexity.

v) Computational Algebra:

Stigler[9]: "they have shown that Boolean functions can be represented by polynomial
functions with the coefficients of 0 or 1 through translation of Boolean logical operators into
algebraic operators." Our noisy data set though will not be suitable for this algorithm as
according to the same paper "the algorithm is sensitive to noise because of the procedure of
fitting input states to the output states". Their model[9] included overall 60 nodes
(Drosophila melanogaster) and they generated 25 networks using 20 of the above nodes, as
resolution in data played the most important role here. According to their conclusion: "The
advantages of our method are that we can explore the whole model space for a data set,
without using enumeration techniques. Furthermore, we have an algorithmic selection
criterion to choose a model. And, finally, our models are multi-state, which allows the choice
of a resolution to fit the characteristics of the data set". So no exhaustive search but the
tested data set is way too small.

vi) Probabilistic Boolean Network:

- As it is stated by K.-H. Cho et al.[6]: "There are probabilistic extensions of Boolean
methods by considering many Boolean functions \( f_1, f_2, \ldots, f_k \) of each node \( x_i \) and the
probabilities with which each Boolean function $f_{ij}$ is chosen to predict the state of $x_i$ [37, 46, 47]. The PBN algorithm [46, 47] can account for the embedded uncertainty of data and models by allowing some error bounds in the Boolean functions. There are, however, too many parameters to be estimated (e.g. 228 of 1077 parameters only for eight nodes). Ching et al. [37] have proposed an extended PBN algorithm that reduces the number of parameters to be estimated by making use of a homogeneous first-order discrete-time Markov chain and regression while keeping the advantages of the PBN algorithms [46, 47]. This algorithm, however, does not guarantee improved accuracy of the inference result.

- According to W.-K. Ching et al.[5], the number of parameters of their proposed model is $O(n^2)$ where $n$ is the number of the genes. Their dataset included only 16 nodes of practical yeast data sequences. Thus the complexity would be very high for our dataset and the concept not promising for our limited time because of the large number of parameters that have to be taken into consideration.

vii) ARACNe:

- According to Katia Basso et al.[2], there is strong evidence that the gene expressions of high-order eucaryotic cells can be represented by scale-free networks (as might be in our mammalian cellular case of the sheep scab disease). "The results suggest a hierarchical, scale-free network, where a few highly interconnected genes (hubs) account for most of the interactions." ARACNe algorithm in their case inferred a network with approximately 129,000 interactions. The inferred sub-network structure included 2,063 genes of the protooncogene MYC gene (hub, transcription factor). Examining their formulas, mutual information from entropy calculations will lead us to probabilities calculation which in turn
lead to calculations of the smoothing parameters \( d \) (1 for each gene) which are acquired from monte carlo simulations (risk analysis algorithms, typically require decades thousands of re-calculation for their predictions in problems such as ours). This method seems to be the best for our sheep scab case but these complex calculations will probably take us far away from the limit of our time scale.

2.2.2 Potential conclusions and the final decision:

Considering all of the above information regarding the algorithms complexities, we reached to 3 potential conclusions for the sheep scab case:

i) The first would be to gradually extract a number of important sub-sets from the main data set (of 1552 genes) based on our time-scale and results. Then we would apply the REVEAL algorithm (using normally the mutual information and creating state transition pair tables) to each one of them, possibly including NN classifiers to train on and deal with their noise as we saw in the work of S.B. Wilson et al.[15].

ii) The second would be to apply a discrete function learning algorithm to our data set, taking the advantage of a smaller set (genes with the highest mutual information) avoiding the massive exhaustive search of the normal REVEAL. Also the noise will be handled by Karnaugh maps as Z. Yun and K.C. Keong[17] suggested.

iii) The third option, which seemed to be the most effective, promising and interesting would be to apply the ARACNe algorithm to our dataset. This algorithm has been applied to a dataset (MYC) as large as ours and suggested that the gene expressions of the high-order eucaryotic
Ioannis Efstathiou – MSc in AI

cells are represented by a scale-free network. Furthermore, by locating direct, related regulations between the genes and by removing non-direct relations it seemed that it produced empirically correct results -fast-. It would be interesting to produce results that either suggest a scale-free topology of the ovine gene network or not. The only problem with the ARACNe is that the formulas and calculations seem that will be very complex for the 1552 genes (as we have mentioned previously in the (vii)) so perhaps we should consider experimenting first with a subset of our dataset.

After a very constructive discussion with Dr. Frisco and based on his expertise, we took the decision to proceed with the option (i). Having in mind mainly parameters such as the algorithm's complexity, the computational power and the project's remaining time, the REVEAL algorithm claimed and successfully secured the protagonist role in our future reverse engineering experiments. The main plan was to work with the whole dataset of 1552 genes and with the potential assistance of parallel programming to attempt recovering the core anatomy of the entire gene regulatory network by inferring relationships of at least two input arguments with the rest of the output ones.
2.3 Fuzzy logic and Fuzzy C-Means data clustering

Restating the Section 2.3.3 of the Research Report, as its repetition is significant in this report, Fuzzy clustering is inspired by the approximate, uncertain nature of Fuzzy logic. The sense of this concept is based on relativity rather than strictness and preciseness. Thus, instead of considering that something can be either 0 or 1, such as good or bad, it considers it as it belongs partially to both, having a specific degree of belonging for each. According to Paul P. Wang et al. [14]: “Fuzzy relations have always been the main concept and hence they play an instrumental role in the world of mathematics of uncertainty because they soften the rigid requirement and very tight constraints in traditional mathematics.” According to this book, Fuzzy logic is usually applied to mathematical modeling, natural language, data mining and knowledge engineering, bio-informatics, Fuzzy logic control, pattern recognition, decision making, geology and management and operations research.
Fuzzy c-means data clustering algorithm

In the same reasoning, Fuzzy data clustering is the process that divides a data set of objects into clusters where the objects are as much similar as possible. Thus, these objects are allowed to belong to at least two different clusters, having a sequence of distinct weights of belonging. Each of these weights refers to each cluster. The algorithm that will be used is the Fuzzy C-Means (implemented by Dunn[7] and optimised by Bezdek[3]) as according to Miyamoto et al.[11]: “A main reason why we concentrate on Fuzzy c-means is that most methodology and application studies in Fuzzy clustering use Fuzzy C-Means, and Fuzzy C-Means should be considered to be a major technique of clustering in general, regardless whether one is interested in Fuzzy methods or not”. This method (which in fact is a Fuzzy version of the normal K-Means) along with the K-Means++ (implemented and applied by Dr. Frisco) are going to be used in this dissertation in order to effectively cluster the gene expressions caused from the sheep scab by exploiting both of the benefits of these different approaches.

Fuzzy c-means algorithm’s steps

According to Miyamoto et al.[11], who are based on Dunn[7] and Bezdek[3], the Fuzzy C-Means algorithm can be separated to the following steps:

Step (i): Initialize a random Fuzzy partition by assigning random values (weights) in it

Step (ii): Use the Fuzzy partition and calculate each centroid of the clusters

Step (iii): Renew the weights of the Fuzzy partition

Step (iv): If there is no change in the centroids then continue to step (v) -otherwise go to step (ii)

Step (v): End
Based on Miyamoto et al.[11], we consider:

- $n$ number of objects $X$
- $k$ number of clusters $C$
- A matrix of the partition $W_{ij}$ (where $i$ is the object number and $j$ is the cluster number) and each element $(wij)$ of $W$ represents the $i$-object's ($X_i$) weight of belonging to the $j$-cluster ($C_j$)
- The sum of the weights of each object equals to 1 (it belongs to every cluster “partially”)
- Each cluster includes at least one object with weight between 0 and 1 (it cannot be either of these two though, $0 < wij < 1$)
- The goal is to minimize the following target function:

\[
\sum_{j=1}^{k} \sum_{i=1}^{n} w_{ij}^{p} \text{dist}(x_i, c_j)^2
\]

- The parameter $p$ is defined as the influence of the weights and can be any real number which is greater than 1 ($p > 1$)
- The equation which defines the centroid is:

\[
c_j = \frac{\sum_{i=1}^{n} w_{ij}^{p} x_i}{\sum_{i=1}^{n} w_{ij}^{p}}
\]
– The above equation is similar to the one that is used for the normal k-means algorithm. The differences here though are that we consider: (i) all of the data objects and (ii) their weights as well.

– When we renew the weights of the Fuzzy partition, the weights equation is defined as:

\[
W_{ij} = \frac{(1/\text{dist}(x_i, c_j)^2)^{\frac{1}{p-1}}}{\sum_{q=1}^{k}(1/\text{dist}(x_i, c_q)^2)^{\frac{1}{p-1}}}
\]

– The above equation is obtained by the hypothesis that the sum of the weights equals to 1, as we saw previously. The weight increases as the distance between the centre of the cluster and the object decreases.

### 2.4 The Bayesian Information Criterion (BIC) formula

In order to measure the errors of the Fuzzy C-Means clustering results and select the best numbers of clusters, we used the BIC formula which has been provided by Professor Gavin Gibson from the Department of Actuarial Mathematics and Statistics of MACS in the Heriot Watt University. The formula provides solutions to the overfitting situation of models that occurs when more than enough parameters are provided in order to increase the likelihood. In fact, its error results are based on the number of parameters involved in the model. So, according to the derived formula of H.S. Bhat and N. Kumar[4] and following our clustering needs, the form that was used is:

\[
\text{BIC} = -2 \times (\text{first parameter} - (0.5 \times \text{second parameter})) + 2 \times k \times \log n
\]
On the above equation:

- k is the number of non-empty clusters

- n is the number of the elements

- first parameter is defined as the summation \( \sum (-n(i) \times \log(\sigma(i))) \), where the n(i) indicates the number of elements in cluster i and \( \sigma(i) \) indicates the variance in cluster i.

- second parameter is defined as the summation \( \sum \left( \text{edc}(i) / \left( \text{vc}(i) \right)^2 \right) \), where the edc(i) is the euclidean distance of cluster i and \( \text{vc}(i) \) is the variance in cluster i.

- the variance is defined as the square root of the summation \( \sum \left( (x(i) - \text{mean})^2 / n(i) \right) \), where x(i) is the element i of cluster i, n(i) is the number of elements in cluster i and mean is the cluster's centroid.

The errors that we produced during clustering were all negative numbers and the most negative values were considered to be the best ones.
Section 3: Requirements analysis

3.1 Research Questions

This work will focus on clustering and reverse engineering sheep gene (1,552 genes) expression data over a 24-hour segmented time course, developing and using two very important -in the domain of data clustering and reverse engineering biological systems respectively- algorithms, in an attempt to provide answers to the following questions:

- Would the choices of the Fuzzy C-Means clustering and that of the REVEAL reverse engineering algorithm respectively be wise in handling the ovine gene expression while providing reasonable and meaningful results (patterns) to the domain of Biology?

