Institute for Visualization and Interactive Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr.

Simulation and Adaptation of Contextual Bandit Algorithms for IoT Service Discovery

Jenny Schmalfuss

Course of Study:	Simulation Technology
Examiner:	Prof. Dr. Albrecht Schmidt
Supervisor:	Dipl.-Medieninf. Tilman Dingler, Dr.-Ing. Darren Carlson
Commenced:	May 15, 2016
Completed:	November 14, 2016
CR-Classification:	I.2.6

Abstract

As the emerging Internet of Things (IoT) makes an increasing ammout of connected services accessible to a broad range of users, the identification of contextually relevant services becomes an indispensable task.

In order to evaluate existing methods for contextual IoT service recommendation, we develop an ambient space simulation to emulate huge numbers of IoT services and user interactions. Secondly we investigate a new extension of the previously evaluated single IoT service recommendation - the recommendation of composite services consisting of multiple services to perform a mutual task. To address this challenge, we construct and implement a framework called ConComM to identify services that are likely to work well together for a joint task. Our previously developed ambient space simulation is then used to evaluate our frameworks performance.

The framework itself utilizes a novel k-cut algorithm based on a modification of the existing k-cut procedure SPLIT. We call this new procedure SPLIT$_{rel}$ and show it to outperform all tested benchmark algorithms for minimum k-cuts on graphs used by ConComM. Our experiments prove ConComM to significantly push the performance of an existing single service recommendation approach for composite service recommendation.

Within this work we do not only provide a simulation that allows to evaluate recommendation systems in environments densly filled with IoT services. We also develop a framework that enables contextual bandit algorithms to provide improved recommendations for composite services.

Contents

Sections marked with * form the propaedeuticum.

List of Figures

1 Introduction

Today prepares us for tomorrow.

(Joe Buddon)

With the Internet of Things (IoT) a huge amount of services enters the web and becomes accessible to users as they move through their daily environments. Projects like Ambient Dynamix [Car16] will enable us to connect to *any* of these services in a foreseeable future; on demand and without the need to install specialized apps or to configure connectivity settings in order to connect to them. Being enabled to potentially connect to hundreds of different services through the smart phone opens incredible new ways of interacting with our environment, but also introduces a variety of challenges.

One of the most crucial questions to answer is how users can discover IoT resources that are relevant in their situation without being overburdened by a vast number of choices. If one could sense all IoT services close by, finding a particular service becomes harder the higher the density of IoT services in the environment and hence the higher the number of possible services becomes. In order to make the number of choices manageable from a users perspective, an intelligent way to decide what services are relevant is required.

To address this challenge previous studies have shown contextual bandit algorithms such as LinUCB to be suited for context sensitive IoT service discovery. However the evaluation of these algorithms is challenging itself, given that most environments in our daily lives have not yet reached the IoT service density these methods are designed to address. Real world datasets do therefore insufficiently reflect the projected situation with 200 billion IoT services by 2020 [IIOT16]. This was our main motivation to develop a simulation that can emulate environments with an IoT service density high enough to test recommendation algorithms under anticipated future conditions. Hence the first part of this thesis is dedicated to our ambient space simulation.

With this tool in hand we aim to extend an existing recommendation approach in the second part. Instead of recommending single IoT services, we present a method to recommend multiple services in order to use them for a joint task. Finding multiple services for a mutual task is more challenging than simply selecting some services that are suitable for a certain user, as they also need to be interoperable in some sense. To address this issue we present a framework called ConComM, whose purpose is the identification of groups of services that can be used for such a combined task. A major part of the following work is spent on ConComM's design, development and evaluation as a whole and its parts. Equipped with additional information about interoperable services provided through ConComM, we test the performance of an existing IoT service recommendation algorithm for composite service recommendation. The second part's evaluation is mostly based on data provided by our previously developed ambient space simulation.

In the following work we explore new ways to evaluate recommendation algorithms for IoT service discovery. Also we extend a method for single service recommendation to make it applicable for composite service recommendation. A brief overview over the chapter structure is given below. The sections 2.2 and 2.3 in *Background and Related Work* form the **propaedeuticum**, which is also indicated by * after the section headings.

Chapter Overview

Chapter 2 – Background and Related Work gives necessary background information on IoT and context awareness and introduces contextual recommendation systems with a focus on bandit approaches. It also provides the required information on graph cuts and classification, which is needed for ConComM.

Chapter 3 – Towards an Ambient Space Simulation describes development and features of the ambient space simulation. It also gives results of a recommendation system evaluation based on a simulated environment.

Chapter 4 – The ConComM Context Composition Machinery addresses the challenging recommendation of composite services. Along with the evaluation chapter it forms the core of this work. In a first section we formulate the challenges and introduces the general idea, which is deeply discussed in the subsequent theoretical development and implementation of ConComM.

Chapter 5 – Evaluation analyses the priorly constructed ConComM framework from an experimental point of view. Before ConComM is evaluated as a whole and in its parts, we discuss input data collection with the ambient space simulation and the evaluation of quality measures.

Chapter 6 – Conclusion unites approach summary and contribution, and considers potential future work.

Thesis Context

The work presented within this thesis was conducted at the National University of Singapore (NUS), where the main author worked in the Ubiquitous Computing Group from April to October 2016. During this time, the group was lead by Dr. Darren Carlson and is part of the Felicitous Computing Institute. Background for this work is a research project started by Nirandika Wanigasekara, who is PhD student in Dr. Carlson's group. She investigated the performance of bandit algorithms for IoT service discovery and developed a modification of the standard LinUCB approach that is particularly suitable in this context.

Her preliminary results in this field are explained in 2.2.4. Along with our evaluation results obtained through the ambient space simulation presented in this thesis, our work was submitted to and accepted by the IoT Conference 2016 in Stuttgart [WSCR16]. A second paper, which comprised our work on Composite Service Recommendation and more precisely the Context Composition Machinery ConComM is currently (November 2016) under submission for PERCOM 2017.

2 Background and Related Work

Theory provides the maps that turn an uncoordinated set of experiments or computer simulations into a cumulative exploration.

(David E. Goldberg)

"Our IoT world is growing at a breathtaking pace, from 2 billion objects in 2006 to a projected 200 billion by 2020. That will be around 26 smart objects for every human being on Earth!"

This statement can be found on Intel's Internet of Things web page in 2016 [IIOT16]. It clearly shows that by now IoT enters not only the big companies but also the private sector.

The IoT vision is a world where everything is connected. This unlocks a whole new generation of services affecting buildings, transportation, manufacturing, energy management, working environments and health care. However, there is still a long way to go in order to use the full IoT potential, as the current Internet of Things is rather an Internet of Thing due to platform fragmentation and protocol barriers. New projects such as Ambient Dynamix [Car16] work on bridging the fragments in order to create a true Internet of Things. But even without the technical limitations, the identification of relevant services among 200 billion options is a challenge on its own. Therefore service recommendation could be the key to utilize the potential of future IoT environments.

Within this work we will use the term **IoT service** instead of the more commonly used term IoT device. This is due to the fact that we do not only cover real world devices with the presented methods, but also conventional web services that can be composed with the services offered by IoT devices. We explain this matter more deeply in 2.1.2 when we cover the Web of Things (WoT) view on IoT.

This chapter is split into three main parts. The first section provides background information on context, IoT and Ambient Dynamix and forms the background of our work. The two next sections form the thesis' propaedeuticum and present related work on contextual recommendation systems and methods required for the ConComM construction, covering graph cuts and classification. Lastly we give a short introduction to quality measures for graph cuts and classification.

2.1 The vision of a connected world

Back in 1991 Weiser [Wei91] introduced what is today known as the **Ubiquitous Computing Paradigm**. It foresees that computers will become a background technology in our every day lives. What was called the 'third wave of computing' back then is reality today. After the first and second wave, namely many persons sharing a computer followed by a one person one personal computer use, the third wave with more than one device per person is reality for developed countries.

Today the **Internet of Things (IoT)** is the next big paradigm, which "envisions an era where billions of sensors are connected to the Internet" [PZCG14]. As this results in a tremendous amount of data, identifying information that is relevant in specific contexts or situations becomes increasingly important.

2.1.1 Context and Context Awareness

When computers moved away from being mere desktop applications, users started to use them in different contexts for the first time. It soon became clear that an applications behaviour could profit from context information. The most common association with the word "context" is "location". However in 1999 Schmidt, Beigl, and Gellersen [SBG99] showed that considering a broader range of context information helps to enhance the interface between users and mobile devices. Examples for contextual information are location, acceleration, elevation, temperature, humidity, mood, activity or time. In 2001 Dey, Abowd, and Salber [DAS01] were the first to publish a guide to developing context aware applications. They did not only provide the first conceptual model for context aware computing. With the Context Toolkit they also offered a tool to practically develop context aware applications.

The problem with previous work was the ambiguous definition of context as a term itself. One of their main contributions is to scientifically analyse the different types of context and to develop a definition that is widely accepted in this area:

Definition 2.1.1 (Context)
Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [ADB+99].

Having defined the concept of context, one can define what context awareness means.

Definition 2.1.2 (Context Awareness)
A system is context aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task [ADB+99].

Context awareness is nowadays part of many applications. Google maps can be to find restaurants nearby when given access to the location of the querying device and smartphones are able to adapt the screens brightness to the intensity of the surrounding light.

With the emerging Internet of Things context awareness becomes increasingly important to many real world applications as they can gain access to sensor data to an extent that was hard to imagine a decade back. The next subsection will therefore give a brief introduction to IoT and one view on it that is most relevant for contextual service discovery.

2.1.2 IoT and WoT

The vision of IoT is an environment that contains billions of sensors, that are all connected to the world wide web. Processing all the collected sensor data in real time will most likely either exceed or at least present a major challenge to our nowadays computing capabilities. Hence it will be crucial to identify the data that needs to be processed - either real time, or on demand. Perera et al. [PZCG14] claim that context awareness will play a critical role in order achieve this.

There exist many definitions of what IoT actually is and they mainly differ in the area on which they focus. To give at least one definition of IoT we use the definition promoted by Strategy and Unit [SU05].

Definition 2.1.3 (Internet of Things)
The Internet of Things allows people and things to be connected anytime, anyplace, with anything and anyone, ideally using any path/network and any service [SU05]

IoT is the vision of a smart environment that makes our daily lives easier by knowing what we prefer, want and need and that reacts by itself based on this information [PZCG14]. It is more than the the vision of a coffee-machine that tells your car to heat up the wind shield when you make your morning coffee in winter, so the wind shield is not crusted by ice when you want to leave. The grand vision is also about personalized healthcare systems that provides long term monitoring and help when you really need it as well as services that are seamlessly integrated in our lives [DMD+10].

Knowing the grand vision, reality shows that there is still a long way to go. Today's IoT can better be summarized as every day objects with an IP address and internet connection [ZGC11]. As most of them do not use internet protocol standards, the sensor data remains inaccessible for conventional web services. To address this problem, Zeng, Guo, and Cheng [ZGC11] postulated a view on IoT which they called **Web of Things (WoT)**. WoT focuses its attention on the part of IoT that describes the connection to and interaction with the web. It depicts a world where objects speak the same language and can communicate or interoperate freely through the web. Hence the web not only offers conventional web services but the palette is extended to services offered by smart things.

This brings the new notion of a smart **service**, which not exclusively describes smart devices as in IoT, but aggregates web services and services offered by smart devices as they seamlessly integrate with the web. Controlling smart objects would not only be possible through specialized apps or web based applications as it tends to be the case nowadays, but simply through the browser.

As smart devices integrate to the existing web, a whole new range of possibilities to compose web services unfolds. A first example of what such a composed web service that includes web and real world services would look like is the Geiger Counter World Map [GM16]. This project provides a real time radiation map, which combines a maps map with radiation data measured by sensors all around the world. In real time users can view the radiation measured by different sensors and request historic activity data for sensors of interest.

Because WoT would allow to freely compose services just like web applications, the possibilities to compose devices are endless. By now, to obtain a health screening system that measures heart beat, EKG and also includes a GPS service, its components have to be designed and implemented to work together by an expert. WoT would allow to take some heartbeat sensor, some sensor to monitor the EKG and some GSP service and compose them just like a web application, which requires no expert knowledge about the functionality of its components any longer. Additionally it could easily be coupled with a map to show the GPS services location as this is only a further service added to the composition.

These compositions are more commonly known as *Mashups* and are the main feature of WoT. As services are available on the web, this would also allow to search for services offered by real world objects just like searching for web sites. In a nutshell, the vision WoT is that one can discover, compose and execute a collection of web services that are not exclusively conventional ones [ZGC11] .

In their conceptual paper Zeng, Guo, and Cheng [ZGC11] also discuss the challenges and unsolved problems that prevent the WoT vision to turn into reality instantly. One of the two key aspects is the challenging integration of real world services in the web, given their heterogeneity. WoT as they present it, standardizes the communication channel at the application layer. However, this is without effect if the lower layers, which are the real world IoT devices, do not support it. As IoT devices use many different protocols which fragments the IoT landscape, there exists no homogeneous solution. The second big challenge concerns search and discovery. Searching for IoT services is even more complicated than searching for classical web content as a services state can change rapidly given its rich context. Querying for sensors that currently measure 35°C are likely to produce a different output for two search requests within 5 minutes. These quick changes due to the contextual information are not addressed by standard web search engines. The question how people find the services they are looking for is not answered by the WoT concept.

The WoT vision to discover, compose and execute services has motivated further research in that field. The next subsection will present an approach, that addresses the two key challenges of WoT that are related to service discovery and offers a possible solution to them.

2.1.3 Ambient Dynamix and Ambient Ocean

IoT objects are often controllable through apps and most manufacturers offer apps that are specifically designed to control their services. Using these apps one quickly ends up with a bunch of installed apps that either do not support service interactions that leave the manufacturers ecosystem or setting up these interactions requires a huge amount of configuration. Apart from the issue of specific smart devices being controlled trough specific and potentially not interoperable apps, there is a second problem. As there are no application layer protocol standards, the heterogeneous practices by different manufacturers split the IoT landscape in a number of “walled gardens” [WSCR16]. In addition to that, many real world devices do not even use web protocols and can therefore not be integrated into WoT [CAS13].

To address these limitations, Carlson, Altakrouri, and Schrader [CAS13] proposed a framework that is called **Ambient Dynamix** or just **Dynamix**. In contrast to approaches that were proposed previously, Ambient Dynamix does not try to connect smart devices with the web by installing smart gateways to each device, as they would have to be physically installed and configured for each target network. Dynamix runs on mobile devices like smartphones, what turns them into smart gateways and enables other smart services without their own gateway to enter the web. This also allows web applications to interact with non-web services from the browser.

To enable a connected service to talk to Dynamix, it needs to implement a simple application programming interface (API). Hence Dynamix does not require a smart gateway to be installed on each and every service, the implementation of a plug-in for each service-type is sufficient. If an interaction with a service is required, Ambient Dynamix automatically discovers and downloads the required plugins to enable the communication. One of the big advantages of this approach and a strength of Dynamix is that plugins can be loaded and installed on demand. Hence plug-ins are only loaded when they are required and can be exchanged if the communication with other services is desired. A similar behaviour holds for protocols. Dynamix is able to detect a services protocol and install the required protocol gateway only if needed.

As Ambient Dynamix enables services to communicate to the web through it, is is situated between the local hardware and the Dynamix app layer. Android and web apps are supported by the framework. Hence it bridges the gap between the web and services with no or proprietary web protocol support and enables those services to become a part of WoT. With Ambient Dynamix, the first hurdle on the way to WoT service discovery is taken. However, the question how users find services has not been addressed yet.

Carlson and Schrader [CS14] proposed **Ambient Ocean (Ocean)** which addresses the challenges of making smart services searchable. The last subsection already suggested that smart resource discovery differs from conventional web search as it is highly non-static. Smart services change their context quite often as they sense non-static environments and can be switched off unexpectedly. The cloud based service Ambient Ocean therefore collects information about interactions with IoT services that were performed by Dynamix users [WSCR16].

When an interaction is reported to Ocean, the interaction is provided along with the context information relevant for this interaction. This context information is called *Context snapshot*. Therefore Ocean can be queried with arbitrary contextual data and is able to return any service that was ever used for some interaction with Ambient Dynamix before. The mechanism of Ocean “memorizing” each service that was used with Ambient Dynamix by any user, is called *Community Based Query Expansion* [CS14].

Even though being enabled to query for services is an improvement, the above described procedure still has shortcomings. When a user queries for a smart display device of a certain brand that is currently switched on, Ocean would return a list with any smart display device of that brand that was ever used by any Ambient Dynamix user. Probably the query was made in order to find a display that the querying user needed to show some slides. It is easy to agree on the fact that not every service is equally relevant for this person, given that a display more than 1000km away from the persons location will not be of much use to show a few slides to a colleague. The relevancy of a service depends on the context of the querying party. Selecting those results that are most suitable for a person is a recommendation task. To allow personalized service discovery Ocean provides integration hooks for *Recommendation Engines* [CS14] that offer support for different recommendation systems. In this work we simulate and improve a recommendation system that can be integrated into Ambient Dynamix.

Summing up the main facts in this subsection, the vision of WoT about discovering, composing and executing services became significantly substantiated by the possibilities offered through Ambient Dynamix and Ambient Ocean. Services can be connected to the WoT even without direct web protocol support and Ocean allows to query for services. However, these services might not yet be the most relevant, given the context of the query. This motivates the need for personalized and contextual service recommendations, that can be embedded in a system such as Ambient Dynamix. The next section will illustrate this need further and introduce existing work regarding recommendation systems.

2.2 Contextual Recommendation Systems*

When one queries for a certain service, there is usually an intention behind the search. Given the vast amount of services that suddenly become searchable through Dynamix, it becomes increasingly important to rank the search outcomes according to their suspected relevancy for the querier. Assuming the query produces google-like search results, a request for a service that can display a certain type of file would output a long list of services. One would potentially find a huge fraction of all smart monitors, displays and TV's in the world, as they are all able to display that type of file and could be accessed through the web. Given that in 2020 a total number of 200 billion smart services is projected [IIOT16], we can not expect users to be able to go through that list of services and being able to pick the one service they are looking for.

Therefore we require highly personalized recommendation systems that can select the services that are most suited for the querier given its current context. When one searches for a display service for a small presentation, services that are located far away from a user are less likely to be useful in his† situation. Or considering a researcher who always works with data in a special format, those recommendations should mainly contain sensors that provide this data format. This shows that contextual personalized recommendations can be the key to unlocking the full potential that comes with WoT rather than just being a vast amount of services that are searchable only for its own sake.

*This section is part of the propaedeuticum.

†Users can explicitly be male and female. As the B.Sc. students in the main autors field of studies are male in 75% of the cases, we will address users as male individuals in this thesis.

2.2.1 From Collaborative Filtering to Multi Armed Bandits

A recommender system always tries to estimate some kind of rating function R : User $\times$ Item $\rightarrow$ Rating that represents how a user would rate an item [AT11]. One of the oldest approaches to recommendation is **Collaborative Filtering** [RRS11]. In its most simple version, collaborative filtering assumes that rating functions of similar users are also similar. Therefore it recommends such items to users that were rated high by other users with a similar profile. Lemire and Maclachlan [LM] present an approach of collaborative filtering that can be updated on the fly and is efficient at query time.

Collaborative filtering works well for "main stream user" while its performance drops for users with more unique preferences. Also it suffers from the so called **Cold Start Problem**, which means recommendations for users of which no past preferences are known or on items that have not been used by any user are poor.

A well studied problem that resembles the recommendation challenge is the **Multi Armed Bandit Problem** that was among others analysed by Robbins [Rob85] or Auer, Cesa-Bianchi, and Fischer [ACF02]. The problem describes a bandit and many gambling machines (arms). In the case of a k-armed bandit there exist exactly k gambling machines. In each time step t, the bandit can play a machine and wins an amount of money that depends on the machine played. This money is called reward r. After a certain number of time steps, the rewards the bandit collected are summed up. This accumulated reward is compared to the reward the bandit *could* have scored if he had always played the optimal machine. The difference between this optimal value and the accumulated reward is called **regret** [AB10].

Given that the bandit does not know which machine is optimal at any time step, the multi armed bandit problem asks which strategy the bandit should adopt in order to minimize regret to maximize his reward. Applied to service recommendation, every possible service can be seen as an arm and for each query the bandit has to select a service. If the service is suitable for a user, the user will select the service and connect to it. This can be considered as a reward of 1. If the user does not like the recommendation, the service is not selected which equals to a reward of 0. Maximizing rewards in this setting equals to maximizing the users satisfaction with the recommendations where each request for a service is a time step.

The key to the multi armed bandit problem is to find a good balance between **exploration** and **exploitation**. If the bandit keeps trying different machines, it can not receive the optimal reward because it often plays suboptimal machines. On the other hand, if the bandit sticks to one machine, it can not know if there is a machine that would probably yield higher rewards. To balance exploration (trying new machines) and exploitation (collecting reward from machines that have proven to give high rewards),

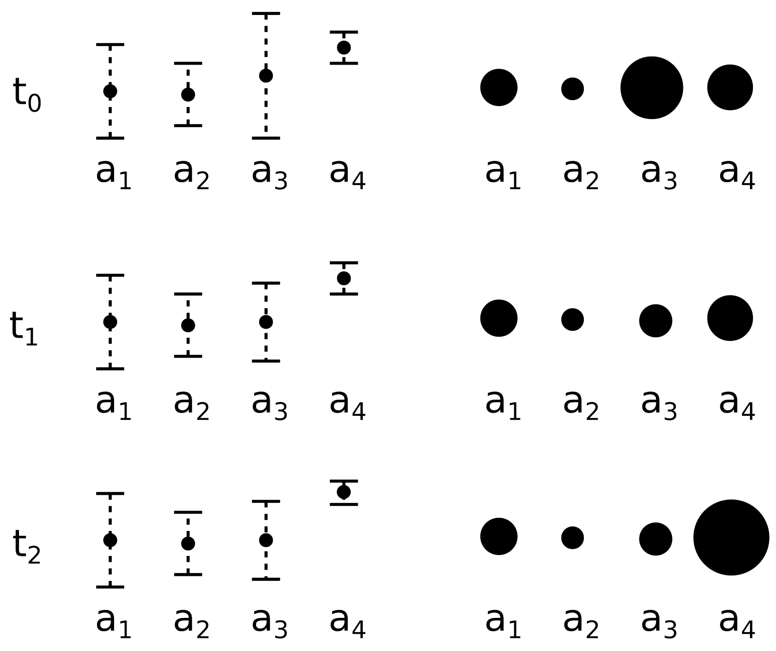


Figure 2.1: The UCB and EXP Principles

The working principles for UCB (left) and EXP (right) approaches. Given the arms a_1, a_2, a_3, a_4, for each arm UCB has a current reward estimate (black dot) and an uncertainty estimate (dotted line). The exponential weights approach assigns a weight to each arm that is represented through the size of the black circle. From top to bottom, the figure illustrates 3 time steps. The first time step t_0 shows an arbitrary initial setup. In t_1, for both approaches arm a_3 was pulled as it has the highest UCB and is most likely to be pulled for EXP. It did not receive a good reward. Therefore the reward estimate for UCB is lowered and the uncertainty estimate shrinks. For EXP, the weight of a_3 is decreased. The third row t_2 illustrates pulling arm a_4 as next arm. Assuming a_4 returned a positive reward, the reward estimate for LinUCB is increased and the uncertainty lowered. For EXP, the weight for a_4 is increased.

two approaches have proven themselves very useful. The first are are **Upper Confidence Bounds (UCB)** and the second are **Exponential Weights (EXP)** [Rey13].

The idea behind UCB is to calculate two values for every arm. The fist represents what average reward was scored from the arm so far. The second reflects how sure one can be that this average score is accurate. Bringing these values together, the upper confidence bound of each arm is the current reward estimate plus an uncertainty estimate [MRTM12]. In each time step, the arm with the highest UCB is played. If an arm was not played very often, the uncertainty about this arm will be very high - even though the current reward estimate is probably not too high. However, an arm that was played multiple times will not have a high uncertainty, but the reward estimate will be very correct. This ensures that rarely played arms with a high uncertainty are played (exploration) but knowledge about good arms with and high reward estimate is also used (exploitation).

Exponential weights are used in algorithms such as Exp3 [ACFS02] and adopt a strategy that relies on assigning weights to each arm. In each time step, an arm is picked randomly. However the probability to play a certain arm is proportional to the weight assigned to the arm. Arms with high weights get played more often. If picking the arm gives a reward, the weight of the arm is increased. Should the arm return no reward its weight decreases. As this increase or decrease is often realized by doubling or taking half of the original weight, this approach is called exponential weights [Rey13]. Figure 2.1 illustrates the working principles for both UCB and EXP.

Such strategies also avoid the cold start problem. In the case of UCB algorithms, a new service has a high uncertainty and would therefore be played at least a few times. This is sufficient to get an estimate about its reward - hence no information about how other users rated or clicked the service is required to incorporate a new service in the recommendation process. However, the approaches presented so far do not use any contextual knowledge. As the context plays an important role for IoT service recommendation, the next section discusses how bandits with context are defined and presents a simple contextual bandit algorithm.

2.2.2 Bandits and Context: A first approach

Generally, a context aware recommender system can be seen as an extension of the conventional recommender system that tries to estimate a rating function that is not only depending on User and Item but also on the context. Therefore the rating function is now R : User $\times$ Item $\times$ Context $\rightarrow$ Rating [AT11].

Until now, a bandit problem was described by arms that return a certain reward. This reward follows a probability distribution P over the arms. For a contextual bandit, the reward does not only depend on the arm but also on the context.

Definition 2.2.1 (Contextual Bandit Problem)
In a contextual bandits problem, there is a distribution P over $(x, r_1, \ldots, r_k)$, where x is context, $a \in \{1, \ldots, k\}$ is one of the k arms to be pulled, and $r_a \in [0, 1]$ is the reward for arm a. The problem is a repeated game: on each round, a sample $(x, r_1, \ldots, r_k)$ is drawn from P, the context x is announced, and then precisely one arm a chosen by the player, its reward r_a is revealed [LZ08].

As an example, take a recommendation system that is supposed to recommend a display device. A user will prefer different displays, which are the arms, when being at home and being at work. So the location as part of the context information plays a role, as the reward for displays is not distributed independently of the users location. Having a definition for a contextual bandit problem, a contextual bandit algorithm can be defined as below.

Definition 2.2.2 (Contextual Bandit Algorithm)
A contextual bandits algorithm determines an arm $a \in \{1, \dots, k\}$ to pull at each time step t, based on the previous observation sequence $(x_1, a_1, r_{a,1}), \dots, (x_{t-1}, a_{t-1}, r_{a,t-1})$, and the current context x_t [LZ08].

Langford and Zhang [LZ08] were the first to propose the contextual bandit problem along with a contextual bandit algorithm which they call **Epoch-Greedy**. The goal for any bandit algorithm is to maximize the accumulated reward R, or to maximize the expectation value of the rewards over all time steps. In the bandit setting, there exist several so called *Hypotheses* $h \in \mathcal{H}$ that map a context $x \in \mathcal{X}$ to a specific arm $h : \mathcal{X} \rightarrow \{1, \dots, k\}$. A hypothesis h basically tells the bandit which arm to choose when given a context. The bandits has to compete with the best hypothesis, even though it does not know which hypothesis is the best.

The epoch greedy procedure [LZ08] is organized in epochs $l \in \mathbb{N}$. Each epoch begins with one exploration step. In the exploration step, one arm is chosen *uniformly at random* and the reward is observed. The tuple of current context, chosen arm and reward $(x_t, a_t, r_{a,t})$ is added to a set of observations $\mathcal{W}$. Afterwards, the best hypothesis $\hat{h}_l \in \mathcal{H}$ for the current epoch is selected by evaluating $\max_{h \in \mathcal{H}} \sum_{(x,a,r_a) \in \mathcal{W}} r_a I(h(x) = a)$. For each hypothesis, this expression sums up the rewards the bandit would have received if it had played with this hypothesis. $I(h(x) = a)$ is an indicator function that returns 1 if the hypothesis would have taken arm a when presented context x. If this is not the case, the indicator is 0. After the exploration step, a predefined number of exploitation steps follows. Here the bandit selects arms according to the best hypothesis in this epoch $\hat{h}_l$ and receives the rewards.

The reason why epoch-greedy draws the arm during the exploration uniformly at random and also only evaluates the hypothesis on arms drawn during the exploration is the following: For each time step, only one reward is observed. In order to evaluate the best hypothesis on an unbiased set of pulled arms, the arms have to be drawn uniformly at random. Else the sample would be biased by some hypothesis that then receives more rewards as it was used to predict the arm.

Epoch greedy is a very simple contextual bandit as it has a predefined number of exploration and exploitation steps per epoch. However, UCB and EXP approaches were designed to balance these two influences themselves. After having seen how the concept of context can be integrated into the bandit approach, the next section will present contextual Upper Confidence Bound algorithms as they balance exploitation and exploration on their own.

2.2.3 The Contextual UCB Family

A family of algorithms that balances the number of exploration and exploitation steps more independently are upper confidence bound algorithms. Contextual UCB algorithms have received a notable attention by the scientific community and are were often trained for online recommendation tasks. This makes them a very good starting point when it comes to IoT service recommendation.

The first contextual upper confidence bound algorithm was proposed by Li et al. [LCLS10]. Their LinUCB is able to quickly adapt to changing pools of content and contexts, as it was developed for personalized news article recommendation. Therefore it must quickly recognize breaking news as especially relevant, but also detect the decreasing relevancy of older articles. A scenario like this is very similar to IoT service recommendation as the pool of services is constantly changing and the relevancy of services changes according to their current state. If a service is in use by another user, the recommendation system should be able to promptly recognize this and stop recommending the service if it is only usable by a single user.

The main idea behind LinUCB is to assume that the reward for each arm a can be modelled using linear regression on features of the context x. In other word, it assumes that for each arm there exists a function that returns the correct reward for this arm when presented the context features x. Linear regression is used to find this function, which has the form $x^T\theta$. This is the scalar product of x with the weights θ, which gives a function that is linear in its features. Hence for each arm a, the expected reward can be modelled by

$$E[r_{t,a}|x_{t,a}] = x_{t,a}^T\theta_a^* \tag{2.1}$$

with $x_{t,a}$ being the feature vector of context x for time step t and arm a. Here every arm gets an own function and no features are shared with other arm. Therefore this model is called LinUCB disjoint. If all arms use the same features, the a in the subscript of x can be dropped. θ_a^* is the unknown optimal coefficient vector for the arm.

In order to obtain the optimal coefficients for each arm a given some training data that consists of tuple $(x_{t,a}, c_{t,a})_{t=1}^s$, one wants to minimize the following loss function:

$$L_a = (\sum_s^t c_s - \sum_s^t x_{s,a_s}^T\theta_a)^2 + \|\theta_a\|^2. \tag{2.2}$$

This function returns the squared difference of the accumulated actual rewards and the rewards the arms function would predict. If the difference is 0, the arms function does completely fit with the actual rewards. The term $||\theta_a||^2$ is used for regularization to avoid overfitting. To find the optimal vector θ_a that minimizes L_a the loss is derived by θ_a, set to zero and solved for θ_a. Deriving the loss function becomes much easier if the sums are rewritten in terms of matrices norms. Therefore we define

$$D_a = \begin{pmatrix} x_{1,a_1}^T \\ \vdots \\ x_{t,a_t}^T \end{pmatrix} \quad \text{and} \quad c_a = \begin{pmatrix} c_{a,1} \\ \vdots \\ c_{a,t} \end{pmatrix}.$$

D_a and c are simply the stacked versions of the feature vectors and their corresponding rewards. Now equation 2.2 can be expressed as

$$L_a = ||c - D_a\theta_a||^2 + ||\theta_a||^2 \tag{2.3}$$

with $|| \cdot ||$ being the euclidean norm. Deriving this term by θ_a, setting it to zero and solving it for θ_a yields the optimal $\hat{\theta}_a$

$$\hat{\theta}_a = (D_a^T D_a + I_d)^{-1} D_a^T c_a = A_a^{-1} b_a \quad \text{with} \quad A_a = D_a^T D_a + I_d \quad \text{and} \quad b_a = D_a^T c_a,$$

where I_d is the d-dimensional identity matrix. Recalling what we wanted the optimal $\hat{\theta}_a$ for, now one can compute the expected reward for each arm. As UCB approaches require and estimate over the reward per arm and an estimate about the uncertainty per arm, the only thing that is still missing is a model for the uncertainty. For LinUCB, the uncertainty for an arm a is modelled as $\alpha\sqrt{x_{t,a}^T A_a^{-1} x_{t,a}}$ with A_a being the inverse of the first factor in the expression for $\hat{\theta}_a$, $A_a = (D_a^T D_a + I_d)$. Now the upper confidence bound for each arm can be written as sum of expected reward and uncertainty, which gives a rule how to choose the optimal arm a_t for every time step t:

$$a_t = \arg\max_{a \in \mathcal{A}_t} \Big(\underbrace{x_{t,a}^T \hat{\theta}_a}_{ExpectedReward} + \underbrace{\alpha\sqrt{x_{t,a}^T A_a^{-1} x_{t,a}}}_{EstimatedUncertainty} \Big).$$

The model parameter α is used to control the balance of exploration and exploitation during the learning phase. A high α encourages exploration while a small α is suppressing it. If α is set to 0, basically no exploration happens, which is equal to choosing only one arm for a context without exploring other options. Now, the whole algorithm for LinUCB disjoint as presented in [LCLS10] can be formulated. The procedure is shown in algorithm 2.1.

The procedure is the following. In each time step the context is converted in the right feature vector for each arm. As a next step, the optimal coefficients $\hat{\theta}_a$ are computed, if the arm did not exist before it is initialized. With the optimal coefficient, the UCB $p_{t,a}$ is

Algorithmus 2.1 LinUCB Disjoint

```
procedure LinUCB Disjoint($\alpha$)
    for $t = 1, 2, \dots, T$ do
        observe features $x_{t,a}$ for all arms $a \in \mathcal{A}_t$
        for all $a \in \mathcal{A}_t$ do
            if $a$ is new then
                $A_a \leftarrow I_d$
                $b_a \leftarrow \mathbb{0}_{d \times 1}$
            end if
            $\hat{\theta}_a \leftarrow A_a^{-1} b_a$
            $p_{t,a} \leftarrow x_{t,a}^T \hat{\theta}_a + \alpha \sqrt{x_{t,a}^T A_a^{-1} x_{t,a}}$
        end for
        $a_t = \arg\max_{a \in \mathcal{A}_t} p_{t,a}$
        $A_{a_t} \leftarrow A_{a_t} + x_{t,a_t} x_{t,a_t}^T$
        $b_{a_t} \leftarrow r_t x_{t,a_t}$
    end for
end procedure
```

calculated for the arms in line 10. In a next step in line 12, for the current time step t the optimal arm a_t is chosen, which is the arm with the highest UCB. Afterwards, for this (and only for this) arm the matrices and vectors required to compute the optimal $\hat{\theta}_{a_t}$ are updated so that θ_{a_t} gets updated during the next timestep. To see why the updates in line 13 and 14 take this form, x_{t,a_t}^T and r_t can be plugged in the formulations for A_{a_t} and b_{a_t} as D_a and c_a, as this is are the new data points for time step t.

