Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

Ville Ojaniemi

Computationally aided product concept generation for field devices

Boosting product concept creation

Master's Thesis
Espoo, June 4, 2018

Supervisor: Professor Antti Oulasvirta
Advisor: Niraj Dayama Ph.D.

Aalto University
School of Science

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER'S THESIS

Author:	Ville Ojaniemi		
Title:	Computationally aided product concept generation for field devices Boosting product concept creation		
Date:	June 4, 2018	**Pages:**	71
Major:	Computer Science	**Code:**	SCI3042
Supervisor:	Professor Antti Oulasvirta		
Advisor:	Niraj Dayama Ph.D.		

This thesis aims to solve the problem of optimally selecting features for internet connected field device, the accompanying cloud service and client devices such that designer goals, feature properties and relationships with other features are considered. This information could be used during concepting of new products as a support to better handle possibly large number of features and relationships between them. This could also bring up new ideas that were not considered before.

We solve the problem by formulating it as an Integer Linear Programming(ILP) problem, and utilize Monte-Carlo methods to find interesting solutions to the problem, and visualize them in a way that allows easy comparison of different solutions to the problem.

The proposed model is very stable, as adjusting weights in the ILP objective function causes fairly linear and predictable change in the outcome. The model also produced feature sets that were fairly comparable to human generated ones. The model was also tested in a real project in real work environment. The model performed well in this environment, however forming the input was considered very laborious, time consuming and complicated in some parts. These issues can be dealt with by organizing training and workshops. With further adjustments the model could be used to generate interesting concepts for many kinds of products.

Keywords:	feature selection, field device, monte-carlo, integer linear program
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Ville Ojaniemi		
Työn nimi:	Laskenta avusteinen tuotekonseptien generointi kenttälaitteille Tuotekonseptoinnin tehostaminen		
Päiväys:	4. kesäkuuta 2018	**Sivumäärä:**	71
Pääaine:	Tietotekniikka	**Koodi:**	SCI3042
Valvoja:	Professori Antti Oulasvirta		
Ohjaaja:	Filosofian tohtori Niraj Dayama		

Tämän työn tavoitteena on ratkaista kuinka valita toiminnot internettiin kytkettyyn kenttälaitteeseen optimaalisesti, sekä siihen kytkettyyn pilvipalveluun sekä asiakaslaittesiin siten, että suunnittelijan tavoitteet, toimintojen ominaisuudet sekä toimintojen väliset suhteet ovat huomioituna. Tätä tietoa voidaan käyttää uusien tuotteiden konseptoinnin tukena auttaen käsittelemään mahdollisesti suuria määriä toimintoja ja niiden välisiä suhteita. Tällä tavalla voidaan myös löytää uusia ideoita joita ei ole aiemmin tullut ilmi.

Ratkaisemme ongelman mallintamalla sen Intereg Linear Programming(ILP) ongelmana, sekä hyödyntäen Monte-Carlo metodeja löytääksemme kiinnostavia ratkaisuja ongelmaan, sekä visualisoimme ratkaisut tavalla joka mahdollistaa eri ratkaisujen helpon vertailun.

Esitetty malli on hyvin vakaa, sillä painoarvojen muuttaminen ILP tavoite funktiossa aihetuttaa melko lineaarisen sekä ennustettavan muutoksen lopputuloksessa. Malli myös tuotti toimintosettejä jotka olivat melko vertailukelpoisia ihmisen tuottamiin. Mallia testattiin myös oikeassa projektissa oikeassa työympäristössä. Malli toimi hyvin tässä ympäristössä, mutta syöte datan tuottaminen koettiin hyvin työlääksi, aikaa vieväksi ja monimutkaiseksi joissain osissa. Nämä ongelmat voidaan kuitenkin selvittää järjestämällä koulutusta ja workshoppeja. Lisäominaisuuksien ja säätöjen avulla mallia voitaisiin käyttää monenlaisten tuotteiden konseptien kehittämiseen.

Asiasanat:	toimintojen valinta, kenttälaite, monte-carlo, integer linear program
Kieli:	Englanti

Acknowledgements

I would first like to thank my thesis supervisor Prof. Antti Oulasvirta and instructor Ph.D Niraj Dayama from Aalto university for providing support, feedback and guidance thorough the thesis. I want to thank both Antti and Niraj for allowing me free hands during the thesis, and also steering me to right direction whenever I needed it.

I also want to thank Panu Kilponen from Vaisala for regular feedback and guidance. Special thanks to Sauli Laitinen from Vaisala for providing support especially in organizing the workshop and interviews. I'm very grateful for both Panu and Sauli for finding time to help me even with their busy schedules.

I would also like to thank Vaisala personnel who contributed to the interviews, workshop and validation of the results, the thesis would not have been what it is now without their input.

Finally, I wish to express my gratitude for Vaisala for funding this thesis and giving me this opportunity to learn. I have learned so many skills, and gained experience on many different areas that are invaluable in working life. Thank you.

Espoo, June 4, 2018

Ville Ojaniemi

Contents

Chapter 1

Introduction

This chapter introduces the thesis, its purpose and background. We first provide background on the problem. Then we state the problem we aim to solve and explain how the results could be used. Finally we provide an overview of the thesis structure.

1.1 Background

Consider a situation where there is a field device, a cloud service, remote client and a local client. The device could be something like Vaisala's WXT530 series sensors, which are compact and flexible, all in one weather instruments[4]. The local client uses the device locally, e.g. with a Ethernet/serial cable and thus needs to physically be in the vicinity of the device. The remote client uses the device remotely through internet. The remote client can connect directly to the device, but can also connect to a cloud service and operate the device through it. This is illustrated in figure 1.1. Each directed arc represents a data connection between different nodes. The implication of the arc's direction becomes apparent in later chapters.

Each of these connection approaches have their pros and cons. Local connection can be hard to establish as one must physically travel to the device, however using local connection is somewhat less risky, as any errors made can be reverted by resetting the device physically. Also verifying that data is correct is easier as you can see what is physically happening around the device. Verifying the data similarly is not possible with remote connections as visual confirmation is not possible.

Remote connections come with greater risk, as mistakes could lead a device to become unresponsive and thus impossible to operate remotely. This would require a trip to the device installation site to correct the problem.

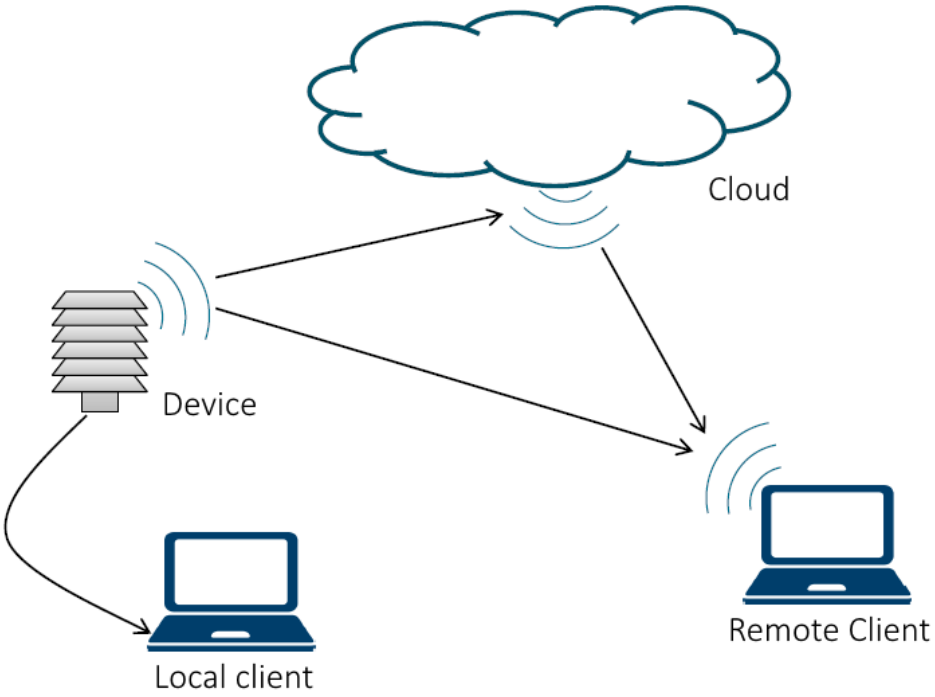


Figure 1.1: Illustration of the scenario of four nodes and four arcs between them. Each node, the device, the cloud, remote client and local client has arc to at least one other node. These arcs represent a data connection between the nodes. Between local client and device the connection can be physical, but in the other 3 arcs the data transfer happens through the internet. The arc's direction is also important which becomes apparent in later chapters

The cloud connection offers additional security and processing power with more powerful hardware. It also allows centralized data collection and monitoring from multiple devices. The cloud connection however does add delays in communication with the device and increase maintenance costs because of the costs of operating the cloud.

Each of the introduced nodes, the device, cloud, remote client and local client, have set of different features which defines how the node operates. Some of the features are shared between the nodes and some are exclusive to certain nodes. In this scenario, for example measuring data should only be possible in the device, as it is the only node with measurement capability.

Each of the features can also have some relations to the features in the same node and to features in the other nodes. For example, if the device can measure data from a sensor, a client should be able to view the data somehow. Also other way around, viewing the data is not possible if there is no data to view.

One feature might also be more important than the other, for example one could value ability to log data to cloud more than ability to log the data in the device. There can be multiple perspectives to this as well. One device should be as low powered as possible, because it runs on batteries. Another device should be as low cost as possible, because the device is intended to be deployed as large networks.

Considering all these at the same time, to produce the optimal set of features for each of the node can be difficult. The selection of functionalities and features is a recognized but under-researched subproblem in interaction design[19]. While it would be easy to just select every feature to the product to cover all the possible needs of every customer, the result would be messy to say the least. Research has shown that the designer should avoid "feature creep" as this can lead to decrease in usability and increase in cost[25]. Another study has shown that unneeded extra features provide reasons against buying the promoted brands and are seen as susceptible to criticism[21].

This provides incentive to limit the number of features and try to select only those features that are needed. Naturally the features that are needed are very product and application specific. This suggests that building products that are very focused on single application would be the best option. However having a different product for every application makes manufacturing, selling, buying and maintaining the product portfolio much more complicated and costly.

The best option is most likely somewhere in the middle of the two extremes, a balance between enough features that it covers enough applications and avoids "feature creep". This balance can be difficult to find, especially if there is very large number of features and relationships between them. To

ease the search of this balance, some help from computational methods could be used.

1.2 Problem Statement

This thesis aims to find a way of choosing optimal feature set considering properties such as usefulness, complexity, configurability and power consumption for each feature, relationships between the different features and designer preferences for the properties. Computational methods are utilized to deal with possibly hundreds or even thousands of different features and relationships between them.

Computational methods are justified as even with only 50 features and four nodes there are $4*(2^{50}) = 4,503,599,627,370,496$ possible combinations of features. Even with multiple constraints in place, such design spaces are too large for manual search.

If successful, this method could be taken into use as a tool in product concepting phase to support the process. The method is not intended to provide the definitive best product concept, as the method almost always simplifies the problem. Instead it is intended to be used as iterative supporting tool to help the designers to come up with new ideas. The tool generates few different concepts and then designers can combine ideas from the generated concepts to the actual concept, or use them as an inspiration for better ideas.

1.3 Structure of the Thesis

This thesis consists of 6 chapters. The first chapter introduces the problem and provides some background and also explains the thesis structure. The second chapter explains the methods used to solve the problem, and also examines previous work on the subject. The third chapter explains how the solution was actually implemented. In the fourth chapter we present experiments and other tests of how well the presented solution actually works. In the fifth chapter we discuss the results and their implications, and see whether we were successful at creating a useful method/tool to help with product concepting and what could be done better. In the final chapter we conclude the work, by providing a summary of all the subjects that were discussed.

Chapter 2

Methods

This chapter introduces the methods used to solve the problem. We also examine previous work on the subject and explain how this thesis uses ideas from them. We also introduce the pre-study that is used to gain an understanding of the scenario before actually attempting to solve it.

First we examine the previous work, and then introduce the pre-study and finally explain how we use Inter linear programming and Monte-Carlo method to solve the problem.

2.1 Related Work

Traditionally product concepts are often generated by coming up with sketches of possible ideas [10, 14, 26, 31]. Sketches are mappings from designer's vision to the physical representation of the vision as a sketch on paper[26]. Studies have shown that during sketching, quantity of sketches, especially in the start and mid sections of design process, plays a big role in the result of the final outcome[14, 31].

Sketching is considered fast and intuitive technique to represent the opportunistic flow of ideas[27]. However considering every property and relation between features simultaneously can be hard or even impossible if there is very large number of features to be considered. To counter this we look into computational methods that could help with the large number of features.

Indeed computational methods have been used in concept generation previously as well, we describe some of them in this section to provide a brief overview of the possible methods.

In papers by Tang[24] and Albritton[5] they used ant colony optimization(ACO) to find the best mix of product features. ACO heuristics are efficient at searching through a vast decision space and are extremely flexible

when model inputs continuously change[5].

Another approach, where design-by-analogy methodology is used by searching patent databases[11, 16, 17], as they provide attractive sources of analogies and concepts that can lead to innovative solutions [13]. Similarity ranking tools are used to retrieve the patents with the highest degree of relevance to the functional description of a given design problem and the most relevant patent results are presented to the user[11, 16, 17].

Another approach is presented in work by Bryant [8]. The presented method works by utilizing predefined functions, with known inputs and outputs, and combining those to fulfill the needs of the product.

The prior work described uses variety of different methods, but do not suit our needs perfectly. Some of the mentioned methods do find the optimal feature sets, but do not consider their relationships, some consider only the relationships between features, but do not allow favoring different kinds of features for different kinds of products. Combining these is not a trivial task either, because they utilize very different methods. However two papers, one by Oulasvirta[19] and other by Park[20] utilize Integer Linear Programming(ILP) to find best balance between features.

In the work by Oulavirta [19] they use ILP to optimize the feature set for an note-taking application. The objective function was to maximize usefulness, satisfaction, ease-of-use and profitability of the chosen feature set. Each feature was rated for its usefulness, satisfaction, ease-of-use and profitability. Dependencies between features are modeled as pair-wise usefulness. Meaning that each feature has usefulness score based on the presence or absence of another feature. This allows creating dependencies between features, by giving negative usefulness for a feature if the dependent feature is not present. As an example they provide print setup making not sense without print, thus pairwise usefulness for having print setup but not print should be negative.

In the work by Oulavirta [19] they also use the Monte-Carlo approach of Beyer and Sendhoff [6] to find a robust solution. In the paper they also present a way of finding diverse solutions [19]. The resulting feature sets were visualized in list form for easy understanding and comparison.

In the work by Park[20] they used ILP to find optimal solution UI element assignment problem across devices. The objective function was to maximize quality and completeness of the elements in different devices. The aim was to create adapting multi-user interface for real time collaborative environment. The model considered each elements' properties and features in each device to make sure that the elements are possible to assign to that device and that they fit on the device screen. Different user roles were also considered as well as user preferences. The elements had minimum size and maximum size determined, and the ILP model could adjust the element size to fit

different elements on limited screen size. Different user roles had different rights to different elements and the ILP model had constraints to allow only the allowed elements on the screen for each user role separately.

Aside from being able to handle large amounts of features, computational methods can be used to generate large amounts of varying concepts, in short time as well and as previously mentioned, the number of sketches plays a great role in the result of the final outcome [31]. This provides reasoning for using computational methods in support for product concept generation.

2.2 Integer Linear Programming

Similarly to the previous work by Oulasvirta[19] and Park[20] we aim to solve this problem as an integer linear programming (ILP) problem. Integer Linear programming in a special case of linear programming in which all variables are required to take on integer values only[29]. Linear programming is the optimization of an outcome based on some set of constraints using a linear mathematical model[18].