- Would the results of these two algorithms agree between them and provide together an integrated package of what can Computer Science offer to the problem? Is there perhaps another useful mediate algorithm missing?

- What exactly should we consider in clustering and reverse engineering the ovine gene expression? Should we only be based on the points of time when the gene expression levels presented a peak? Which and how many time points should we use when comparing the input arguments with each of the output ones in the state transition table?

- Are both of the algorithms' computational complexities allowing us to produce thorough results within our remaining time frame? Should we also take the advantage of parallel computation to avoid running out of time? Do the complexities of both of the algorithms require parallel processing and, if yes, at which stage of their development?

- Would we achieve interesting results, by uncovering crucial relations between the gene
expressions, compared to those of the works that have been already made so far?

– If not, would the above results be meaningless or perhaps similar to those of some of the works that have been implemented already, verifying their accuracy?

– Would it be possible to direct the scientists in producing effective vaccines, based on the results of the above data clustering techniques, which would be able to eradicate the sheep scab disease?

3.2 Requirements from this project

There are two required, parallel potential paths in this study. The first is expected to end up in a solid verification of the correctness of some of the previously tested clustering procedures and other reverse engineering methods and findings of related reviewed studies. As an extension to this case, the second one will hopefully end up in new discoveries to be made, always based on the insightful results of the previous attempts, providing even more effective ways of creating vaccines to control and hopefully eradicate the disease caused by this indomitable ectoparasitic mite.
Section 4: Design

4.1 Clustering by Distance Algorithm

The reason that this program was designed and implemented was to experiment with a simple clustering idea and study its outcome on the ovine gene expression data as an introductory step to the future implementation of the Fuzzy C-Means clustering algorithm. The Clustering by Distance algorithm initially finds the minimum and maximum values of our data points (in our case genes) and therefore defines the space where the clustering will take place as their difference (maximum−minimum). The user then selects the number of clusters (centroids) and the program defines \([\text{number of clusters}+1]\) spaces of equal length called “c-units” according to the formula: 
\[
\text{c-unit} = \frac{\text{max}-\text{min}}{\text{c}+1}
\]
where max is the maximum value of our data set, min is the minimum and c is the number of clusters.

After that, the calculation of the centroids' values from the above “c-units” occurs (these centroids are defined also as centres, clusters or cluster points for the sake of ease despite the fact that normally the centroid and the cluster have different meanings). The program then calculates all of the distances of the data points from the centres and their degree of belonging to each of the clusters (centres) according to the formula 
\[
x=100-((\text{distance}*100)/(\text{max-min}))
\] as percentages. This is inspired by the degree of belonging that Fuzzy C-Means clustering is using in its equations for the calculation of the weights.

However, the reason that these percentages have been included here was to have statistical detailed information for each of the data points. They show exactly how close or distant the points are from all of the centroids in order to reveal any hidden from the 'naked eye' distance relationships. The program then locates and produces results about all of the clusters with their
relevant -closest in distance- data points based on this fast and simple, clustering by distance reasoning.

Figure 4. Segmented line showing the centroids of the Clustering by Distance algorithm.

On the above example the vertical lines are supposed to be the centroids of 11 clusters. The vertical lines of each side of the line (search space), left and right, are the minimum and maximum values respectively of all of the elements. The equal spaces in between are the c-units.
4.1.1 Activity Diagram of the Clustering by Distance algorithm

Figure 5. Activity diagram of the Clustering by Distance algorithm.
4.1.2 Class diagram of the Clustering by Distance algorithm

Figure 6. Class diagram of the Clustering by Distance algorithm.
4.2 Fuzzy C-Means Clustering Algorithm

4.2.1 Activity Diagram of the Fuzzy C-Means clustering algorithm

![Activity diagram of the Fuzzy C-means clustering algorithm.](image)

Figure 7. Activity diagram of the Fuzzy C-means clustering algorithm.
4.2.2 Class diagram of the Fuzzy C-Means clustering algorithm

![Class Diagram](image)

**Figure 8. Class diagram of the Fuzzy C-Means clustering algorithm**
4.3 REVEAL algorithm

4.3.1 Activity Diagram of the REVEAL algorithm

Figure 9. Activity diagram of the REVEAL algorithm.
4.3.2 Class diagram of the REVEAL algorithm

Figure 10. Class diagram of the REVEAL algorithm.
Section 5: Implementation

5.1 Clustering by Distance Algorithm

According to the activity and class diagrams of the section 4 as well as on the code, when the program begins in the main method of the class FuzzyMainie it creates an object of the class FileManipie named “fman”. With the call of its method “readSF”, having parameters the name of the file where the data to be clustered reside in and a double array “genes” which will hold the 1552 different gene expressions values, it checks if the file exists or not (informing the user accordingly). In the case where it does then it reads each line of the defined file and assigns them in the “genes” array. After that, it creates an object of the class MINMAXie named “minmax”:

```java
public static void main (String[] args)
{

    FileManipie fman = new FileManipie();   //file's calls
    fman.readSF("fuzzinput6.txt", genes);    //read file's gene expressions
    MinMaxie minmax = new MinMaxie();

    The program then initializes the double two-dimensional array “clustersArr” through a for loop and prints on the screen all of the contents of the “genes” array. After that it calculates the minimum and maximum values of the “genes” array through the methods “getMin” and “getMax” respectively and prints them on the screen too. It then asks the user to provide the number of clusters through an JOptionPane window, prints them on the screen and assigns the values of the minimum and maximum values of the “genes” array to the double variables “minVal” and
“maxVal” respectively. Then the calculation of the double “cunit” occurs and prints on the screen the equal differences in the distances of the centroids/clusters (definition of the “cunit”):

```java
for (int a=0;a<1552;a++)
{
    for (int b=0;b<500;b++)
    {
        clustersArr[b][a]=1000000;  //initialization of the clustersArr array with 1000000
    }
}

for (int i=0;i<1552;i++)
{
    System.out.println(genes[i] + "\n");
}

System.out.println(minmax.getMin(genes) + " minimum value\n");  //get the minimum double value
System.out.println(minmax.getMax(genes) + " maximum value\n");  //get the maximum double value

clusters = JOptionPane.showInputDialog(null, "How many clusters are we going to use?");
clustersi=Double.parseDouble(clusters); //from string to double for the value of clusters

System.out.println(clustersi + ":number of clusters\n");

minval=minmax.getMin(genes);
maxval=minmax.getMax(genes);

cunit=(maxval - minval)/(clustersi+1);
System.out.println(cunit + ":the difference of clusters\n");
```
After that the program assigns in the double array “cpoints” the calculations of the centroids values according to the minimum value “minVal” and the “cunit” variables through a for loop and prints them on the screen. These “cpoints” are the values of the different centroids and they are stored in each cell of the array “cpoints”, the 1st centroid (cluster) in its 0 cell, the 2nd in its 1st etc.

cpoints[0]=minval+cunit; //1st cluster point

System.out.println(cpoints[0] +" is the cluster\n");

for(int j=1;j<(int)clusters;j++)
{

cpoints[j]=cpoints[j-1]+cunit; //assignment of cluster points in each subsequent cell of the array

System.out.println(cpoints[j] +" is the cluster\n");
}

The program then calculates the distances of all the elements of the “genes” array from the centroids and their degree of belonging to each of these clusters through a for loop according to the formula $x=100-((\text{distance}*100)/(\text{max}-\text{min})$ as a percentage. These distances are kept in the double two-dimensional array “distances”. Its rows indicate the centroids (clusters) and its columns the genes (clustering data points if you prefer) so their assignments are being achieved based on this reasoning as we can see from the code below. For each one of the data points (elements of the “genes” array), it prints on the screen all of these percentages to each of the centroids (clusters):

for (int l=0;l<1552;l++)
{

System.out.println("\nGene " +(l+1) + ": ");
}
for (int m=0; m<(int)clusterssi; m++)
{
    distances[m][l]=Math.abs(genes[l]-cpoints[m]);
    System.out.println(distances[m][l] + " distance " +(100-((distances[m][l]*100)/(maxval-minval))) + "% degree of belonging in cluster " +(m+1) +" ");
}

After that, the program finds the clusters that have the nearest data points through for loops again, by locating smallest distances to the appropriate clusters in the “distances” array and assigns the genes which had the smallest distance from their centroids to the “clustersArr” array. This array that had only been initialised so far as we have seen is similar in the reasoning as the “distances”. Its rows indicate the centroids (clusters) and its columns the data points.

The difference though is that this array now holds the values of the data points (from the “genes” array) which had the smallest distance to a specific cluster (centroid) and therefore they are assigned to a distinct row and column, indicating the cluster's number and gene respectively. Finally the program prints on the screen all of the clusters with their relevant data points by checking in the “clustersArr” array and excluding the cells which are still initialized (they remain unchanged) through the use of for loops.

//finding of the smallest distance and assignment of the data point to the appropriate cluster
for(int h=0; h<1552; h++)
{
}
minloc=distances[0][h]; //assignment of the 1st row of the distances to the minimum
clust=0; //record of the cluster and the number of the gene which has the smallest distance
gen=h;
for (int g=1;g<(int)clustersi;g++)
{
    if (distances[g][h]<minloc)
    {
        clust=g;
gen=h;
        minloc=distances[g][h];
    }
}
clustersArr[clust][gen]=genes[gen]; //assignment of the gene which had the smallest distance to the g (gen) number of cluster
}

for (int a=0;a<(int)clustersi;a++) //clusters
{

}
5.2 Fuzzy C-Means Algorithm

The code of Fuzzy C-Means clustering algorithm was implemented in Java (NetBeans IDE environment) as an additional feature of Dr. Frisco's initial clustering program, which was including at that time the K-Means and K-Means ++ mechanisms. Thus, only the part of the Fuzzy C-Means clustering will be thoroughly explained here (showing some parts of the code as well) along with some essential parts programmed exclusively by Dr. Frisco that need to be included and will be generally mentioned for the sake of clarity.
Based on the activity and class diagrams of the section 4 as well as on the code, when the clustering program begins in the main method of the class Clustering it prints the project's path and the computer's name. It then defines and loads a specified property file, where all of the user options for the clustering program reside in (such as the number of clusters, the fields to be checked, special parameters of the clustering formulas etc. as we can see in Figure 11). After that, it calculates and defines the reference and target arrays according to the columns of the specified (in the properties file) file that contains the gene expressions data set. A typical file like this includes columns with the names, types and mRNA values of the genes for each of the 5 distinct time data points. It then checks if this file has not been defined by the user and informs him/her about that.

If in the property file the user has declared that he wants to work on 2 columns of the input file and the target column's number is greater than the reference column's number, or he wants to work only with 1 and then their numbers are equal, then the program creates a new array list named “vd” of type VectorData which will contain Data (from the same named class) objects to be clustered. It then reads the Data objects from the specified file through the specified reference and target column (in the properties file) and -after checking if they are normalised declared in the property file or percentage- it changes them respectively and assigns them in the “vd” array list giving them all cluster number -1.
Figure 11. Properties file showing the parameters for the clustering program.

After that, the program checks the properties file if the clustering field is defined as true. If this is the case, then if the type of the clustering field is Fuzzy C-Means the program creates a long array named “bestSeedAr” which will hold the best seeds of each number of clusters (long[] bestSeedAr = new long[numClusters.length];). In addition, it creates another double array named “smallestErrorAr” which will hold the smallest BIC error (please refer to section 2.4 for details) for each number of clusters (double[] smallestErrorAr = new double[numClusters.length];). A new array list of type VectorData named “bestvd” which will contain the best clusters is then created (VectorData bestvd = new VectorData();).

Then the program, for the number of iterations which is defined in the properties file (for (int x=0;x<ManagesIO.getIntProperty(props, ConstantsClust.ITERATIONS);x++)), it creates another array list of type VectorData named “fvd” which will hold all of the Fuzzy data points (VectorData fvd = new VectorData();) and copies all of the data that “vd” contains into the new
The seed is stored as the current time in milliseconds plus the counter of the main loop for the above iterations. If the number of clusters exceeds the number of our data points that our Fuzzy list “fvd” contains, then the user is informed appropriately and the program ends:

```
if (fvd.size() < ManagesIO.getIntProperty(props, ConstantsClust.NUM_CLUSTERS)) {
    System.err.println("Fatal error: the number of clusters (" + ManagesIO.getIntProperty(props, ConstantsClust.NUM_CLUSTERS) + ") is bigger than the number of data points (" + vd.size() + ")");
    System.exit(0);
}
```

The program then calls the method named “initialiseClusters” which in turn calls the method “initFClusters” in the case where the type of clustering in the properties file is defined as Fuzzy C-Means. The latter method then calls another one, named “initialiseFCmeans” which assigns positive integer values to the clusters numbers of a number of random data points. This number is equal to the clusters and these data points are now considered to be our means (their clusters numbers are not -1 any more). After that, a means array is created named “mean” of type DoubleArray which will hold these means and another one of the same type named “oldmean” which will hold the
means of the previous turn every time in order to be compared with those of the “mean” as we will see later:

```
DoubleArray mean = new DoubleArray(ManagesIO.getIntProperty(props,
    ConstantsClust.NUM_CLUSTERS));

DoubleArray oldmean = new DoubleArray(ManagesIO.getIntProperty(props,
    ConstantsClust.NUM_CLUSTERS));
```

A method named “calcFuzzyInitCentres” is then called and searches the “fvd” list for data points that don't have clusters numbers -1 (in other words for means or centroids) and assigns them into the “mean” array. The program then uses two more methods named “findMinList” and “findMaxList” in order to locate and keep the minimum and maxim values of the Fuzzy list's “fvd” data points respectively:

```
double fmin=fvd.findMinList();

double fmax=fvd.findMaxList();
```

### 5.2.1 The core mechanism

After that, a while loop is used until the difference in the data points values between the “mean” and the “oldmean” arrays are less than the value of threshold epsilon that is defined by the user in the properties file (while(!mean.lesserThreshold(oldmean, ManagesIO.getDb1Property(props,ConstantsClust.FUZZY_THRESHOLD_EPSILON));). In more detail, a boolean method here called “lesserThreshold” checks whether or not the absolute difference between each of the elements of the current centroids' array “mean” are less or equal than those of the “oldmean” which belong to the previous time step. If all are less than the value of the threshold then the while loop
Inside the above while loop the first action that happens is the copying of the elements of the "means" array into the "oldmeans" array (oldmean.copy(mean);). This happens through the use of a method named "copy" which simply copies all of the elements of one array into the other. Before doing that though, it initially checks if the two arrays to be compared have equal sizes through the use of another method named "compareSizes". If not, then the user is informed about that and the program stops. After the copying procedure, the main Fuzzy C-Means clustering mechanism that we examined in Figure 7. of the section 4.2.1 takes place.

This mechanism first checks if the current centroids of the "mean" array are out of the range between the previously calculated minimum and maximum values (if (mean.getElement(k)>fmax || mean.getElement(k)<fmin)). If this is true then the weights for the current centroid of all of the Data elements of the Fuzzy list “fvd” are assigned the value 0 which has been already given to the weight variable “mWeight” (fvd.getData(j).setWeight(k, mWeight);). Here we should make clear that each of the Data objects that the Fuzzy list “fvd” holds includes a double variable named “value” which holds the value of the data point, an integer variable named “cluster” which holds the number of the cluster that it belongs to and an array named “weight” of type DoubleArray which has the length of the number of the clusters and holds into each of its cells the weights of the specific data point for the corresponding cluster.

```java
public class Data {
    private double value;
    private int cluster;
    private DoubleArray weight; ..........

    If the current centroids of the “mean” array are in the range between the previously
calculated minimum and maximum values, then if the checked data point is in distance from the centroid (in other words there is a difference in their values) the weight is calculated as we have seen in the equation 4.62 of the work of Xu, R. and Wunsch II, D.C.[16] (that can be also seen in Figure 7.) as well as in Section 2.3.

\[
W_{ij} = \frac{(1/\text{dist}(x_i, c_j)^2)^{\frac{1}{p-1}}}{\sum_{q=1}^{k}(1/\text{dist}(x_i, c_q)^2)^{\frac{1}{p-1}}}
\]

If the checked data point is not in distance from the centroid (their values are the same), then the weight equals to 1.

```java
for (int j=0;j<fvd.size();j++) {
    for (k=0;k<ManagesIO.getIntProperty(props, ConstantsClust.NUM_CLUSTERS);k++) {
        double distance =0;
        double mWeight =0;
        if (mean.getElement(k)>fmax || mean.getElement(k)<fmin) { //if the new centers are out of the range of our data objects
            fvd.getData(j).setWeight(k, mWeight);
        } else {
            for (l=0;l<ManagesIO.getIntProperty(props, ConstantsClust.NUM_CLUSTERS);l++){
                if(fvd.calcFuzzyDistances(j,k,mean)!=0) //if the data object is in distance from the center
                ...
            }
        }
    }
}
```
Ioannis Efstathiou – MSc in AI

distance +=

Math.pow((fvd.calcFuzzyDistances(j,k,mean)/fvd.calcFuzzyDistances(j,l,mean)),(2/(ManagesIO.getIntProperty(props,ConstantsClust.FUZZY_M_VALUE)-1)));

else

    distance=1;//ManagesIO.getIntProperty(props,ConstantsClust.NUM_CLUSTERS);

}    

mWeight=1/distance; //this is the membership's weight

fvd.getData(j).setWeight(k, mWeight);

}

}

} //for end, data points weight matrices

After the above calculations the program calculates the new centroids and assigns them into the “mean” array using the formula 4.63 (that can be also seen in Figure 7.) of the work of Xu, R. and Wunsch II, D.C.[16] as well as in Section 2.3

\[ C_j = \frac{\sum_{i=1}^{n} w_{ij}^p x_i}{\sum_{i=1}^{n} w_{ij}^p} \]

double centreWeight1;

double centreWeight2;

    for (int i=0;i<ManagesIO.getIntProperty(props,ConstantsClust.NUM_CLUSTERS);i+ 49
centreWeight1 = 0;

centreWeight2 = 0;

for(int f = 0; f < fvd.size(); f++) {
    centreWeight1 += (fvd.getData(f).getValue() * (Math.pow(fvd.getData(f).getWeight(i),
        ManagesIO.getIntProperty(props, ConstantsClust.FUZZY_M_VALUE))));

    centreWeight2 += Math.pow(fvd.getData(f).getWeight(i),
        ManagesIO.getIntProperty(props, ConstantsClust.FUZZY_M_VALUE));
}

mean.setElement(i, centreWeight1 / centreWeight2);

} //for end, centres (means) weight matrix

} while(!mean.lesserThreshold(oldmean,
    ManagesIO.getDb1Property(props, ConstantsClust.FUZZY_THRESHOLD_EPSILON));
//means(t+1)-means(t) >= epsilon(threshold)

The “mean” array now contains all of the new centroids and as we have already examined their differences from those of the “oldmean” are being compared to the threshold's value at the end of every iteration of the while loop in order for the clustering mechanism to eventually stop.

The program then searches through the use of for loops for the maximum weight in the DoubleArray “weight” variable of each data point. It records the number of the “weight” array's cell
Ioannis Efstathiou – MSc in AI

where the maximum value was found and then it assigns this number (in fact this number plus 1 to convert it to the actual cluster's value because the weight of the cluster 1 resides in the array's cell 0 etc.) to the data point's “cluster” variable:

```java
for (int a=0;a<fvd.size();a++) {
    double max=Double.MIN_VALUE;
    int location=0;
    for (int b=0;b<ManagesIO.getIntProperty(props,ConstantsClust.NUM_CLUSTERS);b++)
    {
        if(fvd.getData(a).getWeight(b)>=max) {
            max=fvd.getData(a).getWeight(b);
            location=b;
        }
    }
    fvd.getData(a).setCluster(location+1); //sets the cluster number from the weights matrix where the weight was the maximum
}
```

5.2.2 The errors calculation part

The program then calculates the BIC error that we examined in the Section 2.4 through the call of the method named “fuzzyerrorBIC”. This method accepts as a parameter the “mean” array which holds all of the Fuzzy C-Means clustering centroids and calculates the error through the use of the BIC formula that is also mentioned in the Section 2.4. After the calculation, the program
assigns this error to a double variable named “error” \((\text{double error} = \text{fvd.fuzzyerrorBIC(props, mean);})\). It then checks if this error was the smallest so far and if it was, it stores its value along with the best seed for future reference. Also it copies every element of the current Fuzzy list “fvd” to those of the list that holds the best clusters and it is named “bestvd” through the use of the method “addSFData”.

\[
\text{if (error<smallestError)}\{
    \text{bestSeed}=\text{seed};
    \text{smallestError}=\text{error};
    \text{for(int i=0;i<fvd.size();i++)} \{
        \text{bestvd.addSFData(i,fvd.getData(i).getValue(), fvd.getData(i).getCluster(), ManagesIO.getIntProperty(props, ConstantsClust.NUM_CLUSTERS));}
    \}
\}
\]

The program then stores the smallest BIC error along with the best seed in the appropriate double arrays “smallestErrorAr” and “bestSeedAr” for all of the different number of clusters that have been defined in the properties file and for the number of iterations that is defined there as well. For the example of the properties file in Figure 11., the num_clusters field is defined as 1,,100. That means the above loop will run for 100 different clusters' numbers and each one will produce errors 1000 times (number of iterations). The best will be recorded:

\[
\text{smallestErrorAr[nc]}=\text{smallestError};
\text{bestSeedAr[nc]}=\text{bestSeed};
\text{System.out.println("Cluster "+(nc+1) +" done!");}
\]
After that, the program attempts to create a new file named “smallestError.txt”. In the case where it already exists it informs the user about that. If it doesn't exist, then it creates it, it locates and records the smallest error for all of the different numbers of clusters in the array “smallestErrorAr” that we examined previously, locates and records the best number of clusters that produced this error and then prints in the above file all of the best errors for all the different numbers of clusters.