Li et al. [LCLS10] also suggest a second version of LinUCB which they call LinUCB hybrid. The expected reward for LinUCB Hybrid does not only consider arm specific features but also features that are shared among arms. This allows to model influences that affect all arms instead of each arm separately. For example, these shared features could be used to model the time dependency that affects each each arm (=service or news article) in the same way. Hence, not every arm has to learn on its own that its relevancy fades over time. It can profit from the coefficients representing this behaviour that have already been trained by the other arms. The expected reward for an arm becomes

$$E[r_{t,a}|x_{t,a}] = z_{t,a}^T \beta^* + x_{t,a}^T \theta_a^* \tag{2.4}$$

which adds the shared coefficient vector β^* and the shared features $z_{t,a}$ to 2.1. The general strategy of LinUCB is sufficiently illustrated by the rather detailed explanation and derivation of LinUCB hybrid above. However, [LCLS10] does not give much detail on how the equations for the hybrid model are obtained and only formulates the basic idea with 2.4 and the full algorithm. As we derived the equations required for the hybrid approach, which are given but not derived in the paper, they can be found in appendix A.1.

Having seen an example for contextual bandits in detail, the remainder of this section will give a more general overview of other contextual UCB algorithms.

An algorithm that adapts the general LinUCB procedure is **LinPRUCB** [CLCL14]. The idea is to achieve a quicker convergence of the coefficient vectors toward the real optimum by estimating the reward for non-played arms. LinUCB only updates the coefficients for the played arm as it lacks feedback for the non-played ones. LinPRUCB utilizes a complex reward estimation to compute pseudo-rewards that are fed to the learner. Hence LinPRUCB is short for LinUCB with pseudo-rewards.

However, LinPRUCB is still quite close to LinUCB with regard to the main idea. Other alternatives are **LogUCB** [MRTM12], which uses logistic regression instead of linear regression to represent the reward distribution of a selected arm, and **NeuralBandit1** by Allesiardo, Féraud, and Bouneffouf [AFB14], which utilizes a neural network for this task.

Having seen that LinUCB is by far not the only contextual bandit, it should be noted that there are multifarious possibilities to specialize and boost bandit algorithms. An idea that was formulated by Tracà and Rudin [TR15] is based on the observation that many real life patterns change periodically. Therefore, exploration activities could be focused on the beginning of those periods. The work of Bouneffouf, Bouzeghoub, and Gançarski [BBG12] focuses on the aspect of context evolution, as change in a contexts' meaning is neglected by most recommendation systems. Especially the highly specialized versions of UCB algorithms can be very costly in terms of computation and time. Yue, Hong, and Guestrin [YHG12] propose an acceleration strategy for contextual bandits that uses a hierarchical coarse to fine approach.

Starting from simple approaches like collaborative filtering, this section covered the key aspects of recommendation that are relevant for IoT service recommendation. As it describes the IoT service recommendation scenario very well, it explained the general multi-armed bandit problem and its extension to the contextual multi armed bandit. LinUCB was introduced as the first and most famous contextual upper confidence bound algorithm. Even though there exist many specialized versions, LinUCB still offers a good trade off between computational complexity and recommendation accuracy. Therefore LinUCB is taken starting point to explore the integration of contextual bandit algorithms in the WoT query system Ocean. To conclude the motivation of the work done in this thesis, the next section will give the preliminary results of integrating a LinUCB modification into Ambient Ocean.

2.2.4 Bandit Algorithms for the WoT Vision

Wanigasekara et al. [WSCR16] proposed an UCB algorithm that can be used IoT service recommendation with the Ocean Search Engine. The algorithms is called **LinUCB Partial** and has a better performance than the standard LinUCB for IoT service recommendation. Inspired by LinPRUCB [CLCL14] (see section 2.2.3) it adopts the idea of receiving more than the reward for only one arm at a time. LinPRUCB used a dedicated pseudo-reward retrieving strategy to estimate the reward of *all* unplayed arm based on the played arms reward. LinUCB partial does not assume full information as a user that rates all suggested services seems very unlikely. However, it assumes to receive *more than one* reward at a time, which means that rewards for a few unplayed arms are received as well.

Method and Procedure: Having explained the algorithmic idea, LinUCB partial was preliminarily tested in two experiments. The first usability test is based on the following scenario: A user wants to be notified whenever he or she receives a new twitter mention. Because smart phones tend to be in pockets, jackets or bags, the notification is supposed to happen via a flashing smart light bulb that is closest to the user. To perform this experiment, a room was equipped with a number of smart lamps and 10 people were asked move through the room. As they wandered around and interacted with the lamps, the algorithm learnt the closest light bulbs for the different user positions. It was able to predict the closest light even for positions, the user had not been in before.

The second experiment was performed based on data gathered during smart kitchen usability testing. The data consists of values like movement (does the user sit, stand or walk), high level activity (Relax, Coffee, Clean up), movement of left and right arm, the object in right and left arm and the mutual movement of both arms. This information was measured through accelerometers around the users arms. The LinUCB approach was employed to predict the service within the kitchen that would be most useful to a person that is performing a complex task. Such a task could be the preparation of a meal or the cleaning of surfaces. In order to receive personalized recommendations, user data was passed to the algorithm.

Results: Figure 2.2 shows the performance of different LinUCB versions for the Smart Twitter notification in comparison to the benchmark bandit approach LinGreedy. All LinUCB approaches score better results than the benchmark approach. LinUCB partial clearly outperforms the other LinUCB variants LinUCB disjoint (LinUCB-D) and LinUCB hybrid (LinUCB-H), especially during the first few number of trials (< 200). Even though not shown here, the results for the different bandit approaches on the smart kitchen data were similar.

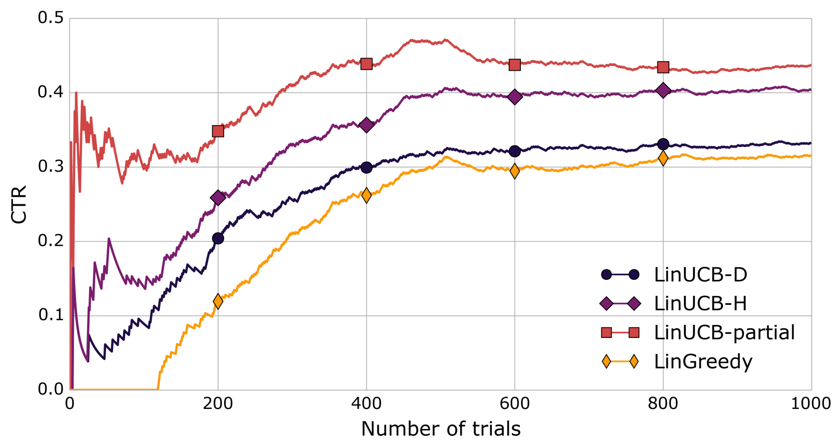


Figure 2.2: Evaluation Smart Twitter Notification

The performance of the LinUCB variants LinUCB disjoint, LinUCB hybrid and LinUCB partial in comparison to the benchmark LinGreedy. The plotted Clickthrough Rate (CTR) is the number of accepted or good recommendations over all recommendation given. All LinUCB versions outperform LinGreedy which shows that LinUCB is well suited for personalized recommendation tasks.

Discussion: For IoT service recommendation, the performance during the first few trials is very important as a user will not continue to use a system that gives poor recommendations for a long time. LinUCB partial is not only outperforming all other approaches on the long run, but also has a very good performance after only a few time steps. This makes the approach a very promising candidate for IoT service recommendation.

However, the generalizability of this evaluation suffers from a lack of usable datasets to test the approaches performance on. Especially the smart kitchen usability test seems quite artificial. When preparing a meal, the choice of using the fridge, a knife or the oven is not so much determined by personal preference, but by the recipe and the task itself. Currently it is hard to find datasets that reflect the high IoT service density that is projected for 2020, as our current density is much lower. Therefore the evaluation of algorithms that are designed to address even those densities, presents a challenge that has not been addressed so far. The first part of our work will address this issue and provide a tool to test recommendation approaches under more general conditions, which are currently hardly represented by most real world datasets.

Here, LinUCB partial is only tested for experiments where the number of services is relatively small and contextual information not very rich. But the preliminary results shown in this section already indicate a good performance of LinUCB partial for IoT service discovery. This clearly demonstrates that the WoT vision does not have to remain one, as these results are a great step towards IoT service discovery.

2.3 Graph Cuts and Classification[*]

In the last two sections we explored the background that makes IoT service discovery possible and required. Now we present algorithms that are used for graph cuts and classification. We require both tasks for our ConComM framework that enables a LinUCB approach to recommend multiple services, which we present in chapter 4.

Topics concerning graphs and graph cuts are covered within the first four subsections. Out of them, the first subsection gives an overview of the terminology that is commonly used for graphs and briefly mentions important algorithms. The next two subsections discuss important algorithms in greater detail, followed by some brief remarks about online graph clustering. The last two subsections in this section are reserved for classifiers, with an extended overview in 2.3.5. Some final remarks on online classification are given in the last subsection.

2.3.1 Graphs and Graph Cuts

The notion of a graph has been briefly mentioned in the last section, but before exploring different graph clustering techniques it seems appropriate to define our notation of a graph.

Definition 2.3.1 (Graph)
*A **graph** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an object consisting of a non empty, finite* vertex set $\mathcal{V}$ *and an* edge set $\mathcal{E} \subseteq \mathcal{V}^2$. *[Tru13]*

For our framework, it will be required to cut graphs into several subgraphs. Therefore the definition of a graph cut used by us is given below to illustrate our notation further.

Definition 2.3.2 (Graph Cut)
A cut $\mathcal{C} = (\mathcal{S}, \mathcal{T})$ is a partition of the vertex set $\mathcal{V}$ into the two subsets $\mathcal{S} \in \mathcal{E}$ and $\mathcal{T} \in \mathcal{E}$. The edges connecting the subsets form the cut-set $\{(v, w) \in \mathcal{E} \mid v \in \mathcal{S}, w \in \mathcal{T}\}$ [R15].

To cut the graph according to the above formulated requirements, we want to find partitions whose accumulated weight of edges in the cut-set is minimal. The cut that gives these partitions is called a *Min-Cut* [SW97].

There exist a number of algorithms that solve the Min-Cut problem such as the *Edmonds-Karp* [EK72] or *Stoer-Wagner* algorithm [SW97]. If only an approximate solution with a

[*]This section is part of the propaedeuticum.

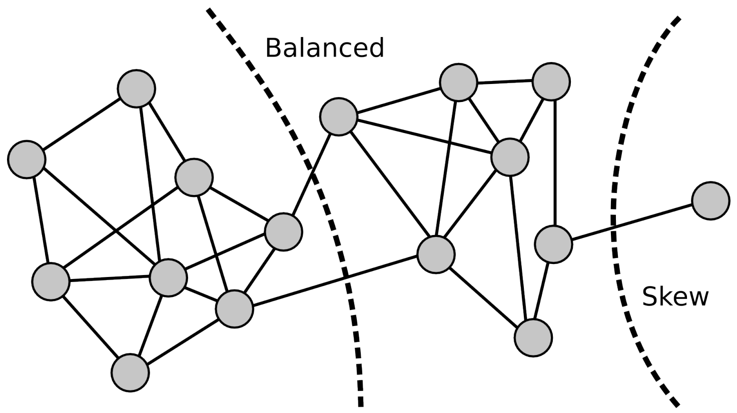


Figure 2.3: Skew and Balanced Cut

Two different cuts through the graph. The lightest cut cuts only a single node, resulting in a skew distribution of nodes. The balanced cut is not necessarily the lightest, but might represent the overall graph structure better.

better runtime is required, the randomized *Kargers algorithm* [Kar93] or its successor the *Karger-Stein* algorithm [KS96] can be used.

However, most of these algorithms tend to produce very skew cuts [DHZ+01] [CW91]. The difference between skew and balanced cuts is explained in figure 2.3. To obtain more balanced cuts a few algorithms have been proposed, namely *ratio cut* [CW91] [HK92], *normalized cut* [SM00] and *Min-Max cut* [DHZ+01]. Min-Max cut is based on the **min-max clustering principle**. The min-max clustering principle tries to maximize the similarity within each subgraph while it minimizes the similarity between subgraphs. Min-Max cut not only encourages balanced cuts, but also outperforms ratio cut and normalized cut [DHZ+01].

All the above mentioned algorithms can be used to split a graph into two subgraphs. If more subgraphs are required, a minimum *k-cut* can be used. It is defined in accordance to the two-partition graph cut, the only difference is in the number of partitions that is now k. The weight of the cut is the summed weight over the cut-set for every pair of sets.

As it turns out, finding a minimum k-cut for $k \geq 3$ is NP-hard [DJP+92]. Goldschmidt and Hochbaum [GH88] proposed an algorithm to find the minimum k-cut for a fixed k in O(n^{k^2}) with n being the number of vertices in the graph. Their algorithm is based on the idea that after finding the min-cut that separates k vertices for all combination of k out of n vertices, the optimal solution was among the cuts.

Considering the runtime of this algorithm, one might not be interested in the optimal but in a good-enough solution. In their paper Saran and Vazirani [SV95] present two simpler algorithms that find an approximate solution to the min-k-cut problem with less computational complexity. The two algorithms are called EFFICIENT and SPLIT. EFFICIENT picks for all edges e a minimum weight cut that separates e's end points. All cuts are sorted by increasing weight; afterwards, repeatedly the lightest cut is picked until their union is a k-cut. SPLIT on the other hand uses a min-cut algorithm on the whole graph and splits the graph according to the min-cuts outcome. Until the number of k subgraphs is reached, the minimal cut for each of the graphs is calculated and the graph with the smallest cut weight is split again.

Most of the above mentioned min-cut algorithms find the minimum cut for a graph rather than through a specific edge. Also, SPLIT requires only O(kn) min cut computations while EFFICIENT consumes O(e) with e being the number of edges. For a fully connected graph this is O($n!$).

2.3.2 Stoer Wagner Algorithm

The following paragraphs show the mode of function for the Stoer-Wagner graph cut algorithm [SW97]. Stoer-Wagner assumes a weighted connected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a edge-weighting function w. Also, the a subset of the graphs vertices $\mathcal{S} \subset \mathcal{V}$ is used for the algorithm. The vertex $\mathrm{s} \in \mathcal{V}$ is an arbitrary vertex that is taken as start-vertex for the algorithm. The algorithm has two methods: one of them is called `MinimumCutPhase(`$\mathcal{G}$`,w,s)` and the other is the `MinimumCut(`$\mathcal{G}$`,w,s)`. Both methods are given in appendix A.2.

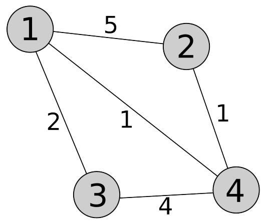
`MinimumCutPhase` takes the start vertex and begins to merge it with the most tightly connected vertex from the free set of vertices. It repeats this until only one vertex is left. A vertex $\mathrm{v} \in \mathcal{V}$ is the most tightly connected vertex if

$$\mathrm{v} \notin \mathcal{S} \text{ and } \mathrm{w}(\mathcal{S}, \mathrm{v}) = \max\{\mathrm{w}(\mathcal{S}, \mathrm{u}) \mid \mathrm{u} \notin \mathcal{S}\}.$$

Then, it determines the cut weight between the free vertex and the merged vertex and stores this value. Note, that the merging is not performed on the real graph since it is only required to find the current cut weight. As a last step, the free vertex and the vertex that was last added to $\mathcal{S}$ are merged with effect on the real graph.

`MinimumCut` on the other hand calls `MinimumCutPhase` as long as there is more than one vertex left. Because `MinimumCutPhase` merges two vertices in every run, for n vertices this gives $(n-1)$ `MinimumCutPhase`-calls. `MinimumCut` also keeps track of the lightest cut that was returned by `MinimumCutPhase` so far, which is the lightest graph cut after the graph was fully merged.

Table 2.1: Matrix Representations for a Graph

Graph Illustration	Degree Matrix	Adjacency Matrix	Laplacian Matrix
	$\begin{pmatrix} 8 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix}$	$\begin{pmatrix} 0 & 5 & 2 & 1 \\ 5 & 0 & 0 & 1 \\ 2 & 0 & 0 & 4 \\ 1 & 1 & 4 & 0 \end{pmatrix}$	$\begin{pmatrix} 8 & -5 & -2 & -1 \\ -5 & 6 & 0 & -1 \\ -2 & 0 & 6 & -4 \\ -1 & -1 & -4 & 6 \end{pmatrix}$

2.3.3 Min-Max Cut Algorithm

While Stoer-Wagner aims to find the minimum cut, Min-Max cut not only tries to find a cut that is small. It also aims to produce clusters that have maximal similarity, but minimal linkage among each other. Before presenting the ideas by Ding et al. [DHZ+01], we briefly introduce the required graph theoretical definitions to demonstrate our notation for them.

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = n$ can be represented in terms of different matrices. The first important matrix is the *Degree Matrix* $\mathrm{D} = \mathrm{diag}(d_1, \ldots, d_n)$, where d_i is the vertex degree of the i^{th} vertex [Wei16b]. A vertexes degree is defined as the number of edges that are connected to it [Wei16d].

The next required matrix is called *Adjacency Matrix* W. Like the degree matrix it is an $n \times n$ matrix with rows and columns labelled according to the graphs vertices [Wei16a]. For a weighted graph, each entry represents the edge weight between two vertices. For an undirected graph, the adjacency matrix is symmetric. A connected graph without self loops (edges of the form $e = (v, v)$ with $v \in \mathcal{V}$), has zero entries on its diagonal.

The third and last matrix that has to be introduced is the *Laplacian Matrix* $\mathrm{L} = \mathrm{D} - \mathrm{A}$ [Wei16c]. It is defined as the difference of degree and adjacency matrix and will play an important role for the Min-Max cut. Table 2.1 shows a small example graph and the corresponding degree, adjacency and laplacian matrices.

Before we formulate the objective for Min-Max cut, a last definition has to be given. Ding et al. [DHZ+01] define the cut weight between two subgraphs A and B with w_{uv} as the weight of the edge between the vertices $u, v \in \mathcal{V}$ as follows:

$$cut(\mathrm{A}, \mathrm{B}) = W(\mathrm{A}, \mathrm{B}) = \sum_{u \in \mathrm{A}, v \in \mathrm{B}} \mathrm{w}_{uv},$$

$$W(\mathrm{A}) = W(\mathrm{A}, \mathrm{A}).$$

These definitions give the weight a graph cut that is in accordance to the definition of a graph in the first subsection. This definition does not only give rule how to calculate the weight of a cut. With $W(\mathrm{A})$ exists a measure to quantify a the strength of a clusters internal connectivity. Now it is possible to give an objective function that represents the two goals of Min-Max: To maximize similarity within a cluster while minimizing the similarity between clusters,

$$\mathrm{M}_{\mathrm{cut}} = \frac{cut(\mathrm{A},\mathrm{B})}{W(\mathrm{A})} + \frac{cut(\mathrm{A},\mathrm{B})}{W(\mathrm{B})}.$$

Hence, the M_{cut} goal is minimized if the internal connections in the clusters are strong, or the cut separating them is light. In their paper, Ding et al. [DHZ+01] continuously relax this Min-Max cut function to find a solution that optimizes the function. They derive that the so called *Fielder Vector* gives a solution to the optimization problem. The Fielder vector is the eigenvector that corresponds to the second smallest eigenvalue of the graphs Laplacian matrix.

One of the Fielder vector's most important properties is that it provides a so called 'linear search order'. Its entries are in the range $[-1, 1]$. Therefore it splits the vertices that form the laplacian matrix into two groups, according to the sign of their Fielder-entry. Computing the eigenvector to the second smalles eigenvalue of the graphs laplacian gives a first clustering.

The second contribution in the paper of Ding et al. [DHZ+01] is called *linkage-based refinement*. It considers the split obtained by the Fielder order as a good first approximation, but not as perfect split. Therefore it tries to identify vertices that have a higher linkage to the cluster they are *not* currently in. If such a vertex is found, it is to test whether moving it to the other cluster would lower the cut-weight and if so, the vertex is moved. Therefore at first a measure of the similarity between the two clusters is required. This measure is called linkage l:

$$l(\mathrm{A},\mathrm{B}) = \frac{W(\mathrm{A},\mathrm{B})}{W(\mathrm{A})W(\mathrm{B})}.$$

As a last step, the linkage difference Δl for a vertex $v \in \mathcal{V}$ can be defined as

$$\Delta l(v) = l(v, \mathrm{A}) - l(v, \mathrm{B}).$$

If $\Delta l(v) > 0$ the linkage of v with A is higher, else it is stronger linked to B. The combination of a preliminary clustering that is obtained by the Fielder order with linkage based refinement leads to the full Min-Max cut procedure.

2.3.4 Remarks on Online Graph Clustering

Before proceeding with classification algorithms, we want to briefly address online graph partitioning. As suggested before, we use graph cut algorithms in our framework that allows the recommendation of more than one service for a combined task. A recommendation system for (composite) IoT services would have to adapt according to new requests during run time, without the need to recalculate the whole graph partition. Especially the need to partition huge and constantly changing social network graphs, have promoted the field of online load balancing and online partition. There exists work like [FK15] or [RPG+13] that can potentially be applied to our ConComM framework. However, our work aims to explore whether a framework like ConComM would significantly improve composite service recommendations. To parallelise the framework and prepare it for online use is an interesting topic for future work, which we discuss in 6.3.

2.3.5 Classifier Overview

After our previous discussion of graph cut algorithms, this section will give an introduction to classification algorithms. The first part covers the different types of binary classification algorithms while extensions to multi class classification are briefly discussed in the end. All algorithms we present in this section were explained in a course held by Toussaint [Tou15] or introduced in the book by Friedman, Hastie, and Tibshirani [FHT01]. This subsection mainly focuses on logistic regression and support vector machines as they are the most common classifiers. Assume labelled data $\mathcal{D} = \{(x_i, y_1)\}_{i=1}^{n}$ with feature vectors $x_i \in \mathbb{R}^d$ and class labels $y_i \in \mathbb{R}$. A data point is characterized by its feature vector and label.

We consider **Logistic regression** to be the first nameable classifier in this context. Its main idea is to find a function to describe each class. If a feature vector belongs to class represented by a certain function, its value for this vector should be higher than the value of any other function. Linear regression is used to obtain the class functions.

The following equations cover binary class classification, which means $y_i \in \mathcal{Y} = \{0, 1\}$. In the binary case, two functions $f(x, 0)$ and $f(x, 1)$ are required to describe the two classes. Depending on whether the feature vector x is in class 0 or 1, the first or respectively second function should return the greater value. Therefore, the class y for a feature vector x can be identified by evaluating

$$y = \underset{\hat{y}}{\operatorname{argmax}} f(x, \hat{y}). \tag{2.5}$$

This goal makes the formulation of the problem more simple, as we can fix one of the functions to be zero. Without loss of generality, we assume $f(x, 0) = 0$. The non-fixed function is sufficient to provide a classification, as it is supposed to be greater than zero if its own class is the correct class and smaller if its own class is not correct. The missing function is obtained by linear regression and can therefore be written in terms of the the feature vector x and some weights β as $f(x, y) = x^T\beta$. Logistic regression aims to minimize an especially designed loss function that is called *neg-log-likelihood (NNL)*

$$L^{NLL}(\beta) = -\sum_{i=1}^{n} \log p(y_i|x_i) + \lambda\|\beta\|_2^2. \tag{2.6}$$

Here, $p(y_i|x_i)$ is the probability of observing class y_i when presented the features x_i. A good classifier should aim to maximize the probability of predicting the right class for a context. Hence, maximizing the probabilities minimizes the neg-log-likelihood. For a fixed set of labels $y_i \in \mathcal{Y}$ the probability $p(y_i|x_i)$ is defined as

$$p(y_i|x_i) = \frac{e^{f(x_i,y_i)}}{\sum_{\hat{y}\in\mathcal{Y}} e^{f(x_i,\hat{y})}} \quad \text{with} \quad f(x_i, y) = x_i^T\beta.$$

Note, that the probability can be calculated for both classes. Hence it not only provides the class for a given feature vector, but also a probability distribution over all possible classes. Therefore, having a data point one gets an estimate for *all* classes about its likelihood to belong to it. A sample probability distribution using multi class logistic regression for an example with 6 classes is shown in figure 2.4.

The β vector for a class is obtained by setting the derivative of the neg-log-likelihood to zero. As there exists no analytical solution, one has to use an iterative method such as the Newton method to extract the solution. A full derivation of all required formulas for the *multi class case* that were used in ConComM can be found in [Li16] and [Tou15].

A second method for classification is to use **Support Vector Machines** (SVMs). The following explanation is inspired by Friedman, Hastie, and Tibshirani [FHT01]. For support vector machines, a hyperplane is constructed that separates the two classes while minimizing the wrongly classified points. To seperate the two classes, the so called margin measures the minimal distance a point that was correctly classified can keep from the hyperplane. A maximized margin means that the hyperplane, which marks the decision boundary, is positioned with maximized distance from both clusters. Hence the clusters are separated in the best possible way. To maximize the margin is therefore the first goal when constructing the hyperplane. We illustrate the situation in figure 2.5a. A suboptimal margin is shown in red, the optimal margin is marked in gray.

However, in most cases it is not possible to construct a hyperplane that separates both classes without data points that lie in the wrong cluster on the other side of the

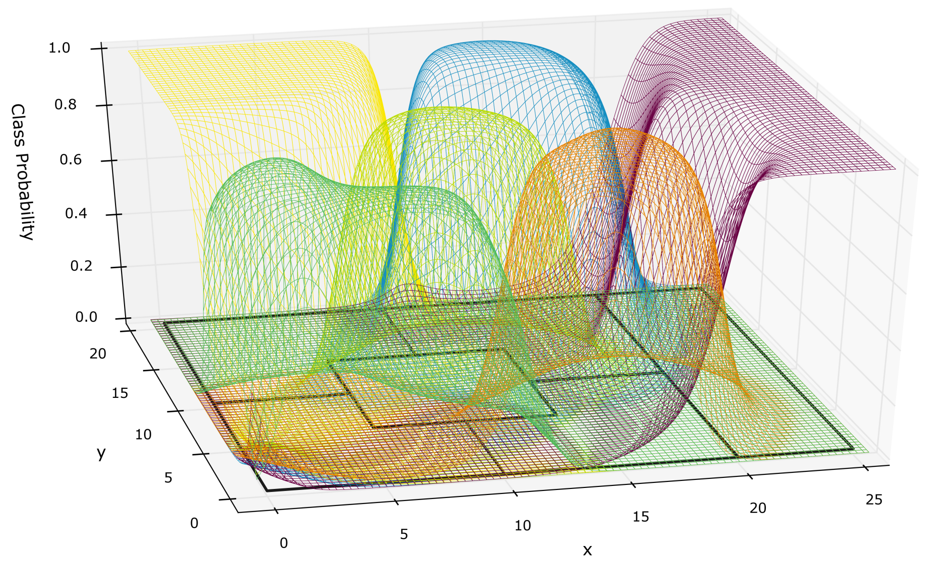
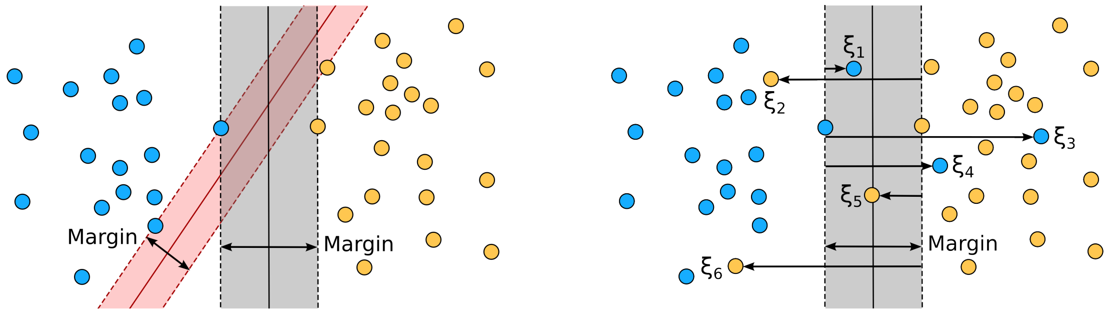


Figure 2.4: Logistic Regression Probability Distribution

Multi Class Logistic Regression trained on sample points that were labelled with their container. The containers are areas in a 2D space and displayed at the bottom. Each sample point was characterized by its x and y location. The logistic regression used quadratic features to obtain the class functions. The final probabilities for each point and each class are shown, using a different colour for each class. Note that these are *not* the class functions.



(a) Different Margins **(b)** Support Vectors

Figure 2.5: Support Vector Machine (SVM)

The figures illustrate the two parameters a SVM tries to optimize. Figure 2.5a demonstrates the positioning of the hyperplane within the feature space. The hyperplane shown as black solid line maximizes the margin that is shown as dashed lines. The hyperplane in red is a suboptimal solution. Figure 2.5b shows the case for non-separable clusters. The points outside the cluster-margin are the support vectors with an assigned a weight ξ.

hyperplane. These data points are called ‘Support Vectors’. The situation is pictured in figure 2.5b. Support vectors are assigned a weight $\xi \neq 0$ that is proportional to the distance to margin for their cluster. This means, a support vector far within the other cluster has a higher weight than a support vector that did only exceed the margin slightly. A data point within the right cluster that not exceeds the margin has $\xi = 0$. SVMs not only aim to maximize the margin but also to minimize the sum over all weights ξ.

Besides logistic regression and SMVs, two other groups of methods are worth mentioning. The first one are **Decision Trees** and **Random Forests**. Decision trees are a chain of binary decisions. Depending on whether a criterion is met, a branch leads to a next criterion-check or a decision. Using the techniques *Bagging* and *Boosting* can turn simple decision trees into very powerful classifiers. The second method is **Deep Learning** where a neural network is simulated and trained to output the correct class. Because deep learning requires huge amounts of training data, this method was not considered any further as the datasets within this work are too small for this method.

To come from binary class classification to multi class classification, there exist two schemes that can turn any binary classifier in a multi class classifier [Bis06]. The first one is based on a one-against-the-rest idea, where for each class a classifier is trained with labels that are 1=inClass or 0=notInClass. In the end, the class is assigned to the point that scored the highest prediction value. This approach is used for logistic regression. The second option is a one-against-one approach, where for a classifier is trained for each pair of classes. In the end, the point is given the class with the highest number of predictions. What makes logistic regression very useful for such a case, is that it internally already uses the one-against-the-rest procedure. Therefore the extension to a multi class case comes quite naturally.

2.3.6 Remarks on Online Classification

As for the cut algorithms, there exist a number of online versions for most classification algorithms. To name a few examples, Cao et al. [CMBP04] performed online motion classification with support vector machines, the online version of regularized classification algorithms were mathematically investigated by Ying and Zhou [YZ06]. Ridge regression is not an online algorithm by name, however it can be trained as long as one wishes. This implies, it also can be trained with the same input but another label in case the label changes and adding new training data only requires one update to obtain the modified classifier.

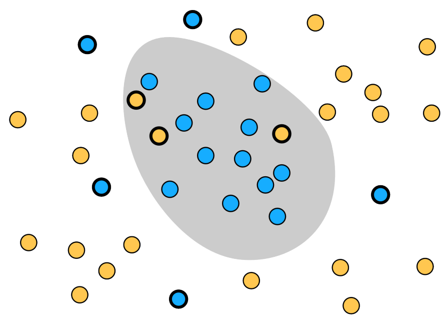
Nevertheless, what was said for online graph cuts equally holds for online cut algorithms: developing our framework to work as online framework is an interesting option that definitely is possible due to the number of possible online algorithms. However, this is not the focus of our work and therefore this topic is not deepened further.

2.4 Quality Measures for Clustering and Classifier

2.4.1 Quality Measures

Clustering is a typical task for machine learning algorithms, therefore most quality measures to evaluate the goodness of a clustering come from that field. This subsection will start to explain quality measures for two clusters which equals having a prediction for one class. Their generalization to the multi class case is discussed in the second part.

Table 2.2: Sample Clustering and Confusion Matrix

Classification Illustrated	Confusion Matrix
	

		Predicted Class	
		True	False
True Class	True	true positives (12)	false negatives (5)
	False	false positives (3)	true negatives (20)

The blue coloured dots represent the class of interest. A sample predicted area for blue points is shown in grey. All points in the grey area form the predicted class that corresponds to the blue dot class. Points with thick outlines are misclassified and therefore yield an error. The confusion matrix for this prediction shows the four prediction cases out of which exactly one describes each point. The total number of points falling into each class is given in the table. To illustrate the example, cells corresponding to cases where the class of a point is blue are coloured in blue. If the true class of a point is not blue, the cells are coloured in yellow. Cells corresponding to the predicted grey class are coloured in a darker tone, while cells representing a negative prediction and therefore the white area have a lighter tone.

To give an illustrated introduction to the different cases of correct and misclassification, assume many data points spread in a 2-dimensional space. There are blue points that form a class, as shown in the illustration in table 2.2. [SL09]. A classifier learnt to predict this class, its prediction for the class is shown as grey area in the figure. If a point is within the grey area, the point is *predicted* to be blue. Now there are four different cases out of which exactly one has to be true for each point. There are two variables that can either be true or negative. The first variable is the **true class**. It is positive if the point is really blue, which means it is in the class of interest. If the point is not in the class of interest, the true class variable is false. The second variable is the **predicted class**. Predicting ‘true’ is equal to claiming that the prediction for the point to be in the blue class is positive. Therefore, the grey area covers all points for which the predicted class is true. The white rest of the background is the opposite case, as points in this class are predicted not to be in the blue class. Therefore the prediction for these points is negative.

Having two options per variable, this yields the following for cases:

- **True Positive (TP)**: The point is predicted to be in the blue class and the point is truly blue - the prediction is therefore positive. Points in this class were correctly predicted to be in the class of interest. In the illustration, these are all blue points in the gray area.
- **False Positive (FP)**: The point is predicted to be in the blue class (prediction positive), but the point is not blue. In the illustration, these points are orange within the grey area. As the classification for them is wrong, this is also called a *type I error*.
- **False Negative (FN)**: A point is predicted not to be in the blue class - the prediction for it to be blue is negative - but its true colour is blue. Those points are blue on a white background. Again the classification is incorrect, this particular type of misclassification is called *type II error*.
- **True Negatives (TN)**: Those points are predicted not to be blue (negative prediction) and as they are true negatives, their true class is not blue. This is the case for all yellow points on a white background.

These four cases are arranged in the so called **confusion matrix** that is also illustrated in table 2.2. This confusion matrix also contains the total number of points falling into each class. All points that introduce an error, irrespective of whether its type I or II, have thicker outlines.

Knowing TP, FP, FN and TN for a clustering, there exist some measures to quantify the quality of a prediction. The following measures are analysed in great detail in the paper of Powers [Pow07]. The most common measures are **Precision** P, **Recall** R and their inverse counterparts **Inverse Precision** P_i and **Inverse Recall** R_i. They are defined as follows:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{and} \quad P_i = \frac{\text{TN}}{\text{TN} + \text{FN}},$$
$$R = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{and} \quad R_i = \frac{\text{TN}}{\text{TN} + \text{FP}}.$$

The precision is also known as *confidence*. A high precision indicates that not many samples were erroneous predicted to be in the class of interest. Aiming to maximize precision often comes to the price of missing out points that would belong to the cluster. To illustrate this fact, assume one would want to maximize the precision of the cluster shown in 2.2. Shrinking the egg shaped grey class until no yellow points are covered by its area leads to a precision of 1.0 as no false positives are left in the area. However, many blue points would be missed out by that procedure and this effect is measured by the recall or sensitivity. The recall quantifies how many blue points were missed out by the current cluster, as it is the fraction of correctly identified blue points over the total number of blue points. Assuming that we only have two clusters, inverse precision and recall measure the same quantities for the second cluster.

