Organising and Controlling Distributed Systems by Clustering Nodes by Context and Electing Leaders Using Inductive Program Synthesis

Filip Hörnsten

Department of Computer and Systems Sciences

Degree project 15 HE credits
Computer and Systems Sciences
Degree project at the bachelor level
Spring term 2018
Supervisor: Rahim Rahmani
Reviewer: Jakob Tholander

Stockholm University

Abstract

A new approach for organising and adaptively controlling distributed systems is presented by utilising clustering based on context and recent advances in inductive program synthesis to facilitate leader election. The proposed approach is meant to solve the problem that it can be difficult to control a distributed system if the number of states that the system can be in at any moment is large and there are many nodes. The research question that is answered is: How can nodes in a distributed system be clustered according to context and how can leader election be done by inductive program synthesis? Design science is used as the research strategy and its five-step method framework is applied to the development of an artefact that can function as a framework for organising and adaptively controlling distributed systems. The empirical results show that it is possible to use inductive program synthesis to synthesise simple leader election programs and that clustering is useful in organising nodes in a distributed system. The use of inductive program synthesis to elect leaders allows developers of distributed systems to express leader election algorithms by supplying abstractions in the form of input-output examples instead of writing complete programs that implement the algorithms. These abstractions may also enable the artefact to adapt to unforeseen circumstances. Thus, the artefact is deemed a promising step towards a framework for organising and adaptively controlling distributed systems, which will hopefully lessen the amount of work needed to write programs that can control them.

Keywords: Inductive program synthesis, clustering, leader election, distributed systems, machine learning, DeepCoder

Synopsis

Background	Distributed systems are systems in which the individual components are separated from each other but are connected through some form of network. The components can only communicate by sending messages to each other. The research conducted in this thesis belongs to the area of Immersive Networking.
Problem	Controlling a distributed system can be difficult if the number of states that the system can be in at any moment is large and there are many nodes in the system.
Research Question	The research question that is answered in this thesis is: How can nodes in a distributed system be clustered according to context and how can leader election be done by inductive program synthesis? This particular research question was chosen because answering it will help solve the problem described above by testing a new way of organising and controlling distributed systems.
Method	The research strategy design science is used in the thesis. Design science and the five activities of its method framework will be applied to the research question to create an artefact that will be evaluated based on the practical problem that it should solve. Both qualitative and quantitative data will be collected by testing the artefact in different scenarios which will be analysed using descriptive statistical analysis.
Result	An artefact that utilises clustering and inductive program synthesis was developed that can solve the problem previously described. The empirical results of the thesis show that it is possible to use inductive program synthesis to synthesise simple leader election programs by using DeepCoder's LIPS framework and that k-medoids clustering can be useful in organising the system into groups based on the context of the different nodes. Thus, the answer to the research question is that k-medoids clustering and DeepCoder's LIPS framework can be used to organise and control distributed systems.
Discussion	The Silhouette values calculated using the Manhattan distance could not be included in the results and Silhouette values calculated using the Euclidean distance are mentioned but not analysed. The artefact is a novel application of DeepCoder and using IPS to elect leaders allows for the expression of leader election algorithms by supplying abstractions in the form of input-output examples instead of writing complete programs that implement the algorithms. These abstractions may also enable the artefact to adapt to unforeseen circumstances. Reduced work needed by programmers to write programs that control distributed systems is a possible implication of the artefact.

Acknowledgements

I would like to thank my supervisor Rahim Rahmani for supporting me during the writing of this thesis. He always answered my questions quickly, sometimes late at night and even on weekends. I would also like to thank my family for supporting me, especially Lena for getting me interested in science at an early age even though I probably didn’t realise it until I was an adult.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DFS Depth-first search
IoT Internet of Things
IPS Inductive Program Synthesis
LIPS Learning Inductive Program Synthesis
PAM Partitioning Around Medoids
RNN Recurrent Neural Network

1 Introduction

1.1 Background

Distributed systems are systems in which the individual components are separated from each other but are connected through some form of network. The components can only communicate by sending messages to each other. Examples of distributed systems are the Internet and other telecommunication networks, distributed databases, industrial control systems and scientific simulation systems. (Coulouris et al., 2012; Cachin, Guerraoui & Rodrigues, 2011)

In order to coordinate their actions, it is required that components in a distributed system are able to communicate, and to control distributed systems reaching consensus is required (Yu, Zhang & Sun, 2010). One way of coordinating a distributed system is by leader election, which appoints one process as the leader that the others will follow (Coulouris et al., 2012; Cachin, Guerraoui & Rodrigues, 2011). Thus, the distributed system can be controlled externally by communicating with the leader instead of every component of the system.

By the definition of a distributed system presented earlier in this section, the Internet of Things (IoT) could be seen as an inherently distributed system. According to Rahman and Rahmani (2018), context information can be used in IoT systems to get value out of raw data and help turn it into intelligence and knowledge. They also foresee in their paper *Enabling Distributed Intelligence Assisted Future Internet of Things Controller* that context information combined with artificial intelligence to provide edge intelligence will be imperative in future IoT solutions. (Rahman & Rahmani, 2018).

Clustering is a way of grouping a set of objects together such that the objects in one group are similar to each other and dissimilar to objects in other groups (Alpaydin, 2014). Program synthesis performs a search over the space of possible programs to generate a program that is consistent with some constraints. Inductive program synthesis can be used to generate a program from, for example, input-output examples. (Gulwani, Polozov and Singh, 2017)

1.2 Problem Description

The problem that this thesis addresses is that controlling a distributed system can be difficult if the number of states that the system can be in at any moment is large and there are many nodes, thus reaching consensus is hard. Leader election where changes are done by the leader and then propagated to the followers is efficient because it minimizes the external interaction required to control the system. However, programming complex systems that change a lot requires a lot of code that control the system depending on its state – as the number of possible states increase, so does the code required to control the system.

1.3 Research Question

To create order in a distributed system, I propose to use clustering based on context information to create groups in the system. This context information may be based on different types of data, such as geolocation data, type of sensors or actuators used by a certain node, computing power of the node etc.

An aggregation of several types of data may also be used. Machine learning is adaptive in that learning can be used to change the behaviour of a program depending on prior belief and experience. As inductive program synthesis can be used to generate a program from input-output data, this means that the system does not need to be pre-programmed with all possible actions and states.

I argue that when combined in an artefact, clustering and inductive program synthesis could potentially decrease the amount of code that has to be written by a human programmer to organise and control a distributed system in an efficient way, while also adapting to new, unforeseen circumstances.

Therefore, the research question in this thesis is: How can nodes in a distributed system be clustered according to context and how can leader election be done by inductive program synthesis?

1.4 Delimitations

The type of distributed systems that are relevant to the research question of this thesis is decentralised distributed systems. The generation of context information is outside the scope of this thesis, instead simulated context values will be used. Clustering is done using the PAM version of the k-medoids algorithm with an implementation of the k-means++ algorithm as the initialiser. The leader election algorithm that is to be implemented is a hierarchical leader election algorithm. Inductive program synthesis is facilitated by the DeepCoder LIPS framework. All code will be written in Python since many machine learning algorithms are already implemented in Python and thus can be reused. Testing is done in a simulated distributed system, not a real one.

1.5 Thesis Structure

Section one introduces the necessary background to understand the research question and problem, followed by a description of the problem, research question and delimitations. Section two explains necessary concepts in-depth based on previous research to provide a scientific base to the research conducted in the thesis. Section three describes and motivates the choice of research strategy and methods as well as how they were applied to the research question, along with alternatives and ethical considerations. Section four looks deeper at the problem that motivates the research conducted in this thesis and defines requirements that the artefact that is to be developed should fulfil. Section five presents the results by first describing the design and development of the artefact and then moving on to demonstrate that the artefact can solve the practical problem that motivates the research. The results are then analysed, and the artefact is evaluated. Section six summarises the results and argues their originality and significance. Then follows a discussion on topics such as the quality of the thesis, limitations of the research and potential future work. Finally, the conclusions of the thesis are presented.

2 Extended Background

This section presents concepts needed to understand the research question and the problem by providing explanations that are based on previous research. The research being conducted in this thesis is within the area of Immersive Networking. Some of the concepts that are presented and explained in this section are: distributed computing and distributed systems, leader election and consensus, inductive program synthesis, deep learning and clustering.

2.1 Distributed Computing

Distributed computing concerns the study of distributed systems. According to Cachin, Guerraoui and Rodrigues (2011, p.1) “distributed computing addresses algorithms for a set of processes that seek to achieve some form of cooperation”.

2.1.1 Distributed systems

A distributed system is a system in which components are located on networked computers and communicate and coordinate only by sending messages to each other. Significant characteristics of distributed systems are: concurrency of components, lack of a global clock and independent failures of components. Examples of distributed systems include financial trading systems, massively multiplayer online games and grid computing. (Coulouris et al., 2012)

There are several types of distributed systems. One type is the decentralised distributed system, which seeks to increase scalability and fault-tolerance by being self-organising and distributing decision-making throughout the system. A distributed program is a program running in a distributed system and distributed programming is the act of writing them. (Coulouris et al., 2012; Cachin, Guerraoui & Rodrigues, 2011)

2.2 Consensus and Leader Election

2.2.1 Consensus

In distributed computing, processes use consensus to agree on a common value out of values that one or more of the processes have proposed (Coulouris et al., 2012). According to Yu, Zhang and Sun (2010, p.5958) “consensus means to reach an agreement regarding a certain quantity of interest that depends on the states of all agents”. In the second edition of their book *Introduction to reliable and secure distributed programming*, Cachin, Guerraoui and Rodrigues (2011) write that reaching consensus is one of the fundamental problems of distributed computing. They further state that any algorithm that facilitates maintaining common state or deciding upon future actions where multiple processes are involved, some of which may fail, involves solving a consensus problem.

2.2.2 Leader election

In distributed computing, leader election is the act of choosing a unique process to perform a certain kind of role, for example to coordinate the actions of other processes (Cachin, Guerraoui & Rodrigues, 2011; Coulouris et al., 2012). Coulouris et al. (2012) go on to explain that an individual process can

initiate a call for one leader election at a time, however different processes in a distributed system can initiate a call for election concurrently. Therefore, the result must be unique meaning that two processes cannot elect two different leaders if they called for an election concurrently without violating the requirements of leader election. If the current leader crashes a new leader should be elected (Cachin, Guerraoui & Rodrigues, 2011).

Consensus can be used to elect leaders and vice versa as shown by Cachin, Guerraoui and Rodrigues (2011, p.225). Therefore, it can be argued that leader election is a specific application of consensus, but also that leader election can be used to reach consensus.

2.3 Machine Learning

There are three distinct types of machine learning – supervised, unsupervised and reinforcement learning. In supervised learning the goal is to make predictions about the future by learning a function that maps inputs to outputs. An example of supervised learning is classification, in which new observations must be classified according to predictions based on previous knowledge. The goal in unsupervised learning is to discover hidden patterns in the data to be able to describe the data in a better way than before. An example of unsupervised learning is clustering. (Alpaydin, 2014)

2.3.1 Clustering

Formally, clustering is a mixture model density estimation approach to grouping a set of objects together (Alpaydin, 2014). According to Schaeffer (2007, p.27) a cluster is a group of data such that the objects within a group are similar to one another and different from the objects in other groups according to some similarity measure. In other words, intra-cluster distances are minimised while inter-cluster distances are maximised. Each object belongs to exactly one cluster (Alpaydin, 2014).

One example of a clustering algorithm is k-means clustering, originally used for vector quantization. It is an iterative procedure that finds groups in data where the groups are represented by their centres. The centres, called centroids, are initialised randomly and are the typical representatives of their group. For each iteration the centres are adjusted to minimize the sum of the Euclidean distance squared to find locally optimal solutions. The number of clusters, *k*, must be chosen before running the algorithm. This means that the result of the algorithm will vary depending on the number of clusters chosen, possibly not reflecting meaningful groups in the data. The initial choice of points can also greatly influence the result of the k-means algorithm, as it can converge with a bad result given a bad placement of the initial points (Alpaydin, 2014; Bradley & Fayyad, 1998).

Thus, there are several issues that arise when the k-means algorithm alone is used for clustering. This motivates the use of other clustering algorithms such as k-means++, first introduced by Arthur and Vassilvitskii in 2007. It is a variant of the k-means clustering algorithm that chooses centres at random from the data points but weighs in the squared distance squared from the closest centre that had already been chosen to choose the initial points. This ensures that the clustering algorithm is not initialised with a bad set of initial points while also potentially increasing the speed of the algorithm and the accuracy of the final clustering. (Arthur & Vassilvitskii, 2007).

Another clustering technique is k-medoids, first proposed by Kaufmann and Rousseeuw (1987), wherein the representative object of a cluster is the object with the minimal dissimilarity compared to all the other objects in the same cluster. Thus, the representative object of the cluster is an actual data

point, called the medoid, rather than the centroid as in the k-means algorithm (Kaufmann & Rousseeuw, 1987).

One of the most common k-medoids clustering algorithms is Partitioning Around Medoids (PAM), which is more robust to outliers than the k-means algorithm and uses the Manhattan distance but is more computationally expensive on large datasets because of its computational complexity. It is not possible for the k-medoids algorithm to converge with a bad result because of the choice of initial points. (Park & Jun, 2009). There are other k-medoids algorithms than PAM that try to increase speed of execution, such as the one introduced by Park and Jun (2009) in their paper *A simple and fast algorithm for K-medoids clustering*.

After clustering has been done on a dataset, it can be evaluated using two classes of evaluation – internal and external evaluation. Internal evaluation metrics evaluate the clustering based only on the data that was used for clustering and typically measure intra-cluster and inter-cluster similarities. It can be used to choose the optimal number of clusters as it measures the compactness and separation of clusters. External evaluation metrics evaluate the clustering based on some external information such as the class labels of the objects that have been clustered and typically measure how close the produced clustering is to predefined classes of the data. External cluster evaluation metrics are mainly used to choose the best clustering algorithm for a dataset (Liu et al., 2010)

One internal evaluation metric is the Silhouette of a clustering, first introduced by Rousseeuw (1987). According to Petrovic (2006), the time complexity of the Silhouette algorithm is quadratic in relation to the number of inputs.

2.3.2 Deep learning and neural networks

According to Alpaydin (2014), deep learning is a part of machine learning where neural networks are employed "to learn feature levels of increasing abstraction without human intervention". In a deep neural network there are usually several hidden layers in the network that can be trained to make the model better fit the data, hence the name "deep" neural networks. A type of neural network is the recurrent neural network, or RNN. (Alpaydin, 2014)

2.3.3 Reinforcement learning

Reinforcement learning is teaching an agent in an environment which actions to take in different situations, e.g. what to do, by interacting with the environment in order to maximise a numerical reward signal. Reinforcement learning is different from both supervised and unsupervised learning and is rather its own machine learning paradigm. The learning agent has a goal to maximise the cumulative reward of actions, which is formalised as the optimal control of incompletely-known Markov decision processes. (Sutton & Barto, 1998)

To maximise the reward, the agent must be able to sense the state of its environment to some extent as well as take actions that affect the state. According to Sutton and Barto (1998, p.1), "the learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them". An important reinforcement learning algorithm is Q-learning, a type of temporal difference learning method. (Sutton & Barto, 1998)

One of the challenges unique to reinforcement learning, compared to other kinds of learning, is the trade-off between exploration and exploitation. As previously mentioned, to maximise the reward, a learning agent must prefer to take actions that it has tried in the past and produced good reward, leading to exploitation of the experience it has gained. But to discover rewarding actions it has to try

actions that it has not yet tried, leading to exploration of new actions. Thus, there is always a trade-off between exploration and exploitation as neither can be pursued exclusively without failing the task at hand. (Sutton & Barto, 1998)

2.4 Program Synthesis

Program synthesis is considered by some to be the holy grail of computer science. In the words of Gulwani, Polozov and Singh (2017), "program synthesis is the task of automatically finding programs from the underlying programming language that satisfy user intent expressed in some form of constraints". Instead of translating fully specified code to low-level machine language, as compilers do, program synthesisers typically perform a search over the space of possible programs to generate a program that is consistent with the constraints. (Gulwani, Polozov & Singh, 2017)

One approach to program synthesis is deductive synthesis. It requires a complete formal specification of user intent to be provided which in many cases is as complicated as writing the program itself.

2.4.1 Inductive program synthesis

This lead to other approaches to program synthesis being developed, such as inductive synthesis based on input-output examples or demonstrations. More recent approaches to program synthesis also allow a grammar of the space of possible programs to be provided in addition to the specification. The search techniques used to find programs in the search space that satisfy the provided constraints can be based on enumerative search, deduction, constraint solving, statistical techniques or a combination of these. Statistical techniques may involve several types of machine learning, including deep learning. (Gulwani, Polozov and Singh, 2017)

2.4.2 DeepCoder

In 2016, Balog et al. presented an approach to program synthesis that synthesises programs from input-output examples while using deep learning to speed up the search. Their novel approach to inductive program synthesis (IPS), called Learning Inductive Program Synthesis (LIPS), uses a neural network to augment the previously used search techniques in IPS, leading to an order of magnitude speedup over baselines and RNN approaches. (Balog et al., 2016)

2.5 Relevance in Relation to Thesis

The material needed to properly understand this thesis and its problem and research question is grounded in research by prominent scientists from all over the world.

Clustering is not a new field of machine learning and thus most recent research has been focused on improving the performance of established clustering algorithms, as evident by the research conducted by Arthur and Vassilvitskii (2007) and Park and Jun (2009). These performance gains might allow for the application of clustering algorithms to a wider range of problems than previously possible, which will be investigated in this thesis.

To understand the significance of advancing research in (and the application of) program synthesis, I believe a quote from Balog et al. (2016) is in order: "A dream of artificial intelligence is to build systems that can write computer programs". This thesis hopes to contribute to a deeper understanding of how program synthesis can be applied to more complex problems than previously possible.

3 Methodology

In this section the choice of research strategy and methods that will be used in the thesis will be presented and motivated. A description of the chosen strategy, design science, is given followed by a discussion on ethical considerations as well as alternative research methods and strategies. Finally, how design science and its five-step method framework will be applied to the research question is presented.

3.1 Design Science

Design research is a kind of research which not only describes and explains the world, but also changes and improves it. According to Johannesson and Perjons (2014), design science is a strand of design research with origins in the areas of information systems and IT. It is a research methodology with the goal of solving practical problems of general interest by studying and creating artefacts. (Johannesson & Perjons, 2014)

Johannesson and Perjons (2014) provide a method framework consisting of five main activities to base design science research upon in which different data collection and analysis methods can be used in the different activities. The five main activities defined by Johannesson and Perjons (2014) will now be presented.

3.1.1 Explicate Problem

The Explicate Problem activity is where a practical problem is investigated and analysed. The problem needs to be precisely formulated and it should be clear how solving it will help improve some practice. The problem should be of general interest and underlying causes to the problem may be investigated.

3.1.2 Define Requirements

This activity defines a solution to the explicated problem in the form of an artefact and the requirements that the artefact needs to fulfil in order to solve the problem.

3.1.3 Design and Develop Artefact

In this activity an artefact is created that addresses the explicated problem and fulfils the defined requirements. There are four sub-activities: Imagine and Brainstorm, Asses and Select, Sketch and Build, and Justify and Reflect.

3.1.4 Demonstrate Artefact

The Demonstrate Artefact activity is where the developed artefact is used in “an illustrative or real-life case, sometimes called a proof of concept”, to prove that it solves an instance of the explicated problem.

3.1.5 Evaluate Artefact

Here, the developed artefact is evaluated to determine how well it fulfils the requirements and to what extent it can solve the practical problem that motivates the research.

3.2 Choice of Research Strategy and Methods

According to Denscombe (2010), a researcher needs to consider three key questions when deciding which research strategy to use: Is it suitable, is it feasible and is it ethical? To answer the question of suitability, we need to consider what the research project is trying to achieve – in the case of this thesis, the research question that needs to be answered is how nodes in a distributed system can be clustered according to context and leader election be done by inductive program synthesis. This will be done by creating an artefact that has to be evaluated.

In his book, Denscombe (2010) also presents a checklist to help researchers choose an appropriate research strategy. When it comes to suitability, the following things should be considered:

- Has the purpose of the research been clearly identified?
- Is there a clear link between the purpose of the research and the chosen strategy?
- Will the strategy produce findings that can answer the research question(s)?

The purpose of the research is to organise and improve the controllability of distributed systems by using a novel approach that will also potentially decrease the amount of work needed to develop such systems. Therefore, the problem is of a practical nature, although theoretical foundations are of course needed to make it possible to achieve the previously mentioned improvements.

That there is a clear link between the research purpose and design science is evident since the problem to be solved is a practical one. By developing an artefact, the problem can be solved and hopefully more insight has been gained into how clustering can be used to organise distributed systems and how program synthesis can be used to solve more complex problems.

That design science will produce good findings can be argued quite easily as design science research produces some kind of artefact. The artefact should solve a practical problem of general interest which will also be used to answer the research question.

When it comes to feasibility, Denscombe (2010) also lists the following to be considered:

- Is there sufficient time for the design of the research, collection of data and analysis of results?
- Are sufficient resources available to cover the costs of the research (e.g. travel, printing)?
- Is it possible and practical to gain access to necessary data (people, events, documents)? Will the chosen strategy be favoured by the key evaluators of the research?

Design science research projects can be large and time-consuming, but also small-scale research projects involving only a single researcher (Johannesson & Perjons, 2014). I am the sole author of this thesis and thus the work conducted is a small-scale research project. Given that an artefact will be developed independently of other people to solve the problem and answer the research question, I argue that it is feasible to use design science as the research strategy. The motivation for this argument is that an artefact will be developed that does not rely on other researchers nor participants and that no expenses, such as travel or printing costs, are expected. Whether there is sufficient time for the project or not is hard to estimate, but I argue that design science is an appropriate choice as it is suitable for small-scale research projects and allows for flexibility when choosing data collection and analysis methods within its method framework.

Finally, when deciding whether it is ethical to choose a certain research strategy, Denscombe (2010) lists the following to consider:

- Can I avoid any harm to participants resulting from their involvement in the research?
- Can I get informed consent from potential participants?
- Will the strategy permit me to work within an appropriate code of research ethics?
- Can I guarantee the confidentiality of the information given to me during the research?

Interaction with people in terms of questionnaires or other methods of data collection that involve collecting data from participants will not be done in this thesis. Thus, there is no need to consider harm to the participants or loss of confidentiality of the data collected, as an artefact will be developed independently of any outside participants. Data will be collected by testing the artefact and analysis of quantitative data will be done by presenting descriptive statistics. Descriptive statistics are statistics that describe the data, for example, measures of central tendency and dispersion (Denscombe, 2010).

After considering all of the above, the research strategy design science is deemed to be the most appropriate strategy for the work to be undertaken in this thesis.

3.2.1 Alternative research methods and strategies

Grounded theory could have been chosen as an alternative research strategy. To quote Denscombe (2010), "it is an approach dedicated to generating theories". It emphasises the importance of empirical fieldwork and linking explanations very closely to what happens in practical situations. It is well suited to small-scale research projects as well as exploratory research. (Denscombe, 2010)

As the approach that this thesis takes to solving the research problem is quite new and innovative I argue that it could be viewed as exploratory research and thus grounded theory could be well suited to use. However, design science was chosen because it explicitly facilitates the creation of an artefact by providing a proven method framework to develop it with.

As several different data collection and analysis methods will be used in the different activities within the design science method framework, alternative research methods are not discussed here. Instead, if applicable, the methods deemed most suitable for each activity have been chosen and motivated in the application of that activity below in Section 3.3.

3.3 Application of Method

Design science and the five activities of the method framework proposed by Johannesson and Perjons (2014) will be applied to the problem described in Section 1 in order to answer the research question. An overview of the method framework is presented below in Figure 1.