String fileName = "smallestErrors.txt";

File fi = new File(fileName);

if (fi.exists()) {
    System.out.println("Warning: the file " + fileName + " already exists. No new file created.");
} else {

    PrintWriter outLine = new PrintWriter(new FileWriter(fileName, true));

    for (int f=0;f<numClusters.length;f++) {
        if (totsmallestError>smallestErrorAr[f]) {
            totsmallestError=smallestErrorAr[f];
            bestCluster=(f+1);
        }

        outLine.println(smallestErrorAr[f]);
    }

    outLine.println(smallestErrorAr[f]);
The program finally informs the user about the best clusters' number, the best error and its best seed by printing on the screen this information. Furthermore, with the calls and use of the similar methods “printFuzzyErr” and “printFuzzySeed”, it creates the files “smallestError.txt” and “bestSeed.txt” respectively after it has checked that they do not already exist. These methods print into each of these 2 files the smallest error and the best seed accordingly. Finally, it prints on a new properties file the range of clusters and then through the call and use of the method “printProperties” it adds to this properties file all of the rest necessary information, such as the input data set file, number of clusters, type of clustering, iterations etc.

System.out.println("The best cluster is number " +bestCluster +" because it's error is the smallest with value of " +totsmallestError +" and its seed is " +bestSeedAr[bestCluster-1]);

printFuzzyErr("smallestError.txt", totsmallestError);

printFuzzySeed("bestSeed.txt", bestSeedAr[bestCluster-1]);

props.setProperty(ConstantsClust.NUM_CLUSTERS, origClusters);//prints on the properties file the range of clusters correctly

printProperties(props, referenceCol, targetCol);

}//end if fuzzy c-means

}//end clustering
Apart from the above program that we examined and is named “ClusteringV3.15”, another similar one (“ClusteringV3.15scl”) was also created to deal exclusively with single numbers of clusters to earn time. With a specified -defined in the properties file- successful seed and the best number of clusters, it produces on a file detailed information about the clustering results sorted by each cluster's number. This is achieved by the program with the call of a method named “printFuzzyRes”. This method takes as parameters the property file, the name of the file to be generated and the list which contains all of our elements. It checks whether or not the file exists and if it does it informs the user accordingly. If not, then it simply creates the file and prints in there all of the clustered elements sorted by their clusters using two for loops (one into the other) for the number of clusters and for the number of the elements respectively.

In addition, this program calls the method “printFuzzyLis” which -similarly with the “printFuzzyRes”- produces a file using the logic that we examined above but with a list of each element and its corresponding cluster number (where it belongs to) next to it. The results of this program were used by another one that was made by Dr. Frisco. It checks whether or not all six animals produced similar clustering results with the same successful number of clusters as we will examine in the Section 6.2.

```java
printFuzzyRes(props,"fuzzy4_5col17clanim6.txt",fvd);

printFuzzyLis(props,"fuzzy4_5col17clListanim6.txt",fvd,error,ManagesIO.getIntProperty(props, ConstantsClust.NUM_CLUSTERS));
```
5.2.4 Typographical mistakes in the main resource of Fuzzy C-Means Clustering Algorithm

Two important mistakes in the book of Xu, R. and Wunsch II, D.C.[16] slightly delayed our research while implementing the Fuzzy C-Means clustering algorithm. In detail, in the formula 4.62 of page 88 the written (1-\(m\)) should have been correctly (\(m-1\)) and the first j of the same formula's Dlj/Dij should have also been i, that is correctly Dli/Dij.

5.3 REVEAL Algorithm

According to the REVEAL's activity and class diagrams of the Section 4 as well as on the code, when the program begins in the main method of the class Reveal, it creates an object of the class FileManipie named “fman”. It then calls the method “readSF” in order to read the appropriate files containing our data and assign them to the relevant input and output String two-dimensional transition tables “inTable” and “outTable” respectively. In more detail, these files included normalised data from Dr. Frisco with values -1, 0 and 1 depending on the genes expressions changes for repression, no change and promotion accordingly. The transition tables would take the form:

<table>
<thead>
<tr>
<th>IN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t to (t+1)</td>
<td>t to (t+2)</td>
</tr>
<tr>
<td>t to (t+2)</td>
<td>t to (t+3)</td>
</tr>
<tr>
<td>t to (t+3)</td>
<td>t to (t+4)</td>
</tr>
</tbody>
</table>

**Figure 12. State transition table of REVEAL**

\(t\) to \((t+1)\) is the difference of the 1552 genes expressions for the time between the pre-infestation and 1 hour later, \(t\) to \((t+2)\) is between the pre-infestation and 3 hours later, \(t\) to \((t+3)\) is between the pre-infestation and 6 hours later and the \(t\) to \((t+4)\) is between the pre-infestation and 24 hours later. Therefore the inputs and outputs have been placed for our comparisons in the order that we see on the above table. The 3 distinct rows of the table will be referred as time steps.
Thus, according to the above table and to the code, the “norm_7_8_f1.5” file includes the genes expressions which were normalised with the way that we explained earlier for the columns of the original dataset's file 7 and 8. These columns indicate the pre-infestation state of the genes expressions and 1 hour after the infestation respectively. The same reasoning is applied to the rest of the files “norm_7_9_f1.5” etc. The “readSF” method checks if the specified in the parameter file exists or not (informs the user for that). If it does exist then it reads all of the data from this file and assigns them one by one to each column of the tables “inTable” and “outTable” accordingly, starting from the row 0 and ending to 2.

These tables now together consist what we see in the table of the above Figure 12. They include -1, 0 and 1 values for all of the 1552 genes expressions of the distinct time points t to (t+1), t to (t+2), t to (t+3) and t to (t+4). Please keep in mind though that the 3 distinct rows of the table will be referred from now on as time steps and here the dimensionality problem arises (mentioned in Section 2.2) as we will see in more detail later in the same Section.

```java
public static void main(String[] args) {

    FileManipie fman = new FileManipie();  //file's calls
    fman.readSF("norm_7_8_f1.5", inTable, 0);  //read file's gene expressions and create the
    input and output transition tables for REVEAL
    fman.readSF("norm_7_9_f1.5", outTable, 0);
    fman.readSF("norm_7_9_f1.5", inTable, 1);
    fman.readSF("norm_7_10_f1.5", outTable, 1);
    fman.readSF("norm_7_10_f1.5", inTable, 2);
```
The program then calls the method calcSingleHs which takes as parameters the “inTable” and a double array “entropiesOneInput” in order to calculate the entropies of the former array and assign them to the latter. It checks for the times the -1s, 0s and 1s appear on each gene's expressions in the “inTable” and with the use of the formula $H = - \sum (p(i) \times \log p(i))$, where according to the work of S. Liang, S. Fuhrman and R. Somogyi[10], H is the Shannon entropy (uncertainty) and p(i) is the probability of the number of times that a specific event or symbol i has been observed in the total number of all of the events or symbols:

calcSingleHs(inTable, entropiesOneInput); // i.e. H(A) according to Liang's paper[10]

private static void calcSingleHs (String tableTrans[][], double tableH[])
{

    for (int i=0; i<1552; i++)
    {
        int minuses=0;
        int zeros=0;
        int positives=0;

        for (int j=0; j<3; j++)
        {
            if (tableTrans[j][i].equals("-1"))
            {
                minuses+=1;
            }
            else if (tableTrans[j][i].equals("0"))
            {

            }
        }
    }
}
zeros+=1;
else
positives+=1;
}

if(minuses>0 && zeros>0 && positives>0) //we have to include all of the cases because the log0 results in infinity

tableH[i]= -(minuses/3.0)*(Math.log(minuses/3.0)/Math.log(2))-
(zeros/3.0)*(Math.log(zeros/3.0)/Math.log(2))-
(positives/3.0)*(Math.log(positives/3.0)/Math.log(2));

else if (minuses>0 && zeros >0) //java's log base is e and by dividing it with log(2) we change the base to 2

tableH[i]= -(minuses/3.0)*(Math.log(minuses/3.0)/Math.log(2))-
(zeros/3.0)*(Math.log(zeros/3.0)/Math.log(2));

else if (minuses>0 && positives >0)

tableH[i]= -(minuses/3.0)*(Math.log(minuses/3.0)/Math.log(2))-
(positives/3.0)*(Math.log(positives/3.0)/Math.log(2));

else if (zeros>0 && positives >0)

tableH[i]= -(zeros/3.0)*(Math.log(zeros/3.0)/Math.log(2))-
(positives/3.0)*(Math.log(positives/3.0)/Math.log(2));

else if (minuses>0)

tableH[i]= -(minuses/3.0)*(Math.log(minuses/3.0)/Math.log(2));
Ioannis Efstathiou – MSc in AI

\[
\text{else if (zeros} > 0) \\
\text{tableH}[i] = -(zeros/3.0)*(Math.log(zeros/3.0)/Math.log(2)); \\
\text{else if (positives} > 0) \\
\text{tableH}[i] = -(positives/3.0)*(Math.log(positives/3.0)/Math.log(2)); \\
\}
\]

\} \text{end calcSingleHs}

In exactly the same way, the program then calls again the method “calcSingleHs” in order to calculate again the entropies of the output transition table “outTable” and assign them in the double array “entropiesOneOutput” \( (\text{calcSingleHs(outTable, entropiesOneOutput); /i.e. } H(A')) \). Then it calls the method “calcPairHs” using the parameters “inTable” that we have seen and “entropiesTwoInput” two-dimensional double array. This method, similarly with the above “calcSingleHs”, calculates the pair entropies of all of the inputs of the input state transition table “inTable”, checking all the different frequencies of the pairs combinations between the values -1, 0 and 1, and assigns the results of the entropies in the “entropiesTwoInput” array in a triangular way that we explain in Section 5.3.1. and we can see in Figure 13.

Whenever we would like to retrieve the entropy calculation between any two inputs then we use the lesser number first to refer to the relevant cell of the “entropiesTwoInput” array. For example, if we want to retrieve the entropy calculation between the 35\textsuperscript{th} gene and the 11\textsuperscript{th} then we would find it in the array's cell “entropiesTwoInput[10][34]” because of this triangular reasoning that we will examine in Section 5.3.1. In order to earn some space, the code of this function will not be included here as its logic is similar to the above. The out of memory error appeared with the introduction of the three dimensional double array “entropiesThreeInput” and it is explained in the
Section 5.3.3. The reason that this array was defined was to keep entropies calculation of triplets of inputs and retrieve them easily with the reasoning that we followed previously with the triangular array. However, as it requires huge amounts of RAM, the parallel processing idea arose on this stage.