However all these values are biased, which is shown by Powers [Pow03]. The author proposes a new measure that is derived based on the bookmaker strategy that aims for a fair bet. Given the odds $X : Y$. X is the amount the betting party wins if the bet was correct and Y is the amount that is lost otherwise. The bookmaker tries to give the odds in a way, that the expectation value for the bet is 0. Therefore he has to give X and Y such that

$$0 = X \cdot \frac{Y}{X+Y} - Y \cdot \frac{X}{X+Y}.$$

Here X has to be low if the probability of Y being correct is high. Applying this thinking pattern to the quality measures above, Powers derives an unbiased quality measure which is called **Informedness** I_{nf}. The informedness gives the fraction of times when the classifier makes an informed decision instead of guessing. It is given by

$$I_{nf} = R + R_i - 1.$$

As the informedness is an unbiased measure using only the recall, there also exists a similar formulation using the precision. This measure is called **Markedness** M_{ark} [Pow07] and defined as

$$M_{ark} = P + P_i - 1.$$

Informedness and markedness can be unified in to a last notable measure that is called **Correlation** C_{orr}. The correlation quantifies how much of the variance in the predicted values can be explained by the reality. It is given by

$$C_{orr} = \sqrt{I_{nf} \cdot M_{ark}}.$$

This concludes the overview of important quality measures.

So far only quality measures for the two class case were given. Sokolova and Lapalme [SL09] summarize the two ways of calculating precision and recall for more than two classes. Both ways are based on the idea of evaluating the confusion matrix for each cluster. Hence the values for the different clusters have to be averaged in a way. The two procedures to obtain the measures for the multi class case differ in the averaging procedure and are called **Macro-Averaging** M and **Micro-Averaging** μ.

Assume $c = 1, \dots, k$ classes that for which one want to calculate precision and recall. In this case, the macro and micro averaged values precision and recall are given to [SL09]

$$P_M = \frac{\sum_{c=1}^{k} \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c}}{k} \quad \text{and} \quad R_M = \frac{\sum_{c=1}^{k} \frac{\text{TP}_c}{\text{TP}_c + \text{FN}_c}}{k},$$
$$P_\mu = \frac{\sum_{c=1}^{k} \text{TP}_c}{\sum_{c=1}^{k} \text{TP}_c + \text{FP}_c} \quad \text{and} \quad R_\mu = \frac{\sum_{c=1}^{k} \text{TP}_c}{\sum_{c=1}^{k} \text{TP}_c + \text{FN}_c}.$$

For micro averaging the result of each class is weighted with the classes number of points influencing the measure. For macro averaging, precision or recall are calculated for every class and averaged with equal weights, regardless of the size and therefore 'importance' of the class.

2.4.2 Cross Validation

Cross validation is an important tool for the evaluation of classification algorithms. The idea is to measure an algorithms performance with a fixed number of training samples if no new samples can be used to test the algorithms performance on them. To do that, the training data is split into m equally sized smaller sets. The most common procedure is the **leave one out cross validation**. The name already explains the procedure, as the classifier is trained on $m - 1$ samples and its quality is evaluated on the predictions for the left out sample. After doing this for all the small sets, the results are averaged. Some more details on how we used cross validation for our evaluation are given in chapter 5.2.2.

3 Towards an Ambient Space Simulation

Essentially, all models are wrong, but some are useful.

(George E. P. Box)

In the first chapter we introduced the concepts of IoT and WoT and explained why context sensitive recommendations play an important role for the WoT vision of discovering, composing and executing services. We also presented preliminary results of a contextual bandit algorithm that was used for IoT service recommendation. In the course of this evaluation, we indicated that a meaningful experimental design often requires environments that are rich in IoT objects and can therefore be problematic. Our simulation is able to emulate environments that do not only possess a hight density of smart services. The environments as well as the objects are rich in contextual information and form an ambient space for simulated users and recommendation algorithms. Therefore we refer to our project as ambient space simulation.

3.1 Back to the Future: The Lack of Real World Data

3.1.1 Motivation

The challenge of evaluating bandit algorithms for service recommendation in a smart space lead to our first contribution, which is covered in this chapter. In order to provide an environment that can be used to test an algorithm's performance under 'real' conditions, we developed a simulation to imitate users in an ambient smart space.

Having a simulation to test IoT service recommendation has multiple advantages over a real usability test. At first, the simulation can emulate a huge number of IoT services that would have to be bought for a usability test. Therefore it is much cheaper than gathering all these services in one building and to ask user to interact with them. Furthermore, a simulation can simulate any interaction and any IoT service - even if the service itself is not yet available on the market. And lastly, simulating a scenario is much quicker than to perform a real usability test. Additionally it can be repeated as often as required and whenever required, which is an advantage especially in early stages when the system needs a lot of testing.

With this simulation we aim to facilitate the evaluation of existing and new approaches to the recommendation challenge. Knowing that a simulation can only be realistic to some extent, it still seems like a good option to review the results obtained by real world data and extend the evaluation to greater scale scenarios. The next subsection will briefly explain how we structured our development process and give details on the requirement analysis, which we conducted as first developmental step.

3.1.2 Methodology and Requirement Analysis

We developed our ambient space simulation according to the waterfall development approach. In the following, we briefly describe our general course of action with a focus on the requirement analysis.

Recalling the data issue discussed in the last section, we formulate some basic **requirements** a simulation would have to fulfil, in order to make it suitable to evaluate recommendation systems. Since we want to investigate how users move and act in an environment that is full of connected services, we need an environment that can be filled with services. Also, one needs agents that are able to move through this environment and interact with different services. In the following, these agents are referred to as 'users'.

We are aiming to evaluate a recommendation algorithm that needs feedback after giving a recommendation. Hence every agent requires a personalized reward function that determines a feedback for each service. Last but not least there has to be a way to communicate with the recommendation system to send the current context, receive recommendations and return feedback.

Knowing the basic requirements, we started to **design** the simulation. Given the strong focus on an environment that consists of objects, we choose an object oriented solution. It was our first goal to create a simulation that is able to emulate one or several users moving through a virtual environment. The environment itself should consists of different rooms, that are filled with a diversity of smart services the user could interact with. The success and the users satisfaction with a certain interaction should hereby be based on two factors: A user profile to represent personal preferences and in addition to that technical factors such as having required apps, or being in the correct network to enable an interaction. Hence these factors should be considered in a reward function. More details on the design are given in 3.2.

Concerning the following **implementation**, we developed the simulation in Java using Java SE 8. All testing was done with Ubuntu 14.04 and selected implementation aspects are discussed in 3.3. Our IntelliJ project can be found on the 'supervisor CD' accompanying this thesis.

Lastly we **verified** our ambient space simulation. We did this by reproducing results of a real usability test for a bandit algorithm trained with simulated data. A detailed description of our experimental setup, results and discussion of this testing is given in 3.4.3.

3.1.3 Refining the Notion of Context

As our ambient space simulation aims to emulate environments that are rich in context, we require a more differentiated notion of context for the following sections. For a general notion of context, the definition 2.1.1 given in *Context and Context Awareness* is used. To adapt this very general definition to the reality of the simulator presented in the following subsections, we introduce the following distinction. Attributes of an object can either be **Geometrical** or **Profile Properties**, both classes combined will be referred to as **Contextual Properties**.

- **Geometrical Properties**: Attributes belonging to this category are important for the physical basis of the simulation. They are used to ensure that objects behave, to some extent, like objects in a real physical environment. As a rule of thumb, every attribute that would be required for an attempt to plot the scene is a geometrical property.
- **Profile Properties**: These Attributes are necessary to model a scenario that does also capture 'soft' influences such as user preferences or current network for services. These properties would not be needed if the goal was just to construct a floor plan, nevertheless these attributes will extensively be used when it comes to modelling the reward function.

Now that we defined the most important term for this section, the focus of the remaining section will be on how the simulation works and what design choices were made to evaluate the results.

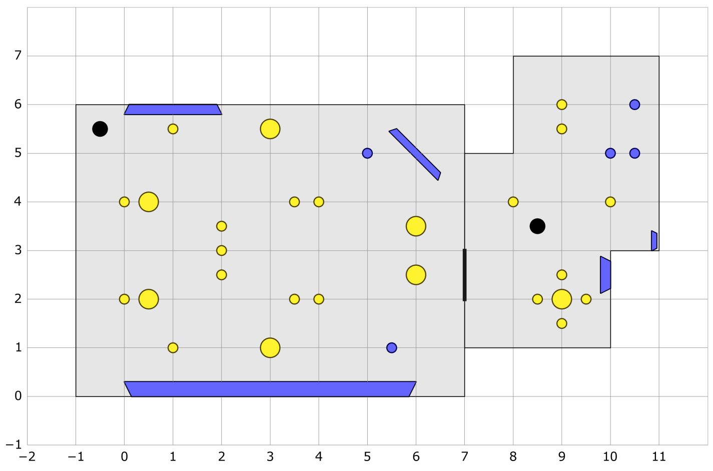


Figure 3.1: Simulation Setup

A sample initial setup for the simulation, the figure gives a birds eye view on the simulated building. We choose a floor plan with two grey rooms that are connected trough a door (thick black line). The floor plan contains IoTDevices such as lamps (yellow dots), monitors (blue boxes), unspecified IoTDevices (blue dots) and two users (black dots).

3.2 Concept and Design

3.2.1 General Concept: The Simulation Work Flow

With the results of our requirement analysis, we designed the following very general simulation work flow: In a first setup step we generate the floor plan of a building. The floor plan itself consists of one or many rooms that can be filled with arbitrary many services. To be consistent with the class names given in the simulation, these services are called IoTDevices and there exist multiple types such as light bulbs and monitors. Finally, an arbitrary number of users is added to the room and equipped with a personal profile.

A sample initial setup is shown in Figure 3.1. The service's and user's contextual properties are employed to generate the feedback for a recommendation in the later steps.

When the setup is completed, we execute the simulation for a previously specified number of time steps. In each time step, every user makes a step in a random direction. The specifics of how user movement is implemented are given in subsection 3.2.3 *User Movement Concept*. Next, we send a so called context snapshot to the recommendation algorithm. A context snapshot is user specific and contains contextual information such as current location, gender, age, current network and sensed services. The recommendation system takes this context snapshot and computes the most suitable service, which it returns as a recommendation to the user. The user receives this recommendation and evaluates its personalized reward function for it.

In a nutshell the reward function compares the properties of the recommended service to the user profile and tries to evaluate how well they match. We give more details on the used properties and how we model their dependencies are given in 3.3.2 *To Take or not to Take: The Reward Cook Book*. Evaluating the reward function results in a decision, where the user accepts or rejects the recommendation. This decision is then communicated back to the recommendation system. Whether the recommendation system uses this feedback to improve its predictions is irrelevant for our simulation. Once all users received recommendations and made a decision about them, the next time step begins until the required number of steps is simulated and the simulation terminates.

It is one of our main goals to keep our ambient space simulation extensible and as general as possible, rather than tied to a specific use case. Hence we designed it to be very flexible with respect to adding new IoTDevice types, or changing the flow of a simulation step. It is supposed to be a tool to set up complex service filled environments that can track users interactions with these services, irrespective of whether the interaction was induced by an algorithms recommendation or by any other influence. We strongly employ this flexibility in the experimental design in Chapter 5. In the next subsections, we give more details on the class conception to build the floor plan, user movement and the reward function.

3.2.2 Class Conception to Build a Floor Plan

In our simulation, a floor plan is a map of the world that supplies the world with a concept of location and space. Hence we consider it as the core of our simulation. It shows the position of different rooms and everything located within these rooms. The way we structured the floor plan strictly follows the object-oriented programming paradigm, which makes the structure very clear. A simplified class diagram that shows our most important classes with selected class variables and functions, is given in figure 3.2.

Our most important class in the whole ambient simulation is the **`World`** class. The world contains everything that will later interact on and move over the floor plan. It also keeps track of all objects currently being part of the ongoing simulation, which is an indispensable task for plotting.

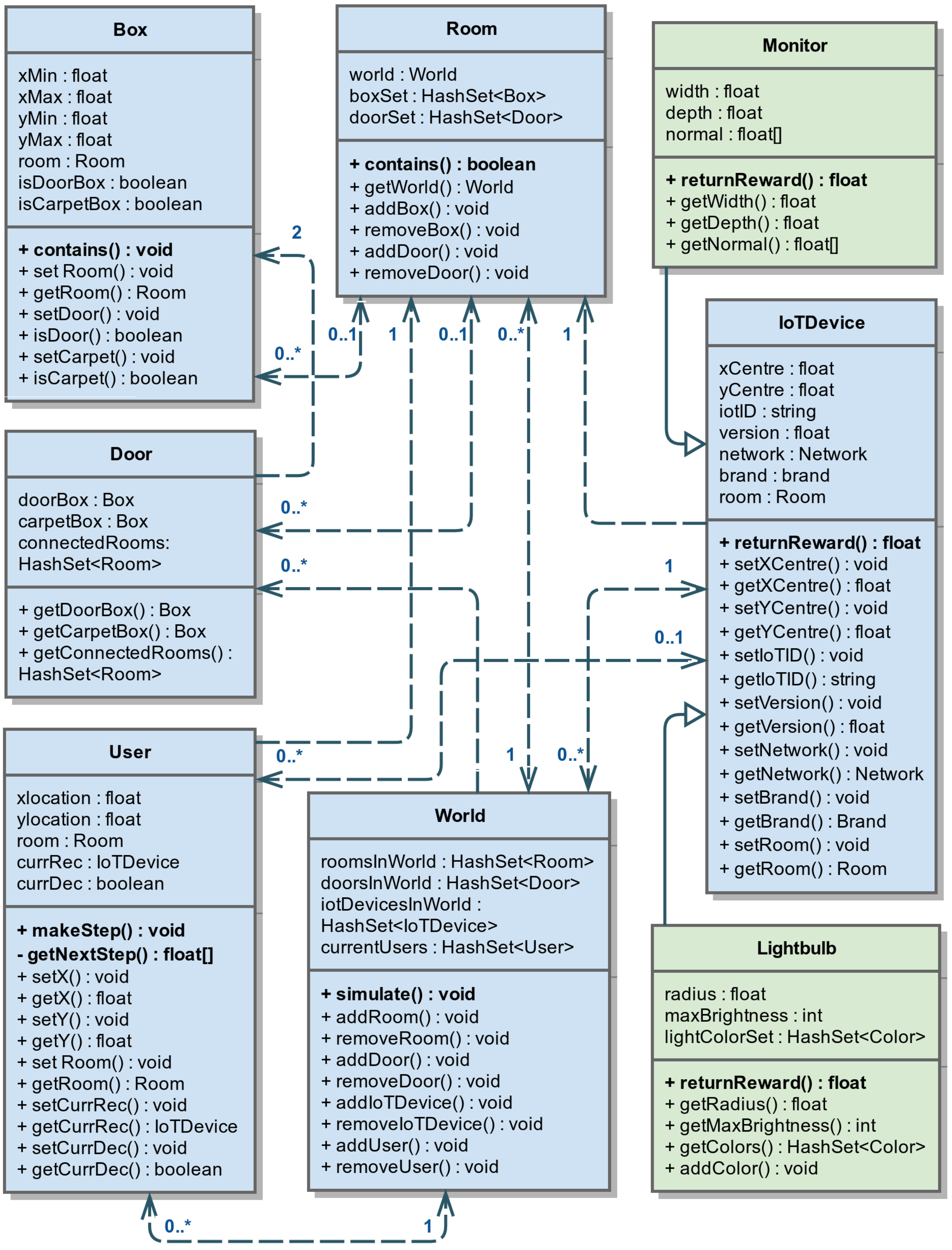


Figure 3.2: Simplified Class Diagram for the Ambient Space Simulation

Normal Classes are coloured in blue while subclasses are marked in green. Class functions that are not getter or setter for the class variables are highlighted in bold.

To have a surrogate for a service filled building was the main motivation to build a simulation. Therefore, the `World` principally requires **`Rooms`** to accommodate IoT services and users. In our simulation concept, a room is not much more but a collection of **`Boxes`**. Our intention behind defining a room as a collection of primitive shapes is that we want to be able to extend our shape collection to other primitives such as polygons or circles. In later stages, these primitive shapes can be used to define the shape of more general rooms. Boxes themselves are simple geometric objects that are defined by their extent in x and y direction. A table with selected class variables that cover the properties discussed in this section is given on page 50. After we created a box, it can be assigned to a room. Two rooms can not posses the same box and boxes assigned to different rooms are not allowed to overlap. When we assign a box to a room, the box will automatically adopt all important room properties such as world the room belongs to.

So far a user would not be able to move freely, at least when we are assuming that walking through walls is forbidden. Therefore we require some **`Door`** construction to enable user movement among rooms. More details on doors and how a they are constructed are given in subsection 3.2.3.

We have the intention to write a smart space simulation that simulates users in a service packed world, therefore it is time to introduce some services. Every smart service inherits from a class called **`IoTDevice`**. When ever we talk about a member of the IoTDevice class, it will be denoted as *IoT device* while *IoT service* will represent real world IoT services. The fact, that the class is called 'IoTDevice' instead of 'IoTService' results from an earlier development stage where the concept of an IoT service as the combination of devices *plus* online services was not yet clear. So far, all presented classes only had geometrical properties. Unlike them, IoT devices also have profile properties, which are marked with $\star$ in table 3.1. Those profile properties are mainly used for the reward function in 3.3.2 and will be explained there. In our simulation we define that a service can not be positioned if its centre location is not within the given room. Also the service must be entirely enclosed by the given room, which becomes for IoT devices with a physical extent. We would like to point out that general IoT devices do only have a centre point, but no physical extent in any direction.

Specialized IoT devices such as **`Lightbulbs`** or **`Monitors`** are subclasses of IoTDevice. These objects are used to represent smart light bulbs such as Hue or LIFX lamps, whereas a smart monitors could be an apple TV to name only one example. All these objects are network connected and can be controlled over this network. Those objects have additional contextual properties, of which we give samples in table 3.1.

Finally we introduce a class to model agents that is called **`User`**. We want users to move over our floor plan, which requires a movement concept that is explained in the next section. In the last subsection we suggested that users in combination with IoT services

Table 3.1: Ambient Space Simulation: Classes and their Variables

Class Name	Variable Name	Variable Type
World	roomsInWorld	HashSet<Room>
	doorsInWorld	HashSet<Door>
	iotDevicesInWorld	HashSet<IoTDevice>
	currentUsers	HashSet<User>
Room	world	World
	boxSet	HashSet<Box>
	doorSet	HashSet<Door>
Door	doorBox	Box
	carpetBox	Box
	connectedRooms	HashSet<Room>
Box	xMin	float
	yMin	float
	xMax	float
	yMax	float
	room	Room
	isDoorBox	boolean
	isCarpetBox	boolean
IoTDevice	centre_location	float tuple
	room	Room
	iotID	String
	version*	float
	network*	Network enum
	brand*	Brand enum
Lightbulb	radius	float
	maxBrightness*	int $\in \{0, \ldots, 6\}$
	lightColorSet*	HashSet<Color>
Monitor	width	float
	depth	float
	normal	float tuple
User	location	float tuple
	room	Room
	currentRecommendation	IoTDevice
	currentDecision	boolean

are subjects of a reward function. This concept is very important for this simulation and also given an own subsection.

3.2.3 User Movement Concept

In this section we explain how we integrate user movement in our simulation concept. We cover basic movement patterns and the realization of doors.

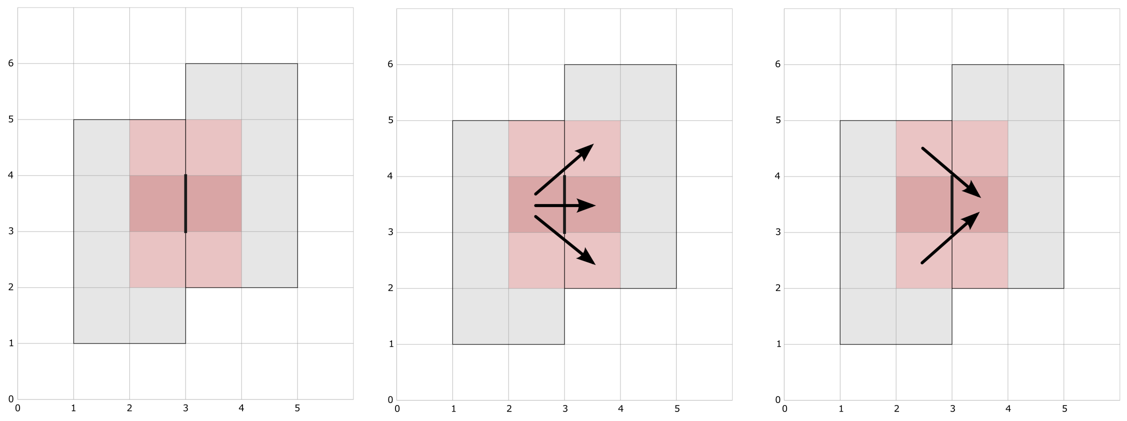
A user starts with some initial position that has to be within a room. In each time step the user can make a step, which means its position moves by a certain value in an arbitrary direction. This value is called `stepSize`. We discuss the specifics of different step pattern in the implementation section 3.3.1.

When we define user movement, we have to make sure that users do not leave a room and "teleport" through walls. Hence for every step we introduce the check, if the step a user tries to take is *valid*. If the step is not valid, we will try a step in another direction which is then again tested for validity. A first definition of validity that prevents teleportation, is to demand on `oldRoom==newRoom` for the old and new user position.

Using only the above `oldRoom==newRoom` validity criteria is realistic in a way that it does not allow teleportation through walls, because it does not allow the transition from one into another room. Nevertheless, in the real world there are object that do allow humans to move between rooms, without any magic being involved. If we want to have a simulation that models human movement patterns in a more realistic way, it needs to incorporate the concept of doors.

Because users do not have any fixed step size or walking pattern, our idea behind a door is based on the definition of special areas. When a user steps into such a special area, this area enables it to perform a step that would not be allowed if the user was not standing in the special area. The most simple door representation using this idea is the following: We define a rectangular area, that is half contained in both rooms. Now we establish the following rule: When a user steps into the area, its next step is also valid it remains in the special area - regardless of what room is there. This means as long as the user's steps are within the special area, the user can move freely from the first to the second room. To transit between rooms now comes down to stepping from room 1 into the special area covering room 1, moving to the special area in room 2. After the user stepped into the special area in the second room, its current room is the second room and now every new step in the second room is valid.

We use a more sophisticated door construction based on this general idea. Employing the box-concept, we define a door as two boxes and call one of them `door-box` and the other one `carpet-box`. The door-box is a smaller box (marked dark red in figure 3.3)



(a) Light red = `carpet-box`, dark red = `door-box`

(b) The additionally allowed positions for user in right `door-box`

(c) The additionally allowed positions for user in right `carpet-box`

Figure 3.3: Door Illustration with `doorWidth=stepSize=1`

Illustration of a floor plan with two rectangular rooms (outlines in black, mostly filled with gray) and a door (thick black line from $(4, 3)$ to $(3, 3)$). The dark red and light red patches visualize the special areas that relax the rules for valid steps when a user steps onto them. Subfigures 3.3b and 3.3c show the additional steps a user can make when situated in both sorts of special areas. A step is represented by an arrow that starts at the old position and points to the new position.

that is enclosed by the carpet-box (light red) as illustrated in figure 3.3a. It is important to note that the boxes overlap, so each area that is covered by the `door-box` is likewise covered by the `carpet-box`. Both box types allow certain steps that would not be allowed otherwise. A user that stands in a `door-box` is additionally allowed take any position within the `carpet-box`. This is demonstrated in figure 3.3b for a user standing in the left part of the dark red `door-box`. Hence the user can additionally enter all patches on the `carpet-box` in the right room. Similarly we allow a user in the `carpet-box` to take any position within the `doorbox-box` - regardless of whether the position is in its previous room. The situation is shown in figure 3.3c, users in the left carpet box are additionally allowed to enter the right `door-box`.

Usually a step would be rejected if `user.previousRoom` $\neq$ `user.newRoom`. However, being located in a door gives two additional allowed transition rules, that will result in a valid step even if the above rule is violated:

1. `doorBox.contains(previousLocation) && doorCarpet.contains(newLocation)`
2. `doorCarpet.contains(previousLocation) && doorBox.contains(newLocation)`

Within our simulation we used the combination of `door-` and `carpet-box` with this set of rules to model doors.

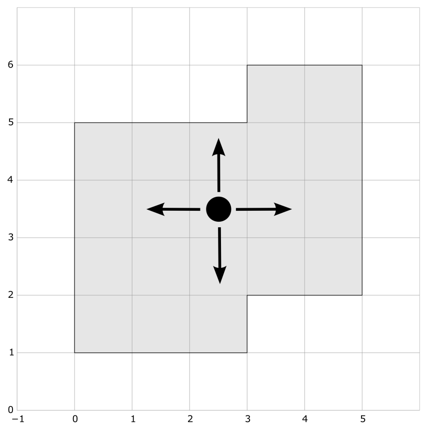
3.3 Implementation

3.3.1 Movement Patterns and Doors

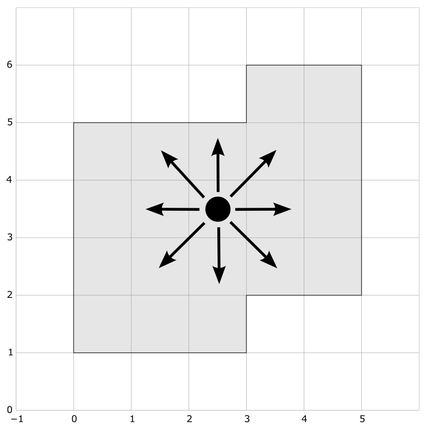
As suggested in 3.2.3, each user has a `stepsize` that influences the step pattern in a way. In our implementation, we assign this step size on inizialization and it can not be changed afterwards. When a user stands at a certain point on the floor plan, its step size regulates how far he can move in one step. We either allow a step to change the users position by `stepsize` in positive or negative x or y direction, as in 3.1. Or we allow a change by the outcome of a vector addition of scaled x and y direction as in 3.2.

$$p_{\text{after Step}} = p_{\text{before Step}} \pm \texttt{stepsize} \cdot w, \tag{3.1}$$
$$p_{\text{after Step}} = p_{\text{before Step}} \pm \texttt{stepsize} \cdot (1,0)^T \pm \texttt{stepsize} \cdot (0,1)^T, \tag{3.2}$$

with $w = (1,0)^T$ or $(0,1)^T$ and p $\in \mathbb{R}^2$. Only using the next steps described by 3.1 leads to a movement pattern that we call *purelyXY*, which is illustrated in figure 3.4a. If we use the steps allowed by equations 3.1 and 3.2 we get a pattern that we call *full* user movement. The pattern is shown in figure 3.4b.



(a) PurelyXY Movement Pattern



(b) Full Movement Pattern

Figure 3.4: User Movement Patterns

The figures illustrate the different step possibilities for a user (black dot) using different movement patterns. Possible next positions are marked with black arrows. The purelyXY movement pattern is described by equation 3.1, the full movement by 3.2.

That we have two patterns also means, we can choose which pattern to use for user movement, in order to determine the next step. An evaluation of both patterns that aims to find their assets and drawbacks can be found in 3.4.1. In addition to these two patterns, another choice to make is whether we allow the user to keep its current position for a time step or not. This decision is saved as a user variable that is called `allowNoMovement`.

To model a step sequence, a user performs a random walk given that each possible new position is equally likely. In this context, *possible* means allowed by the movement pattern. After a new position is drawn from the possible new positions uniformly at random, a check if the new position is valid follows. We explained the criteria for step validity in 3.2.3. New positions are drawn until a valid new position is found or a maximum number of draws is reached. The latter case results in no movement, irrespective of the specified user-rule. For our simulation it seemed favourable to gain the same trajectory for each user, whenever it was started with the same initial setup. Such a construction ensures that each recommender system gets the same situation with the same trajectory, making the results comparable. To achieve this goal, we assinged a private random number generator for every user that is initialized with the bit sequence of the user's initial position. This construction results in the desired behaviour.

After we explained the basic concept for doors in 3.2.3, the last paragraph describes the construction of a door within the simulation. For a door set up, the simulation requires a `centre` point, an `axis` along which the door is placed and a `doorWidth`. To ensure that every user can use the door properly, the door initialization parameter `doorWidth` should at least equal the biggest `user.stepsize` used in the simulation. Figure 3.3 shows a door with `centre=`$(3.0, 3.5)$, `axis='y'` and `doorWidth=1.0`. Using this information, a door box with measures $2\cdot$`doorWidth`$\times 1\cdot$`doorWidth` is created. A rectangle-shaped fraction of $1\cdot$`doorWidth`$\times 1\cdot$`doorWidth` extends into both connected rooms. The carpet-box is bigger with an extend of at least $2\cdot$`doorWidth`$\times 3\cdot$`doorWidth`. For the carpet box, a fraction of $1\cdot$`doorWidth`$\times 3\cdot$`doorWidth` should extend into each room. This is done in such a way that the 1×1 rectangle of the door-box is placed in the middle of the carpet-box's 1×3 segment.

3.3.2 To Take or not to Take: The Reward Cook Book

As we want to give a test environment for recommendation algorithms, we need to introduce a mechanism that evaluates the quality of a given recommendation. Hence we introduce a reward function that takes a user and an IoT device, which includes contextual properties for both, and returns a value in $[0, 1]$. Because most recommendation algorithms expect feedback in the form of acceptance$\rightarrow$1 or reject$\rightarrow$0, the return value of the reward function has to be mapped to a decision.

An easy and not too unrealistic way to achieve this, is to introduce a user specific value which will be referred to as `pickiness`. We define it as threshold in $[0, 1]$ and the reward function's outcome has to exceed or be equal to the pickiness in order to mark the recommended service as accepted. A service is rejected otherwise. If we know the reward function's output value and the user specific `pickiness`, it is possible to map the reward functions outcome into a decision.

By definition, with a pickiness close to 1 we make a user very strict about accepting services as they have to match very well to be taken. Small pickiness values on the other hand make the user accept even imperfect recommendations. So we can mimic users that either do not have strong personal preferences or do not take a lot time to look for better services. Because of this distinction the pickiness is one of the most important values for the simulation. Hence we need to choose it with some care for a direct recommendation comparison. After we applied this simple mapping to the outcome of the reward function, the decision can be communicated back to the recommendation system. But before we receive a decision, we first need to define how the user and service properties are linked in order to get the reward functions outcome in $[0, 1]$.

Every good recipe starts with the ingredients, hence we give the relevant properties used to model the reward function in table 3.2. Profile properties are most relevant for the reward function and marked with $\star$. Most values are self-explanatory, but a short description is required for our variables earlyAdopter and version. We introduced earlyAdopter in order to model a users affinity to use new releases. A earlyAdopter value close to 0 should affect the users satisfaction in such a way that the user does not care whether the service is a new release or not. In contrary a high earlyAdopter close to 1 should lower the users satisfaction with a outdated recommendation, as a high value indicates a strong affection to new services. To make this behaviour representable we introduced a version for IoT devices, which is the service-equivalent to earlyAdopter It is a value in $[0, 1]$ that equals 1 if the service was newly released and decreases to 0 the more outdated it becomes. Currently no ageing-mechanism for services is implemented; once assigned the version value remains as set until it is changed.

Now we use these properties to model relations between users and IoT services, so we connect these attributes in a useful way. In a last step we explain how different reward functions are constructed using these interrelations. To increase readability we use `IoT` to address an IoTDevice object in formulas and pseudo code.

One of the first things that comes to ones mind when thinking about a context dependent function, is to use the distance between objects. A position is a very simple geometrical attribute and the distance between two objects can be easily computed, given that we know the position of everything within the simulation. A modelling choice we made is to assume a reward that decreases linearly with increasing distance. Because we defined

Table 3.2: Variables for the Reward Function

Class Name	Variable Name	Possible Values	Variable Type
User	pickiness*	$[0,1]$	float
	position	$[-\infty,\infty]\times[-\infty,\infty]$	float tuple
	gender*	{0(m),1(f)}	int
	appNames*	all combinations of apps	HashSet<App>
	network*	each implemented network	Enum
	activity*	{Presentation, Conversation, Break, Brainstorming, Reading, VisuallyDemanding}	Enum
IoTDevice	position	$[-\infty,\infty]\times[-\infty,\infty]$	float tuple
	network*	each implemented network	Enum
	availability*	{true, false}	boolean
	version*	$[0,1]$	float
	brand*	each implemented brand	Enum
LightBulb	maxBrightness*	$\{0,,6\}$	int
	lightColors*	all combinations of colors	HashSet<Color>

List of relevant variables for the reward function with their possible values. Variables that represent profile properties are marked with $\star$.

the reward as non negative, the value can not go below 0. The most simple way to compute such a reward Rew is the following:

$$\texttt{Rew} = \begin{cases} 1 - \frac{1}{10}\cdot \texttt{dist(user,IoT)}, & \texttt{if dist(user,IoT)} < 10. \\ 0, & \texttt{otherwise}. \end{cases} \tag{3.3}$$

By this definition, the reward will be 1 if `user.location = IoT.location` and linearly decreases to 0 until `dist(user,IoT) = 10`. Another geometrically motivated interrelation is the following: even though a service is only a few steps away, its position may be in another room or a wall of the current room blocks the view. In this case, a user would most likely not accept the recommendation because he can not see the object and might not be able to physically reach it. With these two considerations we from the geometrical interrelations:

- **Distance Dependency:** The distance based reward is computed using 3.3 which gives a high reward for a small distance and decreases linearly with increasing distance.
- **Unblocked View:** A ray cast from the user to the service stays within the room and is not blocked by any wall. The reward will be set to 0 otherwise.

The *Unblocked View* was the main reason to implement **Ray** objects that would allow to compute intersection points for two rays. For their definition and more details on how they are used within the situation, please refer to *Algorithms and Derivations*. To determine if a ray stays inside a room is not as easy as it might seem at the first moment. A detailed description of my algorithmic solution to this problem, can be found in appendix A.3.

So far we only considered geometrical properties to construct the interrelations. We will now give a number of criteria that have to be met, otherwise the reward is set to 0 immediately.

- **IoT Service Available** The service is not in use by another user and therefore not available to further usage. Also it is not switched off.
- **Networks Compatible** User and IoTDevice are in the same network, which makes it possible for them to connect.
- **Device Usability** The user phone satisfies the required preconditions to connect to the service. This is modelled through apps that enable a user to connect to certain services.

The first criteria requires a simple check of the IoTDevice's attribute `availability`. We implemented a mechanism to block services that works as described below: Users can have an IoTDevice that is the `currentRecommendation` for the user, along with a boolean `currentDecision` that indicates whether or not the recommendation was accepted. If a user accepts a recommendation, we ensure that setting `currentDecision == true` goes along with setting the services availablitiy to `false`, which blocks the IoT device from being accepted by any other user. The second criteria is not hard to verify either. For the third interrelation, the concept of apps was introduced to the simulation. An **App** maps an `appName` to one or multiple `brands`. If a user has an app in its personal app list that enables him to communicate with services sharing the IoTDevices brand, the last criteria is met.

However we do not only have matching criteria, but also interrelations that require more careful modelling. Linking the user's value earlyAdopter to the IoTDevice version is one of them. Given that a high earlyAdopter value means new releases are very important to the user, we model the interrelation as follows:

$$\texttt{Rew} = \begin{cases} \texttt{Rew} - \frac{1}{3}\texttt{earlyAdopter}(1 - \texttt{pickiness}), & \texttt{if version} < \texttt{earlyAdopter}. \\ \texttt{Rew}, & \texttt{otherwise}. \end{cases} \tag{3.4}$$

The term (1 - `pickiness`) is the value by which the reward function can decrease before the user rejects a service. This means, the more important new releases are to the user, the more the reward will decrease if the service does not meet the users preferences. These considerations lead to the next interrelation:

- **Device Too Old** `if user.earlyAdopter > IoT.version`, the reward is reduced according to 3.4, otherwise the reward remains as it is.