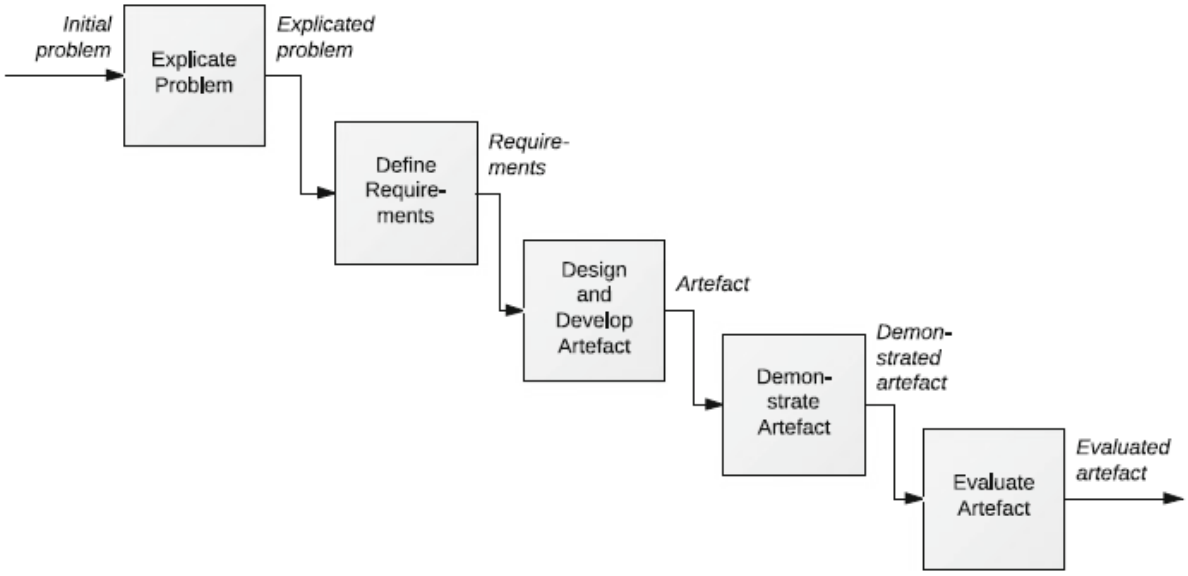

Figure 1 - Overview of the method framework for design science research (Johannesson & Perjons, 2014)

3.3.1 Explicate Problem

In the Explicate Problem activity, the problem that is defined in the introduction in Section 1 will be further explicated. By conducting a literature study, previous solutions will be investigated to gain a better understanding of how the artefact can be developed. Literature will be chosen based on the practical usefulness of the literature and how relevant it is deemed to be.

3.3.2 Define Requirements

After the problem has been properly explicated, the requirements of the artefact will be defined in the Define Requirements activity. Previous solutions to similar problems will be studied and compared to the problem that is studied in this thesis in order to understand what the requirements must be. The type of artefact must also be chosen. There are four different types of artefacts according to Johannesson and Perjons (2014), namely: constructs, models, methods, and instantiations.

3.3.3 Design and Develop Artefact

In this activity, the artefact will be developed by creating Python programs which should fulfil the requirements defined in the previous activity. The four sub-activities of the Design and Develop Artefact activity, as defined by Johannesson and Perjons (2014), will be used while designing and developing the artefact. Existing machine learning algorithms already implemented in Python will be used, if possible, to decrease the amount of work needed to develop the artefact. Its design will be described and motivations for certain design choices will be presented.

3.3.4 Demonstrate Artefact

In the Demonstrate Artefact activity the previously developed artefact will be tested in different scenarios to see whether it can create groups of nodes in a simulated distributed system based on context and elect a leader for each group. Data will be collected during these scenarios to evaluate the artefact at a later stage. Thus, it is demonstrated whether the artefact can solve the clustering and

leader election problems that motivate the research conducted in this thesis or not. Plots will be created with Matplotlib, which is a 2D graphics package for Python (Hunter, 2007).

3.3.5 Evaluate Artefact

In the final activity, the created artefact will be evaluated by checking how well it fulfils the requirements and to what extent it can solve the problem that motivates the research. Quantitative data collected during the demonstration of the artefact, such as performance and cluster evaluation measurements, will be used to evaluate the artefact. Qualitative data, such as clustering plots and whether the synthesised program is consistent with the input-output examples, will also be used. The data will be analysed by using SciPy, a Python library for scientific computing, while visualisations of the data will be created with Matplotlib. The analysis that will be performed is descriptive statistical analysis such as calculating the mean of the results, the standard deviation, the margin of error and the corresponding confidence interval. After evaluating the artefact, it should be possible to answer the research question.

4 Problem and Requirements

In this section the problem that motivates the research conducted in this thesis is looked at more deeply in accordance with the method framework described in Section 3.3.1. Then, requirements on the artefact to be developed are defined based on the findings of the Explicate Problem phase.

4.1 Explicated Problem

When conducting the literature study, no previous solution to the exact problem that is to be solved in the thesis was found, but previous solutions to the individual problems that need to be solved were. However, no solutions were very similar to the one proposed in the thesis, likely because of the unique combination of problems and their respective solutions. Clustering is a mature area of machine learning and there are many implementations of possible algorithms to use, but I was not able to find previous research that applies clustering to bring order to a distributed system and then elect leaders to the created groups. There is, however, other work on bringing order to distributed systems but there the focus has been on more stringently proving or verifying the controllability of the system, such as the work done by Guo, Yan and Lin (2012) as well as Yu, Zhang and Sun (2010).

No single Python library implemented all the algorithms that were found to be necessary, so a combination of several must be used. There were a couple of DeepCoder implementations publicly available but only one that was written entirely in Python. In-depth motivations for design choices are presented later in Section 5.1.

4.2 Requirements

Here, the requirements on the artefact are presented. The type of artefact that was chosen was an instantiation, because the system that was to be developed should be a working system and programs realising several algorithms were to be implemented.

As the artefact is to be used in decentralised distributed systems there are certain constraints that must be adhered to. Primarily, the resources available to the system running the artefact are most likely limited and thus the artefact must be able to produce results within acceptable time limits even on slow processors. Efficiency is therefore extremely important, both in terms of memory usage and speed of execution. Achieving good efficiency is also important because it reduces power usage, which may or may not be a limiting factor for these systems.

The functional requirements on the artefact are:

- Produce the best clustering for a certain set of nodes based on their context by evaluating the clustering results and choosing the best one available
- Be able to synthesise a program that implements a simple leader election algorithm

The non-functional requirements on the artefact are:

- The entire artefact must be efficient enough for implementation on limited devices, both in terms of memory usage and speed of execution

- Leader election must be fast enough to enable near real-time control of distributed systems

5 Results and Analysis

In this section the results of the thesis will be presented. First, how the artefact was designed and developed is laid out followed by a demonstration of the artefact by testing it in different scenarios. Finally, the artefact is evaluated, and the results are analysed.

5.1 Design and Development of the Artefact

The entire artefact was written in Python 3[1], including data generation, formatting and analysis. The code for the entire artefact can be found on GitHub[2]. There are three main conceptual stages of the artefact, presented in Figure 2. The output of each stage is used as the input to the next stage.

The clustering stage involves clustering the data that represents a simulated distributed system and evaluating and comparing the produced clustering to other clustering results to obtain the best result possible. The ranking and leader election stage involves ranking the nodes in the simulated distributed system, which is done by ranking them according to their distance to the medoid of the best clustering produced in the clustering stage. Then, simple leader election functions are performed using the ranks as input. In the last stage, program synthesis, the ranks of the nodes that were produced in the previous stage and their respective leaders are used as input-output examples to synthesise a program that is consistent with the provided examples.

In the sub-activity Asses and Select, different machine learning techniques were investigated for use in the artefact. At one point, reinforcement learning was considered for the program synthesis stage of the artefact. However, it was decided to use DeepCoder's default implementation of program synthesis because of time constraints as well as concerns whether reinforcement learning was appropriate for the problem to be solved. Something that also contributed to this decision was that the reinforcement learning algorithms that could be used for this purpose were deemed to be more complicated than DeepCoder's default implementation. An in-depth explanation of the design of each stage of the artefact now follows.

[1] https://www.python.org/downloads/

[2] https://github.com/CarsonBeckett/deepcoder-le

Figure 2 - Conceptual stages of the artefact

5.1.1 Clustering

Clustering was done as a pre-processing step to generate clusters based on context values for nodes in a dataset, simulating a distributed system. The algorithm used for clustering is the k-medoids algorithm as implemented in Python 3 by the pyclustering library[3] (which in turn is an implementation of PAM), while the k-means++ implementation in the pyclustering library was used to choose the initial points. The k-medoids algorithm was used because it is more robust to outlier than k-means and because using an actual data point (medoid) as the representative object of a cluster facilitates leader election in later stages. There is also no risk that the choice of initial points will produce incorrect clusters, even though that risk would have been eliminated by the use of k-means++ as an initialiser in any case. K-means++ was used as an initialiser to ensure a fair distribution of the initial points and because it can improve the speed and accuracy of the clustering according to Arthur and Vassilvitskii (2007).

To determine the best clustering for a certain dataset, the k-medoids algorithm was run with different values of *k* and each produced clustering was evaluated based on its average Silhouette value. The average Silhouette value was calculated by a function called sklearn.metrics.**silhouette_score** from the scikit-learn module[4] in Python. Scikit-learn is a machine learning module "for medium-scale supervised and unsupervised problems" (Pedregosa, 2011, p.2825).

Clustering was done on a two-dimensional dataset of simulated context values, although it is fully possible to perform clustering on a high-dimensional dataset. There are even algorithms such as subspace clustering that seek to find clusters in different subspaces within a dataset (Parsons, Haque & Liu, 2004). However, I chose to instead put the complexity in the generation of context values that may be based on an aggregation of many different values, which is outside the scope of this thesis. The motivation for this decision is that clustering high-dimensional data is very computationally expensive and would make it harder to meet the performance requirements imposed on the artefact.

The clusters were then passed to the next stage for ranking and leader election to be performed.

---

[3] *Pyclustering library: https://github.com/annoviko/pyclustering*

[4] *scikit-learn library: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn-metrics-silhouette-score*

5.1.2 Ranking and leader election

The leader election algorithm that was to be implemented in this thesis is a variant of the monarchical leader election algorithm (Algorithm 2.6) in Cachin, Guerraoui and Rodrigues (2011), depicted in Figure 3. This leader election algorithm requires each process to have a rank associated with it, which was done in a ranking function that assigns a rank to each node based on its distance to the representative object of its cluster, the medoid. The distance metric that was used was the Manhattan distance. The ranks were then passed to the leader election function which elects the highest ranking node as the leader of each cluster, for use as an oracle later on. The ranks were then passed to DeepCoder in the next stage as the input examples and their medoids, or leaders, as the output examples, thus generating the input-output examples needed for DeepCoder to synthesise a program.

The algorithm defined in Figure 3 only allows election of a new leader if the current leader has crashed, meaning it must be modified to allow election of a new leader in other circumstances as well, for example when new nodes join the system. This was not done in this thesis because the system is simulated and a perfect failure detector is assumed[5], but would have to be done in a real system.

Algorithm 2.6: Monarchical Leader Election

Implements:
LeaderElection, **instance** *le*.

Uses:
PerfectFailureDetector, **instance** $\mathcal{P}$.

upon event $\langle\ le, Init\ \rangle$ **do**
$suspected := \emptyset$;
$leader := \perp$;

upon event $\langle\ \mathcal{P}, Crash \mid p\ \rangle$ **do**
$suspected := suspected \cup \{p\}$;

upon $leader \neq maxrank(\Pi \setminus suspected)$ **do**
$leader := maxrank(\Pi \setminus suspected)$;
trigger $\langle\ le, Leader \mid leader\ \rangle$;

Figure 3 - Monarchical Leader Election Algorithm (Cachin, Guerraoui & Rodrigues, 2011)

5.1.3 Program synthesis

In the final stage of the artefact, a program that is consistent with the input-output examples produced by the previous stages must be synthesised, or in other words, a hierarchical leader election program must be synthesised. This was done by using DeepCoder's LIPS framework.

No reference implementation of DeepCoder seemed to be available, but several third-party implementations could be found on GitHub. Since Python is the language used in this thesis, I chose to use the Python 3 implementation of DeepCoder by GitHub user "dkamm"[6]. It of course required

[5] *See chapter 2.6.1 on page 48 of Cachin, Guerraoui and Rodrigues (2011) for an explanation of what a perfect failure detector is*

[6] *Deepcoder pure Python implementation: https://github.com/dkamm/deepcoder*

Python 3 as well as machine learning and mathematical libraries such as Keras[7], Tensorflow[8], numpy[9] and pandas[10]. Keras is used as a high-level interface that runs on a Tensorflow backend to facilitate the neural network while numpy and pandas are widely used Python libraries for scientific computing and data manipulation and analysis.

[7] https://keras.io/

[8] https://www.tensorflow.org/

[9] http://www.numpy.org/

[10] https://pandas.pydata.org/

5.2 Demonstration of the Artefact

To demonstrate the artefact and verify that it can perform its intended functions it will be tested in different scenarios described later in this section. All five clustering scenarios as well as the program synthesis based on all scenarios were executed exactly 30 times to generate sufficient data to ensure that the results were repeatable and reliable.

Testing was done on an x86-64 architecture Intel Core i5-7600K processor with 16GB of DDR4 DRAM running Windows 10 Education version 10.0.17134 Build 17134 and Python 3.6.5 64-bit. Libraries that the artefact depends on are: scikit-learn 0.19.1; pyclustering 0.8.0; deepcoder 0.0.1 (dkamm version) and their nested dependencies.

All datasets were randomly generated by using the Python function random.**uniform**()[11] or manually typed in at random. Some of the datasets were manually typed in to create natural clusters in the data to illustrate a more realistic scenario, rather than just a truly random uniform distribution. The randomly generated datasets are first and foremost used to test the scalability of the artefact.

These datasets represent the simulated context information for different nodes in a distributed system as floating-point numbers. They contain no duplicates but are not necessarily disjoint from each other as some parts of the smaller datasets are also subsets of the larger datasets (Scenario 2 includes some of the data in Scenario 1, Scenario 3 includes some of the data in Scenario 2 and Scenario 5 includes all the data from Scenario 4).

All performance measurements were taken with the Python 3 built-in function time.**perf_counter**()[12] because it uses the highest available clock resolution. For comparison the elapsed real time (commonly called wall time) was also measured with the function time.**time**()[13] for the same tasks. All plots were created with Matplotlib.

To get the best clustering available for each scenario, clustering was run for k = 2 to 10, e.g. 9 times. The Silhouette analysis of all values of k for all scenarios can be found in Appendix A but were not included in the results because they are simply a way of obtaining the best clustering available for a certain scenario.

Throughout the execution of all the scenarios the memory usage was consistently between 175MB to 235MB.

[11] https://docs.python.org/3/library/random.html#random.uniform

[12] https://docs.python.org/3/library/time.html#time.perf_counter

[13] https://docs.python.org/3/library/time.html#time.time

5.2.1 Scenario 1

Description: The first clustering scenario to be tested with the developed artefact is a scenario where 29 nodes are present in a distributed system. The dataset was manually typed in at random. A situation that could correspond to a real-world example of this scenario is that one or several of the previous leaders have crashed or experienced link failures. Subsequently, new leaders must be elected.

Result: The best clustering that was found for Scenario 1 had an average Silhouette of 0.8485671038005586 and was achieved with k=2. The best available clustering for Scenario 1 is thus two clusters as shown below in Figure 4. The medoids of the different clusters are easy to distinguish from regular members as they are depicted as stars. In this thesis, the medoids are also the leaders that are to be elected later on. Performance measurements for Scenario 1 are presented in Table 1.

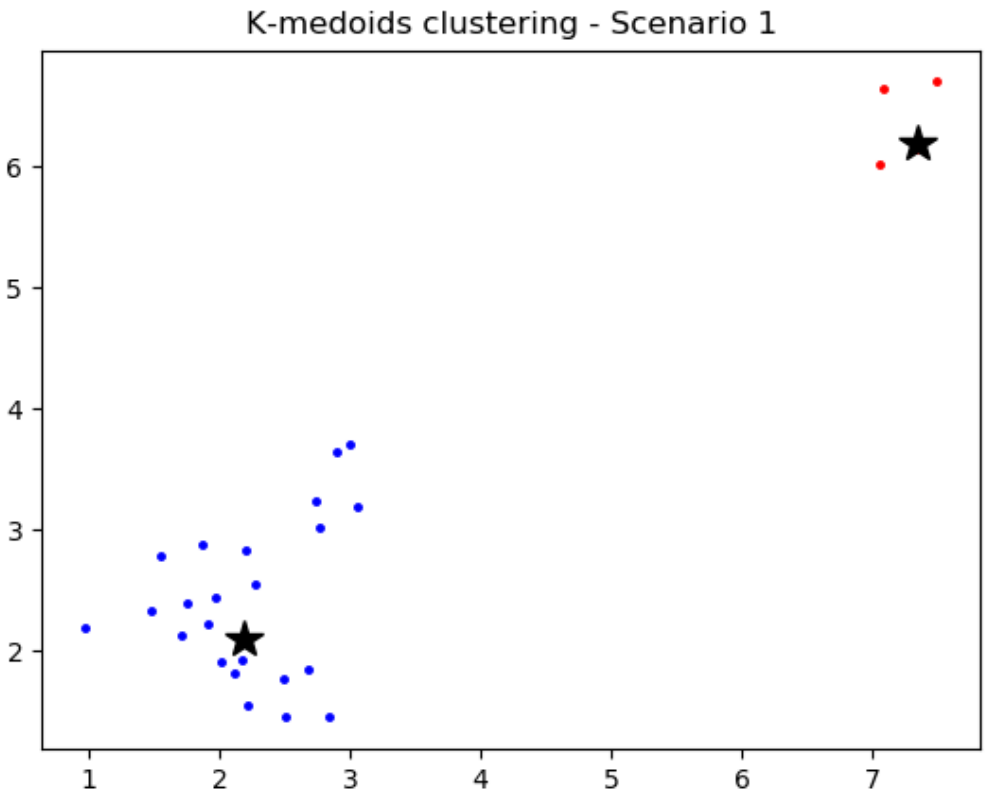

Figure 4 – Best clustering for Scenario 1
Silhouette of 0.8485671038005586 and k=2

Table 1 - Performance measurements for Scenario 1

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Minimum	0.909ms	0.996ms	175.649ms	175.53ms
Maximum	1.547ms	0.998ms	199.329ms	199.466ms
Mean	0.959ms	0.997ms	184.651ms	184.639ms

5.2.2 Scenario 2

Description: In the second clustering scenario five new nodes are added to the system, increasing the total number of nodes to 34. The dataset was manually typed in at random. A situation that could be a real-world example of this scenario is that new nodes have been added to the system and the nodes must be clustered according to context for the system to assign a leader to each group of nodes.

Result: The best available clustering for Scenario 2 is three clusters, as the best clustering found had an average Silhouette of 0.5863200762086528 and was achieved with k=3. The best clustering for Scenario 2 is shown below in Figure 5. Performance measurements for Scenario 2 are presented in Table 2.

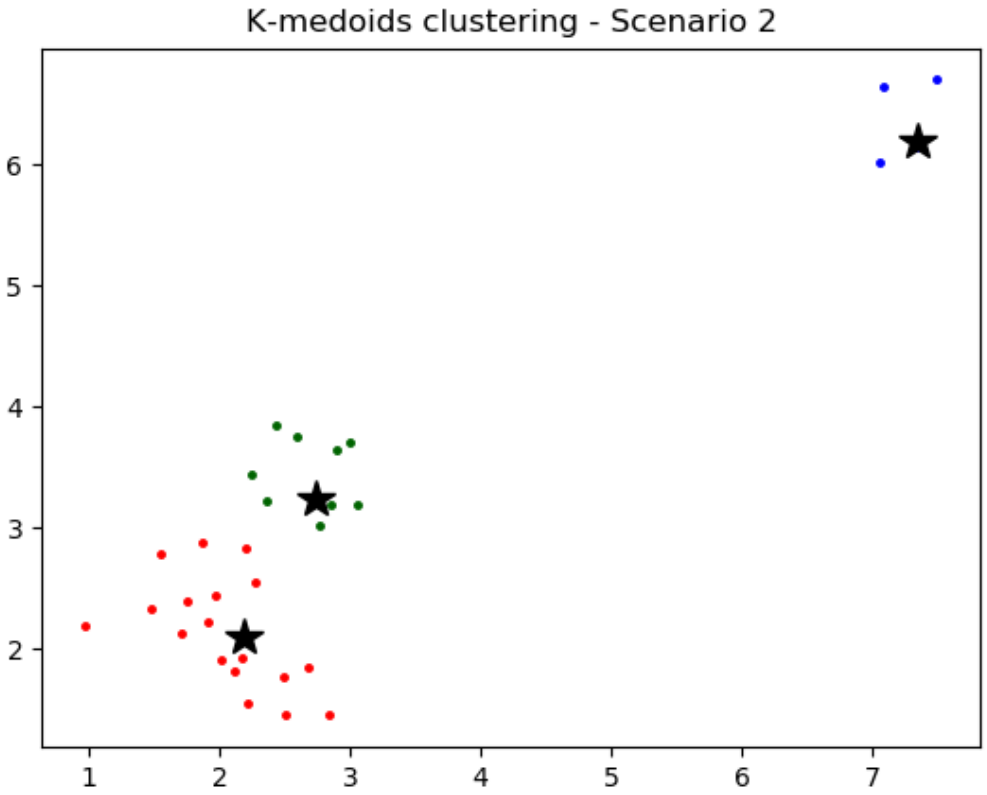

Figure 5 - Best clustering for Scenario 2
Silhouette of 0.5863200762086528 and k=3

Table 2 - Performance measurements for Scenario 2

Labels	*Best clustering run time*	*Best clustering wall time*	*Total scenario run time*	*Total scenario wall time*
Minimum	0.816ms	0.0ms	174.73ms	174.533ms
Maximum	0.884ms	0.997ms	204.086ms	204.452ms
Mean	0.833ms	0.831ms	186.921ms	186.9ms

5.2.3 Scenario 3

Description: The third clustering scenario is meant to test the scalability of the artefact by adding 12 more nodes to the system. The dataset for Scenario 3 was also manually typed in at random.

Result: For Scenario 3 the best clustering had an average Silhouette of 0.8332508360470124 and was achieved with k=2. Thus, the best available clustering for Scenario 3 is two clusters as shown in Figure 6 below. Performance measurements for Scenario 3 are presented in Table 3.

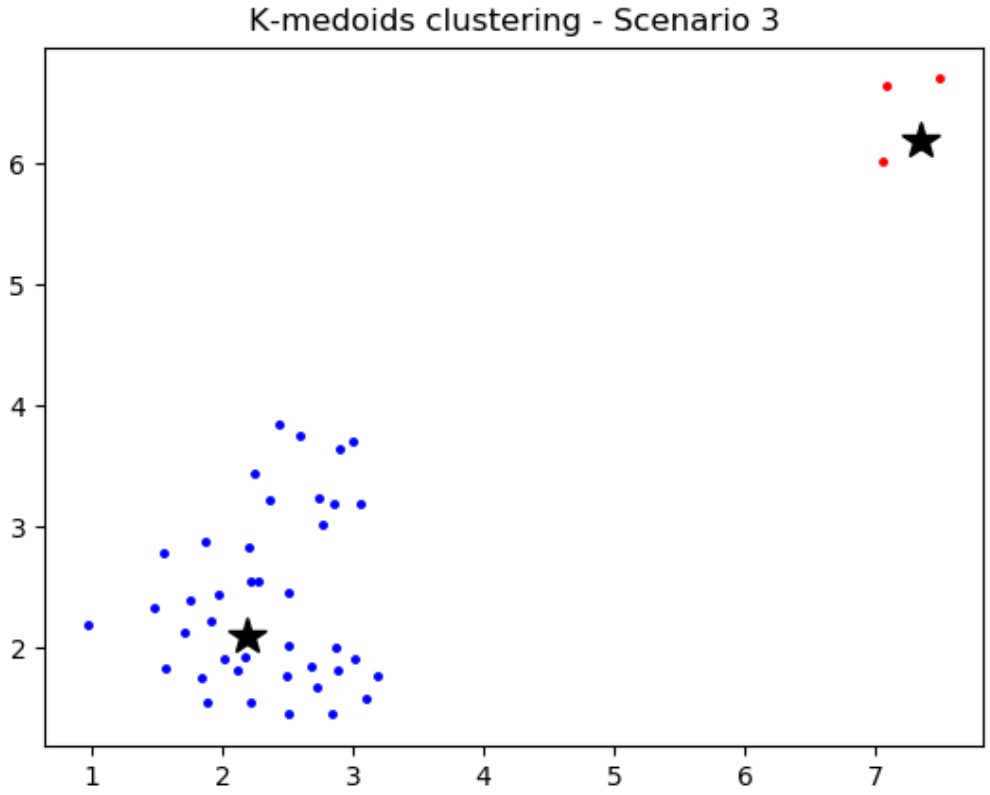

Figure 6 - Best clustering for Scenario 3
Silhouette of 0.8332508360470124 and k=2

Table 3 - Performance measurements for Scenario 3

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Minimum	2.13ms	1.993ms	181.577ms	181.514ms
Maximum	2.78ms	2.992ms	210.025ms	209.439ms
Mean	2.275ms	2.326ms	191.848ms	191.787ms

5.2.4 Scenario 4

Description: The fourth clustering scenario operates on a randomly generated dataset of 200 nodes in order to further test its generalisation ability and scalability.