In more detail, the program calls the method `calcTripletHs` with parameters the “inTable” again and the three dimensional array “entropiesThreeInput”. Similarly with the previous two methods “calcSingleHs” and “calcPairHs”, it calculates the triplets entropies of all of the inputs of the input state transition table “inTable”, checking all the different frequencies of the triplets combinations between the values -1, 0 and 1, and assigns the results of the entropies in the “entropiesThreeInput” array. The lesser numbers have priority again when retrieving them from the appropriate array's cell (i.e. entropiesThreeInput[11][30][952]).

The program then creates an object named “olist1in” of the class `List1in1out` (`
olist1in = new List1in1out();`). This class creates an array list of type `Data1in1out` where a double array is defined called “entropies” . This array is supposed to contain all of the entropies calculations between every output of the transition table “outTable” with each of the inputs (that is every element of our list `List1in1out`) of the transition table “inTable”. In other words, every output has an array which contains 1552 entropies calculations for each of the inputs. Therefore the program calls the method `calcPairHs1in1out` using the above list's object (`
calcPairHs1in1out(inTable, outTable, olist1in);  //i.e. H(C',B)`). This method now, similarly with the “calcSingleHs”, calculates the frequencies of the pair combinations of the values -1, 0 and 1 and assigns the entropies results to each of the above “entropies” arrays that every output has:

```
/**
 * calculates the pair entropies from the String output table with each of the ones of the String
 * input table and assigns them to the double table
```
private static void calcPairHs1in1out (String intableTrans[][], String outtableTrans[][]).
List1in1out lis) {

double tablePH11[] = new double [1552];
List1in1out outputlist = new List1in1out();

for (int b=0; b<1552;b++){
for (int i=0; i<1552;i++){
    int vp00=0;
    int vp01=0;
    int vp0m1=0;
    int vp10=0;
    int vp11=0;
    int vp1m1=0;
    int vpm10=0;
    int vpm11=0;
    int vpm1m1=0;
    int vpm1m1=0;
    double entropy=0;
    for (int j=0;j<3;j++){
if (outtableTrans[j][b].equals("0") && intableTrans[j][i].equals("0"))
    vp00+=1;
else if (outtableTrans[j][b].equals("0") && intableTrans[j][i].equals("1"))
    vp01+=1;
else if (outtableTrans[j][b].equals("0") && intableTrans[j][i].equals("-1"))
    vp0m1+=1;
else if (outtableTrans[j][b].equals("1") && intableTrans[j][i].equals("0"))
    vp10+=1;
else if (outtableTrans[j][b].equals("1") && intableTrans[j][i].equals("1"))
    vp11+=1;
else if (outtableTrans[j][b].equals("1") && intableTrans[j][i].equals("-1"))
    vp1m1+=1;
else if (outtableTrans[j][b].equals("-1") && intableTrans[j][i].equals("0"))
    vpm10+=1;
else if (outtableTrans[j][b].equals("-1") && intableTrans[j][i].equals("1"))
    vpm11+=1;
else
    vpm1m1+=1;
} //end for j

if (vp00>0) {

entropy+=-(vp00/3.0)*(Math.log(vp00/3.0)/Math.log(2));
}
if (vp01>0) {
    entropy+=-(vp01/3.0)*(Math.log(vp01/3.0)/Math.log(2));
}
if (vp0m1>0) {
    entropy+=-(vp0m1/3.0)*(Math.log(vp0m1/3.0)/Math.log(2));
}
if (vp10>0) {
    entropy+=-(vp10/3.0)*(Math.log(vp10/3.0)/Math.log(2));
}
if (vp11>0) {
    entropy+=-(vp11/3.0)*(Math.log(vp11/3.0)/Math.log(2));
}
if (vp1m1>0) {
    entropy+=-(vp1m1/3.0)*(Math.log(vp1m1/3.0)/Math.log(2));
}
if (vpm10>0) {
    entropy+=-(vpm10/3.0)*(Math.log(vpm10/3.0)/Math.log(2));
}
if (vpm11 > 0) {
    entropy += -(vpm11/3.0)*(Math.log(vpm11/3.0)/Math.log(2));
}

if (vpm1m1 > 0) {
    entropy += -(vpm1m1/3.0)*(Math.log(vpm1m1/3.0)/Math.log(2));
}

tablePH11[i] = entropy;

} // end for i

outputlist.setData(tablePH11);

} //end for b

// copying one list into the other

for(int i = 0; i < outputlist.size(); i++) {
    lis.addData(outputlist.getData(i).getArray());
}

After that, the program creates another object named “olist2in” of the class List2in1out (List2in1out olist2in = new List2in1out();). This class creates an array list of type Data2in1out where a double two-dimensional array is defined called “entropies” again. This two-dimensional array is supposed to contain all of the entropies calculations between every output (that is every}
element of our list List2in1out) of the transition table “outTable” with each pair of the inputs of the transition table “inTable”, following the triangular reasoning that we have examined before and is explained in section 5.3.1. In other words, every output has a double two-dimensional array which contains all of the entropies calculations for each pair of the inputs. Therefore the program calls the method calcPairHs2in1out using the above list's object \( \text{calcPairHs2in1out}(\text{inTable}, \text{outTable}, \text{olist2in}); \ //\text{i.e. } H(C',[B,D])) \). This method now, similarly with the previous, calculates the frequencies of the triplets combinations of the values -1, 0 and 1 and assigns the entropies results to each of the above “entropies” two-dimensional array that every output has. As the code again is similar to what we have seen so far, it is not going to be included here in order to earn some space. The reader might refer though to the fully commented provided code to examine any details.

The program then creates another object again named “olist3in” of the class List3in1out \((\text{List3in1out olist3in = new List3in1out();})\). This class creates an array list of type Data3in1out where a double three-dimensional array is defined called “entropies” again. This three-dimensional array is supposed to contain all of the entropies calculations between every output (that is every element of our list List3in1out) of the transition table “outTable” with each triplet now of the inputs of the transition table “inTable”. In other words, every output has a double three-dimensional array which contains all of the entropies calculations for each triplet of the inputs. Therefore the program calls the method calcPairHs3in1out using the above list's object \( \text{calcPairHs3in1out}(\text{inTable}, \text{outTable}, \text{olist3in}); \ //\text{i.e. } H(C',[B,C,J])) \). This method now, similarly with the previous, calculates the frequencies of the quads (quadruplets) combinations of the values -1, 0 and 1 and assigns the entropies results to each of the above “entropies” three-dimensional array that every output has.

After that the program calls the method “compSinOutH1In1OutHs” to compare the entropies, make the appropriate calculations and locate any mutual relationships between each output of the transition table with every one of its corresponding inputs. Therefore this method takes
Ioannis Efstathiou – MSc in AI

as parameters the input transition table “entropiesOneInput”, the output transition table “entropiesOneOutput”, an int two dimensional array named “sinRuleTable” which holds 1 if there is a relationship between the row(input) and the column(output) (i.e. H(C’,A) according to the work of S. Liang, S. Fuhrman. and R. Somogyi[10]), another int array “relationFound” that holds 1 if a relation for the specific element (output) has found and the object “olist1in” that we saw previously of the array list “List1in1out” (compSinOutH1In1OutHs(entropiesOneInput, entropiesOneOutput, sinRuleTable, relationFound, olist1in);).

The method “compSinOutH1In1OutHs” -through the use of a for loop- initially begins initializing the “relationFound” array with 0 and then -with another inner for loop- begins initializing the two dimensional array “sinRuleTable” with 0. It checks for any equalities between the entropies calculations of each input and that of this input with the corresponding output (i.e. H(C)==H(A’,C) according to the work of S. Liang, S. Fuhrman. and R. Somogyi[10]) and if it finds any then it assigns 1 to the “sinRuleTable” and “relationFound” array respectively. This part though has been commented out because we encountered here a problem known as “the dimensionality problem” that we also mentioned in the Section 2.2 according to the work of Cho, K.-H. et al.[6]. The program here located relationships for all of the outputs of the transition table and this occurred because we have only used 3 distinct time steps in the state transition table, for a 3-state system, for the massive amount of 1552 genes.

Therefore, until we will increase the time steps, the above part has been commented out. Having in our entropy calculations arrays only 3 distinct values and many 0s (due to the lack of uncertainty, please refer to Section 5.3.1 for details) the program detected equalities everywhere. However, in the following calculations of the mutual information it was able to provide us with interesting results. Thus, the method “compSinOutH1In1OutHs” calculates the mutual information of each output with each one of the corresponding inputs, divides the result with the entropy of the
input that we have already in the array “entropiesOneOutput” and if the result equals to 1 (i.e. M(A',C)/H(A')==1, where M(A',C)=H(A')+H(C)-H(A',C) according to the work of S. Liang, S. Fuhrman, and R. Somogyi[10]) then it assigns 1 to the “sinRuleTable” and “relationFound” array respectively. This worked fine, it didn't produce mistaken results as before -the division between 0s doesn't equal to 1- and the results will be explained in the next chapter. Because of that, this part is active in the code:

```java
private static void compSinOutH1In1OutHs (double intableTrans[], double outtableTrans[], int sinRuleTable[][], int relationFound[], List1in1out lis) {

    for (int a=0; a<1552; a++) { //each output, i.e. A'
        relationFound[a]=0; //initialization of the table that holds relations findings with 0
    }

    for (int b=0; b<1552; b++) { //each input, i.e. C

        sinRuleTable[a][b]=0; //initialization of the table that holds relations with 0

        // if (intableTrans[b]==lis.getData(a).getElement(b)) //i.e. H(C)==H(A',C)
        // {
        //     sinRuleTable[a][b]=1; //indicates the existence of the relation between a and b
        //     relationFound[a]=1; //indicates that a relation has been found for this element (input)
        // }
        // else {

        if ((outtableTrans[a]+intableTrans[b]-lis.getData(a).getElement(b))/outtableTrans[a]==1) { //M(A',C)/H(A')==1, where M(A',C)=H(A')+H(C)-H(A',C)
            sinRuleTable[a][b]=1; //indicates the existence of the relation between a and b
        }
    }
}
```
The program then defines a new object “pairRuleTable” of the array list “List2In1Out” which is intended to hold 1 if there is a relationship between the element of the list (output of the transition table) AND the row(input) and the column(input) of the 2d triangle array (i.e. H(B',[A,C]) according to the work of S. Liang, S. Fuhrman, and R. Somogyi[10]), otherwise it is 0. After that, it calls the method “compSinOutH2In1OutHs” to compare the entropies, make the appropriate calculations and locate any mutual relationships between each output of the transition table with every pair of its corresponding inputs. Therefore this method takes as parameters the output transition table “entropiesOneOutput”, the input transition table “entropiesTwoInput”, an object “pairRuleTable” of the array list “List2In1Out”, an int array “relationFound” that holds 1 if a relation for the specific element (output) has found and the object “olist2in” that we saw previously of the array list “List2in1out” (compSinOutH2In1OutHs(entropiesOneOutput,entropiesTwoInput, pairRuleTable, relationFound, olist2in);).
Ioannis Efstatthiou – MSc in AI

The method “compSinOutH2In1OutHs” initially defines an object “pplis” of the array list “List2in1out” in order to make all of the appropriate calculations locally. Then through a triple for loop (one into the other), if it finds on the array “relationFound” any cells that contain the value 0 -and that means a relationship hasn't been found for the specific output-, it begins calculating if there are any equalities between the entropies calculations of each pair of inputs and that of this pair with the corresponding output (i.e. $H(A,C)=H(B',[A,C])$ according to the work of S. Liang, S. Fuhrman, and R. Somogyi[10]). It also calculates the mutual information of each pair of outputs with each one of the corresponding inputs and divides the result with the entropy of the input that we have already in the array “entropiesOneOutput”. If the result equals to 1 (i.e. $M(B',[A,C])/H(B')=1$, where $M(B',[A,C])=H(B')+H(A,C)-H(B',[A,C])$) according to the work of S. Liang, S. Fuhrman, and R. Somogyi[10]) then in both cases again it assigns 1 to the “sinRuleTable” and “relationFound” array respectively. At the end, it copies the contents of the temporary array list “pplis” into those of the list “plis” that we used as a parameter.