For lamps, there were extra two extra profile properties: The maximalBrightness and the colorList. Linking the maximal brightness to a user property, requires a slightly more complicated mapping. Zhao et al. [ZARP15] investigated what ambience a person would prefer for a certain task. Their study demonstrated that some tasks require a brighter ambience than other. Based on their results, we came up with an Activity→Brightness mapping that maps user activities to the preferred brightness level: `Conversation`→3, `Break`→3, `Brainstorming`→4, `Reading`→4, `Presentation`→5, `VisuallyDemanding`→6. Combining this mapping with a lamps maximalBrightness in $\{1, \ldots, 6\}$ is now easier.

$$\mathtt{Rew} = \begin{cases} \mathtt{Rew} - \frac{1}{6}(\mathtt{mapping(activity)} - \mathtt{maxBrightness})(1 - \mathtt{pickyess}), \\ \qquad \mathtt{if\ maxBrightness} < \mathtt{mapping(activity)}. \\ \mathtt{Rew}, \qquad \mathtt{otherwise}. \end{cases} \tag{3.5}$$

This function yields in a reward reduction that is equal to the fraction of the brightness-difference over the overall maximalBrightness. The less the levels match, the more the reward is reduced. The following two lamp interrelations conclude the constrains that can be applied to the reward function:

- **Activity Brightness** If the activity value is greater than the light's maximal brightness, the reward is reduced proportional to the value difference as defined in 3.5
- **Pink Issue** A male user will not accept a lamp that can display pink light. If pink is among the lightColor, the reward will be set to 0.

This leaves eight interrelations that can be considered in a reward function. In the remainder of this thesis, there will be two types of reward function that are used to gather data and perform different types of analysis on top of that.

1. **Profile Based Reward** This function assumes an initial reward of 1 and applies all criteria based on profile attributes to it. It does not consider any information about the user's or IoTDevice's location. This leaves the interrelations *IoT Service Available, Networks Compatible, Device Usability, Device Too Old* and in case of a lamp *Activity Brightness* and *Pink Issue*.

2. **Distance and Profile Based Reward** Here the initial reward is obtained from evaluating the *Distance Dependency*. Next, the second geometrical criteria *Unblocked View* is applied. The resulting reward is used as initial reward before applying the profile based criteria.

This concludes the explanation of important aspects, that covered how the ambient space simulations models users, services and rewards. The last section in chapter 3.2 focus on applying the simulation rather than constructing it.

3.4 Evaluation of and with the Simulation

In the last sections, we gave some insights into how an IoT service filled environment is modelled and implmented within the ambient space simulation. Now it is time to explore the simulations features and how real recommendation systems perform in this environment. In this section we first evaluate two movement patterns. Afterwards we perform experiments with LinUCB in a simulated environment and compare the data to the results of a real usablility testing. In a third and last step, the performance of LinUCB for *multiple service recommendation* is investigated.

3.4.1 Movement Pattern Analysis

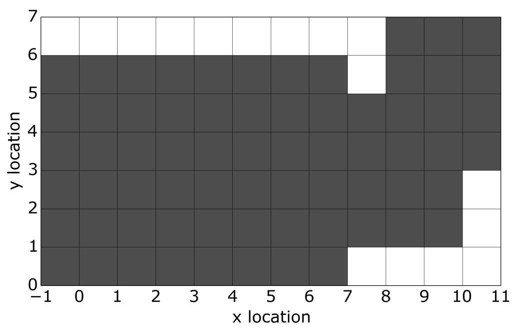
To investigate how well the both movement patterns purelyXY and full presented in 3.3.1 cover the floor plan if a user is moving freely, we performed a simple experiment.

Procedure: To obtain the coverage of user visits over the floor plan, we simulated 10 users with the different movement patterns performing 1000 steps each. For the evaluation we split the floor plan into patches of 1×1 and accumulated the number user visits per patch. The results were logged and plotted.

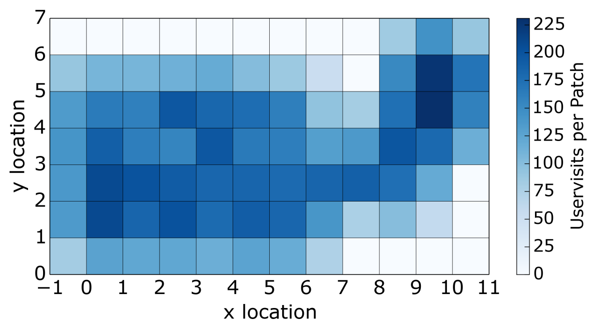
Results: The coverage of a floor plan, using the full and the purelyXY movement style is shown in figure 3.5b and 3.5c. The floor plan is the one illustrated in in figure 3.1. Overall the coverage is steadyly, given that every patch has been visited several times for both patterns. However the edges of a room are not frequented as strongly as the patches in the middle. The purelyXY pattern covers the edges better than the full movement pattern and also leads to a more uniform distribution of visits, given that the full movement pattern produces stronger peak-patches that have been visited very often or very rarely.

Discussion: The effect of rarely visited edges results from the fact that a patch in the centre has 8 for full, or 4 for purelyXY, surrounding positions a user could use to enter the patch. A patch on the edge in contrast can only be entered by 5 respectively 3 patches, of which the two neighbouring edge-patches themselves have a smaller probability of being entered. For a patch in the corner only 3 or 2 patches remain, of which 2 have limited access.

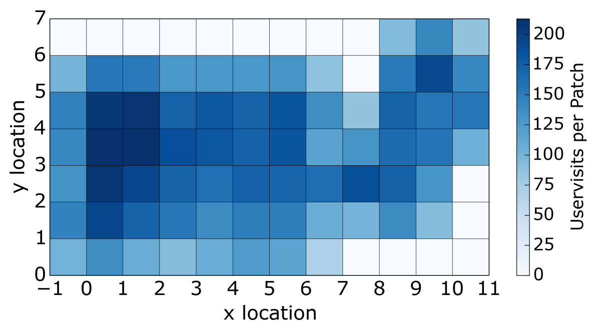
Our result that purelyXY is able to cover the floor plan more smoothly is a consequence of the above described patch blocking, which makes edge-patches more likely to be visited when using the purelyXY pattern. A simple calculation illustrates the case: An edge patch can still be entered by $3/4$ of the patches using purelyXY, while for full only $5/8$ remain. This means that edges are more likely to be visited by a user moving based on purelyXY. The same calculation can be done for patches in a corner, that are accessible by $2/4 = 1/2$ of the overall possible patches for purelyXY, but only by $3/8$ for the full movement pattern. For all these boundary patches, it is $1/8$ more unlikely to be entered by a user employing the full movement pattern.



(a) Floor plan basic shape



(b) Coverage for *full* movement pattern



(c) Coverage for *purelyXY* movement pattern

Figure 3.5: Floor Plan Coverage

Floor plan coverage based on the movement of 10 users with different initial positions, performing 1000 steps each.

This observation also implies that these users are more likely to be trapped in small rooms because it is hard for them to enter doors, given that patches behind doors have a limited number of patches to be entered by. The trapping-effect explains the peak values, that can be observed in the upper right corner in 3.5b for the full movement. The overall coverage is very good for both movement patterns, but the purelyXY movement is favourable due to the smoother coverage explained above.

3.4.2 Gathering Data

Recommender Systems have been extensively applied to providing personalised online advertisements. In this area, one of the key indicators for the performance of a recommendation system is the **clickthrough rate (CTR)**. We use it to describe how often a displayed add was clicked and is calculated as the number of clicks over the total number of display times [Goo16].

Therefore the CTR is a percentage giving the number of good recommendations over the ignored ones. A high CTR is indicating a good recommender while a low CTR shows that the average number of recommendations was not good, because users did not consider the suggested adds worth clicking.

When we apply this thinking pattern to the simulation, each user request for a recommendation can be seen as an opportunity to display a list of service possibilities to the user. More specifically, if our user accepts the recommendation and connects to the device, this equals clicking an advertisement. Hence we implemented the clickthrough rate as quality measure, as our situation is similar to the advertisement model above.

Within the simulation, in each time step for all active users, one recommendation is requested and the total number of requests is increased by one. A second acceptance counter is increased if the reward function for the recommendation is higher or equal to the users pickiness. After we updates both values, the current CTR is computed as user-specific acceptance counter divided by number of requests. The value is then logged to a file. Along with the raw CTR, the users position is saved to evaluate the room-coverage pattern. Saving this information also gives us the opportunity to compute a localized CTR per position, in order to evaluate the spatial quality of recommendations.

3.4.3 Evaluation of LinUCB for Single Service Recommendation

With the information logged as described in the last subsection, in this section we address three questions with our evaluation of LinUCB for single service recommendation. To compare our results of a simulated user to results that already exist from studies with real users as described in 2.2.4, a first test evaluates the influence of the model parameter α on LinUCB's recommendations. If the influence of α on the simulated experiment is similar to its influence on the real experiment, the simulation succeeds in its goal to model the experiment in order to allow an up scaled experimental setup. In a second experiment we evaluate the quality of results obtained by LinUCB partial, which was specifically designed for IoT service recommendation, in comparison to the existing methods LinUCB disjoint and LinUCB hybrid. The two quantities of interest are CTR evolution over time and a view on the model that reveals spatial patterns in

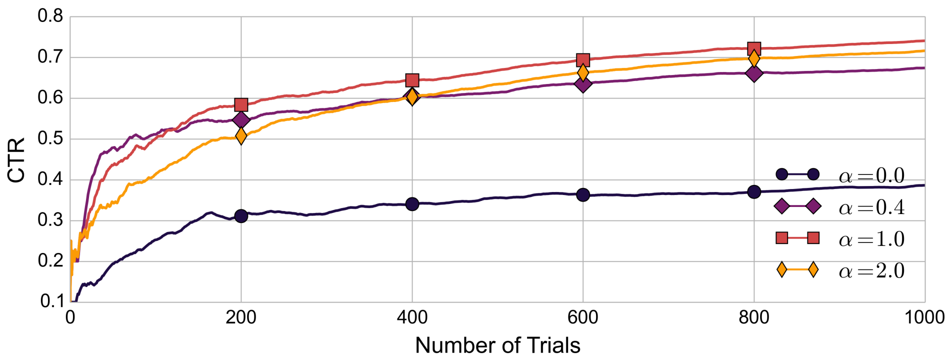


Figure 3.6: Influence of α on LinUCB

Logged Clicktrough Rate for the LinUCB disjoint and different variations of the model parameter α. The algorithm was trained on the sample scenario shown in figure 3.1 with 10 users making 1000 steps each for each α. The plotted curves are averages over the 10 results.

the CTR distribution over the floor plan. The last point is particularly interesting as it investigates whether the quality of recommendations increases, the more often the contextual information is the same. In the simulation this means, if a user stands on the same spot for a while and requests services, the LinUCB approach should reliably learn good recommendations for this spot.

In the first tests for the LinUCB approach we evaluated how the model parameter α influences the quality of recommendations within the simulation. Previous experiments have shown that the best balance of exploration and exploitation is found for values around $\alpha = 1.2$.

Procedure: For the first evaluation, we simulated 10 users with different user profiles and initial positions for 1000 time steps each. The floor plan we used in this simulation is the sample plan shown in figure 3.1. The bandit algorithm itself, LinUCB disjoint, was configured with the α values between 0.0 and 2.0. With each α configuration we trained the algorithm for all 10 users separately. The CTR was logged over the time steps and averaged over the 10 users for all α's.

Results: Our results are plotted in figure 3.6. For $\alpha = 0.0$ the CTR is lowest and does barely exceed 0.35 even for 800 time steps. With increasing α, the CTR also increases over time. The performance for $\alpha = 0.4$ is still lower than for the remaining values $\alpha = 1.0$ and $\alpha = 2.0$ on the long run, even though it outperforms them during the first time steps. Values around $\alpha = 1.0$ (which includes $\alpha = 1.2$ and $\alpha = 1.4$ that are not plotted to keep the plot clear) perform best with a CTR exceeding 0.7 for more than 700 time steps. Even though $\alpha = 2.0$ outperforms $\alpha = 0.4$ for number of trials greater than

400, the CTR increases slowly for the first time steps and is always worse than for values around $\alpha = 1.0$.

Discussion: In this evaluation we observe that very limited exploration ($\alpha = 0.0$) results in poor recommendations and consequently in a low CTR. Further, the quality of recommendations saturates for a good balance of exploration and exploitation, which occurs for α around $\alpha = 1.0$. This is in accordance with earlier experiments and indicates that the simulation serves its purpose to scale up and extend previous experimental setups.

What we also observe in the plot is that the quality of recommendations and hence the CTR decreases again as α reaches $\alpha = 2.0$. Here, exploitation is cut back due to a lot of exploration, which leads to significantly delayed performance growth for the first 400 time steps. Approaches that encourage exploitation score better results in this stage of the learning process as they actively use the exploration results and do not focus on exploration alone.

We see that experiments based on simulated data confirm preliminary findings for this problem class, hence we used it to conduct further experiments. The experiments focus on the LinUCB modification LinUCB partial, which was introduced in 2.2.4. Recalling the algorithmic idea, LinUCB partial only requires feedback for a few service per recommendation with minimum correction rather than demanding on full feedback. This approach should lead to a faster convergence and better overall recommendations compared to LinUCB disjoint and LinUCB hybrid, which were introduced in 2.2.3.

Procedure: Within the simulation, we realized the approach by recommending more than one service per request. The user would then return the feedback for all recommended services. We only logged the CTR for the first recommendation, as we consider this one as the 'real' recommendation the user would follow. For the accompanying services we only evaluated the reward function and returned the result to the LinUCB partial model. Following the logging procedure that was also used for the last experiment, we trained each model on 2 separate users walking for 500 time steps each and afterwards averaged over the users.

Results: The overall clickthrough rates for the approaches LinUCB disjoint, hybrid and partial are shown in figure 3.7. For most time steps, LinUCB partial dominates the disjoint and hybrid model. As the number of time steps exceeds 100, it outperforms the disjoint CTR by 0.1 on average. The disjoint approach is the worst performing out of the tested LinUCB models, as hybrid and partial outperform it for all time steps simulated.

Discussion: The results illustrates that the partial approach has a high performance for single service recommendations given that clickthrough rates around 0.8 can be

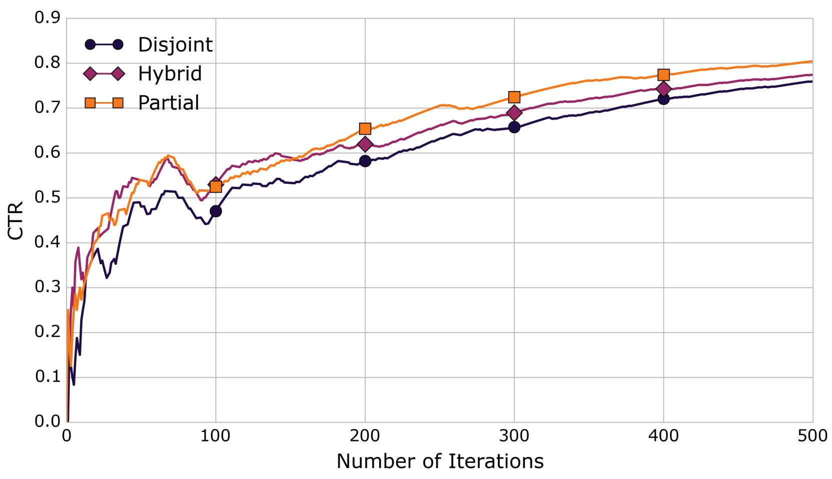
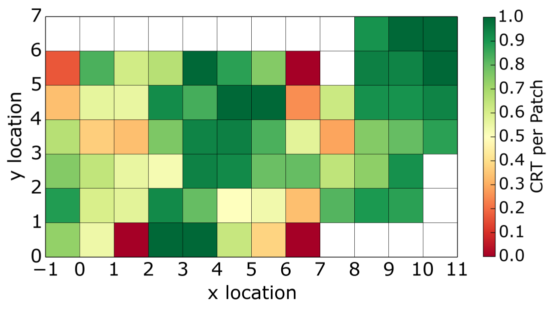


Figure 3.7: Clickthrough Rate (CTR) for Different LinUCB Models

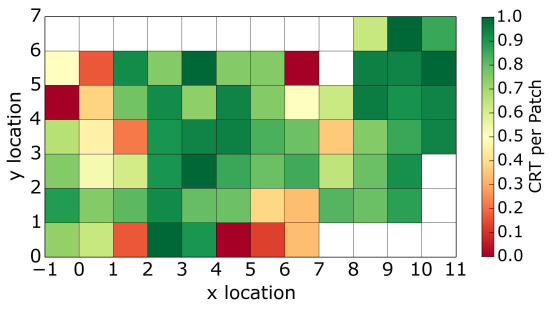
Logged Clickthrough Rate for the LinUCB models disjoint, hybrid and partial with $\alpha = 1.0$. The values were obtained by simulating four users, performing 500 steps each.

considered high for an average running time of 500 time steps. This means that the algorithm does give correct recommendations in 80% of all requests. That LinUCB disjoint does not use shared information is a real drawback in comparison to the other approaches. Due to its construction it performs linear regression over the features for each arm separately, but does not consider that parts of the feature vector is better shared among arms. For example, seeing a device as an arm, it must learn that location is an important feature for *each* arm. If the information that location matters is shared over all arms, it can propagates more quickly. Each selected arm that identifies location as an important factor would contribute to the significance of the shared feature 'location'. Therefore, even an arm that has not been played yet can access the shared information and use the knowledge about the locations importance in its own calculation. The CTR per patch for the partial approach in 3.8c clearly demonstrates its advantage in feedback-information over LinUCB disjoint and hybrid, which results in the overall very good CTR for this approach.

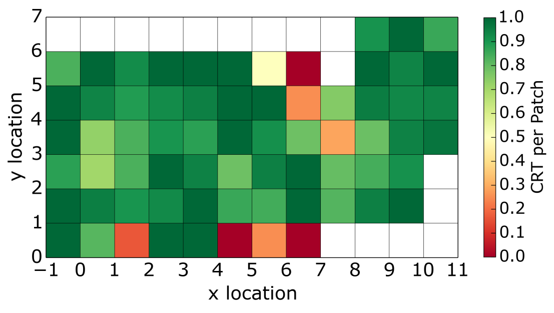
Procedure: Lastly, we wanted to investigate the spatial behaviour of the three models LinUCB hybrid, disjoint and partial. The experimental setup is the same as described above, however this time the information about the users location when receiving the recommendation is also evaluated. Hence the floor plan seen in figure 3.1 is split into 1×1 patches. For each patch we accumulate, the number of user visits, accepted and rejected recommendations over the 500 steps and 2 users.



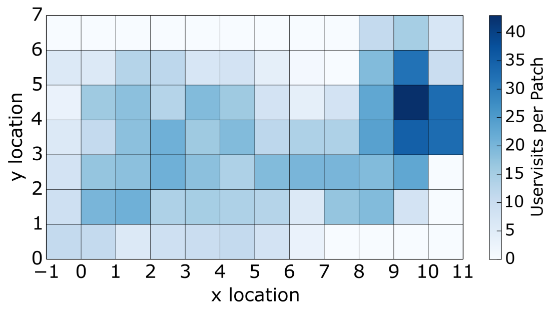
(a) LinUCB Disjoint Model, CTR per patch.



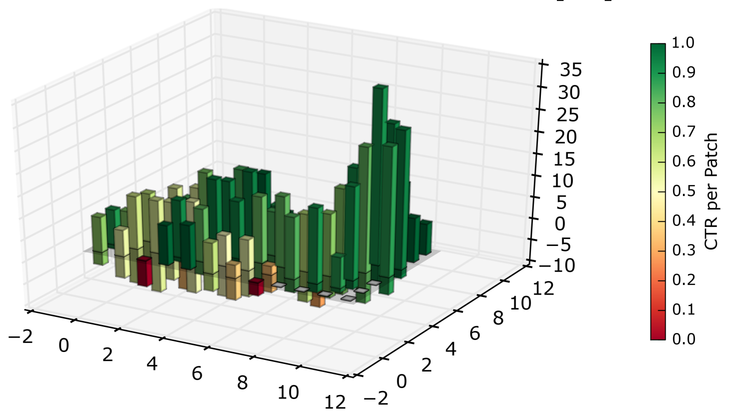
(b) LinUCB Hybrid Model, CTR per patch.



(c) LinUCB Partial Model, CTR per patch.



(d) User visits per patch for all three models.



(e) Combined view of CTR and visits per patch for LinUCB Disjoint.

Figure 3.8: Local Comparison of Different LinUCB Models

CTR per patch for LinUCB disjoint, hybrid and partial for $\alpha = 1.0$ along with the user visits per patch for this experiment. The data was gathered from two users with different profiles moving over the floor plan shown in figure 3.1, simulated for 500 time steps each. Afterwards, the floor plan was split into 1×1 patches and the number of visits, accepted and rejected recommendations was accumulated for each patch and the CTR calculated. Subfigure 3.8e gives a combined view of CTR and number of visits per patch. The bar height gives the total number of visits, while the part below 0 gives total of rejected and accepted (above 0) recommendations. The bar colour encodes the CTR for the bar in question.

Results: Subfigure 3.8d shows the number of visits per patch for the whole experiment. Recalling that the floor plan consist of two rooms, an strong accumulation of visits is visible in the right room. In subfigures 3.8a-3.8c the CTR is shown on a per patch basis and not averaged over the whole area, as given in figure 3.7. As measured by the quantity and colour intensity of green patches that indicate a high CTR, LinUCB partial dominates LinUCB hybrid and LinUCB disjoint as in the last experiment. Once again, the hybrid approach still performs better than the disjoint version. 3.8e gives a combined view for LinUCB disjoint on the CTR per patch and the number of visits, which are represented by the bars height. The bar height above 0 indicates the number of accepted, the height below 0 the number of rejected recommendations. Encoded in the bar colour is the CTR per patch. This representation visualizes a correlation between a high number of visits and a high CTR per patch.

Discussion: All our findings are in accordance to the reasons given for figure 3.7. The correlation between number of visits and CTR is also not surprising, given that LinUCB learns more about a certain user feature vector the more often this vector occurs, as it receives feedback for each recommendation. Hence more visits mean more feedback and more reliable recommendations.

3.4.4 Evaluation of LinUCB for Composite Service Recommendation

When we recall the discussion about IoT and WoT in 2.1.2, the WoT vision was not only about discovering but also about composing different services. So far we covered service discovery with our LinUCB approach coupled with the knowledge about available IoT services, which can be provided by Ambient Dynamix. However the mashup composition still has to be coordinated from client side. Our current approach does not care whether services are mutually compatible and could be used for a joint task, which is an important property for a mashup or *composite service*.

Procedure: To investigate whether LinUCB can be used to recommend multiple services in order to compose them, we designed the following experiment. Our scenario is a user moving through the environment in figure 3.1, who wants to use a display service and two lamps for a composite task. Introducing an additional constraint on the three services, we modify the reward function not only to test if all these services are accepted but also if all of them are in one room. The user rejects the composite service if one of these criteria is not met. Within the experiment, we used LinUCB hybrid to identify the top ranking display service and the two top ranking lamps, which are then forwarded to the simulation as composite service suggestion to the user. As in the evaluation for single service recommendation, the current CTR is logged after each recommendation attempt and plotted over the number of time steps.

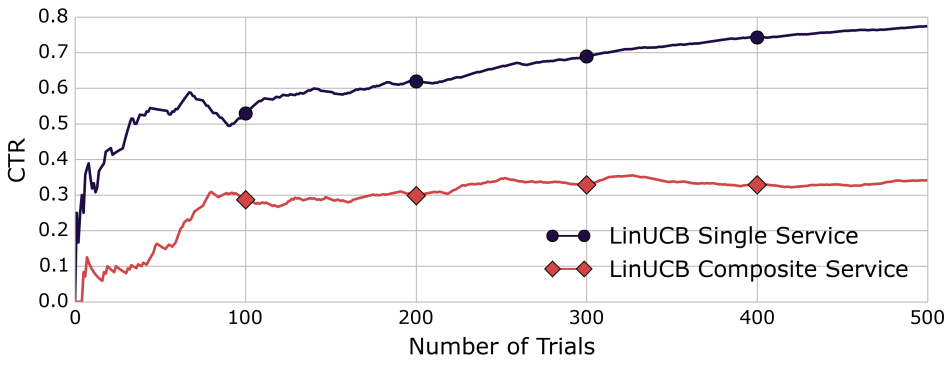


Figure 3.9: Single and Composite Service Recommendation

The CTR by LinUCB hybrid for single and composite service recommendation over 500 recommendations.

Results: Figure 3.9 shows the CTR over number of given recommendations for LinUCB hybrid used on single service (dark blue dot) and composite service (red square) recommendation. The plot clearly shows a dramatic loss of CTR for composite service recommendation. For composite service recommendation, the CTR is less than half of its value for the same algorithm used to recommend single services.

Discussion: We see two reasons for the low performance on composite service recommendation, of which one is a lack of information from a LinUCB perspective and the other is a structural error introduced by the reward function. The reward function error is based on the structure of our composite service recommendation. Even if we assume that all services are within one room, our recommendation approach has a certain probability do recommend a service that will be accepted. This probability is equal to the CTR. Given that we combine three services, the CTR for the composite service is (under the assumption that all are in the same room) $\text{CTR}^3_{\text{SingleService}}$.

The other problem concerns the information given to the algorithm In composite service recommendation, we suddenly assume a service's room to be a determining factor for the recommendation, which has not been the case before. Furthermore the algorithm itself has no clue about the concept of a room, as this is not encoded in any of the context features. In order to retrieve better recommendations for composite services the algorithm needs information about the factors that determine whether services can work together. However, in the current state this information is not directly accessible by only considering the IoT service's feature vectors. We can not do anything about the structural error if we do not change the reward function, however we can address the information shortage. In the next chapters we present a mechanism to provide the required information.

4 The ConComM Context Composition Machinery

The most effective way to cope with change is to help create it.

(L.W.Lynett)

Our main motivation to develop the ambient space simulation was the lack of data, which became an issue for the evaluation of personalized service recommendation systems. Previously we also explained why IoT service recommendation is about to become one of the crucial challenges within the next years. We demonstrated existing approaches such as LinUCB to perform well for single service recommendation, as seen in 2.2.4 and 3.4. However the results received for composite service recommendations are not even close to the good performance for single service recommendation tasks. This motivated our second contribution in this thesis, which is the development and evaluation of a framework called **Context Composition Machinery (ConComM)**, to improve the quality of composite service recommendations.

In the first section of this chapter we motivate such a framework and address the so far neglected question *why* one would be interested to receive a bundle of recommended services. The designed framework itself and an algorithmic change to one of the participating algorithms are presented in *ConComM Framework*. After this theoretic introduction of the ConComM Framework, the last section discusses selected implementation aspects, before the framework is evaluated in the next chapter.

4.1 The Service Composition Scenario

In the 3.4.4 it became obvious that neither the standard LinUCB approach nor its extensions are able to perform well for composite service recommendations. However, the question *why* someone would want recommendations for composite services was not addressed so far. Before we talk about the approach explored in this thesis, the next paragraph motivates the need for more than one recommendation at a time.

We assume a building that is filled with smart services. Each room is equipped with smart light bulbs; the curtains can be opened or closed using a smartphone or any other connectible service that is able to run some kind of curtain-control-app. Further, in some rooms one finds smart sound systems, smart monitors or even beamers. In addition to that, the temperature can be controlled individually for different places to ensure that it suits the needs of the person/persons working there.

A person in this building is planning to give a presentation in an arbitrary room later that day. In order to give a good presentation, this person wants a room with a beamer, a sound system, some lamps that can be dimmed and curtains that can be controlled by its phone. Now, it is important to keep in mind that there are many rooms in the building, each offering some of these services. We assume that the person does not know all the services and how they are spread over the rooms, hence the selection of a suitable set of services becomes a real challenge. As seen before, when we merely request some suitable beamer, some suitable sound system and so on this does not lead to the desired bundle of services that can be used to give a presentation. There is no guarantee that these services work together or are at least within the same room.

A framework like Ambient Dynamix can ensure that the required services are linked to the internet, which allows to compose them in the WoT sense as explained in 2.1.2 and 2.1.3. This implies that there already exists a way to make sure that services can work together in a technical sense. Nevertheless, the challenge of finding services that are physically within the same room and can therefore be used together still remains unsolved. In the first part of this section we discuss existing methods to find objects that 'work together' in some sense. After we explored these existing methods, we spent some thought on how composite service recommendations can be realized using the LinUCB approach that was evaluated in the last chapter. We conclude by outlining a framework that we use to provide additional data about service interoperability. This data can then be used by a LinUCB approach in order to give better recommendations for composite services.

4.1.1 Comparing Playlists Generation to Composite Services

As we motivated above, composite service recommendation needs a mechanism to identify whether or not services can be used together in a physical sense; in the previous example, the limiting factor is their localization within the same room. For tasks that require such information to provide good recommendations, the classical LinUCB is not sufficient any more. In a first attempt, we examine a scenario that is in a way similar to composite service recommendation. Here we explore the different approaches and discusses whether these techniques can be applied to our problem.

To generalize the composite service scenario, we look for situations that require different entities that share a certain type feature. The last requirement can be softened further. Instead of demanding on shared features, we can also require the features to be similar or more precisely close in some metric. For our “composite service in a room scenario”, the first definition could mean that each service in the composite service has to be within the same room. Here the service is the entity and the service’s room is its feature or context. Depending on the feature that is required to be the same or similar, the problem definition changes. Also, is important to keep in mind that a user’s satisfaction with a recommended entity will still depend on the contextual information given by the user’s and the entities’ properties. Just because a set of services is within the same room, this does not imply that a user would accept the entities because they can still mismatch the users personal taste.

If we consider a general scenario as formulated above, a well studied problem is the challenge of playlist generation. We understand it as the structurally most similar problem that received scientific attention in recent years. Here, the songs have to work well together and need to suit the listeners preferences. In a nutshell, a good playlist contains songs that have approximately the same style and each song goes along with the listeners taste. The challenges of playlist generation strategies are quite alike to those that are faced for composite service recommendation: The number of available tracks/services is very high, while information about entities is often neither structured nor complete. Further, the user feedback to allow personalized recommendations is limited [BJ13].

The work of Bonnin and Jannach [BJ13] gives a very good and detailed summary of proposed techniques for playlist generation, the interested reader should refer to their work for more details. Generally, there exist a several approaches for playlist generation, out of which the three most important are mentioned here. A first approach is the use of **Markov Chains**. A Markov Chain or Markov Process is a random process without memory, which means the next state is only influenced by the current state [Nor98]. Applied to playlists this implies that the probability distribution over the possible next songs is only influenced by the current song. Such an approach has been used for playlist generation by Mcfee and Lanckriet [ML11] and Chen et al. [CMTJ12]. Another idea is to mine for **frequent patterns** in order to find association rules. An association rule is an implication, where a set of items implies another set of items [AIS93]. As an example, take the implication that users that like The Rolling Stones and Nickelback also like Queen. The last notable approach are **neighbourhood recommenders**. Here a k-nearest-neighbour recommender is either applied the tracks [ML11] or to whole playlists. The latter approach with whole playlists was proposed by Hariri, Mobasher, and Burke [HMB12]. Using this procedure on songs, the next song will be one of the k contextually most similar songs to the current one.

All these methods have been applied to playlist generation, but there is one big difference between playlist generation and IoT composite service recommendation. Playlists are *sequential*, which makes them very suitable for Markov Chains and models that assume a sequential structure. But in the case of composite service recommendation, the user requests a *set* of services rather than a sequence. Further, most sequential solutions look for a successor in a local way, which means the current song (and maybe a few preceding songs) is taken into account when selecting the next one. In the composite case however, all these services have to work together. As an example, we consider the services A, B and C. In the sequential case, we could find that A works with B and B works with C. However, this does not necessarily imply that A would work with C, which is required in the composite service case for a request of three services. For composite services, a more global interaction guarantee is required.

Association rules as non-sequential approach suffer from the cold start problem, as they require existing data to mine for rules. In an IoT composite service scenario the context of a service changes often, as services are moved through the environment. This change of context must influence the services suitability to be part of a certain composite service, as a service can be good to interact with in one room but too far away or inaccessible in another location. Therefore, new association rules are required whenever a service changes its context. It is utopian to assume that after each change in context, new data about interactions with the service instantly exists. Hence no association rule exists for a service after a context change, which presents a repeatedly occurring version of the cold start problem.

This motivates the need for a new solution that differs in its structure from what has been done for playlist recommendation and similar problems. In the next section we present a framework that tackles the composite service challenge. We dedicate the remainder of of this work to the frameworks development and evaluation.

4.1.2 Problem Definition: Addressing LinUCB Limitations

For our problem comparison in the last subsection we only required a very rough problem specification to realize that playlist approaches are not well suited for service composition. To get a better impression of what is actually required for a set-recommendation, we refine our view on the problem before we present our solution.

First of all, it is important to define what kind of composite services we want to recommend. In our introductory presentation scenario, a the composite service did consist of single services that allow a person to give a presentation. The task itself requires all services to be within one room. The constraint to have all services united in the same room holds for most scenarios in a smart building context, since most actions are performed without a change of location. Therefore we consider it as very important to have services united in a certain location. However, we also need to consider other barriers such as different network affiliation that prevents service interaction. Summing up all these factors, splits an environment into what we call **interaction enabling containers**. *Within* such a container it can be guaranteed that a way of interaction among each pair of services exists, which we require for a composite service. A real world room can represent such a container, but a container can also be a room and a certain network. In this case all services belonging to this network and located within this room, are in the same container. In order to recommend composite services, we consider it important to firstly identify services that are in the same container and therefore are able to work together.

When we assume that information about a services affiliation to a container exists for each service, there are several possible ways to incorporate this knowledge in the recommendation process. A first and very simple approach is to select a suitable container first. Afterwards, a sequential service recommendation is performed using all services within the selected container. However, this procedure may seem a bit cumbersome, given that a first recommender system would have to be trained on the containers given the task and a second recommender needs to be trained on the services given the user. A more elegant solution that only requires one recommender system is to treat information about a services container as part of its contextual information. There is only one difference to the low performing LinUCB approach that has been used for the bandit algorithm evaluation on composite services in 3.4.4. The new bandit additionally receives the service container and user task as part of the contextual information. This approach could be used without rewriting the existing LinUCB approach, as it simply uses additional knowledge.

It seems plausible that information about a services interaction enabling container can improve a contextual recommendation system's performance for composite services. This leads to better formulation of the challenge that we tackle in the second part of this thesis: How can we obtain the correct interaction enabling container for a given service? Scaled down to the example scenario that was used in the introduction, this would yield the question: How can we obtain the correct room for a given service?

It is important to keep in mind that a room is only one way to characterize these containers. A services affiliation to a container can be influenced by many factors such as network affiliation or even the brand of a service.

4.1.3 Introducing the ConComM Idea

In the last subsection we motivated why it is important to determine a services container in order to get good composite service recommendations. This container information has to be gained from data that hopefully exist in such a scenario. For the remainder of this work, we make a very weak assumption concerning the type of existing data. Since our overall goal is to recommend services that can interact, it is valid to assume that users already interacted with different services at a time. Therefore, we assume data that captures interactions between two or more different services to exist.