Result: The best available clustering for Scenario 4 is four clusters as shown below in Figure 7, as the best Silhouette value of 0.3830742916987218 was achieved with k=4. Performance measurements for Scenario 4 are presented in Table 4.

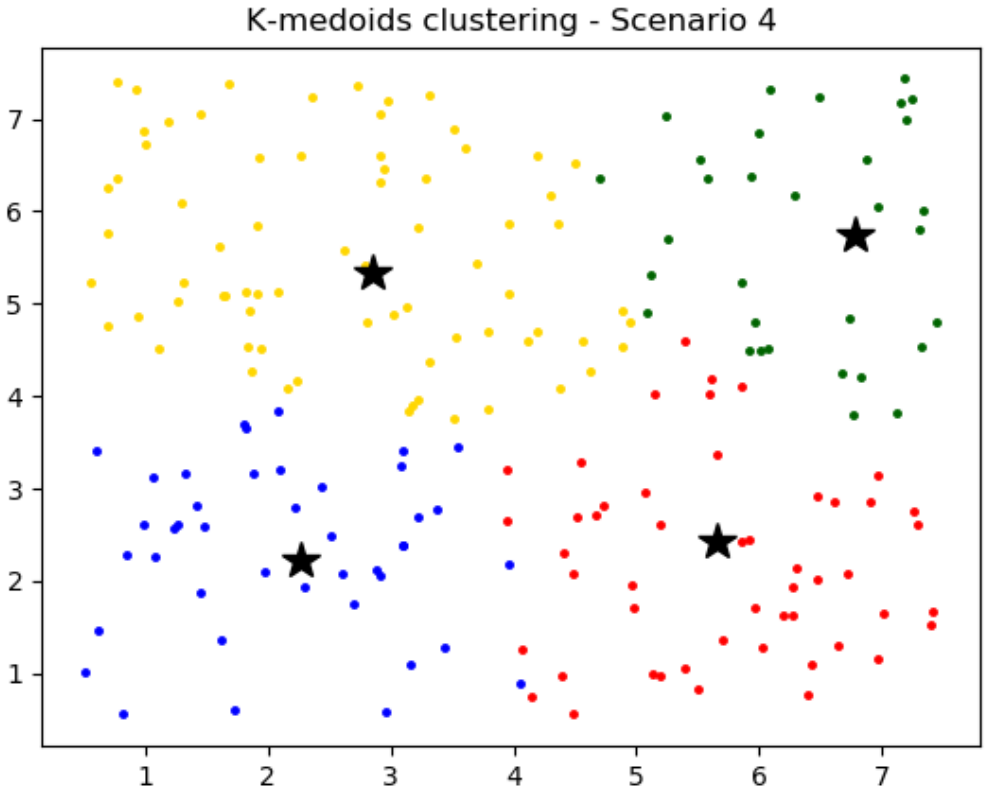

Figure 7 - Best clustering for Scenario 4
Silhouette of 0.3830742916987218 and k=4

Table 4 - Performance measurements for Scenario 4

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Minimum	12.329ms	11.967ms	280.336ms	280.25ms
Maximum	47.76ms	47.871ms	316.7ms	316.154ms
Mean	13.732ms	13.862ms	290.743ms	290.722ms

5.2.5 Scenario 5

Description: The fifth and final clustering scenario aims to further test the scalability of the system by adding 800 nodes to the system, for a total of 1000 nodes. The dataset was generated the same way as in Scenario 4.

Result: The Silhouette of the best clustering for Scenario 5 was 0.363725300745691 which was achieved with k=7. The best available clustering for Scenario 5 is seven clusters as shown by Figure 8 below. Performance measurements for Scenario 5 are presented in Table 5.

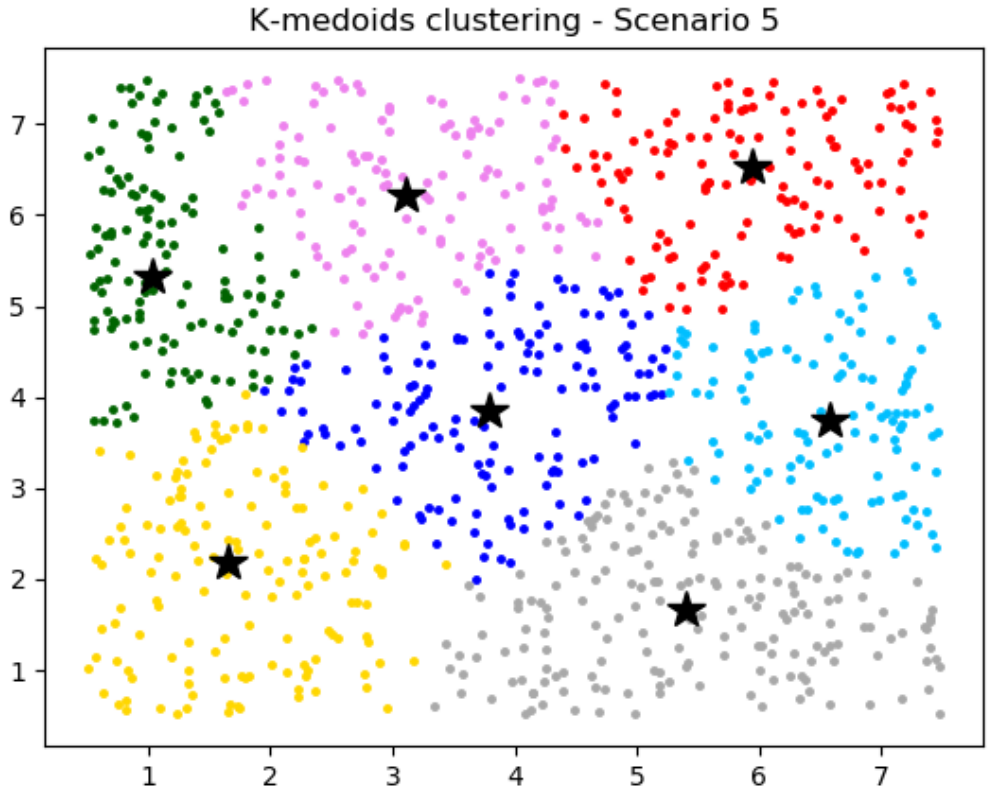

Figure 8 - Best clustering for Scenario 5
Silhouette of 0.363725300745691 and k=7

Table 5 - Performance measurements for Scenario 5

Labels	Best clustering run time	Best clustering wall time	Total scenario run time	Total scenario wall time
Minimum	158.421ms	158.575ms	2582.879ms	2582.093ms
Maximum	194.582ms	194.479ms	2701.757ms	2701.772ms
Mean	160.64ms	160.703ms	2627.132ms	2627.239ms

5.2.6 Results of IPS and leader election

Description: After all the clustering scenarios had finished running, the best clustering was used as the representative measurement of that scenario. All the representative clusters were then passed to DeepCoder for program synthesis. The input-output examples for DeepCoder were generated by ranking all the nodes in each cluster according to the distance to the medoid of the cluster. The medoid had the highest rank, the object closest to the medoid the next highest rank, and so on. I now present one of the main results of the thesis: whether inductive program synthesis is capable of synthesising a program consistent with the input-output examples for a leader election algorithm, and if so, how.

Results: It is clear by looking at Table 6 that DeepCoder is in fact able to synthesise a correct and consistent leader election program for the simple hierarchical leader election algorithm used in this thesis. Both DFS and sort and add search techniques produced the same program, leading to increased confidence in the results. The synthesised program is of the format INPUT|FUNCTION,ARGUMENT meaning that a list is taken as the input to the program which consists of the function MAXIMUM with variable 0 as the argument. As there is only one variable present in this program, the input, this is argument variable 0, and the function is applied to the input list. The steps used is how many steps the search technique had to take before finding a consistent program. In Table 7, the performance measurements for synthesising a consistent program for the two different search techniques is presented.

Table 6 - Results for IPS

Labels	***Synthesised program***	***Program consistent***	***Steps used***
DFS	LIST|MAXIMUM,0	Yes	5
Sort and add	LIST|MAXIMUM,0	Yes	[2, 3, 4, 5]

Table 7 - Performance measurements for IPS

Labels	***DFS run time***	***DFS wall time***	***Sort and add run time***	***Sort and add wall time***
Minimum	0.874ms	0.0ms	1.263ms	0.997ms
Maximum	0.964ms	0.997ms	1.442ms	1.995ms
Mean	0.888ms	0.764ms	1.315ms	1.462ms

5.3 Evaluation of the Artefact

The artefact was evaluated to determine how well it fulfilled the requirements – several qualitative and quantitative measurements were used to evaluate the efficiency and correctness of the artefact. The output data generated by the scenarios from the Demonstrate Artefact activity in Section 5.2 was the basis for this evaluation.

The following measurements were used to evaluate the clustering program: the execution time of the clustering function; the Silhouette value for best number of *k*; and the total execution time of the entire scenario.

The following measurements were used to evaluate the IPS program: the execution time of the IPS program that generates the leader election program; and the correctness of the leader election program, e.g. whether the node with the highest rank was always elected as the new leader.

The leader election program outside of DeepCoder was not evaluated as it only produced the input-output examples for the IPS program. The data generation, output formatting and analysis programs were also not evaluated as they provided only auxiliary functions to the artefact that must be executed before any other programs can run.

Statistics presented are descriptive statistics. All presented confidence intervals have a confidence level of 95%, or in other words, they are 95% confidence intervals.

5.3.1 Evaluation and analysis of clustering scenarios

Table 8 shows the statistical analysis performed on the measurements for Scenario 1 that were presented in Table 1. The standard deviation is not too high compared to the measurement data but still high enough to indicate that some of the measurements are spread out a bit away from the mean, something that is supported by the confidence interval and margin of error. Still, the results seem to be consistent. The same is true for the total run time of the scenario, as also shown in Table 8. The results for Scenarios 2-5 are similar and are presented in Tables 9-12.

A comparison of the average run time of the best clustering and the average total run time of all the scenarios is shown in Figures 9-13. It is evident that the best clustering is a quite small part of the total scenario run time. This is probably the case because the Silhouette calculations are run 9 times for each scenario and the time complexity of the Silhouette algorithm is quadratic in relation to the number of inputs according to Petrovic (2006). Thus, the higher the value of *k* (the number of clusters), the longer it takes to calculate the Silhouette of the clustering. Also, the presented clustering time is only the run time of the best clustering for each scenario, even though clustering was performed nine times for every scenario, once for each different value of *k*. If one were to multiply the average run time of the best clustering found by nine in every scenario, it would be shown that the clustering takes up a significantly larger portion of the total run time of each scenario than the Silhouette calculations. For example, looking at the comparison of Scenario 5 as presented in Figure 13, clustering would take up more than half of the total run time of the scenario.

Also, by comparing the portion of the total run time that is taken up by the best clustering in Figures 9-13, it can be seen that the clustering takes up a progressively larger portion of the total run time. Following this observation, it is reasonable to conclude that the time complexity of k-medoids clustering must be higher than linear, something that is not necessarily in conflict with the Silhouette calculations having a quadratic time complexity. Both can be polynomial in their time complexity and still produce the results depicted in Figures 9-13, provided they take up significantly different absolute

time, e.g. clustering must take up significantly more time than the Silhouette calculations, even if their time complexity is similar.

Because the results seem to be consistent and the artefact is able to choose the best available clustering for each scenario within a reasonable amount of time relative to input sizes, I argue that the artefact fulfils the requirements related to clustering.

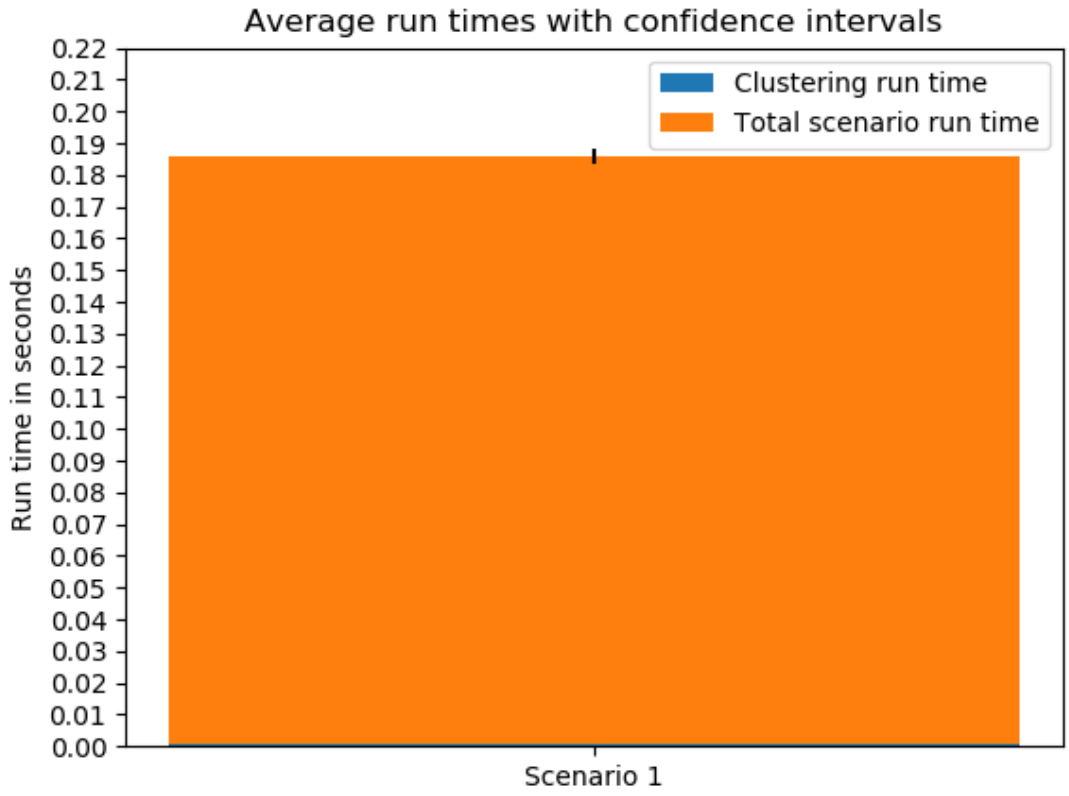

Figure 9 - Plot of analysis run time of Scenario 1

Table 8 - Statistical analysis of Scenario 1 (in seconds)

Labels	***Best clustering run time***	***Best clustering wall time***	***Total run time***	***Total wall time***
Standard deviation	0.00011462	0.00000031	0.00626657	0.00633260
Confidence interval	(0.00091601, 0.00100295)	(0.00099719, 0.00099743)	(0.18227483, 0.18702791)	(0.18223797, 0.18704113)
Margin of error	0.00004347	0.00000012	0.00237653	0.00240157

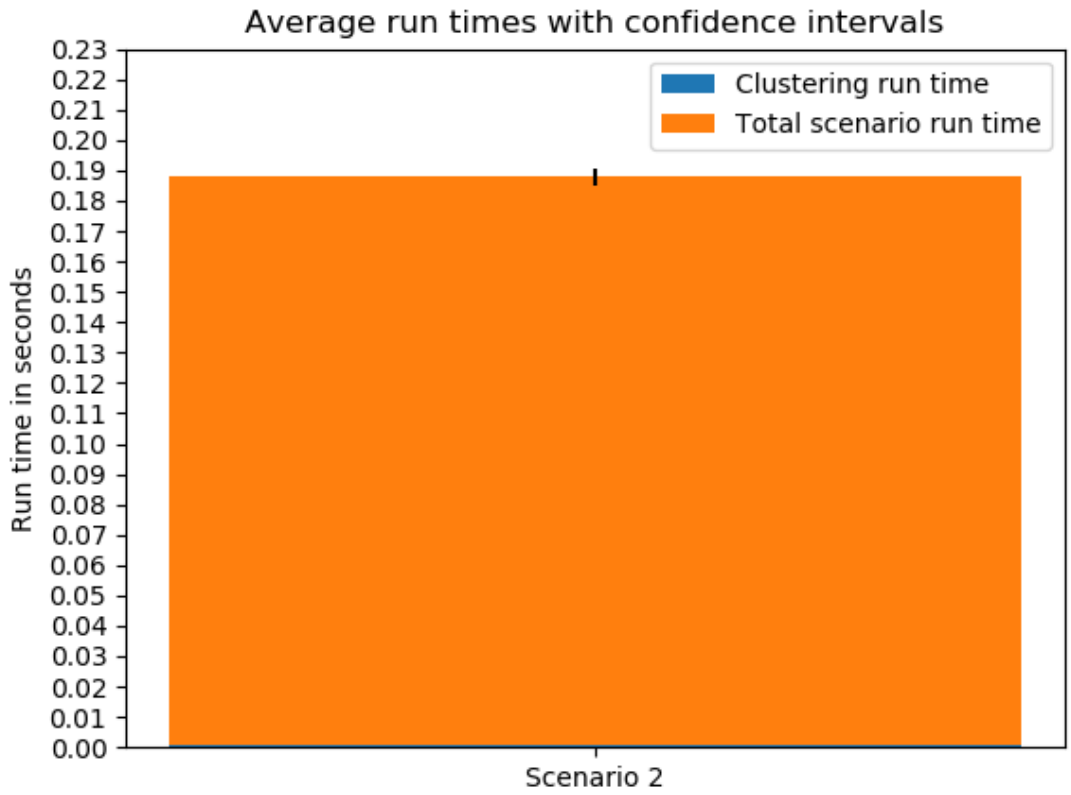

Figure 10 - Plot of analysis of run time of Scenario 2

Table 9 - Statistical analysis of Scenario 2 (in seconds)

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Standard deviation	0.00001914	0.00037169	0.00720935	0.00722680
Confidence interval	(0.00082624, 0.00084076)	(0.00069017, 0.00097209)	(0.18418748, 0.18965564)	(0.18415931, 0.18964070)
Margin of error	0.00000726	0.00014096	0.00273407	0.00274069

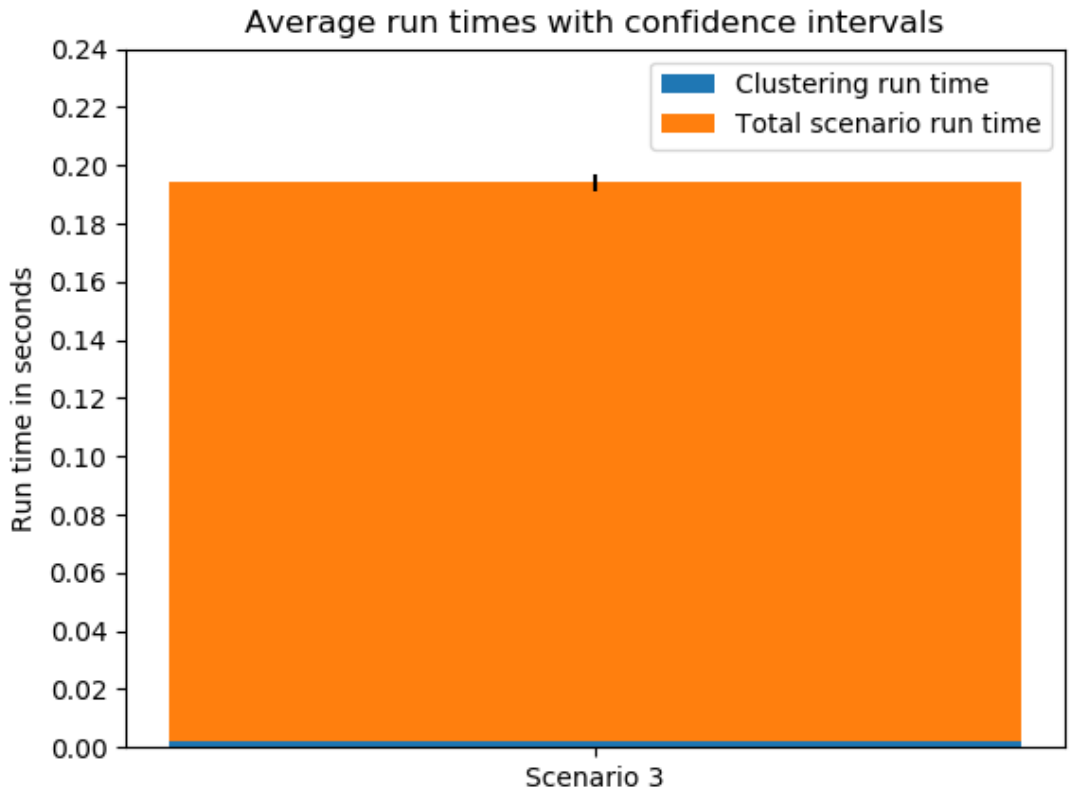

Figure 11 - Plot of analysis of run time of Scenario 3

Table 10 - Statistical analysis of Scenario 3 (in seconds)

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Standard deviation	0.00016005	0.00047012	0.00744485	0.00725222
Confidence interval	(0.00221483, 0.00233623)	(0.00214857, 0.00250515)	(0.18902536, 0.19467214)	(0.18903668, 0.19453735)
Margin of error	0.00006069	0.00017829	0.00282339	0.00275033