Here again though we have commented out the equalities calculation part of the entropies between the pairs of inputs and that of those pairs with the corresponding outputs because we encountered the “the dimensionality problem” that we also mentioned in the Section 2.2 according to the work of Cho, K.-H. et al.[6]. It is explained previously on the “compSinOutH1In1OutHs” method’s part too. However, the calculations of the mutual information are not excluded as they also resulted in interesting conclusions according to the next Section of our experiments.

```java
private static void compSinOutH2In1OutHs (double out1tableTrans[], double in2tableTrans[][], List2in1out plis, int relationFound[], List2in1out lis) {

    List2in1out pplis = new List2in1out();

    for (int a=0;a<1552;a++) { //each output, i.e. B'
```
for (int b=0; b<1552;b++){ //each input, i.e. A actually here this for loop with the next one are together for the pairs of inputs

for (int c=(b+1); c<1552;c++){ //each other input, i.e. C that is the next member of the pair

//pplis.addSpData(a,b,c,0); //initialization of each cell of the 2d array for each element of the list with 0

if (relationFound[a]==0) { //if no relation has been found from the previous method where k=1

    if  ((out1tableTrans[a]+in2tableTrans[b][c]-lis.getData(a).getElement(b,c))/out1tableTrans[a]==1) { //i.e. M(B',[A,C])/H(B')==1, where M(B', [A,C])=H(B')+H(A,C)-H(B',[A,C])

        pplis.addSpData(a,b,c,1);//assigns 1 in the 2d array of the list's element indicating the existence of the relation between a (list's element-output of the transition table) AND b(input1) and c(input2)

        relationFound[a]=2;//indicates that a relation has been found for this element (input)

    }

}

else {


}

}
relationFound[a]=2; //indicates that a relation has been found for this element (input)

//
}

}// end if (relationFound[a]==0)

}//end for c

}//end for b

}// end for a

//copying one list into the other

for(int i=0;i<pplis.size();i++) {
    plis.addLData(pplis.getData(i).getArray());
}

}// end compSinOutH2In1OutHs

At the end, the program using a for loop prints on the screen the numbers of all of the outputs of the state transition table along with a number that indicates whether or not a relation for the specific output has been found or not. This number is stored inside each of the cells of the integer array “relationFound”. It is 1 in the case where at least one relation has been found with one of the state transition table's inputs, 2 if at least one input pair has been found to relate to this output or 0 when no relations have been found. Therefore in the code here we can select which of the
above cases we like to print just by modifying the if statement and setting it appropriately to filter out only the desired results which are based on the specific relations number (1, 2 or 0).

```java
for (int a=0; a<1552; a++)
{

    //if (relationFound[a]==1)

    System.out.println(a+1 +" " +relationFound[a]);
}
```

Similarly, the program can print each output of the state transition table with the related inputs (if any) by scanning the elements of the two-dimensional integer array “sinRuleTable” which holds 1 if there is a relationship between the row (input) and the column (output). Otherwise it holds 0.

```java
for (int a=0; a<1552; a++)
{

    System.out.println("Element " +(a+1) +":\n");

    for (int b=0; b<1552; b++)
    {

        if (sinRuleTable[a][b]==1)

            System.out.print((b+1) + ");

    }
}
```
5.3.1 Comments on entropies and the simple 1552 genes' pairs calculation

The entropies are usually calculated from “bits” (binary digits) when using the logarithm on base 2; e.g. the maximum uncertainty (Hmax) equals to 1 for a 2-state system. But for a 3-state system like ours (including -1, 0, 1) then the maximum uncertainty is 1.584962500721156. In other words, when we have one -1, one 0 and one 1 the entropy is 1.584962500721156 (equally distributed). When we have 3 of one kind, for example 3 -1s then the entropy is 0 (because there is no uncertainty) and when we have 2 of a state and one of another then the entropy is 0.9182958340544896 (biased).

After the single calculation of H we wanted to calculate the number of pairs and even of the triplets of H's. It is known that the numbers of these pairs can be calculated from the formula q!/(n! (q-n)!) where q is the number of our genes and n is the number of either pairs or triplets, so 2 or 3. So during the development of the REVEAL algorithm, we needed to know exactly how many pairs of gene inputs we would have to deal with in order to have a further estimation of the algorithm's complexity. In other words, we had to calculate first the factorial of the number of all of our genes, that is 1552! However according to the scientific calculators: 1552! = infinity...

With 20 genes we would have 190 pairs and a lot more triplets that seems to be computationally possible. With 50 genes we would have 1225 pairs, with 100 genes we would have 4950 pairs and 161700 triplets etc. While programming the code of REVEAL in Java, two-dimensional matrices were used in order to keep the entropy calculations of the genes pairs. Since it is not possible to calculate the 1552! with a scientific calculator in order to retrieve the precise number of the possible pairs, the two dimensional arrays offered us an idea of how to find out the exact numbers that we were looking for.

Every pair calculation is stored in the array in a triangle formation (as in a triangular matrix, see Figure 13.). By dividing the total number of the array's squares in half (diagonally) and then
“detaching” the half diagonal that remains and contains comparisons of pairs between them (that is not in our case needed), we could find out the exact number of the pairs. So the derived formula \((q^2/2)-(q/2)\), where \(q\) is the number of the objects, the first parenthesis is the array's division and the second one is the removal of the half diagonal objects (half of our table's diagonal line), resulted in 1203576 pairs.

![Figure 13. Diagram showing the stored entropy pair calculations in our “triangular” two-dimensional array][21].

In the calculation of 5.3.1 and according to the derived formula \((q^2/2)-(q/2)\), \(q\) is the number of the objects vertically or horizontally, the first parenthesis is the array's division in half (i.e. forming the above triangle of the vertical arrows) and the second one is the removal of the half table's diagonal line that remains after the previous division.

5.3.2 Observations on logarithms and calculations between doubles in Java

All of the logarithmic calculations that are written in the work of Cho, K.-H. et al.[6] use the base of 2. Java's default logarithmic base though is \(e\) and in order to change it to 2 it needs to be divided by \(\log[2]\). So a lot of attention has to be paid here to prevent future mistakes. In addition, the division of doubles in Java has to be only between doubles in order to produce the correct result. For example, the calculation of double \(ar = 5.67/3\) will produce \(ar = 0.0\), while double \(ar = 5.67/3.0\) will correctly produce \(ar = 1.89\).
5.3.3 Java heap space issue – out of memory error

This error occurred while compiling the Java project in the NetBeans IDE. In more detail, when the need of keeping the entropies calculations of triplets of elements arose, the first idea was to assign them in three-dimensional arrays. However, the amount of RAM that those arrays require is way too large to handle a dataset such as ours (as an example an \texttt{int[20000][20000]} two-dimensional array needs approximately 1.5GB of RAM). In our case, the initial idea was to create a three-dimensional array consisting of \texttt{1552x1552x1552} cells in order to easily manipulate its data (according to the code, \texttt{private static double[][][] entropiesThreeInput = new double [1552][1552][1552];}). As we suspected though, the RAM couldn't handle it and therefore the need of parallel processing arose.

This attempt wouldn't work even when we tried to take the advantage of most of the tested PC's RAM (6 GB). The method to do that in the NetBeans IDE is by right-clicking on the project, selecting the “Properties” and then “Run”. In the “VM Options” field then we can set the heap size according to our memory needs, for example \texttt{-Xmx 4096m} for 4GBs (but in a 64 bit JVM only, in a 32 bit it typically cannot go more than 1.6GBs). Parallel computing was definitely our next step.

![Figure 14. Changing the heap size in NetBeans IDE](image)

*In the “VM Options” field we can set the heap size according to the memory's needs.*
Section 6: Experiments

6.1 Introduction

The implementation of the Fuzzy C-Means Clustering algorithm on the ovine gene expression data would provide us with a few chosen numbers of clusters where important biological effects should be revealed through correlations between the gene expressions. Ideally, a specific number of clusters would be attempted to be found amongst them, expressing the network's general “wire frame” of the ovine gene expression data as much precisely to the reality as possible. Thus, the Bayesian Information Criterion formula (BIC) calculating the smallest negative errors was used, as we have already seen in every clustering iteration so far, in order to provide us results of successfully refined numbers of clusters. As the complexity, the computational power and the time were expected to be high, we attempted to take the advantage of each of these parameters to the maximum, as we will examine below, always keeping in mind future potential and alternative ways.

In the meantime, as the need of a deeper gene expression interaction knowledge arose, reverse engineering based on the REVEAL algorithm was expected to provide us with details such as relationships and pathways between the genes of the dataset. It's boolean reasoning would be a solid first step in uncovering the gene regulatory network's (GRN) core anatomy while potentially validating the correlated results of Fuzzy C-Means and K-Means ++ clustering algorithms. The inferred output of REVEAL was going to be generated exclusively from gene expression data only and therefore -similarly with the result of the Fuzzy C-Means Clustering- was planned to be directed back to biologists for deeper and further processing as no additional biological data was imported (such as DNA sequence data).
6.2 Experiments and Results using Fuzzy C-Means Clustering

After the finalization of the Fuzzy C-Means clustering's implementation, the first thing that we did was to produce results based on Dr. Frisco's earlier work with K-Means ++ clustering. Thus, clustering the normalised gene expressions differences between the pre-infestation stage and 1 hour after the infestation of the first animal's dataset (this time interval was used in all of the experiments of the Fuzzy C-Means clustering algorithm that follow), we produced 12 individual results for the clusters 2, 3, 4, 5, 6, 8, 13, 26, 36, 50, 51 and 86 respectively. Running 1000 iterations for each of the previous clusters, we calculated and kept the best (smallest negative) errors according to the Bayesian Information Criterion (BIC). By comparisons, we saw that the results that we received from the Fuzzy C-Means clustering algorithm were different than the K-Means ++ ones. 36 clusters (that is number 9 on the plot below) produced the smallest BIC error (-23026.996991799984).

![Figure 15. Plot of Fuzzy C-Means. First Results on Animal 1. BIC errors for the tested 12 clusters: 2, 3, 4, 5, 6, 8, 13, 26, 36, 50, 51 and 86 respectively. Cluster 36 produced the smallest BIC error.](image)

The above results strongly disagreed with the results that Dr. Frisco received from the K-Means ++ clustering. The number of clusters that produced the lowest BIC error according to the K-
Means ++ algorithm was 230. Therefore, in an attempt to ensure ourselves about the two different clustering techniques' disagreement, we proceeded on clustering individual clusters by including in our experiments the clusters 34, 35, 37, 38, 39, 40, 41, 42, 230. The BIC errors were again different and in particular the best clusters (230) of K-Means ++ proved to produce in Fuzzy C-Means a very high BIC error comparing to all of the results that we had so far. It was actually one of the worst and required the highest compilation time of 236 minutes on a CPU with 4 cores.

After the above initial testing processes, we proceeded on performing experiments that were continuous in the number of clusters but of course longer in compilation times. Clustering again using the Fuzzy C-Means on the dataset of Animal 1, for the normalised gene expressions differences between the pre-infestation stage and 1 hour after infestation, we separated this stage into 7 steps dividing the range of the tested clusters to:

- Step 1 : 1-20 clusters – Compilation time of 10 minutes (in a quad core CPU)
- Step 2 : 21-40 clusters – Compilation time of 77 minutes (in a quad core CPU)
- Step 3 : 41-60 clusters – Compilation time of 194 minutes (in a quad core CPU)
- Step 4 : 61-80 clusters – Compilation time of 337 minutes (in a quad core CPU)
- Step 5 : 81-100 clusters – Compilation time of 539 minutes (in a quad core CPU)
- Step 6 : 101-120 clusters – Compilation time of 2.718 minutes (in a dual core CPU)
- Step 7 : 121-140 clusters – Compilation time of 3.714 minutes (in a dual core CPU)

From these experiments and according to the plot below, the Fuzzy C-Means clustering algorithm resulted in 12 clusters, having the smallest BIC error of -23195.272671546278.
Figure 16. Plot of Fuzzy C-Means. Results for clusters 1-140 on Animal 1 and high threshold. *BIC errors for the tested 140 cases of clusters. 12 clusters produced the smallest BIC error.*

According to the above plot, the BIC error increases evidently as the experiments approach 140 number of clusters. Considering also the fact that 230 clusters produced a BIC error above -20000, were enough to convince us that performing experiments for more than 140 clusters on Fuzzy C-Means would be a waste of time (as the compilation times increase exponentially according to our 7 steps above) and therefore would be incredibly high. Thus, we concluded on the fact that 100 clusters would be enough for future experiments.

In the same way, using the datasets that we had for the animals 2, 3, 4, 5, and 6 we run the same tests to see if there were any similarities. These tests can be seen from the plots below.
Figure 17. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 2 and high threshold. BIC errors for the tested 100 cases of clusters. 14 clusters produced the smallest BIC error.

Figure 18. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 3 and high threshold. BIC errors for the tested 100 cases of clusters. 42 clusters produced the smallest BIC error.
Figure 19. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 4 and high threshold.
BIC errors for the tested 100 cases of clusters. 37 clusters produced the smallest BIC error.

Figure 20. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 5 and high threshold.
BIC errors for the tested 100 cases of clusters. 28 clusters produced the smallest BIC error.
Figure 21. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 6 and high threshold. *BIC errors for the tested 100 cases of clusters. 54 clusters produced the smallest BIC error.*

From the above tests that occurred on the gene expressions of the 6 different animals we verified that indeed there was no need to run the experiments for more than 100 clusters. The BIC errors obviously increased as we were approaching 100 clusters for all of the 6 animals. Furthermore, we concluded on the fact that the Fuzzy C-Means clustering algorithm produced in general very small errors for the range of clusters between 12-17. That was something important to notice and work on further despite the fact that apart from the 12 and 14 clusters that are in this range for the Animals 1 and 2 respectively, we also accepted as the best clusters the 42, 37, 28 and 54 for the Animals 3, 4, 5 and 6 accordingly.

<table>
<thead>
<tr>
<th>Animal 1</th>
<th>Animal 2</th>
<th>Animal 3</th>
<th>Animal 4</th>
<th>Animal 5</th>
<th>Animal 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIC Errors for clusters 12-17</td>
<td>-</td>
<td>23195.272</td>
<td>16156.235</td>
<td>15292.368</td>
<td>17293.566</td>
</tr>
<tr>
<td></td>
<td>671546278</td>
<td>663041256</td>
<td>492382711</td>
<td>214992305</td>
<td>950477584</td>
</tr>
<tr>
<td></td>
<td>21834.736</td>
<td>17192.554</td>
<td>15375.441</td>
<td>15244.035</td>
<td>18240.257</td>
</tr>
</tbody>
</table>

83
Figure 22. Table of Fuzzy C-Means. Results for clusters 12-17 on Animals 1-6 and high threshold.

BIC errors for clusters 12-17. The BIC error of the best cluster is always equal or close to one of those in the range between 12-17.

According to the table and based on our above hypothesis, we initially conclude on the best clusters for each animal in the range between 12-17 as:

- Animal 1: cluster 12 (the best Cluster)
- Animal 2: cluster 14 (the best Cluster)
- Animal 3: cluster 12 (closest to the best Cluster 42)
- Animal 4: cluster 17 (closest to the best Cluster 37)
- Animal 5: cluster 17 (closest to the best Cluster 28)
- Animal 6: cluster 17 (closest to the best Cluster 54)

All of the above tests required a lot of time and computational power as we have seen. However, we reached a point were we should conclude on the exact number of clusters that
evidently would give us the smallest BIC error considering the datasets of all of our 6 animals. Thus, we proceeded with two ideas that should run simultaneously:

- We should decrease the threshold of the Fuzzy C-Means Clustering algorithm a lot more in order to produce deeper and more clear results for the range of clusters between 12-17. However, we knew that the compilation times would also increase significantly.

- We should run the 6 above experiments again for the range of clusters 1-100 on data that were normalised from biologists in a final attempt to verify the validity of our results.

We managed to acquire the normalised (from the biologists) datasets immediately and began running the experiments as soon as possible. The decreased threshold for the clusters 12-17 for each animal required on average 740 minutes of compilation for each one and produced the following results:

![Figure 23. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 1 with low threshold.](image)

*Figure 23. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 1 with low threshold.*

*BIC errors for the tested 6 cases of clusters. 17 clusters produced the smallest BIC error.*
Figure 24. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 2 with low threshold.  
*BIC errors for the tested 6 cases of clusters. 16 clusters produced the smallest BIC error.*

Figure 25. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 3 with low threshold.  
*BIC errors for the tested 6 cases of clusters. 17 clusters produced the smallest BIC error.*
Figure 26. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 4 with low threshold. 
*BIC errors for the tested 6 cases of clusters. 16 clusters produced the smallest BIC error.*

Figure 27. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 5 with low threshold. 
*BIC errors for the tested 6 cases of clusters. 17 clusters produced the smallest BIC error.*
Figure 28. Plot of Fuzzy C-Means. Results for clusters 12-17 on Animal 6 with low threshold.

BIC errors for the tested 6 cases of clusters. 16 clusters produced the smallest BIC error.
Figure 29. Table of Fuzzy C-Means. Results for clusters 12-17 on Animals 1-6 and low threshold.

BIC errors for clusters 12-17. The BIC error of 17 clusters is the lowest on most of the occasions so far.

According to the above plots and the table with the results we see that the best number of clusters according to the Fuzzy C-Means clustering algorithm (using a very “narrow” threshold) is 17. Even in the three cases where 16 was the best we clearly notice that 17 was very near.

On the other hand, the normalised from the biologists datasets for the 6 animals respectively resulted in the following plots:

Figure 30. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 1 with normalised from biologists data and high threshold.

BIC errors for the tested 100 cases of clusters. 12 clusters produced again the smallest BIC error.
Figure 31. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 2 with normalised from biologists data and high threshold.

\(BIC\) errors for the tested 100 cases of clusters. 16 clusters produced the smallest \(BIC\) error (14 before).

Figure 32. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 3 with normalised from biologists data and high threshold.

\(BIC\) errors for the tested 100 cases of clusters. 26 clusters produced the smallest \(BIC\) error (42 before).
Figure 33. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 4 with normalised from biologists data and high threshold.
BIC errors for the tested 100 cases of clusters. 26 clusters produced the smallest BIC error (37 before).

Figure 34. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 5 with normalised from biologists data and high threshold.
BIC errors for the tested 100 cases of clusters. 20 clusters produced the smallest BIC error (28 before).
Figure 35. Plot of Fuzzy C-Means. Results for clusters 1-100 on Animal 6 with normalised from biologists data and high threshold.

BIC errors for the tested 100 cases of clusters. 11 clusters produced the smallest BIC error (54 before).

<table>
<thead>
<tr>
<th>Animal 1</th>
<th>Animal 2</th>
<th>Animal 3</th>
<th>Animal 4</th>
<th>Animal 5</th>
<th>Animal 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BIC Errors for clusters 12-17</strong></td>
<td><strong>BIC Errors for clusters 12-17</strong></td>
<td><strong>BIC Errors for clusters 12-17</strong></td>
<td><strong>BIC Errors for clusters 12-17</strong></td>
<td><strong>BIC Errors for clusters 12-17</strong></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9621.2327</td>
<td>25275315</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8930.1937</td>
<td>29177598</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8965.8250</td>
<td>5915615</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>9015.3183</td>
<td>33356045</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8945.3140</td>
<td>79913169</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>8728.5322</td>
<td>87140813</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BIC Error of</th>
<th>BIC Error of</th>
<th>BIC Error of</th>
<th>BIC Error of</th>
<th>BIC Error of</th>
<th>BIC Error of</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
From the above table of results and plots it is apparent that despite the fact that the normalised results from the biologists are not identical to our previous ones, they are similar. The BIC errors of old and new best clusters are near on every animal, if not exactly the same, compared always to the worst values that have been observed approaching 100 clusters (and therefore the BIC errors of 100 clusters have been presented in the above table). Thus, we reached to a solid verification of our initial results between clusters 1-100. Furthermore, the proximate BIC errors again of the ranges between 12-17 with the current best clusters reinforced our beliefs that 17 clusters might indeed represent an interesting, valid biological “wire frame”.

The last step was to check that all six animals produced similar clustering results with 17 clusters. Using a program created by Dr. Frisco that checks and intersects similar clusters for all of the genes according to their names, we found out that only 554 genes were in different clusters out
of the 1552. This is an interesting result as 2/3 of our data set's objects agreed on all of the six different animals. Furthermore, when we used only the animals 1,3,5 with 17 clusters (these animals had initially resulted in 17 clusters as their best ones according to the above results of Figure 29., while the animals 2,4,6 had 16 as their best) the results were much better with only 209 genes being in different clusters. That was 86.5% similarity.

The above significant results strongly verified the fact that 17 clusters may indeed represent the core anatomy of the eukaryotic scale-free gene ovine network under the infestation of the ectoparasitic mite psoroptes ovis. The scale-free evidence is based on the fact that 3 out of the successful 17 clusters included most of the 1552 genes expressions. In addition, the largest of these 3 clusters consists of 1075 genes and is considered to include the main hubs of the ovine gene network. Its vast size is emphasising the scale-free nature.

According to the work of K. Basso et al.[2] these properties are valid, as their results suggested: “a hierarchical, scale-free network, where a few highly interconnected genes (hubs) account for most of the interactions” -in eukaryotes-. Furthermore, our apparent scale-free results also agree with the work of S.A. Ramsey et al.[12] where they inferred a macrophage transcriptional network. According to their statement: “The out degree distribution appears to be scale-free, consistent with previous reports for mammalian networks”. The outcome of our case has definitely a place among these reports. Seventeen ovine eukaryotic clusters -including a giant one which consists of 1075 genes- might indeed account for all of the interactions between the 1552 mammalian genes.

6.3 Clustering by Distance results

On the ovine gene expression data, the results of the algorithm for 17 clusters presented
many similarities with the results of the Fuzzy C-Means Clustering algorithm reinforcing its validity. Both agreed in the creation of one giant cluster containing similar expressions, in the creation of a few distinct others containing a small number of mostly the same data points and on a few cases, two distinct but sequential clusters of the Fuzzy were presented as one in the results of the Clustering by Distance algorithm. The nature of the latter though left many clusters empty.

### 6.4 Experiments and Results using the REVEAL algorithm

According to the REVEAL's implementation of Section 5.3, the methods “compSinOutH1In1OutHs” and “compSinOutH2In1OutHs” provided us with interesting results. In more detail, the first method initially produced relationships between every output of the state transition table with each of the inputs, finding equalities in the entropies calculations between the members of any “pair”. This result may be correct, it is not insightful at all though as we need depth in our conclusions when attempting to extract the anatomy of the ovine gene network. The dimensionality problem that we have mentioned in the Section 2.2 as well as in the implementation of the algorithm in Section 5.3, wouldn't allow us to extract any more details on this point with 1552 genes and only 3 distinct time steps in out transition table. For our 3-state system the entropy results that we had (please refer to Section 5.3.1 for details) were either 1.584962500721156 (equally distributed), 0 (because there is no uncertainty) or 0.9182958340544896 (biased). The 0s were many as well as their cases of equality.

Due to the next, more advanced step of REVEAL though we managed to produce thorough results. Here the calculation of the mutual information occurs and then follows its division by the entropy of a specific output. If the division equals to 1 then we have just “revealed” a new single relationship between the output and the corresponding input (therefore 0s are not considered here as
successful nominators and denominators of fractions that equal to 1). That was the case for 500 genes out of the total number of 1552, that is approximately 1/3 of our dataset. These genes presented single relations with each of the inputs of the transition table and therefore we concluded on the fact that a giant cluster (component probably including the main hubs) might had just been found.

This significant finding verified the work of Katia Basso et al.[2] where there was strong evidence that the gene expressions of high-order eucaryotic cells can be represented by scale-free networks, as well as the work of S.A. Ramsey et al.[12]. Their degree distribution follows a power law and therefore we have the birth of a giant robust cluster. Relations (or connections) between its gene-elements (or nodes) are typically multiple and, as in our case, can be fully connected. Furthermore, the number of the directly related genes which are forming the giant cluster is a part of that which was produced from the Fuzzy C-Means clustering algorithm reinforcing its validity even more. Thus, strong evidence of the ovine gene network's scale free nature was retrieved again.

In addition to the above comparisons and results, the algorithm's second method “compSinOutH2In1OutHs” which produces relationships between outputs of the state transition table with each pair of the inputs, attempting to find similarities between the members of any “triplet”, also verified their validity. It resulted again in exactly the same 500 genes which had direct relations with each pair of the state transition table's inputs through the calculations of their mutual information. This second verification clearly presented to us the potential existence of the ovine gene network's giant cluster, along with each of its genes' details, under the infestation of the psoroptes ovis parasitic mite.
Section 7 : Evaluation

7.1 Fuzzy C-Means and K-Means ++ clustering results

According to the results that both of the algorithms have produced by clustering the same elements, we noticed that Fuzzy C-Means algorithm tends to apply some of the values of these elements as the highest of their clusters. On the other hand, the K-Means ++ algorithm usually applies the same values as the lowest of their clusters. This is mainly related to the non-strict logic that Fuzzy C-Means clustering follows by applying weights on all of its data points according to the values of each of the clusters as well as on the fact that it ensures by this way that no empty clusters will be ever created.

However, K-Means ++ usually creates empty clusters and mostly when their pre-defined number increases significantly. This is happening because of its arbitrary behaviour as we have seen in the Research Report. Therefore, on the errors' calculations from the BIC functions that we examined in Section 2.4, K-Means ++ algorithm's function doesn't include the number of the clusters that the user has pre-defined in the properties file (like in the Fuzzy C-Means algorithm) but uses the real number of clusters excluding the empty ones.

Furthermore, the Fuzzy C-Means algorithm's errors which were generated from normalised data presented a wider variety (and better results) when originating from smaller number of clusters. Bigger number of clusters and especially close to 100 (and after that) presented big errors which were almost in the same range. The same behaviour was observed from the K-Means ++ clustering algorithm too regarding the variety part. However, the main difference with this algorithm is that its best number of clusters was 260, in contrary to the 17 of the Fuzzy C-Means. Bigger number of clusters here presented good results. Many smaller errors were observed compared to those of fewer
number of clusters and therefore Dr. Frisco's experiments reached 500 clusters.

7.2 Clustering by Distance algorithm's discussion

This algorithm was created for experimental purposes as an introductory step to the implementation of the Fuzzy C-Means clustering algorithm. It is a fast method that produces detailed clustering results indicating the exact distances (in percentages) of every data object to each of the clusters too. Like K-Means ++, it creates empty clusters but unlike Fuzzy C-Means, during its execution all of the centroids remain always in the range of the clusters. That means there are no centres allocated outside of the space that is defined between the difference of the minimum and maximum values of the data points and therefore there is no delay.

It's simple, straight-forward reasoning is ideal to produce quick clustering results based only on distance where detailed relations between the data points are necessary to be uncovered, such as in data sets where the elements values vary, covering as much of the space as possible and their number typically is large. In the case where the elements aren't spread in the space but are mainly located on a specific part of it then the algorithm is unable to take the advantage of the empty clusters properly and that's a result of its reasoning. However, by increasing the pre-defined number of clusters and ignoring the empty ones, we can increase the detail of the clustering results according to our needs and produce more suitable results.

7.3 REVEAL's evaluation

In contrary to the Fuzzy C-Means clustering algorithm, REVEAL runs a lot faster. Despite the fact that it performs exhaustive search we didn't experience any problems with its speed at all. Only 12 minutes on a quad core machine were enough to produce results from 1-by-1 and 1-by-2 relations between the outputs with the inputs respectively of the state transition table. The only
problem here is the huge amount of memory that requires for the comparisons of 3 or more arguments in our dataset (consisting of 1552 gene expressions) and therefore multiple processing and the resources of more than one PCs are needed.

Section 8: Conclusion

REVEAL extracted a cluster of 500 fully connected genes which has been found to be a subset of the Fuzzy C-Means giant cluster out of the 17 successful ones. It's results were produced from 1-by-1 and 1-by-2 relations between the outputs with the inputs respectively of the transition table, both agreeing on the same 500 genes. In addition, the different in nature Clustering by Distance algorithm agreed in most of the parts with the Fuzzy C-Means for 17 clusters and extracted a giant cluster containing similar gene expressions. Despite the fact that we only had 3 time steps (rows) in REVEAL's transition table and we suspected that we wouldn't have thorough results, we still retrieved strong evidence about the ovine gene network's scale free nature with detailed information of the genes that consist its giant cluster (component).

8.1 Future Suggestions

The next step is to conclude on and increase the time steps of the REVEAL's transition table in order to produce more detailed results. This means that the code we have examined needs to be modified and then the concluded time steps need to be added in the table to solve the dimensionality problem that has been mentioned in Sections 2.2 and 5.3. More variety in the entropy calculations will definitely produce more details. Thus, the addition of at least 1-2 more time steps in the transition table is necessary to achieve that. For example, we could consider the in between time
intervals of the 1st hour after the infestation with the pre-infestation state, of the 3rd hour with the 1st, of the 6th with the 3rd and of the 24th with the 6th.

The main question though here is whether or not the addition of the extra potential time points would make sense for Biology, keeping this project on the correct path and therefore their selection and addition should be made with the agreement of biologists too. After the finalization of this part, the next step would be to run REVEAL on a parallel task dispatching program for processing of 3 input arguments of the transition table in order to take the advantage of the parallel computation and of the memory resources of multiple Pcs/processors. Furthermore, as only the first time interval of the ovine gene expressions has been used in all of the Fuzzy C-Means clustering experiments so far, it would be wise to work on the other four too in the future. The potential results from the additional four might reinforce even more the validity of those which were produced from the time interval between the pre-infestation stage and 1 hour after the infestation. The outcome of this project along with the future suggestions can be brought further in a PhD programme.
Section 9 : References


**References of figures**


Section 10 : Appendices

10.1 Screenshots from Clustering by Distance algorithm
10.2 Screenshots from Fuzzy C-Means Clustering algorithm
for (int j=0; j<prop.size(); j++) {
    double mean = prop.getDouble(j).doubleValue();
    for (int i=0; i<props.size(); i++) {
        if (props.get(i).getDouble(i).doubleValue() < mean) {
            prop.set(i, mean);
            prop.add(i, prop.getDouble(i));
        }
    }
    double err = prop.getError(prop.getDouble(j), props.get(i).getDouble(i));
    if (err < smallestError) {
        smallestError = err;
        for (int j=0; j<prop.size(); j++) {
            prop.set(i, prop.getDouble(i));
        }
    }
}
//System.out.println("error: "+error);
/*
    if (error < smallestError) {
        smallestError = error;
        for (int i=0; i<prop.size(); i++) {
            prop.set(i, prop.getDouble(i));
        }
    }
*/
}
The best obtained cluster has been obtained for 17 clusters (17 real clusters) and error -18710.301780525348

6.202558705014705E-4 2
0.0017746638836544011
8.49173506366154E-5 2
-0.00464392366157688115
-2.3069703577932048E-4 2
-2.476190102812021E-42
-0.0010539370082203639 3
0.00220000814862091295 1
1.0092518603009495E-4 2
0.00217426768166711 1
0.010903270956230891 6
-0.004211050781248056 4
-0.006544201234010289 15
-2.826625408717257E-43
10.3 Screenshots from REVEAL algorithm