Before we dive into the explanation how the framework works, we need to address an important issue of notation. So far we used the word 'service' whenever we talked about things that can interact and are assigned to a container. From now on, this is not sufficient any more. Until now this was acceptable because no details on how we obtain the services container were discussed. Now it is important to point out that the things we actually want to assign to containers are not services, but a service's context. Why is this distinction necessary? We take a smart lamp as a sample service. This lamp L_1 has a certain model and manufacturer, is located at a specific position p_A within room A and uses network N_1. The lamp itself is the service, while manufacturer, model, position and network are its context. When the lamp is moved into room B, the service is still the same but the context changed. Services that worked with the lamp when it still was in room A do not have to work with it any longer after it was moved to room B. On the other hand, a new smart lamp L_2 that has exactly the same manufacturer, model and network as the first lamp that is moved to room A and set to p_A, will work with all services that worked with L_1. Even though the services differ, the context is the same - which ideally leads to the same interactive behaviour. This is the reason why we are not interested in the container for a given service, but for a given *context*. The context itself already contains all necessary information about the service.

At this point we spent some thought on what we eventually want to achieve. It would be favourable to have a model that could predict the container for a given services context, even if the service was not part of any interaction yet. If we assume that every service context has exactly one correct container, this is a typical classification task and the field of machine learning offers various algorithms to solve it. Hence we formulate our goal to obtain a classifier that determines the container for a given context.

We specified that we have interaction data and want to end with a classifier, as described above. The next open question is how to transform this data about previous interactions into a working classifier. We take a reverse approach and start with the main requirement to train a classifier, which is training data. Hence for each data point, which is a service context, we need the correct container or class, what is sometimes called *ground truth*. But in general we can not access this information directly, as we have no information

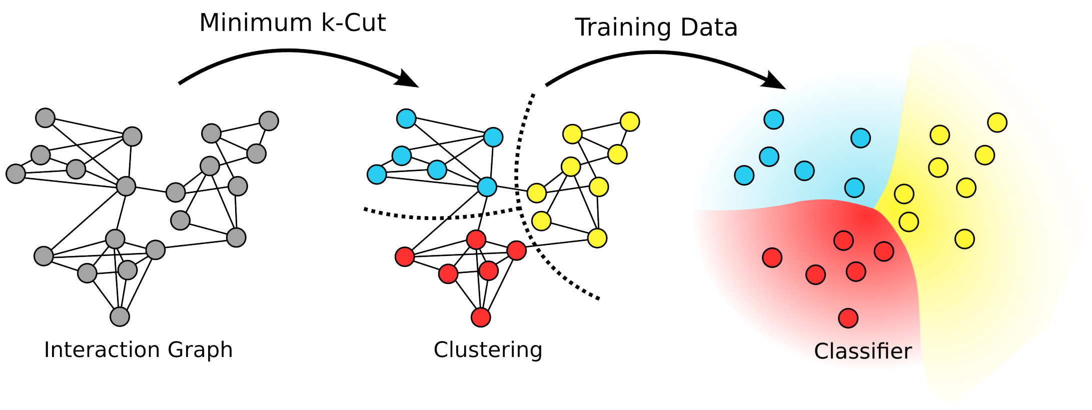


Figure 4.1: ConComM Framework Illustration

The general ConComM workflow. Starting with an interaction graph, we apply a graph cut to obtain a clustering. This clustering is then used to train a classifier that can output a class even for new service contexts.

about what fatures shape these containers. The only data we can access is interaction data, which can be represented as a graph. Its nodes are given by service contexts and each interaction is an edge with a specific weight. Under the assumption that services within the same room are more often used together than services in different rooms, the graph should represent this fact. Therefore a graph cut that separates service contexts that share a container should be more expensive than a cut through context-pairs that belong to separate containers. In order to build a ground truth from such a graph, the easiest way is to assume a number of rooms and then cut the graph into this number of subgraphs, using a minimum cut.

Even though presented in reverse, the above explained procedure outlines the framework we designed to obtain a classifier that can give the container for a given service context. In a nutshell, our framework works in two stages. In the first stage we use a minimum k cut to split the graph into rooms and to obtain training data; in the second stage we train a classifier with this data. Figure 4.1 illustrates the framework. Because the framework composes the contextual information of services in order to obtain a classifier, we call it Context Composition Machinery, abbreviated to ConComM.

What makes this framework remarkable is that it requires no information about the underlying container structure, but tries to learn the containers only from an interaction graph. If we do not learn what shapes the containers, we require the knowledge about impacting factors explicitely. However this knowledge is highly ambiguous, as the container structure can be influenced by many factors including not only technical but also psychological ones. The most likely reason for the container structure is that the task itself requires services to be within the same location, which they not necessarily are. But we must also consider more complex dependencies, which are not limited to rooms or networks, that prevent services from being used in a composite setting. Hence simply using accessible information such as a services location may not be sufficient. Therefore we designed ConComM to obtain the container structure without making any assumptions about what shapes it. ConComM is more than a system that merely trains a classifier to output a container when presented a services context, as it has to identify the container structure on its own.

4.2 ConComM Framework

After we presented the general ConComM idea in the last chapter, this section finally explores ConComM in detail. In the first subsection, we refine the basic idea behind ConComM and complete it with algorithms presented previously. The second subsection goes a step further and showcases a modification we introduced to SPLIT in order to inherently anchor the min-max clustering principle within ConComM. After developing the basic ConComM idea, we show how to prepare the initial interaction data to process it with ConComM. Hence we discuss the representation choices when we convert interaction data into a graph. As the graph formed by the initial data is not necessarily connected, we also analyse the chosen cut algorithm's behaviour for unconnected graphs.

4.2.1 Algorithms for ConComM

Recalling the two stages of ConComM, graph cut and classification, the focus of our work is slightly shifted towards the first stage. Given the fact that the classification can only be as good as the presented ground truth, it is justifiable to pay more attention to the graph cut that leads to a clustering.

For the first stage, the interaction graph is cut into a predefined number of subgraphs using a 2-cut algorithm and the k-cut procedure SPLIT, mainly for two reasons. Firstly because SPLIT requires less cut calculations than EFFICIENT. The second reason is that most 2-cut algorithms we discussed in 2.3 find the lightest cut for a graph, and not the lightest cut that separates two nodes. SPLIT only requires a procedure that cuts a graph, while EFFICIENT needs an algorithm to find the lightest cut separating two nodes. Hence most 2-cut algorithms integrate more smoothly with SPLIT than with EFFICIENT. Using the algorithm of Goldschmidt and Hochbaum [GH88] is not an option since it still has a polynomial runtime. Because SPLIT only gives a procedure, we also need to choose appropriate 2-cut algorithms to obtain a full k-cut procedure.

For ConComM, two 2-cut algorithms were implemented and compared. The first one is the well known and exact Stoer-Wanger algorithm; the second is Min-Max cut, which encourages more balanced cuts. Generally we are good with an approximate solution, as the interaction graph is also inaccurate to some extent. This is because the interaction graph itself is build based on sample interactions that are not necessarily perfect samples. Hence Stoer-Wagner is considered as benchmark, since it finds the precise minimum cut.

In our case, the containers we try to identify by cutting the graph are most likely comparable in size. Min-Max cut is designed to find balanced cuts that satisfy the min-max clustering principle. The fact that it avoids skew cuts makes it seem most suitable for cutting a graph formed by ‘real world’ data, where cutting outliers would potentially give the lightest cut, but not the anticipated ground truth. Hence Min-Max cut is our most promising candidate for the 2-cut algorithm used for the k-cut procedure in ConComM’s first stage. To better integrate with the aim of the Min-Max clustering principle, we developed a modified version of SPLIT which is presented in 4.2.2.

In ConComM’s second stage we train a classifier on the sample contexts over the newly gained ground truth. We decided for multi-class logistic regression as classifier mainly for the following reason: Logistic regression not only returns the class for a given feature vector, but the outcome can be interpreted as probability distribution over the possible classes. This comes very handy when the probability distribution can be added to the contextual information of a service that is later used as input context for LinUCB. In the end, one has a classifier that takes an arbitrary context and returns the contexts probability to belong to each container. This probability vector can then be used as a context extension for LinUCB or any other contextual Bandit.

4.2.2 Improving SPLIT

Our ConComM version presented in the last section is a carefully designed combination of algorithms. In this section we present a modification, which makes the min-max clustering principle an inherent part of SPLIT and hence ConComM. Recalling the min-max clustering principle, it is used to obtain clusters that are internally connected, but not well connected among each. ConComM is supposed to find services that work well together, so cutting through such a cluster of services is definitely not desired. Therefore the cut weight separating clusters should be small. On the other hand, if we split only a few services this is not favourable either as it leaves clusters that are too big. These clusters would not represent containers with services that work well together, as they include too many services. Hence finding only the lightest cut does not resemble the well motivated min-max principle.

As the min-max principle seems very appropriate to represent interaction enabling containers, we tried to incorporate the idea not only in the clustering algorithm but within ConComM itself. In a first step, we identify the parts of ConComM that would potentially conflict with the min-max principle. ConComMs second part, the classification, is only influenced by the output of the first part. Therefore the min-max principle is only of use during the clustering. A clustering algorithm such as Min-Max cut already uses this principle. Hence the only part we have not considered by now, is the algorithm that is used to obtain a k-cut instead of a 2-cut. In ConComM, we choose SPLIT for this task.

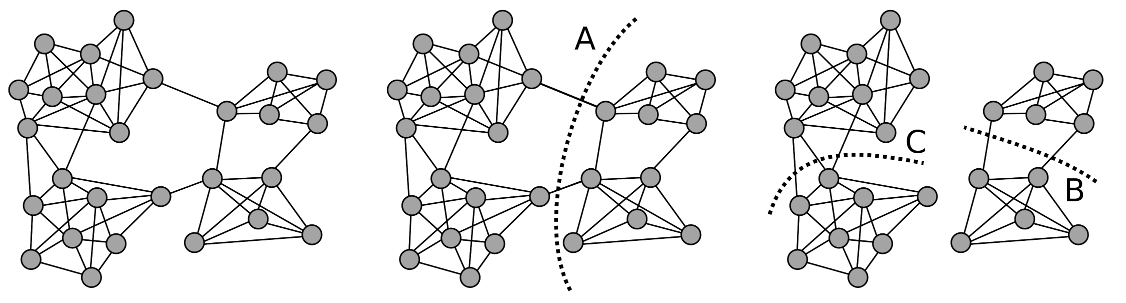
The SPLIT procedure itself is very simple. Given a initial clustering that can consist of one or several partitions, the following procedure is repeated until the clustering contains the desired number of k clusters [SV95]:

1. calculate a minimum 2-cut for all clusters in the clustering
2. select the cluster whose minimum cut is lightest
3. remove the cluster from the clustering and add its two subclusters that were obtained by the minimum 2 cut.

Here, 2 is the crucial step. If we select the minimum cut this is not in accordance with the min-max principle. For our current procedure, the following scenario is not unlikely: After splitting the graph for the first time, we obtain a smaller and a bigger cluster. As the smaller cluster contains less nodes and hence has fewer connections, a cut through the smaller cluster is very likely to be lighter than a cut that separates the bigger cluster. Following this logic, the current procedure has the tendency to get stuck in cutting a small and not so well interconnected cluster into tiny clusters. Their cut weights will always be smaller than a cut through the big cluster, that would potentially resemble the reality better.

After we identified this problem, we attempt to adapt SPLIT by not only considering the absolute weight of a cut. The min-max principle demands on us to take the interconnectivity of a graph into account. For a very interconnected graph, the cut weight per node in the graph would be high. For a graph that is not very connected at the cut position, the cut weight per node should be small. To avoid cuts that chose a very light cut but only separate a single node, we choose the following alternative criterion for 2. Instead of taking the minimum cut weight, our modified SPLIT selects the cut that minimizes

$$\text{weight}_{rel} = \frac{\text{cut weight}}{(\text{number of nodes in smaller cluster})^2}. \tag{4.1}$$



(a) Sample Graph **(b)** First Cut (**A**) for SPLIT and SPLIT$_{rel}$ **(c)** Second Cut for SPLIT (**B**) and SPLIT$_{rel}$ (**C**)

Figure 4.2: Graph Cut with SPLIT and SPLIT$_{rel}$

A sample graph cut for SPLIT and SPLIT$_{rel}$. Figure 4.2a shows the initial setup with one graph. In the first step 4.2b, for both procedures the lightest cut is **A** and no alternatives exist. In the second step 4.2c we have two subgraphs, so the procedure has to choose which subcluster to cut. The conventional weight for the cuts **B** and **C** is 2 and 3, hence SPLIT chooses the lighter cut **B**. This results in a big subgraph with 14 nodes and two small ones with 5 nodes each. The SPLIT$_{rel}$ weights calculated with 4.1 for **B** and **C** are 0.08 and 0.05. Therefore SPLIT$_{rel}$ chooses **C**, which gives similarly sized subgraphs with 8, 9 and 10 nodes.

As weight$_{rel}$ gives a relative cut weight per nodes, we call this modified SPLIT version SPLIT$_{rel}$. The procedures for SPLIT and SPLIT$_{rel}$ only differ in step 2, where SPLIT selects the cut according to its minimum weight while SPLIT$_{rel}$ considers the relative weight given in 4.1. We give a sample scenario to illustrate the structural difference between SPLIT and our SPLIT$_{rel}$ in figure 4.2.

To sum up our approach, SPLIT$_{rel}$ is the result of our efforts to incorporate the min-max principle in ConComM. With SPLIT$_{rel}$ we aim to produce more balanced cuts, not only on a 2-cut basis as Min-Max cut does, but also for k-cuts obtained by the SPLIT procedure. In the evaluation chapter 5, our min-max principle optimized version of a k-cut using Min-Max cut and SPLIT$_{rel}$ is benchmarked against Min-Max cut with SPLIT and Stoer Wagner with SPLIT.

4.2.3 Generating a Graph Representation

Our previous subsections are sufficient to give a full description of our framework. However, our considerations usually started with the assumption of a graph that would represent the interactions between services. This subsection will bridge the gap between our preliminary assumption of interaction data and the graph that most descriptions were based on. Thus, we describe what choices can be made when interaction data is transformed into a graph.

We represent an interaction between services as a set of their contexts. Therefore, each interaction I can be written as $I = \{c_1, \ldots, c_n\}$, with $c_1, \ldots, c_n$ being the contexts of the participating services. In the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, each context is represented by a vertex $v \in \mathcal{V}$. Since a graph can be fully described by its laplacian matrix, each context is also represented by a specific row and column index in the laplacian.

When we add an interaction to the graph, the most natural way to do this is to increase the edge weight between all interacting contexts by 1. This gives a procedure to add interactions, but so far we did not address how we initialized the graph. There exist two principal ways that strongly influence the graphs properties. The first way is to assign no edge weight between the nodes as we initialize the graph. Doing so results in a completely unconnected graph, as no two contexts are connected by an edge, assuming that an edge with weight 0 is not existing. Afterwards, we add known interaction data, which connects certain nodes with higher edge weight than 0. The second way is to initially assign a certain edge weight to any connection in the graph. This does not change the information stored in the graph, as the edge weights are equal to a graph in which all eges are initialized to zero and we just introduced an uniform offset. However, in this case the graph is connected.

For ConComM, we assume an unconnected initial graph as this seemed like a more natural way, than enforcing an edge weight lift by an arbitrary value for all connections.

A point we will not cover any further, but that seems worth mentioning with later applications in mind, is customer feedback. So far we only assumed that interactions were monitored passively. However, at some point there might be feedback for certain interactions, judging whether they are favourable or not. If the feedback for a certain interaction is positive, it would be worth increasing the edge weight between the contexts by more than 1, as this interaction was actively rated as good. The other case is negative feedback. If a certain interaction does not work at all, negative feedback could be given. In this case it would be good to decrease the weights between the interacting contexts instead of ignoring this information. However, for this action we need to check that the edge weight does not decrease below its value on initialization, as graphs with negative weights might cause problem for some cut algorithms.

4.2.4 Unconnected Graphs

A problem that can occur with natural datasets is an unconnected interaction graph, given that the graph is initialized with zero edges between context. The min-cut algorithms Stoer-Wagner and Min-Max cut do not explicitly demand a connected graph, nevertheless unconnected graphs presented a challenge during the implementation. To circumvent problems we need to consider how the algorithms behave when presented an unconnected graph.

Due to its construction, Stoer Wagner is very well behaved for unconnected graphs. For simplicity, we assume a scenario with two unconnected graphs. Our argumentation also holds for scenarios with more than two graphs. The only difference is that the distribution of graph parts over the two output clusters will mainly depend on the implementation.

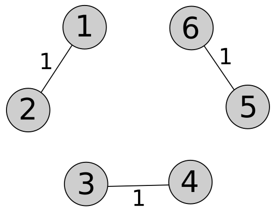
In every MinimumCutPhase, we add the most tightly connected vertex. Hence the MinimumCutPhase will start to add all vertices that are directly connected to the start vertex. After the start vertex consumed all vertices that are within its graph part, it will only have edges with the weight 0 left. Adding one of the vertices with zero edge weight 'bridges the gap' between the two unconnected graph parts. Now, that a vertex connected to the second graph part is added to the start vertex, the next most tightly connected vertex will come from the second graph. The cut that is returned by MinimumCutPhase is either 0, if the second graph did only consist of one vertex, or has some weight because it cuts through the second graph. If the second graph has more than one vertex, the last added and therefore merged vertices were both from the second graph. The MinimumCutPhase is repeated until all vertices in the second graph are merged. Now, the second graph only consists of 1 vertex, which results in a cut weight of 0 in the next MinimumCutPhase. No cut through the first or the second graph part can be smaller than 0, therefore MinimumCut must select the cut separating the two graph parts as cut with minimal weight.

This explains why Stoer-Wagner works out-of-the-box for implementations that allow to merge vertices with connection weight 0. Min-Max cut on the other hand is not as well behaved. Because the laplacian of a unconnected graph looses many of its useful properties, the Fielder vector does not provide a linear search order of the expected form.

For connected graphs, the preliminary graph cut can be obtained by forming a cluster from all vertices whose Fielder entry is ≥ 0 and a second cluster from those with Fielder < 0. For unconnected graphs, the Fielder vector will only indicate *one* cluster with entries that are either strictly greater or smaller than 0. Any other entry is 0. We illustrate the case in table 4.1.

If we partition the the sample graph using the above described rule, this leads to the obviously undesired partitions $\{1, 3, 4, 5, 6\}$ and $\{2\}$. A working, but not optimal solution to overcome this behaviour is the following: Instead of taking zero as the threshold, we can also consider it as a third class. If the number of entries in the zero-class exceeds a certain number (for example two or three), we treat all members of the zero-class as

Table 4.1: Fielder Vector for Unconnected Graph

Graph Illustration	Laplacian	Fielder Vector
	$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & 1 \end{pmatrix}$	$\begin{pmatrix} -0.7071 \\ 0.7071 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

one partition, and the second partition is formed from entries that are less or greater than zero.

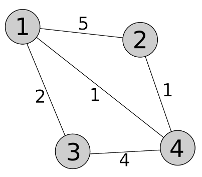
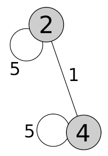
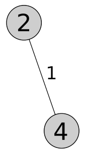
One of the easiest options is just to avoid disconnected graphs. If we initialize every edge of the graph with a small value as described in the last subsection, this produces a connected graph. However, our work showed that using the above described procedure to treat zero entries an own graph if their number exceeded 3, did not perform any different to initializing the graph with a small edge weight. Therefore we decided to go with an initial edge weight of 0, as this occurred to us as the most natural representation.

4.3 Implementation Challenges

In the following section we talk about implementation changes we faced during the ConComM implementation with Java 8. This section is not meant to be an implementation guide, as this would be out of the scope of this work. We rather attempt to provide insight in some interesting aspects that we faced during the implementation. We ask those who are interested in the implementation details to refer to the source code.

In the first subsection we present our space efficient representation for graphs that stores the graphs full laplacian once for the whole simulation. SPLIT is an important part of ConComM and min-cut computations can be costly, therefore we present an implementation that minimizes the number min-cut computations in the second subsection. Lastly, we give a few practical notes and detail on the classifier implementation.

Table 4.2: Correct Subgraph Representation with the Laplacian

Matrix Name	Indices	Matrix	Illustration
Full Laplacian	$\mathcal{L} = \{l_1, l_2, l_3, l_4\}$	$\begin{pmatrix} 8 & -5 & -2 & -1 \\ -5 & 6 & 0 & -1 \\ -2 & 0 & 6 & -4 \\ -1 & -1 & -4 & 6 \end{pmatrix}$	
Laplacian Cropped	$\mathcal{I} = \{l_2, l_4\}$	$\begin{pmatrix} 6 & -1 \\ -1 & 6 \end{pmatrix}$	
Laplacian Correct	$\mathcal{I} = \{l_2, l_4\}$	$\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$	

The issue with cropping the full laplacian in order to obtain the subgraphs laplacian. The mere selection of laplacian entries that are associated with the subgraphs vertices is not sufficient, as the degree still represents the connection with the full graph. This leads to phantom self-loops in the graph. To get the correct laplacian for a subgraph, the degree has to be adjusted to the new number of vertices.

4.3.1 A Space Efficient Representation for Graphs

As the interaction graphs laplacian matrix L contains all information about the graph and is required to compute the Fielder vector, we chose this representation to store all information about the interaction graph. During construction, we assign each row or column index uniquely to a vertex, which equals a context $c_1, \ldots, c_n$ in the interaction graph. We call the laplacians indices *Global Indices* $\mathcal{L} = \bigcup_{i=1}^{n}\{l_i\}$ and we need to store the bijective globalIndex-context-mapping $l_i \to c_i$. An entry of the Laplacian matrix with row l_i and column l_j will be denoted as $\mathrm{L}(l_i, l_j)$ or $\mathrm{L}_{i,j}$ for short.

The weight of each connection between two contexts is already stored in the laplacian and will not change if we only consider a subgraph. Hence, it seems unwise to duplicate this information for each subgraph by saving a smaller version of the full laplacian. In our implementation, a graph $\hat{\mathcal{G}}$ is not defined as set of vertices and edges, but only as set of vertex-indices $\mathcal{I} \subset \mathcal{L}$ that is a subset of the global indices contained in the laplacian. However it is not sufficient to store only a subset of indices and construct the laplacian matrix for the subgraph by taking the laplacian entries L_{ij} whenever $l_i, l_j \in \mathcal{I}$ holds. This results in the cropped Laplacian $(\mathrm{L}_{i,j})$ with $l_i, l_j \in \mathcal{I}$.

The problem that arises with cropping is illustrated in table 4.2. The second row shows the subgraph we obtained by cropping the laplacian with the above described technique. Cropping the relevant entries leaves the correct connections between two different nodes, but the diagonal still shows the vertex-degree for the full graph. The sum over a row or a column in the laplacian is the weight of a nodes self-loop, as self loops increase the degree but do not appear in the adjacency matrix. When we crop the laplacian we therefore produce phantom self-loops, that do not exist in the correct subgraph. To circumvent this problem, our the subgraph definition not only requires the index set $\mathcal{I}$ but also the new degrees $d(l_i, \hat{\mathcal{G}})$ for each vertex $l_i \in \mathcal{I}$. Calculating the new degree for l_i is easy, as it is the negative sum over a column or row in the subgraphs cropped laplacian without the diagonal entry:

$$d(l_i, \hat{\mathcal{G}}) = \sum_{l_j \in \mathcal{I}, l_i \neq l_j} -L_{i,j} = \sum_{l_j \in \mathcal{I}, l_i \neq l_j} -L_{j,i}.$$

To take the sum over row *or* column is only possible because the graph is undirected and adjacency as well as laplacian matrix are symmetric.

Now, our graph can be defined as $\hat{\mathcal{G}} = (\mathcal{I}, \mathcal{D}(\hat{\mathcal{G}}))$ with the index or vertex set $\mathcal{I}$ and the associated degrees for all contained indices $\mathcal{D}(\hat{\mathcal{G}}) = \bigcup_{l_i \in \mathcal{I}} \{d(l_i, \hat{\mathcal{G}})\}$. Our construction saves a lot of memory. To store the full laplacian for a subgraph with n indices would require n^2 values, while storing our new subgraph construction only requires $2n$ values.

4.3.2 Implementing SPLIT

Within ConComM, SPLIT [SV95] is used to cut the initial interaction graph in k subgraphs. To achieve this, it repeatedly calculates min-cuts for all current subgraphs and only performs the lightest cut, until the desired number of cuts is reached. In our implementation we only calculate the minimal required number of min-cuts.

In order to minimize the number of cut calculations, we use two sets that contain graphs and a tree structure to hold the graphs. To distinguish the sets, they are given the names *CurrentClusters* (CC) and *UnexpandedClusters* (UC). Each graph can have none or two subgraphs. Also, for each graph we store the cut weight between the subgraphs. If the graph has no subgraphs, we define the cut weigh to be infinite.

The SPLIT implementation 4.1 takes the number of desired clusters k and an initial graph $\mathcal{G}$. It returns a set of graphs CC. Our SPLIT implementation calculates min-cuts on a graph only when they are needed, which is the case if the cut through the graph could be the lightest. Because we store a calculated cut with its cut weight and the resulting subgraphs, we need to calculate the min-cut for a graph only once. Graphs that are

Algorithmus 4.1 SPLIT with Minimal Number of Cut-Calculations

```
procedure SPLIT($k, \mathcal{G}$)
    CC $\leftarrow \mathcal{G}$
    UC $\leftarrow \mathcal{G}$
    while $|\text{CC}| < k$ do
        for Graph G in UC do
            GETMINCUT(G)
            set the graphs resulting from the min-cut as subgraphs for G
            remove G from UC
        end for
        smallestCut $\leftarrow \infty$
        relevantGraph $\leftarrow \emptyset$
        for Graph G in CC do
            if Subgraph cut weight for G smaller than smallestCut then
                update smallestCut
                relevantGraph $\leftarrow$ G
            end if
        end for
        Add relevantGraphs subgraphs to CC
        Add relevantGraphs subgraphs to UC
        remove relevantGraphs from CC
    end while
    return CC
end procedure
```

currently part of the minimum k-cut are stored in CC. UC holds graphs that have not yet been split using a min-cut.

Our algorithm starts by putting the input graph in the current and unexpanded clusters. While the desired number of clusters is not reached, we perform the following steps: Firstly, all graphs in unexpandedClusters are expanded and removed from the unexpandedClusters. We save the cut-weights set the sub-graphs as children of the cut graphs. In the first run this only computes the min-cut for the original graph. The unexpandedClusters should be empty afterwards. This assures that no currentCluster is unexpanded. In a second step, we identify the current cluster whose cut results in a minimal cut weight. We then remove this graph from the currentClusters, while its children are added to currentClusters and unexpandedClusters. Hence, the number of currentClusters increased by one. Before we choose the minimal cut the next time, the newly added subgraphs are already cut, because unexpandedClusters are expanded at the beginning of run.

Our SPLIT version computes the minimal number of min-cuts because it only computes a min-cut when the result is required to identify the next graph to split.

4.3.3 Classification Implemented

For the final classification, we use multi class logistic regression. As we know the equations that have to be solved in order to get the optimal β_c vectors that describe the class functions, the basic implementation is comes down to implementing the formulas.

However we noticed a numerical issue that can lead to wrong results. When we introduced logistic regression in 2.3.5, the class specific probability that is required to compute the neg-log-likelihood 2.6 was defined as

$$p(y_i|x_i) = \frac{e^{f(x_i,y_i)}}{\sum_{\hat{y}\in\mathcal{Y}} e^{f(x_i,\hat{y})}} \quad \text{with} \quad f(x_i,y) = x_i^T\beta.$$

As both, nominator and denominator can potentially be very large, the result is numerically not stable. A simple division of nominator and denominator by the nominator, which equals multiplying by one, gives the following and more stable version

$$p(y_i|x_i) = \frac{1}{\sum_{\hat{y}\in\mathcal{Y}} \frac{e^{f(x_i,\hat{y})}}{e^{f(x_i,y_i)}}} = \frac{1}{1+\sum_{\hat{y}\neq y_i,\hat{y}\in\mathcal{Y}} \frac{e^{f(x_i,\hat{y})}}{e^{f(x_i,y_i)}}}.$$

Here we put the 'dangerous' fraction of two exponential functions in the denominator. The 1 in the denominator that originates from $\frac{e^{f(x_i,y_i)}}{e^{f(x_i,y_i)}}$ dominates, if the fraction approaches 0, giving a probability of 1. If the fraction dominates, the probability approaches 0. In our implementation we used this formula, which gives a numerically stable scheme.

As the β computation can not be performed analytically, we implemented Newtons method to obtain the result. In our implementation, we abort the Newton method and return the result if one of the two conditions is fulfilled:

- The maximum elementwise absolute difference between the previous and the new β vector is smaller than 10^{-7}
- A total number of 10000 newton steps is exceeded

As there is exists no rule how to choose these values, only the application can tell if these values are chosen appropriate. For all our test runs this combination gave good results.

Concerning the feature vectors, we implemented linear and quadratic features for ConComM. Linear features take the context of a service and return a vector that contains 1 as first entry as well as an entry for every context component. Quadratic features take the vector obtained by the linear features and add entries that contain the product of every context component combination.

Given the context $C = (c_1, c_2, c_3)^T$, its linear feature vector is $x_l = (1, c_1, c_2, c_3)^T$ and its quadratic feature vector is $x_q = (1, c_1, c_2, c_3, c_1c_2, c_1c_3, c_2c_3, c_1^2, c_2^2, c_3^2)^T$. By multiplying these feature vectors with the weights beta, we can therefore describe every linear or quadratic function in a 3-dimensional space. This concludes the remarks on our ConComM implementation.

In this chapter we motivated the need for a framework like ConComM to improve the performance of contextual bandit algorithms for composite service recommendation. By adapting SPLIT to act in accordance with the min-max principle, we aim to achieve an improved overall performance of ConComM when it comes to identifying the container structure. After our brief discussion about interesting aspects during ConComMs implementation, the next chapter evaluates how different design choices influence the frameworks performance.

5 Evaluation

I have not failed. I've just found 10,000 ways that won't work.

(Thomas A. Edison)

Within this chapter, we test the performance of our ConComM framework with different experiments. One of the key questions we try to answer is whether k-cut procedures using SPLIT$_{rel}$ are able to produce more accurate cuts, and whether the framework as a whole can predict the container of a new service correctly. Lastly it is of importance to find out whether the ConComM information about a services container does improve the quality of composite service recommendations. All three questions are addressed in 5.3 *ConComM Evaluated* and 5.4 *Closing the Circle: LinUCB for Composite Service Recommendation II* respectively.

In preparation for the evaluation, in the first section we describe how the data, on top of which ConComM is evaluated, is gathered utilizing the ambient space simulation we developed earlier. To quantify the quality of ConComM results, we describe how we measured the quality of our outcomes. In the last section we investigate the question whether the additional information about a services container provided through ComComM enables LinUCB to give better recommendations for composite services.

5.1 Ambient Space Simulation Revisited - Gathering Data

In order to evaluate ConComM, the first thing we require is *Interaction Data*. One interaction I is represented by a set of contexts $I = \{c_1, \ldots, c_n\}$ that belong to the services participating in the specific action. Hence interaction data is a set of such interactions $\{I_1, \ldots, I_k\}$. It is important to note that a single interaction is equal to a composite service, as a composite service is nothing but a set of contexts belonging to the services that are used together. Therefore interaction data is also a set of composite services.

For our evaluation, the lack of real word data was a problem as we did not find data sets for the usability testing with sufficiently high IoT service density. Therefore we utilized the ambient space simulation to evaluate ConComM.

The first subsection takes a closer look at the process of composing multiple services into a composite service. We explore how different design choices during the service assembly influence the interaction graph and describe the scheme we used to retrieve interactions from the simulation. As the presented strategy to obtain composite services depends on the user who 'composes' the services, we focus on the design of our user variables in the second subsection.

5.1.1 Composing Services

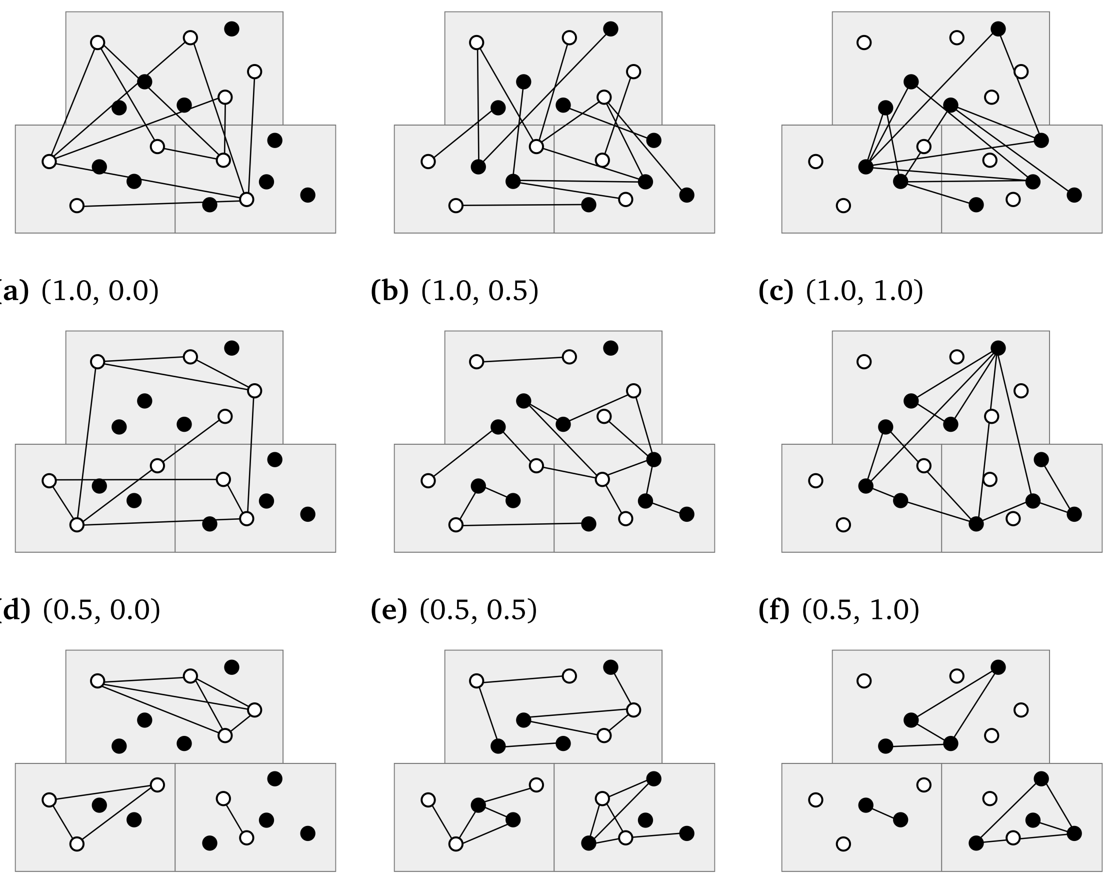
Certainly, the easiest way to obtain a composite service is to select a number of services that are accepted by a specific user. As the components obtained by the above outlined procedure are not even guaranteed to be within one container, the process of composing services has to differ from the just described. Hence we need to take a service's container into account.

Given that it is not likely that *only* services within one container are used together, we introduced a probability that is called **incompatible container probability (iCP)**. With the incompatible container probability we provide a measure, how often users use services in different containers together. An iCP of 0.0 therefore does not allow a composite service to consists of single services that are in different containers. A high iCP on the other hand encourages composite services whose components are scattered over many containers. Thinking in terms of an interaction graph, which is built from composite services that were gathered using a certain iCP, the iCP does significantly determine the graphs properties: An iCP of 0 would therefore result in a disconnected graph since there is no interaction among containers. The full disconnection is only reached as long as the containers are non-overlapping, which is the case for a classification problem. As the iCP increases, the number of connections for services belonging to different containers increases. An iCP of 1.0 makes it impossible to extract information about the container structure by only considering the weighted connections between them, as a composite service is not allowed to contain services within the same container.