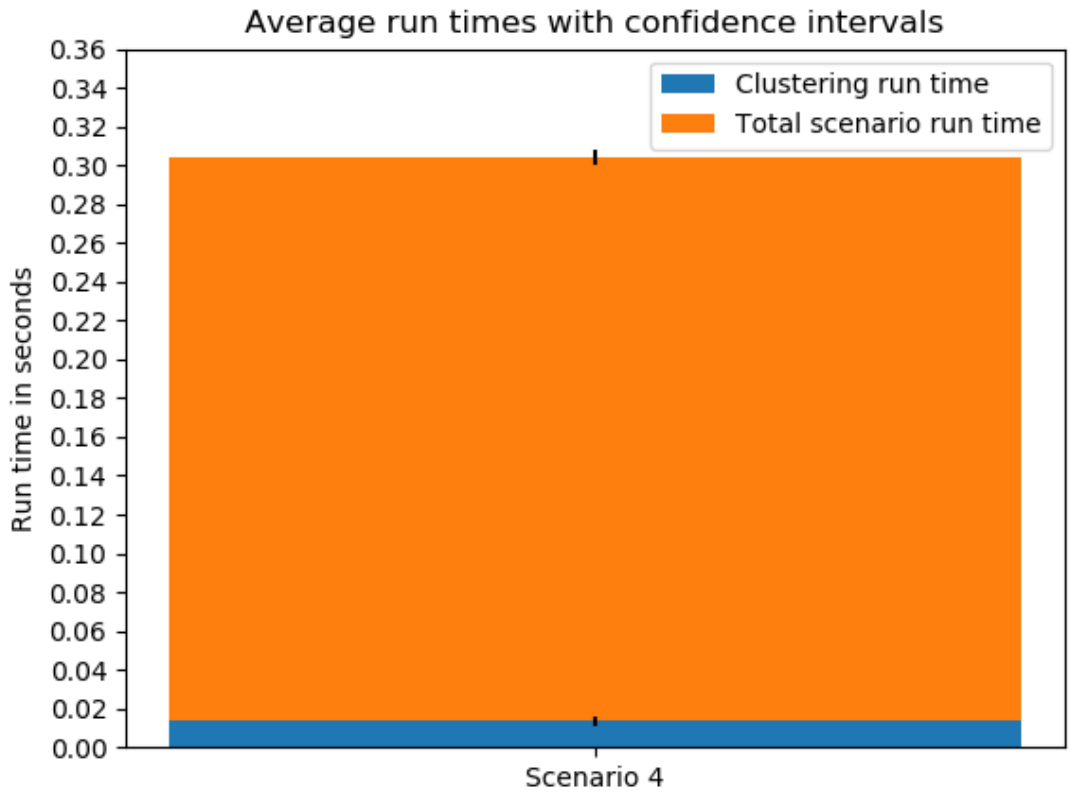

Figure 12 - Plot of analysis of run time of Scenario 4

Table 11 - Statistical analysis of Scenario 4 (in seconds)

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Standard deviation	0.00632203	0.00633048	0.01003863	0.01002875
Confidence interval	(0.01133527, 0.01613041)	(0.01146190, 0.01626345)	(0.28693667, 0.29455078)	(0.28691904, 0.29452566)
Margin of error	0.00239757	0.00240077	0.00380705	0.00380331

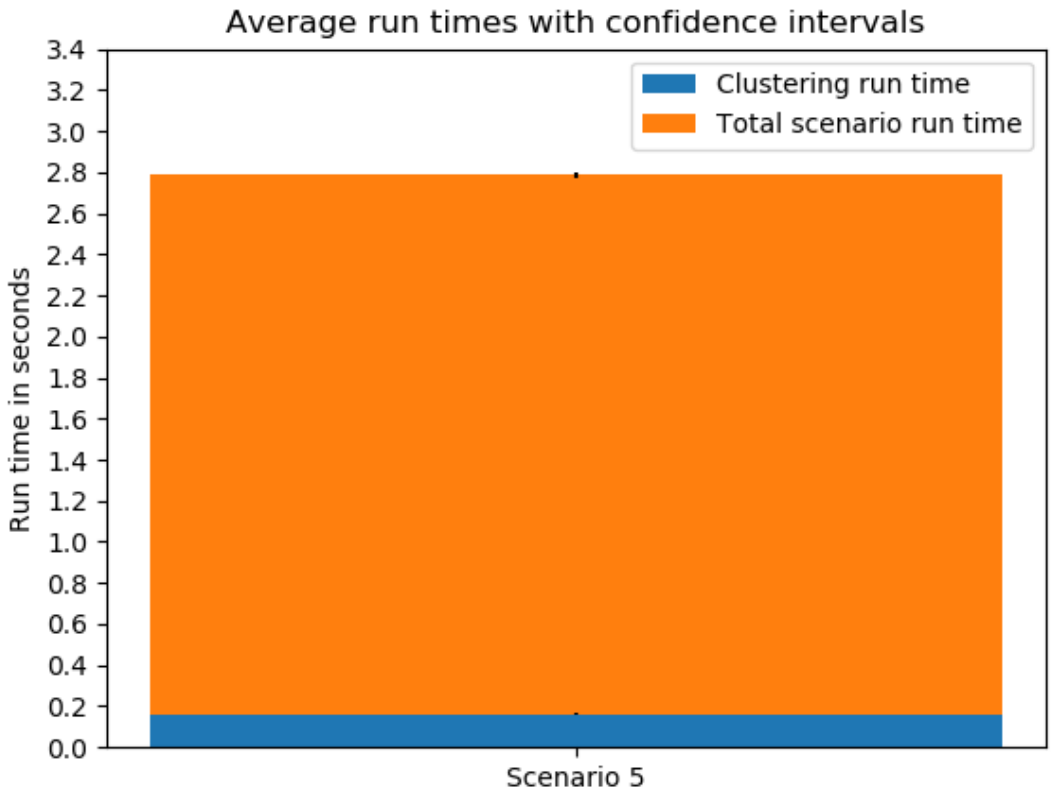

Figure 13 - Plot of analysis of run time of Scenario 5

Table 12 - Statistical analysis of Scenario 5 (in seconds)

Labels	***Best clustering run time***	***Best clustering wall time***	***Total scenario run time***	***Total scenario wall time***
Standard deviation	0.00635509	0.00634820	0.03540428	0.03536511
Confidence interval	(0.15823063, 0.16305085)	(0.15829586, 0.16311086)	(2.61370578, 2.64055926)	(2.61382713, 2.64065089)
Margin of error	0.00241010	0.00240749	0.01342673	0.01341188

5.3.2 Evaluation and analysis of IPS

As Table 7 shows, DFS synthesised a consistent program in 0.0008882038225876669 seconds on average while sort and add required 0.0013154731219639467 seconds on average to do the same. The DFS technique is thus approximately 39% faster than the sort and add search technique, which is shown well in Figure 14. I believe this can be attributed to the fact that the synthesised program is very simple and as such the search depth, called *T* in Balog et al. (2016), is only 1.

The fact that the average DFS wall time shown in Table 7 is lower than the run time calculated with a higher precision clock is somewhat odd. This can be attributed to the fact that some of the measurements of the DFS wall time are 0.0, leading to a lower mean value than expected. Most likely this occurs because the precision of the wall time clock is sometimes not high enough to capture very small intervals of time – this is the main reason for using the higher precision performance counter as the primary performance measurement.

Table 13 shows the statistical analysis done on the data in Table 7. A plot of the analysis is shown in Figure 14. The standard deviation of the run time for both the DFS and sort and add search techniques is low which indicates that the measurements are consistent as they are close to the mean. The corresponding 95% confidence interval and margin of error support this observation. Any conclusions drawn by analysing the wall time may not be reliable as the underlying data includes several zero values and are therefore not presented.

Because the results presented in Table 6 show that the synthesised programs are consistent with the input-output examples and the performance measurements presented in Table 7 show that the program synthesis is very fast, I argue that the artefact fulfils its program synthesis related requirements. The analysis done in Table 13 shows that the performance measurements are also consistent over several executions of the program, increasing the reliability of the results and thus the confidence of my argument.

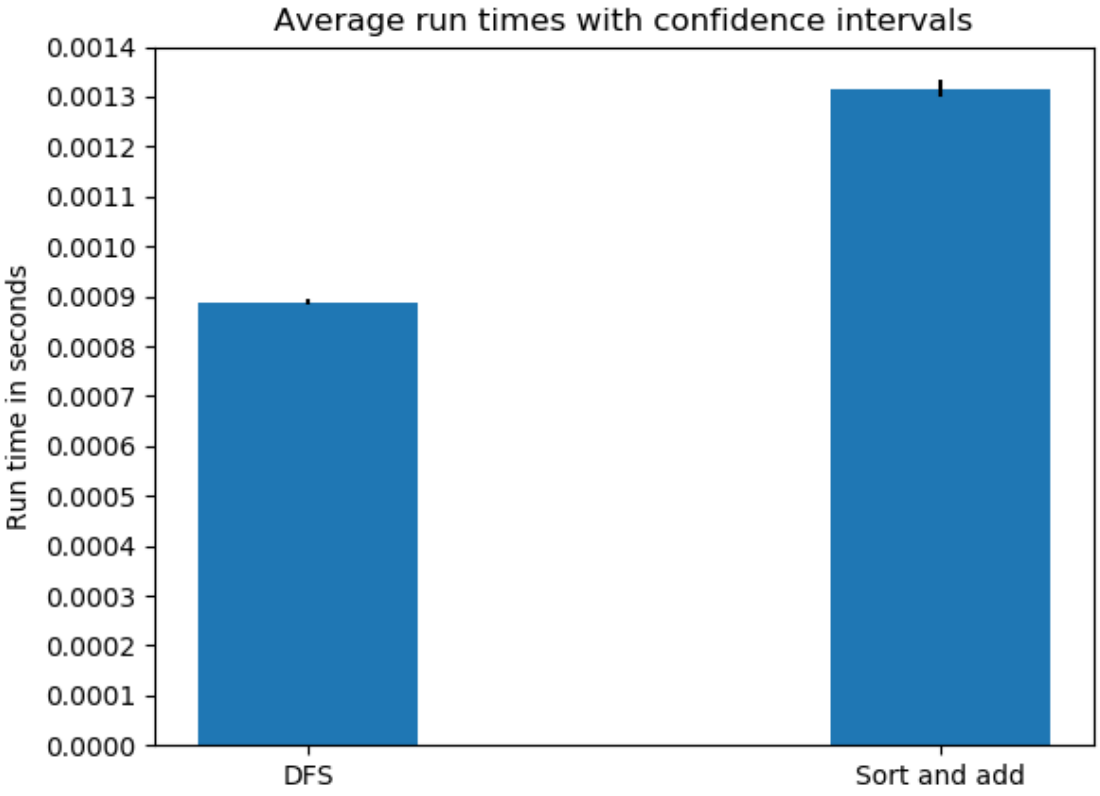

Figure 14 - Plot of analysis of IPS search techniques (lower is better)

Table 13 - Statistical analysis of inductive program synthesis (in seconds)

Labels	***DFS run time***	***DFS wall time***	***Sort and add run time***	***Sort and add wall time***
Standard deviation	0.00001559	0.00042182	0.00004483	0.00049754
Confidence interval	(0.00088229, 0.00089411)	(0.00060465, 0.00092459)	(0.00129847, 0.00133247)	(0.00127398, 0.00165136)
Margin of error	0.00000591	0.00015997	0.00001700	0.00018869

6 Discussion

In this section the results will be summarised, the originality and significance of the results will be argued, and the quality of the thesis will be discussed, including the quality of its measurements. Limitations of the work done in the thesis are also brought to light and future work is discussed. Finally, conclusions are drawn, and ethical and societal consequences of the artefact are discussed.

6.1 Summary of Results

The empirical results of the research show that it is possible to use inductive program synthesis to synthesise simple leader election programs by using DeepCoder's LIPS framework. The results also show that k-medoids clustering can be useful in organising a distributed system into groups based on the context of the different nodes operating in the system. Performance can certainly be improved but was good enough to allow for implementation on limited hardware devices.

There is to my knowledge no previous research that involves the combination of clustering and inductive program synthesis to organise and control distributed systems. However, without the research previously conducted within these individual fields, the work done in this thesis would of course have been impossible. Inductive program synthesis (Gulwani, Polozov & Singh, 2017), k-medoids clustering (Kaufmann & Rousseeuw, 1987) and Silhouettes (Rousseeuw, 1987) are central concepts of this thesis and without the practical implementations of DeepCoder (Balog et al., 2016)[14], scikit-learn (Pedregosa et al., 2011) and Matplotlib (Hunter, 2007) the results would have been harder to obtain and present, if not impossible.

6.1.1 Originality and significance of the research

There was no previous solution to the problem found when conducting the literature study, as described in Section 4.1, and the approach proposed in this thesis is a novel application of DeepCoder. As also mentioned in Section 4.1, other work on bringing order to distributed systems is mostly focused on stringently proving or verifying that a system is controllable, rather than practical approaches. Although systematic verification and proofs could improve the quality of the artefact and research, the results indicate that the proposed approach is feasible and could be useful in practical scenarios where more formal approaches may not be possible.

As the scenarios in Section 5.2 were run 30 times and produced the same results each time, within relatively similar time frames, I argue that the results are consistent. Considering the results in Section 5.2, the functional requirements described in Section 4.2 seem to be fulfilled, and by looking at the analysis of the results in Section 5.3 it can be argued that the non-functional requirements are also met.

The use of IPS to elect leaders in the artefact allows developers of distributed systems to express leader election algorithms by supplying abstractions in the form of input-output examples instead of writing complete programs that implement the algorithms. These abstractions may also cover scenarios that were not thought of beforehand by the developers, thus making the artefact able to adapt to unforeseen circumstances.

[14] *The Python 3 implementation was written by GitHub user dkamm*

Considering all the previously presented information, I argue that the research and its results are both original and significant.

6.2 Quality of the Thesis

When evaluating the quality of a scientific thesis, there are two main aspects of measurements that are important to discuss – reliability and validity. Reliability is a way of assessing whether the measurements measure the same thing and produce reproducible results. Validity assesses whether measurements measure what they actually claim to measure.

6.2.1 Reliability

It is possible to reproduce the results by running the program main_program.py[15]. To obtain data in a format more akin to that presented in Section 4 of this thesis, the programs format_output.py[16] and analysis.py[17] can be run together with some manual data handling. The scenarios that were used to test and demonstrate the artefact were run 30 times each to make sure that they did not produce significantly different results.

6.2.2 Validity

Accuracy is important when it comes to discussing the validity of the performance measurements conducted in this thesis. One thing that could have impacted the accuracy negatively, and thus also the validity, is the fact that both performance counter time and wall time were measured for the same piece of code at the same time. To be done this way, it had to follow the order depicted in Figure 15.

[15] https://github.com/CarsonBeckett/deepcoder-le/blob/master/clustering/main_program.py

[16] https://github.com/CarsonBeckett/deepcoder-le/blob/master/clustering/format_output.py

[17] https://github.com/CarsonBeckett/deepcoder-le/blob/master/clustering/analysis.py

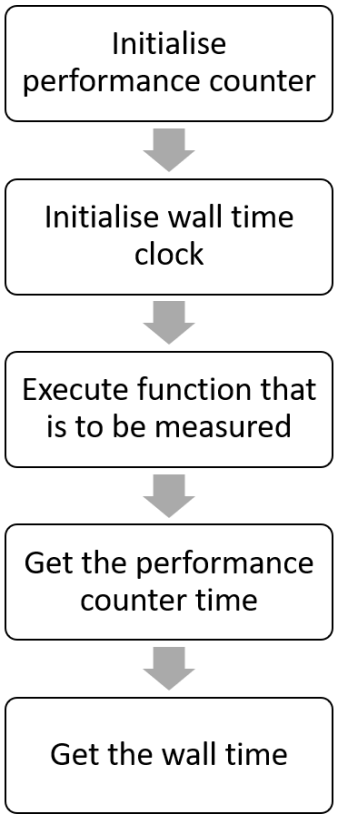

Figure 15 - Order of performance measurement timings

This means that each measurement of the performance counter time also includes the time it takes to initialise the start of the wall time, and the wall time also includes the time it takes to get the end of the performance counter time. This might lead to greater measurement values for both timings which indicate that it took more time to execute a function, thus misrepresenting the true value of the measured quantity. However, the difference should be quite small as Python's core time functions should be well optimised and consistent enough that it should not decrease the validity of the results by too much[18].

External validity could have suffered from the artificial setting of the simulated environment that the artefact was tested in. However, there was not enough time to test it in a real distributed system.

[18] *Readers who are interested in looking deeper into this risk are advised to read a comparison of Python clock functions, for example this one: https://www.webucator.com/blog/2015/08/python-clocks-explained/*

6.3 Limitations

To further evaluate the artefact, the runtime of the synthesised leader election program could have been measured, but as the ranking was done outside of DeepCoder and the leader election algorithm was so simple (choose the maximum value of a collection of values) it was deemed unnecessary. If other, more complicated, leader election algorithms were implemented with IPS or if the ranking would be synthesised using IPS it would have been of greater importance to measure this as the run time of these programs could potentially not be trivial.

Cluster analysis was performed by analysing the average Silhouette value of each clustering. The metric that was used in the distance calculation for the Silhouette value was the Euclidean distance, not the Manhattan distance that was used to perform the actual clustering. This was done because the implementation of the Silhouette calculation in scikit-learn used the Euclidean distance by default and it would have taken more time than available to change it to use the Manhattan distance. The Silhouette calculation itself worked fine with the Manhattan distance and the results were very similar to those generated by using the Euclidean distance. A consistent improvement in execution time ranging from 2% to 29% was observed for the Silhouette value calculations, with the improvement increasing with the input size. This was to be expected since the Manhattan distance is easier to compute than the Euclidean distance according to Park and Jun (2009) and the time complexity of the Silhouette algorithm is quadratic according to Petrovic (2006). The Silhouette values calculated for each clustering were very similar to the values obtained using the Euclidean distance, with some being slightly worse and others slightly better, but not enough to produce significantly different results. The consistent improvement in execution time would however help in improving the scalability of the artefact as throughput requirements increase. It would have been beneficial to the thesis to include such analysis as well, however, for reasons unknown the IPS program could not synthesise a program that was consistent with all input-output examples when using the Manhattan distance to calculate the Silhouette of a clustering. Thus, the Silhouette values calculated using the Manhattan distance were not included in the results because of the aforementioned issues with the IPS program and a lack of time to fix them.

6.4 Future Work

6.4.1 Extensions to the DeepCoder DSL

The DSL specified in the DeepCoder paper (Balog et al., 2016) is sufficient to express the leader election algorithm that was implemented in this thesis. However, this leader election algorithm is very simple and potential future improvements to the artefact would be to implement more complex leader election algorithms with DeepCoder's LIPS framework as well as let DeepCoder synthesise the ranking functions. The DSL might thus have to be extended to be able to express the more advanced leader election and ranking algorithms.

6.4.2 Train and use DeepCoder's neural network

In this thesis the search techniques for IPS were not augmented by DeepCoder's neural network. This was in part due to time limitations but also because it was simply not necessary to speed up the synthesis of the leader election program because it was so simple, and synthesis was extremely fast. This might change in the future when more advanced leader election programs are to be synthesised and possibly other functions, such as ranking, are moved into programs synthesised by DeepCoder's

LIPS framework. The better performance gained by utilising DeepCoder's neural network will be more useful, if not necessary, in this situation.

6.4.3 Overall performance improvements

Many of the algorithms that were used in the different programs of the artefact come from widely used Python libraries. However, there is certainly room for improvement when it comes to the efficiency of the code that was purpose-made for the artefact, both in terms of memory usage and speed of execution.

As memory usage ranged from 175MB to 235MB, it is possible to implement the artefact on embedded devices with limited resources. As data handling is a big part of any project that involves machine learning, however, much could be improved when it comes to passing data between functions and data conversion. One example of such an improvement would be to not pass parts of already analysed data to a function that is repeatedly executed in a loop. This would reduce the memory usage of the artefact and might increase its speed of execution as well.

6.4.4 Test in a real distributed system

The artefact that was developed in this thesis was only tested in a simulated distributed system. In order to prove that it is truly capable of solving the real-world problem formulated in the problem description in Section 1.2, the artefact also needs to be tested in a real-world environment. This would increase confidence in the artefact by improving validity as well as hopefully providing even more insight into how the artefact can be further improved.

6.4.5 Investigate network topology for the distributed system

For the artefact to be used effectively in a real distributed system, network topologies must be investigated. The work on distributed verification of the controllability of weighted out-tree topologies by Guo, Yan and Lin (2012) could be an excellent starting point for this investigation.

6.4.6 Use other machine learning techniques for program synthesis

Even though reinforcement learning was deemed too complicated to be used for the work conducted in this thesis because of time constraints and uncertainty whether it was well-suited to the problem at hand, it could yield interesting results to replace the neural network with an on-line reinforcement learning algorithm that does not only learn before synthesising programs but continues to learn during operation.

6.5 Conclusions

An artefact was developed to test how clustering and inductive program synthesis could be used to organise and control distributed systems.

The artefact can indeed be used to solve the practical problem of clustering nodes in a distributed system by context and electing leaders for the produced clusters using IPS. It fulfils the requirements set upon it well, which brings the possibility of implementation on limited hardware. In a real system, the clustering stage of the artefact should be implemented on a higher level than every node, for example, on sinks in an IoT environment. The IPS stage is of course intended to be implemented on every node to facilitate leader election and the results make this possible.

Thus, the answer to the research question is that DeepCoder's LIPS framework and k-medoids clustering, coupled with other machine learning and scientific libraries used in this thesis, can be used to organise and control distributed systems.

It is important, however, to clarify that the artefact does not provide a way of completely controlling an actual system. Rather, it can be used as a framework for facilitating organisation and control of the system – every system will have different sensors, actuators and requirements. In other words, how the elected leaders are supposed to control the system needs to be implemented by programmers for each system, but the artefact can help in controlling the system once these functions are implemented.

The practical usefulness of the artefact is also somewhat limited in its current state. The leader election algorithm used is extremely simple, and testing was conducted in a simulated distributed system, not an actual one. DeepCoder's LIPS is not utilised to its full extent as the neural network was not used to augment the search techniques, something that might be necessary to use in the future as to not render more advanced leader election algorithms intractable to synthesise within reasonable time frames.

I believe, however, that the concept is promising and that the produced artefact can be expanded to implement more advanced leader election algorithms and further utilise the power of clustering to organise and control distributed systems in an adaptive way. Especially interesting is that the neural network of DeepCoder's LIPS should make it feasible to synthesise more advanced leader election algorithms within reasonable time frames.

6.5.1 Ethical and societal consequences of the conclusions

It was mentioned in Section 1.3 that one of the potential improvements that the artefact could bring is reduced work needed by human programmers to write programs that organise and control distributed systems. This can of course be viewed as a negative consequence of the work done in this thesis as a result of lost employment for some people, but I believe that good programmers will still have jobs by the end of the day, as there is simply so much work available for this group of professionals. There are still things that need to be done by human programmers when it comes to organising and controlling distributed systems, but hopefully they will be able to do more in the same amount of time than they did before by using the developed artefact.

Another potential consequence of the work done in this thesis is that distributed systems may be controlled more autonomously than before. This might be a concern to some people, but it is quite far from taking its own decisions as the artefact only facilitates which nodes should control the system, not how – this is something that must be implemented by human programmers as every system has different requirements on what needs to be controlled and how.

References

Alpaydin, E. (2014). *Introduction to machine learning*. 3rd edition. MIT press.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027-1035). Society for Industrial and Applied Mathematics.

Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2016). Deepcoder: Learning to write programs. In: *International Conference on Learning Representations* (*ICLR) 2017*.

Bradley, P. S., & Fayyad, U. M. (1998). Refining Initial Points for K-Means Clustering. In: *ICML* (Vol. 98, pp. 91-99). Microsoft Research.

Cachin, C., Guerraoui, R., & Rodrigues, L. (2011). *Introduction to reliable and secure distributed programming*. 2nd edition. Springer Science & Business Media.

Coulouris, G. F., Dollimore, J., Kindberg, T. & Blair, G. (2012). *Distributed systems: concepts and design*. 5th edition. Pearson Education.

Denscombe, M. (2010). *The good research guide: for small-scale social research projects*.4th edition. McGraw-Hill Education (UK), pp.4-9.

Gulwani, S., Polozov, O., & Singh, R. (2017). Program synthesis. *Foundations and Trends® in Programming Languages*, *4*(1-2), pp.1-119.

Guo, D., Yan, G., & Lin, Z. (2012). Distributed verification of controllability for weighted out-tree based topology. In *American Control Conference (ACC), 2012* (pp. 1507-1512). IEEE.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(3), 90-95.

Johannesson, P., & Perjons, E. (2014). *An introduction to design science*. Springer.

Kaufmann, L., & Rousseeuw, P. (1987). Clustering by Means of Medoids. In: *Data Analysis based on the L1-Norm and Related Methods* (405–416).

Liu, Y., Li, Z., Xiong, H., Gao, X., & Wu, J. (2010). Understanding of internal clustering validation measures. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on* (pp. 911-916). IEEE.

Park, H. S., & Jun, C. H. (2009). A simple and fast algorithm for K-medoids clustering. In: *Expert systems with applications*, *36*(2), 3336-3341.