ILP suits our problem well, as the variables we control are all binary variables, having values 1 or 0. Each variable describes whether a certain feature is present in a certain node similarly as in previous work[19, 20]. Linear objective function is also suitable for our problem, because linear terms are intuitive to the designer to understand and thus control[19]. Also constraints for features and their relationships can be modeled with linear models[19, 20].

Another advantage to using ILP is its speed. ILP solvers are generally very fast at solving the problem and are guaranteed to provide the most optimal result. These properties are exploited in the Monte-Carlo approach explained in 2.3. Using ILP with the Monte-Carlo method yields multiple optimal solutions in relatively short time.

Another strength of modeling the problem as ILP problem is that there are numerous ILP solvers readily available, such as the open source Cbc (Coin-or branch and cut)[1] or the commercial Gurobi[2]. This makes modeling the problem as an ILP problem attractive, as we can use these solvers to actually solve the problem and find the optimal solutions. Thus we only need to define the problem in ILP form and feed it to the solver. This makes the implementation faster, and it is also guaranteed to find the optimal solution, especially when using previously proven and tested solvers, such as the Cbc and Gurobi.

2.3 Monte-Carlo Approach

The Monte-Carlo Method means any method which solves a problem by generating suitable random numbers and observing that fraction of the numbers obeying some property or properties. The method is useful for obtaining numerical solutions to problems which are too complicated to solve analytically[30].

We use Monte-Carlo method described in previous work[6, 19]. This method includes taking many random samples for each rating for input data and solving the problem optimally for each sample. From these samples a robust solution[6, 19] and diverse solutions[19] are searched for. The robust solution is the best compromise among all the solutions, while the diverse solutions are a set of most distinct solutions in the solution set. This idea of presenting several diverse options is a technique called Design Gallery[15].

Using this method allows us to find differing and interesting solutions to the problem. The random sampling of the input data is also important, as the input data most likely is not the objective ground truth for the data. The input data is entered by humans, and thus is likely to be somewhat biased. The random sampling allows us to diminish the effects of the bias as we do not rely completely on the user inputted data, but use it more as an estimate and guide for the actual input. It also outputs multiple different solutions at the same time, allowing the user to find interesting ideas more easily.

2.4 Pre-study Interviews

Before the actual problem is considered, a pre-study is conducted by interviewing relevant people on field device usage, uses and needs. Main goals are to understand who are the users and why they use the device and what are their goals. Also the uses are mapped, what the device is supposed to do and why. The aim is to gather an sufficient understanding of the scenario. With this information it is possible to define the most important properties to consider for each feature in the device, cloud and the clients.

Chapter 3

Implementation

This chapter introduces the ILP model implementation, and how the model is used with the Monte-Carlo method to find the robust and diverse solutions. First we discuss the pre-study interviews and how the collected data is used in the ILP model.

From the ILP model, we first introduce the nodes and arcs between them. Then we discuss how each feature is rated for different properties, and how they are considered in the objective function. Then we introduce the dependencies between different features. After explaining the functionality of the model on higher level, we show how it is actually implemented. We describe all the variables, constraints and objective function of the ILP model.

Then we describe how this model is used with the Monte-Carlo method to find the robust and diverse solutions. Lastly we show how the results are visualized for easy comparison and briefly describe the user interface made for the model for ease of use.

3.1 Pre-study Interviews

The interviews were done to gain a sufficient understanding of the device's users and uses. Main goals were to understand who are the users and why they use the device and what are their goals. Also the uses were mapped, what the device was supposed to do and why.

As this thesis was done for Vaisala, the interviewees were mainly Vaisala employees, with varying titles, experience and expertise. This was done to avoid favoring single areas, which could lead to insufficient and biased results. A single person outside Vaisala was also interviewed to slightly mix the data with a different viewpoint. List of the interviewees can be seen in figure 3.1.

All the interviewees were given a short introduction to the subject in the

Product Manager	31 years
R&D Manager	20 years
Business Development Manager	18 years
Senior Scientist	17 years
Application Manager	17 years
R&D Project Manager	9 years
Product Manager	7 years
Product Manager	7 years
Application Manager	5 years
Postdoctoral Researcher	1 year

Figure 3.1: Titles of the interviewees and years in Vaisala or Aalto University

invitation before the interview to make sure the interviewee could prepare for the interview and to allow them to think the subject beforehand.

All interviews were done face-to-face each in 30-60 minutes long session. The interview was loosely structured around interviewing template in appendix A. The interview was kept only loosely structured to encourage the interviewee explaining his/her ideas and views as naturally as possible in a conversation-like manner, getting into "storytelling mode". This was done to allow the conversation naturally branch, possibly exposing things that would not have come up with strictly scripted conversation.[12, p. 4] The conversation was returned to the template if the conversation started to branch too far from the subject.

The interviews were documented by taking notes as the conversation went on. The interviews were also recorded to avoid missing details. However some interviews were not recorded due to the wish of the interviewee.

The data gathered from the interviews contained different notes about the device, containing ideas, functionalities, requirements, problems, strengths, customer hopes, and other observations about the device and its uses and users.

3.2 ILP Model

3.2.1 Nodes and Arcs

We represent the problem in terms of a directed graph $G(N, A)$. The nodes N of this graph represent the different locations at which the specified features can be chosen (device, cloud, remote- and local client). The arcs of this graph

node	description
The device	The field device itself, e.g. remote sensor
The cloud	A cloud service, running on a server
Remote client	A client device, located away from the site
Local client	A client device, located at the site

Table 3.1: Table describing the nodes

tail	head
device	cloud
device	remote client
device	local client
cloud	remote client

Table 3.2: Table describing arcs between the nodes, the arcs are directed from tail to head

represent the data connections between these nodes. The arcs are directed from tail node to head node[28]. The direction of an arc is used in the inter-dependency explained in following sections. In our scenario there are four nodes and arcs described in tables 3.1 and 3.2 and previously in section 1.1.

In this scenario the device is a remote sensor measuring data. The clients are computers that run a client software and the users interact with the system through these computers. The Local client is located near the device at the sensor installation site, physically connected to the device. To establish this local connection the user must travel to the device physically. The remote client operates the device remotely. The remote operation is done directly through the internet or through the cloud. The cloud is a server running a service, to co-operate with the device and the remote client.

3.2.2 Features

Features are the items that may or may not be selected for a certain node. Because the solver works on very abstract level the features can be as specific or general as needed.This is useful when generating product concepts on different stages of product concepting. At the beginning of concepting the features can be very general, e.g. 'Graphical UI', 'wireless connectivity', and as the concepting progresses and ideas start to clear, more specific features can be added, e.g. 'log in button','registration form', 'LoRa','WiFi'.

Some features can be allowed only on certain nodes. This is important as some features just aren't possible to be implemented in some nodes. This could be for example displaying graphical elements on a device that does not have a screen. Another case might be that data logging cannot be implemented in the clients, as there is no guarantee that the client is online or connected to internet to log the data, while the cloud and device itself should always be available for data logging.

Similarly to work by Oulasvirta[19] we give each possible feature scores based on different properties. The different properties come from the pre-study introduced in the previous section. Each feature can have many ratings for each property, which are combined to single normal distribution, based on the mean and standard deviation of the ratings.

As in work by Park[20] the ILP model allows features to be selected in the different components, also allowing some features in only some components by defining constraints for these features.

Each feature is rated based on its usefulness, complexity, cost, power consumption and configurability. The ratings are used in the objective function to describe each features desirability. This section describes the different properties on which the features are rated for.

Many of the properties in this section are aggregates of many different things, instead of all of the different things being separate, they are grouped into larger entities to ease the process of forming the input data. While this does sacrifice some details, it is a compromise between granularity and effort needed to form the input data.

Aside rating the properties the ratings also contain a confidence value, indicating how sure each person is about his/her ratings being close to the underlying "truth". This information is later used in the sampling of the ratings as described in section 3.3.1.

Usefulness is related to user satisfaction and competitive edge. Highly distinct features compared to competitors is considered useful as it attracts customers.

Complexity of feature means increase of complexity of the system if the feature is chosen. It relates to ease of use inversely, meaning that highly complex feature makes the device harder to use. Very low complexity means that the selection of the feature does not increase the complexity of the system and can even make it simpler.

Cost is related to all costs associated with the feature. This can mean work hours or actual money related to implementation and maintenance for example. It is also related to increase of cost of the device. Some feature might need lot of memory or processing power which require better hardware, which comes with a cost. Cost can also be interpreted as risk, if a feature is

prone to information security problems, it can increase the cost.

Power consumption means increase of power requirements of the device. Heating for example uses lots of electricity. Also a feature might require the device to be constantly on and listening for input for example, this naturally increases the power consumption of the device as it cannot go to low power sleep mode.

Configurability describes how configurable and customizable the feature is, and how much can this feature change the behaviour of the device. This is important metric as the device is often not just a standalone device but has to integrate into larger systems and these systems can be very different. Making the device flexible in configuration sense, makes it appealing to more customers. Features such as 'set IP address' allow some, although very little, configurability, while allowing the device to read configuration files and custom programs makes it very configurable.

3.2.3 Dependencies

The dependencies are relations between features. They generate constraints for the ILP solver, by not allowing every combination of features. This makes the output of the solver more useful, as illegal combinations are never generated and it also speeds up the solver, as the constraints limit the search space for the optimal solution.

Dependencies can be used to group features together, meaning that if a feature is chosen, it's counterpart defined by a dependency is also chosen. They are also used to make sure that a feature is possible to be selected in a certain node. A feature might depend on other feature's output, and cannot be used without this other feature. Defining a dependency between these features ensures that the feature is not chosen if it cannot be used. There are two different types of dependencies called inter-dependency and intra-dependency. There also exists two variants of each dependency, weak and strong. These dependencies can be chained to make complex dependency structures between the features.

Inter-dependency

The inter-dependency is a dependency along an arc, between nodes. Each arc is directed from tail node to head node. To define this dependency let us consider a situation between the cloud and device nodes. There exists an arc from device to cloud, Making device the tail and cloud the head. The inter dependency in this case means that if there is feature A in the head node

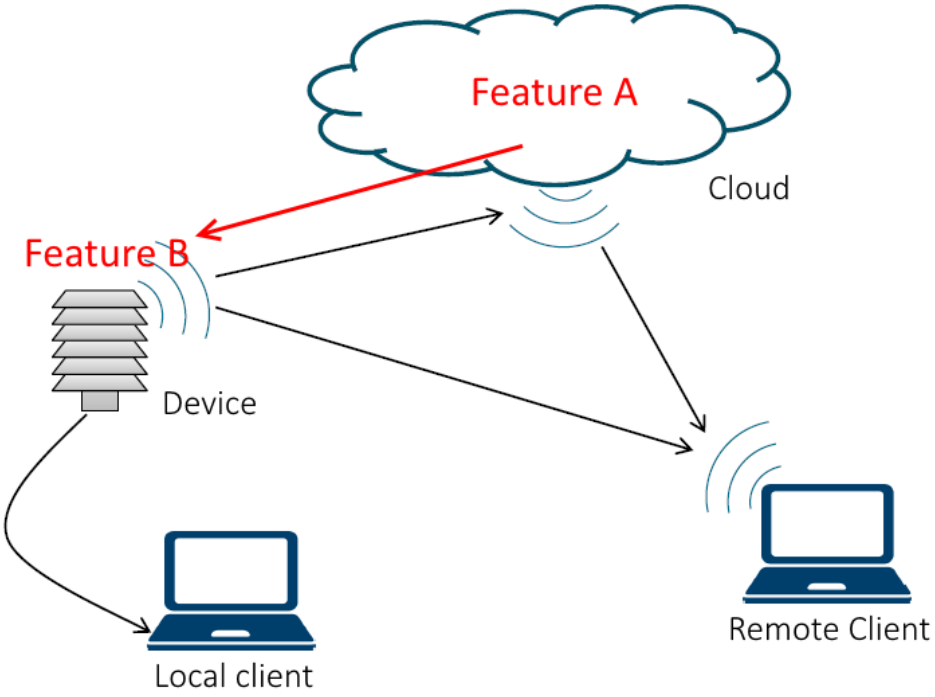


Figure 3.2: Illustration of inter-dependency. If there is inter dependency from feature A to feature B, and feature A is chosen in a head node, i.e. node where an arc points to, then feature B must be chosen in the tail node as well.

with inter-dependency to feature B, then feature B must be selected in the tail node to satisfy the dependency. This is illustrated in figure 3.2 .

This dependency is used to define dependencies where a feature needs another feature to be present in another node for the feature to be useful. As an example data logging in cloud is possible only if a device transmits data to the cloud. This means that feature ”data logging” has inter-dependency to feature ”transmit data”.

Inter-dependency is also used to ensure a feature is ”supported” in the tail node. For example feature ”set device IP address” might not have any dependencies, and could thus be freely chosen in any node. However this feature in a client node is useless if it not supported in the device i.e. in client software there is possibility to set the address, but the address cannot be changed in the device. This is solved by defining an inter-dependency from the feature to the feature itself. This way if a feature is selected in a client node, it also must be selected, ”supported”, in the device node.

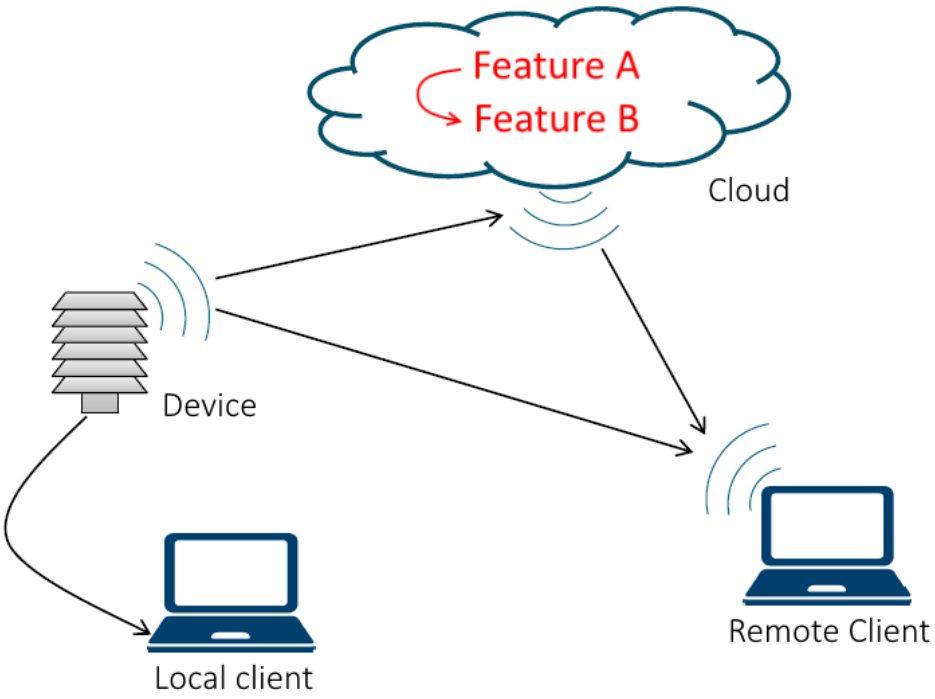


Figure 3.3: Illustration of intra-dependency. Dependency within node, if feature A has intra-dependency to feature B, and feature A is selected in a node, then feature B must also be selected in the same node.

Intra-dependency

The intra-dependency is dependency within a node. This means that if feature A has intra-dependency to feature B and feature A is selected in a node, then the feature B must also be selected in the same node. Illustrated in figure 3.3.

This dependency is useful for grouping items together. For example feature ”log in” should always be accompanied with ”log out”, as either of them alone is not very useful. This could be defined as ”log in” having intra-dependency to ”log-out” and vice versa.

Intra-dependency can also be used to define that a feature belongs to a certain group, e.g. permission level. A device could have feature ”admin-level” and all features that should be allowed only to system administrators, should have intra-dependency to ”admin-level”. This way a administration feature cannot be selected in a node where the ”admin level” feature is not allowed.

Weak and strong dependencies

The two previously mentioned dependency types, intra and inter, also have two versions of them,weak and strong, making the total number of different dependencies to four.

Strong dependency as the name suggests must always be satisfied. Meaning if a feature has a dependency to another feature, and the feature is selected, then the other feature must always be selected. The strong dependency acts as logical AND case when there are multiple strong dependencies for a feature, meaning that each one of them must be satisfied.

Strong dependencies are useful for ensuring that another feature is always present. This could be the case for features 'log in' and 'log out', which should always be together to be any use. In this case both features should have strong dependency to each other.

Weak dependency on the other hand is not as strict. The weak dependency acts as logical OR case when there are multiple weak dependencies, meaning that it is enough that one of the weak dependencies are selected.

Using weak dependencies allow the feature to be satisfied by multiple different ways. For example feature 'data logging' depends on having some data to be logged, this could be acquired by taking a sensor measurement directly, or listening to a transmission from a remote sensor. This could then be defined as 'data logging' having two weak dependencies to features 'measure data' or 'receive data', and the 'data logging' feature would be satisfied if either of the two features are chosen.

In case a feature has multiple weak and strong dependencies, one of the weak dependencies and all of the strong dependencies must be selected to satisfy the feature dependencies. If a feature has only a single dependency, it does not matter whether it is weak or strong, as in the weak case atleast one, and in strong all of the dependencies must be selected and in this case they both mean that the one dependent feature must be chosen.

Dependency combinations

While the four previously described dependencies: weak inter-dependency, weak intra-dependency, strong inter-dependency and strong intra-dependency, are useful on their own, using them together is what allows complex and flexible dependency structures to be defined.

As an example of the dependency structures let us consider features 'data logging', 'measure data', 'transmit data' and 'receive data'. The 'data logging' naturally needs data to log to be useful. The data could be acquired by measuring the data ourselves, or receiving the data from somewhere else.

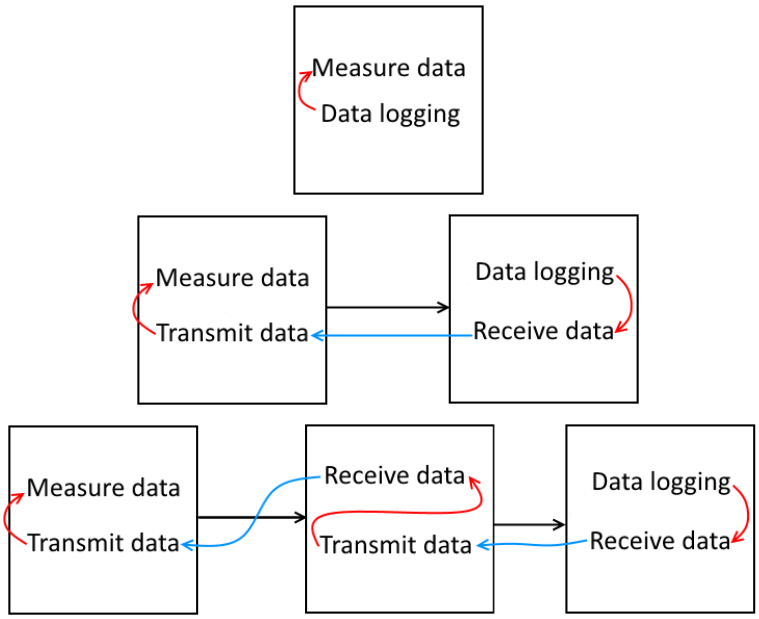


Figure 3.4: Three solutions to the example scenario. The black squares represent nodes, black arrows arcs between nodes, red arrows intra-dependencies and blue arrows inter-dependencies. In the top most solution, there exists only one node, which measures and logs the data, the middle one, has two nodes, one of which logs received data and one measures and sends the data, the bottom one, is similar to the middle one, but with one extra node, that forwards the data to the logging node.

Transmitting data also requires either measuring the data or receiving it from somewhere else. Measuring data does not depend on anything. Receiving data naturally depends on someone transmitting data.

The dependencies in this case should be defined as follows. The 'receive data' should have strong inter-dependency to 'transmit data', 'transmit data' should have weak intra-dependency to measure data and receive data. Data loggin should have weak intra-dependency to measure data and receive data. We present three solutions to this scenario in figure 3.4.

Different combinations can also be used to define 'feature modules', where choosing a feature chooses a set of other features. For example there could be a feature called 'admin user level' which itself is not an actual feature, but groups together different admin level features using strong dependencies. Another example could be "wireless communication" that has weak dependencies to different wireless communication methods, e.g. WiFi,LoRa. Then a feature could have strong dependency to 'wireless communication' without

needing to define the different communication methods explicitly for each feature that needs some sort of wireless communication. This also simplifies defining dependencies allowing utilizing sort of 'divide and conquer' method by allowing feature abstraction.

3.2.4 Objective Function

The objective function is based on seven terms. These terms are based on the objective functions as discussed by Oulasvirta et al., where the objectives are Usefulness, Satisfaction, Ease-of-use and Profitability[19]. These objectives are adjusted based on the pre-study interviews to fit the current scope better. The resulting terms are Ease of use and installation(E), Usefulness(U), Configurability(G), Simplicity and cost of device(S), Overall cost(C), Low power(P) and independence(I).

The objective function is defined as maximizing the linear combination of the terms E, U, G, S, C, P and I, each term is also associated with a weight ω to allow weighting some terms more than others. This is useful in generating concepts for different products. If the concept is for a device that should be battery powered, then the weights for low power, and simplicity should be increased. And if the device should be standalone multipurpose super device, then usefulness, independence and configurability should be increased.

The following descriptions of each objective are based on work by Oulasvirta[19] and the pre-study interviews.

Term E considers the ease of use and install of the system. Naturally easy setup and usage of system is important, because customer most often is interested in the data the device provides, and not the device itself, thus the installation should be as painless as possible. The terms is defined as negative sum of complexity. To have easy to use and install device, we should avoid having lot of complex features.

Term U considers the usefulness of the system. The system should of course be of maximum use, and as attractive as possible for the customer. The term is defined as sum of usefulness of all features. We should favor features with high usefulness score.

Term G considers the configurability of the system. There are many needs for the device, and thus the system should be modifiable to meet these needs. The term is defined as sum of configurability scores of the features.

Term S considers the cost of device, which is an important factor, especially if the device is intended to be deployed in large numbers to form a network of devices. Also simple device, with less features is cheaper and less likely to break and thus need maintenance. S is defined as sum of negative cost scores and number of features in device.

Term C considers the total cost of the system, while S considered only the device itself. The cost of device is not considered in this term, as it already is considered in S, otherwise the costs of the device would be counted twice. Thus C is defined as sum of negative cost scores of all features except those selected in device.

Term P considers power usage of the device. Low power can be important factor in many devices, especially those running on battery. This means that features that use lot of power should not be selected in the device. P is defined as sum of negative power consumption scores of features selected in device.

Term I considers how standalone the device is. Independence in this context means that the device should not rely on other parties to operate. The device should implement all of its functionality on its own, as connecting to a cloud service is not always feasible or even possible. I is defined as negative sum of features selected in cloud.

3.2.5 ILP Model Implementation

The ILP formulation is coded in Python 3.6 and the actual ILP solver used is the default in the PuLP python package called CBC [1, 3]. All the variables are defined in table 3.3, and explained in more detail in the following subsections.

Nodes and arcs

Each node is represented with an integer index, $0, 1, 2, 3$ corresponding to device, cloud, remote client and local client respectively.

The arcs are defined as binary decision variable, directed from node $j \in \{0, 1, 2, 3\}$, to node $i \in \{0, 1, 2, 3\}$.

$$z_{ji} = \begin{cases} 1 & \text{if there is an arc from j to i} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

If some connections are not permitted or undesirable, they are precluded by forcing the variable to be 0 with a constraint

$$z_{ji} = 0, \text{if arc from } j \text{ to } i \text{ is not allowed} \tag{3.2}$$

Features

Each feature is represented with an integer index ranging from 0 to $F - 1$, where F is number of features.

Variable	Description
$x_{fi} \in \{0,1\}$	assignment of feature f to node i
$z_{ji} \in \{0,1\}$	existence of arc from node j to i
$\alpha_{figj} \in \{0,1\}$	satisfaction of weak inter-dependency of feature f in node i to feature g in node j
$\beta_{fgi} \in \{0,1\}$	satisfaction of weak intra-dependency of feature f to feature g in node i
$\gamma_{figj} \in \{0,1\}$	satisfaction of strong inter-dependency of feature f in node i to feature g in node j
$\delta_{fgi} \in \{0,1\}$	satisfaction of strong intra-dependency of feature f to feature g in node i
$\mu_{fu} \in \mathbb{R}$	usefulness of feature f
$\mu_{fx} \in \mathbb{R}$	complexity of feature f
$\mu_{fc} \in \mathbb{R}$	cost of feature f
$\mu_{fp} \in \mathbb{R}$	power consumption of feature f
$\mu_{fg} \in \mathbb{R}$	configurability of feature f
$\omega_E \in \mathbb{R}$	Weight for term E
$\omega_U \in \mathbb{R}$	Weight for term U
$\omega_F \in \mathbb{R}$	Weight for term F
$\omega_S \in \mathbb{R}$	Weight for term S
$\omega_C \in \mathbb{R}$	Weight for term C
$\omega_P \in \mathbb{R}$	Weight for term P
$\omega_I \in \mathbb{R}$	Weight for term I

Table 3.3: Table describing the variables in the ILP implementation

Each feature $f = 0, 1, ...F$ can be chosen in any node $i \in \{0, 1, 2, 3\}$, if not restricted. This is represented with a binary decision variable x_{fi}.

$$x_{fi} = \begin{cases} 1 & \text{if } f \text{ is selected in } i \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

For each feature $f = 0, 1, ...F-1$ that is not allowed in node $i \in \{0, 1, 2, 3\}$, we force the feature in that node to be not selected by defining constraint:

$$x_{fi} = 0, \text{if } f \text{ not allowed in } i \tag{3.4}$$

Each feature has real valued scores for usefulness, complexity, cost, power consumption and configurability. The scores for feature f is marked with following variables.

$$\mu_{fu} \in \mathbb{R} \text{ for usefulness} \tag{3.5}$$
$$\mu_{fx} \in \mathbb{R} \text{ for complexity} \tag{3.6}$$
$$\mu_{fc} \in \mathbb{R} \text{ for cost} \tag{3.7}$$
$$\mu_{fp} \in \mathbb{R} \text{ for power consumption} \tag{3.8}$$
$$\mu_{fg} \in \mathbb{R} \text{ for configurability} \tag{3.9}$$

Dependencies

For each defined weak inter-dependency between two features there is variable α_{figj},

$$\alpha_{figj} = \begin{cases} 1 & \text{if dependency is satisfied} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Where $f = 0, 1..F - 1$ is a feature with weak inter-dependency to feature $g = 0, 1..F - 1$, and $i \in \{0, 1, 2, 3\}$ is the node for f, and $j \in \{0, 1, 2, 3\}$ is the node for g

For each defined weak intra-dependency between two features there is variable β_{fgi},

$$\beta_{fgi} = \begin{cases} 1 & \text{if dependency is satisfied} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

Where $f = 0, 1..F - 1$ is a feature with weak intra-dependency to feature $g = 0, 1..F - 1$, and $i \in \{0, 1, 2, 3\}$ is the node for f and g.

For each defined strong inter-dependency between two features there is variable γ_{figj},

$$\gamma_{figj} = \begin{cases} 1 & \text{if dependency is satisfied} \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

Where $f = 0, 1..F-1$ is a feature with strong inter-dependency to feature $g = 0, 1..F-1$, and $i \in \{0, 1, 2, 3\}$ is the node for f, and $j \in \{0, 1, 2, 3\}$ is the node for g

For each defined strong intra-dependency between two features there is variable δ_{fgi},

$$\delta_{fgi} = \begin{cases} 1 & \text{if dependency is satisfied} \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

Where $f = 0, 1..F-1$ is a feature with strong intra-dependency to feature $g = 0, 1..F-1$, and $i \in \{0, 1, 2, 3\}$ is the node for f and g.

To enforce that α_{figj} is 1 exactly when the dependency is satisfied, and 0 otherwise we define the following constraints.

$$\alpha_{figj} \geq x_{gj} + z_{ji} - 1 \tag{3.14a}$$
$$\alpha_{figj} \leq x_{gj} \tag{3.14b}$$
$$\alpha_{figj} \leq z_{ji} \tag{3.14c}$$

Here equation 3.14a forces the variable α_{figj} to be 1 if the dependent feature is selected in the tail node and the arc from tail to head node is also selected. However, this equation alone does allow the variable to be 1 in other cases as well. Equation 3.14b forces the variable to be 0 if the dependent feature is not selected in tail node, and equation 3.14c forces the variable to be 0 if the arc is not present. Together these equations force the variable to be 1 when dependency is satisfied and 0 otherwise.

Similar equations apply also for γ_{figj}.

$$\gamma_{figj} \geq x_{gj} + z_{ji} - 1 \tag{3.15a}$$
$$\gamma_{figj} \leq x_{gj} \tag{3.15b}$$
$$\gamma_{figj} \leq z_{ji} \tag{3.15c}$$

Equations 3.14 and 3.15 were about forcing the inter-dependencies to correct values. In the following we do the same for intra-dependencies. To force β_{fgi} to be 1 exactly when the dependency is satisfied and 0 otherwise we define a constraint:

$$\beta_{fgi} = x_{gi} \tag{3.16}$$

Similarly for strong intra-dependency:

$$\delta_{fgi} = x_{gi} \tag{3.17}$$

Equations 3.16 and 3.17 constrain the dependency variable to be 1 if the dependent feature is selected and 0 otherwise.

The above equations 3.14, 3.15,3.16 and 3.17, indicate whether a feature dependency could be satisfied if the feature is chosen. Now we must also make sure that a feature is only selected when its dependencies are selected.

As explained in section 3.2.3 weak dependencies mean that atleast one of the weak dependencies defined for a feature must be selected to satisfy the weak-dependencies for this feature. In our ILP implementation this is ensured with defining following constraint for each feature $f = 0, 1...F-1$ in each node $i \in \{0, 1, 2, 3\}$.

$$\sum_{g=0}^{F-1}(\beta_{fig} + \sum_{j=0}^{3} \alpha_{figj}) \geq x_{fi} \tag{3.18}$$

Equation 3.18 counts the number of satisfied weak dependencies for a feature, and requires that sum to be greater or equal to the feature decision variable. This results the sum to be at least 1, if the feature is selected.

Each strong intra-dependency has the following constraint:

$$\delta_{fgi} \geq x_{fi} \tag{3.19}$$

A node can be the head node for multiple arcs, and it is enough that one of these arcs satisfy a strong inter-dependency for a feature. This means that if a feature has a strong inter-dependency to another feature this another feature does not need to be in every node there is an arc coming from. So for each strong inter-dependency there is a following constraint.

$$\sum_{j \in N} \sum_{j=0}^{3} \gamma_{figj} \geq x_{fi} \tag{3.20}$$

Objective function

The objective function is defined as maximizing H

$$H = \omega_E * E + \omega_U * U + \omega_G * G + \omega_S * S + \omega_C * C + \omega_P * P + \omega_I * I \quad (3.21)$$

Where G is the total objective,E, U, S, G, C, P, I are the terms, and $\omega_E, \omega_U, \omega_F, \omega_S, \omega_C, \omega_P, \omega_I$ are the corresponding weights for each term.

$$E = \sum_{f=0}^{F-1} \sum_{i=0}^{3} -x_{fi} * \mu_{fx} \quad (3.22)$$

$$U = \sum_{f=0}^{F-1} \sum_{i=0}^{3} x_{fi} * \mu_{fu} \quad (3.23)$$

$$G = \sum_{f=0}^{F-1} \sum_{i=0}^{3} x_{fi} * \mu_{fg} \quad (3.24)$$

$$S = \sum_{f=0}^{F-1} -1 - x_{f0} * \mu_{fc} \quad (3.25)$$

$$C = \sum_{f=0}^{F-1} \sum_{i=1}^{3} -x_{fi} * \mu_{fc} \quad (3.26)$$

$$P = \sum_{f=0}^{F-1} -x_{f0} * \mu_{fp} \quad (3.27)$$

$$I = \sum_{f=0}^{F-1} -x_{f1} \quad (3.28)$$

The equations 3.22, 3.23, 3.24 and 3.26 are sums of all selected features' relevant scores. The equations 3.25 and 3.27 are sum of relevant scores for features selected in device. This is noted with the value 0 in x_{f0} as the node with index 0 corresponds to the device node. Similarly in equation 3.28 the value 1 in x_{f1} notes the cloud node.

3.3 Monte-Carlo Method

We used a Monte-Carlo method similarly as in work by Oulasvirta[19] and work by Beyer [7]. The method consists of two parts, first sampling the ratings and solving the model for each sample and then searching the solutions for robust and diverse solutions.

3.3.1 Random Sampling

Each feature has multiple ratings for each of the properties, also each rating has a confidence level. The confidence level is used to give more weight for ratings with high confidence and less weight to those ratings where confidence level was low. This is useful as not all persons who rate the features are experts on every feature domain. Someone can know more about network related features and some about UI related features, and their vote should matter more on features related to their field.

The confidence levels and the different ratings are used to compute a mean and standard deviation for each property for each feature. These values are used to construct a normal distribution for each property for each feature.

Before the ILP solver is run, each feature property's normal distribution is sampled. This sampled value is then used in the solver as the property value. The LP problem is constructed with these samples and then solved. Each run of the solver solves the problem optimally for those sampled ratings.

This process of sampling the ratings, and solving the problem with the samples is repeated multiple times, few thousand times for example. Multiple runs are necessary as each time the ratings are sampled, the optimal solution for those samples can be very different from the other solutions. Sampling multiple times increases the likelihood of finding the samples which, when solved with, produces the most useful solutions.

3.3.2 Mining Solutions

After running the solver with different samples for some hundreds or thousands times, we have some hundreds or thousands solutions to the problem. This section introduces how this set of solutions are searched for the robust and diverse solutions similarly as in previous work[6, 19].

The difference in two solutions is computed as the Hamming distance, i.e. number of differing elements. In our case it means the number of differing features chosen. It can be computed as XOR of the two feature sets, and counting the number of elements in the resulting set. This number is called the distance between the two sets in the following sections.

Robust Solution

The robust solution is the best compromise between all the solutions. It is the solution that has lowest total distance to every other solution.

Robust solution is found with algorithm 1. The algorithm computes the total distance to all the other solutions for each solution, see line 4 in al-

gorithm 1, and choosing the one with lowest total distance, see line 8 in algorithm 1

Algorithm 1 Robust solution search

Input: *solutions* list of solutions from monte-carlo method

```
1:  minTotDist ← ∞
2:  robust ← null
3:  for all s1 ∈ solutions do
4:      totDist ← 0
5:      for all s2 ∈ solutions do
6:          totDist ← totDist + XOR(s1, s2)
7:      end for
8:      if totDist < minTotDist then
9:          robust ← s1
10:         minTotDist ← totDist
11:     end if
12: end for
13: return robust
```

Output: the solution with lowest total distance to other solutions

Diverse Solutions

The diverse solutions are a set of most distinguished sets, or most different from each other. The set contains K solutions, and this set should have the highest possible distance to each other.

The set is searched with algorithm similar to the one introduced in work by Oulasvirta[19]. The algorithm first chooses a random solution, and inserts it to the set, as seen in algorithm 2, line 4. and then finds the most distant solution to that, at line 11, and inserts to the set, at line 16, then finds the most distant solution to all of the previous solutions in the set similarly, this is repeated K times. After that we have a set of K solutions that are maximum distance away from each other.

This process of finding the K solutions is repeated M times, and the set with highest mean distance,see line 23 between the solutions in the sets is chosen as the diverse set, at line 30, and the solutions in that set are the diverse solutions.

Algorithm 2 Diverse solutions search

Input: *solutions* list of solutions from monte-carlo method

```
1: maxMeanDist ← 0
2: maxMeanset ← null
3: for m = 0 to M do
4:     set ← []
5:     set[0] ← random s ∈ solutions
6:     for k = 1 to K − 1 do
7:         maxD ← 0
8:         maxDsol ← null
9:         for all s1 ∈ solutions do
10:            minD ← ∞
11:            for all s2 ∈ set do
12:                if XOR(s1, s2) < minD then
13:                    minD ← XOR(s1, s2)
14:                end if
15:            end for
16:            if minD > MaxD then
17:                maxD ← minD
18:                maxDsol ← s1
19:            end if
20:        end for
21:        set[k] ← maxDsol
22:    end for
23:    dsum ← 0
24:    for k1 = 0 to K − 1 do
25:        for k2 = 0 to K − 1 do
26:            dsum ← dsum + XOR(set[k1], set[k2])
27:        end for
28:    end for
29:    dsum ← dsum/k
30:    if dsum > maxMeanDist then
31:        dsum ← maxMeanDist
32:        maxMeanset ← set
33:    end if
34: end for
35: return maxMeanset
```

Output: set of K solutions, with maximum mean distance to each other

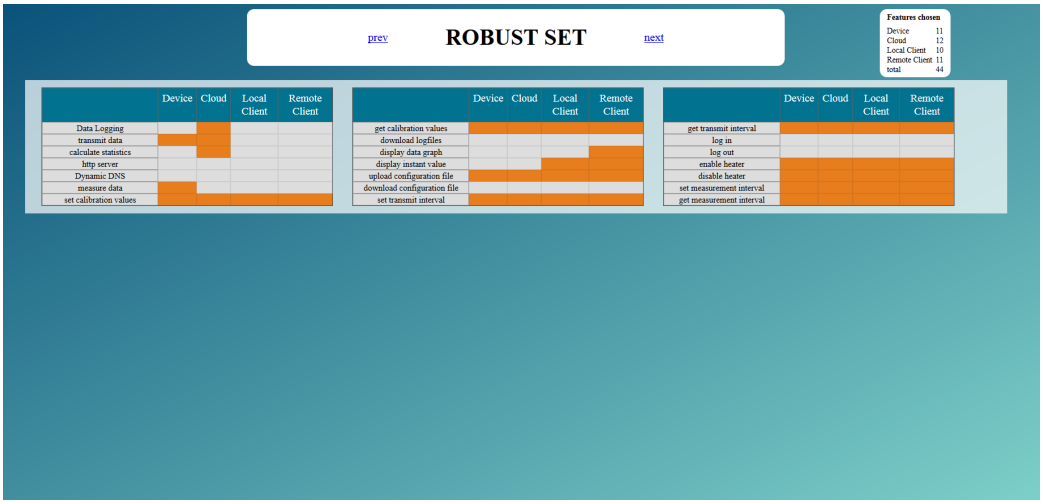


Figure 3.5: Screenshot of the visualization for the computed feature set with dummy data. A colored cell indicates that the feature on the row, is selected in the node on the column.

3.4 Visualization

The robust and diverse sets are visualized in a grid form, where each row corresponds to a feature, and a column corresponds to a node. at the intersections, there either is a empty cell or a colored cell. A colored cell indicates that the feature on the row, is selected in the node on the column. Empty cell indicates that the feature is not selected in that node. All features are displayed on the visualization to see which features were not selected and which were not. The visualizations are in HTML form. The HTML form allows using hyperlinks to jump between solutions, this makes comparing the solutions easy. The HTML can be shared easily and can be opened basically on any device, without needing other applications than a web browser. A screenshot of the visualization can be seen in figure 3.5

3.5 Concepting Tool

The system is used with a simple user interface, allowing the user to choose the different features, ratings, nodes, arcs and dependencies by selecting files to read. The ratings are read from multiple files, each containing a single set of ratings for each feature, and the allowed nodes for each feature. The different arcs between nodes are read from a file as well, arcs defined as an adjacency matrix. The dependencies are also read from a file, with $F \times F$

matrix of dependencies. Screenshot of the tool can be seen in figure 3.6.

The tool has three run configurations, test, normal and comprehensive. The test mode runs fast as it does only 10 iterations of the Monte-Carlo method. It outputs 3 diverse solutions and the robust solution. The test mode is used to test the tool to see what the output is like with the given settings. The normal mode is slower and runs 1000 iterations, and outputs 5 diverse solutions, and the robust solution. It's used to find more diverse solutions than the test case, as it iterates 100 times more than the test mode. The comprehensive is the slowest as it runs 10000 iterations, and finds 6 diverse solutions and the robust solution. The comprehensive mode tries to find even more diverse solutions. It is intended to be used when the weights are determined to produce suitable results, using test and normal mode. It is not very suitable for fast iterative use, as single run can take long time. With 103 features and 154 dependencies the test mode runs for about 30 seconds, normal mode about 10 minutes and comprehensive couple of hours.

Once the tool is done computing the solutions it opens the visualization in the computer's default browser. The console also outputs information about the progress, and gives some information about the performance, for example how long it took to iterate and solve the solutions, how long it took to find the robust solution and the diverse solutions.

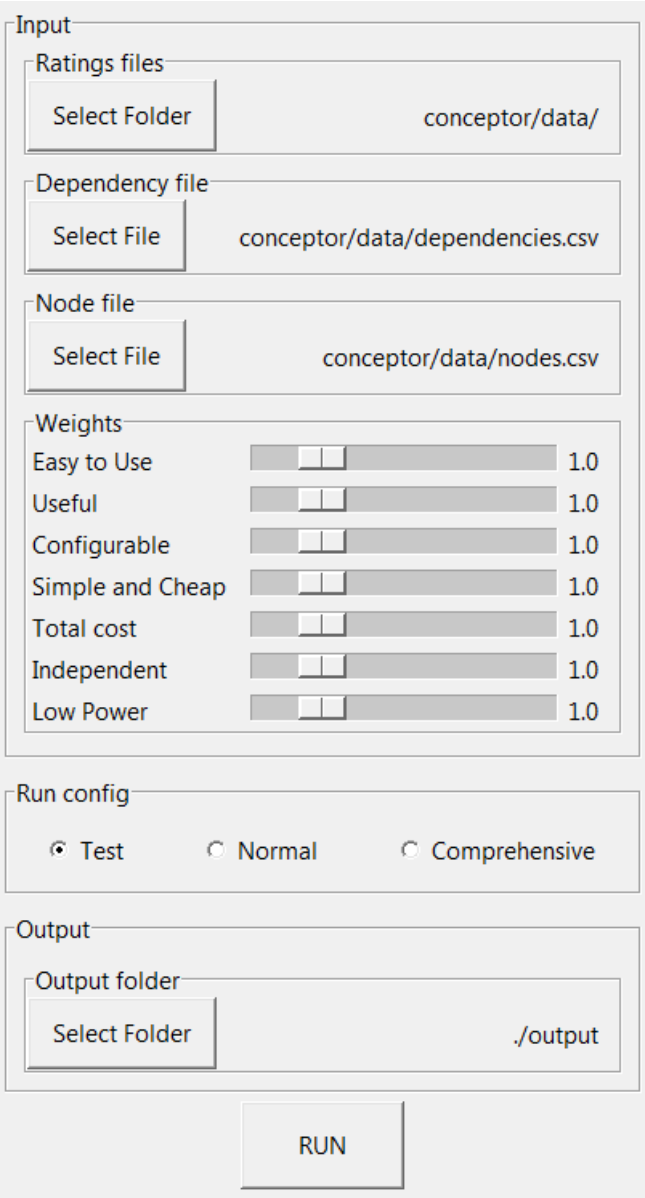


Figure 3.6: Screenshot of the user interface for the tool. The UI allows selecting files and folders to define the optimization problem, and allows the user to adjust sliders to weight different terms differently. There is also possibility to run the tool with different configurations. The test configuration runs the Monte-Carlo method 10 times, and find the robust solution and 3 diverse solutions. Normal mode runs 1000 iterations and find the robust solution and 5 diverse solutions while comprehensive runs 10000 iterations and find the robust solution and 6 diverse solutions.

Chapter 4

Evaluation

This chapter shows how the model stability was tested, by varying the weights and seeing how sensitive it is. This information is useful for evaluating the usefulness of the tool.

In this chapter we also show how the output of the tool was validated. The validation is a lightweight test to see how they compare to user generated ones. The test is conducted by having the tool generate solutions, and we also generate own solutions, then ask few people in what order they would rank the solutions. This was done as a part of the real use case test.

4.1 Sensitivity Analysis

The model's sensitivity to weight adjustments was tested by adjusting the weights and seeing how much does each term's weight affect the results. The test was conducted by running the model while adjusting a single term's weight with values 0, 0.5, 1, 1.5, 2 and 3, while keeping the rest at a default value of 1.0. This was repeated for each term. At each run the number of features were logged for each node. Optimal result is that the weight change causes a linear change in the number of features chosen for the affected nodes. Linear effect is easy to understand and adjusting the model to find suitable feature sets becomes easier.

In the tests there were 103 features, and in total 154 dependencies defined between the features. The data set is same as in chapter 4.2. The model was run with 100 iterations and $K = 4$. Each run took approximately 25 seconds.

The seven terms E, U, S, G, C, P, I were tested with 5 different weight values. With 4 diverse sets and a robust set this results in 7*5*5 = 175 data points, the model was also run with all weights at the default 1.0, resulting

in 5 more data points, thus in total there were 180 data points.

The weights' effect to the number of features in each node for each objective are visualized in figures 4.1 - 4.7. Each figure also contains a linear fit to see how linearly the number of features change in each node when the weights are adjusted. The robust solutions are marked with an orange cross marker, while the diverse solutions are marked with blue dots.

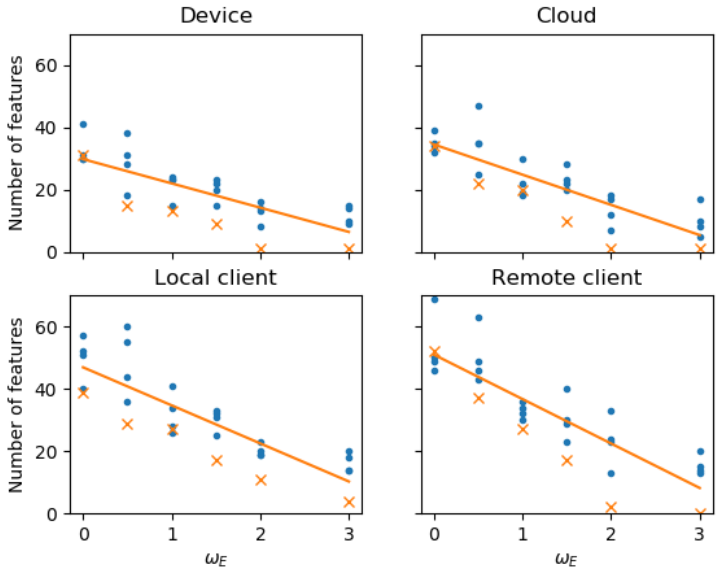


Figure 4.1: Effect of ω_E on the number of features on each node

Figure 4.1 shows that increasing ω_E causes the number of features to decrease fairly linearly in each node. Each graph is very linear when ω_E is between 0 and 2, but at 3 the graph start to curve a bit. This is likely caused by the number of features starting to plateau somewhere between 2 and 3.

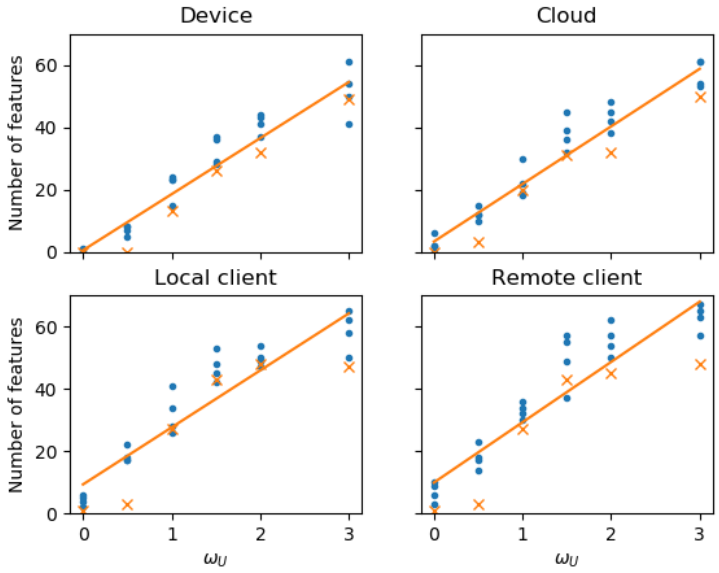


Figure 4.2: Effect of ω_U on the number of features on each node

Figure 4.2 shows that increasing ω_U causes the number of features to increase fairly linearly in each node. Quadratic fit could have been a better fit in Local client and Remote client.

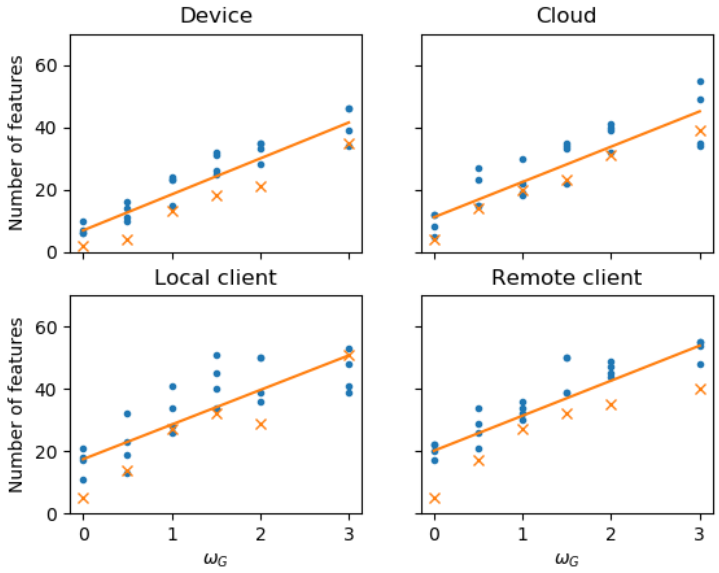


Figure 4.3: Effect of ω_G on the number of features on each node

Figure 4.3 shows that increasing ω_G causes the number of features to increase very linearly in each node. Only Local client might have been a better fit with a quadratic curve.

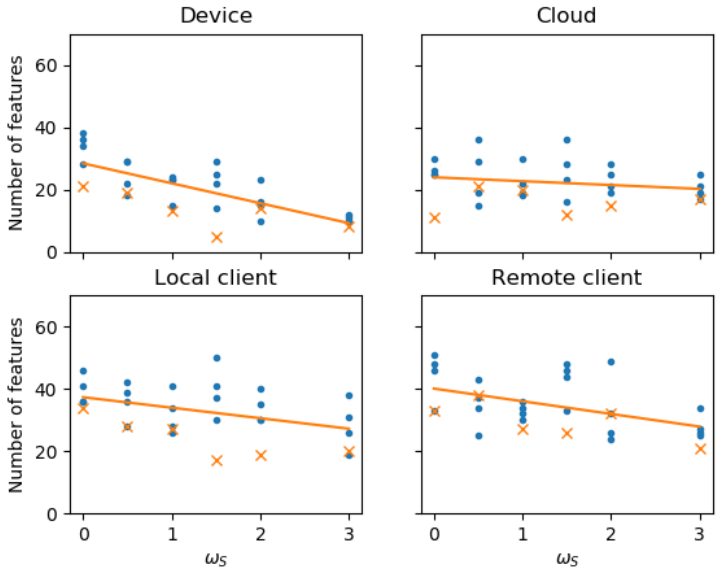


Figure 4.4: Effect of ω_S on the number of features on each node

Figure 4.4 shows that increasing ω_S causes the number of features to decrease fairly linearly, clearly faster in Device node than other nodes however. This is expected as The term S punishes for each feature in the device. This reflects to other nodes as well, because of the dependencies. Some features can only be selected if a specific feature is selected in the Device node, and now we limit features in the device node causing the features in other nodes to be limited as well.

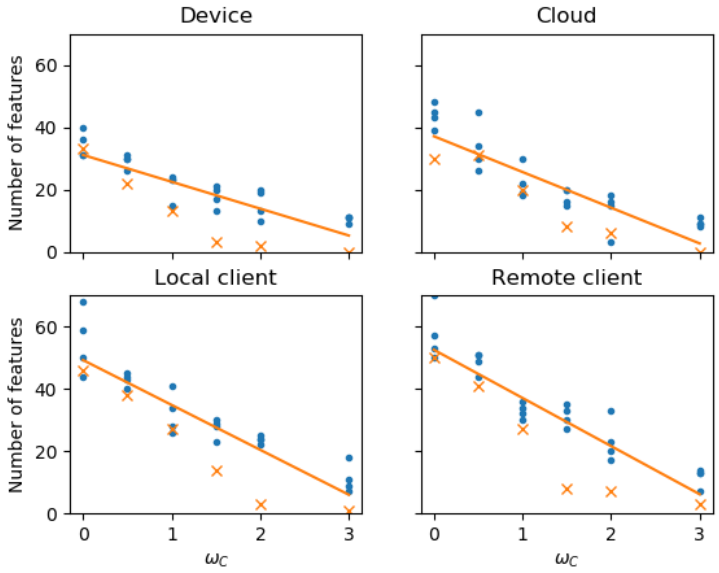


Figure 4.5: Effect of ω_C on the number of features on each node

Figure 4.5 shows that increasing ω_C causes the number of features to decrease linearly, slightly slower in Device node than other nodes however. This is expected as the term C punishes for features that are not in Device. This reflects to Device as well, because of the dependencies. There is no use selecting features in the device if they cannot be used in other nodes.

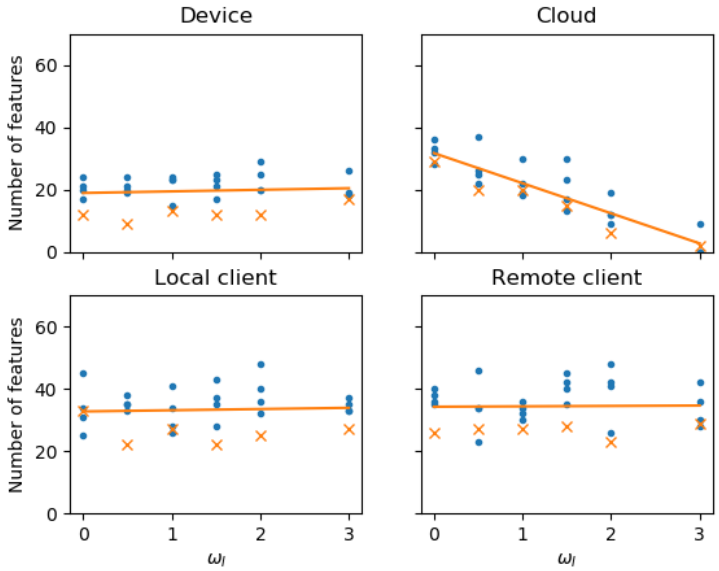


Figure 4.6: Effect of ω_I on the number of features on each node

Figure 4.6 shows that increasing ω_I causes the number of features to decrease linearly in the Cloud node, but other nodes are not affected. This is expected as well as the term I punishes for each feature in the cloud. This does not reflect on other nodes, because there is also arc from Device to Remote Client bypassing the cloud completely.

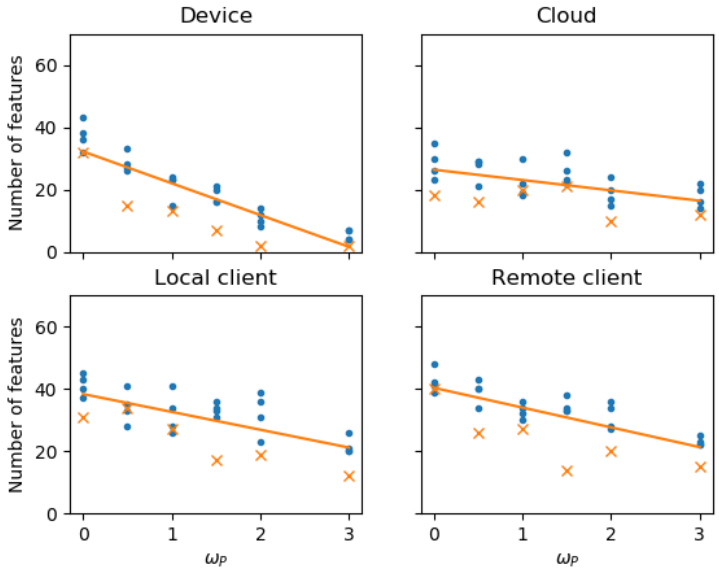


Figure 4.7: Effect of ω_P on the number of features on each node

Figure 4.7 shows that increasing ω_P causes the number of features to decrease linearly in the nodes, clearly faster in the Device node however. This has the same effect as explained previously with figure 4.4 causing the other nodes' features to decrease as well.

4.2 Real Use Case and validation

This section introduces how the method was used on a real use case in Vaisala. The main point of this chapter is to discuss how well this method works in actual work environment with a real project and timetables. We also do result validation in this section to see how well the computer generated solutions compare to user generated ones. This section consists of three parts. First we discuss how the input feature list was generated and then and then we discus how ratings and dependencies were generated for these features. Finally we see how good the results were in a light weight empirical study.

4.2.1 Feature Generation

The list of candidate features was generated based in a workshop which was done was done in co-operation with Vaisala personnel. Workshop event was created to come up with lot of varying features. The workshop was also

organized to prevent fixating only on certain point of view. Research has shown that co-designing and collective creation can be very powerful and can lead to more relevant results than individual creativity and is not just the sum of the individual creativities[7, 22, 23].

Prior to the actual workshop event, a short 30 minute briefing meeting was organized for the participants. The purpose was to introduce the subject and make sure everyone understands the purpose of the workshop. This was done as a separate event before the actual workshop to make sure everyone has prepared for the right thing and that the actual workshop event could focus on it's goals instead of introducing the event.

In this meeting the participants were also briefly introduced to the computational methods that would be used later. This was done to make the participants understand that the workshop event's purpose is not to find the optimal set of features but instead come up with as many different features that could be in the device, even useless and silly ideas were welcomed.

The participants consisted of 10 people. The number was kept fairly low to increase collaboration between all the participants. The workshop was organized in two parts.

The notes taken in the pre-study interviews act as an input for the first part of workshop. In total there were 298 notes, each containing an idea,observation or some other insight related to the subject. From the pre-study data and help of Vaisala personnel also different proto-personas were generated to represent the different users, customer groups and organizations that could use such a device. Proto-personas are a modified version of the personas that stimulate the same type of empathetic and user oriented thinking but with less investment in time[9]. These personas were used in the second part of the workshop. In total 9 different personas were found. Examples being "field technician" and "Technical support".

The purpose of the first part was to organize the data. This was done by utilizing method called Affinity clustering. Affinity clustering is a graphic technique for sorting items according to similarity. It helps identifying issues and insights, reveals thematic patterns, builds a shared understanding.[12, p. 40-41] We began with unsorted set of notes on table, and began grouping them towards similar notes. Once the groups started to form, they were given a label to describe the group. Example labels were "data visualization", "cloud platform", "price". Illustration of this method can be seen in figure 4.8. In total we identified 30 groups.

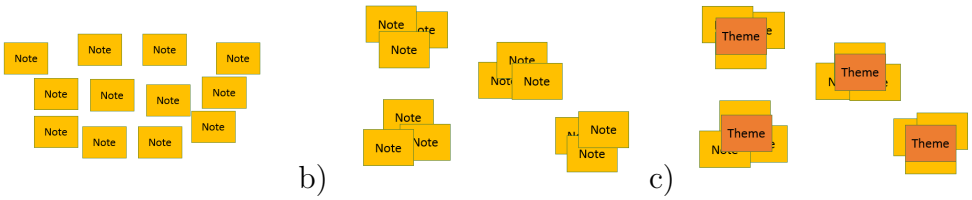


Figure 4.8: Illustration of the Affinity clustering method used to reveal themes in the interview data. a) The notes before clustering, ordered randomly, b) notes clustered based on similarity, c) clusters named based on the clusters theme.

Then in the second part, the actual ideation of features was done. This was done in the form of creative matrix, where the columns represented each persona, identified during the interviews and each row corresponded to one cluster of notes from the clustering phase.

Creative matrix is a format for sparking new ideas at the intersections of distinct categories. It helps generating large number of ideas, promotes divergent thinking and helps with thinking new and unusual ideas.[12, p. 62-63]

This makes Creative matrix a good choice for our purpose, as the purpose is to come up with as many different ideas as possible. This also helps to come up with the basic features that are needed and possibly some new features, that we had not considered before.

The participants were given an A0 paper sheet with the creative matrix printed on it, with the personas on top row, illusrated in figure 4.9a. Then the participants picked one of the clusters, and placed on the left most column. Then the participants started going from left to right filling features at each intersection illustrated in 4.9b. The direction was also important as the personas were set up so that they represented the life cycle of the device.

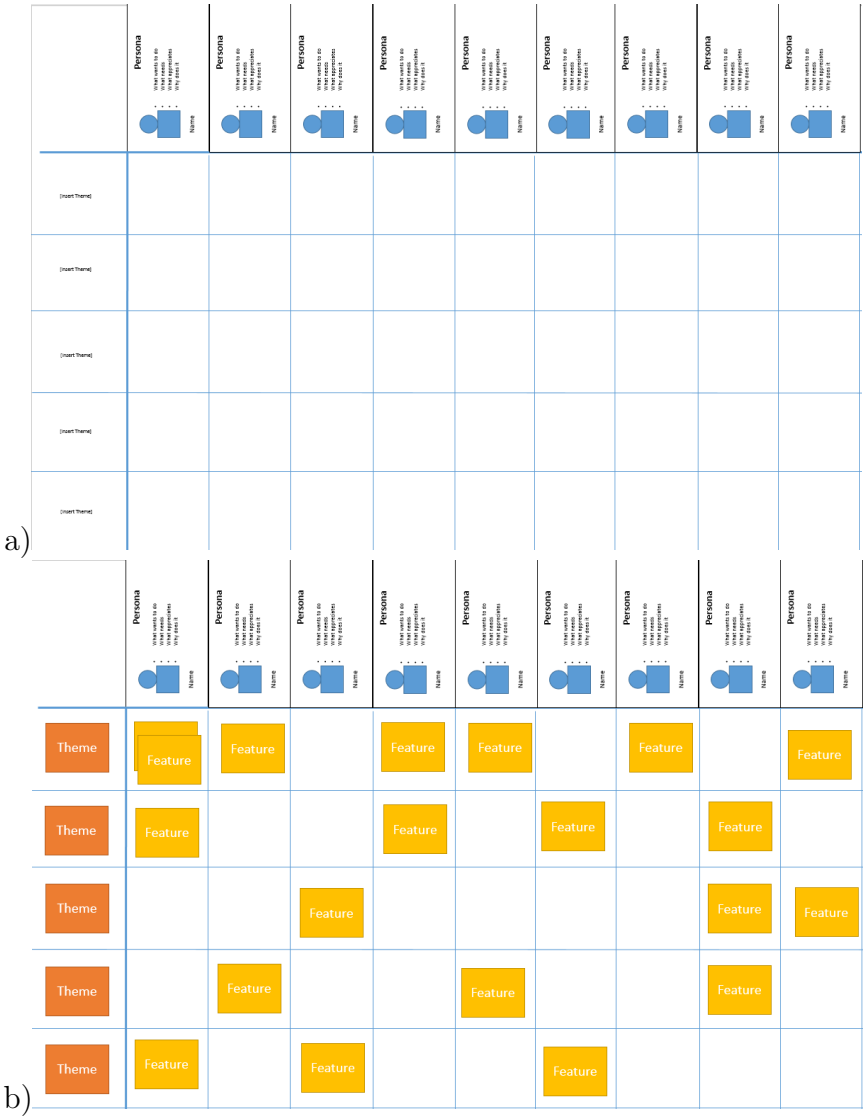


Figure 4.9: Illustration of Creative Matrix. a)Empty matrix with personas on the columns b) filled matrix, with themes on rows and corresponding features on the columns

After the workshop we had a list of all the possible functionalities the system could have. In total 103 different features were identified.

4.2.2 Ratings and Dependencies

A questionnaire was issued to all participants of the workshop. The questionnaire asked to rate each feature found in the workshop based on the properties described in chapter 3. The participants could also define dependencies between the features, and mark which feature was allowed in which node. The questionnaire contained two files, the ratings file and the dependencies file.

The ratings file contained a row for each feature, and each row had columns for each of the property the feature was to be rated for, columns for the allowed nodes, and a confidence column, as illustrated in figure 4.10. Each rating was on scale 0-3, allowed nodes either 0 or 1 confidence in range 1-3.

Feature	U	X	C	P	G	D	C	R	L	confidence
set communication Interval	0	0	0	0	0	1	1	1	1	2
view communication Interval	0	0	0	0	0	1	1	1	1	2
set measurement interval	0	0	0	0	0	1	1	1	1	2
view measurement interval	0	0	0	0	0	1	1	1	1	2
set device parameters	0	0	0	0	0	1	1	1	1	2
view device parameters	0	0	0	0	0	1	1	1	1	2
view device status	0	0	0	0	0	1	1	1	1	2
view measurement status	0	0	0	0	0	1	1	1	1	2
view communication device status	0	0	0	0	0	1	1	1	1	2
view sensor status	0	0	0	0	0	1	1	1	1	2

Figure 4.10: Illustration of ratings questionnaire. For each feature on a row there is column for usefulness(U),complexity(X),cost(C),power consuption(P) and configurability(G), allowed nodes, device(D), cloud(C), Remote Client(R), Local client(L) and confidence.

The dependency file contained a row and a column for each feature making it a 103×103 matrix. Each cell represented dependency for the feature on the row to the feature on the column. Possible values for each cell were empty or one of '**WE**', '**WA**', '**SE**' or '**SA**', representing **w**eak int**e**r, **w**eak intr**a**, **s**trong int**e**r and **s**trong intr**a** respectfully. This is illustrated in figure 4.11

	set communication Interval	view communication Interval	set measurement interval	view measurement interval	set device parameters	view device parameters	view device status	view measurement status	view communication device statu	view sensor status
set communication Interval	SE	SA								
view communication Interval	SA	SE								
set measurement interval			SE	SA						
view measurement interval			SA	SE						
set device parameters					SE	SA				
view device parameters					SA	SE				
view device status							WE			
view measurement status								WE		
view communication device status									WE	
view sensor status										WE

Figure 4.11: Illustration of dependencies questionnaire. Each cell represents a dependency from feature on the row to feature on the column. Values '**WE**', '**WA**', '**SE**' or '**SA**', represent **w**eak int**e**r, **w**eak intr**a**, **s**trong int**e**r and **s**trong intr**a** respectfully.

4.2.3 Validation

Validation of the results was done to see how good the resulting concepts are. The validation is conducted as light weight empirical study, to get some rough estimate how well the results compare to user generated ones. The validation is conducted by showing the relevant people few different concepts. some of which are computer generated and some are generated by ourselves. The concepts generated ourselves are based on the understanding that was gathered during the interviews and workshop.

We tested the tool with two set of concepts. Set one was called *MVP*, for minimum viable product, and set two was called *MVP+*, which was still minimal, but more fleshed out than the MVP. Use of these minimal concepts was done to allow us to make good enough concepts ourselves and push the limits of the tool. With small number of features each feature plays an important role and small number of features is easier to handle for a human generating the concept. This really tests how well the tool works, because the tool must only choose the only most important ones, and allowing the tool to choose large number of features, it could accidentally choose the important ones as well without considering them important and this mistake would not be noticeable. Also limiting the number of features allows more diverse

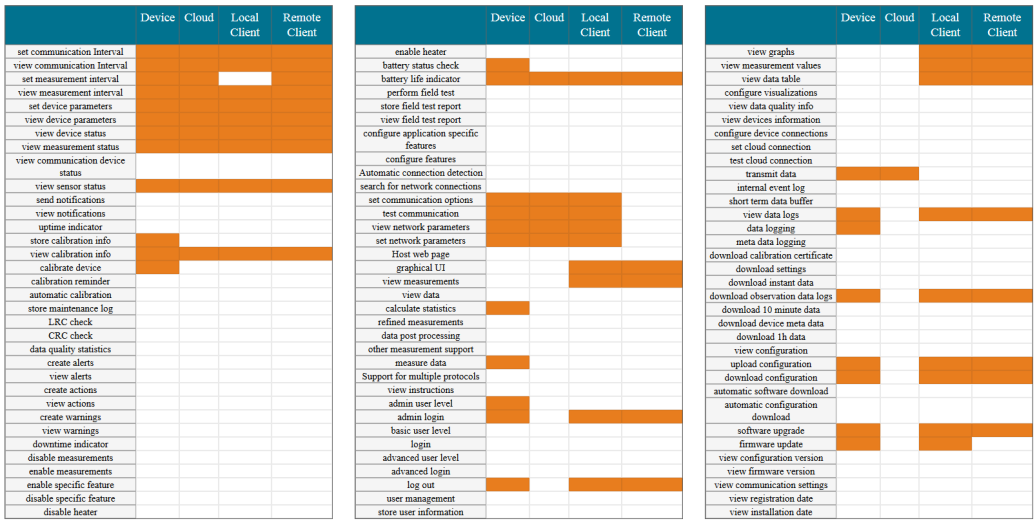
participant 1	*MVP*	E-D-B-A-C
	MVP+	A-D-B-C-E
participant 2	*MVP*	D-A-B-E-C
	MVP+	B-D-A-C-E
participant 3	*MVP*	E-D-B-A-C
	MVP+	A-B-E-C-D
participant 4	*MVP*	B-E-A-D-C
	MVP+	A-B-D-C-E

Table 4.1: Table showing participant rankings

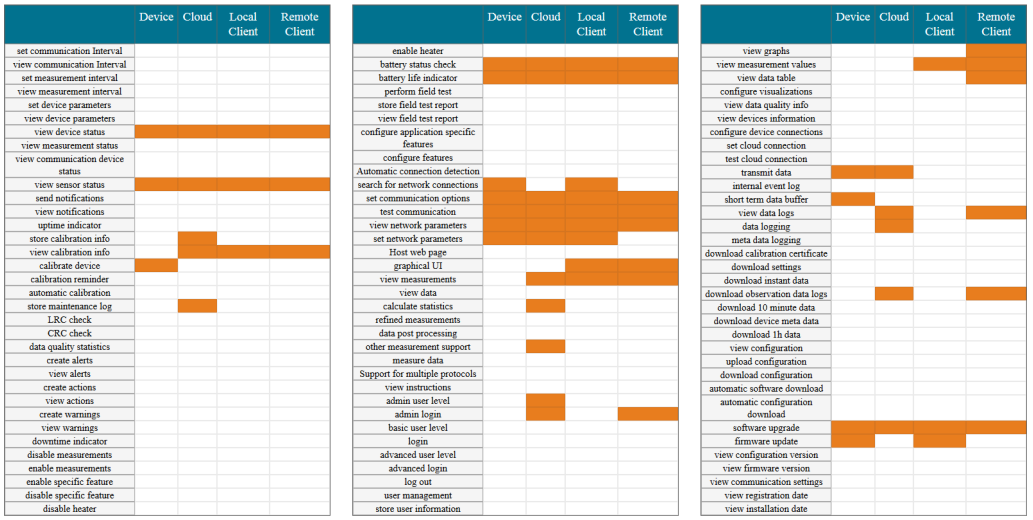
concepts.

Both of the concept sets contained 5 concepts labeled A,B,C,D and E. A and B were generated by us. C was the robust solution and E and D were diverse solutions, generated by computer. These concepts can be seen in figures 4.12 and 4.13. Observant reader might notice that the background on these images is white, as opposed to blue in figure 3.5. This is because the concepts here were printed on paper, and we wanted so save ink.

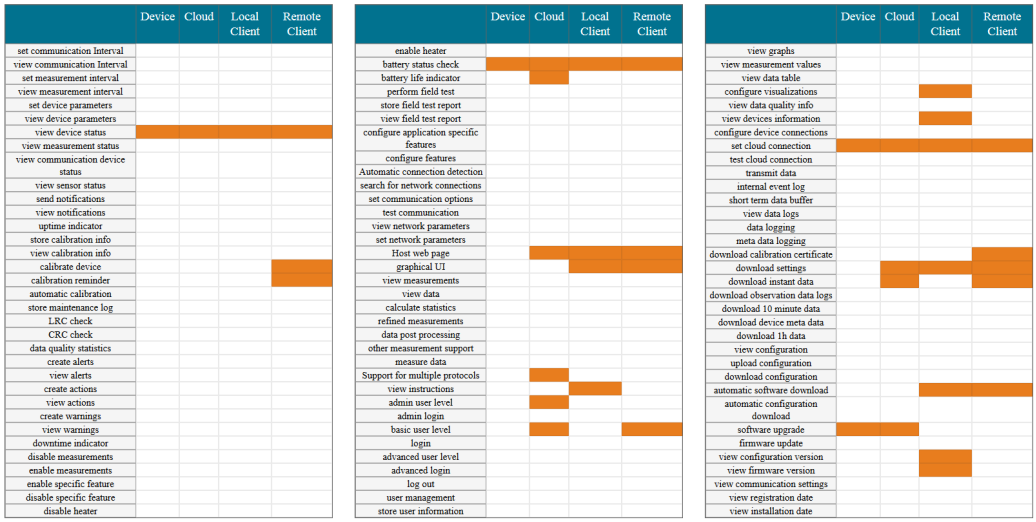
Four participants were asked to rate the concepts from best to worst based on their preferences and opinions. All four participants also participated in the workshop, so they already had an understanding of the scenario and features. The participants' rankings can be seen in table 4.1 and they are discussed in chapter 5.



(a) concept A

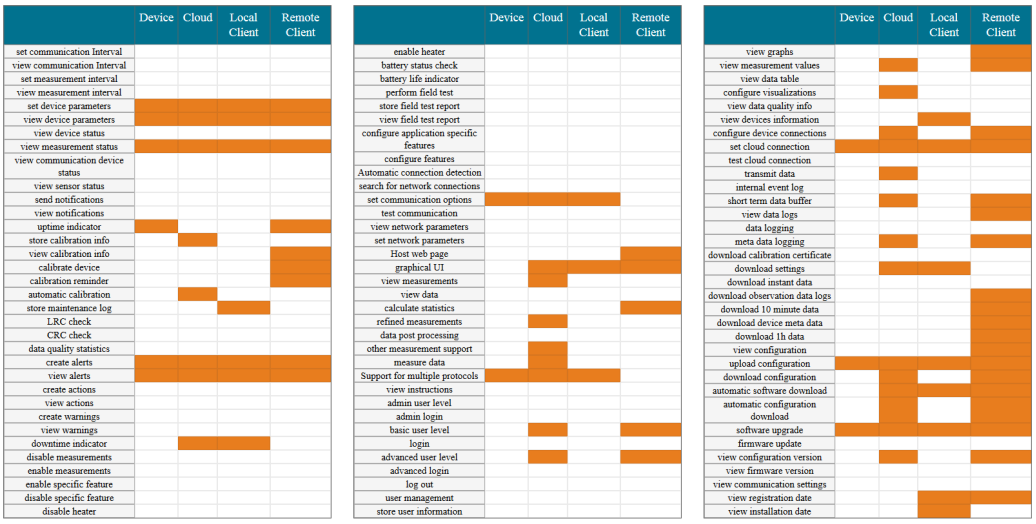


(b) concept B

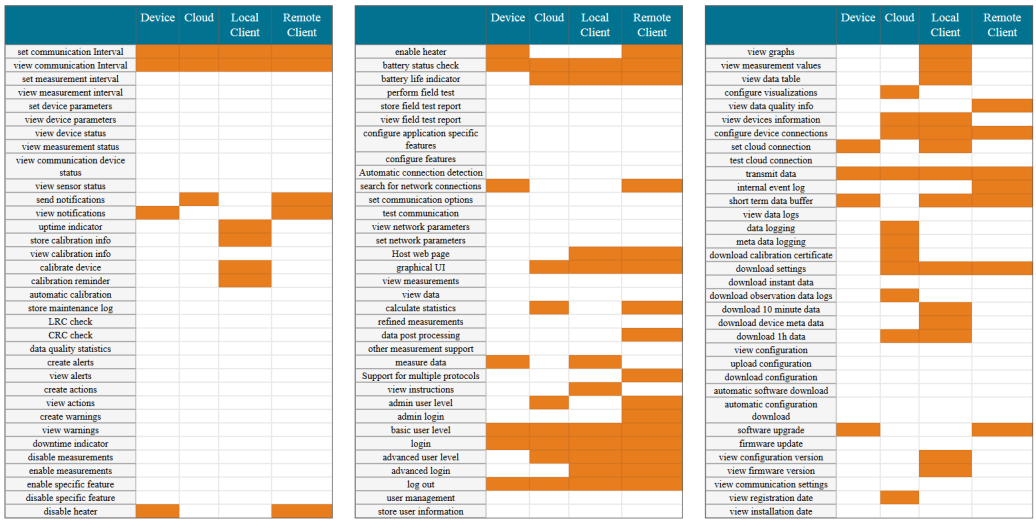


(c) concept C

Figure 4.12: Generated concepts for case *MVP*

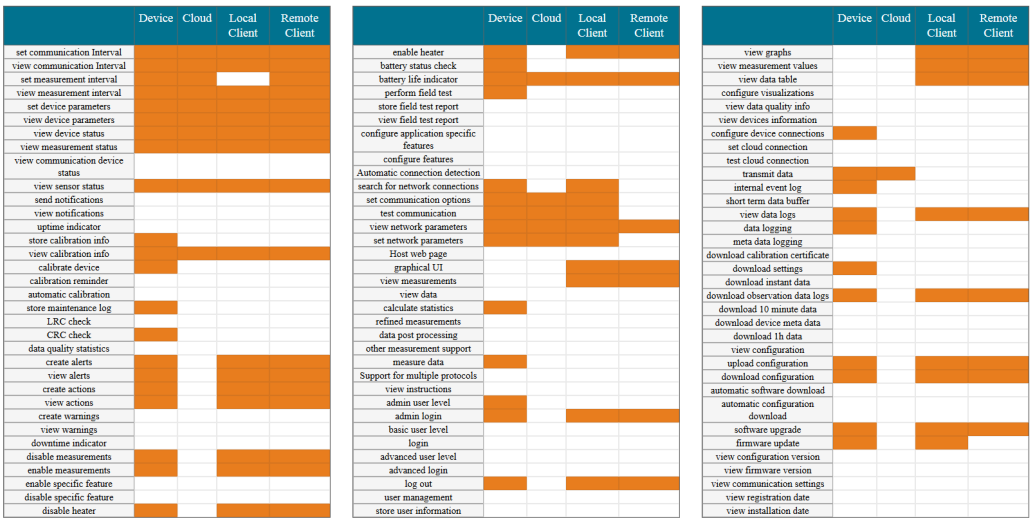


(d) concept D

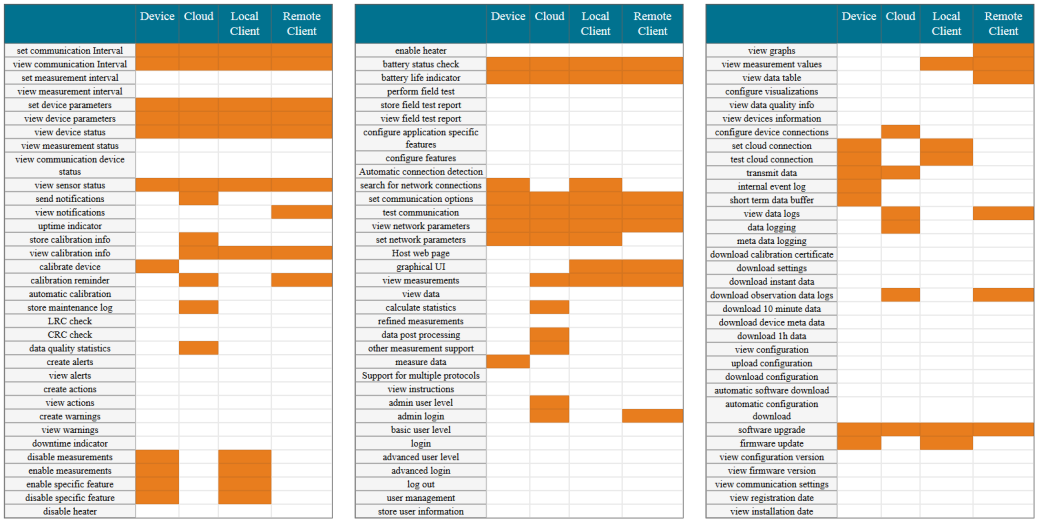


(e) concept E

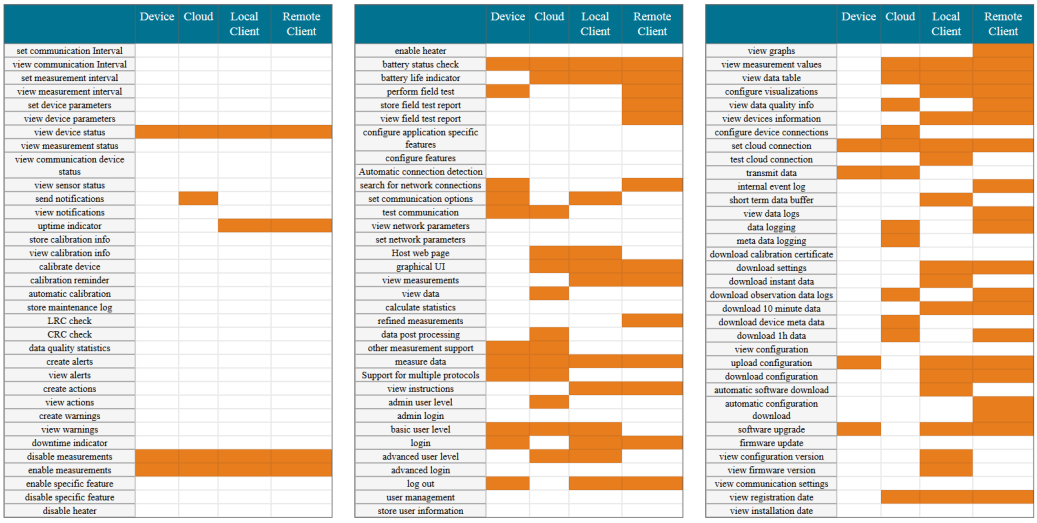
Figure 4.12: Generated concepts for case *MVP*



(a) concept A

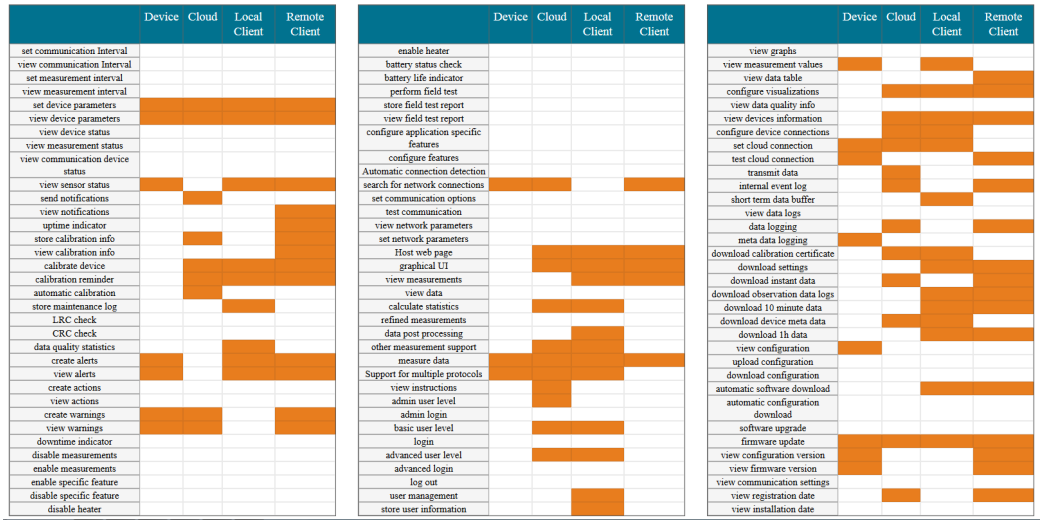


(b) concept B

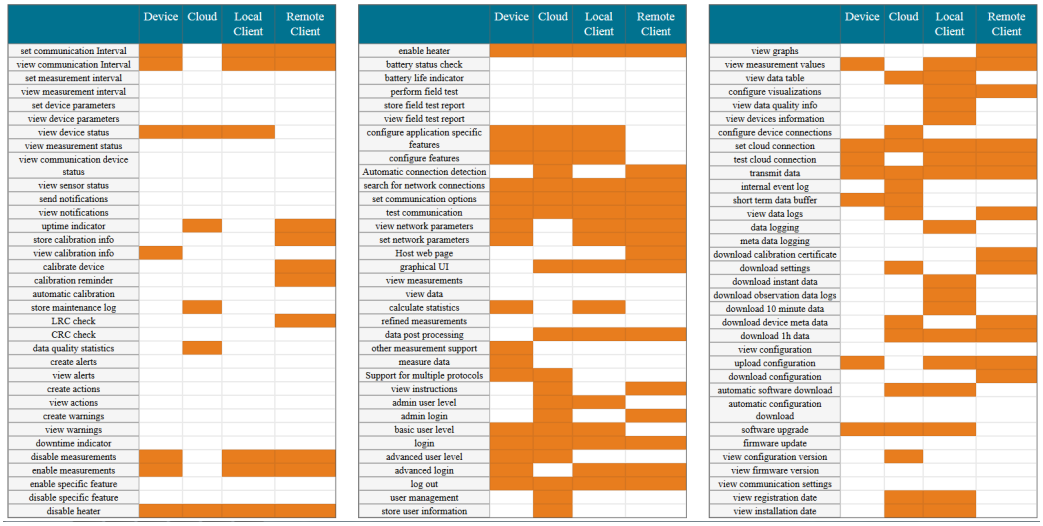


(c) concept C

Figure 4.13: Generated concepts for case *MVP+*



(d) concept D



(e) concept E

Figure 4.13: Generated concepts for case *MVP+*

Chapter 5

Discussion

The presented model combines the feature selection method used in work by Oulasvirta[19] and feature distribution method from work by Park[20]. The model also extends this work by introducing the different dependency types called inter and intra, and their variations called weak and strong. We also evaluated the model in a more realistic case in a real work environment with real project and timetables.

The model presented produces promising results. The weight adjustments cause predictable change in number of features chosen. Small adjustments cause a small change and large adjustments cause large change, and the change is fairly linear, making it easier to estimate the effect of a weight adjustment. This makes iterative design with the model more efficient, because making the adjustments is not just guessing and the effect of each term can be learned fairly easily. This is important because the model is intended to ease product concepting. If the model is very cryptic and produces seemingly random results, the designers are less likely to use the model.

Adjusting the weights "correctly" is naturally very important for finding the best results. To find a good starting point, one could take an existing product and try to adjust weights such that the resulting concepts are similar to that of the existing products feature sets. These weights could then be further adjusted based on the specifications of the new product. This method could also be used to find the most important features in current product by starting with the current product, and increasing weight for example total cost term to see which features are most crucial ones.

The real use case in chapter 4.2 taught us many things about the usage of the model. The people asked to fill in the questionnaire reported that the questionnaire was too laborious. This is also reflected by the fact that only 5 people of the 10 who were asked, filled the ratings at least partially and no-one filled the dependencies completely. One participant did try to

fill the dependencies as well, but had to give up, as he ran out of time. Understanding the differences between the dependencies and how they actually affect the results and then considering these for each pair of dependencies such that together they form a sensible dependency structure was too much work for too little preparation, training and time allocated.

We filled the questionnaires ourselves for the 103 features as well. Filling the questionnaire took 45+45 minutes, for the ratings and dependencies totaling 90 minutes. Naturally for us the dependencies were easier to fill as we have been working with them for a while, and have had time to assimilate the different dependencies. Many reported that the dependencies questionnaire was so overwhelming that they could not even get started. This is understandable as the questionnaire contained 103 features, meaning 103×103 matrix and each cell could have 5 different values, empty or one of the four dependency types. This totals to $103 * 103 * 5 = 53,045$ choices. Of course in practice the matrix is very sparse. Our filled matrix contained only 154 defined dependencies between the features. Once we got the hang of filling the matrix it was actually fairly straight forward. Some features were very obviously dependent on only a single specific feature, e.g. 'log out' and 'log in', while some were obviously not dependent on anything, e.g. 'measure data'. In future the questionnaire should be organized in a workshop form, making quitting have higher threshold. This could be organized by first going through all the features and see that every one understands the features in the same way, then together fill out the dependency matrix. Finally everyone should fill out the ratings on their own, but still in 'controlled' environment in the workshop, where people can easily ask for clarification and help. The ratings should still be done individually, as the model relies on sampling the ratings, and if there is no variation the samples are always identical. This way everyone's opinion is also considered, instead of only the most dominant opinion.

In the validation the computer generated concepts did compare to the human generated ones fairly well. In the *MVP* case computer generated concepts were considered best by almost all participants, and in many cases the second best place was also computer generated. However in *MVP+* case the user generated was always best, but computer generated was still often the second best. Interestingly the robust solution, concept C in both cases, was always last or second to last. This is likely because the robust solutions have lowest number of features selected. This is caused by the robust search algorithm finding the compromise among all the solutions. This in turn causes the robust set to mainly contain only those features that are selected in most of the other solutions as well. However this could be used as an advantage by setting the weights such that the model produces solutions

with very high number of features. In this case the robust solution might work better than the diverse solutions, as the robust chooses the solutions that are common to most other solutions.

The participants often reported that they ranked the concepts by presence of certain key features that they thought were the most important ones. Many also ignored the information on which node the feature was chosen, and mainly focused on whether the feature is selected at all.

The low participant rate of 5 participants, might have caused the ratings to not be very representative of the true values for each feature. With more participants each feature could have more realistic and less noisy values. Examining the answers the participants gave for the ratings we noticed large variation in the answers. This causes the random samples to be sampled on very wide range, making the results more random. This does also make our findings of the computer generated concepts being comparable to human generated slightly unreliable. However as explained in 4.2.3 the MVP and MVP+ cases were intentionally hard for the computer and the fact that the model still produced similarly good results compared to our understanding shows that this model is very promising. This claim is based on observing that the top two best solutions in the validation were fairly well tied between human and computer generated concepts.

Future expansions to the model could include allowing multiple different dependency types between two features. In the current form the model only allows a single dependency type between two features. For example a feature could have weak inter- *and* weak intra-dependency to some feature, forcing the other feature to be in the same node or in a connected node. This would be useful for example in defining dependency from a feature that needs some other feature to be present but is does not matter where the other feature is implemented. For example 'view data logs' needs 'data logging' but it does not really matter whether they are in same or different nodes, as long as there is some way of accessing the dependent feature. The actual ILP model does support this even in the current form, but as the dependencies are read from a *.csv* file, where each cell corresponds to a single dependency, the multiple dependencies between two features cannot be defined. This need came up too late in the development so it was left out from this version, but the implementation should be fairly straight forward.

Another addition to the model would be more dependency types. One such additional dependency, or restriction actually, would be to allow defining two features to not be present in the same node together. This would allow defining choices better, where only one option is allowed. For example 'data buffer' and 'data logging' basically do the same thing but in different scale. The buffer stores only a small amounts of data to be transmitted later,

while the logging stores large amounts of data more permanently. These two have their uses but it does not make much sense to have both. Defining a constraint that allows maximum of one of these to be present in a node would solve this.

Another possible future addition to dependencies would be to find a sensible way to aggregate multiple dependency files together. The dependencies in a file together form a dependency structure between all the features. Combining dependencies from two dependency files is not a trivial task, and this could be subject for future research to determine if it is possible to combine two dependency structures in a sensible manner and if so, is it useful. One possible starting point for such research could be finding chains from the dependency files and treating them as single units and combining those from different files. For example in figure 3.4 there would be two chains, "data logging" → "measure data" and "data logging" → "receive data" → "transmit data" → "measure data". The dependencies could also be defined in similar way by defining tree structures in some graphical manner. This could also be less overwhelming for participants to do than the current way.

The model was made into a tool with a simple user interface to ease the use and testing of the model. The tool is described in section 3.5. The tool is rather generic in the sense that it can take any amount of ratings files with any number of features(as long as they are same in all the files). It also allows any configuration of dependencies read also from a file, it even allows defining the nodes and arcs with a file. But the objective function is hard coded into the model. This makes defining additional nodes fairly useless, as the objective function does not consider them. For future work this could be fixed by coming up with a way to define the objective function externally as well. This would make the tool very generic and suitable for many other applications, products and product types as well.

The terms in the objective function are linear, which makes them simple to understand and their behavior is easy to predict. However changing the ILP solver to Mixed Integer Programming(MIP) solver would allow quadratic objectives and constraints as well. Quadratic functions could be useful from example in the cost objectives. With properly adjusted curve, the cost punishment would be lower for small number of features than in the linear case, and higher for very large numbers. Both the previously mentioned solvers, Cbc[1] and Gurobi[2], already support MIP.

With these changes the model could be improved and made into more general concepting tool to produce concepts for many kinds of products. As of now it allows only very similar product to be concepted. Based on the feedback from Vaisala the model could be taken into use in product concepting phase with some modifications. Most importantly the input data

forming should somehow be made easier. This seemed to be the bottleneck in our research as well. Even in the current form we believe that the model could be useful for product concepting, if given proper training for it.

Chapter 6

Conclusion

The problem that this thesis aims to solve is, how to optimally select features for internet connected field device, the accompanying cloud service and client devices such that designer goals, feature properties and relationships to other features are considered. This information could be used during concepting of new products.

We solve this problem by utilizing computational methods to handle possibly large amount of features and their relationships. We model the problem as an ILP problem. We rate each feature for its usefulness, complexity, cost and configurability. We define four different types of dependencies between the features, weak and strong inter-dependency and weak and strong intra-dependency. The inter dependencies are dependencies across an arc between two nodes in the graph describing the devices and their connections. The intra-dependencies are dependencies in a node. Weak and strong variants correspond to logical *or* and *and* case respectfully when there are multiple dependencies for a single feature. We also utilize Monte-Carlo method to find diverse and robust solutions to the problem, and then visualize the sets in grid form to easily see which features were selected and where and which features were not selected at all. The system was also given an simple user interface to adjust preferences for each of the 7 terms: Ease of use and install, Usefulness, Configurability, Simplicity and cost of device, Overall cost, Low power and Independence.

This work builds on previous work by combining feature selection and distribution methods from two papers and extends them by introducing different dependency types and evaluates this model in a real use case.

Sensitivity analysis of the model was carried out by adjusting weights for each objective one by one and seeing how much the adjustment of each weight affected the number of features chosen. The model is stable, as the number of features change linearly as the weights are adjusted.

The issues we encountered with the real use case were related to input data formation. Forming the data was considered laborious and time consuming. The dependencies were considered too complicated to understand. The validation showed that the model could still produce competitive result when compared to user generated concepts. The issues with current implementation could be fixed by organizing training and workshop events where people together work on the input data. This could lead to even better results and more useful product concepts, as everyone understands the features similarly.

Based on these results we find that the model looks promising, and with additional work and training the tool could be improved to produce even more interesting and useful concepts. With more additions to the model it could be turned into more general concepting tool as well, as now it only can be used in similar scenarios.

Bibliography

[1] Coin-or cbc user guide. https://www.coin-or.org/Cbc/cbcuserguide.html. referenced: 2018-04-18.

[2] Gurobi optimizer. http://www.gurobi.com/products/gurobi-optimizer. referenced: 2018-05-09.

[3] Pulp project page. https://pythonhosted.org/PuLP/. referenced: 2018-04-18.

[4] Vaisala website wxt530 product page. https://www.vaisala.com/en/products/instruments-sensors-and-other-measurement-devices/weather-stations-and-sensors/wxt530. referenced: 2018-01-08.

[5] Albritton, M. D., and McMullen, P. R. Optimal product design using a colony of virtual ants. *European journal of operational research 176*, 1 (2007), 498–520.

[6] Beyer, H.-G., and Sendhoff, B. Robust optimization–a comprehensive survey. *Computer methods in applied mechanics and engineering 196*, 33-34 (2007), 3190–3218.

[7] Bissola, R., and Imperatori, B. Organizing individual and collective creativity: Flying in the face of creativity clichés. *Creativity and Innovation Management 20*, 2 (2011), 77–89.

[8] Bryant, C. R., McAdams, D. A., Stone, R. B., Kurtoglu, T., and Campbell, M. I. A computational technique for concept generation. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2005), American Society of Mechanical Engineers, pp. 267–276.

[9] Buley, L. *The user experience team of one: A research and design survival guide*. Rosenfeld Media, 2013.

[10] Elsen, C., Häggman, A., Honda, T., and Yang, M. C. Representation in early stage design: An analysis of the influence of sketching and prototyping in design projects. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (2012), American Society of Mechanical Engineers, pp. 737–747.

[11] Fu, K., Murphy, J., Yang, M., Otto, K., Jensen, D., and Wood, K. Design-by-analogy: experimental evaluation of a functional analogy search methodology for concept generation improvement. *Research in Engineering Design 26*, 1 (2015), 77–95.

[12] Institute, L. *Innovating for People: Handbook of Human-centered Design Methods*. LUMA Institute, 2012.

[13] Kang, I.-S., Na, S.-H., Kim, J., and Lee, J.-H. Cluster-based patent retrieval. *Information processing & management 43*, 5 (2007), 1173–1182.

[14] Liu, Y.-C., Chakrabarti, A., and Bligh, T. Towards an ideal approach for concept generation. *Design Studies 24*, 4 (2003), 341–355.

[15] Marks, J., Andalman, B., Beardsley, P. A., Freeman, W., Gibson, S., Hodgins, J., Kang, T., Mirtich, B., Pfister, H., Ruml, W., et al. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 389–400.

[16] Murphy, J., Fu, K., Otto, K., Yang, M., Jensen, D., and Wood, K. Function based design-by-analogy: a functional vector approach to analogical search. *Journal of Mechanical Design 136*, 10 (2014), 101102.

[17] Murphy, J. T. *Patent-based analogy search tool for innovative concept generation*. PhD thesis, 2011.

[18] Noyes, J., and Weisstein, E. W. Linear programming. `http://mathworld.wolfram.com/LinearProgramming.html`. referenced: 2018-05-22.

[19] Oulasvirta, A., Feit, A., Lähteenlahti, P., and Karrenbauer, A. Computational support for functionality selection in interaction design. *ACM Transactions on Computer-Human Interaction (TOCHI) 24*, 5 (2017), 34.

[20] Park, S., Gebhardt, C., Rädle, R., Feit, A., Vrzakova, H., Dayama, N., Yeo, H.-S., Klokmose, C., Quigley, A., Oulasvirta, A., et al. Adam: Adapting multi-user interfaces for collaborative environments in real-time. *arXiv preprint arXiv:1803.01166* (2018).

[21] Simonson, I., Carmon, Z., and O'curry, S. Experimental evidence on the negative effect of product features and sales promotions on brand choice. *Marketing Science 13*, 1 (1994), 23–40.

[22] SonicRim, L. S. Collective creativity. *Design 6*, 3 (2001), 1–6.

[23] Steen, M., Manschot, M., and De Koning, N. Benefits of co-design in service design projects. *International Journal of Design 5 (2) 2011, 53-60* (2011).

[24] Tang, D., Yin, L., and Ullah, I. Matrix-based computational concept design with ant colony optimization. In *Matrix-based Product Design and Change Management*. Springer, 2018, pp. 55–82.

[25] Thompson, D. V., and Norton, M. I. The social utility of feature creep. *Journal of Marketing Research 48*, 3 (2011), 555–565.

[26] Tovey, M., Porter, S., and Newman, R. Sketching, concept development and automotive design. *Design studies 24*, 2 (2003), 135–153.

[27] Visser, W. Designing as construction of representations: A dynamic viewpoint in cognitive design research. *Human–Computer Interaction 21*, 1 (2006), 103–152.

[28] Weisstein, E. W. Direction. `http://mathworld.wolfram.com/Direction.html`. referenced: 2018-05-23.

[29] Weisstein, E. W. Integer programming. `http://mathworld.wolfram.com/IntegerProgramming.html`. referenced: 2018-05-22.

[30] Weisstein, E. W. Monte-carlo method. `http://mathworld.wolfram.com/MonteCarloMethod.html`. referenced: 2018-05-22.

[31] Yang, M. C. Observations on concept generation and sketching in engineering design. *Research in Engineering Design 20*, 1 (2009), 1–11.

Appendix A

Interviewing template

- Who are you?
  - Title/job description?
  - How long have you been at Vaisala?
- Who uses the device?
  - From what kind of organization they are (universities, meteorological institutes, airports...)?
  - What is special about them?
  - Technical know-how
  - Special requirements
- What is the device used for?
  - Why have customers bought the device?
  - What does the device do?
  - What is the devices purpose?
  - When is the device used?
  - Are there requirements that current devices cannot do?
- Where/how is the device used?
  - Why is the device accessed?
  - In what situations is a remote connection used?
    - Who does that?
  - In what situations is a local connection used?

        * Who does that?

- Lifespan of device
    - What information do clients need about the device before purchase?
    - What kinds of problems do customers have with operating/installing the device?
    - Why is the device decommissioned?

Appendix B

Code snippets

In following sections there are few of the most crucial parts of the Python implementation as snippets. Each section briefly describes what the part does and then the actual code is shown.

B.1 Reading dependencies

For each row, with row number $fnum$, in the dependency file we read the dependencies for feature with index $fnum$. For inter-dependencies we add a dependency for each head node in every arc. Intra-dependencies are made for each node.

```
dnum = 0
for n in range(1,cols):
   if row[n] == "":
      dnum = dnum + 1
   if row[n] == "WE":
      dnum = dnum + 1
      for con in self.connections:
         self.weak_inter[con[1]+"_f"+str(fnum)].append(con[0]+"_f"+str(dnum))
   if row[n] == "WA":
      dnum = dnum + 1
      for nd in self.nodes:
         self.weak_intra[nd+"_f"+str(fnum)].append(nd+"_f"+str(dnum))
   if row[n] == "SE":
      dnum = dnum + 1
      for con in self.connections:
         self.strong_inter[con[1]+"_f"+str(fnum)].append(con[0]+"_f"+str(dnum))
   if row[n] == "SA":
      dnum = dnum + 1
```

```
      for nd in self.nodes:
         self.strong_intra[nd+"_f"+str(fnum)].append(nd+"_f"+str(dnum))
```

B.2 Weak dependencies

There exists a list $weak\_inter[v]$ that describes all features that the feature v has weak inter dependency to. The first for loops define a variable between feature v and d to describe whether the weak inter-dependency between them is satisfied. The second loops does the same for weak intra-dependencies similarly using list $weak\_intra[v]$. Finally we enforce atleast one of these variables to be satisfied.

```
def addWeakdependencyConstraints(self):
      #define dependecy sat = 1 if dependency between v and d is
          satisfied
      #dependecy 1: weak inter dependecy 2: weak intra dependecy

      #weak inter-dependencies
      for v in self.weak_inter:
         for d in self.weak_inter[v]:
            for i in range(len(self.connections)):
               c = self.connections[i]
               #check that the connection arc is between the nodes
                   of features v and d
               if c[0]+"_" == d[:2] and c[1]+"_" == v[:2]:
                  con = "c"+str(i)
                  #sat = d and c
                  self.prob += self.ILP_dependencysat[v+"_1_"+d] >=
                      self.ILP_variables[d] + self.ILP_variables[con]
                      - 1, ""
                  self.prob += self.ILP_dependencysat[v+"_1_"+d] <=
                      self.ILP_variables[d], ""
                  self.prob += self.ILP_dependencysat[v+"_1_"+d] <=
                      self.ILP_variables[con], ""
                  break

      #weak intra-dependencies
      for v in self.weak_intra:
         for d in self.weak_intra[v]:
            self.prob += self.ILP_dependencysat[v+"_2_"+d] ==
                self.ILP_variables[d], ""
```

```
#require at least one of the weak dependencies to be
    satisfied for each feature
for v in self.variables:
   weaklist = []
   if v in self.weak_intra:
      for d in self.weak_intra[v]:
         weaklist.append(self.ILP_dependencysat[v+"_2_"+d])
   if v in self.weak_inter:
      for d in self.weak_inter[v]:
         weaklist.append(self.ILP_dependencysat[v+"_1_"+d])
   if len(weaklist) > 0:
      self.prob += pulp.lpSum(weaklist) >=
          self.ILP_variables[v], "weak depenendencies for "+v
```

B.3 Strong dependencies

Strong dependencies also have lists $strong\_inter[v]$ and $strong\_intra[v]$ defining the features feature v has dependencies to. The first for loops define a variable between feature v and d to describe whether the weak interdependency between them is satisfied. Then we define that all of them must be satisfied, such that if the node where feature v is selected is the head node for multiple arcs, it is enough that one of the tail nodes satisfy the dependency.

```
def addStrongdependencyConstraints(self):
   #define dependecy sat = 1 if feature can be satisfied
   #and force it to be satisfied
   #dependecy 3: strong inter dependecy 4: strong intra dependecy

   #inter dependencies
   for v in self.strong_inter:
      for d in self.strong_inter[v]:
         for i in range(len(self.connections)):
            c = self.connections[i]
            #check that the connection arc is between the nodes
                of features v and d
            if c[1]+"_" == v[:2] and c[0]+"_" == d[:2]:
               con = "c"+str(i)
               self.prob += self.ILP_dependencysat[v+"_3_"+d] >=
                   self.ILP_variables[d] + self.ILP_variables[con]
```

```
                - 1, ""
            self.prob += self.ILP_dependencysat[v+"_3_"+d] <=
                self.ILP_variables[d], ""
            self.prob += self.ILP_dependencysat[v+"_3_"+d] <=
                self.ILP_variables[con], ""

        sumlist = []
        for i in range(len(self.connections)):
           c = self.connections[i]
           if c[1]+"_" == v[:2]:
              nd = c[0]+"_" + d[2:]
              sumlist.append(self.ILP_dependencysat[v+"_3_"+nd])

        self.prob += pulp.lpSum(sumlist) >=
            self.ILP_variables[v], "strong inter
            dependencies"+v+"-"+d

    #intra dependencies
    for v in self.strong_intra:
       for d in self.strong_intra[v]:
          self.prob += self.ILP_dependencysat[v+"_4_"+d] ==
              self.ILP_variables[d], ""
          self.prob += self.ILP_dependencysat[v+"_4_"+d] >=
              self.ILP_variables[v], ""
```

B.4 Reading feature ratings

For each feature, i.e. row in the ratings file we have four dictionaries, one for each node, that contains the name of the feature, Sampler object for determining whether the feature is allowed in the node, and 5 Sampler objects for each of the usefulness, complexity, cost, power requirements and configurability. For each row the values for these aspects are added to the samplers, the value on column 12 indicates how many times the value is added to each sampler. This is used to allow weighting user answers differently based on their confidence. Negative values are discarded to allow answering only partially. $fnum$ indicates the row number, and it is used as feature index number.

```
if self.nodes[0]+"_f"+str(fnum) not in self.variables:
   for n in self.nodes:
      self.variables[n+"_f"+str(fnum)] = dict([ ("name",row[0]),
```

```
                                    ("allowed",Sampler()),
                                    ("u",Sampler()),
                                    ("x",Sampler()),
                                    ("c",Sampler()),
                                    ("p",Sampler()),
                                    ("g",Sampler())
                                ])
    if row[1] != '0' or row[2] != '0' or row[3] != '0' or row[4] !=
        '0' or row[5] != '0':
       #if row has at least one rating, add the row to samplers,
           otherwise skip row.
          nodeindex = 0
          for n in self.nodes:
             for i in range(0,int(row[8+len(self.nodes)])):
                var_u = int(row[1])
                #add only positive values, to allow partial answers
                #by marking unanswered cells with -1
                if var_u >= 0:
                    self.variables[n+"_f"+str(fnum)]["u"].addValue(var_u)
                var_x = int(row[2])
                if var_x >= 0:
                    self.variables[n+"_f"+str(fnum)]["x"].addValue(var_x)
                var_c = int(row[3])
                if var_c >= 0:
                    self.variables[n+"_f"+str(fnum)]["c"].addValue(var_c)
                var_p = int(row[4])
                if var_p >= 0:
                    self.variables[n+"_f"+str(fnum)]["p"].addValue(var_p)
                var_f = int(row[5])
                if var_f >= 0:
                    self.variables[n+"_f"+str(fnum)]["g"].addValue(var_f)
                self.variables[n+"_f"+str(fnum)]["allowed"].addValue(int(row[7+nodeindex]
             nodeindex = nodeindex + 1
```

B.5 Sampler class

The sampler class that is used for ratings. A value for feature's aspect is generated by combining multiple ratings into a normal distribution and then sampling this distribution.

```
class Sampler:
```

```
def __init__(self,values = None):

   self.sample = 0
   if values:
      self.mean  = np.mean(values)
      self.std   = np.std(values)
      self.values = values[:]
   else:
      self.values = []
      self.mean  = 0
      self.std   = 0

def addValue(self,value):
   #add new value to list of values, and compute new mean and std
   self.values.append(value)
   self.mean  = np.mean(self.values)
   self.std   = np.std(self.values)

def newSample(self):
   #generates new sample
   self.sample = np.random.normal(self.mean,self.std)
   return self.getSample()

def getSample(self):
   return self.sample
```