To make the service composition more realistic, we also introduced the **incompatible service probability (iSP)**. When we recall the example given in *The Service Composition Scenario*, the user who wanted to give a presentation required a beamer, a sound system, different lamps and so forth. In such scenario, the user would more likely accept single devices that does not completely fit its personal taste for the sake of having them all in the same room. Hence selecting a composite service makes it more likely for user to also accept single services that do not match completely with its preferences. We model this factor by the iSP. As in the previous case, an iSP of 0.0 only composes a composite service from services that are accepted by a user, given the outcome of the users reward function. As the iSP approaches 1.0, a composite service consists of more services that are not be accepted by the user if they were recommended as single services. Different sample interaction graphs obtained by certain combinations of iCP and iSP values are presented in figure 5.1.

As we have no knowledge about the iCP/iSP for a realistic scenario, different probability tuple were assumed to obtain the ConComM evaluation interactions. However, there are probability for iCPs/iSPs that are more likely to represent composite services composed by real users. For example, iCP values greater than 0.5 are highly unlikely, as every second service within a composite service using this probability would be in a different container. Smaller values in the range $[0, 0.2]$ are more likely, as composite services composed following this probability range would mainly be within one container but also accept a few services in different containers. For iSP values the limits are much harder to guess, as the users willingness to accept less suitable services will mainly depend on the services importance within the composite service and the users overall willingness to compromise. In our presentation example, it would be more likely that the user accepts a less suitable lamp than a hopelessly outdated beamer.

After we defined the two key influences that determine the selection of services for a composite service or interaction, we describe how the interaction data was gathered using the simulation. Our procedure to obtain a single interaction is straight forward and only needs the required number of interacting services n, and a user who composes the services as well as the iCP and iSP that are used to assemble the services. In a first step, we draw a service uniformly at random among all possible services that fulfil the iSP. To do that, we draw a random number in $[0, 1]$ and if the value is bigger than the iSP value, the service has to be accepted by the user in order to join the composite service. If the random number is smaller than the iSP value, the service has to be rejected to be part of the composite service. Afterwards, we draw services until a service that is accepted, or respectively rejected, is found. The obtained service serves as seed, since the incompatible container probability requires a container to compare the other containers to. Until the required number of services n is obtained, we repeat the following procedure: To determine if the next service has to be accepted and if it needs to be within the same container, two new random numbers in $[0, 1]$ are drawn. Assigning the choices works as described for the first service. Now, that we announced the requirements (acceptable/rejected and same/different container), a service is drawn uniformly at random from the remaining services. Here *remaining* means all services that are not part of the current composite service. Then we evaluate the service, which means its room and the value of the user's reward function are determined and added to the composite service, if the evaluation results match the requirements. If the service did not match, a new service is drawn until a suitable service is found or a maximum number of steps is reached.



(a) (1.0, 0.0) **(b)** (1.0, 0.5) **(c)** (1.0, 1.0)

(d) (0.5, 0.0) **(e)** (0.5, 0.5) **(f)** (0.5, 1.0)

(g) (0.0, 0.0) **(h)** (0.0, 0.5) **(i)** (0.0, 1.0)

Figure 5.1: Interaction Graph Illustration for (iCP, iSP) Combinations

Illustration of three containers (gray) containing services that would be accepted (white nodes) and rejected (black nodes) by a user. The graphs represent *accumulated* interactions that would be allowed according the corresponding (iCP, iSP) probability tuple. Therefore the graph does not represent a single interaction but rather accumulated interactions that were allowed by the probability tuple. When iCP $= 0.0$, the interactions have to remain within a single container as in the bottom row. As it increases to 0.5, half of all the interactions have to be within one container. This can be seen in the middle row. An iCP of 1.0 does not allow interactions within one container, but rather enforces interactions among containers, which is illustrated in the top row. A similar logic holds for the iSP: an iSP of 0.0 requires the interacting services to be accepted by the user as in all graphs in the left column. While iSP $= 0.5$ requires the interacting services to have equally many accepted and rejected services (middle column), an iSP of 1.0 enforces the services not to be accepted by the user, which is shown in the right column.

Now that we explained the generation of composite services, we need to answer the question how the containers for composite services are designed in the simulation. As a device's room is surely a key factor when it comes to composing composite services, the container structure in the simulation is equal to its room structure.

In *Introducing the ConComM Idea* we spent some time explaining why there are many factors that influence the container structure and how ConComM is designed to still extract the underlying structure. Now we evaluate ConComM on a container structure that is exclusively influenced by the room structure. This is possible for one good reason. The ConComM framework itself has no knowledge about the concept of rooms. It extracts the clusters only from the interaction graphs structure, which happens to be influenced by the room structure because we choose this to be the determining factor for interactions. The factors influencing the graph structure can have a nearly unlimited complexity, the graph cut will only cut the graph into a predefined number of subgraphs.

The more interesting question is whether the classifier can learn the clustering based on the features that are given by the nodes contexts. Here again, the classification algorithm itself does not know anything about the fact that we choose random squares in a two dimensional space to be the determining factor for a container. But if it can learn this pattern, it can probably learn any other pattern that is more complex and influenced by more factors than just a x and y coordinate.

5.1.2 The Reward Function

For this evaluation, we used a reward function that uses only the profile based reward components, introduced in 3.3.2. The reward function using those components is evaluated for all services in the composite service. If all services are accepted through the reward function and all of them are within the same room, we accept the composite service as a whole. If any of these criteria is not met, we reject the composite service.

5.1.3 The User Variable

As the reward function that decides over a devices acceptance or rejection is user specific, the users profile strongly influences the outcome of the service composition procedure described above. In order to get a realistic simulation, we put some effort into the design of representative users. The key factors that influence a users reward function are the users `pickiness`, `gender`, `activity`, `early adopter` and `apps`.

Out of those, the early adopter value is best motivated from the diffusion of innovation theory that was first introduced by Beal, Bohlen, et al. [BB+57]. They claim, that every innovation is first used by a small group of so called *innovators*. This group is followed by a slightly bigger group to adapt to that new technology, the *early adopters*. After a while, also the *early majority* integrates the former innovation in their lives, they again

Table 5.1: Early Adopter Values Used to Describe Users

Early Adopter Percentage	**`early adopter`**	**Total Amount Composite Services**
2.5% Innovators	0.99	25
13.5% Early Adopters	0.93	135
34.0% Early Majority	0.68	340
34.0% Late Majority	0.32	340
16.0% Laggards	0.08	160

The different types of innovation acceptance according to [BB+57] with their estimated percentage of the total population is given in the first column. It is followed by our translation to the `earlyAdopter` value used in the simulation. Lastly, the total number of composite services provided by users of each group for a total 1000 composite service simulation is given.

are followed by the *late majority*. The group that is using the innovation either last or not at all is called *laggards*.

A rough estimation of the percentages each group represents in the overall population is given in table 5.1. We translated these percentages into the corresponding user value `early adopter` in [0,1] as follows: The `early adopter` value is 1 minus the sum over all predecessors divided by 100 minus half of the own percentage divided by 100. These values are also given in table 5.1.

Table 5.2: Additional Values Used to Describe Users

`pickiness`	**`activity`**	**`apps`**
1.0	Presentation	{`H,W`}
0.8	Conversation	{`L,W`}
0.6	Break	{`A,H,W`}
0.4	Visually Demanding	{`A,L,W`}
0.2	Brainstorming	{`H,L,W`}
0.0	Reading	{`A,H,L,W`}

Parameters that are randomly assigned to the 6 users for each group. The table gives the possible values for a users `pickiness`, `activity` and `apps`. Concerning apps, the letters encode the following app types: `Hue (H)`, `LIFX (L)`, `Apple (A)` and `WorksAlways (W)`

We choose to design 30 users, 6 for each group. The `early adopter` value is fixed for each group, while the other values still have to be assigned. Six different values for `pickiness`, `activity` and `app` are given in table 5.2. Over all 6 users in a group we assigned these values to the users so that every value is used once, but the corresponding user is picked randomly. Further, a randomly picked selection of 3 users is defined to be female, the other half is male. The apps `Hue (H)`, `LIFX (L)` and `Apple (A)` can be used to connect to Hue or LIFX lamps and to Apple TVs and prevent an acceptance of such services if they are not installed. Also, each user has an app called `WorksAlways` by default that works with devices of the same brand. Such devices can be used by all users without any additionally assigned app.

The graphs we present in this evaluation are based on interaction data obtained by these users. The number of devices forming one composite service is 3 for all evaluations done within this work. Whenever interaction data for a certain probability distribution is requested, these 30 users are used to obtain a number of interactions that depends on the users group. We gave the group specific numbers of contributed services in the last row of table 5.1. Summing up each groups' services, a randomly selected set of one user from each group always results in 1000 interactions in total. This concludes the description how we designed users in order to obtain interaction data to evaluate ConComM.

5.2 How to Evaluate Quality Measures

After the extended discussion how we obtained the interaction data to test ConComM, this section describes how we quantify the quality our clustering results. In 2.4 we describe how the quality of a clustering can be measured. These methods do only work, if one already knows how the clustered clusters are associated to the true clusters, which are the containers in our case. Our approach to match a clustering's clusters to the container's clusters is outlined in the second subsection. As cross validation is an important tool to a get more representative evaluation, the last subsection explains how this validation technique is used to evaluate the classification results. For the whole evaluation we used micro averaging, as the number of relevant points in a cluster determines how strongly the cluster influences the quality measure.

5.2.1 Cluster Comparison

When we introduced quality measures for a clustering in 2.4.1 one might already have wondered how the association between the true class and the predicted class is made. In the illustrated clustering with blue dots and its prediction in 5.2 a human does instantly determine that the mapping *gray cluster $\rightarrow$ blue dots* is the most suitable match. But how can one determine, which mapping *predicted $\rightarrow$ true* class is the best and how do we define the best? Obviously the mapping matters, as the quality measures for *white cluster* $\rightarrow$ *blue dots* would surely not be as good as the 'true' best mapping.

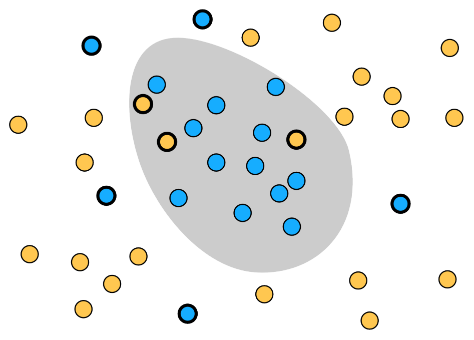


Figure 5.2: Ground Truth and Clustering

A human quickly associates the grey clustering with the cluster of blue dots. However, more effort is required to find the mapping *gray cluster* $\rightarrow$ *blue dots* with an algorithmic approach.

It should be noted that the words *clustering* and *cluster* are used to describe different objects. A clustering is a set of clusters where every cluster is a set of points. In order to evaluate the quality of a mapping, we need to answer the question how to map the clusters of two different clusterings. In the case of the ConComM evaluation, the container structure is given by the rooms and the clusters obtained by graph cuts are to be compared with this ground truth. So initially there is no information about a good mapping between these two clusterings.

A mapping in this context is defined as function $m : C^1 \rightarrow C^2$ that maps the clusters c^1 of a clustering C^1 to the clusters c^2 of a second clustering C^2. The mapping we used does not allow two clusters in C^1 to be mapped to one cluster in C^2, which equals injectivity of the function. There is one exception to that rule, which allows several clusters in C^1 to be mapped to an empty cluster if and only if the number of clusters in C^1 is bigger than the number of clusters in C^2. If any ground truth is part of the mapping, we assume to map the clusterings clusters to the clusters of the ground truth. $|\cdot|$ gives the number of clusters in a clustering. We can distinguish the following three cases:

1. $|C^1| < |C^2|$: In this case the clustering is *injective* as every element in C^1 must be uniquely mapped to an element in C^2
2. $|C^1| = |C^2|$: Here the clustering is *bijective* as every element is mapped to one and only one element in C^2. Bijectivity implies injectivity.
3. $|C^1| > |C^2|$: This case occurs if the first cluster has more elements than the second cluster. If this is the case, the second clustering gets an additional cluster that contains no elements. We allow several clusters of C^1 to be mapped to this empty cluster in C^2, as this is convenient for evaluating the quality measures. Therefore, this case yields a surjective function.

Our mapping for the last case can be brought down to an injective function if clusters are removed from the first clustering until $|C^1| = |C^2|$. The procedure we used to obtain the mapping implicitly does that, as it only looks for a number of associations that is equal to the smaller number of clusters in both clusterings. The empty cluster will be implicitly assumed for all clusters that do not have a directly mapped cluster.

Our procedure to find the best mapping is rather simple. It tries all possible mappings that have the required number of associations and selects the mapping with the maximal sum of true positives over all classes in the mapping. For case 3 all clusters in the first clustering without a corresponding cluster in the second cluster are implicitly mapped to the empty cluster. This yields a total of 0 TP for clusters mapped to the empty cluster.

After the best mapping is obtained, we save all association rules of the form $c^1 \in C^1 \to c^2 \in C^2$. To calculate the quality measures for the clustering, developed algorithm 5.1 to evaluate the TP, TN, FP, FN scores.

Algorithmus 5.1 Cluster Mapping Evaluation

```
procedure CalculateQuality($C^1, m$)
    for each cluster $C$ in clustering $C^1$ do
        for each node $N$ in clustering do
            if $N \in C$ && $N \in m(C)$ then
                increase TP$_c$
            else if $N \in C$ && $N \notin m(C)$ then
                increase FP$_c$
            else if $N \notin C$ && $N \in m(C)$ then
                increase FN$_c$
            else
                increase TN$_c$
            end if
        end for
    end for
    calculate $P_\mu$, $R_\mu$,...
end procedure
```

It is important to note that the clustering in line 2 has to be the clustering that represents the prediction and not the ground truth. We obtain a node's cluster in the ground truth by evaluating $m(c)$ for $c \in C^1$. Also, line 3 goes over all nodes in the clustering - not only those in the current cluster. In our actual implementation, the following lines are only evaluated for those nodes that are existent in *both* clusterings. In the case that cluster C is mapped to the empty cluster, by definition the node can not be in $m(C)$ as $m(C) = \varnothing$. Therefore, clusters mapped to the empty cluster can only increase the false or true negatives.

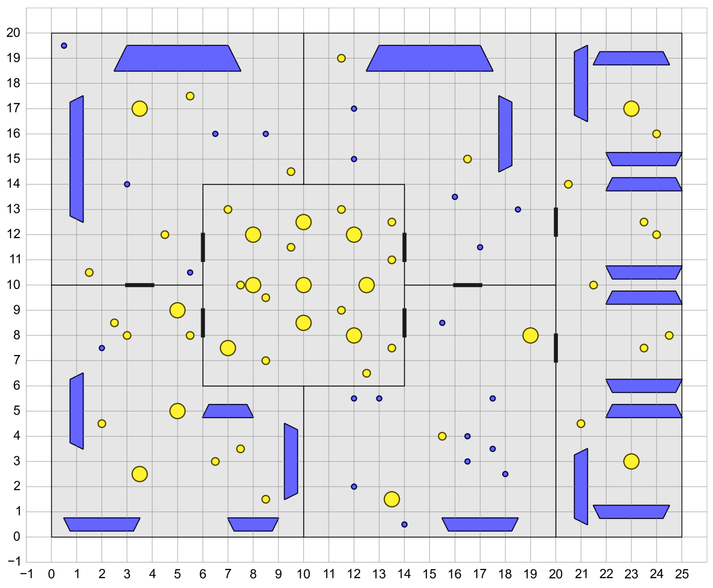


Figure 5.3: Floor Plan for ComComM Evaluation

The floor plan used to extract interactions for the ConComM evaluation contains 91 services in 6 rooms. Out of those services, 50 are light bulbs, 20 are monitors and 21 are unspecified IoT services. Apart from their location, all other properties were assigned randomly.

5.2.2 Cross Validation Parameters

We implemented leave one out cross validation for ridge regression. For the evaluation, each dataset is split into 5 parts and the classifier is trained on 4 of them. Afterwards, classifiers prediction for all data points in the training sample is used as clustering on which the cluster mapping with the ground truth is evaluated. Now that the clustering exists, the evaluation procedure is applied to the left out points with the mapping obtained by the training data.

5.3 ConComM Evaluated

In this section we present the performance of ConComM on a simulated scenario. Our scenario is a simulated floor plan with 6 rooms and 91 services, which was used to gather the composite services. The floor plan is shown in figure 5.3. We obtained the interaction data using the functions and users explained in 5.1.1 and 5.1.3.

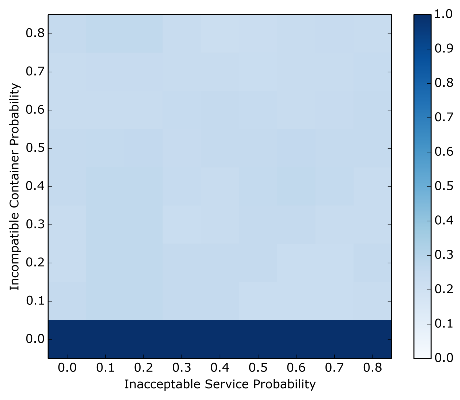
Our evaluation is split into the three main experiments, which we performed to analyse ConComM's performance. Corresponding to the first stage of ConComM, the first two parts analyse the different cut procedures. In 5.3.1 different cut strategies are evaluated for interaction graphs obtained by different iCP and iSP values. In this case, we fix the number of clusters produced by the cut strategy to the real number of clusters. Secondly, we evaluate the stability of the cut techniques for different cluster numbers and representative iCP iSP tuple. In the third and last part of our evaluation we focus on the classification and investigate the cluster number stability for different features.

5.3.1 Clustering Methods over Probability Grid

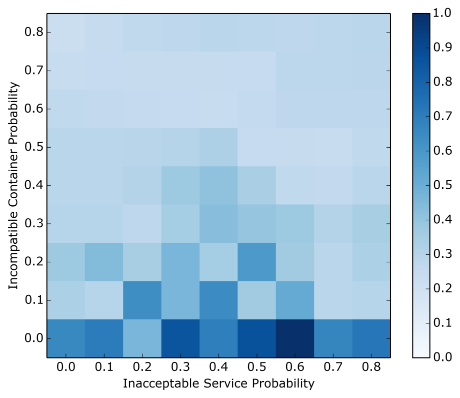
In 5.1.1 we explained in great detail how iCP and iSP, which are used to gather the interaction data, influence the properties of the interaction graph. Therefore our first experiment investigates how the the cut methods perform on interaction graphs obtained for different (iCP, iSP) tuple.

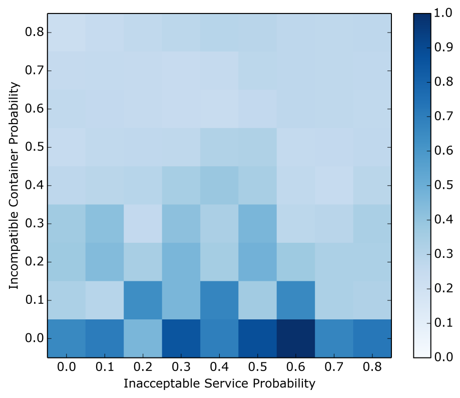
Procedure: We perform the evaluation on a probability grid that covers both probabilities in a range from $[0.0, 0.8]$ with step size 0.1. We investigated the cut procedures SPLIT using Stoer Wagner without refinement, SPLIT using Min-Max clustering with and without refinement and SPLIT$_{rel}$ using Min-Max clustering with and without refinement. SPLIT using Stoer Wagner is here considered as the benchmark, as SPLIT is unmodified and Stoer Wagner is one of the standard graph cut algorithms. As linkage based refinement was introduced for Min-Max cut, we always test Stoer Wagner without refinement, even though the procedure can be applied to any existing cut algorithm. We compared the clustering output against the real container structure, and calculated recall, precision, inverse recall, inverse precision, informedness, markedness and correlation based on these two clusterings.

Figure 5.4 shows the recall values obtained by the 5 cut procedures for every point on the iCP iSP grid. The recall gives the fraction of services that are assigned to the correct container over the number of services that are in the predicted container. Hence a high recall is desirable as it indicates that the correct container structure is estimated very well. Recalling the theoretical analysis of iCP-iSP combinations whose interaction graphs were illustrated in figure 5.1, it is important to figure out what to expect from these plots. As an iCP ≥ 0.5 enforces more than 50% of the interacting services to be in a different room, graphs with these iCP values barely carry any information about the room structure. The recall measures which fraction of the services within a room were identified to be in this room by the clustering. Bringing these two pieces of information together, we can not expect the cut algorithms to perform better than a algorithm that randomly assigns services to clusters, as the iCP exceeds 0.5. Given that we have 6 rooms, randomly assigning services to 6 clusters would yield an expectation value of $1/6$

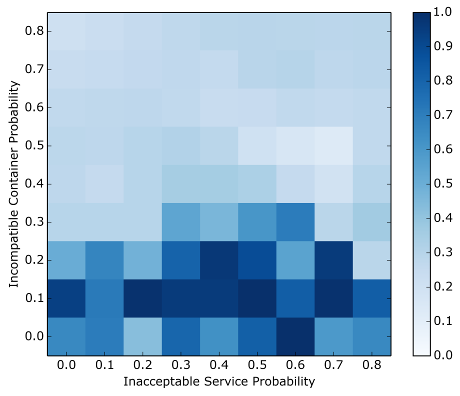


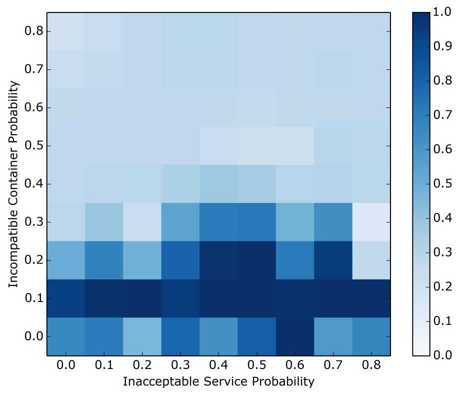
(a) Stoer Wagner SPLIT no refinement





(b) Min-Max SPLIT no refinement **(c)** Min-Max SPLIT refined





(d) Min-Max SPLIT$_{rel}$ no refinement **(e)** Min-Max SPLIT$_{rel}$ refined

Figure 5.4: Recall Evaluated over iCP iSP Grid

The recall value for different k-cut clustering methods evaluated over an iCP iSP grid. The x-axis represents the iSP while the y-axis displays the different iCPs. The values were calculated for a k-cut that cuts the interaction graph obtained by each iCP-iSP combination into 6 subgraphs.

correctly identified services per cluster. Therefore the recall should be approximately $^1/_6$ for iCP ≥ 0.5.

Knowing that we can not expect much for high iCPs the results become increasingly relevant for small iCPs. An iCP of 0.0 yields unconnected graphs that become increasingly connected between rooms as the iCP approaches 0.5. The cut procedures should at least be able to perform well for small iCP values since the cuts that separate rooms are lightest for those values. The more stable a cut procedure is, the better it will perform even for bigger iCP values as the cuts separating rooms are harder to identify due to their increasing weight.

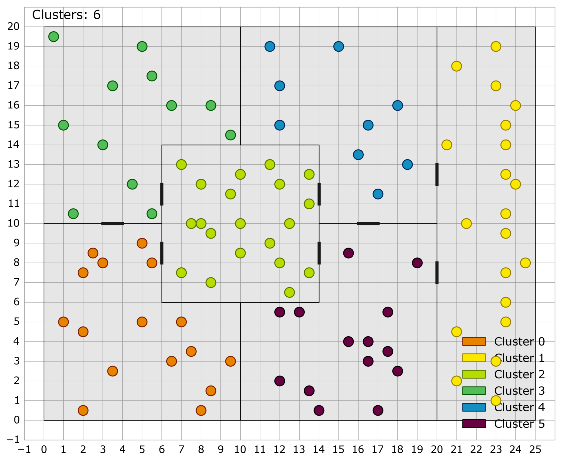
For the different iSP values, the rooms are internally most tightly connected if the iSP ≈ 0.5. In this case, acceptable and unacceptable services are equally likely to participate in an interaction, therefore neither the preferred nor the unacceptable services are left out. Hence the best performance can be expected for iSP values around 0.5 as a cut through a room is likely to have very high weight.

Taking the information about iCP and iSP together, the cut procedures should roughly perform better than random in a 'triangle' of which one edge covers the iCP $= 0.0$ line and the opposite vertex is positioned on the iSP $= 0.5$ line. The bigger the triangle, the more robust is the procedure with respect to changing iCP and iSP values, which is desirable as these values might change depending on the user and the type of composite service that is required.

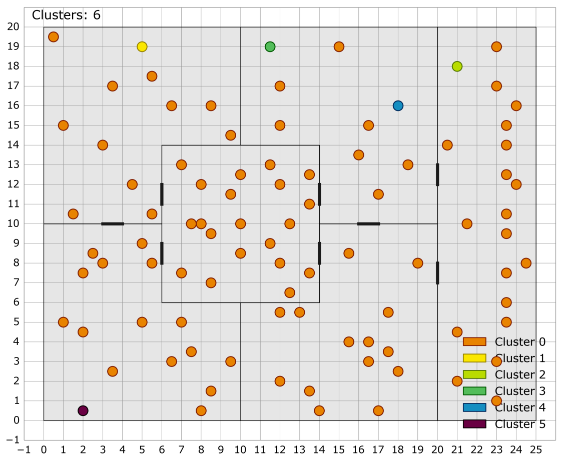
After we identified what is desirable and to expect, the analysis of the recall results for the cut procedures in figure 5.4 becomes more intuitive. As the cut algorithms were used to cut the graph in 6 parts and the floor plan consists of 6 rooms, it is theoretically possible score a recall of 1.0 with perfect graph cut.

Results: As shown in figure 5.4, Stoer Wagner scores the best possible recall values of 1.0 for unconnected graphs (iCP $= 0.0$). For iCP > 0.0 however, is not significantly better than random. Neither of the Min-Max derived methods scores such high values for all samples with iCP $= 0.0$. But heir performance for those probabilities is still much better than random. Furthermore the recall values for iCP > 0.0 do not instantly drop to a random-like performance as in the case of Stoer-Wagner. Both cut procedures using Min-Max and SPLIT$_{rel}$ generally score higher recall values than the ones using SPLIT, especially when the graph becomes fully connected for iCP > 0.0. Using refinement slightly betters the recall values for SPLIT$_{rel}$ but does not have a significant positive influence for Min-Max with SPLIT.

Discussion Figure 5.4a demonstrates what we stated about the Stoer Wagner algorithm with respect to its behaviour for unconnected graphs and its tendency to produce skewed cuts. For unconnected graphs (iCP $= 0$) it scores top recalls of 1.0 as it can



(a) The 6-cut of a graph obtained by (iCP,iSP) = (0.0,0.2)



(b) The 6-cut of a graph obtained by (iCP,iSP) = (0.1,0.2)

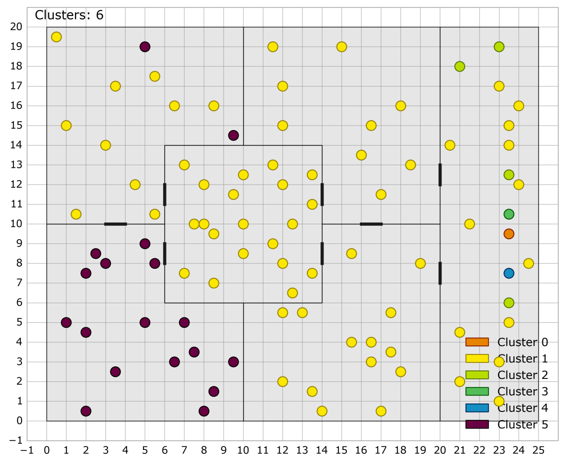
Figure 5.5: Clustering Example for Stoer Wagner

The good performance of Stoer Wagner for an unconnected graph $(0.0, 0.2)$ and the skewed cuts as soon as the graph becomes connected for $(0.1, 0.2)$. The iSP has no influence on the performance of Stoer Wagner and is therefore fixed to the arbitrary value 0.2.

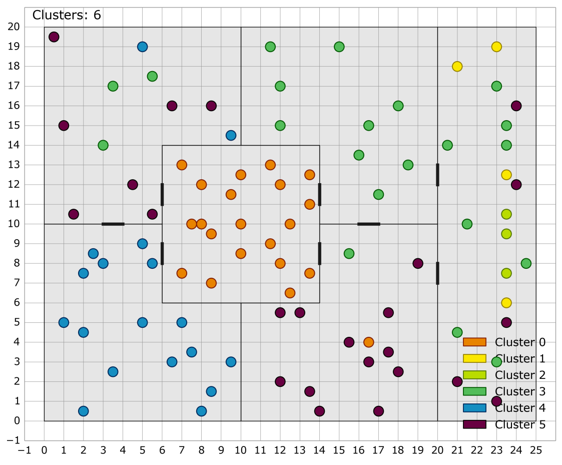
reliably identify the unconnected room structure. However, as the containers become slightly connected, Stoer Wagner starts cutting single vertices as they give the lightest cut. The two clusterings obtained by Stoer Wagner for $(\text{iCP}, \text{iSP}) = (0.0, 0.2)$ and $(\text{iCP}, \text{iSP}) = (0.1, 0.2)$ in figure 5.5a illustrates the matter. Hence Stoer Wagner only performs well for iCP $= 0$, irrespective of the iSP.

Due to their problems with unconnected graphs, all four strategies based on the Min-Max clustering perform not as good as Stoer Wagner for iCP $= 0.0$. However all of them outperform Stoer Wagner as soon as the graph becomes slightly connected for iCP > 0. Also, our prognosticated triangle-like shape with recall greater than $1/6$ can be seen for all of them. Nevertheless our both strategies using SPLIT$_{rel}$ outperform the Min-Max SPLIT combination, as Min-Max SPLIT$_{rel}$ scores top results for both iCPs 0.1 and 0.2 and still gives reasonable results for 0.3 in the middle iSP range. The both SPLIT based procedures only achieve results that are similar to those of SPLIT$_{rel}$ with iCP $= 0.3$ for the much smaller iCP of 0.1.

On the probability grid the effect of the modification in SPLIT$_{rel}$ becomes visible. SPLIT tends to make one or two good cuts and afterwards gets stuck in splitting the smaller graph as this yields lighter cuts. Since SPLIT$_{rel}$ chooses not the lightest cut but the cut that is lightest given the number of nodes it separates, it cuts ‘meaningful’ partitions much longer before it falls into the SPLIT behaviour. One of the many samples for such a situation is shown in figure 5.6.



(a) SPLIT for (iCP,iSP) = (0.2,0.1)



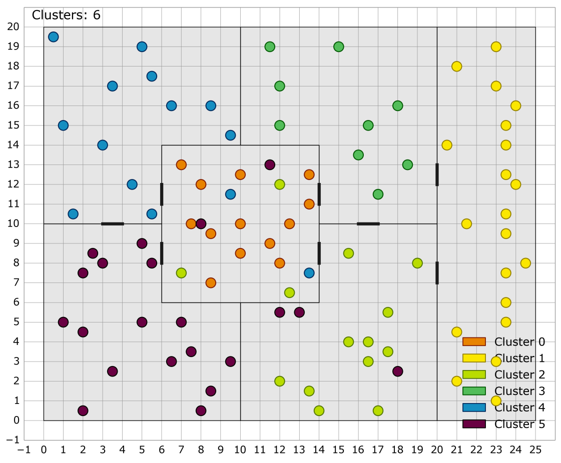
(b) SPLIT$_{rel}$ for (iCP,iSP) = (0.2,0.1)

Figure 5.6: Behaviour of SPLIT and SPLIT$_{rel}$ for (iCP,iSP) = (0.2,0.1)

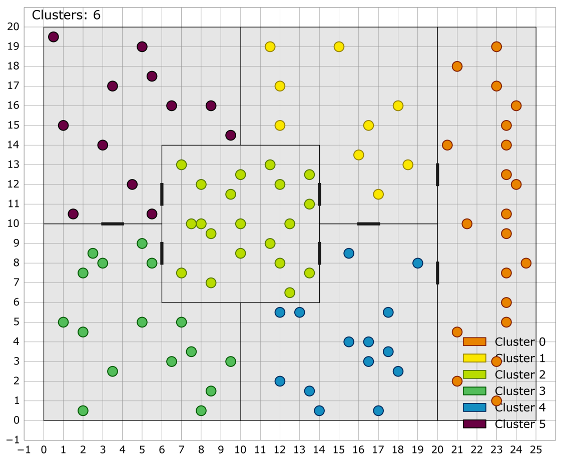
Illustration of the behavioural difference of SPLIT and SPLIT$_{rel}$ for a 6-cut, both using Min-Max cut without refinement on $(0.2, 0.1)$. SPLIT quickly gets lost in cutting the smallest subgraph into smaller subgraphs as they have the lightest cut weights. SPLIT$_{rel}$ in contrast is longer choosing cuts that split more than a few points, as its selection routine considers not only the cut weight but also the number of nodes separated.

The last influence we did not analyse so far is the linkage based refinement, which can be applied to refine the results obtained by Min-Max clustering. Generally, this procedure identifies nodes that are better linked to the other cluster and reassigns them if they lower the overall cut weight. In the case of SPLIT$_{rel}$ this often leads to an improved clustering, as the original cut is a rough approximation of the cut direction while the linkage based refinement 'cleans' the cut by ordering the directly affected nodes into the right clusters. Figure 5.7 shows a good example for a successful linkage based refinement.

However, there are cases where applying refinement makes things worse instead of improving the result. This might be the case for cuts that already did represent some part of underlying container structure well but are forced to do more cuts. It can happen that a previously good approximation of a container is cut. Taking the lightest cut would hereby often result in only cutting a few nodes, as the nodes are very well interconnected. But the refinement often adds more nodes to the enforced cluster, making the cut equally sized. This is not good for the hight level performance of the clustering as it was more accurate without the new nodes. Even though these cases can occur, linkage based refinement tends to make things better instead of worse. Nevertheless, it is important to be aware of this fact to know the method's limitations.



(a) No refinement for Min-Max cut



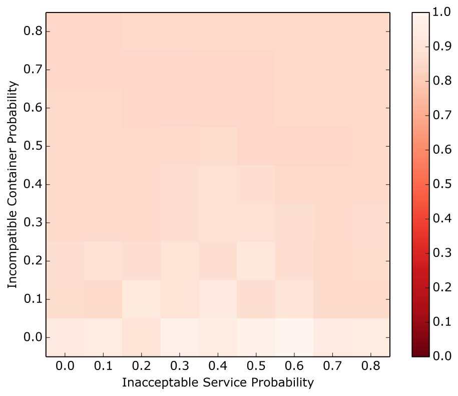
(b) Refinement with Min-Max cut

Figure 5.7: The Effect of Using Refinement for SPLIT$_{rel}$ with Min-Max Cut on (iCP,iSP) = (0.2,0.5)

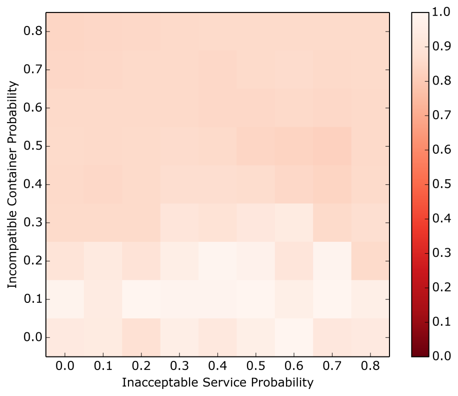
The positive effect of linkage based refinement for SPLIT$_{rel}$ and Min-Max cut applied to $(0.2, 0.5)$. This illustrates the successful reordering of nodes from a preliminary cut in order to further lower the cut weight. The preliminary cut result is shown in the version without refinement, while the reordering to lower the overall cut weight can be seen for the version with refinement.