Parsons, L., Haque, E., & Liu, H. (2004). Subspace clustering for high dimensional data: a review. *Acm Sigkdd Explorations Newsletter*, *6*(1), 90-105.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, *12*, 2825-2830.

Petrovic, S. (2006). A comparison between the silhouette index and the davies-bouldin index in labelling ids clusters. In *Proceedings of the 11th Nordic Workshop of Secure IT Systems* (pp. 53-64).

Rahman, H., & Rahmani, R. (2018). Enabling distributed intelligence assisted Future Internet of Things Controller (FITC). *Applied Computing and Informatics*, *14*(1), 73-87.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, *20*, 53-65.

Schaeffer, S. E. (2007). Graph clustering. *Computer science review*, *1*(1), 27-64.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1, No. 1). Cambridge: MIT press.

Yu, F., Zhang, J., & Sun, C. (2010). Consensus of multi-agent systems based on controllability under leader-follower topology. In: *Intelligent Control and Automation (WCICA), 2010 8th World Congress on* (pp. 5958-5963). IEEE.

Appendix A – Silhouette analysis for different values of k

This section shows the Silhouette analysis of all the samples of a clustering for different values of *k* for each scenario presented in Section 4.2.

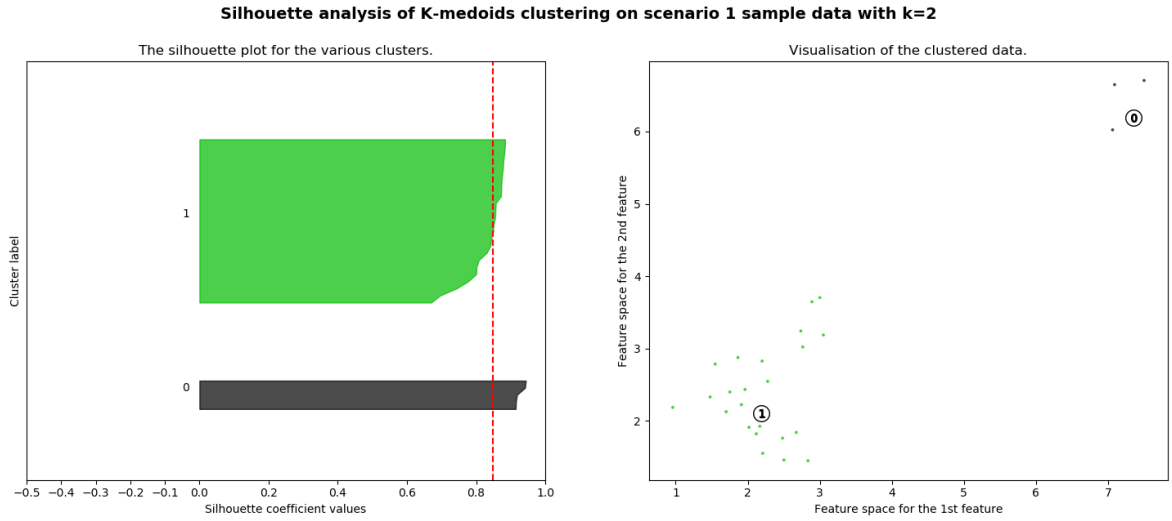

Figure 16 - Silhouette analysis of Scenario 1 with k=2

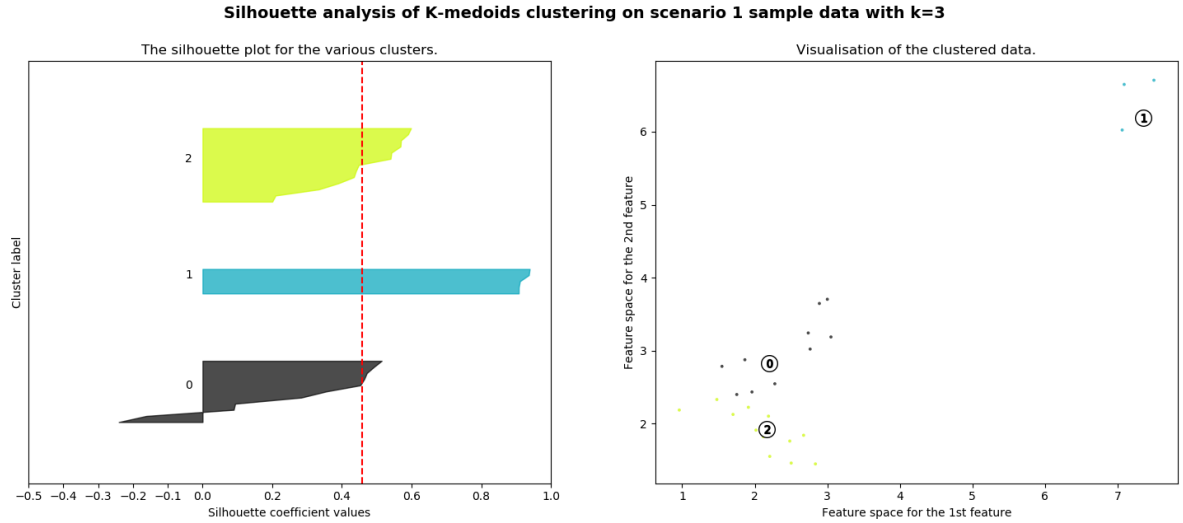

Figure 17 - Silhouette analysis of Scenario 1 with k=3

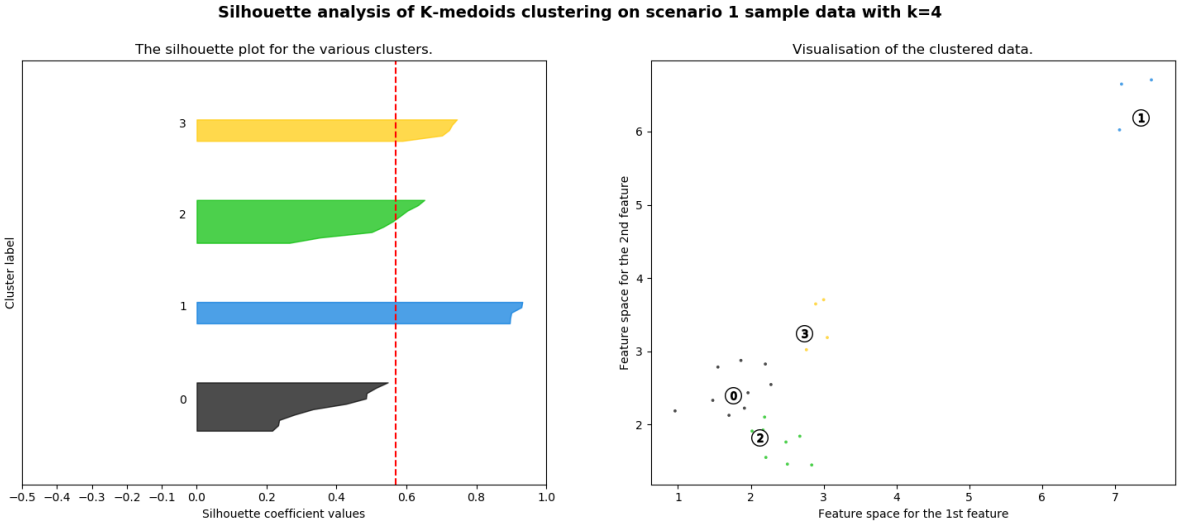

Figure 18 - Silhouette analysis of Scenario 1 with k=4

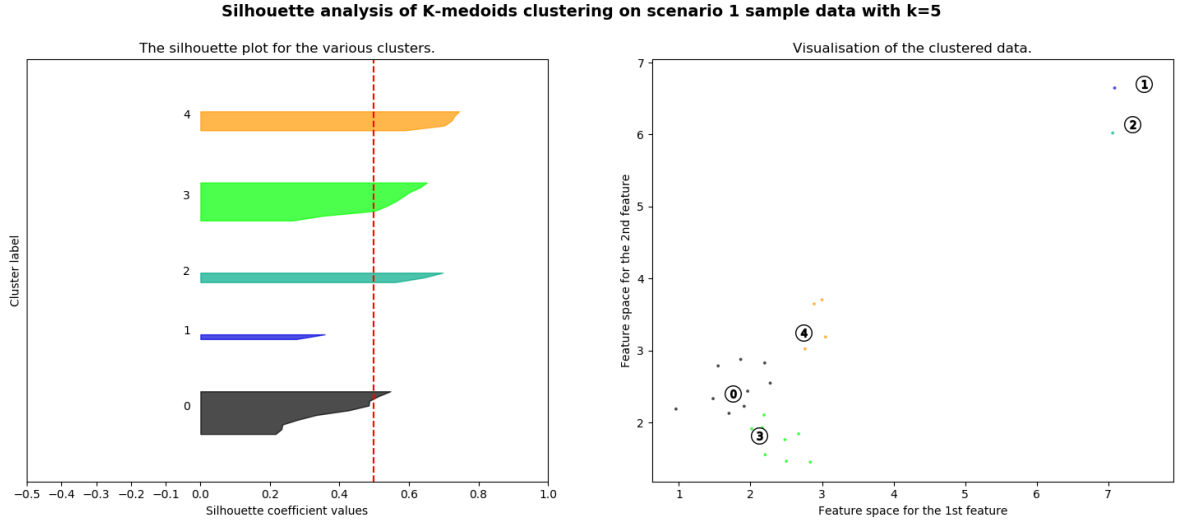

Figure 19 - Silhouette analysis of Scenario 1 with k=5

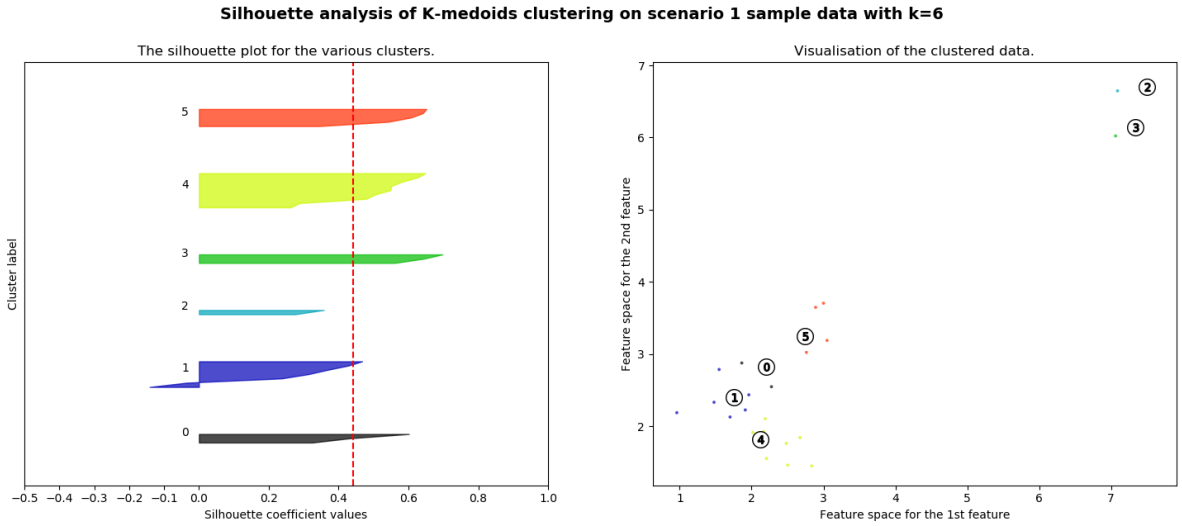

Figure 20 - Silhouette analysis of Scenario 1 with k=6

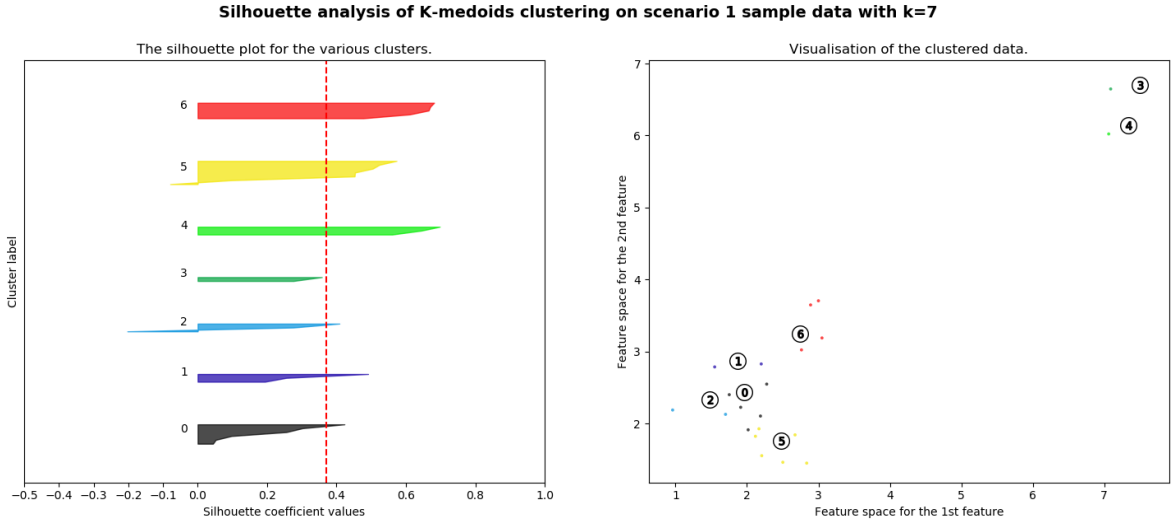

Figure 21 - Silhouette analysis of Scenario 1 with k=7

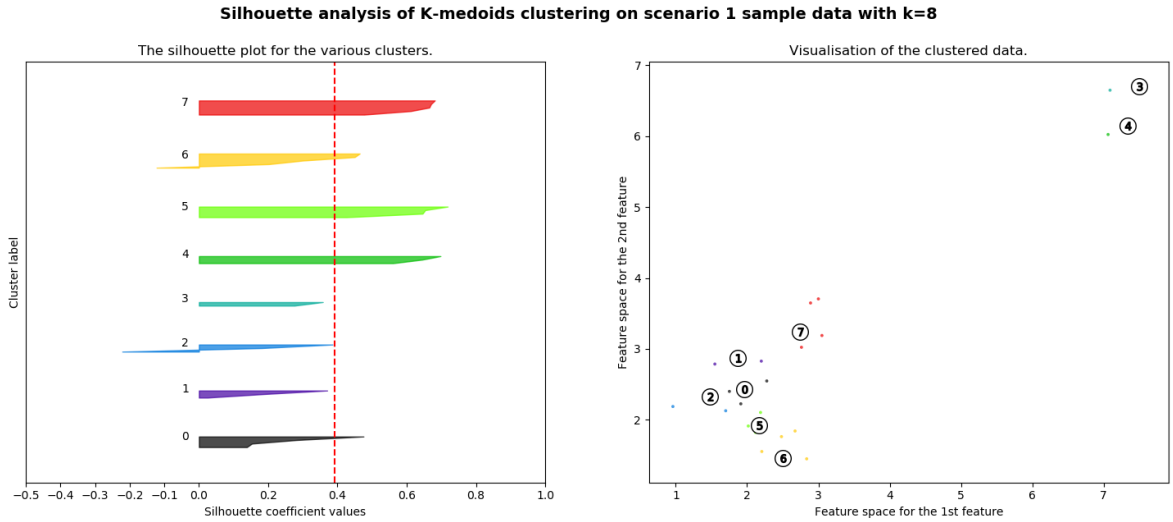

Figure 22 - Silhouette analysis of Scenario 1 with k=8

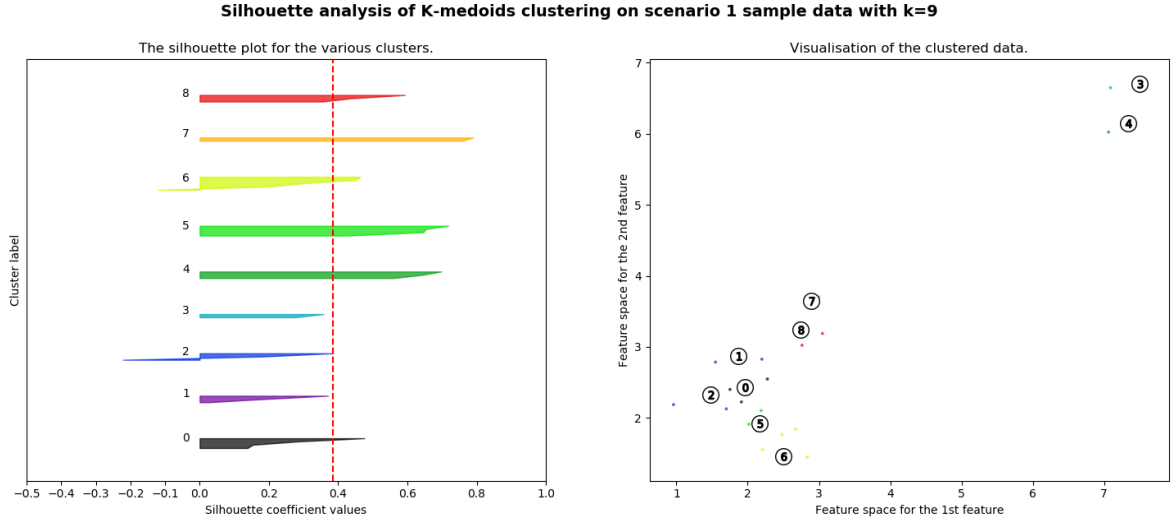

Figure 23 - Silhouette analysis of Scenario 1 with k=9

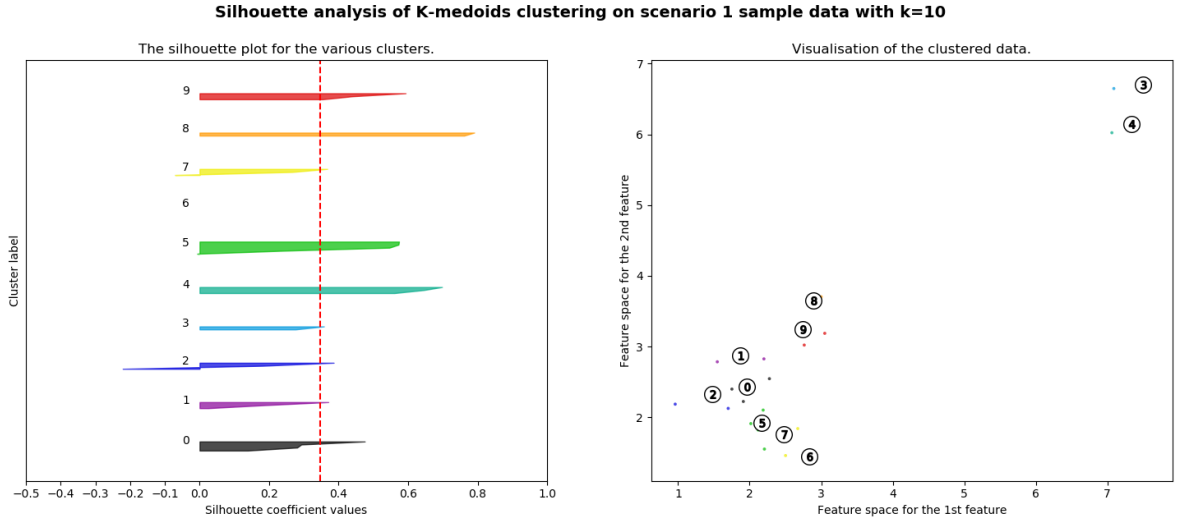

Figure 24 - Silhouette analysis of Scenario 1 with k=10

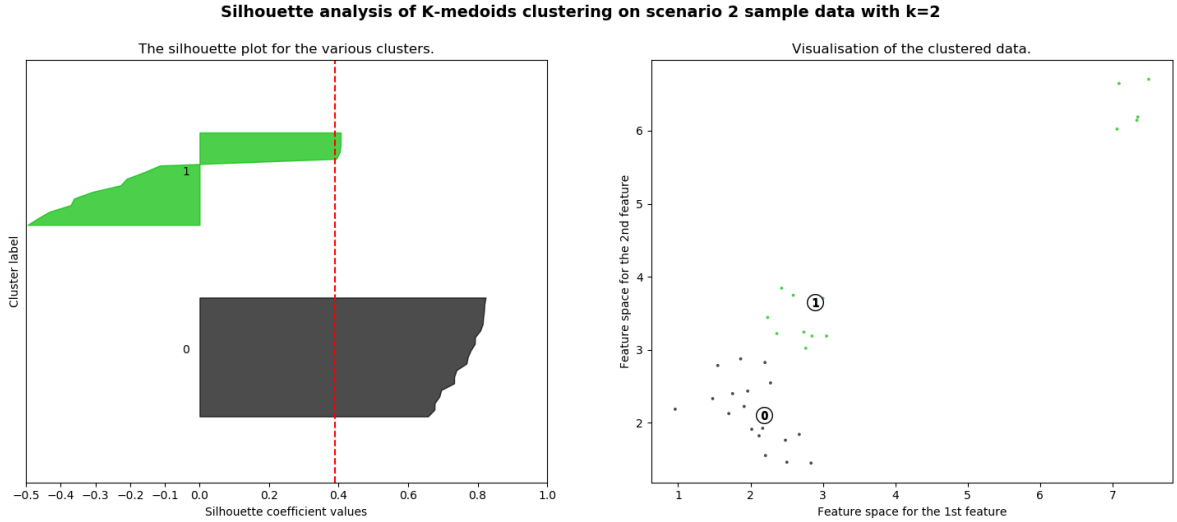

Figure 25 - Silhouette analysis of Scenario 2 with k=2

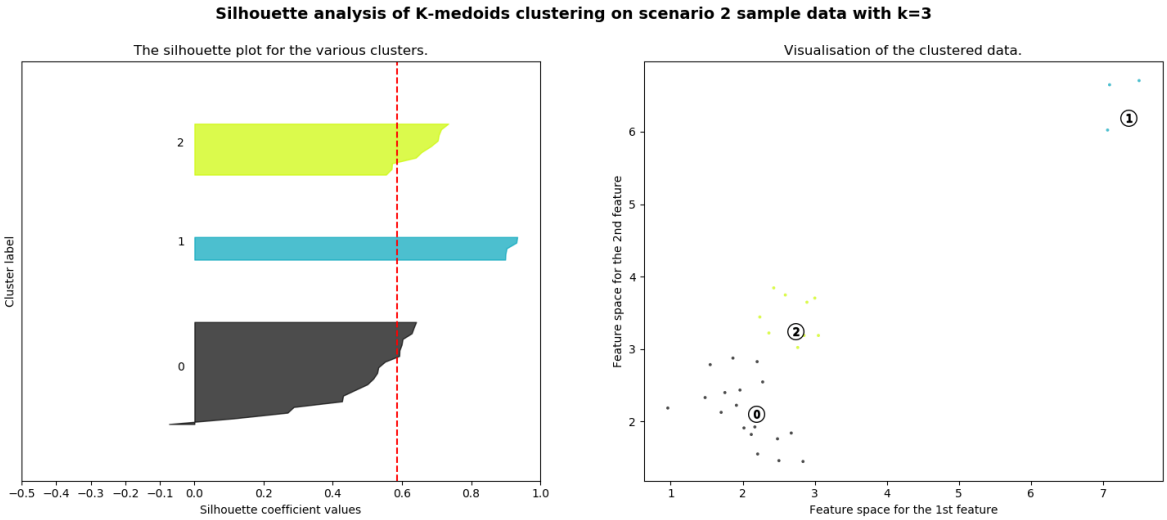

Figure 26 - Silhouette analysis of Scenario 2 with k=3

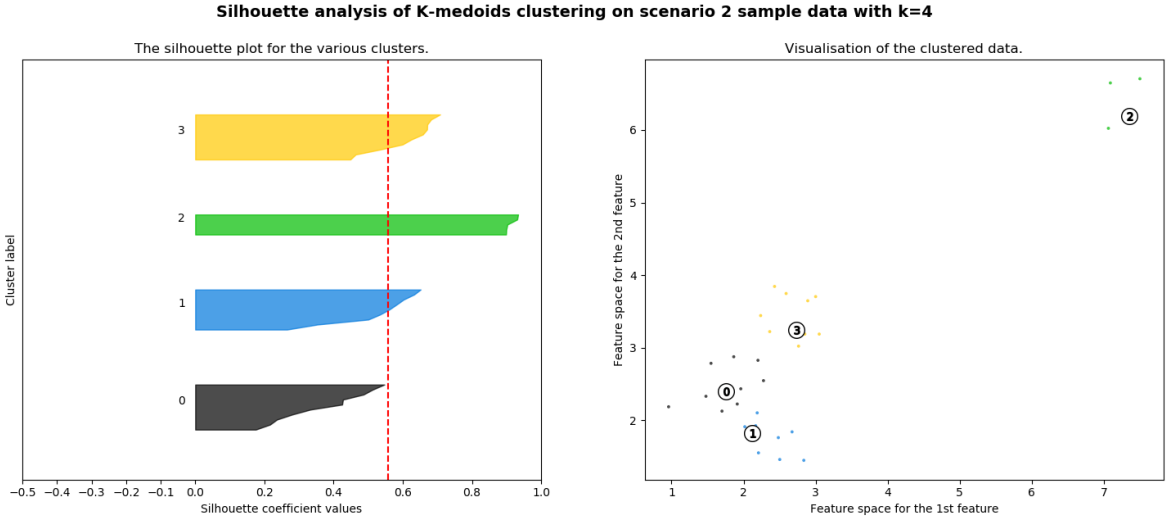

Figure 27 - Silhouette analysis of Scenario 2 with k=4

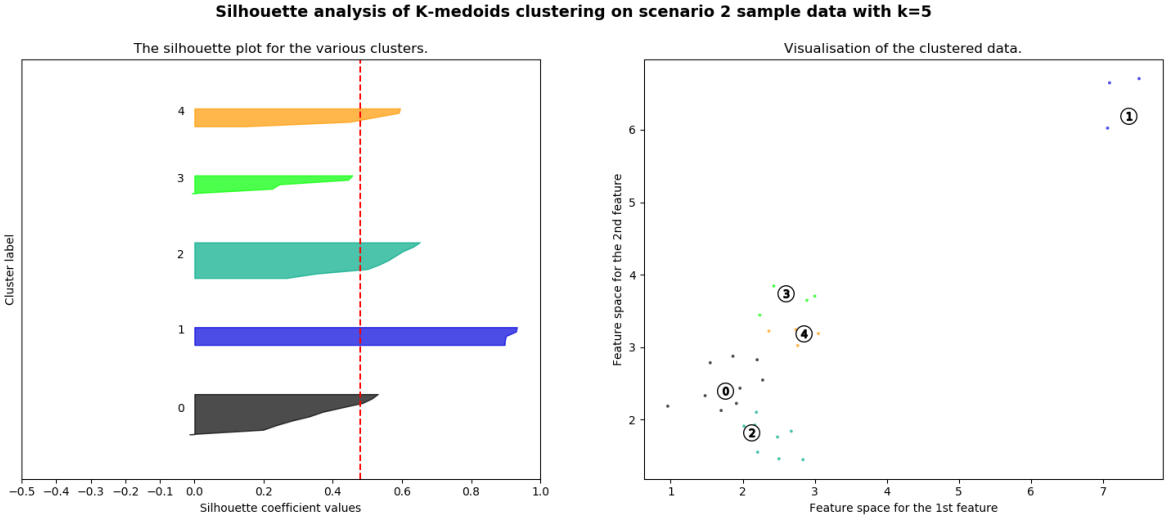

Figure 28 - Silhouette analysis of Scenario 2 with k=5

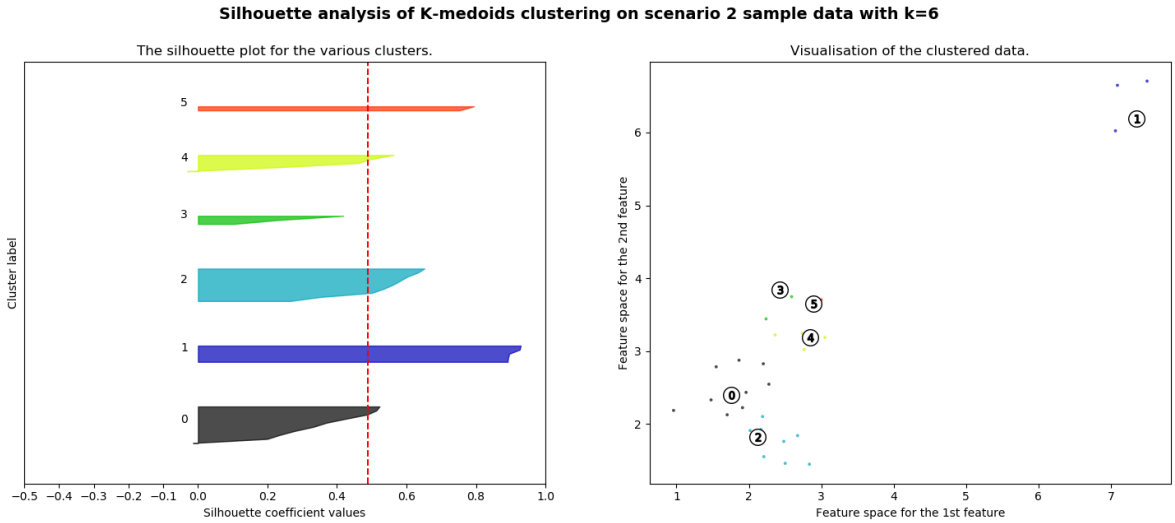

Figure 29 - Silhouette analysis of Scenario 2 with k=6

