Självständigt arbete på avancerad nivå

Independent degree project – second cycle

Datateknik
Computer Science

Association Rules in Parameter Tuning for Experimental Designs

Henrik Hållén
Abstract
The objective of this thesis was to investigate the possibility of using association rule algorithms to automatically generate rules for the output of a Parameter Tuning framework. The rules would be the basis for a recommendation to the user regarding which parameter space to reduce during experimentation. The parameter tuning output was generated by means of an open source project (INPUT) example program. INPUT is a tool used to describe computer experiment configurations in a framework independent input/output format. INPUT has adapters for the evolutionary algorithm framework Watchmaker and the tuning framework SPOT. The output was imported in R and preprocessed to a format suitable for association rule algorithms. Experiments were conducted on data for which the parameter spaces were discretized in 2, 5, 10 steps. The minimum support threshold was set to 1% and 3% to investigate the amount of rules over time. The Apriori and Eclat algorithms produced exactly the same amount of rules, and the top 5 rules with regards to support were basically the same for both algorithms. It was not possible at the time to automatically distinguishing useful rules. In combination with the many manual decisions during the process of converting the tuning output to association rules, the conclusion was reached to not recommend association rules for enhancing the Parameter Tuning process.

Keywords: Evolutionary Computation, Evolutionary Algorithms, Data mining, association rules, parameter tuning, INPUT, SPOT.
# Table of Contents

1 Introduction ................................................................................. 1
  1.1 Background and problem motivation .................................. 2
  1.2 Overall aim ........................................................................... 3
  1.3 Scope .................................................................................. 3
  1.4 Problem statement ............................................................... 3
  1.5 Outline ................................................................................ 3

2 Theory ..................................................................................... 5
  2.1 Evolutionary Computation .................................................. 5
    2.1.1 Elements of evolution .................................................. 6
    2.1.2 Elements of evolutionary algorithms ......................... 6
  2.2 Watchmaker framework ...................................................... 9
  2.3 Sequential Parameter Optimizations Tool (SPOT) ............... 10
  2.4 Intelligent Parameter Utilization Tool (InPUT) .................. 11
  2.5 Data Mining ......................................................................... 13
    2.5.1 Process of Data mining .............................................. 14
    2.5.2 Data mining algorithms .............................................. 15
  2.6 Association rules ................................................................. 16
    2.6.1 Frequent itemsets ....................................................... 16
    2.6.2 Evaluation measures ................................................ 17
    2.6.3 Rule inference ........................................................... 18
  2.7 Algorithms ......................................................................... 18
    2.7.1 Apriori ......................................................................... 18
    2.7.2 Eclat ............................................................................ 19

3 Methodology ............................................................................. 20

4 Experiments ............................................................................. 23
  4.1 Experimental setup .............................................................. 23
    4.1.1 Hardware setup .......................................................... 23
    4.1.2 Data generation with InPUT and SPOT ....................... 23
    4.1.3 Performance value aggregation per design ................ 23
    4.1.4 Vertical attribute pruning ........................................... 24
    4.1.5 General discretization ................................................. 25
    4.1.6 Data visualization ....................................................... 26
  4.2 Partitioned ranges in 2 discrete steps .................................. 29
    4.2.1 Minimum support threshold of 1% ............................. 30
    4.2.2 Minimum support threshold of 3% ............................. 30
  4.3 Partitioned ranges in 5 discrete steps .................................. 30
    4.3.1 Minimum support threshold of 1% ............................. 30
    4.3.2 Minimum support threshold of 3% ............................. 30
  4.4 Partitioned ranges in 10 discrete steps ................................ 30
    4.4.1 Minimum support threshold of 1% ............................ 31
    4.4.2 Minimum support threshold of 3% ............................ 31

5 Results .................................................................................... 32
  5.1 Partitioned ranges in 2 discrete steps .................................. 32
5.1.1 Minimum support threshold of 1%.................................32
5.1.2 Minimum support threshold of 3%.................................34
5.2 Partitioned ranges in 5 discrete steps.............................36
5.2.1 Minimum support threshold of 1%.................................36
5.2.2 Minimum support threshold of 3%.................................38
5.3 Partitioned ranges in 10 discrete steps............................40
5.3.1 Minimum support threshold of 1%.................................40
5.3.2 Minimum support threshold of 3%.................................41

6 Discussion.................................................................43
6.1 Ethical considerations..................................................45

7 Conclusions..............................................................47
Terminology

Abbreviations
CI  Computational Intelligence
Design  A specific set of parameters
Design space  The value ranges parameters can take
EA  Evolutionary Algorithms
EC  Evolutionary Computation
ES  Evolution Strategies
EP  Evolutionary Programming
GA  Genetic Algorithms
InPUT  Intelligent Parameter Utilization Tool
SPO  Sequential Parameter Optimization
SPOT  Sequential Parameter Optimization Tool box
1 Introduction

In certain kinds of search problems, finding a perfect solution is not the primary goal. There might not be a perfect solution or it could be very difficult to find, taking too much time or resources. The problem could be to plan a logistic route, constructing a booking schedule, optimizing a layout to some preference. Each problem has a unique set of circumstances. These can be time, cost, physical surroundings, environmental considerations etc.

If people are part of the problem, as in some scheduling problems, different people have different preferences, some are early birds and some are night owls. So when finding a solution to one of these problems, it has to be possible to compare different solutions. A good problem solution for an early bird might be a bad solution for a night owl. If the problem solution has to satisfy both an early bird and a night owl it is different again. The way to evaluate the problem solution is to use a fitness function. A fitness function is constructed so that it weights all the preferences against each other and gives the solution a performance value. Depending on the type of problem the highest or lowest performance value is considered the best.

In complex problems, such as those given above, conducting an exhaustive search and picking the best solution is not practically possible. In these cases, the use of a heuristic search algorithm is common. A heuristic algorithm differs in that it makes choices not to explore parts of the search space depending on the heuristic used.

Some types of heuristic algorithms, such as evolutionary algorithms, can themselves be configured and depending on the configuration a sufficiently good problem solution can take different amounts of time to find. An algorithm, well suited for a problem, can have a bad performance if it is badly configured. In these cases, much time can be saved by tuning the search algorithm against the specific problem. Tuning an algorithm by hand to obtain a good performance is extremely time consuming. In response to that, frameworks have been developed to tune parameters for heuristic algorithms. One such framework is the Sequential Parameter Optimization (SPO).
SPO is a framework for understanding algorithms by experimentation and consists of 4 parts. It starts with a scientific question or assumption about an algorithm (SPO-1). The question is then broken down into some statistical hypothesis that is desired to be tested (SPO-2). Experiments are performed with regards to the hypothesis (SPO-3). Lastly, the the results are analysed, conclusions are drawn and the scientific meaning is assessed (SPO-4). This is of course a short description, more details can be found in *spot: An R Package For Automatic and Interactive Tuning of Optimization Algorithms by Sequential Parameter Optimization* [1]

Sequential Parameter Optimization Toolbox (SPOT) is an implementation of the experimental (SPO-3) section in SPO for the programming language R. The language R is an open source language and environment for statistical experimentation [2]. Many statistical and graphical functions are implemented in the core language, which make it ideal to use for exploring data and for example algorithms.

The language R also has support for data mining algorithms such as association rules and frequent patterns in data [3]. Data mining is a concept of extracting knowledge or information from large amounts of data. This is sometimes referred to as a knowledge discovery process [4].

In a knowledge discovery process it is possible to distinguish three major parts; data preprocessing, actual mining, evaluation of results and knowledge presentation. Depending on what information or knowledge that is required, different kinds of algorithms are used, and depending on the algorithm there are different ways to pre-process the data. The one area of algorithms to be examined more closely in this case involves: frequent patterns and associations.

1.1 **Background and problem motivation**

Even though parameter tuning frameworks (tuners) can simplify and assist in determining good parameter configurations, problems can occur when there is a large parameter space, correlating the structure between a configuration matrix and the performance measure. If it is possible to reduce the parameter spaces, experiments would require less computational resources, and thus reduce the time and cost for experimentation.
There are several different frameworks that can be used when experimenting with heuristic algorithms. The main problem that arises is that most frameworks use a proprietary format when describing their experiments. This makes it difficult for researchers to correlate experiments performed in different frameworks. *INPUT: The Intelligent Parameter Utilization Tool* \(^5\) suggests a solution to that problem by using a “common experimental input/output format” that is able to bridge the gap between frameworks.

### 1.2 Overall aim

The purpose of this master thesis is to investigate a way to reduce the parameter space using association rules. If the user could be provided with statistically supported knowledge, then the possibility could exist to make an informed decision to reduce the parameter spaces that does not lead to improvements and thereby making the search space for the tuner smaller.

### 1.3 Scope

This thesis is restricted to finding rules suggesting a bad parameter space by running association rule algorithms on a pre-defined dataset consisting of results output from a parameter tuning framework. The tuning problem generating the output result is not investigated, because the aim is to find patterns in the results from the experiment not the experiment itself. The algorithms evaluated are *Apriori* and *Eclat* \(^6\), both available in the R language.

### 1.4 Problem statement

The primary goal is to find evidence to support the assumption that the process of parameter tuning can be improved by using association rules to reduce the parameter space (P1). The second goal is to explore different algorithms to decide which works best and produces the most interesting rules, in the parameter tuning process. The evaluations should result in a proposal for a suitable association rules algorithm to be used for noisy data as presented by parameter tuners (P2).

### 1.5 Outline

The structure of this document is arranged in the following fashion. The **Introduction** provides an overview with regards to where this paper fits in the current research it also briefly introduces the techniques that the research question revolves around. In the **Theory** chapter each relevant area is expanded
and explored, for a more sound understanding of the concepts. The **Method** chapter connects the parts discussed in the theory chapter to create a map regarding how these are used to create the experiments. The **Experiments** chapter describes the experiments in detail and what the expected results of the experiments are. In the **Results** chapter the output of the experiments is presented. In **Discussion** the results are analysed and correlated with the expectations from the Method chapter. In the **Conclusion** chapter the reasoning of the discussion chapter is formalized into recommendations and possible future actions. **Appendix A** shows the R code used to explore and experiment with the tuning framework output.
2 Theory

This section provides a background regarding frameworks, tools, suggested algorithms and evaluation measures that are relevant for these experiments.

2.1 Evolutionary Computation

This is a research area that is influenced by Darwinian theories of natural evolution. The area of research has evolved from three different but similar research areas that came into existence during the 1960s \[7\].

In California, USA, Lawrence Fogel developed an evolutionary framework called “Evolutionary Programming” to use with intelligent agents, in which the aim was to develop better finite state machines.

In Germany, at the technical university of Berlin, Ingo Re申enberg and Hans-Paul Schwefel had ideas regarding how evolutionary processes could be used to solve difficult real-value optimization problems. Algorithms based on their ideas have been called “Evolution Strategies”.

In Michigan, USA, John Henry Holland used the inspiration of evolution to design systems that self-adapt to a changing environment over time. Algorithms derived from his designs have been called “Genetic Algorithms”.

Each of the approaches was influenced by natural evolution but their focus differed. In the 1990s each field had grown to a size where it became natural to host annual conferences. During these conferences, researchers from the different areas started to review each others work and it soon became obvious to them that they should unify their research under a common name and thus “Evolutionary Computation” as a research area was born. Today there are several conferences concerning evolutionary computation but one of the biggest is the Genetic and Evolutionary Computation Conference (GECCO) \[8\].

Evolutionary Algorithms are the name for any algorithm in the area of Evolutionary Computation, independent of its sub-area or origin in the field. Today, many algorithms do not strictly belong to one of the original sub-areas but mix elements from several.
### 2.1.1 Elements of evolution

To understand how evolutionary algorithms work, it is useful to start with the general process of evolution.

The concept of evolution can be described as; the progress of a population and individuals in the population over time in a competitive environment. This short sentence implicates many aspects. There is a population of undefined size and this population changes over time. This means that new individuals come into the population while others disappear. This change in discrete steps in time can be called generations of the population. Thus from one step in time to another step in time the individuals in the population have evolved through a generation. The specific environment also set constraints on the population. Depending on the constraints of the environment and the diversity of the population, it can be said that some individuals are better adapted for that specific environment and are more successful and have a higher fitness. An individual with a higher fitness has a higher chance to survive and have offsprings.

### 2.1.2 Elements of evolutionary algorithms

To be able to use an evolutionary algorithm on a problem, the problem must be represented in a suitable format \(^9\). Common formats in which to represent problems, are arrays, which can be either binary, integer, real numbers or characters arrays. A position is able to represent a problem characteristic (genotype or phenotype). Each array is an individual and represents a solution to the problem at hand.

To generate an initial population, as many arrays as the population size are created with a random valid value in each position of the arrays.

In the population, the best suited individual are selected to act as parents. The best suited individuals are those with the highest fitness. To measure the fitness of individuals a fitness function needs to be constructed. The fitness function should calculate the individual's different qualities against the environmental constraints, and give back an objective evaluation number. Constructing the fitness function is something that must always be done, since the function is unique for each particular problem and is the single most important piece of information in order to obtain good results.
Selection is used at two different places during an evolutionary algorithm, namely, during parent selection and when choosing individuals to be in the next generation of the population. There are several algorithms to use during selections. Some of the most common are [7]:

- Uniform selection is a method where any one individual can be chosen, independent of fitness.

- Roulette-wheel (fitness-proportional) selection can best be described as having a roulette wheel and giving each individual slots on the wheel depending on the individual's fitness, the higher the fitness the more slots it receives. Then a random slot is selected and the individual belonging to that slot is chosen.

- Tournament selection randomly selects individuals into groups and then the best individuals in each group are chosen.

- Truncation selection is the method where only the individuals with the highest fitness are chosen.

When parents have been chosen, a new generation of children are produced. Children are produced by their parent by combining their genotype/phenotype constitutions with a method called crossover. Crossover is done by defining 1 or more crossover positions. Let it be stated that 1 point of crossover is defined. Splitting the parent arrays at that position results in 2 arrays per parent. The crossover is completed by taking the second array from parent 1 and attaching it to the first array of parent 2, and taking the second array from parent 2 and attaching it to the first array of parent 1. If there are more defined crossover points, the arrays are split at those positions, and the resulting segments are switched between parents in order to create the children. When the crossover is finished the children may be subjected to mutation.

A mutation is a random small change in the individual's genotype/phenotype representation. This is usually a statistical method (Gaussian distribution) for which there is a higher likelihood of the new value of the mutation being in the neighbouring values of the original value and a decreasing likelihood further away from the original value. If a genotype is represented by values between 1-20 and the current value is 10, then it is more likely to mutate to 12 than to 15 if a Gaussian distribution is used.
When all children have been produced, the population size will have grown. To keep the population at a constant level, a reduction of the individuals is required. To accomplish that, individuals who will survive to the next generation will have to be selected. To do this, one of the above mentioned selection algorithms is used.

When designing an evolutionary algorithm the rate of convergence has to be considered \[^7\]. If there is a high convergence rate, there is a chance that not enough of the solution space has been explored and the algorithm converges to a sub-optimal location. It means that some better solution does exist in the solution space, that has not been reached by the algorithm because it did not sufficiently explore the solution space. To balance the rate of convergence, different selection algorithms for parent selection and survival selection can be chosen. The different algorithms described above have different selection pressures. They range from uniform selection, having a very low pressure to the truncation selection, having a very high pressure. If truncation selection was to be chosen for both parent and survival selection then the convergence rate would be very high and would result in too narrow a search in the solution space.

When the population size is back to its original size, the population has evolved to the next generation and the process starts over again.

This brief introduction is sufficient to obtain a starting point in most evolutionary algorithms, the general points are described as:

- An initial population of a certain size is generated, usually randomly.
- New individuals are generated or bred. Individuals from the existing population are selected to act as parents for new individuals. The new individuals are made up from material of their parents (crossover). New individuals may be subjected to mutation, a slight and random change in their constitution.
- Individuals in the population are selected for removal or survival. This is usually achieved by means of a selection algorithm which will somehow match the fitness of individuals against each other.
• The process repeats with the new generation of the population.

These are the basic steps in all Evolutionary Algorithms, the difference is in how the steps are handled in the algorithm, the amount of influence that takes place based on the selection, mutation and crossover and the structural representation of the problem.

2.2 Watchmaker framework

The watchmaker framework \[^{10}\] is an open source framework in Java, used for experimenting with evolutionary algorithms. It supports many advanced scenarios of Evolutionary Algorithms. The only function that has to be implemented by the user is the fitness function since it is specific to the problem's solution space.

The framework revolves around the EvolutionEngine \[^{11}\], which is the core component. To be able to initialize the EvolutionEngine some decisions with regards to the behaviour of the algorithm must be made. There has to be means of generating a population and, later on, new individuals. This is implemented by the CandidateFactory. The CandidateFactory must match the type of problem representation, strings, lists, arrays etc. Evolutionary operations such as crossover and mutation are required to be configured. The fitness function (FitnessEvaluator) must be written and decisions made regarding selection strategies. Finally, a random number generator has to be passed to the EvolutionEngine, which should be both fast and statistically random. There are different qualities of random number generators so the options to provide the one of choice is a nice feature.

When the EvolutionEngine is initialized the population is evolved by calling the evolve method. The method takes 3 arguments. The population size, elite count (the percentage of the fittest parents to be included in the next generation), and the termination conditions.

To be able to see the evolution over time it is possible to add an EvolutionObserver. This provides the possibility to observe the progress of the population toward the final solution.
2.3 Sequential Parameter Optimizations Tool (SPOT)

The SPOT package \cite{12} for the language R \cite{2} is a toolbox used for tuning heuristic algorithms so as to find the optimal solution, depending on time and resources, for a problem design.

SPOT can be viewed to be in 4 different states:

1. Initialize
2. Run
3. Sequential
4. Report

The means by which SPOT reaches the optimal parameter setting is by evaluation. It starts out in the initialization phase by reading configuration files and then it generates a number of designs, that is, parameter values for the heuristic algorithm, which in the case of evolutionary algorithms includes population size, elite count, selection strategies, crossover points, mutation rate etc. It continues to the Run phase and runs the heuristic algorithm with each set of generated parameter values (designs) and evaluates its performance in a utility value “\( \text{Y} \)”, which is prepended to the parameter values and written to a results file.

Since heuristic algorithms are stochastic, i.e. it will not perform in exactly the same manner if run multiple times with the same parameter settings. This means that to obtain a realistic result, it will run the same design a specified number of times. This appears in the result file as lines where the only difference is the “\( \text{Y} \)” values.

When all the designs have been run, the Sequential phase starts, in which the designs are evaluated against the utility value “\( \text{Y} \)”, to determine which design has the best performance. Since each design has run multiple times, the design’s utility “\( \text{Y} \)” value is calculated from a utility function based on all the runs of a design e.g. the mean value of all the runs with a specific parameter value setting (design).

SPOT then generates a large new set of designs, from which it then selects the designs it predicts will perform well, based on the evaluations of the previous designs. It will then run the next generation of designs, evaluate these, generate and select new better designs until a stop criterion is reached. A stop
criterion could be the number of times required to run designs. When the stop criterion has been fulfilled the Report phase starts, SPOT can generate reports, plot charts, or use plugins for specific behaviours.

To use SPOT on an optimization problem, configuration is required. The CONF file specifies how the tests will be performed. A generation method is involved in order to generate designs. This involves many designs that must be run before evaluation and a prediction model to select new designs to run in the next iteration. The Region of Interest (ROI) file specifies the required parameters and the ranges for each parameter to be tested. Finally, the Algorithm Design (APD) file which holds information about the algorithm and its behaviour, is specified.

In [13] the authors discuss the usefulness of SPO on real live situations and contrasting some of its strengths against some of its weaknesses, of which one is that it only works on numerical values.

Another thing that could be perceived as a weakness is that SPOT uses its own format in the configuration files, experiment correlation with other framework can become a challenge.

2.4 Intelligent Parameter Utilization Tool (InPUT)

One of the problems with computer experiments is the difficulty of redoing and comparing results with experiments originally made by other researchers. The problem stems from the fact that experiments are often performed in different languages and frameworks and, as such, are not using a standardized description.

InPUT is an open source project that aims to be a solution with regards to this shortcoming [14]. InPUT can be seen as a neutral description and configuration interface against different tuning frameworks and languages.

InPUT can be used in several programming languages, for example, Java and C++, with Python [15] being available in the near future. It uses adapters for support in different languages which makes it easy to include support for new languages. At the moment, the Java implementation of InPUT is the most stable and it will be the one referred to in the text.

InPUT specifies all experimental data in xml files. The tool recognises two different parameter types, numerical and struc-
tural. Structural parameters can be nested to contain more subparameters. A structural parameter can also be of a type choice, where the legal choices are listed in the configuration. InPUT is comprised of three types of abstraction layers used to describe experiments, namely Design space, Designs and Code mappings.

Design space grammar, is the description of the total set of parameters and the range intervals that are to be explored. When defining the parameter space it is also possible to define inclusion or exclusion of extreme values for numerical parameters.

```xml
<DesignSpace>
  <SParam id="EA">
    ...
    <NParam id="PopSize" type="integer" inclMin="20" inclMax="100" />
    <NParam id="EliteCount" type="integer" inclMin="0" inclMax="10" />
  </SParam>
  <SChoice id="GA">
    <SParam type="integer" id="Selection" fixed="Truncation">
      <NParam id="AmountSelected" inclMin="1" inclMax="3" />
    </SParam>
    <SChoice id="Roulette" />
    <SChoice id="Rank" />
    <SChoice id="Truncation">
      <NParam type="double" id="Probability" exclMin=".5" exclMax="1" />
    </SChoice>
    <SChoice id="Tournament">
      <NParam type="double" id="Probability" exclMin=".5" exclMax="1" />
    </SChoice>
    <SChoice id="Sigma" />
    <SChoice id="StochasticSampling" />
  </SChoice>
  <SChoice id="StochasticSampling" />
</DesignSpace>
```

Figure 1: Example of a Design Space Grammar

The Design space grammar is also where choice parameters are defined, since they are similar to a numerical range representing a selection. From a design space, InPUT generates designs.

In the case of Design Space Grammar parameters, these are specified with parameter spaces, which for a numerical para-
meter is a range, and for a structural parameter is a choice with several defined values. Designs consists of distinct parameter sets, in which the numerical parameter has a distinct numerical value. The structural choice parameter has selected a choice value.

```
<in:SValue id="EA" value="GA">
  ...
  <in:NValue id="PopSize" value="86" />
  <in:NValue id="NaturalFitnessScores" value="false" />
  <in:NValue id="EliteCount" value="7" />
  <in:SValue id="Selection" value="Truncation">
    <in:NValue id="AmountSelected" value="2.0" />
    <in:NValue id="Probability" value="0.7928008776449134" />
  </in:SValue>
</in:SValue>
```

**Figure 2: Example of a Design Grammar**

InPUT uses one design space to generate multiple valid designs to be run in the framework algorithm. It can be viewed as an instantiation of values represented in the design space.

Code mapping is the process of converting the values from xml to the target language. In Java numerical values are mapped directly by autoboxing. The structural parameter must be mapped manually. This is conducted with the support of an understanding (contract) between InPUT and the user. The understanding is that all structural parameters are required to have get and set methods. Java then uses reflection to instantiate the objects at runtime.

More about InPUT can be read in [5] or at the project's open source repository [14][15]. It is here that the latest news is posted and where the code can be downloaded and in which code examples provide insights in relation to its use.

### 2.5 Data Mining

Data mining is the process of discovering useful information from large amounts of data. This is also described as a “knowledge discovery from data” [4]. It is a relatively new research area that has become interesting with the growing amount of data. It is data that are stored daily from many different areas in our society, such as financial data, observational data, computer logs etc. The list can be very long. Data mining has actually been firmly integrated into our lives without us having any
knowledge of it. This chapter will provide an overview of data mining, its structure and how it can be used.

### 2.5.1 Process of Data mining

The process of data mining can roughly be divided into the three stages of; data pre-processing, data mining and knowledge evaluation/representation.

In *Data mining: concepts and techniques* [4] the authors use an even higher granularity and separate knowledge discovery process into 7 steps:

1. Data cleaning
2. Data integration
3. Data selection
4. Data transformation
5. Data mining
6. Pattern evaluation
7. Knowledge representation

These 7 steps can still be grouped into the 3 stages of data mining. Steps 1-4 correspond to data pre-processing, step 5 is the actual mining, and steps 6-7 are the knowledge evaluation/representation.

The whole process actually starts with a quest for information of some kind. It is most likely to be an open ended question that is asked, e.g. is there a relation between items bought by customers in a retail store? If so, what is it and can it be explored to increase the sales? Another example could be finding a profile for people donating to charities. What are the defining characteristics for those making the highest donations to charities. If they are found, it is possible to make an extra effort in targeting individuals with these traits in order to obtain more large donations to a cause. It can also be applied on parameters for computer programs or algorithms; is there a relation with a sub-range of one parameter with a sub-range of another parameter that produces extraordinary good results?
When a question for some specific information has been defined, the characteristics of the supposed answer must be analysed. Depending on the solution characteristics, one or possibly several data mining algorithms can be explored to find an answer to the question.

2.5.2 Data mining algorithms
Data mining algorithms mine for specific patterns in the data. So it is important to use an appropriate algorithm for the desired information. Algorithms can be used separately or in combination. The major categories of algorithms are:

- Characterization, discrimination
- Classification, regression
- Frequent patterns, associations
- Clusters
- Outliers, anomalies

In data there can be groups with characteristics that are more interesting than others, returning to a previous example, characteristics for people making large donations to charity. To find these characteristics, data characterization algorithms can be used. If, as a contrast, how this group of people differs from another group, say those less likely to make donations, then it is possible to use discrimination algorithms.

Classification on the other hand is the process of predicting what class or category new additions to the data should be placed in. To accomplish this, sets of training data are required, where the data have already been divided into the different classes or categories. The algorithm then learns from the training set how to predict what group the new data belongs to. If the wanted prediction is a numeric value instead of a category, then regression is used. Regression builds on statistical methods and can also be used to predict distribution trends in the data.

Frequent patterns identify patterns in the data that may not be obvious to the human eye. It can be frequent itemsets, sub-patterns or sub-structures in the data. To connect these frequent patterns and thus obtain a more nuanced picture as to how they are internally connected, an association analysis can be performed. The association analysis produces rules on
the basis of the patterns for possible internal connections, it also provides a support count regarding the rules in the data and a confidence level of how well the rule fits.

Cluster analysis searches, without prior knowledge, the data for data-sets with similarities and thus creates groups. It attempts to have as much similarity in the groups as possible and as many differences as possible between the groups.

In data mining, when values in the data deviate from the norm or are missing, it is counted as noise in the data, and is usually corrected during the data pre-processing. However, in outlier analysis and anomaly detection this is the data of interest. Deviation from the norm can be very valuable information, examples of this can be credit card frauds, where someone attempts to use the credit card in a non typical way for a specific credit card holder, or intrusion detection in a computer network where a user attempts to access parts of the system where he/she has no access rights.

This chapter has acted as an overview for some different data mining techniques and their uses. In the next chapter, there will be a closer look at frequent patterns and association rules.

2.6 Association rules

Association rules involves a 2 step process. All frequent item-sets in a data frame are firstly determined and then, secondly they generate rules from these frequent itemsets. The next subchapters describes the data structure and terminology. Following this there is a short description of the two different algorithms, Apriori and Eclat, to illustrate their differences. More details can be found in [4].

2.6.1 Frequent itemsets

To be able to use data mining techniques the data is required to be in certain structure, different algorithms require different structures. Association rules algorithms require the data to be in the structure of a binary matrix. The data can be seen as being in a grid layout where the columns correspond to items/attributes and the rows to transactions/events, see Table 1. In this section the terms; items and transactions will be used. Later in the document the terms designs and parameter/attribute will be used. The resulting table or data frame, as it is called, can then be processed by the association rules algorithms.
Data to be processed by means of association rules does not always adhere to the binary matrix format and it must then be preprocessed. In preprocessing the data issues such as missing values and/or bad formatting of values, removal of columns that are of no interest for the problem instance, are also addressed.

To be able to describe how the different association rule algorithms work it is vital to know what itemsets are. To have an itemset it is firstly necessary to have a total set of items to choose from, which could be \( I = \{ I_1, I_2, I_3, \ldots, I_n \} \) as seen in Table 1. \( I \) in itself is an itemset as are any types of combinations in \( I \). The \( \{ \} \) parenthesis denotes that it is a set.

If there is \( \{ I_1, I_3 \} \), then you have a 2-itemset. Itemsets are addressed by how many members there are. An arbitrary number of members are usually referred to as 'k', hence a k-itemset is a itemset with k members. The means of achieving a frequent itemset is to analyse a data set D which can be seen as a list of transactions.

Frequent itemsets are items that often show up together in many transactions. To be frequent, itemsets have to occur in a user specified percentage of all the transactions. This is decided by the minimum support threshold. If the minimum support threshold is set to 10%, then \( \{ I_1, I_3 \} \) will be a frequent itemset, if they appear together in more than 10% of all transactions.

2.6.2 Evaluation measures

Rules and frequent itemsets are directly related. The rules are inferred from frequent itemsets.

The two most used measures of interestingness are support and confidence. They are indicators with regards to how useful and accurate the rules are. The support a rule has is measured

<table>
<thead>
<tr>
<th>DF</th>
<th>( I_1 )</th>
<th>( I_2 )</th>
<th>( I_3 )</th>
<th>( I_4 )</th>
<th>( I_5 )</th>
<th>\ldots</th>
<th>( I_n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>( x )</td>
<td>( x )</td>
<td>( x )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_2 )</td>
<td>( x )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( x )</td>
<td>( x )</td>
<td>( x )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\ldots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( t_n )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Layout of a Data frame
in how many transactions the rule can be found in, within the total set of transactions. Say that there is a rule that includes $I_1$ and $I_3$.

$I_1 \Rightarrow I_3$  
(example rule)

The support is all transactions that include both $I_1$ and $I_3$. The confidence is how much $I_3$ is present together with $I_1$. If $I_3$ is always present then there is a confidence of 100%. In a large data set the amount of rules can be overwhelming so the possibility to specify the minimum support threshold and the minimum confidence threshold limits the amount of rules to be mined as opposed to those which are merely deemed interesting. There are six more common interestingness measures that can be used; X2, lift, all_conf, max_conf, Kulczynski, cosine, these will, however, not be considered further here, but a more detailed comparison can be found in chapter 6.3 in [4].

2.6.3 Rule inference
When the frequent itemsets are found, the rule creation is a basic operation. Since the rules are created from frequent item-sets, the rules also satisfy the minimum support threshold. To generate rules the formula below is used.

$$\text{confidence}(A \Rightarrow B) = \frac{\text{support}_{\text{count}}(A \cup B)}{\text{support}_{\text{count}}(A)}$$  
(R1)

2.7 Algorithms
The algorithms for association rules are different with regards to how they generate the list of frequent itemsets. The rules are generated in the same way in both algorithms. The algorithms are described in more detail and with illustrations in [4].

2.7.1 Apriori
This is the most basic algorithm described in texts which relate to the association rules context. It uses iterations to scan through the transactions and build frequent itemsets. In the first iteration, it starts by counting each item's occurrence, then the 2-itemset in the second iteration, then the 3-itemsets the third iterations etc. It uses something called the Apriori property that assists in reducing the search space. The apriori property says "All non-empty subsets of a frequent itemset
must also be frequent”. This means that if there is a frequent itemset \{I_1, I_3, I_5\} then the subsets \{I_1, I_3\}, \{I_1, I_5\} and \{I_3, I_5\} are also frequent itemsets.

2.7.2 Eclat
While the Apriori iterates over transactions/rows, called a horizontal data format, the Eclat (equivalent class transformation) algorithm works with the items/columns, which is known as a vertical data format. Eclat starts by transforming the data from the horizontal format to a vertical format. In a vertical format, items have a list of each transaction that it belongs to, and the mining is performed by intersecting the itemsets by transactions. e.g. \(I_1\) has a list of transactions it occurs in and \(I_3\) has its own list of transactions where it occurs, and both are frequent. To receive support of an itemset \{I_1, I_3\} the intersections of \{I_1\} and \{I_3\} are counted. If it intersects more times than the minimum support threshold, then \{I_1, I_3\} is a frequent itemset. The Eclat algorithm also uses the Apriori property to reduce the search space.
3 **Methodology**

The approach is to apply an association rule algorithm to the output of a parameter tuning framework run on a problem instance. The goal is to find rules for certain parameter spaces that produce bad performance evaluations, in comparison with other designs. This is so that a recommendation can be made to remove the unproductive parameter space.

A possibility exists that the tuning frameworks selection process on new designs is so good that it masks possible bad parameter spaces. This will impair the association rule algorithm so it will not be able to find any rules with bad parameter spaces in later evaluations, since they did not pass the frameworks selection and were, thus, not used.

To use association rule algorithms to find patterns in the output from a parameter tuning framework requires a tuning problem, to produce the output. In this case, a One Max benchmark problem is chosen. The problem is to create the predefined string “HELLO WORLD” from a random selection of letters using an Evolutionary Algorithm. Since it is a benchmark problem it does not really have any real world application except for evaluating algorithms. The performance of the evolutionary algorithm can be tuned by changing one or more of its parameters. The example problem is described in more detail in the Watchmaker framework project \[10\].

The tuning framework (SPOT \[12\] in this case) generates many parameter setting combinations (designs) for the evolutionary algorithm. It will choose those that are perceived to produce a good performance, after which it runs the designs and evaluates their performances. The performance value and design parameters are written to the output file.

To make the tuning problem configuration reusable and keep the configuration readable for both machines and humans, the open source project InPUT \[14\] is used to handle all configurations. InPUT also has implemented support for the Watchmaker and SPOT frameworks, which simplifies the process of generating an output.

When the data from the tuning is available it must be converted to a format suitable for processing by the association rule
algorithm. Attributes that do not impact upon the actual performance during tuning must be removed from the tuning output.

Since an Evolutionary Algorithm is stochastic i.e. it does not have the exact same performance each time it runs, the tuning framework runs the designs several times. Thus, each design exists multiple times with different performance values in the output from the tuning framework. This is not a good format when using association rules. To remedy this, all performance values are aggregated to the mean value + standard deviation on a per design basis.

At this time a visual presentation of each parameter is created to act as a reference to the rules that are to be mined later on. In the presentation it is possible for convergence of tuner selection to be seen over time, and a distribution of good versus bad performance evaluations.

The data is now in a relatively good format but, it is still not possible to process it with association rules. The data must be in a binary matrix format. At present the parameters in the designs have either discrete or real values and these are required to be converted to true/false values.

The discretization process is performed so that parameters with discrete values are converted so that each discrete value of a parameter is converted to several parameters in the data frame. E.g. a parameter with 10 possible discrete steps is converted to 10 parameters with true/false values.

In the case in which the parameter value is a real number from a predefined range, then this is discretized to a number of subintervals and the occurrence of the value in one of the subintervals is denoted by means of true/false.

How to split a predefined range into subintervals is one of the concerns when producing association rules. If the range is split into too few subintervals then the result is too blunt. If there are too many subintervals then the algorithm will not produce any relevant rules. Different sizes of subintervals will be explored in the experiments.

Now when the data is in the form of a binary matrix the evaluation measures for the association rule algorithm are chosen. To generate rules, the evaluation measures, minimum support threshold and minimum confidence threshold are used.
Methodology

Henrik Hållén

16/04/14

The minimum support threshold is interesting since it involves an earlier concern, namely that the tuning frameworks design selection process is so effective in selecting new better designs, that unproductive parameter spaces are never explored. In attempting to circumvent this, the minimum support threshold is set low. This will produce more rules, including some uninteresting ones, but hopefully also some relevant ones. The minimum support threshold will also be increased somewhat to examine the impact on the amount of rules generated.

Another interesting factor that involves the minimum support threshold is the amount of designs from which to generate the rules. Does the association rule algorithm produce more rules at the start of the tuning process or at the end? This question could indicate that the tuning framework favours certain parameter spaces or restricts selection, making the association rule algorithm working on uneven terms. The experiments will also explore rules generated at different times from the start of the tuning process to the end.

The minimum confidence threshold is just set to 100%. If there is not complete certainty that a parameter space is bad, then it should not be removed from the exploration.

All preprocessing and experimentation is conducted in R[^2].
4 Experiments

4.1 Experimental setup

4.1.1 Hardware setup

All the experiments are run on an Acer Aspire 5830TG laptop, with an Intel Core i5 2430M 2.4 GHz and 8GB RAM. Ubuntu 13.04 64-bit.

4.1.2 Data generation with InPUT and SPOT

To generate data, the Watchmaker T uning program was used. Watchmaker T uning is an example program available from the InPUT project \[14\], that uses the Watchmaker framework, SPOT and R to allow an evolutionary algorithm to find the solution matching the target string “HELLO WORLD”. Each time the algorithm runs it generates a performance value indicating the evaluated performance of the design.

The example code was downloaded from github \[14\], and version 0.42 was used. The code was opened in Eclipse and modified to perform 5,000 runs. Otherwise, default parameters of the example program are used.

The resulting output file RES, was loaded in R to what is called a data frame. All runs of a design are printed in the results file which thus includes several identical configurations and different result evaluation values. These are reduced to a unique result evaluation value per design. The RES data frame also contains several columns that are not of interest with regards to the data mining process and are removed before running the data mining.

4.1.3 Performance value aggregation per design

To make the data processable, each design must have just one evaluation value. To achieve this, the mean value + standard deviation are calculated during preprocessing and the rows are aggregated for each specific design. This approach has the upside that it is statistically sound and it can involve some other measure instead of the mean + standard deviation. Since the problem at hand is a minimization problem, the mean value + standard deviation are used instead of only the mean value, as
this creates a more accurate value when taking outlier performance values into account.

The value in column CONFIG in the frame set shows which design the row belongs to. The “Y” value is the performance value to be aggregated, for all the rows belonging to the design. This is achieved by extracting “Y” and “CONFIG” to a separate frame, then calculating the mean value + standard deviation of “Y” with respect to “CONFIG”. The resulting frame only has one row for each design (CONFIG).

To compress the original data frame, it also only has one unique row for each design, the attributes added by the tuning framework differentiating each row are required to be removed. The “Y” and “SEED” values are removed and the frame compressed to unique values.

The two frames are combined with the “CONFIG” value as key. The resulting data frame has a unique row for each design with the mean+sd “Y” as an evaluation value. Since each design is run at least 10 times the resulting data frame is roughly one tenth of the original size, at most. In this case 455 rows.

4.1.4 **Vertical attribute pruning**

Since SPOT also includes information about the experiment setup in the RES file, running the data mining on the default file would create many rules that are of little or no interest. To minimize unnecessary rules, the attributes of no interest are pruned from the data frame. The default RES file attributes are:

1. Y
2. EA.PopSize (PopSize)
3. EA.EliteCount (ElitCount)
4. EA.GA.Selection.AmountSelected (EGS.ASelect)
5. EA.GA.Selection.Truncation.Probability (EGS.TruncProb)
7. Operators.Crossover.Points (OC.Points)

9. FUNCTION

10. DIM

11. SEED

12. CONFIG

13. STEP

The last five attributes (FUNCTION, DIM, SEED, CONFIG, STEP) related to the parameter optimizations experiments are pruned. The names in parentheses are abbreviations used in the data frame headers.

4.1.5 General discretization

Some of the parameters do not change between the experiments and are described here. Parameters that change between the experiments are described in each experiment setup.

Performance measure “Y” is the most difficult to come to terms with and it will be discretized to 5 steps. The reason for this is so it becomes possible to distinguish the best and worst designs according to their performance measures. This is the only value to be discretized by frequency instead of interval. This means that they are partitioned by how many “Y” values there are, not by the range they can take. The reason for this is in relation to the outliers in the performance value. If an outlier was found to be the worst performance value, it would probably be the only one in that interval. Whereas, when partitioning by frequency, the worst 20% of the performance measurements are related to the amount of evaluated designs and will always show the worst designs so far.

The algorithm parameters of Amount Selected and Operators Crossover Points are discretized to 3 steps, since they only have 3 discrete values to start with and this does not fit with the experiments discretization values.
4.1.6 Data visualization
In order to obtain a feel for the dataset on which the rules are mined, each parameter is visualised per tested design in a graph, and in which the x-axis are designs over time and the y-axis is the parameter space. The different design points are also distinguished by their performance values. A design in the best 20% is represented by a blue standing triangle. A design in the worst 20% is represented by a red inverted triangle. The designs in between are represented by black dots.

![Figure 3: Population size parameter per design tested](image1)

![Figure 4: Elite count parameter per design tested](image2)
Figure 5: Genetic Algorithm parameter Amount Selected per design tested

Figure 6: Genetic Algorithm parameter Truncation Probability per design tested
Association Rules in Parameter Tuning - for Experimental Designs

Henrik Hållén

Experiments

16/04/14

Figure 7: Operations Mutation Probability parameter per design tested

Figure 8: Operators Crossover Points parameter per design tested
4.2 Partitioned ranges in 2 discrete steps

This is the most simple discretization that can be made on the parameters and which is also the easiest on which to make a recommendation since each step represents a range at the upper or lower boundaries. Parameters partitioned and discretized are:

1. EA.PopSize (PopSize)
2. EA.EliteCount (ElitCount)
3. EA.GA.Selection.Truncation.Probability (EGS.TruncProb)
4. Operators.Mutation.Probability (OM.Prob)

The partitioning is, however, very crude, but should give some indication regarding which parameters are most important in the tuning process.
4.2.1 Minimum support threshold of 1%
The minimum support threshold are set to 1% to be certain that rules are produced.

4.2.2 Minimum support threshold of 3%
Increasing the minimum support threshold to 3% restricts the rules produced and gives a reference as to how the minimum support threshold impacts upon the amounts of rules generated.

4.3 Partitioned ranges in 5 discrete steps
Here the partitioning is increased to 5 discrete steps. It should produce rules with more nuances. However, it will be harder to find interesting rules since the amount of uninteresting rules will probably have increased. Parameters partitioned are as stated previously:

1. EA.PopSize (PopSize)
2. EA.EliteCount (ElitCount)
3. EA.GA.Selection.Truncation.Probability (EGS.TruncProb)
4. Operators.Mutation.Probability (OM.Prob)

4.3.1 Minimum support threshold of 1%
The minimum support threshold are set to 1% to be certain that rules are produced.

4.3.2 Minimum support threshold of 3%
Increasing the minimum support threshold to 3% restricts the rules produced and gives a reference regarding how the minimum support threshold impacts upon the amounts of rules generated.

4.4 Partitioned ranges in 10 discrete steps
These are the most discrete steps to be partitioned in these experiments. This is to create a reference regarding how the rule generation behaves in accordance with the partitioning of the parameter spaces. Parameters partitioned are as before:
1. EA.PopSize (PopSize)

2. EA.EliteCount (ElitCount)

3. EA.GA.Selection.Truncation.Probability (EGS.TruncProb)

4. Operators.Mutation.Probability (OM.Prob)


4.4.1 **Minimum support threshold of 1%**
The minimum support threshold are set to 1% to be certain that rules are produced.

4.4.2 **Minimum support threshold of 3%**
Increasing the minimum support threshold to 3% restricts the rules produced and gives a reference regarding how the minimum support threshold impacts upon the amounts of rules generated.
5 **Results**

It is important to remember, when reading the results, that there are 22 distinct steps on the x-axis and each step is 20 designs (“Rule generation per 20 designs”). A new rule set including all previous designs is generated every 20th design, which brings the last evaluated design to 4400. The last 15 designs of the total 455 are not included in the graphs.

5.1 **Partitioned ranges in 2 discrete steps**

5.1.1 **Minimum support threshold of 1%**

![Graph 1](image1.png)

**Figure 10: Rules from 2 discrete steps and support of 1%**
Association Rules in Parameter Tuning - for Experimental Designs

Henrik Hållén

Results

16/04/14

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 2: Top 5 rules from Eclat in first rule generation with regard to rule support, from 2 discrete steps and support of 1%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.075</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.075</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.075</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{EGS.ASelect}=3, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.05</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{OM.Prob}=2, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 3: Top 5 rules from Apriori in second rule generation with regard to rule support, from 2 discrete steps and support of 1%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.15</td>
</tr>
</tbody>
</table>

Table 4: Top 5 rules from Apriori in first rule generation with regard to rule support, from 2 discrete steps and support of 1%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.075</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.075</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.075</td>
</tr>
<tr>
<td>(\text{PopSize}=2, \text{ElitCount}=1, \text{EGS.ASelect}=3, \text{OC.Points}=2) =&gt; (Y=5)</td>
<td>0.05</td>
</tr>
<tr>
<td>(\text{ElitCount}=1, \text{EGS.ASelect}=3, \text{OC.Points}=2, \text{OC.Prob}=1) =&gt; (Y=5)</td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 5: Top 5 rules from Eclat in second rule generation with regard to rule support, from 2 discrete steps and support of 1%

33
5.1.2 Minimum support threshold of 3%

Figure 11: Rules from 2 discrete steps and support of 3%

Table 6: Top 5 rules from Apriori in first rule generation with regard to rule support, from 2 discrete steps and support of 3%
### Results

16/04/14

#### Table 7: Top 5 rules from Apriori in second rule generation with regard to rule support, from 2 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopSize=2, ElitCount=1, OC.Points=2 =&gt; Y=5</td>
<td>0.075</td>
</tr>
<tr>
<td>ElitCount=1, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.075</td>
</tr>
<tr>
<td>PopSize=2, ElitCount=1, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.075</td>
</tr>
<tr>
<td>ElitCount=1, EGS.ASelect=3, OC.Points=2 =&gt; Y=5</td>
<td>0.05</td>
</tr>
<tr>
<td>ElitCount=1, OM.Prob=2, OC.Points=2 =&gt; Y=5</td>
<td>0.05</td>
</tr>
</tbody>
</table>

#### Table 8: Top 5 rules from Eclat in first rule generation with regard to rule support, from 2 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopSize=2, ElitCount=1, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.15</td>
</tr>
<tr>
<td>ElitCount=1, OC.Points=2 =&gt; Y=5</td>
<td>0.15</td>
</tr>
<tr>
<td>ElitCount=1, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.15</td>
</tr>
<tr>
<td>PopSize=2, ElitCount=1, OC.Prob=1 =&gt; Y=5</td>
<td>0.15</td>
</tr>
<tr>
<td>PopSize=2, ElitCount=1, OC.Points=2 =&gt; Y=5</td>
<td>0.15</td>
</tr>
</tbody>
</table>

#### Table 9: Top 5 rules from Eclat in second rule generation with regard to rule support, from 2 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>PopSize=2, ElitCount=1, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.075</td>
</tr>
<tr>
<td>ElitCount=1, OC.Points=2 =&gt; Y=5</td>
<td>0.075</td>
</tr>
<tr>
<td>PopSize=2, ElitCount=1, OC.Points=2 =&gt; Y=5</td>
<td>0.075</td>
</tr>
<tr>
<td>PopSize=2, ElitCount=1, EGS.ASelect=3, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.05</td>
</tr>
<tr>
<td>ElitCount=1, EGS.ASelect=3, OC.Points=2, OC.Prob=1 =&gt; Y=5</td>
<td>0.05</td>
</tr>
</tbody>
</table>
5.2 Partitioned ranges in 5 discrete steps

5.2.1 Minimum support threshold of 1%

Table 10: Top 5 rules from Apriori in first rule generation with regard to rule support, from 5 discrete steps and support of 1%
### Results

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ElitCount=1} \Rightarrow {Y=5}</td>
<td>0.075</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{ElitCount=1,OM.Prob=3} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{ElitCount=1,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{ElitCount=1,EGS.ASelect=3} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Table 11: Top 5 rules from Apriori in second rule generation with regard to rule support, from 5 discrete steps and support of 1%**

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ElitCount=1} \Rightarrow {Y=5}</td>
<td>0.15</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,EGS.ASelect=3, OM.Prob=3,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.1</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,OM.Prob=3,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.1</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,EGS.ASelect=3,OM.Prob=3} \Rightarrow {Y=5}</td>
<td>0.1</td>
</tr>
<tr>
<td>{ElitCount=1,EGS.ASelect=3,OM.Prob=3,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.1</td>
</tr>
</tbody>
</table>

**Table 12: Top 5 rules from Eclat in first rule generation with regard to rule support, from 5 discrete steps and support of 1%**

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ElitCount=1} \Rightarrow {Y=5}</td>
<td>0.075</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,EGS.ASelect=3, OM.Prob=3,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,EGS.ASelect=3,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,OM.Prob=3,OC.Points=2} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1,EGS.ASelect=3,OM.Prob=3} \Rightarrow {Y=5}</td>
<td>0.05</td>
</tr>
</tbody>
</table>

**Table 13: Top 5 rules from Eclat in second rule generation with regard to rule support, from 5 discrete steps and support of 1%**
5.2.2 Minimum support threshold of 3%

### Figure 13: Rules from 5 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>Apriori</th>
<th>Eclat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>392</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>41</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 14: Top 5 rules from Apriori in first rule generation with regard to rule support, from 5 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ElitCount=1} =&gt; {Y=5}</td>
<td>0.15</td>
</tr>
<tr>
<td>{ElitCount=1,OM.Prob=3} =&gt; {Y=5}</td>
<td>0.1</td>
</tr>
<tr>
<td>{EGS.ASelect=3,OM.Prob=3} =&gt; {Y=5}</td>
<td>0.1</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1} =&gt; {Y=5}</td>
<td>0.1</td>
</tr>
<tr>
<td>{ElitCount=1,EGS.ASelect=3} =&gt; {Y=5}</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### Table 15: Top 5 rules from Apriori in second rule generation with regard to rule support, from 5 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{ElitCount=1} =&gt; {Y=5}</td>
<td>0.075</td>
</tr>
<tr>
<td>{PopSize=5,ElitCount=1} =&gt; {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{ElitCount=1,OM.Prob=3} =&gt; {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{ElitCount=1,OC.Points=2} =&gt; {Y=5}</td>
<td>0.05</td>
</tr>
<tr>
<td>{ElitCount=1,EGS.ASelect=3} =&gt; {Y=5}</td>
<td>0.05</td>
</tr>
</tbody>
</table>
Association Rules in Parameter Tuning - for Experimental Designs

Henrik Hållén

Results

16/04/14

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ElitCount=1} =&gt; {Y=5}</code></td>
<td>0.15</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,EGS.ASelect=3,OM.Prob=3,OC.Points=2} =&gt; {Y=5}</code></td>
<td>0.1</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,OM.Prob=3,OC.Points=2} =&gt; {Y=5}</code></td>
<td>0.1</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,EGS.ASelect=3,OM.Prob=3} =&gt; {Y=5}</code></td>
<td>0.1</td>
</tr>
<tr>
<td><code>{ElitCount=1,EGS.ASelect=3,OM.Prob=3,OC.Points=2} =&gt; {Y=5}</code></td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 16: Top 5 rules from Eclat in first rule generation with regard to rule support, from 5 discrete steps and support of 3%

<table>
<thead>
<tr>
<th>rules</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>{ElitCount=1} =&gt; {Y=5}</code></td>
<td>0.075</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,EGS.ASelect=3,OM.Prob=3,OC.Points=2} =&gt; {Y=5}</code></td>
<td>0.05</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,EGS.ASelect=3,OC.Points=2} =&gt; {Y=5}</code></td>
<td>0.05</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,OM.Prob=3,OC.Points=2} =&gt; {Y=5}</code></td>
<td>0.05</td>
</tr>
<tr>
<td><code>{PopSize=5,ElitCount=1,EGS.ASelect=3,OM.Prob=3} =&gt; {Y=5}</code></td>
<td>0.05</td>
</tr>
</tbody>
</table>

Table 17: Top 5 rules from Eclat in second rule generation with regard to rule support, from 5 discrete steps and support of 3%
5.3 Partitioned ranges in 10 discrete steps

5.3.1 Minimum support threshold of 1%

Table 19: Top 5 rules from Apriori in second rule generation with regard to rule support, from 10 discrete steps and support of 1%
Association Rules in Parameter Tuning - for Experimental Designs

Henrik Hållén

Results

5.3.2 Minimum support threshold of 3%

Figure 15: Rules from 10 discrete steps and support of 3%
Association Rules in Parameter Tuning - for Experimental Designs
Henrik Hållén

Results
16/04/14

rules support

{ElitCount=1} => {Y=5} 0.1
{ElitCount=1,EGS.ASelect=3} => {Y=5} 0.1
{ElitCount=1,OC.Points=2} => {Y=5} 0.1
{ElitCount=1,EGS.ASelect=3,OC.Points=2} => {Y=5} 0.1
{ElitCount=2} => {Y=5} 0.05

Table 22: Top 5 rules from Apriori in first rule generation with regard to rule support, from 10 discrete steps and support of 3%

rules support

{ElitCount=1} => {Y=5} 0.05
{OM.Prob=9} => {Y=5} 0.05
{OC.Prob=2} => {Y=5} 0.05
{ElitCount=1,OC.Points=2} => {Y=5} 0.05
{ElitCount=1,EGS.ASelect=3} => {Y=5} 0.05

Table 23: Top 5 rules from Apriori in second rule generation with regard to rule support, from 10 discrete steps and support of 3%

rules support

{ElitCount=1} => {Y=5} 0.05
{EGS.ASelect=3,OC.Points=2} => {Y=5} 0.1
{ElitCount=1,OC.Points=2} => {Y=5} 0.1
{ElitCount=1,EGS.ASelect=3} => {Y=5} 0.1
{ElitCount=1} => {Y=5} 0.1
{PopSize=5,ElitCount=2,EGS.ASelect=2,EGS.TruncProb=1,OM.Prob=10,OC.Points=3,OC.Prob=9} => {Y=5} 0.05

Table 24: Top 5 rules from Eclat in first rule generation with regard to rule support, from 10 discrete steps and support of 3%

rules support

{OM.Prob=9,OC.Points=2,OC.Prob=2} => {Y=5} 0.05
{OM.Prob=9,OC.Points=2} => {Y=5} 0.05
{OC.Points=2,OC.Prob=2} => {Y=5} 0.05
{OC.Prob=2} => {Y=5} 0.05
{ElitCount=1,EGS.ASelect=3,OC.Points=2} => {Y=5} 0.05

Table 25: Top 5 rules from Eclat in second rule generation with regard to rule support, from 10 discrete steps and support of 3%
Discussion

Looking at the results from the experiments, there are a few observations that can be made.

The most obvious to note is that both the Apriori and the Eclat algorithms produce exactly the same number of rules. Inspecting the top 5 rules from each algorithm, it can be seen that the rules also appear to match. The differences in the ordering of rules between the algorithms can be that, aside from being sorted on support count, the sub-ordering within the same support count differs. This is the probable reason as to why a rule might be present in the top 5 in one of the algorithms but not in another. To relate to the problem statement (P2) in the Introduction, no recommendation can be made as to the use of the either Apriori or Eclat, since, in the rules they generate they do not appear to differ.

There are rather substantial differences in the rule count between the minimum support thresholds of 1% and 3%. However, comparing the rules between 1% support and 3% support, it can be stated that they are basically the same. This is probably because the minimum support of 3% when comparing it to 1%, cuts more of the tail rules, namely those at the bottom of the list. Hence, there is a difference in rule count but not the in top 5 rules.

Where the minimum support threshold is 1%, there is also a significant drop at the 6th rule generation. This could be explained based on the number of designs which have been evaluated having exceeded 100. Previously, it was assumed to be sufficient to find one frequent itemset and mine rules from it, in order for them to appear in the result. However, after a second itemset, it becomes necessary to support the found rule. If looking closer at the rule count, it can be noticed that the count drops after every 5th generation. This correlates directly to the 1% minimum support threshold. A similar pattern is not as easily recognised in the experiments with the 3% minimum support threshold.

There is no direct evidence that the tuner selection reduces the rules with a bad parameter space in later evaluations, but the dropping rule count in later evaluations indicates that it is a possibility.
Comparing the rule count between the different partitions for the minimum support threshold of 1%, reveals another interesting observation. The rule count is, as expected, the 5 first generations, more discrete steps more rules. But after that, the partition with 5 discrete steps has the most rules. This suggests that the partitioning is important and it should also be possible to tune in order to optimize performance in accordance to the number of rules generated.

An alternative way to partition the parameters based on distribution was suggested at a later stage and involves the use of an intermediate step, where the distribution of values in a parameter space is analysed and used as a basis regarding how to divide it into discrete steps before running the algorithms. This approach has the merit that it uses a unique number of discrete steps for each separate parameter, and it should be possible to automatise the discretization of parameters. The difficult part in this approach is how to conduct the analysis of the distribution. That is not clear at this point in time.

Looking at the usability of the rules generated, the problem of this approach becomes evident. Going back to the problem statement (P1) in the Introduction, the goal was to investigate the use of association rules to identify a bad parameter space, so that the parameter space can be reduced.

The reduction of parameter space has to be performed on a parameter basis. The reduction also has to be from one of the sides of the parameter space, upper or lower boundaries. If a span is identified in the middle of a parameter space, and the desire was for it to be removed, the experiments must be forked, one experiment running on the upper space and one running on the lower space. Imagine such holes in several parameter spaces, this would result in a combinatorial nightmare of experiments. However combined rules can still be valid if they have the support of other rules explicitly identifying the parameter spaces at the boundaries.

Examining the rules, an interesting rule in the 5-discrete steps experiment can be found. The rule:

\[
\{ \text{ElitCount} = 1 \} \Rightarrow \{ Y = 5 \}
\]  

states that the lowest 20% of the ElitCount parameter does not produce any good results. Cross-checking with the data visualization in Figure 4, it appears that the rule is correct.
A concern during experimentation was that no rules with inequalities were found. Correlating the ElitCount rule from 5-discrete steps to the experiment with 10-discrete steps it can be seen that there are 2 rules matching the one in 5-discrete steps experiment seen in Table 22:

\[ \{ \text{ElitCount} = 1 \} \implies \{ Y = 5 \} \quad (R3) \]
\[ \{ \text{ElitCount} = 2 \} \implies \{ Y = 5 \} \quad (R4) \]

However, it should be noted that the rule support for the 2 rules in 10-discrete steps are different, see Table 22. This explains why no equalities are found in the rules, since the probability that 2 adjacent discretization steps will have the same rule and support, intuitively must be very small.

Stepping back and reviewing the aim of the project, which was to provide an automatic feature suggesting that bad parameter spaces be removed from experiments. Taking into account all the manual decisions made when preparing the data, and the difficulty evaluating the generated rules, the conclusion drawn is that association rule algorithms are not the correct algorithms for this problem, and can thus not be recommended.

6.1 Ethical considerations

Front-line scientists and experimenters engaged in research can be caught up in the details of their research and they may lose track of the bigger picture. It is important to consider the ethical implications the research will have on a larger society.

In [16] the issue of ethics in data mining is raised. One of the considerations is that it can be used as a basis for discrimination. Personal characteristics can be labelled as unwanted by insurance companies thus giving certain individuals higher costs. Personal integrity can be an issue if the anonymizing of the data has been conducted in a poor manner and reidentification of individuals are made possible.

The experiments in this thesis do not appear to have any foreseeable direct ethical pitfalls.

However! It is worth remembering a very important distinction, that it is very easy to loose track of the bigger picture.

In the eagerness in relation to discovering whether something is possible and can be achieved, the more important question
regarding whether or not it should be pursued, is often forgotten.
Conclusions

Using association rules on parameter tuning data requires many manual decisions. Reducing the dimensions of the tuning of output data to fit the relevant parameters could be a possibility as all problems, algorithms and tuning parameters are available in the InPUT format of the experiment description. However, making decisions as to how to discretize the parameters, it is necessary for this to be investigated manually. In addition, the choice with regards to minimum support threshold is a manual decision that has a significant influence on the results. An interpretation of the rules in order to determine those which are of interest is no trivial task to automatize.

The conclusion is that association rule algorithms are not suitable in relation to enhancing parameter tuning frameworks, as too many manual decisions are required to be made.

During the progress of the project, the realization has come that the problem is more suited to a classification algorithm as, in the end, designs with certain parameter spaces must be classified as not desired.

Future work could be to investigate the use of classification algorithms, for example, RandomForest for use as an enhancement to the tuning process.
References

Appendix A: R code for evaluation of association rules

# Author: Henrik Hållén
# Desc: To create and analyze association rules from the output.res file generated by InPUT and SPOT

*******************************************************************************

AggregateOnConfig <- function(df){
  #Dimension reduction by aggregating mean values by designs
  df.y <- df[,c("Y","CONFIG")]
  Y.mean <- tapply(df.y$Y, df.y$CONFIG, mean)
  Y.sd <- tapply(df.y$Y, df.y$CONFIG, sd)
  #Remove attributes making each row unique
  df$Y <- NULL
df$SEED <- NULL
  #Compress table so each row is one design
  u_df <- unique(df)
  #Recombine each design with it's mean Y
  agg_df <- cbind(Y.mean + Y.sd, u_df)
  return(agg_df)
}

PrintPerParameter <- function(df, y.steps){
  # Prints the total sets of design evaluations per parameter
  df <- MakeDiscreteY(df, y.steps)
  for(i in 1:length(df)){
    filename <- paste(dimnames(df)[[2]][i], "svg", sep=".")
    svg(filename, width=5.55, height=4)
    par(mar=c(4.5,4.2,1,1))
    plot(1:nrow(df), df[[i]], xlab="Design", ylab=dimnames(df)[[2]][i],
         cex=0.6, pch=rep(c(24,21,21,21,25)[df$Y],
         col=c("blue","black","black","black","red")[df$Y],
         bg=c("blue","black","black","black","red")[df$Y])
    dev.off()
  }
}

MakeDiscreteY <- function(res.n, steps){
  res.n[1] <- discretize(res.n[[1]], "frequency",
                         steps, labels=1:steps)
  res.n[1] <- as.ordered(res.n[[1]])
  return(res.n)
}

MakeDiscrete <- function(res.n, steps){
  #Discretization of numerical values.
  res.n[2] <- discretize(res.n[[2]],
Appendix A: R code run during evaluations

16.04.14

GetRulesApriori <- function(df, supp, conf){
  #Converting the data frame to transactions to be mined.
  trans <- as(df, "transactions")
  #Rules for apriori algorithm
  rules <- apriori(
    trans,
    parameter=list( 
      support=supp,
      confidence=conf)
  )
  rules <- subset(rules, subset = rhs %ain% c("Y=5"))
  return(sort(
    rules,
    decreasing = TRUE,
    na.last = NA,
    by = "support")
  )
}

GetRulesEclat <- function(df, supp, conf){
  #Converting the data frame to transactions to be mined.
  trans <- as(df, "transactions")
  #Rules for the eclat algorithm
  itemsets.eclat <- eclat(df, parameter = list(support = supp))
  rules.e <- ruleInduction(year = "1998")
}

"interval",
  steps,
  labels=1:steps)
res.n[2] <- as.ordered(res.n[[2]])
res.n[3] <- discretize(
  res.n[[3]],
  "interval",
  steps,
  labels=1:steps)
res.n[3] <- as.ordered(res.n[[3]])
res.n[4] <- factor(
  res.n[[4]],
  levels=1:3,
  labels=1:3,
  ordered=TRUE)
res.n[5] <- discretize(
  res.n[[5]],
  "interval",
  steps,
  labels=1:steps)
res.n[5] <- as.ordered(res.n[[5]])
res.n[6] <- discretize(
  res.n[[6]],
  "interval",
  steps,
  labels=1:steps)
res.n[6] <- as.ordered(res.n[[6]])
res.n[7] <- factor(
  res.n[[7]],
  levels=1:3,
  labels=1:3,
  ordered=TRUE)
res.n[8] <- discretize(
  res.n[[8]],
  "interval",
  steps,
  labels=1:steps)
res.n[8] <- as.ordered(res.n[[8]])
return(res.n)
}
itemsets.eclat,
transactions = trans,
confidence = conf,
control = list("ptree"))
rules.e <- subset(rules.e, subset = rhs %in% c("Y=5"))
return(sort(
    rules.e,
    decreasing = TRUE,
    na.last = NA,
    by = "support"))
}

Experiment123Run <- function(res.n){
  res.backup <- res.n
  ev.count <- floor(nrow(res.backup)/20)
  rule.2.count <- matrix(0, ev.count, 2)
  rule.5.count <- matrix(0, ev.count, 2)
  rule.10.count <- matrix(0, ev.count, 2)

  #Partitions Data frame to simulate evaluation during runtime
  i=1
  while((i*20) < nrow(res.backup)){
    res.n <- head(res.backup, i*20)
    #Splits the Y values by 20% frequencies,
    res.n <- MakeDiscreteY(res.n, 5)

    res.2 <- MakeDiscrete(res.n, 2)
    a.2.rules <- GetRulesApriori(
        res.2,
        alg.support,
        alg.confidence)
    e.2.rules <- GetRulesEclat(
        res.2,
        alg.support,
        alg.confidence)
    rule.2.count[i,1] <- length(a.2.rules)
    rule.2.count[i,2] <- length(e.2.rules)

    res.5 <- MakeDiscrete(res.n, 5)
    a.5.rules <- GetRulesApriori(
        res.5,
        alg.support,
        alg.confidence)
    e.5.rules <- GetRulesEclat(
        res.5,
        alg.support,
        alg.confidence)
    rule.5.count[i,1] <- length(a.5.rules)
    rule.5.count[i,2] <- length(e.5.rules)

    res.10 <- MakeDiscrete(res.n, 10)
    a.10.rules <- GetRulesApriori(
        res.10,
        alg.support,
        alg.confidence)
    e.10.rules <- GetRulesEclat(
        res.10,
        alg.support,
        alg.confidence)
    rule.10.count[i,1] <- length(a.10.rules)
    rule.10.count[i,2] <- length(e.10.rules)

    if(i<3){
}
incl.count <- 5
a.2.df <- as(a.2.rules[1:incl.count], "data.frame")
write.table(
a.2.df,
  file=paste(
    "iter",
    i,
    "supp",
    alg.support,
    "a-2-discrete.csv",
    sep='-',
  ),
  sep=';'
)
e.2.df <- as(e.2.rules[1:incl.count], "data.frame")
write.table(
e.2.df,
  file=paste(
    "iter",
    i,
    "supp",
    alg.support,
    "e-2-discrete.csv",
    sep='-',
  ),
  sep=';'
)a.5.df <- as(a.5.rules[1:incl.count], "data.frame")
write.table(
a.5.df,
  file=paste(
    "iter",
    i,
    "supp",
    alg.support,
    "a-5-discrete.csv",
    sep='-',
  ),
  sep=';'
)e.5.df <- as(e.5.rules[1:incl.count], "data.frame")
write.table(
e.5.df,
  file=paste(
    "iter",
    i,
    "supp",
    alg.support,
    "e-5-discrete.csv",
    sep='-',
  ),
  sep=';'
)a.10.df <- as(a.10.rules[1:incl.count], "data.frame")
write.table(
a.10.df,
  file=paste(
    "iter",
    i,
    "supp",
    alg.support,
    "a-10-discrete.csv",
    sep='-',
  ),
  sep=';'
)e.10.df <- as(e.10.rules[1:incl.count], "data.frame")
write.table(
e.10.df,
  file=paste("iter",
    i,
    "supp",
    alg.support,
Appendix A: R code run during evaluations

```r
# e-10-discrete.csv

e-10-discrete.csv

# 2-discrete steps - apriori
rule.2.count[1:ev.count,1],
main="2-discrete steps - apriori",
cex=0.4,
xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l'

# 2-discrete steps - eclat
plot(
rule.2.count[1:ev.count,2],
main="2-discrete steps - eclat",
cex=0.4,
xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l'
)

dev.off()
write.csv2(
rule.2.count,
file=paste(
"supp",
alg.support,
"2-discrete.csv",
sep=';'))

# 5-discrete steps - apriori
plot(
rule.5.count[1:ev.count,1],
main="5-discrete steps - apriori",
xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l'
)

# 5-discrete steps - eclat
plot(
rule.5.count[1:ev.count,2],
main="5-discrete steps - eclat",
xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l'
)

dev.off()
write.csv2(
rule.5.count,
file=paste(
"supp",
alg.support,
"5-discrete.csv",
sep=';'))

# 10-discrete steps - apriori
plot(
rule.10.count[1:ev.count,1],
main="10-discrete steps - apriori",
xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l'
)

# 10-discrete steps - eclat
plot(
rule.10.count[1:ev.count,2],
main="10-discrete steps - eclat",
xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l'
)

dev.off()
write.csv2(
rule.10.count,
file=paste(
"supp",
alg.support,
"10-discrete.csv",
sep=';'))
```

53
rule.10.count[1:ev.count,2],
main="10-discrete steps - eclat",
`xlab="Rule generation per 20 designs",
ylab="Rule count",
type='l')
dev.off()
write.csv2(
  rule.10.count,
  file=paste(
    "supp",
    alg.support,
    "10-discrete.csv",
    sep='-
')
)

library(arules)
alg.support <- 0.03
alg.confidence <- 1
setwd("~/git/arulesinpt/ARulesInPT")
res <- read.table("Data5000/input.res", head=T)
res.agg <- AggregateOnConfig(res)

#Dimension reduction by removing attributes concerning experiment
setup
res.n <- res.agg[1:8]
#Renaming the headers
names(res.n) <- c(
  "Y",
  "PopSize",
  "ElitCount",
  "EGS.ASelect",
  "EGS.TruncProb",
  "OM.Prob",
  "OC.Points",
  "OC.Prob")
#PrintPerParameter(res.n, 5)
Experiment123Run(res.n)