After we discussed the results for the recall values over the iCP-iSP grid in great detail, we give some insight in the measures inverse recall, informedness and correlation in figure 5.8. The results are only shown for Min-Max with SPLIT and SPLIT$_{rel}$ without refinement, as the refinement does not change the general tendency and Stoer Wagner scores top results for iCP $= 0.0$ and acts random like for all other values. Due to our mode of cluster assignment presented in 5.2.1, the values for precision/recall and inverse precision/inverse recall are the same. This is sound as every value that is a false negative in one cluster, which means a point was wrongly predicted not to be within this cluster, is a false positive in another cluster. The micro averaging strategy adds up all false positives and false negatives over the clusters, which leads to the same number of summed false positives and false negatives. Consequently precision and recall take the same value. Due to their computation, the values for markedness and informedness are equal as they are based on precision (markedness) and recall (informedness). What we said about precision and recall also holds for inverse precision and inverse recall. Therefore only one plot for each of these indicator pairs is given.

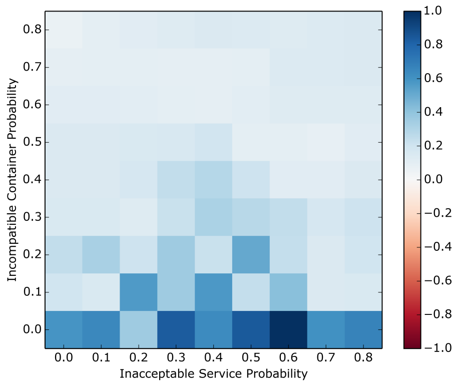
Results: The inverse recall (5.8a and 5.8b) is very high for SPLIT and SPLIT$_{rel}$. Even if they approach random results for iCP > 0.5, the inverse recall is not smaller than 0.9. However for iCP $\rightarrow 0.0$ the inverse recall goes to 1.0. Within the relevant probability triangle the values for Min-Max with SPLIT$_{rel}$ are generally closer to the optimal value



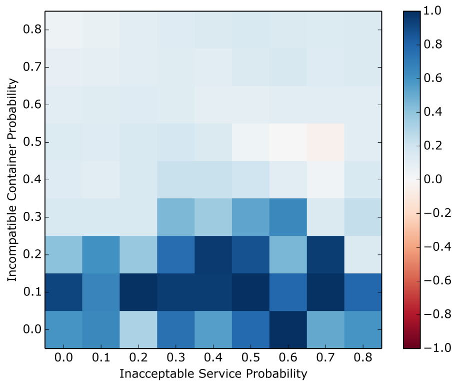
(a) InverseRecall for SPLIT with Min-Max



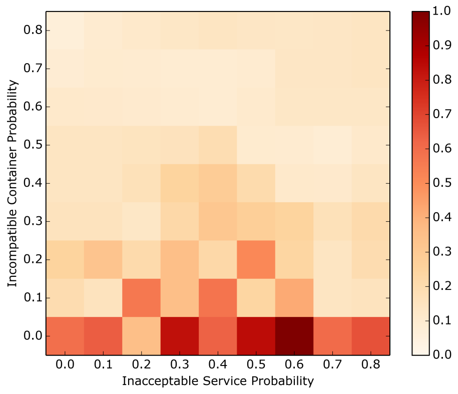
(b) InverseRecall for SPLIT$_{rel}$ with Min-Max



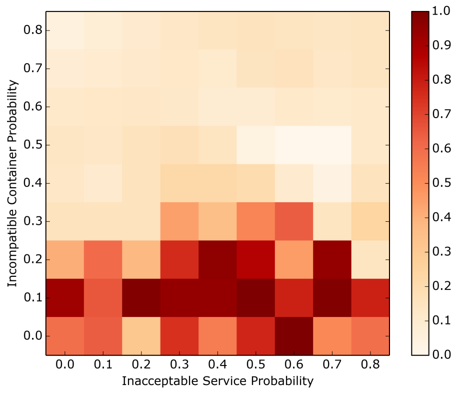
(c) Informedness for SPLIT with Min-Max



(d) Informedness for SPLIT$_{rel}$ with Min-Max



(e) Correlation for SPLIT with Min-Max



(f) Correlation for SPLIT$_{rel}$ with Min-Max

Figure 5.8: Inverse Recall, Informedness and Correlation for SPLIT and SPLIT$_{rel}$

Further evaluation of SPLIT and SPLIT$_{rel}$ with Min-Max without refinement on the probability grid for a 6-cut. Due to the micro averaging strategy, the values for precision and recall, inverse precision and inverse recall, informedness and markedness are the same.

1.0 than those for Min-Max with SPLIT, following the same pattern that was already observed for the recall values. For iCP > 0.5 the informedness is very close to 0.0, negative values are not obtained for any of the k-cut procedures. The values are mostly greater than 0.5 within the relevant triangle and even tend to approach the optimum 1.0 for Min-Max with SPLIT$_{rel}$. As the micro averaging strategy results in the same values for informedness and markedness, the correlation has to be equal to informedness and markedness which is the case.

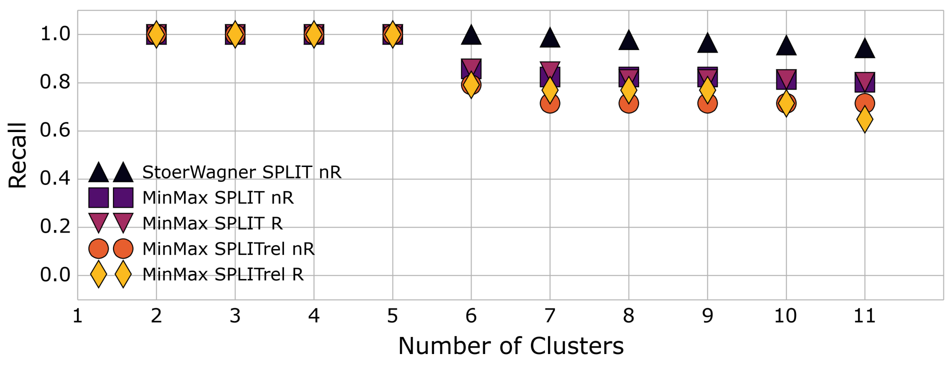
Discussion: An inverse recall of 0.9 is the expected value for randomly assigning points to 6 equally sized classes. Hence the inverse recall has to be bigger than 0.9 if the obtained clusters are more accurate than randomly assigned ones. As Min-Max with SPLIT$_{rel}$ approximates the real container structure better than Min-Max with SPLIT, the inverse recall is also higher for this method. The same argument holds for the informedness. A as the clusters obtained by this method represent the container structure and do not actively try to exclude points within a certain cluster, the informedness has to be at least 0.0 and can not be negative - which would represent such an ‘uninformed’ clustering. Due to its definition, for micro averaging the correlation is the absolute value of the informedness.

Our evaluation over the probability grid clearly shows that cut procedures based on our specifically designed SPLIT$_{rel}$ tend to be more accurate than those based on SPLIT. So far we assumed that the number of clusters obtained by the k-cut is equal to the number of real clusters. In the next subsection we investigate how well the cut procedures perform if the number of clusters obtained by the procedures is not equal to the real number of clusters.

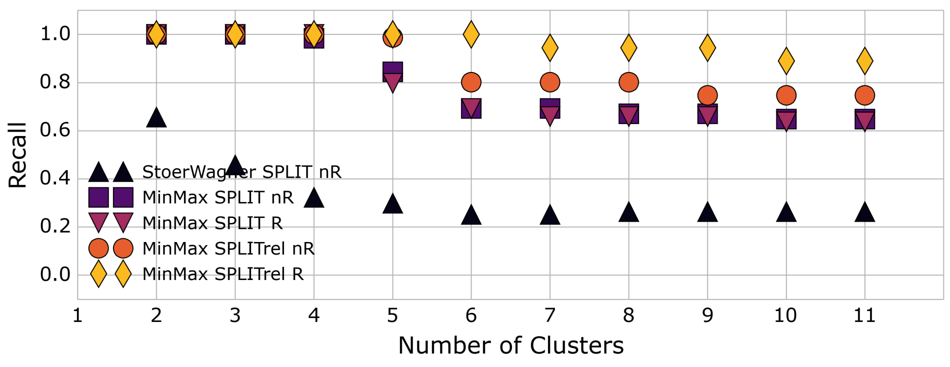
5.3.2 Clustering Methods: Cluster Number Stability

In a real scenario we can not generally assume that the number of containers is known. In the best case, we can expect a rough estimation. Therefore it is important to investigate whether the cut procedures used for ConComM still produce good cuts, if the selected number of clusters is not equal to the real number of containers.

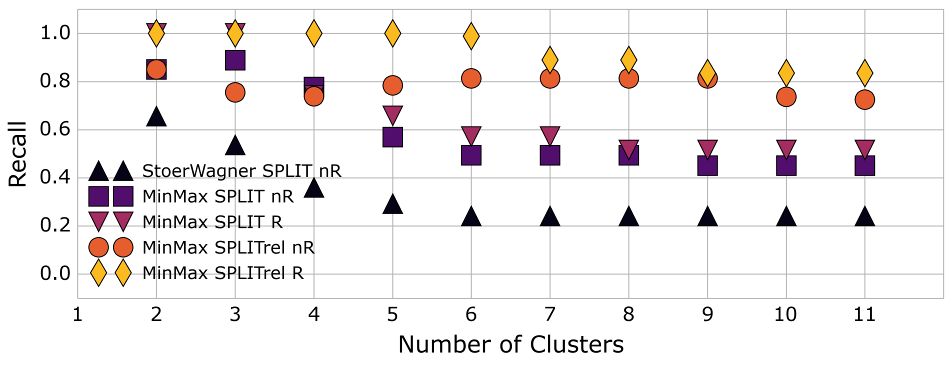
Procedure: For selected probability tuple the number of clusters k for the k-cut was varied from 2 to 11, knowing that the real number of containers was 6 as in the previous experiment. Once again, we evaluated the quality measures presented in 5.2 for every number of clusters and cutting procedure. Figure 5.9 shows sample results for the iCP iSP tuple $(0.0, 0.3)$ (5.9a), $(0.1, 0.2)$ (5.9b) and $(0.2, 0.5)$ (5.9c).



(a) (iCP,iSP) = (0.0,0.3)



(b) (iCP,iSP) = (0.1,0.2)



(c) (iCP,iSP) = (0.2,0.5)

Figure 5.9: Recall over Number of Clusters

Changes in the recall for 5 cutting strategies and different iCP iSP tuple if the numbers of clusters varies from 2 to 11. The cutting strategies are Stoer Wagner with SPLIT without refinement (StoerWager SPLIT nR), Min Max cut with SPLIT refined (MinMax SPLIT R) and without refinement (MinMax SPLIT nR) and Min Max cut with SPLIT$_{rel}$ refined (MinMax SPLITrel R) and unrefined (MinMax SPLITrel nR). The evaluation is based on interaction graphs obtained by a scenario with 6 containers.

Results: Without refinement, Stoer Wagner and SPLIT only scores high precision values for the whole range of cluster numbers for iCP = 0.0, which is demonstrated in 5.9a. For all other iCP values, the recall starts around 0.6 for 2 clusters and decreases to a value around 0.2. There it remains for any number of clusters that is bigger than the real value 6.

Even though the Min-Max based cut procedures do not score recalls as high as those for Stoer-Wagner and iCP = 0.0, their values are not lower than 0.7 even for 11 clusters. We consider this quite stable given that 11 clusters are twice as many clusters as there are in reality. For iCP > 0.0 Min-Max based procedures score recalls that are better than those for Stoer-Wagner by 0.2 at least. It also is well observable that SPLIT$_{rel}$ improves the cuts recall by 0.2 in comparison to the SPLIT procedures on average. Also the positive effect of using refinement (R) in contrast to not using it (nR) can be seen in all three figures.

Within 5.9b and 5.9c the positive effect of SPLIT$_{rel}$ on a cut-procedures performance is demonstrated again. Even without refinement, Min-Max with SPLIT$_{rel}$ (circular marker) rarely scores recall values smaller than 0.8 as it exceeds the real number of clusters. The results with refinement (diamond marker) are even better.

Discussion: Generally, the cut procedures recall is very stable after they reached number of real clusters. For smaller cluster numbers, the recall is generally higher. This is due to the mode of evaluation that was used to calculate the recall for these experiments. If the number of clusters is smaller than the real number of containers, only those clusters are considered for the recall evaluation that are mapped to a real cluster. Hence each cluster is mapped to a real cluster that fits well and points that are not within both clusters are not considered for the evaluation.

To give a small example, if only one cluster exists this cluster is mapped to some real cluster. If the recall is small, the cut cluster missed out many points that had to be within the container. For a few big clusters, the number of missed out points is small which results in a high recall close to optimal 1.0. However, the negative effect of too few clusters is represented by the inverse precision. The inverse precision gives the fraction of values that correctly identified not to be within the cluster over all values the cluster does not contain. In other words, the more values are wrongly added to a cluster, the lower the inverse recall. If a small number of cluster is mapped to a clustering that contains more clusters, the few clusters will most likely contain many values that do not belong to the real clusters they are mapped to. Therefore the inverse recall will be lower, the smaller the number of cut clusters and the bigger the difference to the real number of clusters. The effect is demonstrated in figure 5.10 for the representative example (iCP,iSP) = (0.2,0.5). For all cutting procedures the inverse precision increases until the real number clusters is reached where the increase terminates.

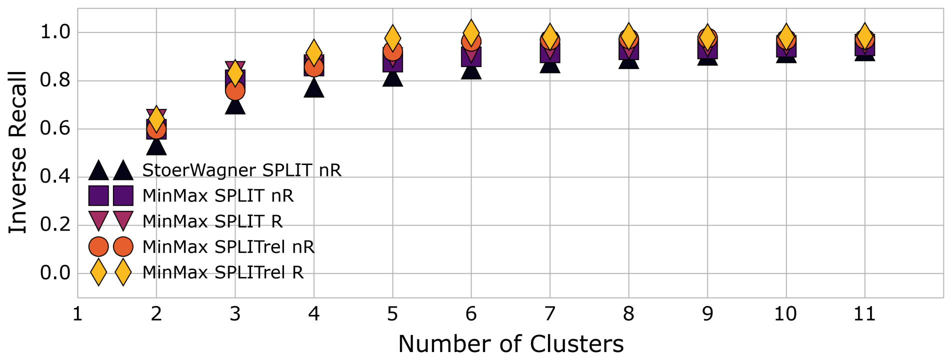


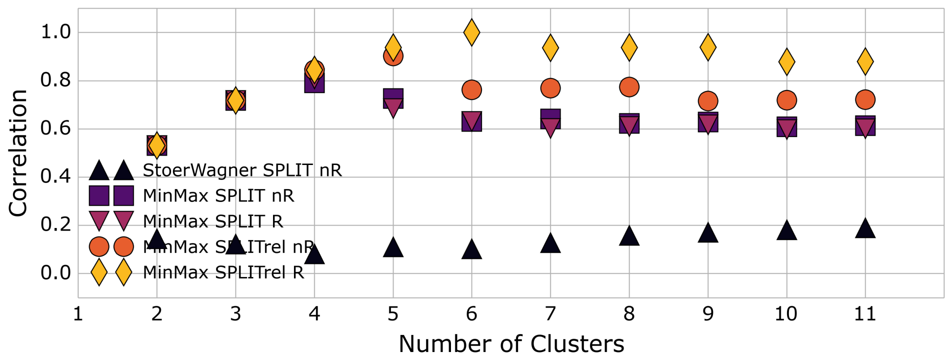
Figure 5.10: Inverse Recall for Cut Strategies (iCP,iSP) = (0.2,0.5) over Number of Clusters

Changes in the inverse recall for 5 cutting strategies if the numbers of clusters varies from 2 to 11. The true number of containers is 6.

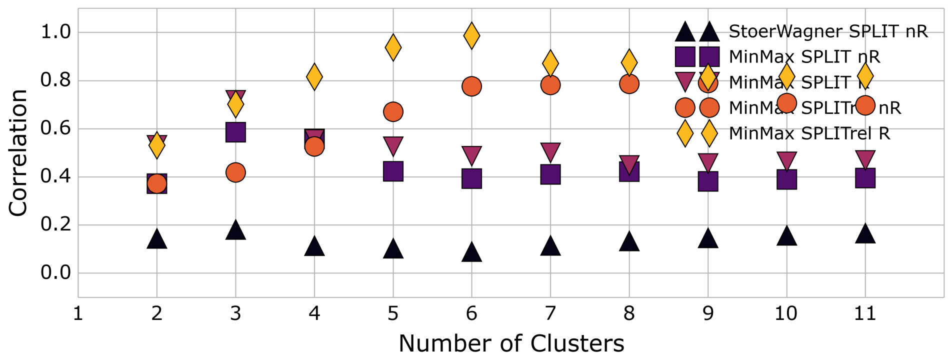
Procedure: As the correlation is calculated using not only recall and precision but also their inverse values, it is a good measure for the overall quality of a clustering. In figure 5.11 we plotted the correlation over the number of clusters for the probabilities $(0.1, 0.2)$ and $(0.2, 0.5)$ as these plots are representative samples.

Results: As a general tendency, the correlation increases until it reaches the real number of clusters and remains within 0.1 of this peak correlation value with a tendency to decrease for higher cluster numbers. Both examples in figure 5.11 show a strong increase in correlation until the real number of clusters is reached. This increase is followed by a stagnation or very slow decrease around a value that is not more than 0.2 below the optimal. For iCP > 0, which we assume to be the case for the majority of real world graphs, cut procedures with SPLIT$_{rel}$ outperform those based on SPLIT. As for the SPLIT based procedures, those using Min-Max cut clearly outperform the Stoer-Wagner using procedure. The use of refinement (R) further improves the results for SPLIT$_{rel}$ by 0.1 on average, the increase for SPLIT is lower. With a stable average correlation of 0.8 for all cluster number exceeding 6, cut procedures using SPLIT$_{rel}$ give also high quality results for wrongly estimated cluster numbers.

Discussion: The correlation visualizes the combined influence of recall and inverse recall in the combined measure. For cluster numbers smaller than the real value 6, the low inverse recall reduces the overall clustering quality for the cutting methods. As the number of cut clusters exceeds 6, the increased number of clusters separates values that originally belong to one cluster. Hence an imperfect cluster is mapped to the real cluster, missing out relevant values which leads to a decrease in recall.



(a) Correlation for (0.1,0.2)



(b) Correlation for (0.2,0.5)

Figure 5.11: Correlation for Cut Strategies over Number of Clusters

Changes in the correlation for 5 cutting strategies and different iCP iSP tuple if the numbers of clusters varies from 2 to 11. The true number of containers is 6.

Overall, the recall has a high stability for an increasing number of clusters. This implies that even if the number of clusters is estimated incorrectly, the graph cut still represents the original container structure as good as if the correct number would have been given. Considering the results in 5.11, we consider it advisable to use the cutting strategy Min-Max with SPLIT$_{rel}$ and refinement in ConComM. This shows that Min-Max with SPLIT$_{rel}$ is indeed very suited to identify meaningful cuts, even if the graph is strongly interconnected as for (iCP,iSP) = (0.2,0.5). If the number of real containers is unknown, our results suggest to overestimate the number of clusters as the correlation remains high even for cluster numbers that are wrong by a factor of 2 compared to the real container number.

5.3.3 Classification Results

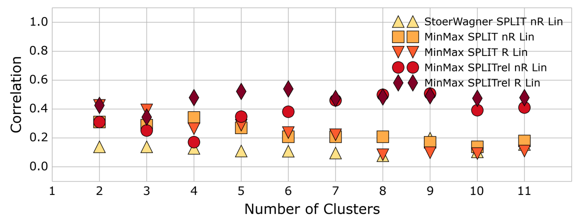
In the last two subsections we evaluated the first ConComM stage. Our results indicated that the cutting procedure Min Max cut with SPLIT$_{rel}$ and refinement is the most capable k-cut technique for our investigated cases. But ConComM not only consists of a graph cutting part. We use the clustering obtained by the cut to train a classifier, in order to predict the container even for those points that have not yet been part of any interaction and therefore the interaction graph.

Methodology: In this subsection we quantify the quality of classifiers that were trained on the outcome of the different k-cut strategies. The classifiers were obtained using logistic regression with different features. In this classification evaluation, we investigate two main questions. Firstly we are interested to find good features that produce reliable classifiers, even if the number of classes is much higher than the real number of containers. Hence we trained logistic regression trained using linear and quadratic features and compared them against each other.

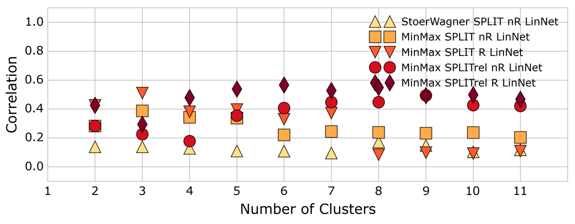
The second question addresses the uncertainty in factors influencing the container structure. If the classifier should be able to learn the container structure, the influencing factors have to appear in the features used for classification. As one we not tell what is required (influencing) information and what is not when we define the features for the first time, we have to assume that the features include a lot information that does not influence the container structure. Therefore the feature vectors contain a lot of noise due to these unnecessary information parts. We consider it important to find out how the classification results change if unnecessary information is added to the features. For the evaluation, the relevant parameters to describe containers are x and y coordinate, as the containers are given by the room. To add noise, we added information about a service's network to the features. Within the simulation we introduced three networks that were randomly assigned to the services. In order to use the network information for the features, the networks were mapped to the numbers 1, 2 and 3.

Procedure: The classifier is trained on the services contexts labelled with the clusters obtained by the k cut. The whole dataset of 91 service contexts is randomly split into 5 nearly equally sized subsets. Next, leave one out cross validation is applied to the sub datasets. This results in 5 classifiers for *each* combination of feature, cutting method and number of cuts. We calculated the quality measures for the left out data points and averaged the measures over the 5 classifiers.

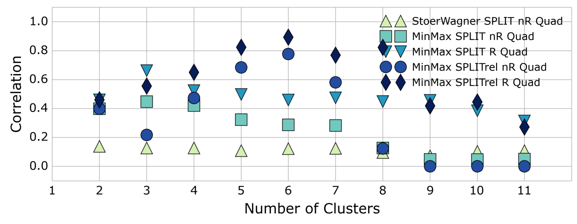
Results: As the results and tendencies are very similar for all relevant iCP iSP tuple only the results for (iCP,iSP) = (0.2,0.5) are shown here. Figure 5.12 gives the correlation of classifiers based on four different features types and the different k-cut strategies



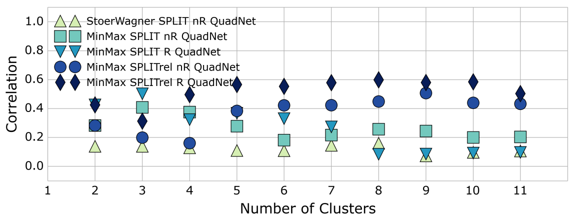
(a) Linear Features for x and y coordinate



(b) Linear Features for x, y coordinate and Network



(c) Quadratic Features for x and y coordinate



(d) Quadratic Features for x, y coordinate and Network

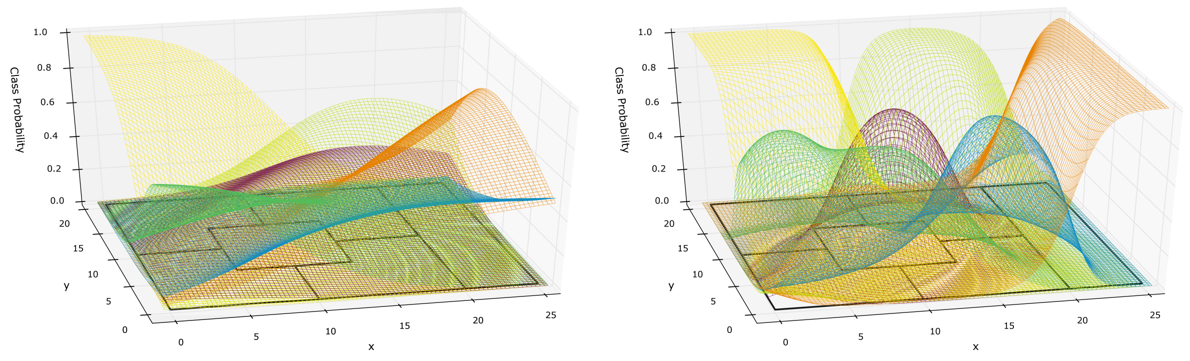
Figure 5.12: Classifier Correlation for Different Features

The correlation of classifiers trained using logistic regression on the k-cut results for different cut procedures. Here the classifiers were trained using different features. All of them use x and y coordinate of a training point (IoT service) with additional network information in two cases. Subplots 5.12a and 5.12b use linear features represented through an orange colour scheme, while 5.12c and 5.12d use quadratic features marked with a blue colour scheme.

over the investigated number of clusters. The different feature types are linear features (orange colour scheme) for x and y coordinate (*LinXY* 5.12a) and x, y with network (*LinXYNet* 5.12b) as well as quadratic features (blue colour scheme) with purely x and y (*QuadXY* 5.12c) or additional network (*QuadXYNet* 5.12d).

We observe that the results for the feature types LinXY, LinXYNET and QuadXYNet look very alike in terms of overall performance and stability over number of clusters. Especially the stability over number of clusters is constant and high, as the results classifiers trained on SPLIT$_{rel}$ based cut results are in range $[0.4, 0.6]$ for all cluster numbers > 5. All other methods perform worse than this.

The outlier are classifiers trained with QuadXY features. Here the peak performance for SPLIT$_{rel}$ is around 0.9 for 6 clusters. However, the performance decreases as the number of clusters exceeds 8, which indicates a suboptimal stability. For quadratic features is is therefore important to note that introducing the network information smooths the results and makes them more stable. The same can be observed for a comparison of quadratic and linear features. Reducing the features complexity from quadratic to linear has the same flattening effect as the introduction of noise.



(a) Linear Features for x and y coordinate **(b)** Quadratic Features for x and y coordinate

Figure 5.13: Classifier Probability Distribution over Container for Different Features

The probability distribution for a point in the x-y-plane to belong to a certain container (room) for the example setup shown in figure 5.3. The probability functions for the 6 different classes are shown in yellow, orange, light green, dark green and blue and the real container outlines are given in black. The classifiers were both trained on the 5-cut outcome of SPLIT$_{rel}$ with Min Max cut without refinement on interaction data obtained by (iCP,iSP)$=(0.2, 0.5)$. The plots show the structural difference in probability distributions of classifiers trained with linear or quadratic features.

Discussion: Both observations can be explained through regularization. The more complex features get, the more prone to overfitting the classifier becomes. Features that allow the classifier to strongly adapt to single outliers can lead to an overall performance reduction. This is illustrated in 5.12c. If the input data is very correct, the classifier is able to perfectly model the underlying ground truth as for 6 clusters. However, if the input data labels do not represent the underlying ground truth perfectly (as for more than 8 clusters), the classifier is rather learning the noise instead of the general tendency.

Figure 5.13 shows the structural difference in the probability distribution of two different classifiers, trained with linear and quadratic features. The probability distribution for quadratic features 5.13b represents the underlying room structure much more accurately. This is because quadratic functions allow a more accurate model of the geometry than linear functions. However it can also model noise more accurately. Linear features on the other hand can not model the geometry as accurately but this limitation turns into a strength when noise makes generalization important in order to see the general tendency. The introduction of network information as feature also acts as regulator, which explains why quadratic features with network information produce more stable results.

In comparison to the correlation results for the pure cut strategies in figure 5.11b, the correlations for the trained classifiers are lower by at least 0.2 on average. It is not surprising that the quality of classifier is lower as it was trained on imperfectly labelled data and depending on the choice of features the classifier also introduces an error. But given given that each logistic regression is trained with about 72 data points only, the performance of the classification can be considered high. Under the (not completely correct) assumption that each cluster contains about the same amount of samples, this gives less than 10 training points per class for all cluster numbers greater than 7. Even though the assumption of equally distributed samples is wrong, the number of samples is still small what makes a correlation of 0.5 a good result.

Summing up the results of this section we can state the following: SPLIT$_{rel}$ with Min Max cut is a very powerful method to cut graphs with approximately equally sized clusters. Not only does it give reliable results even for graphs with strongly interconnected clusters, it also produces highly correct cuts if the number of cuts is twice as big as the number of real clusters. For this task it outperforms all other tested methods. The quality of a classifier trained on the corresponding graph cuts is lower than the cut quality and more stable over the cluster numbers if regularization is introduced. As adding non relevant information to the feature vector introduces noise and therefore acts as regularization, we require no knowledge about the factors influencing the container structure to train well performing classifiers. Taking all this together, ConComM works as we intended it to. It takes only an interaction graph and learns the container structure without prior knowledge about the factors shaping the structure.

After we evaluated our tool to predict a services container, in the last section we investigate whether this additional information improves composite service recommendation with bandit algorithms.

5.4 Closing the Circle: LinUCB for Composite Service Recommendation II

In 3.4.4 we demonstrated that LinUCB's performance significantly decreases when it is used to recommend composite services. ConComM provides additional contextual information about each service's interaction enabling container, what can be incorporated in the LinUCB approach.

Procedure: Our experimental setup is the same as in 3.4.4. This included that a composite service is rejected a whole if either one of its component services is not accepted by the users reward function or if they are not within one room. We added the ConComM probability distribution over the rooms for each service as service feature to the bandit. Furthermore, we added the users room as user feature.

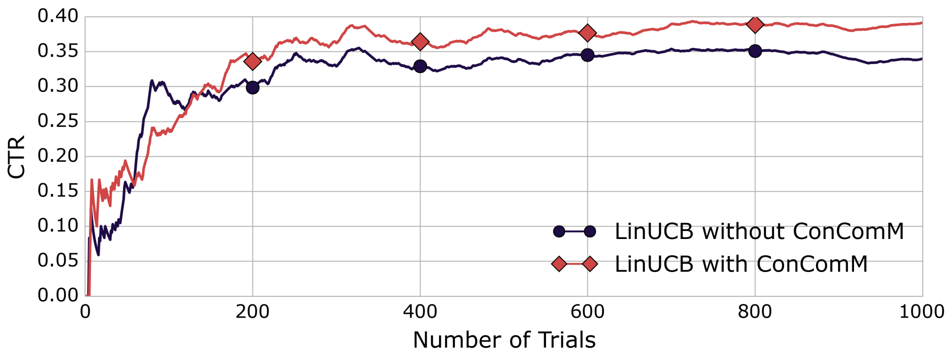


Figure 5.14: CTR for Composite Service Recommendation with ConComM

The CTR for LinUCB used for composite service recommendation over the total number of recommendations made. The dark blue line with circular markers shows the results for LinUCB that has no information about a services container. Plotted in read with square markers are the results for a LinUCB approach that utilizes container information about the services provided through ConComM.

Figure 5.14 shows CTR for composite service recommendation with LinUCB using the ConComM container estimates (square marker) and not using it (circle marker). The CTR is plotted over the number of consecutive recommendations.

Results: Utilizing the ConComM information improves the CTR by about 14% in comparison to the standard approach if the number of trials exceeds 200. This long term trend validates the initial conjecture, which assumed knowledge about the container structure to improve the results for composite service recommendation. Another interesting observation can be made for the first 100 recommendations. Within the first 50 recommendations, the approach using ComComM achieves CTR's twice as high as for the benchmark approach. Afterwards, the benchmark outperforms the ConComM approach for a few iterations before ConComM takes over again and consolidates its position.

Discussion: The first few (<50) recommendations are most crucial for a system relying on user feedback. It has to perform acceptable especially with only a few samples in order to motivate users to keep using it. Hence the strong CTR increase within the first 50 recommendations is one of the biggest advantages of the ConComM using LinUCB over a standard LinUCB. Our above presented strategy to incorporate the ComComM information as feature into LinUCB is only a first approach. Further possibilities exist, but their implementation and evaluation is beyond the scope of this work. But even for one of the most simple strategies, the results are already very promising. The container information we provided through ComComM can therefore be used as starting point for further investigation to improve bandit approaches for composite service recommendation.

5.5 ConComM Method Discussion

Given the CTR increase for composite service recommendation with LinUCB and incorporated ConComM data in comparison to an unmodified LinUCB, the general framework clearly served its purpose. The modification of SPLIT in order to choose more meaningful cuts that resemble the min max clustering principle was also a clear success. For a broad variety of graphs, the methods based on SPLIT$_{rel}$ outperformed the benchmark procedures by far and also showed a high stability even if the number of cuts was much higher than the real number of clusters.

For ConComM we focused on the development of a good cut method as we hoped that a cut procedure that reliable extracts the container structure would give more accurate classifiers, as they can only be as good as the training data. This connection is well observable, given that the classifiers trained on SPLIT$_{rel}$ cut results score much higher correlation values than the classifiers trained on SPLIT based procedures. However the loss in quality for the classifiers in comparison to the raw cut results can be explained by the construction of our reward function. The probability that all three services are accepted even if we guarantee that they are all within the same room is the probability of one service being accepted to the power of three, for three services in a composite service. As LinUCB hybrid scored CTR values around 0.7, which can be seen as the probability that a device is accepted, the probability for an accepted composite service is 0.343. This matches very well with the CTR for LinUCB hybrid with ConComM information.

However, there is still room for improvement as we did not spend a lot thought on a sophisticated features for the classifier. We rather tried to identify good cut algorithms to obtain high quality training data than to tune the classifiers features to give good results. With better feature engineering one might be able to further improve the classification results.

Currently we only evaluated ConComM based on one sample scenario with 91 IoT services. To truly explore the full potential and possible shortcomings of the approach, more experiments would have been desirable. The reason for conducting all tests only based on one scenario was the high computational effort to evaluate the different ConComM stages and linked to that, the long time per evaluation.

While the use of ConComM makes the computation more complex in comparison to only using LinUCB, an improvement by 14% is a significant increase. Unfortunately we could not investigate the effect of designing different feature sets for LinUCB and how they effect the method's performance. Given the current simplicity of linking LinUCB with ConComM, it is a justified expectation that a more sophisticated way of feature incorporation will further improve the results.

6 Conclusion

Success is not final, failure is not fatal: it is the courage to continue that counts.

(Winston S. Churchill)

6.1 Approach Summary

The emerging Internet of Things and the integration of real world services into the web creates new challenges in terms of IoT service discovery. One of the key problems is the recommendation of contextually relevant services to users who request a certain type of service. Preliminary studies have shown Contextual Bandit algorithms like LinUCB to be well suited for IoT service recommendation.

Testing recommender systems for this purpose is problematic due to the lack of reliable data sets, as current buildings and environments rarely offer the high density of IoT services these systems are designed to handle. To address this issue an ambient space simulation was developed in chapter 3, which can be used to simulate large numbers of IoT services in an artificial building environment. Utilizing the simulation it was possible to confirm the good performance of the LinUCB approach for single service IoT service discovery.