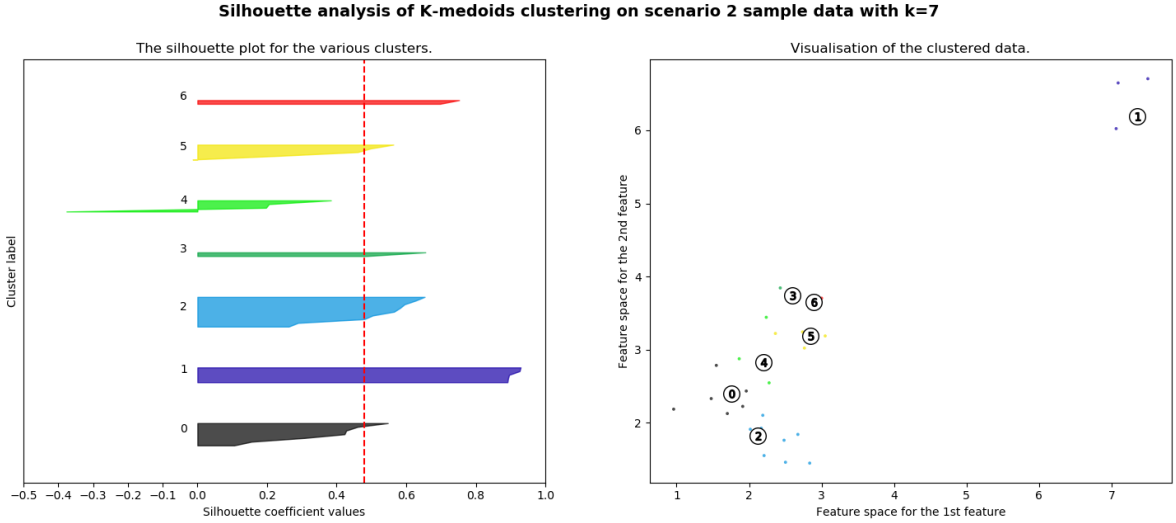

Figure 30 - Silhouette analysis of Scenario 2 with k=7

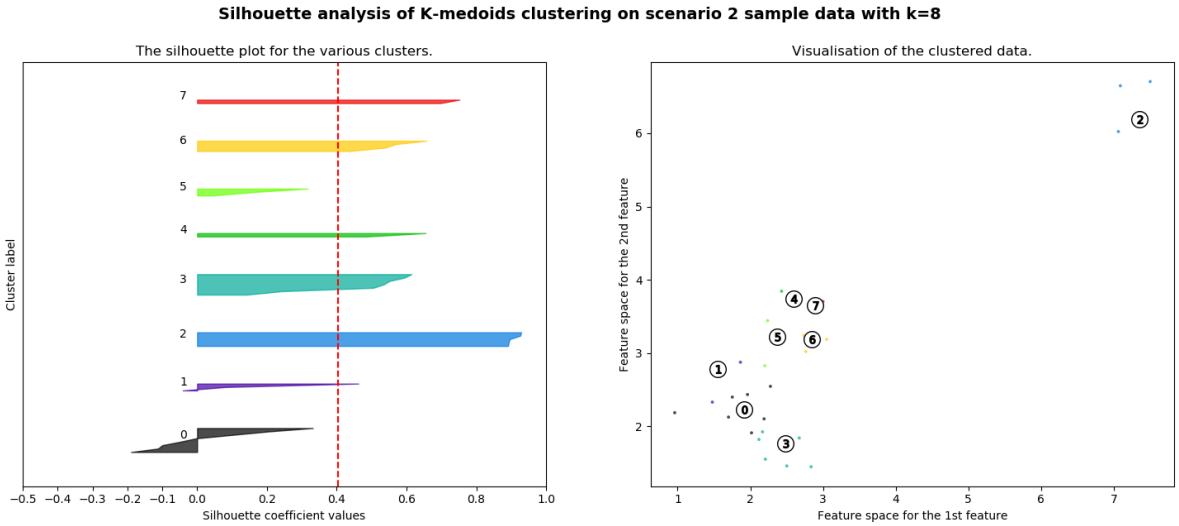

Figure 31 - Silhouette analysis of Scenario 2 with k=8

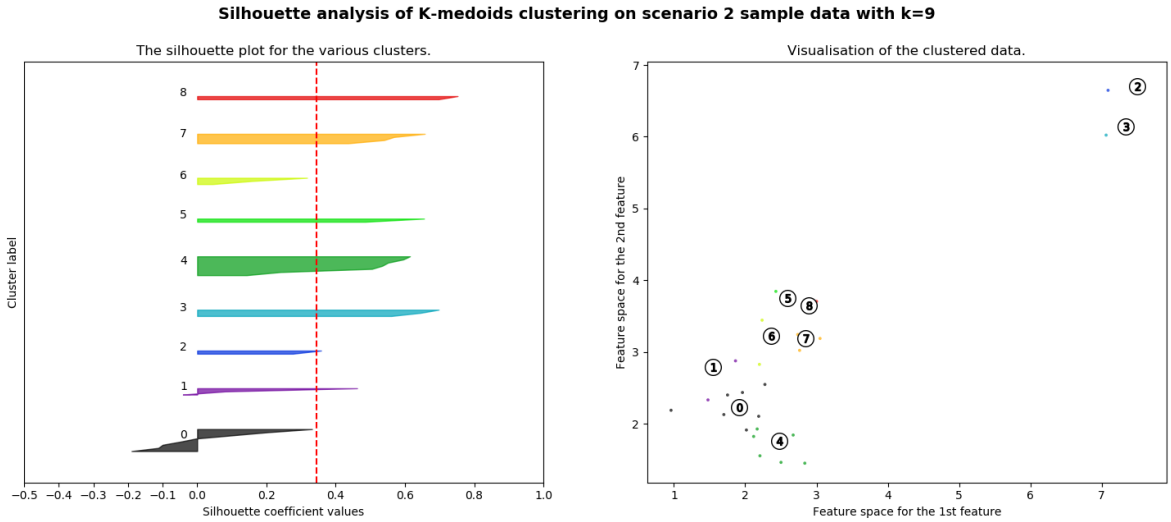

Figure 32 - Silhouette analysis of Scenario 2 with k=9

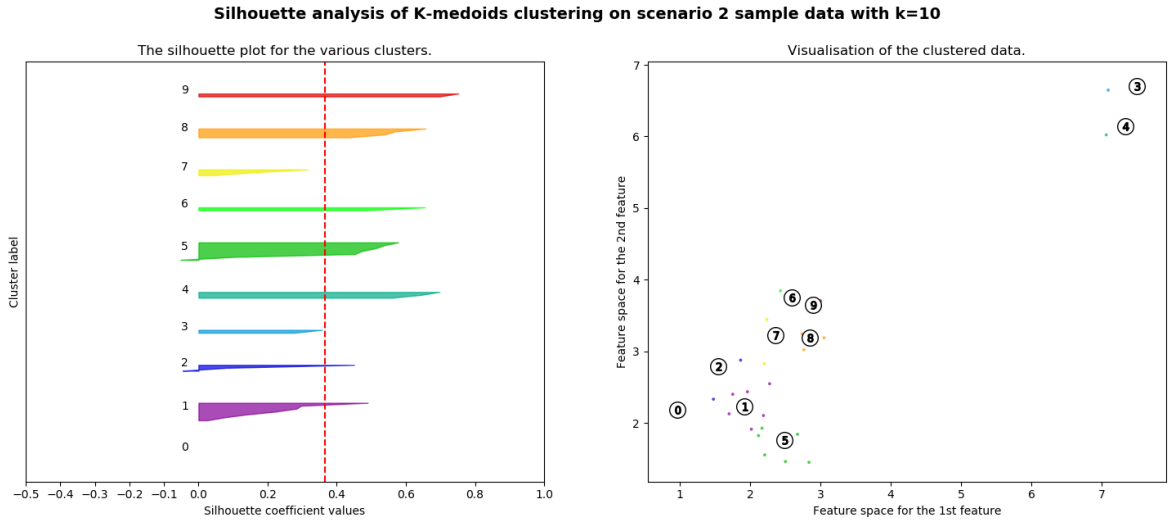

Figure 33 - Silhouette analysis of Scenario 2 with k=10

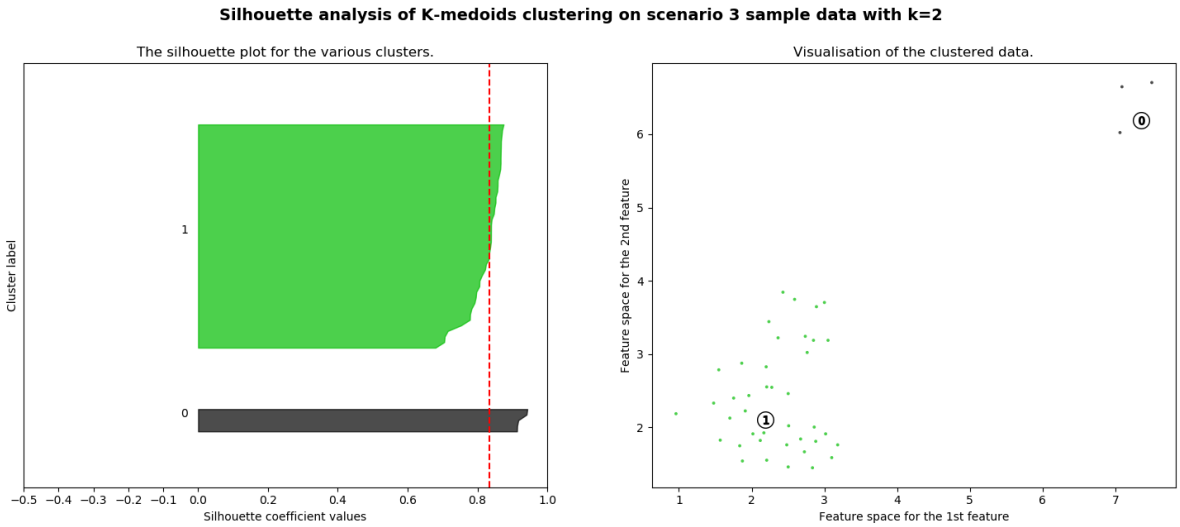

Figure 34 - Silhouette analysis of Scenario 3 with k=2

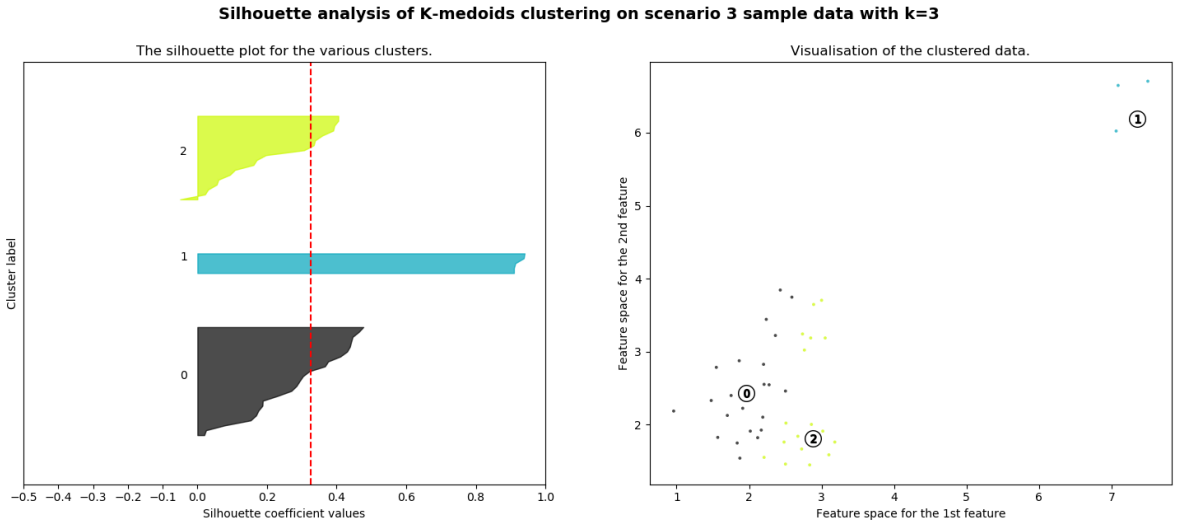

Figure 35 - Silhouette analysis of Scenario 3 with k=3

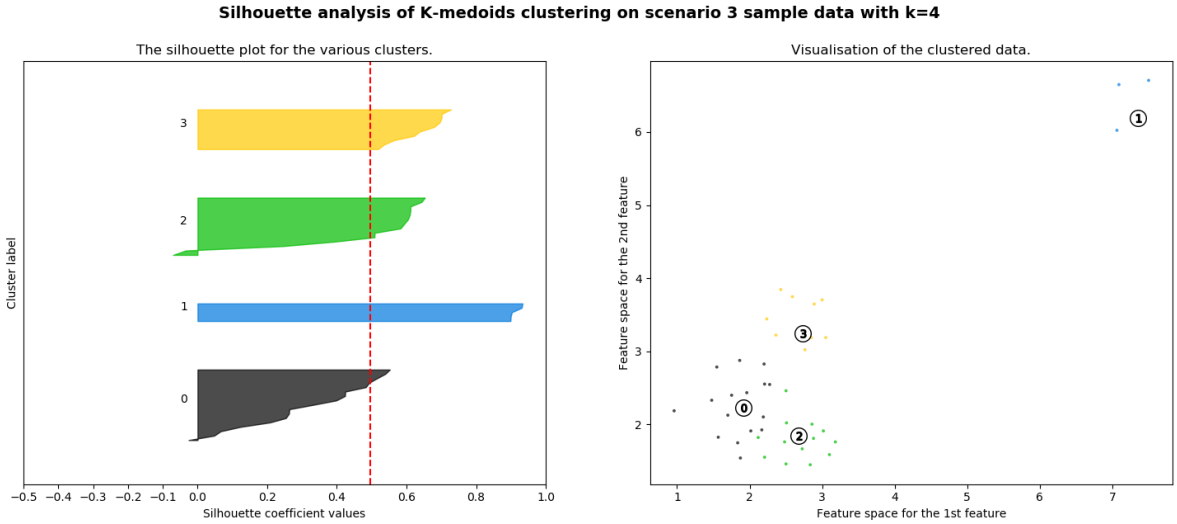

Figure 36 - Silhouette analysis of Scenario 3 with k=4

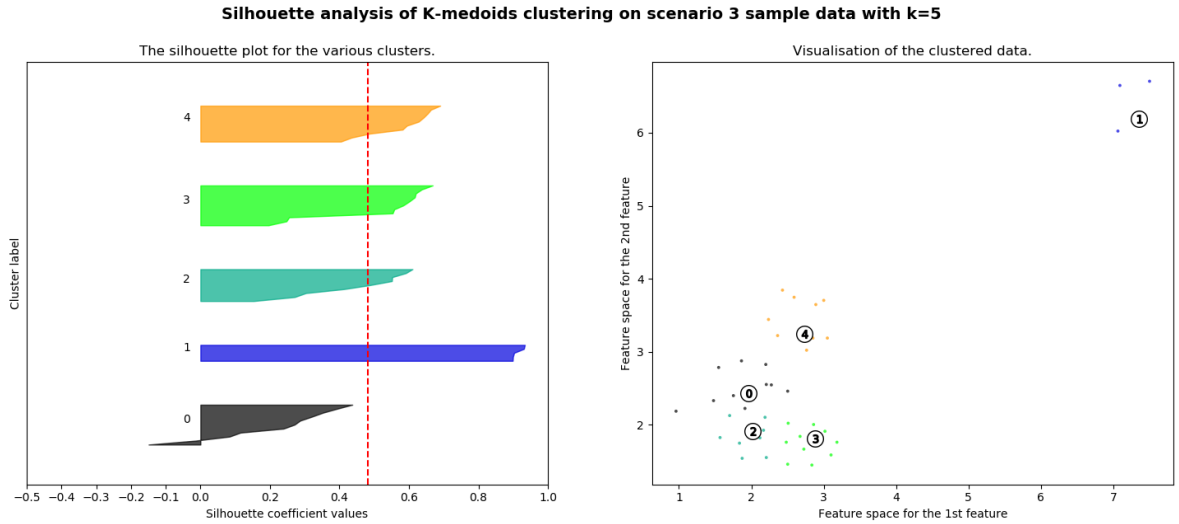

Figure 37 - Silhouette analysis of Scenario 3 with k=5

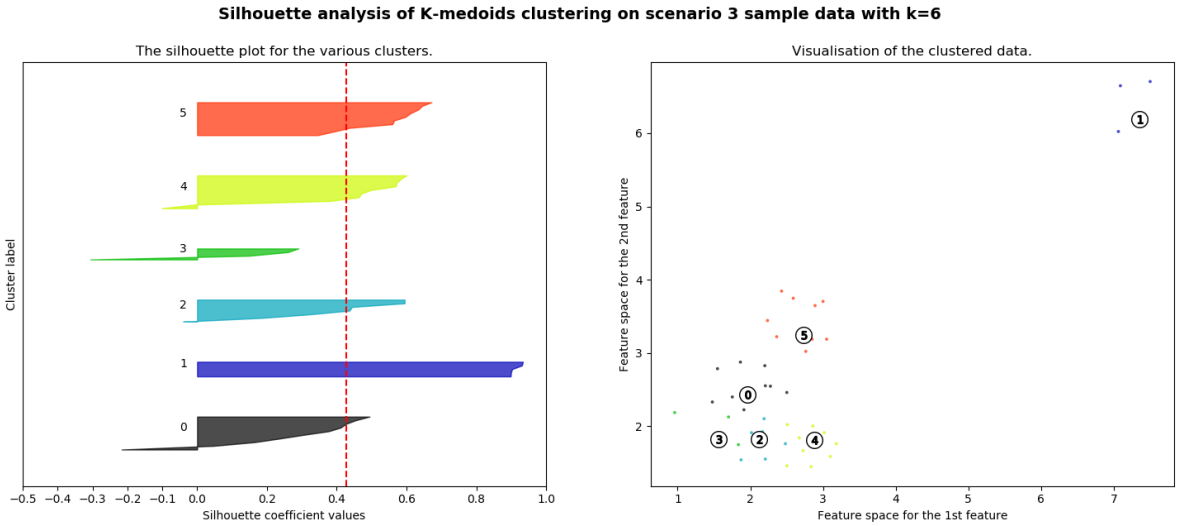

Figure 38 - Silhouette analysis of Scenario 3 with k=6

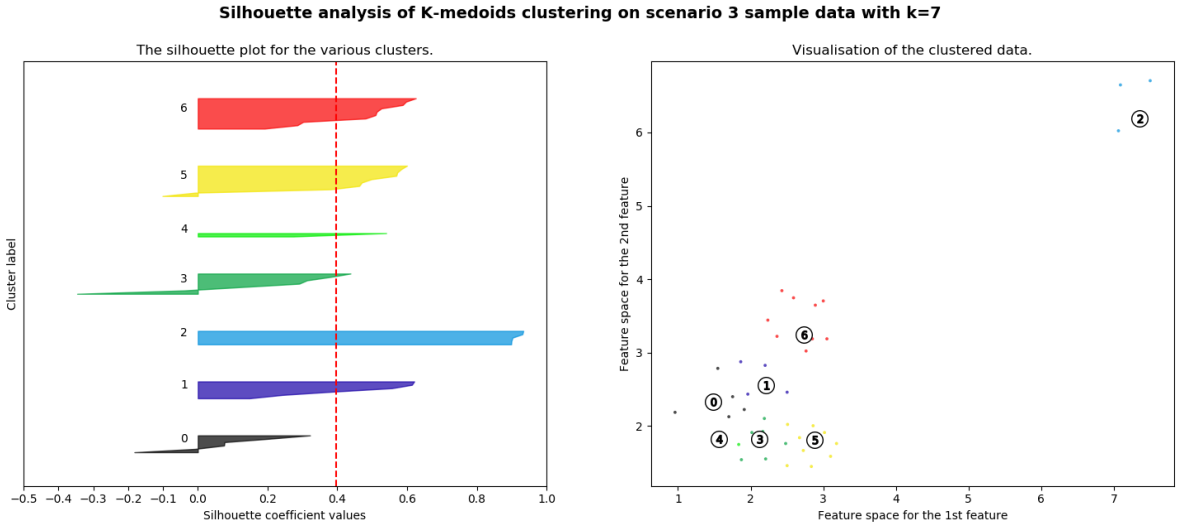

Figure 39 - Silhouette analysis of Scenario 3 with k=7

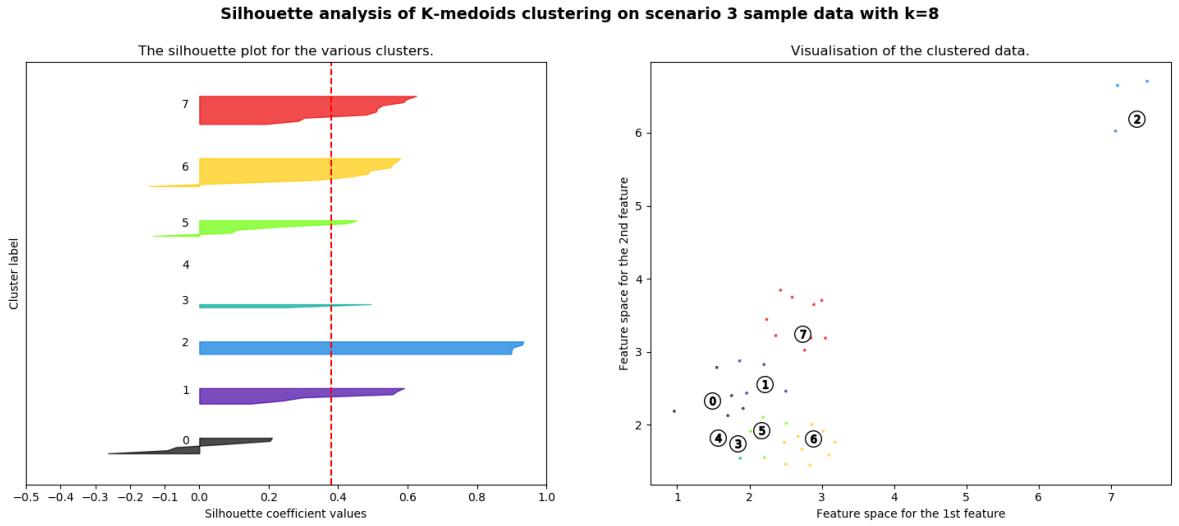

Figure 40 - Silhouette analysis of Scenario 3 with k=8

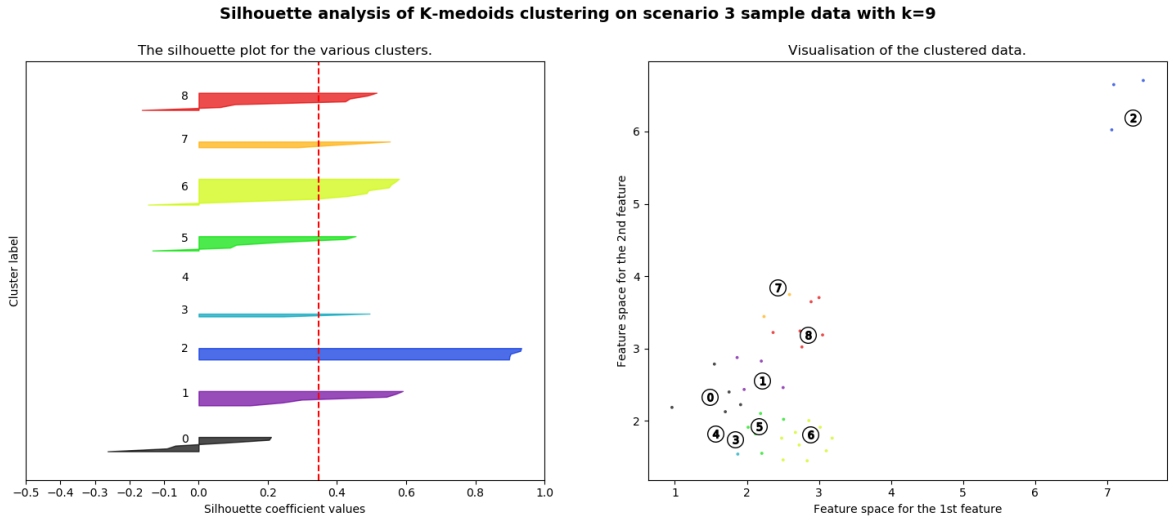

Figure 41 - Silhouette analysis of Scenario 3 with k=9

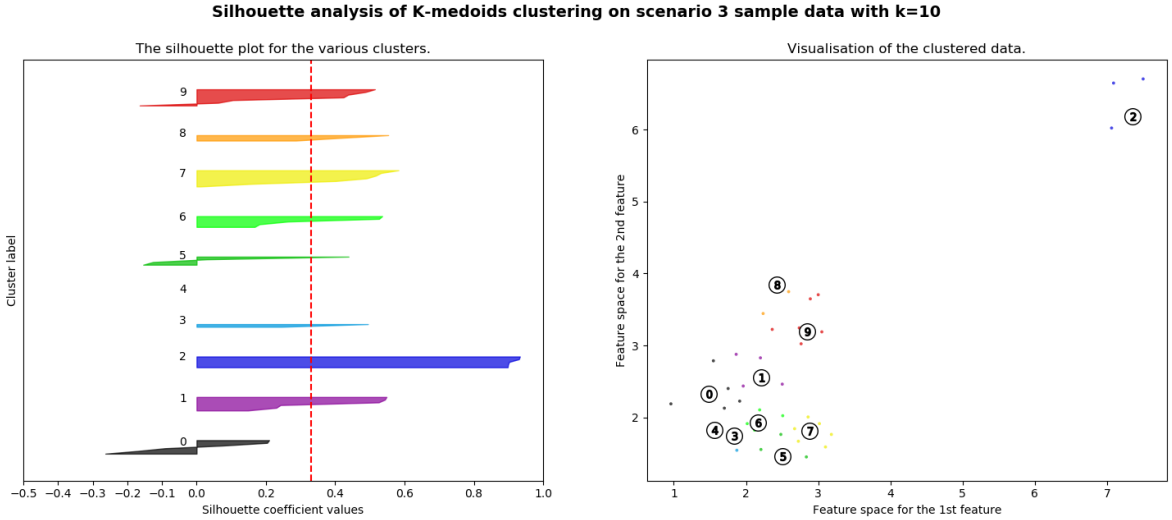

Figure 42 - Silhouette analysis of Scenario 3 with k=10

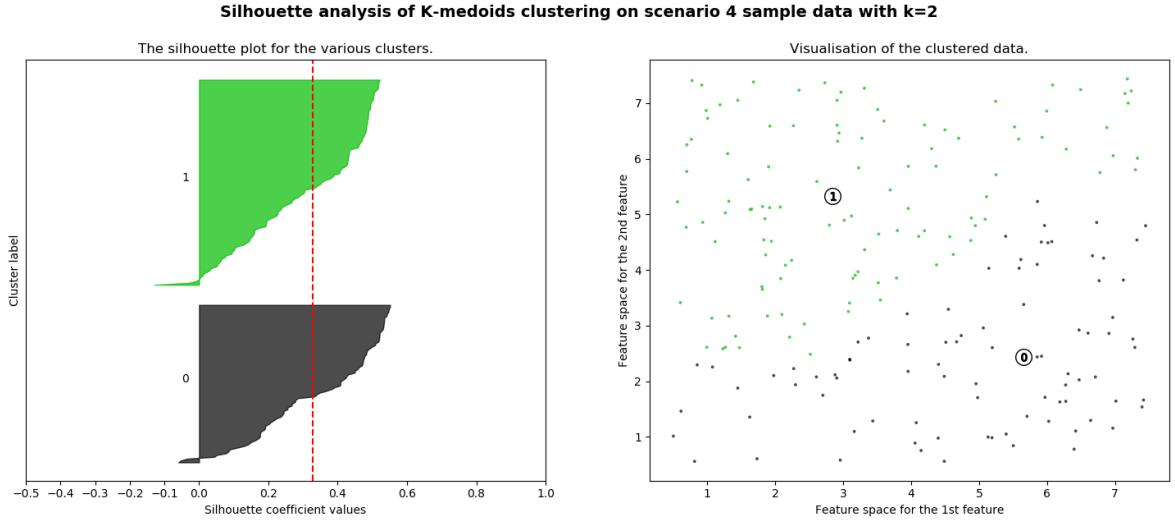

Figure 43 - Silhouette analysis of Scenario 4 with k=2

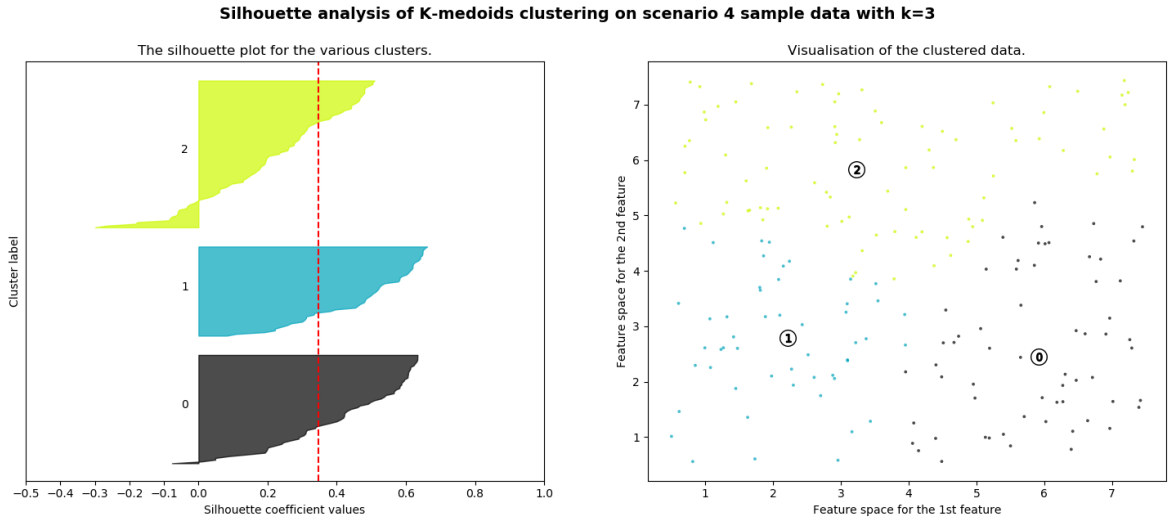

Figure 44 - Silhouette analysis of Scenario 4 with k=3

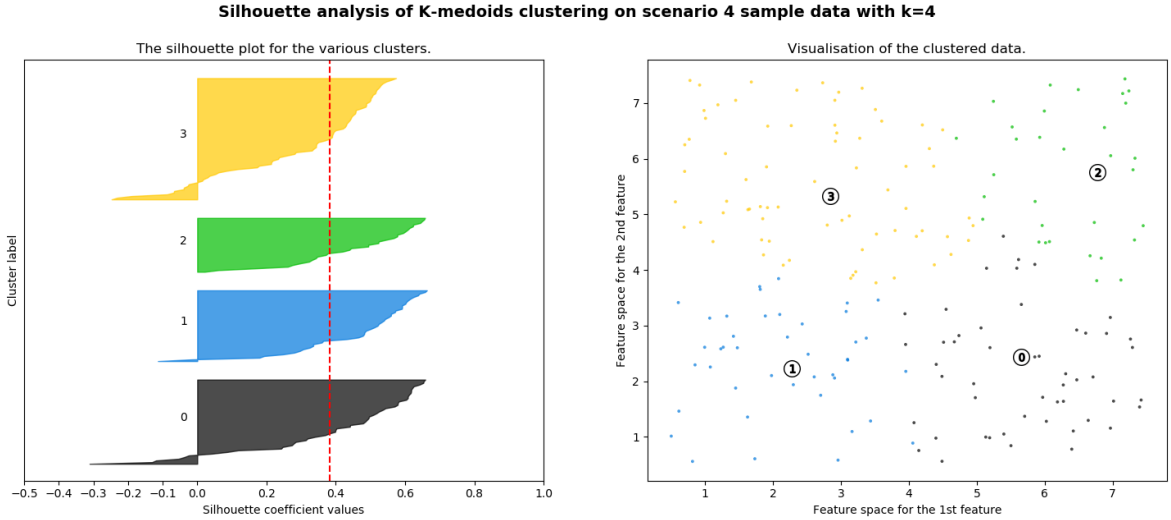

Figure 45 - Silhouette analysis of Scenario 4 with k=4

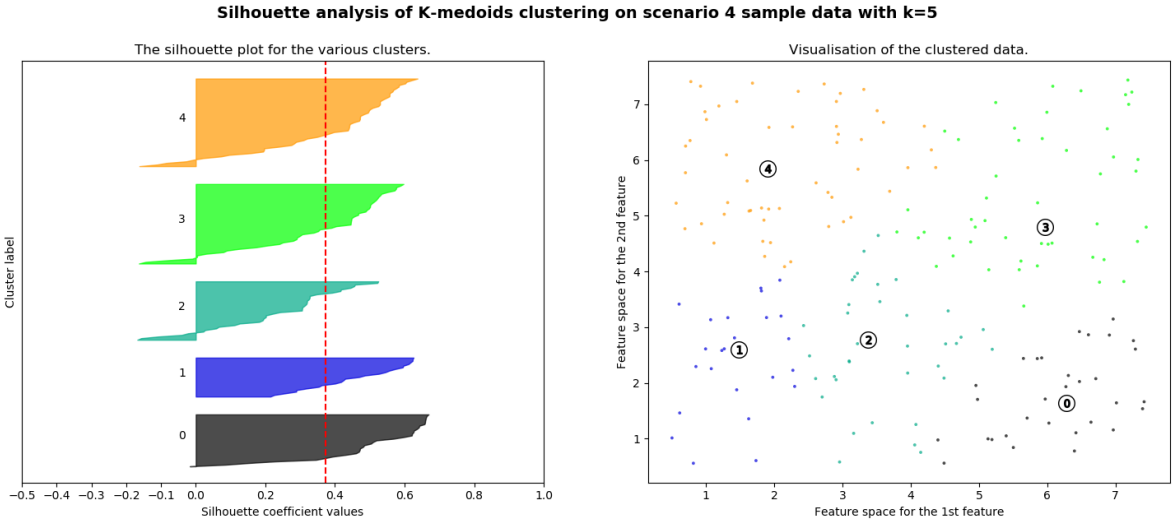

Figure 46 - Silhouette analysis of Scenario 4 with k=5

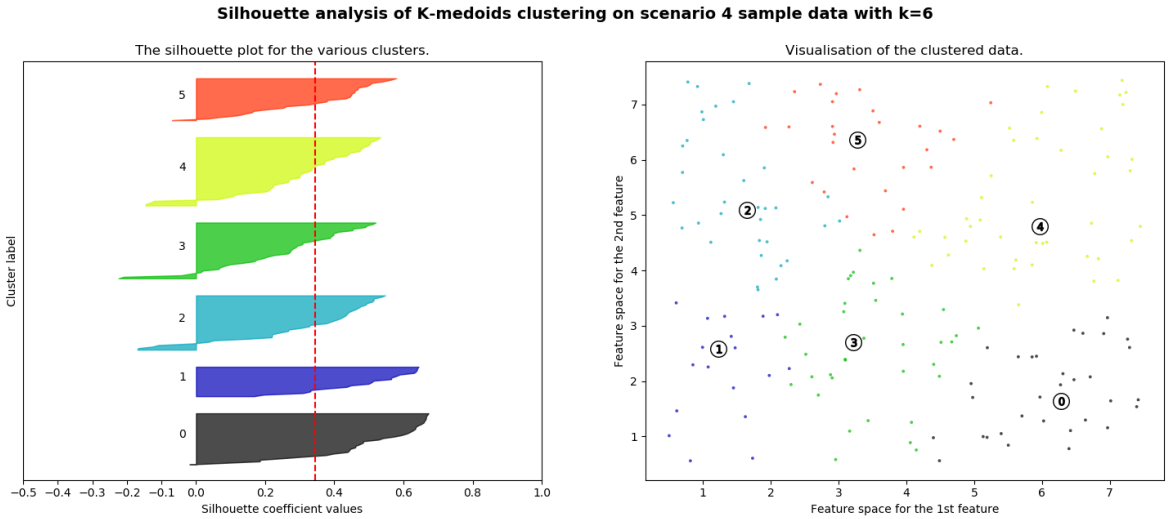

Figure 47 - Silhouette analysis of Scenario 4 with k=6

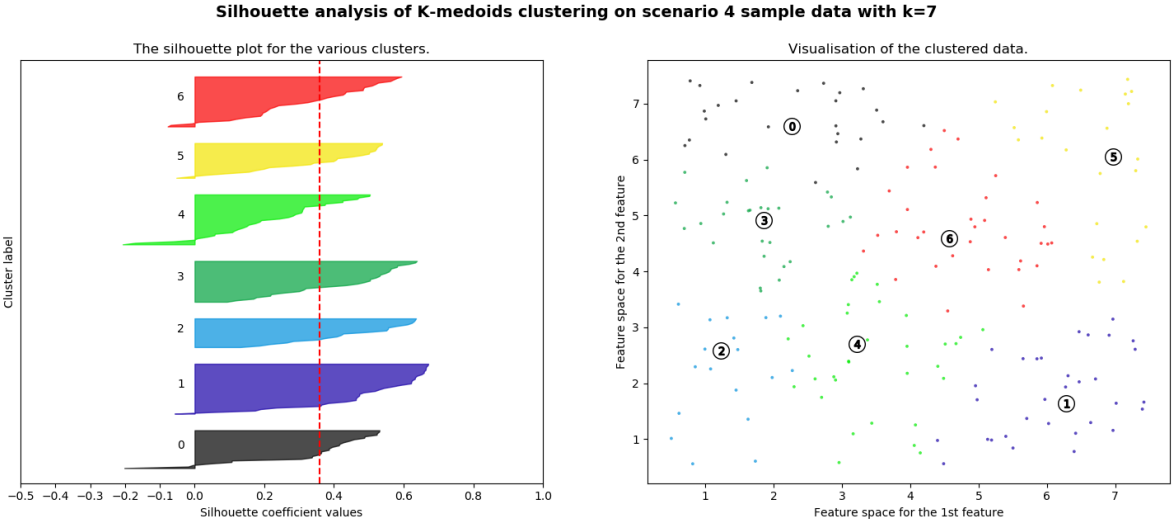

Figure 48 - Silhouette analysis of Scenario 4 with k=7

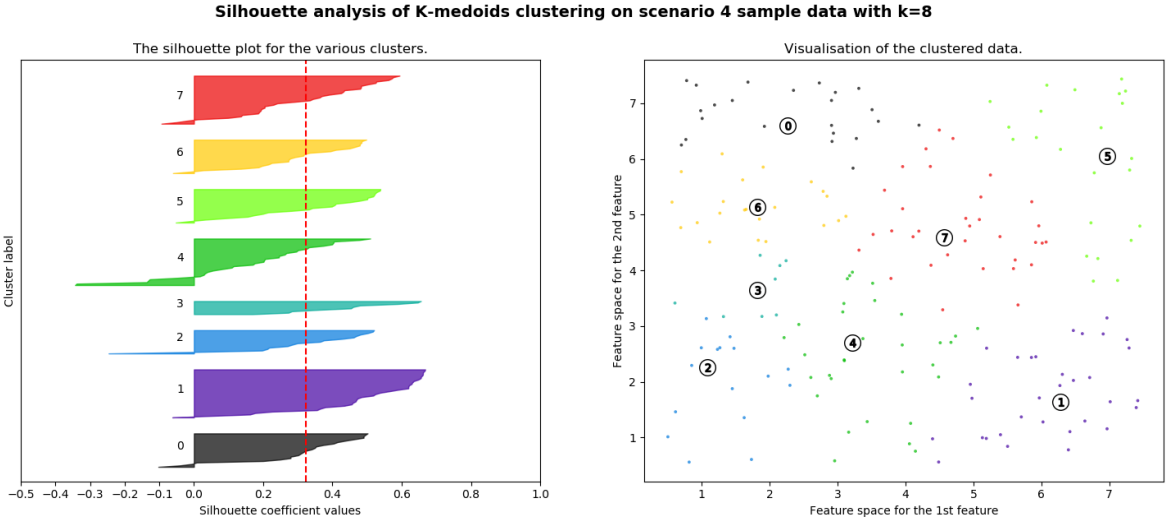

Figure 49 - Silhouette analysis of Scenario 4 with k=8

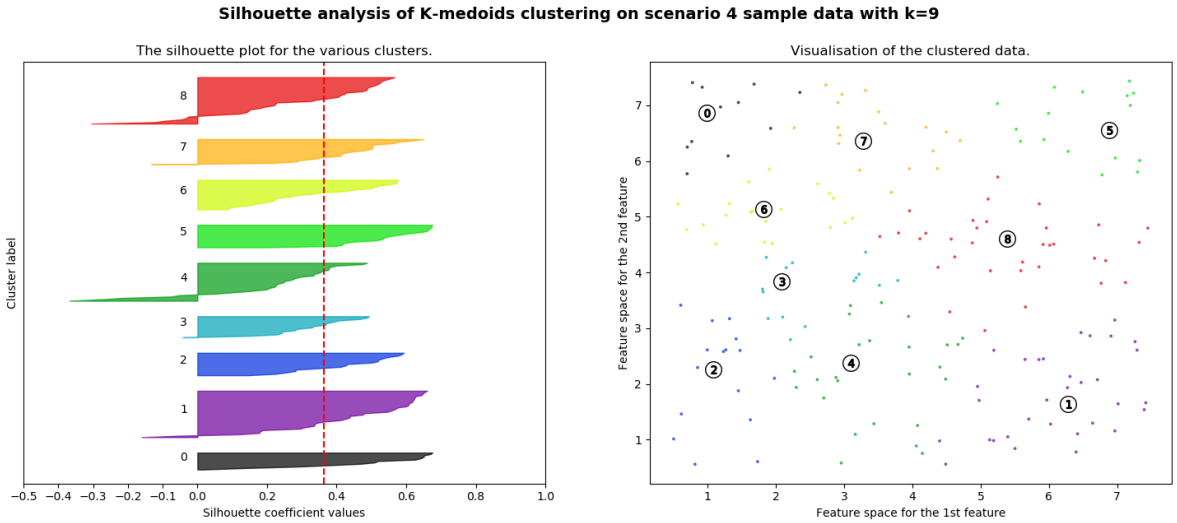

Figure 50 - Silhouette analysis of Scenario 4 with k=9

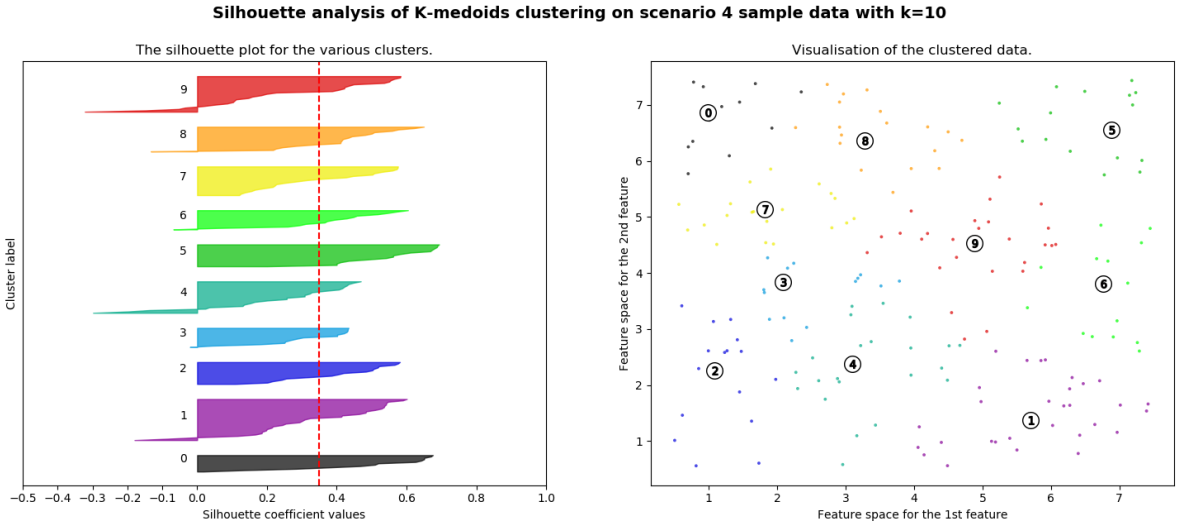

Figure 51 - Silhouette analysis of Scenario 4 with k=10

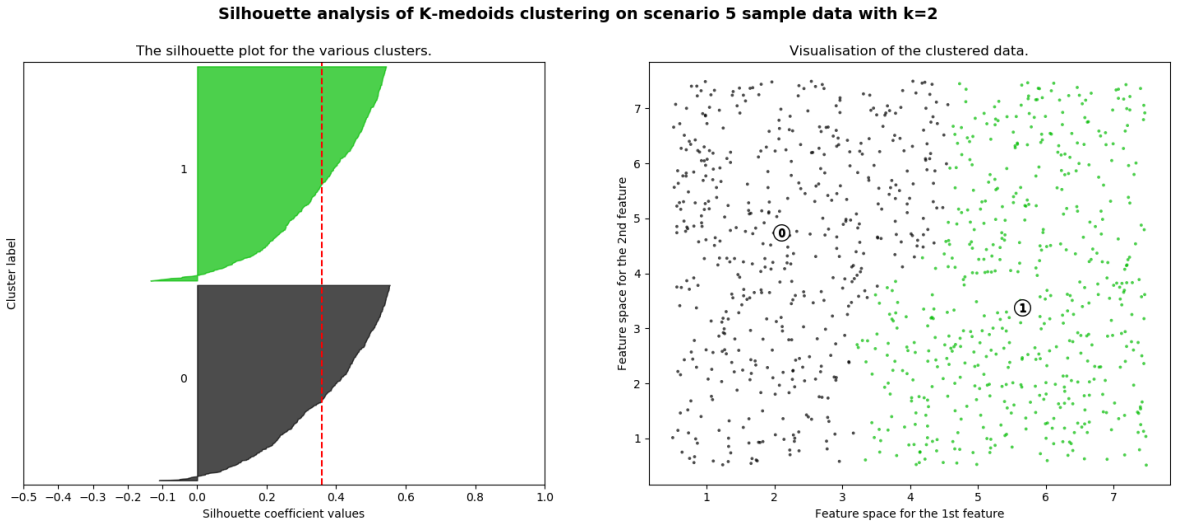

Figure 52 - Silhouette analysis of Scenario 5 with k=2

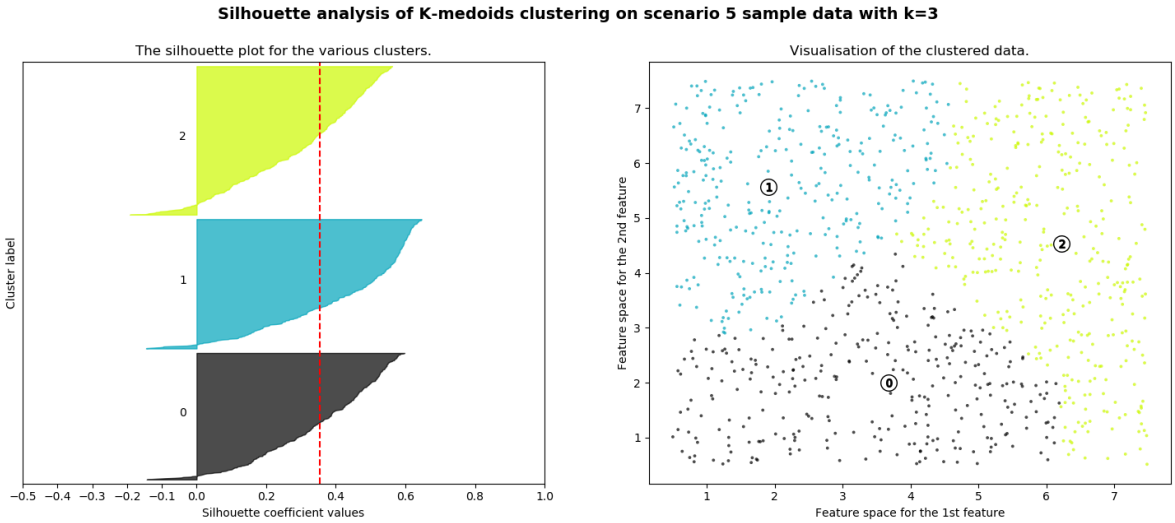

Figure 53 - Silhouette analysis of Scenario 5 with k=3

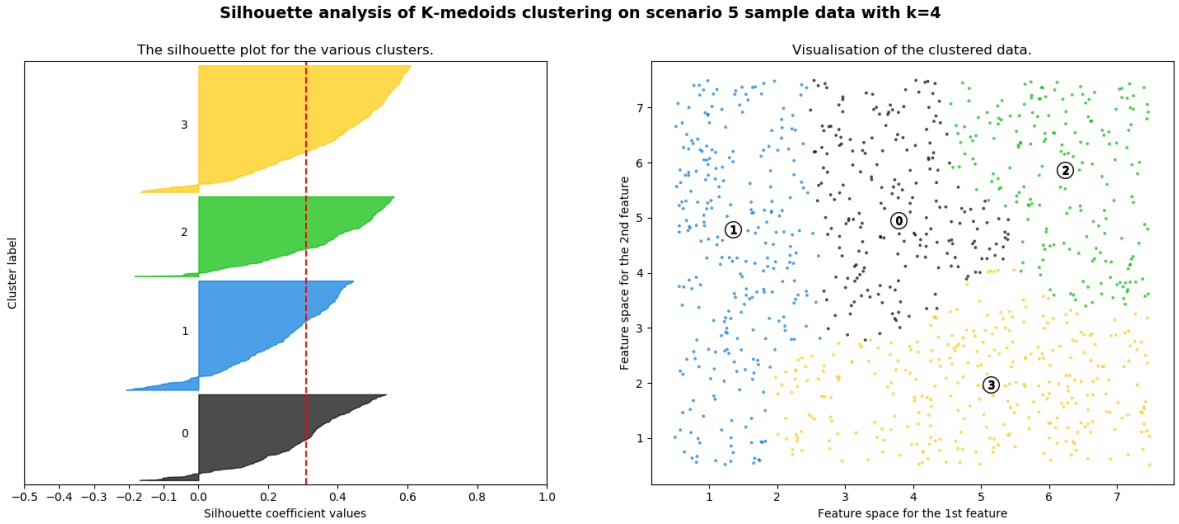

Figure 54 - Silhouette analysis of Scenario 5 with k=4

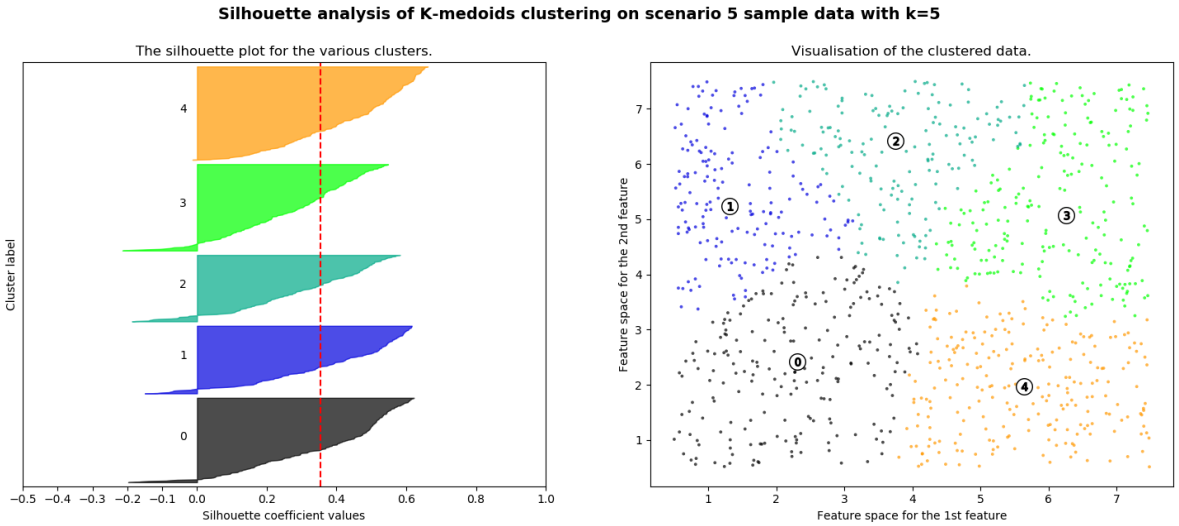

Figure 55 - Silhouette analysis of Scenario 5 with k=5

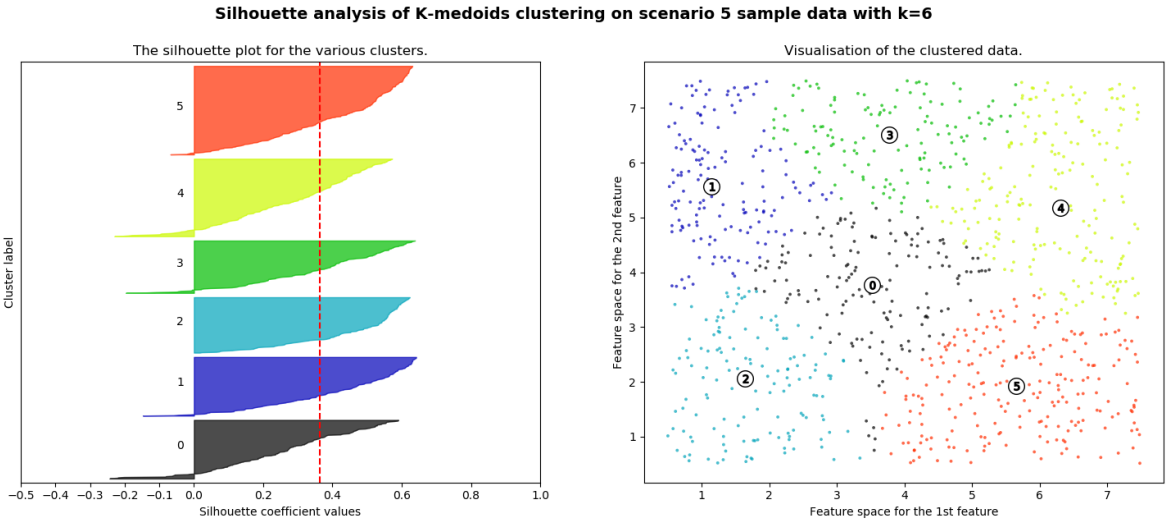

Figure 56 - Silhouette analysis of Scenario 5 with k=6

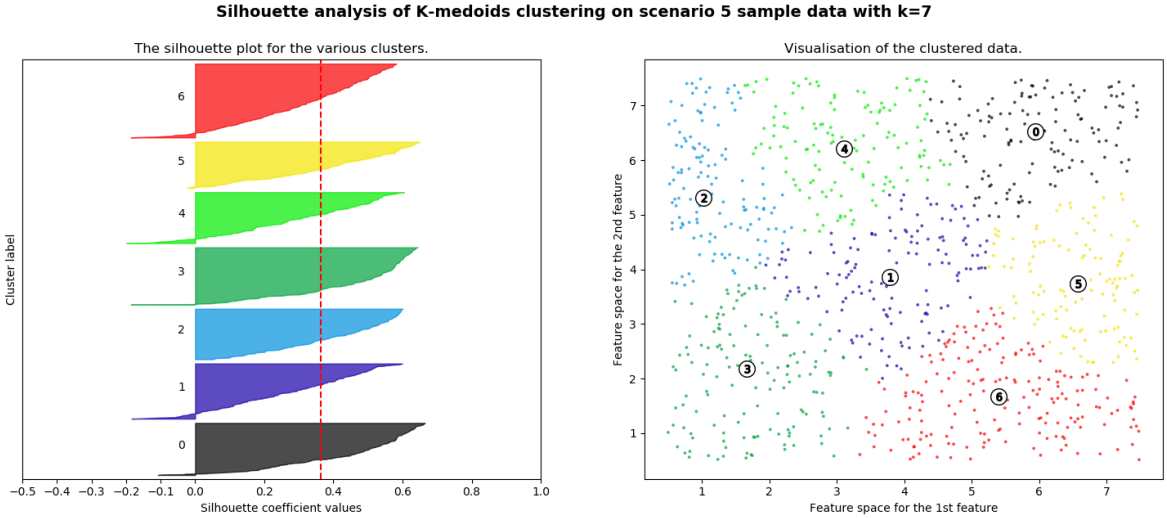

Figure 57 - Silhouette analysis of Scenario 5 with k=7

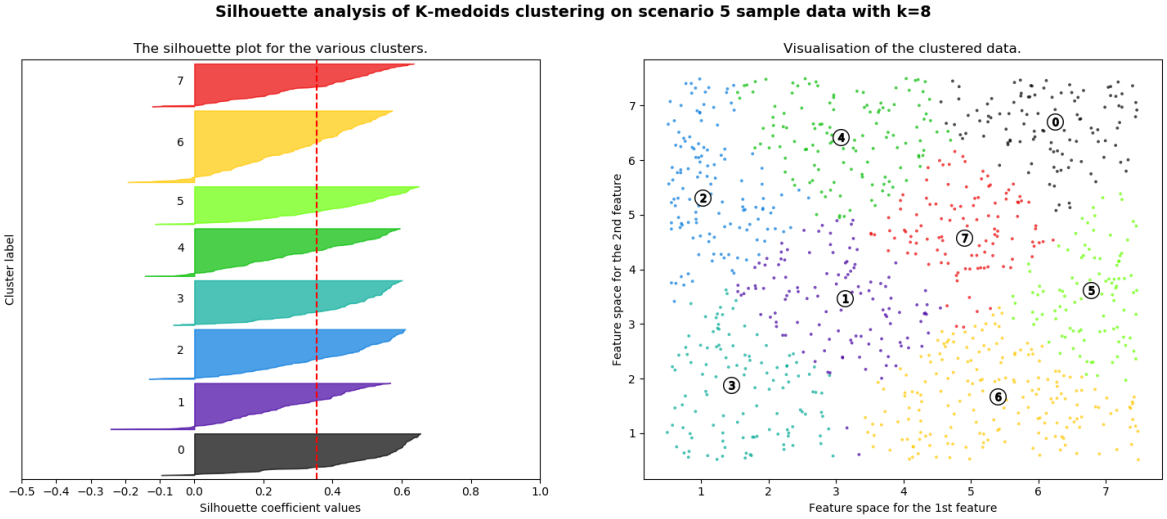

Figure 58 - Silhouette analysis of Scenario 5 with k=8

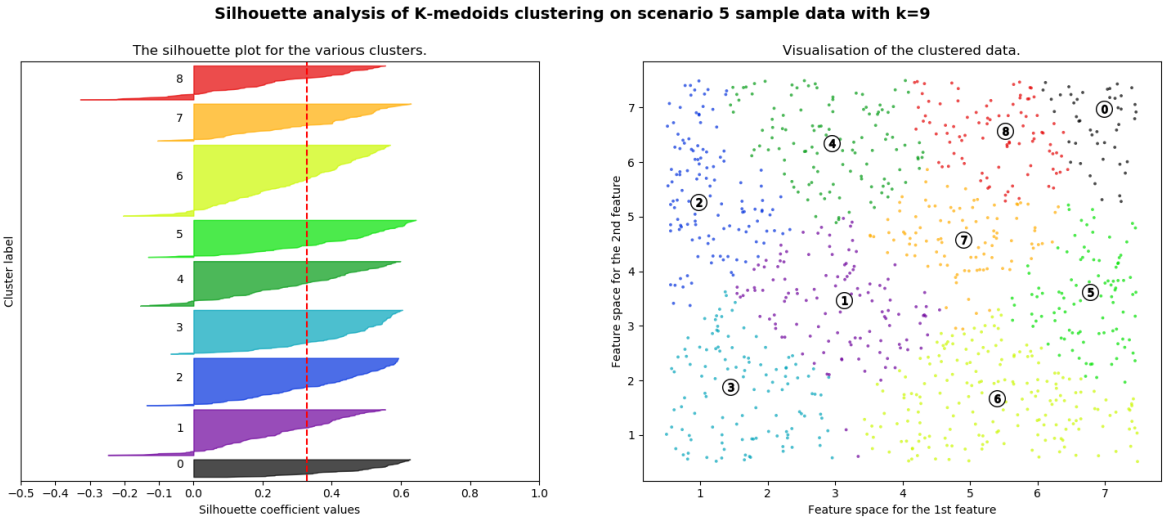

Figure 59 - Silhouette analysis of Scenario 5 with k=9

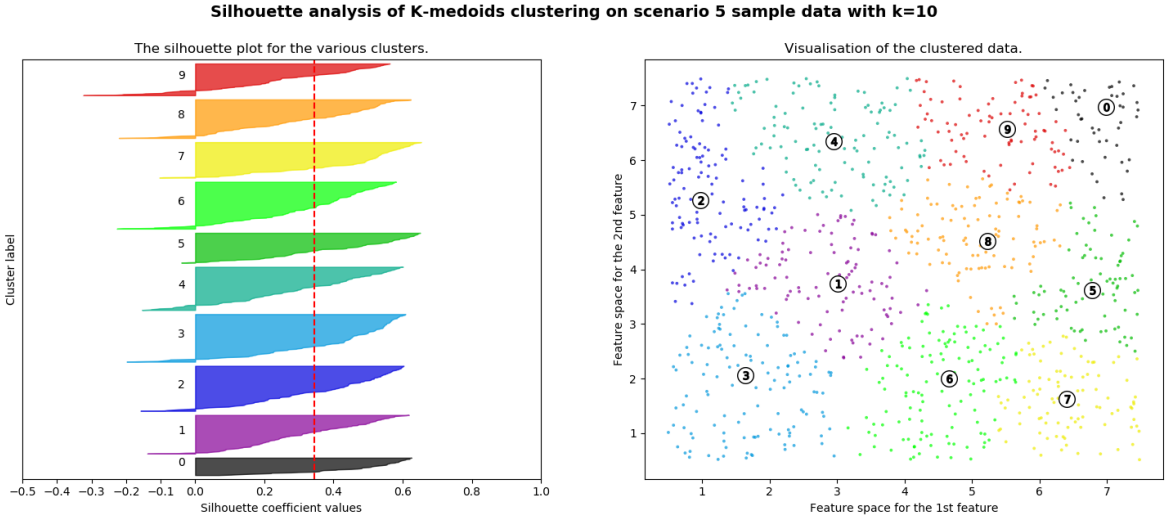

Figure 60 - Silhouette analysis of Scenario 5 with k=10

Appendix B – Reflection Document

While writing this thesis, I learned much about the scientific process as a whole – from peer reviews and opposition to choosing a research strategy and presenting coherent argumentation. The language used throughout the thesis is professional and follows common scientific guidelines - for example, capitalisation of titles and headings follow the APA Style guidelines and citations follow Harvard referencing.

To be able to conduct the research behind the thesis, I had to read many extensive scientific research papers. This was time consuming but very interesting and I think that the scientific literature presented in the introduction and extended background is highly relevant. I tried to be as concise as possible, but some sections simply had to include more information than others as they were essential to the work done in the thesis. There is some motivation for why certain pieces of literature were included, but I should have been more critical when analysing them.

Ethical aspects are discussed, although not in detail as the research did not require any interaction with other people during the writing of the thesis. Societal consequences of the artefact are, however, discussed more in-depth as they are much more interesting for this thesis.

I think that the selected scientific methods are relevant. Design science provided a very good method framework for developing the artefact, while still allowing for flexibility when choosing the data collection and analysis methods used in the different steps of the framework. However, I should have presented the details of how each method was applied to a certain problem within the design science method framework earlier than I did. The actual methods were of course used during data collection and analysis but some of them were not documented properly in the section on methodology, which led to justified remarks during my defence of the thesis. This error was of course corrected quickly afterwards.

Planning was continuously done in advance for the coming week in dialogue with my supervisor. As previously mentioned, a lot of time was spent on reviewing existing literature to build a basis for the research that was to be conducted. Although, in retrospect, some of the articles were not used in the thesis, I think it is hard to avoid since one does not know in advance what material will be useful or not. Something that could have been avoid, however, was the limited amount of time for developing the artefact. I started a bit too late, when I should have started working on developing code for the artefact maybe a week earlier. This was due to the fact that the scientific base had to be established and the research methods selected and documented before development could start, something that was not helped by the fact that I worked alone on the thesis. Nonetheless, looking back, I should have sped up the necessary preparations, so I could have started development earlier. Apart from this predicament, I think that the planning went well.

Although writing a thesis understandably requires knowledge of scientific research in general, such as that which is taught in the courses METOD and VESK, I found many other courses that I have taken to be highly relevant. A solid understanding of programming was necessary to develop the artefact, such as that taught in OOP, PROG2, PROG3 and PROP. As machine learning was a significant part of the developed artefact, I probably could not have developed it without having taken the course DAMI, as well as general knowledge about data structures and algorithm design from ALDA. The course IoT helped in understanding how the artefact can be used in real-world systems, while PARADIS laid the

foundation for understanding distributed systems. All in all, I feel that my education was highly relevant to the work done in the thesis and that I could not have done without it.

I think that the thesis is valuable for my future work and studies. I intend to study mathematics and later a master's degree in computer science or a related field, as I was intrigued by scientific research while writing this thesis. I am satisfied with the thesis and its results, as I think that it shows that I can take on a complex problem and come up with a practical solution while standing on a theoretical foundation. I was able to take a complex problem and break it down into smaller parts, solve the individual problems and then combine the solutions to solve the more general problem. The results could in some places have been presented in better ways than they currently are but were not because of technical limitations and a lack of time. I still think they are generally well presented and significant, although many improvements need to be made to the artefact itself.