In the course of investigating LinUCBs abilities for IoT service recommendation tasks it became apparent that the bandit approach, however suited for single service recommendation, did not perform well for recommending multiple services for a combined task. The development of a framework that would improve LinUCB's performance for composite service recommendation became the second and main contribution presented in this thesis. Based on the idea that information about services that work well together would improve LinUCB's performance, the ConComM framework aims to identify services that can be guaranteed to work together in a composite task.

Only requiring information about previous interactions among IoT services, ConComM uses a specifically designed k-cut procedure based on Min-Max cut and a modified version of SPLIT that is called SPLIT$_{rel}$ to cut the interaction graphs in groups of compatible services or 'containers'. In order to know the container even for those services that have not yet been part of an interaction, logistic regression is used to obtain a classifier that outputs the container given a services context. Extensive testing has shown that classifiers trained on the SPLIT$_{rel}$ based k-cut procedure score a correlation of more than 0.5 on average, which outperforms classifiers trained on cuts by benchmark algorithms by far. Also it was possible to document a quality improvement by 14% for recommendations that used information about a services container, even for a rather simple method of information incorporation.

6.2 Contribution

In this thesis we present a specialized and easily extensible simulation that can emulate hundreds of services in different rooms. It is equipped with a carefully modelled reward function. This reward function enables us to imitate user interaction with recommendation systems, which can not be accomplished by existing mere IoT device simulators. The reward function qualifies the simulation as testing ground for different context sensitive recommendation algorithms in an IoT service discovery context. To the best of our knowledge, such an evaluation tool has not been introduced before.

Furthermore we developed a framework to identify interoperable IoT services, without any preliminary knowledge about the constraints that influence interoperability. The system itself as well as the incorporation of the framework's outcome into existing bandit approaches is a novel approach, as the recommendation of multiple interoperable services in IoT context did not receive any scientific attention so far. Therefore ConComM enables contextual recommendation approaches that work for single service recommendation, to be applied to composite service recommendation.

6.3 Potential Future Work

In *ConComM Method Discussion* we talked about possible starting points for further evaluation, namely the better embedding of ConComM results into the bandit approach and the need for more feature engineering for the classifier. Both problems did not receive the appropriate attention due to the time limitations of a bachelors thesis. It would be very interesting to investigate whether different feature types such as radial basis functions could improve the quality of predictions and what happens if the unnecessary information is the dominant part of a feature vector.

So far we used logistic regression to obtain a classifier that maps a service to a probability distribution over the different possible containers. Hence We treated the problem of assigning service context to containers as multi class problem. An approach that would require a completely restructured framework is to treat the mapping *service context* $\rightarrow$ *container* as multi-labelled instead of multi class problem. Within this work we assumed that each service context was part of one and only one container. This might not be true in reality, as a service could be part of different interaction enabling containers. Logistic regression already allows more detailed statements, given that the probability distribution is a more detailed statement than the strict mapping described above. Therefore it would be interesting to investigate whether the probability distribution can - where necessary supported by more feature engineering towards that goal - already be used for multi label like statements. For example, a service context with a probability distribution scoring more than 30% for two discriminative classes could be assigned to both of them.

Given the long computation time for ConComM, another very important point of contact for future work is adaptation of ConComM towards an online and on demand framework. Currently the cut and classifier have to be newly computed whenever the interaction graph changes, which will permanently be the case in a real scenario. As our work aimed to explore whether a framework like ConComM would improve composite service recommendations, the incorporation of online and on demand cut and classification algorithms is a further interesting topic when it comes to considering a large scale application.

The promising results for composite service recommendation with LinUCB using ConComM information and the importance of IoT service recommendation within the next years alone are enough reason for further studies. Our investigation can only be a start to explore the full potential of adapted bandit approaches for composite IoT service recommendation. The development of the highly reliable k-cut procedure SPLIT$_{rel}$ with Min Max cut and the good performance of classifiers trained on top of these results are excellent outcomes on their own. Our results and the fascinating unexplored options described above motivate further investigations in this field.

A Algorithms and Derivations

A.1 LinUCB Hybrid Model Derived

In this section we derive the hybrid model for LinUCB [LCLS10]. The Loss function for the hybrid model $L_{t,a}$ at a specific time step t is given by

$$L_{t,a} = (\sum_s^t r_s - \sum_s^t z_{s,a_s}^T \beta - \sum_s^t x_{s,a_s}^T \theta)^2 + \beta_t^2 + \theta_t^2. \tag{A.1}$$

In order to obtain the optimal values for β_t and θ_t the Loss function (A.1) is seperately derived by both variables and each derivation set to 0. First the Loss is derived by β_t:

$$0 = \frac{\partial L_t}{\partial \beta_t} = \frac{\partial}{\partial \beta^*}\Big((\sum_s^t r_s - \sum_s^t z_{s,a_s}^T \beta - \sum_s^t x_{s,a_s}^T \theta)^2 + \beta_t^2 + \theta_t^2\Big)$$

$$0 = 2\sum_s^t z_{s,a_s}(\sum_s^t r_s - \sum_s^t z_{s,a_s}^T \beta - \sum_s^t x_{s,a_s}^T \theta) + 2\beta_t$$

$$\rightarrow \beta_t = (\sum_s^t z_{s,a_s} z_{s,a_s}^T + I)^{-1}(\sum_s^t z_{s,a_s} r_s - \sum_s^t z_{s,a_s} x_{s,a_s}^T \theta_t)$$

To improve readability the following abbreviations are introduced:

$$A_{a_t} = I + \sum_s^t x_{s,a_s} x_{s,a_s}^T$$

$$B_{a_t} = \sum_s^t x_{s,a_s} z_{s,a_s}^T$$

$$b_{a_t} = \sum_s^t r_s x_{s,a_s}$$

This gives a formulation for β_t in a more compact form:

$$\beta_t = (\sum_s^t z_{s,a_s} z_{s,a_s}^T + I)^{-1}(\sum_s^t z_{s,a_s} r_s - B_{a_t}^T \theta_t) \tag{A.2}$$

So far we have not derived a sufficient formulation for θ_t in this context. Therefore we derive the loss function (A.1) again, but this time by θ_t:

$$0 = \frac{\partial L_{t,a}}{\partial \theta_t} = \frac{\partial}{\partial \theta_t}\Big((\sum_s^t r_s - \sum_s^t z_{s,a_s}^T \beta - \sum_s^t x_{s,a_s}^T \theta)^2 + \beta_t^2 + \theta_t^2\Big)$$

$$0 = 2\sum_s^t x_{s,a_s}(\sum_s^t r_s - \sum_s^t z_{s,a_s}^T \beta - \sum_s^t x_{s,a_s}^T \theta) + 2\theta_t$$

$$\to \theta_t = (\sum_s^t x_{s,a_s} x_{s,a_s}^T + I)^{-1}(\sum_s^t x_{s,a_s} r_s - \sum_s^t x_{s,a_s} z_{s,a_s}^T \beta_t)$$

Using the compact notations from above, this yields

$$\theta_t = (A_{a_t})^{-1}(b_{a_t} - B_{a_t}\beta_t). \tag{A.3}$$

For each time step t, both equations (A.2) and (A.3) for β_t and θ_t have to be fulfilled. Plugging the newly gained formulation for θ_t in (A.2) leads to the term

$$\beta_t = (\sum_s^t z_{s,a_s} z_{s,a_s}^T + I)^{-1}(\sum_s^t z_{s,a_s} r_s - B_{a_t}^T A_{a_t}^{-1} b_{a_t} + B_{a_t}^T A_{a_t}^{-1} B_{a_t} \beta_t).$$

Solving this for β_t leads to

$$\beta_t = \Big(I - (\sum_s^t z_{s,a_s} z_{s,a_s}^T + I)^{-1} B_{a_t}^T A_{a_t}^{-1} B_{a_t}\Big)^{-1}\Big(\sum_s^t z_{s,a_s} z_{s,a_s}^T + I\Big)^{-1}\Big(\sum_s^t z_{s,a_s} r_s - B_{a_t}^T A_{a_t}^{-1} b_{a_t}\Big),$$

$$\beta_t = \Big(\Big(\sum_s^t z_{s,a_s} z_{s,a_s}^T + I\Big)\Big(I - (\sum_s^t z_{s,a_s} z_{s,a_s}^T + I)^{-1} B_{a_t}^T A_{a_t}^{-1} B_{a_t}\Big)\Big)^{-1}\Big(\sum_s^t z_{s,a_s} r_s - B_{a_t}^T A_{a_t}^{-1} b_{a_t}\Big),$$

$$\beta_t = \Big(\sum_s^t z_{s,a_s} z_{s,a_s}^T + I - B_{a_t}^T A_{a_t}^{-1} B_{a_t}\Big)^{-1}\Big(\sum_s^t z_{s,a_s} r_s - B_{a_t}^T A_{a_t}^{-1} b_{a_t}\Big).$$

Following the form that solutions for coefficients in linear regression usually take, we are aiming to express $\beta_t \in \mathbb{R}^k$ as $\beta_t = A_{0,t} b_{0,t}$ where $A_{0,t} \in \mathbb{R}^{k \times k}$ and $b_{0,t} \in \mathbb{R}^k$. We therefore define $A_{0,t}$ and $b_{0,t}$ as follows:

$$A_{0,t} = (\sum_s^t z_{s,a_s} z_{s,a_s}^T + I - B_{a_t}^T A_{a_t}^{-1} B_{a_t})^{-1},$$

$$b_{0,t} = (\sum_s^t z_{s,a_s} r_s - B_{a_t}^T A_{a_t}^{-1} b_{a_t}).$$

To update $A_{0,t}$ and $b_{0,t}$ incrementally, they are initialized with $A_{0,t} = I_k$ and $b_{0,t} = 0_k$. In each iteration the following calculations have to be performed:

$$A_{0,t} = A_{0,t-1} + z_{t,a_t} z_{t,a_t}^T - B_{a_t}^T A_{a_t}^{-1} B_{a_t} + B_{a_{t-1}}^T A_{a_{t-1}}^{-1} B_{a_{t-1}},$$

$$b_{0,t} = b_{0,t-1} + z_{t,a_t} r_t - B_{a_t}^T A_{a_t}^{-1} b_{a_t} + B_{a_{t-1}}^T A_{a_{t-1}}^{-1} b_{a_{t-1}},$$

where A_{a_t}, $B_{a_t}^T$ and b_{a_t} are the parameters associated with the arm having the hightest confidence bound in the t^{th} step. To reduce the amount of data that needs to be stored, it is advisable to add the terms belonging to $t-1$ first. Then the update of $A_{a_{t-1}}$, $B_{a_{t-1}}^T$ and $b_{a_{t-1}}$ can be done. As a final step, the new values A_{a_t}, $B_{a_t}^T$ and b_{a_t} can be used to finish the computation of $A_{0,t}$ and $b_{0,t}$.

A.2 Stoer Wagner Cut Methods

Algorithmus A.1 MinimumCutPhase for Stoer-Wagner Algorithm

```
procedure MINIMUMCUTPHASE($\mathcal{G}$, w, s)
    $\mathcal{S} \leftarrow s$
    while $\mathcal{S} \neq \mathcal{V}$ do
        Add the vertex, that is most tightly connected to $\mathcal{A}$ to $\mathcal{A}$
    end while
    shrink $\mathcal{G}$ by merging the two vertices last added to $\mathcal{A}$
    return cut-of-the-phase
end procedure
```

Algorithmus A.2 MinimumCut for Stoer-Wagner Algorithm

```
procedure MINIMUMCUT($\mathcal{G}$, w, s)
    minimumCut $\leftarrow \infty$
    while $|\mathcal{V}| > 1$ do
        cut-of-the-phase $\leftarrow$ MINIMUMCUTPHASE($\mathcal{G}$, w, s)
        if cut-of-the-phase $\leq$ minimumCut then
            minimumCut $\leftarrow$ cut-of-the-phase
        end if
    end while
end procedure
```

A.3 In Room Visibility

An interesting problem that occurred during the interrelation definitions in *To Take or not to Take: The Reward Cook Book* was the visibility issue. How do we trace whether a user can actually see the device that is in the same room. As a counterexample, imagine a L shaped room which is made out of two boxes. Now picture a device places in left part of the lower box forming the L's vertical line and a user standing up in the L's horizontal line-box. The user could can not see the device from its position, because the device is blocked by the rooms wall. How can we test if two objects in the room have an unblocked free view of each other? A similar situation is illustrated in figure A.1 for the user whose intersection points are denoted by x.

The way described here uses `Ray` objects and traces the ray from the user to the device as long as it stays within room boxes. A ray has an origin r_{orig} in the 2 dimensional space $\mathbb{R}^2$ as well as a direction r_{dir} that also is a vector in $\mathbb{R}^2$. The first step of the here proposed solution is to construct a ray between the users position, which we call start position $SP \in \mathbb{R}^2$ and the end position $EP \in \mathbb{R}^2$, where the device is located. The following process is based on the fact that a room is constructed from boxes, which are *concave* geometric objects. By definition, concave means that the connecting line between two arbitrary points within a concave area itself is fully contained in the area. Applied to our problem we can state: as long as we are following a ray *within* a single box, we can be sure a user could see any point on the ray because the ray itself is fully contained in the box. The basic idea is now to follow the ray from the start position within some box containing SP as far as possible towards the endpoint. As far as possible means that we stop when we would leave the box that has the furthest extend towards EP and contains SP. If we did not hit EP on the way, we set our current position CP to that point we have just reached. We repeat the described process with CP instead of SP until we either find EP or tested all boxes that contain CP without finding a box that would allow us to walk any further.

Having explained the general idea, the next paragraphs will explain the computation in deeper detail. Once the ray between EP and SP is constructed, the whole computation takes place *on* the ray. This transforms the previous 2D problem into a 1D problem. In a first step SP and EP are transformed into positions on the ray. The new coordinate system on the ray is denoted by $\mathcal{T} \in \mathbb{R}$, points in this system will be denoted by t in $[-\infty, \infty]$, with $t = 0$ being in the rays origin. The position t on the ray for an arbitrary point $p \in \mathbb{R}^2$ in an 2-dimensional space can easily be computed by solving the following equation for $t \in \mathcal{T}$:

$$p = r_{orig} + t \cdot r_{dir}. \tag{A.4}$$

This formulation also holds for rays in higher dimensions with $p, r_{orig}, r_{dir} \in \mathbb{R}^n$. Positive t's on the ray are such positions that are obtained by *adding* the direction that is multiplied by some scalar value to the origin. Negative t's are obtained by *subtracting* the scaled direction to the origin. Transforming SP and EP into the t-System using A.4, we obtain $t_s = 0$ and t_e.

By construction, the ray in the simulation has its origin in SP which results in $t_s = 0$. For other ray construction techniques it can not be ensured that transforming SP onto the ray will result in $t_s = 0$. Because of this, a second transformation is done to transform the rays coordinate system. The new system is denoted by $\mathcal{X}$. The $\mathcal{X}$-system is constructed to have its origin in t_s, with the the additional constraint that transforming t_e into x_e in the $\mathcal{X}$ system gives a positive number. This implies that increasing a value $x \in \mathcal{X}$ from $x = 0$, which is SP, will be like walking towards EP and having passed it it after exceeding x_e. How to transform a value in the $\mathcal{T}$-system into a value in the $\mathcal{X}$-system is given in A.5. Here, $d_{walking}$ is merely factor to adjust the sign in order to ensure that $x_e \in [0, \infty]$ is truely positive.

$$d_{walking} = \frac{(t_e - t_s)}{\|t_e - t_s\|_2},$$

$$x = (t - t_s) \cdot d_{walking}, \tag{A.5}$$

with $\| \cdot \|_2$ being the euclidean norm for n-dimensional vector spaces. Using A.5 to transform t_s and t_e into x_e and x_s results in the following values that have the desired properties:

$$x_s = 0,$$
$$x_e = (t_e - t_s) \cdot d_{walking} = \frac{(t_e - t_s)^2}{\|t_e - t_s\|_2} \geq 0.$$

Algorithm A.3 begins with assigning the transformation of SP to the current position x_currr in $\mathcal{X}$, which leads to $x_{curr} = 0$. For the scenario illustrated in figure A.1, $x_{curr} = 0$ for the users denoted by **x** and **y**. Now we will compute intersections of the ray with all boxes, the current position x_{curr} is currently in. All the intersection points are transformed into the $\mathcal{X}$-system, where then the biggest x value is chosen. Choosing this value equals choosing the box that allows us to follow the ray furthest towards the end point x_e. We denote this biggest value by x_{test}. The process of computing the intersections and getting the biggest x value is summarized as *getFurthestXPoint(ray, box)*. For the example obtaining x_{test} would work as follows: Both users are only contained in Box 1. Computing the intersection points with box 1 and transforming them into the $\mathcal{X}$ system for both users gives the possible points $\{x_{-1}, x_1\}$ for user **x** and

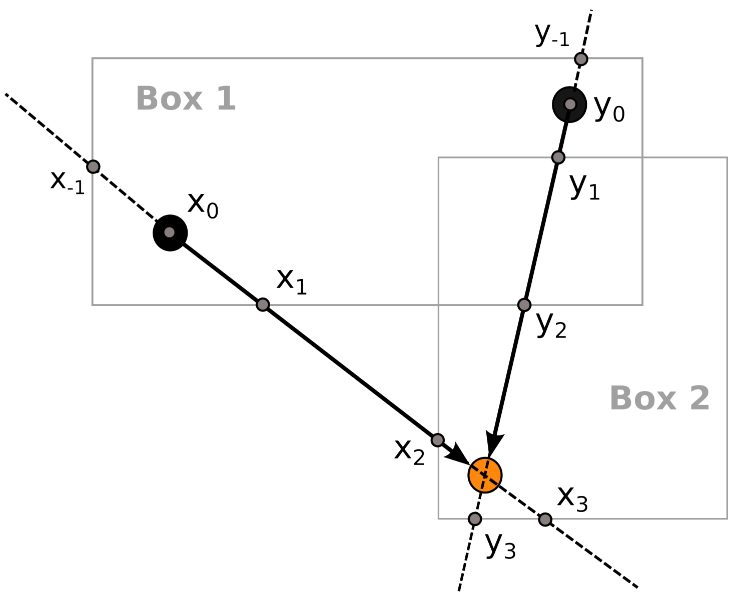


Figure A.1: Object Visibility within a Non Convex Room

Visibility of an IoTDevice (orange circle) for two different user positions (black circles) in a non-convex room that consists of Box 1 and Box 2. All points on the ray are given within the $\mathcal{X}$-coordinate system for the rays originating in x_0 and y_0 respectively, with their positive axis pointing towards the IoTDevice. Therefore, all intersection positions x and y with positive indices are scalars $[0, \infty]$.

$\{y_{-1}, y_2\}$ for user **y**. Selecting the point on the ray that is closest to the orange IoT device, results in x_{test} being assigned x_1 for user **x** and y_2 for user **y** respectively.

If $x_{test} \geq x_e$, x_e is within the same box and can therefore be seen from x_s. The search is concluded here and the result *visible* can be returned. If x_{test} is smaller than x_e, but strictly bigger than x_{curr}, x_{test} becomes the new x_{curr}. Afterwards, the next set of boxes that contain x_{curr} is tested for a bigger x_{test}. If, on the other hand, $x_{test} \leq x_{curr}$, we will not update x_{curr}. This happens when x_{curr} is the last position before leaving the room, which means it is impossible to follow the ray any further without leaving the room. In this case we did not find x_e before leaving; the result *invisible* will be returned. Applying this procedure to the example gives the following: In the first run x_{curr} is updated for both users, because $x_{test} < x_e$ and $x_{test} \geq x_{curr}$ for both cases. The next iteration leads to the possible candidates $\{x_{-1}, x_1\}$ for user **x** and $\{y_1, y_3\}$ for user **y**. x_{test} therefore becomes x_1 and y_3 respectively. For user **y** $x_{test} \geq x_e$ holds, which returns *visible*. For user **x**, on the other hand, $x_{test} \leq x_e$ and $x_{test} \leq x_{curr}$, which correctly returns *invisible*.

Algorithmus A.3 Test if Connection between two Points Stays within a Room

```
procedure RayStaysInside(room, ray, $EP$, $x_e$)
    $x_{curr} \leftarrow 0$
    $xCurrChanged \leftarrow$ true
    while room has unchecked boxes && $xCurrChanged$ do
        $xCurrChanged \leftarrow$ false
        box $\leftarrow$ getUncheckedBox(room)
        if $EP$ in box then
            return true
        else
            $x_{test} \leftarrow$ getFurthestXPoint(ray, box)
            if $x_{test} > x_{curr}$ then
                $xCurrChanged \leftarrow$ true
                $x_{curr} \leftarrow x_{test}$
                if $x_{test} \geq x_e$ then
                    return true
                end if
            end if
        end if
    end while
    return false
end procedure
```

Bibliography

[AB10] J.-Y. Audibert, S. Bubeck. “Best Arm Identification in Multi-Armed Bandits.” In: *COLT - 23th Conference on Learning Theory - 2010*. Haifa, Israel, June 2010, 13 p. URL: https://hal-enpc.archives-ouvertes.fr/hal-00654404 (cit. on p. 20).

[ACF02] P. Auer, N. Cesa-Bianchi, P. Fischer. “Finite-time Analysis of the Multiarmed Bandit Problem.” In: *Machine Learning* 47.2 (2002), pp. 235–256. URL: http://dx.doi.org/10.1023/A:1013689704352 (cit. on p. 20).

[ACFS02] P. Auer, N. Cesa-Bianchi, Y. Freund, R. E. Schapire. “The Nonstochastic Multiarmed Bandit Problem.” In: *SIAM Journal on Computing* 32.1 (2002), pp. 48–77. eprint: http://dx.doi.org/10.1137/S0097539701398375. URL: http://dx.doi.org/10.1137/S0097539701398375 (cit. on p. 22).

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, P. Steggles. “Towards a Better Understanding of Context and Context-Awareness.” In: *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*. HUC ’99. Karlsruhe, Germany: Springer-Verlag, 1999, pp. 304–307. URL: http://dl.acm.org/citation.cfm?id=647985.743843 (cit. on pp. 14, 15).

[AFB14] R. Allesiardo, R. Féraud, D. Bouneffouf. “A neural networks committee for the contextual bandit problem.” In: *International Conference on Neural Information Processing*. Springer. 2014, pp. 374–381 (cit. on p. 27).

[AIS93] R. Agrawal, T. Imieliński, A. Swami. “Mining association rules between sets of items in large databases.” In: *Acm sigmod record*. Vol. 22. 2. ACM. 1993, pp. 207–216 (cit. on p. 71).

[AT11] G. Adomavicius, A. Tuzhilin. “Context-aware recommender systems.” In: *Recommender systems handbook*. Springer, 2011, pp. 217–253 (cit. on pp. 20, 22).

[BB+57] G. M. Beal, J. M. Bohlen, et al. *The diffusion process*. Agricultural Experiment Station, Iowa State College, 1957 (cit. on pp. 93, 94).

[BBG12] D. Bouneffouf, A. Bouzeghoub, A. L. Gançarski. "A Contextual-Bandit Algorithm for Mobile Context-Aware Recommender System." In: *Neural Information Processing: 19th International Conference, ICONIP 2012, Doha, Qatar, November 12-15, 2012, Proceedings, Part III*. Ed. by T. Huang, Z. Zeng, C. Li, C. S. Leung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 324–331. URL: http://dx.doi.org/10.1007/978-3-642-34487-9_40 (cit. on p. 27).

[Bis06] C. M. Bishop. "Pattern recognition." In: *Machine Learning* 128 (2006) (cit. on p. 38).

[BJ13] G. Bonnin, D. Jannach. "A comparison of playlist generation strategies for music recommendation and a new baseline scheme." In: *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*. 2013 (cit. on p. 71).

[Car16] D. Carlson. *Ambient Dynamix*. 2016. URL: http://ambientdynamix.org/ (cit. on pp. 9, 13).

[CAS13] D. Carlson, B. Altakrouri, A. Schrader. "An ad-hoc smart gateway platform for the web of things." In: *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE. 2013, pp. 619–625 (cit. on p. 17).

[CLCL14] K.-C. Chou, H.-T. Lin, C.-K. Chiang, C.-J. Lu. "Pseudo-reward Algorithms for Contextual Bandits with Linear Payoff Functions." In: *ACML*. 2014 (cit. on pp. 27, 28).

[CMBP04] D. Cao, O. T. Masoud, D. Boley, N. Papanikolopoulos. "Online motion classification using support vector machines." In: *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*. Vol. 3. IEEE. 2004, pp. 2291–2296 (cit. on p. 38).

[CMTJ12] S. Chen, J. L. Moore, D. Turnbull, T. Joachims. "Playlist prediction via metric embedding." In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 714–722 (cit. on p. 71).

[CS14] D. Carlson, A. Schrader. "Ambient ocean: A web search engine for context-aware smart resource discovery." In: *Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing (CPSCom), IEEE*. IEEE. 2014, pp. 177–184 (cit. on pp. 18, 19).

[CW91] C.-K. Cheng, Y.-C. Wei. "An improved two-way partitioning algorithm with stable performance [VLSI]." In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 10.12 (1991), pp. 1502–1511 (cit. on p. 31).

[DAS01] A. K. Dey, G. D. Abowd, D. Salber. "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications." In: *Hum.-Comput. Interact.* 16.2 (Dec. 2001), pp. 97–166. URL: http://dx.doi.org/10.1207/S15327051HCI16234_02 (cit. on p. 14).

[DHZ+01] C. H. Q. Ding, X. He, H. Zha, M. Gu, H. D. Simon. "A Min-max Cut Algorithm for Graph Partitioning and Data Clustering." In: *Proceedings of the 2001 IEEE International Conference on Data Mining*. ICDM '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 107–114. URL: http://dl.acm.org/citation.cfm?id=645496.658058 (cit. on pp. 31, 33, 34).

[DJP+92] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, M. Yannakakis. "The Complexity of Multiway Cuts (Extended Abstract)." In: *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*. STOC '92. Victoria, British Columbia, Canada: ACM, 1992, pp. 241–251. URL: http://doi.acm.org/10.1145/129712.129736 (cit. on p. 31).

[DMD+10] A. Dohr, R. Modre-Osprian, M. Drobics, D. Hayn, G. Schreier. "The Internet of Things for Ambient Assisted Living." In: *ITNG* 10 (2010), pp. 804–809 (cit. on p. 15).

[EK72] J. Edmonds, R. M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems." In: *J. ACM* 19.2 (Apr. 1972), pp. 248–264. URL: http://doi.acm.org/10.1145/321694.321699 (cit. on p. 30).

[FHT01] J. Friedman, T. Hastie, R. Tibshirani. *The elements of statistical learning*. Vol. 1. Springer series in statistics Springer, Berlin, 2001 (cit. on pp. 35, 36).

[FK15] I. Filippidou, Y. Kotidis. "Online and on-demand partitioning of streaming graphs." In: *Big Data (Big Data), 2015 IEEE International Conference on*. IEEE. 2015, pp. 4–13 (cit. on p. 35).

[GH88] O. Goldschmidt, D. S. Hochbaum. "Polynomial algorithm for the k-cut problem." In: *FOCS*. 1988, pp. 444–451 (cit. on pp. 31, 76).

[GM16] Google. *Geiger Counter World Map*. 2016. URL: http://www.gmcmap.com/ (cit. on p. 16).

[Goo16] Google. *Google – AdWords Help*. 2016. URL: https://support.google.com/adwords/answer/2615875?hl=en&from=6305&rd=2 (cit. on p. 61).

[HK92] L. Hagen, A. B. Kahng. "New spectral methods for ratio cut partitioning and clustering." In: *IEEE transactions on computer-aided design of integrated circuits and systems* 11.9 (1992), pp. 1074–1085 (cit. on p. 31).

[HMB12] N. Hariri, B. Mobasher, R. Burke. "Context-aware music recommendation based on latenttopic sequential patterns." In: *Proceedings of the sixth ACM conference on Recommender systems*. ACM. 2012, pp. 131–138 (cit. on p. 71).

[IIOT16] Intel. *A guide to the Internet of Things*. 2016. URL: http://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html (cit. on pp. 9, 13, 19).

[Kar93] D. R. Karger. "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm." In: *SODA*. Vol. 93. 1993, pp. 21–30 (cit. on p. 31).

[KS96] D. R. Karger, C. Stein. "A New Approach to the Minimum Cut Problem." In: *J. ACM* 43.4 (July 1996), pp. 601–640. URL: http://doi.acm.org/10.1145/234533.234534 (cit. on p. 31).

[LCLS10] L. Li, W. Chu, J. Langford, R. E. Schapire. "A Contextual-bandit Approach to Personalized News Article Recommendation." In: *Proceedings of the 19th International Conference on World Wide Web*. WWW '10. Raleigh, North Carolina, USA: ACM, 2010, pp. 661–670. URL: http://doi.acm.org/10.1145/1772690.1772758 (cit. on pp. 24–26, 121).

[Li16] J. Li. *Logistic Regression*. 2016. URL: http://sites.stat.psu.edu/~jiali/course/stat597e/notes2/logit.pdf (cit. on p. 36).

[LM] D. Lemire, A. Maclachlan. "Slope One Predictors for Online Rating-Based Collaborative Filtering." In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. Chap. 43, pp. 471–475. eprint: http://epubs.siam.org/doi/pdf/10.1137/1.9781611972757.43. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611972757.43 (cit. on p. 20).

[LZ08] J. Langford, T. Zhang. "The Epoch-Greedy Algorithm for Multi-armed Bandits with Side Information." In: *Advances in Neural Information Processing Systems 20*. Ed. by J. C. Platt, D. Koller, Y. Singer, S. T. Roweis. Curran Associates, Inc., 2008, pp. 817–824. URL: http://papers.nips.cc/paper/3178-the-epoch-greedy-algorithm-for-multi-armed-bandits-with-side-information.pdf (cit. on pp. 22, 23).

[ML11] B. Mcfee, G. Lanckriet. *THE NATURAL LANGUAGE OF PLAYLISTS*. 2011 (cit. on p. 71).

[MRTM12] D. K. Mahajan, R. Rastogi, C. Tiwari, A. Mitra. “LogUCB: An Explore-exploit Algorithm for Comments Recommendation.” In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. CIKM ’12. Maui, Hawaii, USA: ACM, 2012, pp. 6–15. URL: http://doi.acm.org/10.1145/2396761.2396767 (cit. on pp. 21, 27).

[Nor98] J. R. Norris. *Markov chains*. 2. Cambridge university press, 1998 (cit. on p. 71).

[Pow03] D. M. Powers. “Recall & Precision versus The Bookmaker.” In: International Conference on Cognitive Science. 2003 (cit. on p. 41).

[Pow07] D. Powers. “Evaluation: From Precision, Recall and F Factor to ROC, Informedness, Markedness & Correaltion.” In: *Sch. Informatics Eng. Flinders* (2007) (cit. on pp. 40, 41).

[PZCG14] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos. “Context aware computing for the internet of things: A survey.” In: *IEEE Communications Surveys & Tutorials* 16.1 (2014), pp. 414–454 (cit. on pp. 14, 15).

[R15] W. R. *Graph Cuts Approach to the Problems of Image Segmentation*. 2015. URL: http://www.coe.utah.edu/~cs7640/readings/graph_cuts_intro.pdf (cit. on p. 30).

[Rey13] L. Reyzin. *New Algorithms for Contextual Bandits*. 2013. URL: http://www.levreyzin.com/presentations/CMU_bandits.pdf (cit. on pp. 21, 22).

[Rob85] H. Robbins. “Some aspects of the sequential design of experiments.” In: *Herbert Robbins Selected Papers*. Springer, 1985, pp. 169–177 (cit. on p. 20).

[RPG+13] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, S. Haridi. “Ja-be-ja: A distributed algorithm for balanced graph partitioning.” In: (2013) (cit. on p. 35).

[RRS11] F. Ricci, L. Rokach, B. Shapira. *Introduction to recommender systems handbook*. Springer, 2011 (cit. on p. 20).

[SBG99] A. Schmidt, M. Beigl, H.-W. Gellersen. “There is more to context than location.” In: *Computers & Graphics* 23.6 (1999), pp. 893–901 (cit. on p. 14).

[SL09] M. Sokolova, G. Lapalme. “A systematic analysis of performance measures for classification tasks.” In: *Information Processing & Management* 45.4 (2009), pp. 427–437 (cit. on pp. 39, 41, 42).

[SM00] J. Shi, J. Malik. “Normalized cuts and image segmentation.” In: *IEEE Transactions on pattern analysis and machine intelligence* 22.8 (2000), pp. 888–905 (cit. on p. 31).

[SU05] I. Strategy, P. Unit. “ITU Internet Reports 2005: The internet of things.” In: *Geneva: International Telecommunication Union (ITU)* (2005) (cit. on p. 15).

[SV95] H. Saran, V. V. Vazirani. “Finding k cuts within twice the optimal.” In: *SIAM Journal on Computing* 24.1 (1995), pp. 101–108 (cit. on pp. 31, 78, 84).

[SW97] M. Stoer, F. Wagner. “A Simple Min-cut Algorithm.” In: *J. ACM* 44.4 (July 1997), pp. 585–591. URL: http://doi.acm.org/10.1145/263867.263872 (cit. on pp. 30, 32).

[Tou15] M. Toussaint. *Introduction to Machine Learning*. 2015. URL: https://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/15-MachineLearning/15-MachineLearning-script.pdf (cit. on pp. 35, 36).

[TR15] S. Tracà, C. Rudin. “Regulating greed over time.” In: *arXiv preprint arXiv:1505.05629* (2015) (cit. on p. 27).

[Tru13] R. J. Trudeau. *Introduction to graph theory*. Courier Corporation, 2013 (cit. on p. 30).

[Wei16a] E. W. Weisstein. *Adjacency Matrix*. 2016. URL: http://mathworld.wolfram.com/AdjacencyMatrix.html (cit. on p. 33).

[Wei16b] E. W. Weisstein. *Degree Matrix*. 2016. URL: http://mathworld.wolfram.com/DegreeMatrix.html (cit. on p. 33).

[Wei16c] E. W. Weisstein. *Laplacian Matrix*. 2016. URL: http://mathworld.wolfram.com/LaplacianMatrix.html (cit. on p. 33).

[Wei16d] E. W. Weisstein. *Vertex Degree*. 2016. URL: http://mathworld.wolfram.com/VertexDegree.html (cit. on p. 33).

[Wei91] M. Weiser. “The computer for the 21st century.” In: *Scientific american* 265.3 (1991), pp. 94–104 (cit. on p. 14).

[WSCR16] N. Wanigasekara, J. Schmalfuss, D. Carlson, D. S. Rosenblum. “A Bandit Approach for Intelligent IoT Service Composition across Heterogeneous Smart Spaces.” In: (2016) (cit. on pp. 11, 17, 18, 28).

[YHG12] Y. Yue, S. A. Hong, C. Guestrin. “Hierarchical exploration for accelerating contextual bandits.” In: *arXiv preprint arXiv:1206.6454* (2012) (cit. on p. 27).

[YZ06] Y. Ying, D.-X. Zhou. “Online regularized classification algorithms.” In: *IEEE Transactions on Information Theory* 52.11 (2006), pp. 4775–4788 (cit. on p. 38).

[ZARP15] N. Zhao, M. Aldrich, C. F. Reinhart, J. A. Paradiso. “A Multidimensional Continuous Contextual Lighting Control System Using Google Glass.” In: *Proceedings of the 2Nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*. BuildSys ’15. Seoul, South Korea: ACM, 2015, pp. 235–244. URL: http://doi.acm.org/10.1145/2821650.2821673 (cit. on p. 58).

[ZGC11] D. Zeng, S. Guo, Z. Cheng. “The web of things: A survey.” In: *Journal of Communications* 6.6 (2011), pp. 424–438 (cit. on p. 16).

All links were last followed on November 10, 2016.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature