Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

TU WIEN Informatics

Large-scale multi-agent meeting scheduling using an altruistic matching heuristic

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Logic and Computation

eingereicht von

Florian Wiedemair, BSc
Matrikelnummer 01525933

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Nysret Musliu
Mitwirkung: Panayiotis Danassis, MSc
Prof. Boi Faltings

Wien, 3. Februar 2022

Florian Wiedemair | Nysret Musliu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Informatics

Large-scale multi-agent meeting scheduling using an altruistic matching heuristic

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Logic and Computation

by

Florian Wiedemair, BSc
Registration Number 01525933

to the Faculty of Informatics

at the TU Wien

Advisor: Priv.-Doz. Dr. Nysret Musliu
Assistance: Panayiotis Danassis, MSc
Prof. Boi Faltings

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Vienna, 3rd February, 2022

Florian Wiedemair   Nysret Musliu

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Florian Wiedemair, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Februar 2022

Florian Wiedemair

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Zuallererst möchte ich mich bei meinem Betreuer Priv.-Doz. Dr. Nysret Musliu sowie bei Panayiotis Danassis, MSc und Prof. Boi Faltings bedanken. Ihre durchgehende Unterstützung, Geduld und Ihre Orientierung haben mir massiv beim Bewältigen dieses Projekts geholfen.

Ich will auch meiner Familie und meinen Freunden danken. Sie haben meine Bildung ermöglicht, mich durch mein Studium geführt und mich in all der Zeit unterstützt.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

First and foremost I would like to thank my advisor Priv.-Doz. Dr. Nysret Musliu as well as Panayiotis Danassis, MSc and Prof. Boi Faltings. Their continuous support, patience and guidance helped me massively throughout this project.

I also want to thank my family and friends who enabled my education, guided my studies and supported me through all this time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Das Problem eine große Anzahl an Treffen für mehrere Parteien zu planen tritt in vielen Szenarien auf. Dies geht von der Organisation privater Treffen über Social Media bis hin zu Geschäftsterminen oder dem Erstellen eines Stundenplans in der Schule. Normalerweise ist es nötig für jedes dieser Szenarios eine große Anzahl an Bedingungen zu berücksichtigen. Dies macht das Lösen solcher Probleme für gewöhnlich schwer.

Nehmen wir als Beispiel an, man müsste einen Stundenplan für eine Schule erstellen. Abgesehen von allgemeinen Bedingungen, wie der Tatsache, dass jede Klasse ihre Unterrichtsstunden mit passendem Lehrpersonal zugewiesen bekommen muss, müsste man beim Erstellen eines solchen Plans auch individuelle Präferenzen der Lehrer oder Schüler, sowie die Verfügbarkeit der Klassenräume, berücksichtigen. Für ein Individuum würde eine solche Aufgabe schnell überwältigend werden. Selbst unter Zuhilfenahme von Computern kann es immer noch schwer sein, exakte Lösungen für Probleme dieser Art zu finden. Aus diesem Grund fokussieren wir uns in dieser Arbeit auf einen heuristischen Zugang.

Wir gehen das Problem an indem wir zuerst eine Formulierung angeben, welche geeignet ist, um eine große Zahl an Terminplanungsproblemen (meeting scheduling problems) zu modellieren. Die Formulierung ist durch viele Beispiele ähnlicher Problemstellungen aus der Literatur motiviert.

Als nächstes bringen wir einen verteilten Algorithmus vor, welcher solche Probleme heuristisch lösen kann. Er ist mithilfe eines Multiagentensystems implementiert um die teilnehmenden Parteien zu simulieren. Der Algorithmus hat den Vorteil, dass er gut auf größere Probleminstanzen skaliert, gleichzeitig gute Resultate liefert und dabei einen gewissen Grad an Privatsphäre ermöglicht. Um dies praktisch zu testen, wurde ein Instanzengenerator entwickelt. Die so generierten Instanzen basieren auf Daten aus Studien zu Geschäftsterminen in Amerika. Das bedeutet, dass gewisse Schlüsseleigenschaften, wie Beispielsweise die Länge der Termine, aus Verteilungen entnommen werden, welche auf den erwähnten Daten basieren.

Die Resultate der Auswertung unseres Algorithmus auf den generierten Instanzen sind vielversprechend. Sie zeigen, dass unser Algorithmus gute Ergebnisse erzielen kann und gleichzeitig auch in der Praxis gut auf größere Probleminstanzen skaliert. Die Resultate erlauben auch eine genauere empirische Analyse des Algorithmus selbst. In diesem Kontext tragen wir auch Methoden vor, um den Algorithmus an spezielle Problemstellungen anzupassen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

The problem of scheduling a large number of meetings for multiple parties can be found in many scenarios from organizing private events via social media to business meetings or timetabling in school. Usually each of those scenarios requires to take into consideration a large number and variety of constraints, which usually makes it hard to solve such problems.

Take, for example, the problem of creating a class timetable for a school. Apart from general constraints, such as that every class needs to have all of their respective lessons with the right teacher scheduled, one would also have to take into consideration individual preferences of teachers or students, availability of the classrooms etc.. This can quickly become insurmountable for an individual to handle. Even when using computers, it can still be hard to find exact solutions for problems like that. Therefore, we focus on a heuristic approach in this thesis.

We start to tackle this issue by first providing a formulation which is suitable to model a large number of meeting scheduling problems. The formulation is motivated by many examples of similar problem statements in the literature.

Next, we propose a distributed algorithm to tackle the meeting scheduling problem using a heuristic approach. It is implemented through the means of a multi-agent system simulating the different participating parties in an instance. The algorithm has the benefit of scaling well to larger instances while still providing decent privacy and social welfare. In order to test that, an instance generator is further developed. The generated instances are based on data from real-life studies on meetings in corporate America. Meaning that key properties, such as the meeting length, are generated from distributions based on that data.

The results of an evaluation of the aforementioned algorithm using the generated instances are promising. They show a high social-welfare for our algorithm, while it still scales well for larger problem instances. The results also allow for a deeper empirical analysis of the various aspects of the algorithm. In that context we also propose methods on how to optimize the algorithm for specific tasks.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Aims of this Thesis 2
1.2 Contributions 2
1.3 Structure 3

2 Related Work **5**
2.1 ALMA 7
2.2 ALMA-Learning 8

3 Problem Formulation **11**
3.1 Problem Description 11
3.2 Complexity 14
3.3 ALMA 16
3.4 Further Thoughts and Considerations 17

4 Algorithm **19**
4.1 Setup and Initialization 19
4.2 Strategy 22
4.3 Privacy 28
4.4 Convergence and Runtime Considerations 30

5 Instance Generation **33**
5.1 Assigning Length 33
5.2 Assigning Attendees 34
5.3 Assigning Preferences 35

6 Evaluation **39**
6.1 Reference Algorithms 39

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.2 Testing 42

7 Conclusion **55**

Bibliography **57**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

The problem of scheduling a large number of meetings for multiple parties can be found in many scenarios from organizing private events via social media to planning business meetings or timetabling in school. For each of these different applications there is usually a large number of individual constraints to take into account. The difficulties of tackling the meeting scheduling problem therefore lie in finding a common framework that is suitable for most varieties of the problem and then solving the problem by taking all these constraints into consideration simultaneously.

Take, for example, the problem of having to create a class timetable for a school. First, the creator would need to make sure that every class has their respective lessons scheduled with the right teacher. In addition to that, one also would have to take into consideration the availability of classrooms at any given time. These are some of the more strict constraints that would apply to such a problem.

In addition to the constraints mentioned above, class timetables usually follow some less strict rules as well. For example, (especially in lower-grade) lessons in the morning are generally preferred over lessons in the afternoon. Also, some teachers might have individual preferences for certain weekdays. Altogether, even for a smaller school, that would amount to a large number of constraints to handle.

In order to be able to properly put together such a schedule, someone would need to take into account all of these constraints. Needless to say, for larger instances this problem would quickly become insurmountable for an individual to handle without significantly neglecting some constraints.

Clearly, we can use computers to automate or at least assist with this task. This way we can significantly speed up the process. To do this, a centralized approach is often easiest, where all preference data is collected in one central entity and then the schedule is built there. While finding optimal solutions for large meeting scheduling problems is very time consuming, we can find appropriate solutions within a reasonable timeframe. This,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

centralized approach, however, usually requires each user to expose their preferences to the other users or at least some central entity, which leads to a possibly unwanted loss of privacy. In order to tackle this issue many distributed algorithms have been proposed in the past.

Most of these algorithms can only handle smaller meeting scheduling problems without exceeding a reasonable runtime or neglecting user preferences significantly. This leads to the need for a decentralized algorithm that keeps the exposure of personal data or preferences to a minimum while still achieving appropriate solutions within a reasonable timeframe.

In [DFRF19] the authors introduced a distributed algorithm for the assignment problem. The mentioned algorithm fulfills all criteria above in that it does not explicitly share data between the agents while still achieving good results for the problems given in the paper. The ideas of that algorithm can now be used to build a new algorithm for the meeting scheduling problem.

1.1 Aims of this Thesis

This thesis aims to give insight into the (distributed) meeting scheduling problem by first giving a short overview of related works and also providing a general formulation for the problem that can be used to describe a wide variety of different meeting scheduling problems.

Secondly, we aim to describe and implement an algorithm using a multi-agent system that uses the general idea of [DFRF19] to heuristically solve instances of the meeting scheduling problem. We also want to look at possible alternations and improvements for this particular algorithm.

Lastly, we aim to test and evaluate said algorithm by using it to solve instances inspired by real-life situations. Since it is hard to find data in the exact format needed, we will also aim to provide an instance generator that creates random instances based on real-world observations. To properly evaluate our algorithm we will also implement other algorithms for comparison. We will use those to not only look into the quality of our results, but also further investigate performances with respect to runtime, fairness and privacy.

1.2 Contributions

The first contribution of this thesis is our formulation of the meeting scheduling problem, which has the benefit that it lets us model a variety of different real-life problems and also enables a lot of personalization through its highly customizable soft constraints. In addition to that, the formulation is held very general which also allows for further constraints to be added easily in order to adapt to more specific problems.

The main contribution of this thesis is focused around the distributed algorithm to heuristically solve meeting scheduling problems specified by our formulation. It is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

implemented through the means of a multi-agent system simulating the different users in an instance. The algorithm is based on the novel altruistic approaches described in [DFRF19] and [DF20]. It has the benefit of scaling well to larger instances while still providing decent privacy and social welfare. Furthermore, it is designed in a way that allows users a high degree of individualization for their preferences. This not only gives users the ability to encode meeting- or time-specific constraints but also gives them the possibility to transmit obscured preferences to improve privacy with little downside.

An instance generator is further developed to test the algorithm on a diverse array of instances. Those are based on data from real-life studies on meetings in corporate America. Meaning that key properties, such as the meeting length, are generated from distributions based on that data. It also simulates the forming of groups for participants of meetings. At the same time, the generator allows to set varying problem sizes and consequently degrees of difficulty.

The evaluation of the aforementioned algorithm against a number of other approaches on those instances underlines the strengths of the proposed algorithm. The results allow for a deeper empirical analysis of the various aspects of the algorithm. In that context we also propose methods on how to optimize the algorithm for specific tasks. Some of these results were also published at the IJCAI Conference 2021 in [DWF21].

1.3 Structure

This thesis is structured as follows. In Chapter 2 we look into literature related to the meeting scheduling problem. We also introduce the altruistic matching algorithm in that chapter. In Chapter 3 we introduce our formulation for the meeting scheduling problem as a constraint satisfaction problem. Furthermore we give a brief analysis of the formulation and draw first parallels to the assignment problem. Chapter 4 is dedicated to the detailed description of our algorithm to tackle the meeting scheduling problem. We also look into certain aspects of that algorithm such as privacy and runtime. In Chapter 5 we describe how the instances, that are later used to test the aforementioned algorithm, are generated. In Chapter 6 our algorithm is then evaluated on said instances. The results are then used to compare our algorithm with other approaches. Chapter 7 gives a final summary of the thesis and its results.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Related Work

Since computers are clearly a helpful tool to tackle the meeting scheduling problem, or scheduling problems in general, they have been studied in this context since the earlier days of computing. An early example can be found in [GM86]. More recently Gelain et al. [GPR+17] have published a paper considering a centralized approach. The authors apply local search techniques to try and optimize the soft constraints of the meeting scheduling problem.

The research on solvers for meeting scheduling problems quickly shifted its focus on distributed algorithms. In [SD91] the authors describe the meeting scheduling problem as "inherently distributed", due to the need to consider individual preferences and privacy. In [GS96] the authors claim that their experiments show that keeping certain information private in meeting scheduling can even lead to more stable results.

Most state-of-the-art methods for distributed meeting scheduling involve multi-agent systems. Multi-agent systems are distributed computing systems where each individual node is represented by an intelligent i.e. autonomous agent. These agents work towards a, possibly common, goal. In the case of meeting scheduling, for example, the agents would negotiate with each other in order to yield a satisfying schedule. Intelligence in the context of multi-agent systems usually refers to an agents ability to observe its environment and autonomously act within it. The authors of [DKJ18] define an agent as "an entity which is placed in an environment and senses different parameters that are used to make a decision based on the goal of the entity. The entity performs the necessary action on the environment based on this decision." The ability to learn and self-improve is common in this context as well, however, it is not a necessary requirement for an agent.

One example of an algorithm which uses a multi-agent system to solve a meeting scheduling problem can be found in [FFRW02]. In that paper each agent represents a user and knows the personal preferences of that user for one meeting. The agents interact

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

with each other in that they propose a possible meeting time, given their preferences, and then negotiate with each other in rounds in order to find the optimal time for that meeting to take place. After each proposal, the agents record the responses in order to gain as much knowledge as possible and improve their actions for future proposals. While this allows the agents to keep a certain amount of privacy, it usually takes a long time for a large number of agents to finish these negotiations. Another downside of the algorithm is that it can only find the optimal scheduling time for a single meeting. Therefore if one wants to schedule multiple meetings at once(e.g. for timetabling) one would have to repeatedly perform the algorithm on a given order of meetings. Depending on the order the meetings are processed this most likely would not yield the overall optimal result. A similar approach can be found in [dMGS18].

The authors of [BH07] improved on this idea by also allowing dynamic meeting scheduling, i.e. allowing meetings to be rescheduled. Therefore, if there is a conflict between two meetings the one with a lower utility value would be dropped and scheduled for another time. To make this possible, said paper also introduces preferences between the meetings, i.e. each user has to give a preference order for the meetings to be scheduled. Furthermore, the authors also claim that this algorithm minimizes the number of messages passed between the different agents. This algorithm also serves as a base for a reference implementation in Section 6.1.

Other modern approaches also attempt to further minimize the number of necessary messages to pass before reaching an agreement on a timeslot for a meeting by using machine learning methods to process knowledge about previous proposals and thereby infer the preferences of a user. In [ZC09, NS20] specifically Bayesian Networks were used to reason about the availability of possible meeting attendees. This can then be used to improve the negotiation strategy. In [NS20] the authors were able to schedule events with up to 2000 participants. The algorithm in [NS20] also allowed to define a subset of attendees that had to attend an event and then defined the attendance of other possible participants as a soft constraint, thereby making it possible to work with a quorum.

The meeting scheduling problem can also be formulated as a distributed constraint satisfaction problem (DCSP). As the name suggests, DCSPs are constraint satisfaction problems in which variables and constraints are distributed, usually in a system of autonomous agents. The formalism is described by Yokoo et. al. in [YDIK98]. In that paper the authors identify the cost of communication between nodes in the system as well as privacy concerns as the main research issues of DCSPs. By now, there are various established strategies that can then be used to solve DCSPs. This can yield the optimal solution while preserving privacy to a certain extend, however, as with CSPs in general, it does usually not scale well for meeting scheduling problems. Examples of this can be found in [MTB+04, ODF12].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.1 ALMA

As many parts of the design of the core algorithm for this thesis are heavily inspired by the altruistic matching algorithm (short ALMA) as described in [DFRF19], we want to briefly lay out the general idea of that algorithm in this section. In order to explain the idea properly, we first need to introduce the assignment problem, which is addressed in that paper.

Given is a set of agents $\mathcal{A} = \{A_1, \ldots, A_n\}$ and a set of resources $\mathcal{R} = \{R_1, \ldots, R_m\}$. Each agent A_i is interested in a subset of the total resources $\mathcal{R}_i$. That interest is then quantified by assigning a utility in $[0, 1]$ to each pair (A_i, R_j), where $R_j \in \mathcal{R}_i$. Each agent can acquire at most one resource. The paper then considers the problem of maximizing social welfare i.e. maximizing the sum of utilities.

In order to solve this problem, each agent first sorts its subset of resources descending by utility. Then, the agents try to acquire resources in a round-based fashion, each starting with the resource with the highest utility for themselves. Inspired by the concept of altruism the main idea of the ALMA heuristic is that if two agents try to obtain the same resource in a round, each of them considers its alternative options. If the alternatives yield a utility that is close to the current resource, the agent would back-off with a high probability. More specifically, each agent computes its personal loss given as the normalized sum of the differences between the utility of the current resource and the next k resources in the sorted list i.e.

$$\text{loss}_n^i = \frac{1}{k} \sum_{j=i+1}^{i+k} u_n(r_i) - u_n(r_j)$$

where r_i denotes the i-th resource in the sorted list of resources for the current agent n and u_n is the utility function.

This loss is then used as an input for a non-increasing function mapping to [0,1] in order to yield the back-off probability. If the agent decides to back-off of the currently considered resource, it will start monitoring the next resource on its list in the following round. If the motioning shows that the resource is still available it will try to acquire it in the round after that. Otherwise, the agent would instead try to acquire the same resource again in the following round. This process is repeated until a complete matching is found.

We will see in Sections 3.3 and 4.2 how this algorithm can be applied to the meeting scheduling problem. However, even without having a detailed formulation of the meeting scheduling problem to work with, we can already see similarities between it and the assignment problem. More specifically, how scheduling meetings for specific timeslots could be considered a form of assignment and how ALMA could be used in a similar way for that task.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2 ALMA-Learning

While ALMA has been shown to yield good results in [DFRF19], one issue of the algorithm is that it does not account for the actions of other (possibly competing) agents. For example, when using the loss function as given in Section 2.1, an agent does not know if the next k considered resources are actually going to be available or not. This can lead to potentially sub-optimal results. Similarly can the results be sub-optimal if two agents try to compete for the same resource at the start while potentially good alternative resources get acquired by other agents in the meantime. In [DF20] the authors found a way to tackle these issues by letting the agents learn certain parameters throughout multiple iterations of the ALMA algorithm.

Specifically, they learn an improved loss function as well as which resource to pick at the beginning of the algorithm. Learning the latter is achieved by introducing an expected reward for each resource in order to determine the best resource to start with. If that expected reward is not often met in the following rounds, the expected reward will be decreased. In order to improve the loss function from the original ALMA (see Section 2.1) the authors adapt the expected loss of backing-off a given resource if the high-utility alternative(s) turn out to be occupied already. More specifically, the following variables are introduced:

- $rewardHistory[R][L]$: This 2D array stores for each starting resource r the L most recent rewards received by the agent when starting with resource r. At Initialization the utility of a resource $u_n(r)$ is added to the corresponding rewardHistory.
- $reward[R]$: This 1D array estimates the expected reward for a given resource r by storing the mean of the L most recent received rewards for that resource. I.e. $reward(r) \leftarrow rewardHistory[r].mean()$. The starting resource r_{start} is then determined as $\text{argmax}_{r \in R}\, reward(r)$.
- $loss[R]$: This 1D array estimates the expected loss for passing on a given resource r. The array is initialized as the loss function given in Section 2.1. With each iteration the loss array entry for the starting resource r_{start} is updated according to the following rule:

  $$loss(r_{start}) \leftarrow (1-\alpha)loss(r_{start}) + \alpha(u(r_{start}) - u(r_{won})),$$

  where r_{won} is the ultimately acquired resource and α the learning rate. If no resource is acquired by the agent, $u(r_{won})$ will be set to 0.

Algorithm 2.1 gives an example code of what the implementation of ALMA-Learning would look like, given that we already have an implementation of ALMA. If we look at the code, we can see that once we have an adaption of ALMA for the meeting scheduling problem, we can directly apply ALMA-Learning as described here. Therefore, we will focus on finding such an adaption.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 2.1: Example implementation of ALMA-learning.

```
1  initialize rewardHistory, reward, loss;
2  r_start = argmax_r reward[r];
3  foreach i ∈ 1...T do
4      r_won = ALMA(r_start, loss);
5      rewardHistory[r_start].update(r_won);
6      reward.update();
7      if u(r_start) > u(r_won) then
8          loss.update(r_start, r_won, α);
9      if r_start ≠ r_won then
10         r_start = argmax_r reward[r];
11 end
```

In [DF20] the authors have proven that ALMA-Learning converges (i.e. eventually r_{start} does not change anymore). Furthermore, their experiments show that ALMA-Learning does improve social welfare over the original ALMA algorithm from [DFRF19].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 3

Problem Formulation

In order to motivate our formulation of the meeting scheduling problem we want to consider a specific real-world scenario. Imagine a business with a number of employees that have to regularly schedule meetings among smaller groups or departments. Once every week the company wants to be able to generate a schedule for all necessary meetings in the upcoming week. Each employee has preferences as to when he or she would like to schedule any specific meeting. Additionally, we also include times when an employee is simply not available. For example due to personal reasons or another meeting already being scheduled. We assume that all employees assigned to a meeting have to be able to attend in order for the meeting to be considered for scheduling at a given time. With that knowledge we now want to generate a valid schedule for the whole week that takes into account the preferences as well as possible.

3.1 Problem Description

We want to formalize the problem as a constraint satisfaction problem (CSP). CSPs were first introduced by Montanari in [Mon74]. The fairly simple formalism allows to model a wide variety of problems. Therefore it is quite common to formulate problems such as the meeting scheduling problem as a CSP. In [Kum92] the author gives a few examples of problems that can be formulated as CSPs as well as possible solution techniques. A commonly used definition of a CSP would be that it consists of a set of variables with respective domains and a number of constraints between the variables. Since we are looking at an optimization problem we also need an objective function to maximize or minimize. For the following formulation we will use functions with predefined co-domains instead of variables with respective domains. It is, however, easy to see that these things can be interchanged without any problems.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.1.1 Constants

In this section we want to start by describing the constants that define a problem instance and are therefore considered given. First, we start out with defining the following sets for our problem:

- A set of individuals $\mathcal{P} = \{P_1, \ldots, P_m\}$. These would usually represent people in the context of meetings scheduling e.g employees or friends in a group. They can attend meetings.
- A set of meetings $\mathcal{M} = \{M_1, \ldots M_n\}$, that we want to schedule. For each of those meetings we will also need to define a set $A \subseteq \mathcal{P}$ of attendees. In this formulation we will use a fixed set of attendees for each meeting. In order for a meeting to take place all of them have to attend. If that is not possible, we consider the meeting unschedulable. For comparison, other publications such as [NS20] use the concept of a quorum to allow meetings to be scheduled even if part of the invited individuals are not available.

To formally describe the participation of people in a given meeting, we additionally define a function mapping each event to the set of its participants

$$\text{part} : \mathcal{M} \to \mathfrak{P}(\mathcal{P})$$

where $\mathfrak{P}(\mathcal{P})$ denotes the power set of the set of people.

Now that we have defined the set of meetings, we need to formulate a way of scheduling them. For that we define a calendar as a finite number of timeslots. Using those, we can then set a meeting by assigning it to a timeslot. While timeslots like that could already be used to formulate our problem, we want to narrow down the definition a little in order to make it easier to work with. Therefore, instead of defining a number of timeslots directly, we define a number of *days* and *slots* to denote the number of days and time slots per day for a calendar that we want to use for scheduling. The slots per day are then assumed to be of equal size. E.g. $days = 7, slots = 24$ would define a calendar for one week where each slot is 1 hour long. This way we can make scheduling for meetings easier when it comes to real-world scenarios. The downside of this is that each slot now has the same length in that they all represent the same amount of time.

Meetings, however, are in general not all the same length. Therefore we add another property for each meeting in the form of a function mapping meetings to their length as the number of slots needed for the meeting:

$$\text{len} : \mathcal{M} \to \mathbb{N},$$

where $\mathbb{N}$ denotes the set of natural numbers (excluding 0). This gives us the ability to assign a variable length to each meeting. The precision for the length can be increased arbitrarily by increasing the number of slots. Note that we do not limit the length of a

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

meeting, which allows for events to exceed a single day and even the whole calendar, if needed.

Finally, we need a way to enable people to specify their preference for attending certain meetings at any given starting time. For this we define the following function:

$$\text{pref} : \mathcal{M} \times \text{part}(\mathcal{M}) \times \{1, \ldots, days\} \times \{1, \ldots, slots\} \to [0, 1].$$

This function allows each participant to set a preference as a value between 0 and 1 for each meeting at each available timeslot. For example $\text{pref}(M_1, P_1, 2, 6) = 0.7$ would mean that person P_1 has a preference of 0.7 to attend meeting M_1 starting at day 2 and slot 6. This, admittedly, quite specific function allows the participants to also differentiate between different kinds of meetings. For example one could be available in the evening for personal events while giving a higher preference to work-related meetings during the day. Also priorities between different meetings could be encoded that way. Note that we will later use constraints to define a preference of 0 as being unavailable.

3.1.2 Variables

All functions, sets and constants from the previous section are given to us for each instance. Now we need to define variables that actually describe the assignment of meetings to the timeslots. In other words, the goal of finding any schedule, given the properties from above, consists of finding a function

$$\text{sched} : \mathcal{M} \to (\{1, \ldots, days\} \times \{1, \ldots, slots\}) \cup \emptyset$$

that assigns each meeting to a given starting time. This is our only variable function. Note that we have only assigned meetings to single slots and have not used the length property yet. This will be handled in the constraints later. Also $\text{sched}(M) = \emptyset$ means that the meeting M is not scheduled. At this point we could already use a CSP solver for the problem. However, one would quickly notice that the results are not very interesting as, for example, we still have no way of avoiding overlap. In order to generate a valid schedule, the function sched has to meet certain constraints.

3.1.3 Constraints

For this formulation of the meeting scheduling problem we demand the following hard constraints be met:

1. No two scheduled meetings with common participants must overlap. As mentioned before we only consider the starting time for a meeting when scheduling it. For this constraint we now also have to take into account the length of each meeting.
2. No meeting shall be scheduled at a (day, slot) tuple if any of the participants is not available. In order to encode the notion of not being available into this problem we define that a preference of 0 (as given by the function pref) denotes the unavailability of the given person for that meeting at the given (day, slot) tuple.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

More formally, the hard constraints are:

$$\forall M_1 \in \mathcal{M}, \forall M_2 \in \mathcal{M} \setminus \{M_1\} : \tag{3.1}$$
$$(\text{sched}(M_1) \neq \emptyset \wedge \text{sched}(M_2) \neq \emptyset \wedge \text{part}(M_1) \cap \text{part}(M_2) \neq \emptyset)$$
$$\Rightarrow (\text{sched}(M_1) > \text{end}(M_2) \vee \text{sched}(M_2) > \text{end}(M_1)),$$
$$\forall M \in \mathcal{M} : (\exists P \in \mathcal{P}, \exists d \in [1, days], \exists s \in [1, slots] : \tag{3.2}$$
$$\text{pref}(M, P, d, s) = 0 \Rightarrow \text{sched}(M) \neq (d, s)),$$

where $\text{end}(M)$ returns the ending time (last slot) of the meeting M as calculated by the starting time $\text{sched}(M)$ and the length $\text{len}(M)$. Note, that since we allow meetings to extend over a single day the ending time could also be on another day. The comparator $>$ needs to be defined accordingly.

3.1.4 Objective Function

In addition to finding a valid schedule we want to make sure that the preferences of the participants are met using a suitable metric. For this work the focus lies on optimizing social welfare i.e. the sum of the preferences for all scheduled meetings. So in addition to satisfying the given hard constraints above, one would also have to maximize the following expression:

$$\sum_{\substack{M \in \mathcal{M} \\ \text{sched}(M) \neq \emptyset}} \sum_{P \in \text{part}(M)} \text{pref}(M, P, \text{sched}(M))$$

While we will use social welfare as a metric throughout this work, note that other approaches are also possible and useful here. For example, instead we could try to look at egalitarian welfare, where we would have to optimize the minimal sum of preferences for each person in the system. For that, the objective function would have be such that it would maximize the following expression:

$$\min_{p \in \mathcal{P}} \sum_{\substack{M \in \mathcal{M} \\ \text{sched}(M) \neq \emptyset \\ P \in \text{part}(M)}} \text{pref}(M, P, \text{sched}(M))$$

3.2 Complexity

In this section we will briefly discuss the complexity of the aforedescribed problem. It might seem obvious to some, that the meeting scheduling problem as formulated here is NP-hard, i.e. that there is no known algorithm to solve such a problem in polynomial time. However, we want to further support this by providing a polynomial time reduction from the 0-1 Knapsack Problem, which has been shown by Karp in [Kar72] to be NP-complete, to our meeting scheduling problem. For this, we start by providing the following definition of the 0-1 knapsack problem (based on the definition in [Kar72]):

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

In the 0-1 knapsack problem we are given a set of n items, each with respective weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ (for simplicity we assume non-zero, integer values and weights). Informally, we then pack a knapsack with those items trying to maximize the value of packed items while simultaneously trying to not exceed a given weight limit W. Each item can be packed at most once. Formally, we solve the following integer linear problem:

$$\max \sum_{i=1}^{n} v_i x_i$$
$$s.t. \sum_{i=1}^{n} w_i x_i \leq W$$
$$x \in \{0,1\}^n,$$

where the variable x is a binary array describing whether or not the i-th item is packed into the knapsack.

For the reduction we are now looking for a polynomial time algorithm to convert any instance of this problem into a new instance of the meeting scheduling problem described above. We are therefore given a 0-1 knapsack instance i.e. a number of items n, their values v and weights w as well as the maximum allowed weight of the knapsack W. If we consider the items as meetings and their weights as their length, we can see that scheduling meetings and the knapsack problem are very similar in nature with the only difference being that in meeting scheduling we have an additional order on the meetings i.e. a specific starting slot.

The idea is now to formulate an instance of the meeting scheduling problem such that our items correspond to meetings and that the number of slots correspond to the weights. This, however, leaves us with a small issue. In the formulation above we allowed meetings to extend beyond a single day, which makes it hard to impose the strict weight limitation from the knapsack problem as a time limitation in the meeting scheduling problem. In order to tackle this, we add an extra meeting that in any optimized case has to occupy the last slot, thereby preventing any meetings to exceed the original number of slots. Formally we can now generate an instance of the meeting scheduling problem with the following parameters:

- $\mathcal{P} = \{P_1\}$: We only have a single person for the meeting scheduling instances. This person will be assigned to all meetings.
- $\mathcal{M} = \{M_0, M_1, \ldots, M_n\}$: Each meeting corresponds to one item from the 0-1 knapsack problem, where M_0 is the additional meeting meant to occupy the last slot.
- $\forall i \in 0, \ldots, n : \text{part}(M_i) = P_1$: Each meeting has P_1 as its only participant. This ensures that no meetings can overlap.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- $\forall i \in 1, \ldots, n : \text{len}(M_i) = w_i, \text{len}(M_0) = 1$: Each meeting has a length corresponding to the weight of the respective item. M_0 has a length of 1 slot.
- $days = 1, slots = W + 1$: We have one day and the number of slots are equal to the maximum possible weight of the knapsack plus an additional slot to be occupied by M_0.
- $\forall i \in 1, \ldots, n \; \forall k \in 1, \ldots, W + 1 : \text{pref}(M_i, P_1, 1, k) = \frac{v_i}{\sum_{j=1}^{n} v_j + 1}$: Each Meeting that corresponds to an item has a preference correlating to the respective value, regardless of the slot. Note that this implies that the preference for any meeting that is not M_0 is lower than 1.
- $\forall k \in 1, \ldots, W : \text{pref}(M_0, P_1, 1, k) = 0, \text{pref}(M_0, P_1, 1, W + 1) = 1$: The additional meeting has preferences such that it will always be scheduled for the last slot. This causes the last slot to be effectively blocked, therefore avoiding any meetings to occupy more slots than available.

Given that we can now solve this instance of the meeting scheduling problem we can again transform this solution to a solution of the 0-1 knapsack problem. To do this we simply define the values of x as follows:

$$\forall i \in 1, \ldots, n : x_i = \begin{cases} 0 & \text{if } \text{sched}(M_i) = \emptyset \\ 1 & \text{else.} \end{cases}$$

We can argue that for this solution the weight constraint of the 0-1 knapsack problem is satisfied as the weight corresponds to the number of slots in the schedule. Since there cannot be any overlap between the meetings (see Constraint 3.1) and there are only W slots available the weight constraint has to be satisfied. It is also easy to see that the selected meetings (i.e. the ones with be best objective value) also maximize the objective function of the 0-1 knapsack problem, since any valid solution of the 0-1 knapsack problem is also a valid solution of our meeting scheduling formulation.

With this we have shown that our formulation of the meeting scheduling problem is NP-hard. In simple terms this means that it is unknown whether or not a polynomial time algorithm for the problem can exist. Therefore it is no surprise that any known exact algorithms for this and similar problems do not scale very well. As a consequence we are looking further into heuristics that can help us find good solutions for the meeting scheduling problem.

3.3 ALMA

As mentioned before, this thesis focuses on using the ALMA heuristic (see Section 2.1) for the meeting scheduling problem. ALMA was designed for the assignment problem and essentially works by having agents negotiate with altruistic behavior. Looking back at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the problem statement for meeting scheduling in Section 3.1 we can already see parallels between it and the assignment problem, where each agent had to be assigned a resource.

If we consider each meeting as an agent and each starting time, i.e. (day, slot) tuple, as a resource, the formulations look rather similar. The utility function for a meeting/agent and the corresponding slot/resource would then be given as the sum of all preferences of all attendees. There are, however, some important differences as well. For one, two meetings can occur at the same time as long as their sets of respective attendees are disjoint. Also two meetings can be conflicting without using the same exact starting slot if one of them is long enough. However, if we detect these conflicts, the concept of ALMA can still be applied. As we have seen in Section 2.2 this also implies that we can apply ALMA-Learning to our algorithm as well.

3.4 Further Thoughts and Considerations

One major benefit of our formulation of the meeting scheduling problem is that it allows users to set very specific preferences for each meeting and each slot. This makes it possible to encode various soft constraints into a users preferences. We already mentioned examples such as preferring private meetings over business meetings at certain hours. Other examples would be ordering meetings by importance and giving preferences accordingly or preferring shorter meetings over longer ones. When doing this, however, it is important to note that since we optimize the overall social welfare, the choices of preference of the other attendees heavily influence the results. Thereforem, it is advisable for all users in the system to use the same scale of preferences for meetings.

However, while powerful, the soft constraints are not enough to formulate the full variety of circumstances that can appear when trying to schedule meetings. For that we can also easily add more hard constraints to our formulation. For example consider a situation where a business is based in multiple cities. We would then have to additionally make sure that there is always enough time for a person to travel between the cities in order to attend meetings. This can easily be modeled by adding a function mapping meetings to cities. Then we only need to add a constraint that prohibits a person from attending two meetings in different cities in a given timeframe (depending on the chosen cities). We will later see that our algorithm as described in Chapter 4 also makes it very easy to add such constraints as long as they can be enforced by a single user or all attendees of a meeting.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Algorithm

Now that we have a problem formulation and a motivation to tackle it using a heuristic approach, we focus on formulating an algorithm to do just that in this chapter. As mentioned before, we will base our approach the ALMA heuristic. Furthermore, we will describe a multi-agent system to use for our algorithm. We have briefly introduced multi-agent systems in Chapter 2. We will also look into possible variations of the algorithm as well as give a short complexity analysis.
Note: From this section onward we may also refer to meetings as events.

4.1 Setup and Initialization

In order to implement an algorithm to tackle the meeting scheduling problem we are using a multi-agent system. This provides us with the means to model the problem and represent individuals. We are using three different agent classes for that:

- **RepresentationAgent**: For each individual there will be one RepresentationAgent to work on its behalf. The agent has all necessary local knowledge and will act on behalf of the individual by relaying information to other agents if needed.
- **EventAgent**: For each event there will be one agent representing it. These agents will negotiate with each other in an ALMA-based way to find possible slots for scheduling meetings. They will also relay necessary information to the RepresentationAgents.
- **SynchronizerAgent**: Since the ALMA algorithm works in a synchronized round-based way, we also create a special centralized agent to keep all other agents synchronized. Note that this is not necessary as there are several other state of the art synchronization methods, such as vector clocks, allowing for a truly decentralized system.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

For simplicity we use the variables introduced in Section 3.1 to describe these agents. I.e. $\mathcal{E}$ is the set of EventAgents and $\mathcal{P}$ the set of RepresentationAgents. Furthermore, from now on we will refer to a (day, slot) tuple simply as a slot. Since each agent in $\mathcal{P}$ represents an individual, they initially possess knowledge of the events that individual wants to attend, such as their length and the corresponding personal preferences. Additionally, each agent in $\mathcal{P}$ creates an empty personal calendar for events to be added later. The agents in $\mathcal{E}$ on the other hand, know their set of attendees.

In order to be able to use the ALMA heuristic for negotiation, the EventAgents in $\mathcal{E}$ need to have some way to quantify the preferences of their attendees. A simple way of doing this would be for all RepresentationAgents in $\mathcal{P}$ to simply relay their complete preferences to the corresponding EventAgents upon initialization. This is also the main strategy we pursue in this thesis. However, this implies that for each meeting a central entity (i.e. the corresponding EventAgent) has complete knowledge of the preferences for all attendees, which might be unwanted in some cases. Therefore, we also propose alternatives to increase privacy by adding noise to that data or even not relaying preference data at all and instead only sending a ranking of possible slots. The latter allows the corresponding EventAgent to choose a scale and does therefore not rely on all users resorting to the same metric when it comes to their preferences. These and other thoughts on privacy are laid out in more detail in Sections 4.3 and 6.2.4.

Once the agents in $\mathcal{E}$ receive the preference data from the RepresentationAgents (regardless of the form) they will then combine the data. E.g. for social welfare we would simply sum up the given preferences for each slot. Since we defined our preference such that slots with a preference of 0 are considered unavailable, slots where one of the received preferences is 0 will assume the value 0 instead of the sum. The slot entries will then be converted into a list $\mathcal{L} = [L_1, \ldots, L_{days \cdot slots}]$ sorted descending by preference, where each entry is a tuple of the form $L_i = \langle day_i, slot_i, preference_i \rangle$. Entries with a preference of 0 will be dropped from that list.

Finally, as mentioned before, we need a way to synchronize our agents. In order to implement this algorithm in a distributed fashion, we decided to use a dedicated SynchonizerAgent. For later experiments we need a central entity to collect all the calendar data anyway in order to be able to measure the performance of the algorithm. Therefore, we decided to go for a dedicated centralized agent instead of other state-of-the-art methods such as vector clocks.

The SynchronizerAgent collects that data and also makes sure that all other agents are synchronized. In order to keep all agents synchronized we say that a round ends by the SynchronizerAgent receiving an "end round" signal from all EventAgents and begins with the SynchronizerAgent sending a "start round" signal to all EventAgents. This way we make sure that no two agents operate in conflicting rounds. Before the first round it waits until a "ready" signal is received from all EventAgents indicating successful initialization. A pseudocode for this initialization, when relaying actual preference data, can be found in Algorithms 4.1, 4.2 and 4.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.1: Pseudocode for initialization of a RepresentationAgent.

```
   Input: Id i of the person to be represented by the current agent
 1 events ← loadEvents(i) ;  // load list of all events for person i
 2 lengths ← [];
 3 preferences ← [];
 4 calendar ← new Calendar(days, slots) ;          // create a new empty
    calendar
 5 foreach e ∈ events do
 6 |   length ← loadLength(e) ;                     // load length of event e
 7 |   preference ← loadPreference(e, i) ; // load preferences of event e
 |      for person i
 8 |   send(e, preference) ;              // send preference data to the
 |      corresponding EventAgent
 9 |   lengths.add(length) ;
10 |   preferences.add(preference);
11 end
```

Algorithm 4.2: Pseudocode for initialization of an EventAgent.

```
   Input: Id j of the event to be represented by the current agent
 1 attendees ← loadAttendees(j) ; // load list of attendees for event
    j
 2 receive(attendees, preferences) ;    // receives the preferences from
    all attendees
 3 preference ← sumPreferences(preferences) ; // combine preferences from
    all calendars into a single calendar by summing them up.
    (Entries with preference 0 will be assumed unavailable)
 4 utilitylist ← sortPreference(preference) ;    // convert the preferences
    into a sorted list and drop entries with preference 0
 5 if utilitylist.isEmpty() then
 6 |   notify all other agents to abort this event;
 7 else
 8 |   send(sync, "ready") // ; // let the SynchronizerAgent know that
 |      initialization was successfull
 9 end
```

Algorithm 4.3: Pseudocode for initialization of the SynchronizerAgent.

```
 1 events ← loadEvents() ;                       // load list of all events
 2 receive(events, "ready") ;       // receives "ready" signal from all
    EventAgents
 3 send(events, "start round") ;    // send "start round" signal to all
    EventAgents
```


Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.2 Strategy

In this section we will describe the strategy that EventAgents in our algorithm use to negotiate for a slot. In simple terms, an EventAgent selects a slot and then identifies possible collisions. If a collision is found it uses the loss function to determine its loss and based on that will decide what to do in the following round. In this section we also consider some variations of that strategy.

4.2.1 Identifying collisions

With this setup we can now apply the ALMA heuristic in a similar fashion as described in Section 2.1. Our EventAgents now have a list of resources i.e. slots they want to acquire and corresponding utility values for them i.e. the sum of preferences of the attendees. Based on those, they will start a round by trying to acquire the slot with the highest value in their list.

In [DFRF19] the authors use cases where collisions, i.e. multiple agents trying to acquire the same resource at the same time, are detected through the given environment. In our case collisions and the respective detection is based on the attendees of each event. Therefore, we decided to let the RepresentationAgents detect possible collisions. In order to be able to do that, the EventAgents send a message to all their attendees containing information on which slot they would like to acquire. Once the RepresentationAgents have received that information from all working EventAgents with events they attend, they check for possible collisions. These are identified by taking into consideration all simultaneous attempts to acquire slots, as well as an internal calendar for each RepresentationAgent that contains already scheduled slots. Afterwards, the RepresentationAgents relay that information back to the corresponding EventAgents. Algorithm 4.5 gives a pseudocode for collision detection and Algorithm 4.4 describes the foregoing "start round" action of the EventAgents.

Algorithm 4.4: Pseudocode for an EventAgent to start a round.

Input: Id j of the event to be represented by the current agent

```
1 Event receive "start round" signal from the SychronizerAgent:
2     preferredChoice ← utilitylist[0] ; // save the first element in the
        list of utilities as the prefrered choice
3     send(attendees, preferredChoice) ;  // send the preferred slot to
        all attendees
```

Leaving the decision of collision detection to the attending RepresentationAgents does not only make sure that the relevant information for scheduling a meeting stays within the group of participants, but also allows to enforce additional personal constraints, as mentioned in Section 3.4. For example, if two events were impossible to attend after another, due to locational or other constraints, an attendee could simply enforce those by not allowing both events to be scheduled in succession.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.5: Pseudocode for collision detection of a RepresentationAgent.

```
   Input: Id i of the person to be represented by the current agent
1  Event receive preferredChoices from all agents in events:
2      foreach proposal in preferredChoices do
3          collision ← checkCollisions(preferredChoices, proposal, calendar) ;
            // check for possible collisions of the given
            proposal with other proposals and the already
            scheduled events in the calendar
4          send(e, collision) ;    // send the information on collisions
            back to the corresponding EventAgent
5      end
```

Furthermore this can also be used for people to improve privacy. Specifically when a person does not want to reveal for which slots they are not available directly, they could instead give false non-zero preferences to the EventAgents at initialization. If the EventAgent proposed such a slot that person, or rather the corresponding RepresentationAgent, could then block that slot.

After having proposed a slot to the RepresentationAgents, the EventAgent will wait until it receives feedback from all corresponding attendees. In case there is no collision for any of the participants, the EventAgent will acquire the slot by first informing its respective RepresentationAgents and then the SynchronizerAgent. All informed agents will delete the EventAgent from their list of working agents and the EventAgent itself will terminate.

4.2.2 Altruistic collision handling

If there is a collision, we work in the same way as ALMA, described in Section 2.1. That is, we calculate the loss as the mean difference in preference between the current option and the k next best alternatives.

$$\text{loss}_n^i = \frac{1}{k} \sum_{j=i+1}^{i+k} u_n(r_i) - u_n(r_j)$$

The back-off probability is then determined as $f(loss)$, where f can be any non-decreasing function with $f[\mathbb{R}] \subseteq [0, 1]$. At this point it is important to note that the preference for an EventAgent is not in [0,1] as in Section 2.1, but is instead bound by the number of attendees as a result of summing up the individual preferences. Therefore, the loss is also not in [0,1] anymore which, depending on f, could cause some issues. As a consequence, we also considered the possibility of normalizing the preferences in later experiments in Section 6.2. One such option is to simply divide the preference sum by the number of attendees. This has the downside, that it distorts the comparability of loss between the agents significantly since events with fewer attendees would be considered as important

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

as ones with more participants, even though the latter usually contributes more towards social welfare. Another option is to find a global variable to use for normalization. For example one could use the maximum number of attendees over all events or the highest preference value for any event. This would require each EventAgent in our system to share at least one property globally, which in our case can be easily accomplished by using the centralized SynchronizerAgent.

We want our probability function f to be designed in such a way that it inversely correlates with the loss function. The optimal form of the function, the slope and other properties are to be determined by experiments later. Since the properties we need for f are similar to the ones of a cumulative distribution function (cdf) we used some common cdfs as a motivation for possible candidates for f. This resulted in the following candidates:

$$f(loss) = \frac{1}{1 + e^{-\gamma(0.5 - loss)}} \tag{4.1}$$

$$f(loss) = e^{-loss \cdot \lambda} \cdot \mathbb{1}_{(0,\infty)}(loss) \tag{4.2}$$

$$f(loss) = 1 - \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{loss} e^{-\frac{(t-\mu)^2}{2\sigma^2}} dt \tag{4.3}$$

$$f(loss) = \begin{cases} 1 - \varepsilon, & \text{if } loss \leq \varepsilon, \\ \varepsilon, & \text{if } 1 - loss \leq \varepsilon, \\ 1 - loss & \text{otherwise.} \end{cases} \tag{4.4}$$

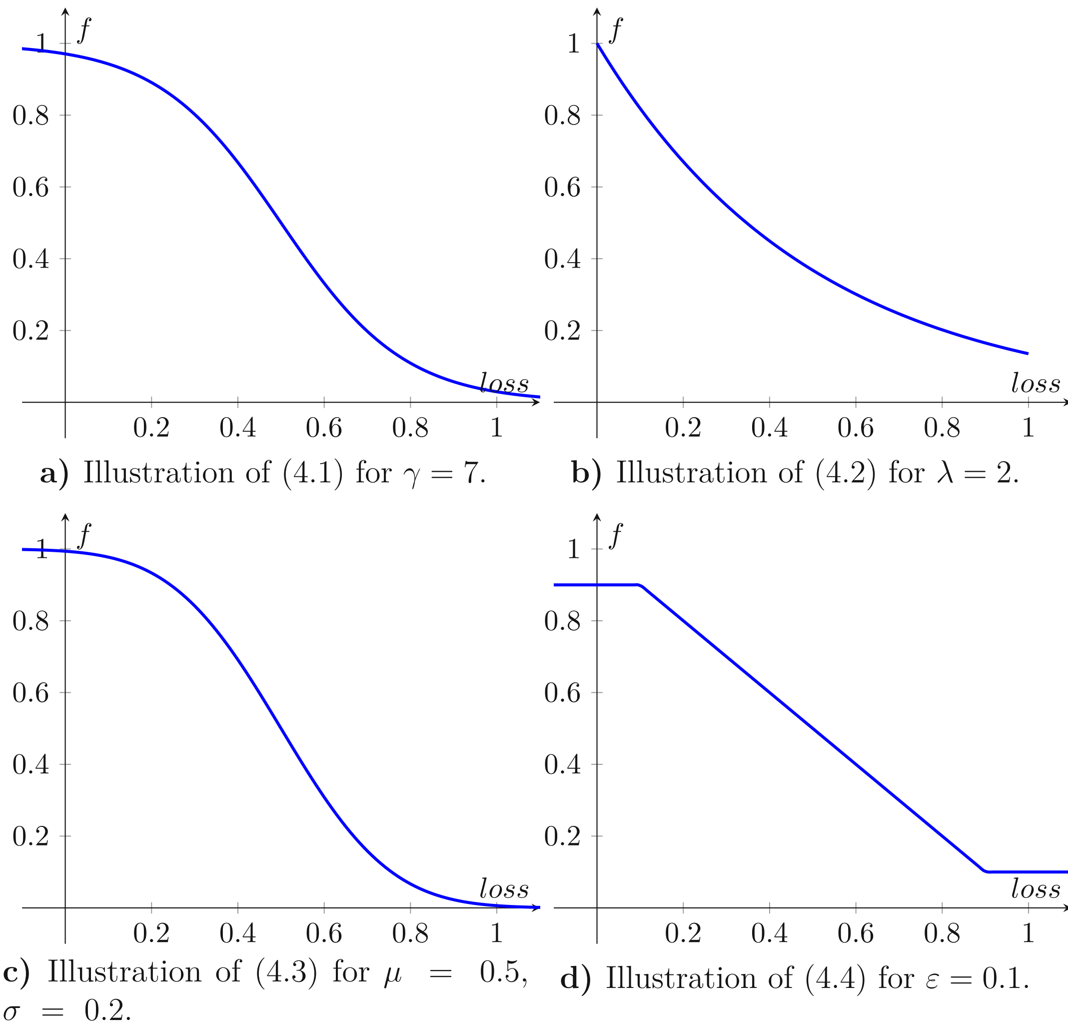
Where (4.1) and (4.4) are inspired by their previous use in [DFRF19] and are based on a logistic function and the cdf of a uniform distribution respectively. Furthermore (4.2) is based on the cdf of an exponential distribution and (4.3) on the cdf of a normal distribution. $\mathbb{1}$ is the characteristic function. Illustrations of these functions are displayed in Figure 4.1.

Using one of the proposed functions we then back-off with the calculated probability. If the agent does not back-off, it informs its attendees of that decision, ends the round by sending the corresponding signal to the SynchronizerAgent and tries to acquire the same slot again in the next round.

On the other hand, if an EventAgent decides to back-off, it will delete the currently selected slot from its list of possible slots, then inform the attendees, or rather the respective RepresentationAgents, of its decision and finally end the round by sending the corresponding signal to the SynchronizerAgent. It will try to acquire the next slot in the following round. Once all agents have ended their round, the SynchronizerAgent will start the next round. Algorithms 4.6 and 4.7 show the decision making process of the EventAgents as well as the following handling of that decision by the RepresentationAgents. Additionally Figure 4.2 displays the messages passed between the agents for this process as well as a few simplified intermediary steps.

The algorithm terminates once all EventAgents have terminated. This happens either if an EventAgent successfully schedules a meeting or if the list of possible slots becomes empty i.e. when it has unsuccessfully tried to acquire all initially considered slots.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek, WIEN Your knowledge hub



a) Illustration of (4.1) for $\gamma = 7$.

b) Illustration of (4.2) for $\lambda = 2$.

c) Illustration of (4.3) for $\mu = 0.5$, $\sigma = 0.2$.

d) Illustration of (4.4) for $\varepsilon = 0.1$.

Figure 4.1: Illustrations of the candidates for the back-off probability function.

4.2.3 Alternations to the Strategy

Some small alternations to the strategy described in this section were also considered for the experiments later. One such alternation was to modify the strategy such that the agent would not directly try to acquire the next slot in the following round but instead only monitor it. If the monitored slot was free, the agent would try to acquire it in the round after that. Otherwise it would move on to monitor the next slot. This small change allows agents to revisit slots. So it would be possible that instead of removing slots after backing off, we could add them to the end of the list so that they would eventually be reconsidered and the agents loop over the available slots until either an available slot is found or all slots are taken.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

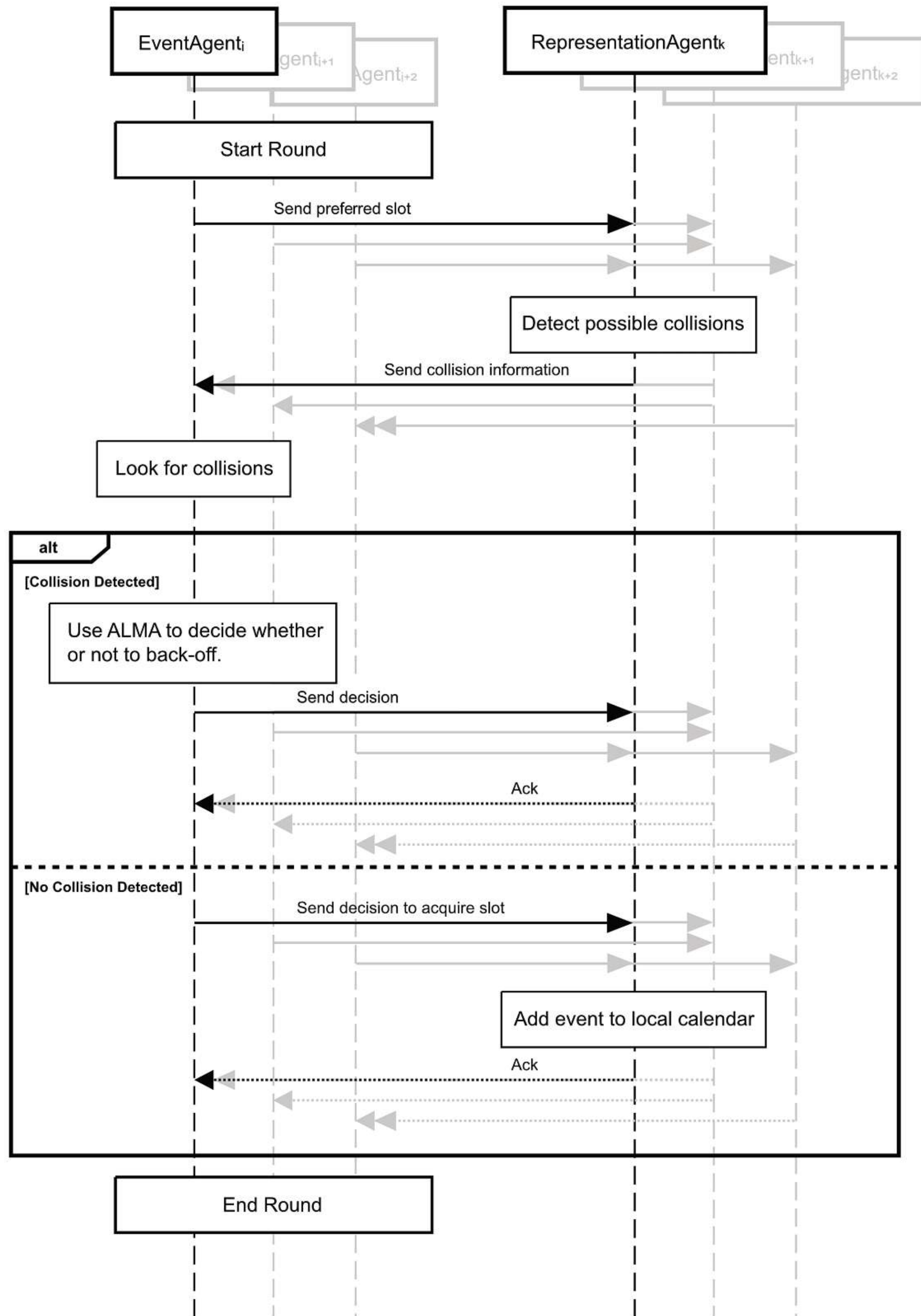


Figure 4.2: Sequence diagram for the negotiation protocol. Displays the communication between EventAgents and multiple RepresentationAgents as well as some simplified intermediary steps.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.6: Pseudocode for the decision process of an EventAgent.

```
   Input: Id j of the event to be represented by the current agent
1  Event Receive collisionInformation from all attending RepresentationAgents:
2      if collision ∈ collisionInformation then
3          status ← "continue" ; loss ← calculateLoss(utilitylist[0..k]) ;
            // calculate loss based on the k next slots in
            utilitylist
4          backOffProb ← f(loss) ;   // calculate back-off probability
            based on loss
5          x ← uniformRandom(0,1) ;        // generate uniform random
            distributed variable
6          if x < backOffProb then
7              utilitylist.delete(0) ;          // remove current slot from
                consideration if we decide to back-off
8              if utilitylist.isEmpty() then
9                  status ← "abort" ;
10                 send(attendees, "abort") ;     // inform attendees if no
                    more slots are available
11             else
12                 send(attendees, "backoff") ;    // inform attendees about
                    the decision to back off
13         else
14             send(attendees, "continue") ; // inform attendees about the
                decision to not back off
15     else
16         status ← "acquired" ;
17         send(attendees, "acquire" + preferredChoice) ; // inform attendees
            about the decision to acquire slot
```

Note: In the original strategy revisiting slots is not possible as it could end in a deadlock. That happens if two or more agents try to acquire the same slot and have only that one slot left to consider. In that case those agents would collide every round and never be able to either acquire the slot or remove it from consideration.

Another modification that was considered to the original protocol is to change the way an agent backs off. Instead of directly deleting the current slot and going to the next one, the agent would spend another round simply monitoring the current slot and not compete for it. If, in that round no other agent tried to acquire the slot, then the agent would go back to competing for it in the following round. This would prevent situations where multiple agents back-off from a slot at the same time and then no one acquires that (potentially high utility) resource. On the other hand this reduces the solution space and therefore could yield worse results as well.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.7: Pseudocode for the decision handling of a RepresentationAgent.

```
   Input: Id i of the person to be represented by the current agent
1  Event Receive decision message from all agents in events:
2      foreach e ∈ events do
3          msg ← message received from e ;
4          if msg contains "acquire" then
5              calendar.addEvent(msg.getPreferredchoice()) ;        // Adds the
                preferred choice contained in msg to the internal
                calendar
6              events.remove(e);       // Remove current EventAgent from
                consideration
7          else if msg == "abort" then
8              events.remove(e) ;      // Remove current EventAgent from
                consideration
9          send(e, "ok");
10     end
```

4.2.4 ALMA-Learning

Now that we have a finished algorithm using ALMA we can easily integrate ALMA-Learning as well. Looking back at the description in Section 2.2 we can see that we only need to add a few extra variables for ALMA-Learning to work. We have to replace the loss function with the respective array that learns its values over time via value iteration. Additionally, we have to keep the reward history for each starting slot in order to be able to determine the best slot to use in future iterations. With that setup we can then execute ALMA-Learning by repeatedly running our algorithm and updating the respective variables after each run. That process is then repeated until a certain exit condition is met. In this thesis we have used a fixed number of steps for that in order to keep things simple and consistent for the evaluation later. Other possible exit criteria would look at the change in values for e.g. the starting slot and terminate once no agent changes starting slots anymore.

4.3 Privacy

In previous sections we have already mentioned the importance of privacy and have also hinted at possible ways of achieving it to some degree. In this section we will further look into a variety of different privacy enhancing methods as well as possible downsides to them. Since our algorithm has two agent classes that might want to keep their information private, we will look at both of them separately.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.3.1 RepresentationAgent Privacy

The RepresentationAgents know all the preferences of the respective user. Therefore, in order to provide privacy they would have to keep the preferences to themselves. However, as part of the algorithm they have to share that information with the given EventAgents to some degree. If those are not considered secure, we will have to find a way to let the EventAgents select suitable slots without directly giving them the preference values. One simple way would be to obscure the existing preferences. This could be done by adding gaussian noise to the data and then transmitting the noisy preferences. Alternatively, the RepresentationAgent could relay a ranking of slots, thereby obscuring the detailed preferences. Since our algorithm also allows us to block slots during runtime we are even able to obscure the occupied slots that way at initialization.

There are several other methods one could choose to improve privacy. For example the agents could only relay their n most preferred slots to the EventAgent. This keeps most of their other data private while still providing a good overview of the preferences for the EventAgent. The RepresentationAgents could alternatively also find preferred days or time-ranges through negotiation and then relay their data only for those slots. These and various other methods can be used to make sure that the EventAgents only gain little knowledge about the preferences of the RepresentationAgents while still providing enough data to make it possible for them to schedule a meeting. However, we also want to make sure that the EventAgents each keep their information.

4.3.2 EventAgent Privacy

In order to make sure that the privacy of the attendees is guaranteed the EventAgents have to keep their personal utility function i.e. the preference values for the slots private as well. Our algorithm already provides a high degree of privacy as there is no direct communication of any kind between EventAgents and agents that are not assigned to the corresponding meeting. Therefore information regarding the attendees preferences should stay mostly within the group. However, the order in which slots are visited is deterministic and therefore privacy cannot be guaranteed. This aspect is exploited in the use of ALMA-Learning for example.

For that reason Panayiotis et al. designed a privacy-preserving version of ALMA (PALMA) in [DTF21]. In simple terms, instead of always acting true to their own preference, agents also perform random resource selection or randomly back-off with a certain probability. The authors have shown that their approach yields good results in terms of social welfare for the considered scenarios, while also giving strong privacy guarantees.

One thing that all of the methods mentioned above have in common is that they all usually worsen the social welfare to some degree. That is no surprise, since they all reduce the knowledge we have for finding a solution to a given problem or add some randomness to our actions. Therefore, it is always important to keep in mind that there is a trade-off between privacy and social welfare. This is why it is crucial to investigate that trade-off for any given method before using it in practical scenarios. In our case we

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

have performed some experiments with noisy data. The results can be found in Section 6.2.4.

4.4 Convergence and Runtime Considerations

For the algorithm in the pseudocode it is easy to find a bound on the expected number of rounds to finish. For that we define p as the lowest back-off probability for this instance. In the worst case scenario an agent would consider to acquire each slot but never succeed. As given by the negative binomial distribution, the expected number of steps until that agent backs-off from a slot is given as $1/p$. In the worst-case scenario we have to repeat that process for each slot. Therefore the number of rounds for an agent to finish can be bound by $\mathcal{O}(\frac{R}{p})$, where R is the number of slots.

This bound does, however, not hold if we modify our algorithm. Especially if we allow events to be reconsidered instead of deleting them after backing off. In [DFRF19] the authors have considered that strategy for their problem and were able to bound the expected number of rounds for the algorithm to converge with

$$\mathcal{O}\left(R\frac{2-p^*}{2(1-p^*)}\left(\frac{\log N}{p^*}+R\right)\right), \tag{4.5}$$

where N is the number of agents, R the number of slots and $p^* = \min(p_{min}, 1-p_{max})$, with p_{min} and p_{max} being the smallest and biggest back-off probabilities respectively.

Furthermore the authors have shown that the expected number of rounds until an agent converges can be even bound by the convergence time of the sub-system said agent belongs to. Therefore we can replace N in (4.5) with N^* the maximum number of agents competing over any single resource at the same time. We can also replace R in the same equation with R^* the maximum number of resources to be considered by any agent. The new equation would then be

$$\mathcal{O}\left(R^*\frac{2-p^*}{2(1-p^*)}\left(\frac{\log N^*}{p^*}+R^*\right)\right). \tag{4.6}$$

While this might seem like a minor improvement over the runtime mentioned before. The authors have later gone on to find several test cases where the values R^* and N^* are naturally bound leading to a constant runtime for the algorithm.

Going back to our adaption of the ALMA heuristic for the meeting scheduling problem, we want to investigate what quantities N^* and R^* would correspond to.

- R^*: Since resources roughly correspond to timeslots in the meeting scheduling problem, R^* corresponds to the maximum number of possible slots for any event. That number is technically only bound by the number of all available timeslots. However, it is often impractical to consider all timeslots. Given a large calendar, it is more feasible for attendees to only give preferences for a bound number of slots.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The EventAgent could also disregard events with a low utility. Those steps would lead to a runtime improvement.

- N^*: In the meeting scheduling case two agents have a conflict if they want to set a meeting where the timslots as well as the sets of attendees overlap. Therefore, N^* is the maximum number of events competing for a timeslot with overlapping attendee sets. If we could now assume that we can cluster our total set of people into smaller groups such that either inter-group events are rare or preferred meeting times of groups do not overlap, then we could describe N^* as the maximal number of events of any such group. This form of clustering naturally occurs in many situations. People in the corporate world often build clusters by their work hours or their respective departments.

To summarize, the number of rounds is bound by the number of possible timeslots per event and the number of events per cluster, if a clustering exists. Furthermore we should also note that given the necessary communication between RepresentationAgents and EventAgents one round takes at least $\mathcal{O}(E^* + A^*)$ steps, where E^* is the maximum number of events for any single attendee and A^* the maximum number of attendees for any single event.

In all expressions above, describing the expected number of rounds, we can see that it grows with small p or p^*. When experimenting with several hyperparameters, specifically for the different candidates for f, we noticed that, with the exception of (4.4), the curves can become very steep and therefore p^* can indeed reach very small values. That leads to high runtimes where there is no progress for thousands of rounds. While it would be simple to not consider such steep curves, we also noticed that for some instances, where the algorithm actually terminated, good results were achieved. Therefore the decision was made to instead slowly reduce the steepness of the curve with increasing rounds. Since we only wanted to significantly change the steepness of the curve for high round numbers, we chose a modifier that grows slowly in the beginning. Specifically that modifier value was chosen to be $\exp(rounds/10000)$. That value was then used to modify the parameters responsible for steepness λ, γ and σ accordingly.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 5

Instance Generation

The first step in evaluating our model is to find some data to properly test it on. Unfortunately, we were not able to find real-world data for the meeting scheduling problem that contained information (e.g preference data) to the extend that would have been needed for this algorithm. Therefore, the decision was made to generate our own data. For this task we wanted to design a generator that allows us to variably set the number of events, the number of people in the system, the number of days and the number of slots per day. Based on those parameters the generator is supposed to create a random instance of the meeting scheduling problem for our algorithm to solve.

The generator was designed with situations for corporate meeting scheduling in mind. All people in the system are considered employees of a company that wants to schedule several meetings in a given timeframe. In order to properly portray the average company meeting schedule, we inspired some choices for possible parameters by the data published in [RN01]. The paper contains data on various studies with information on meetings in corporate America in the 80s such as meeting lengths, number of attendees, etc..

As mentioned before, we generate our instances for the meeting scheduling problem with a given number of meetings and people as well as a calendar i.e. given number of days and slots per day. Based on that we can generate a problem instance by generating and assigning properties for each event one-by-one. The properties we have to assign are length, a set of attendees and preferences for each attendee (see Section 3.1.1).

5.1 Assigning Length

First, we assign a length to the event. That number is based on the data published in [RN01]. More specifically, the paper looks at several intervals of meeting lengths (e.g. 0-30min, 30-60min, etc.) and gives us the percentage of observed meetings with a length in the respective interval. We then use those datapoints to fit a logistic curve. That

logistic curve is in turn used as a base for a random generator to generate meeting lengths with a distribution corresponding to the data. The function is additionally designed such that the maximum number of hours was 11. According to the data, less than 3% of meetings exceeded that limit. Note that the data from [RN01] is in hours or minutes and our algorithm works only with slots. Therefore we have to discretize the distribution based on the number of slots selected. After the length is assigned, the event gets further processed by assigning attendees.

5.2 Assigning Attendees

In order to chose attendees for an event, we first decide on the number of people to take part in the given event. That number again gets randomly generated based on data published in [RN01]. In this case the paper has data on intervals of meeting sizes (e.g. 1-5,6-10,etc.). Again, that data is fitted using a logistic curve which then is used for a random generator to return random meeting lengths following the distribution from to the data. No more than 90 people are chosen for an event at a time as there is only a small percentage of meetings exceeding that number in the data.[1] Additionally, for instances where the number of people in the whole system is below 90, that bound is reduced accordingly.

Now, that we have the size of the meeting, we want to think about how to properly assign attendees. A simple way would be to just pick the required number of attendees from the pool of people in the system completely at random. As we have already briefly mentioned in Section 4.4 this is usually not a natural way to assign attendees. When thinking about the problem in terms of a corporate scenario, for example, one would be quick to see that this is not how meetings are usually set, as a meeting across a number of different faculties or departments in a company is arguably rather rare. In this and many other scenarios there are usually specific groups that tend to have meetings with each other. In the corporate scenario these groups are usually defined by departments. In a social environment these groups could simply be groups of friends. However, that grouping is usually also not absolute. Sometimes groups invite new people or departments have a common meeting. We therefore need to find a way to assign attendees to each meeting that somehow simulates these conditions.

5.2.1 Generating Clusters

Our idea for this problem was to simulate a sort of clustering of people by assigning a point on a 1×1 plane to each person in our system in such a way that clusters start to form. These clusters would then represent a department or a group of friends. Then, the assignment of people to an event would work in such a fashion that people which are closer are much more likely to attend the same meeting. This would more or less mimic

[1] Note: The median is significantly lower.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the behaviour we wanted to create, as groups are more likely to be in the same meeting while rarely also selecting outsiders to attend.

In more detail, we create the points in an iterative way. The first person is assigned to a uniformly random point on the plane. For each following person there is a 30% probability that they also get assigned a uniformly random point. With a 70% probability the person would be assigned a point based on a normal distribution centered around one of the previously created points. Which of these previous points would be chosen as the center is decided based on the time of creation. I.e. a recently created point is exponentially more likely to be chosen as the center than an older one. In our experiments this method with the mentioned parameters has caused the desired building of clusters. The randomness and preference of more recent points when adding a new point avoids that a single cluster grows too big. Figure 5.1 displays the resulting points after a few steps of such an iteration.

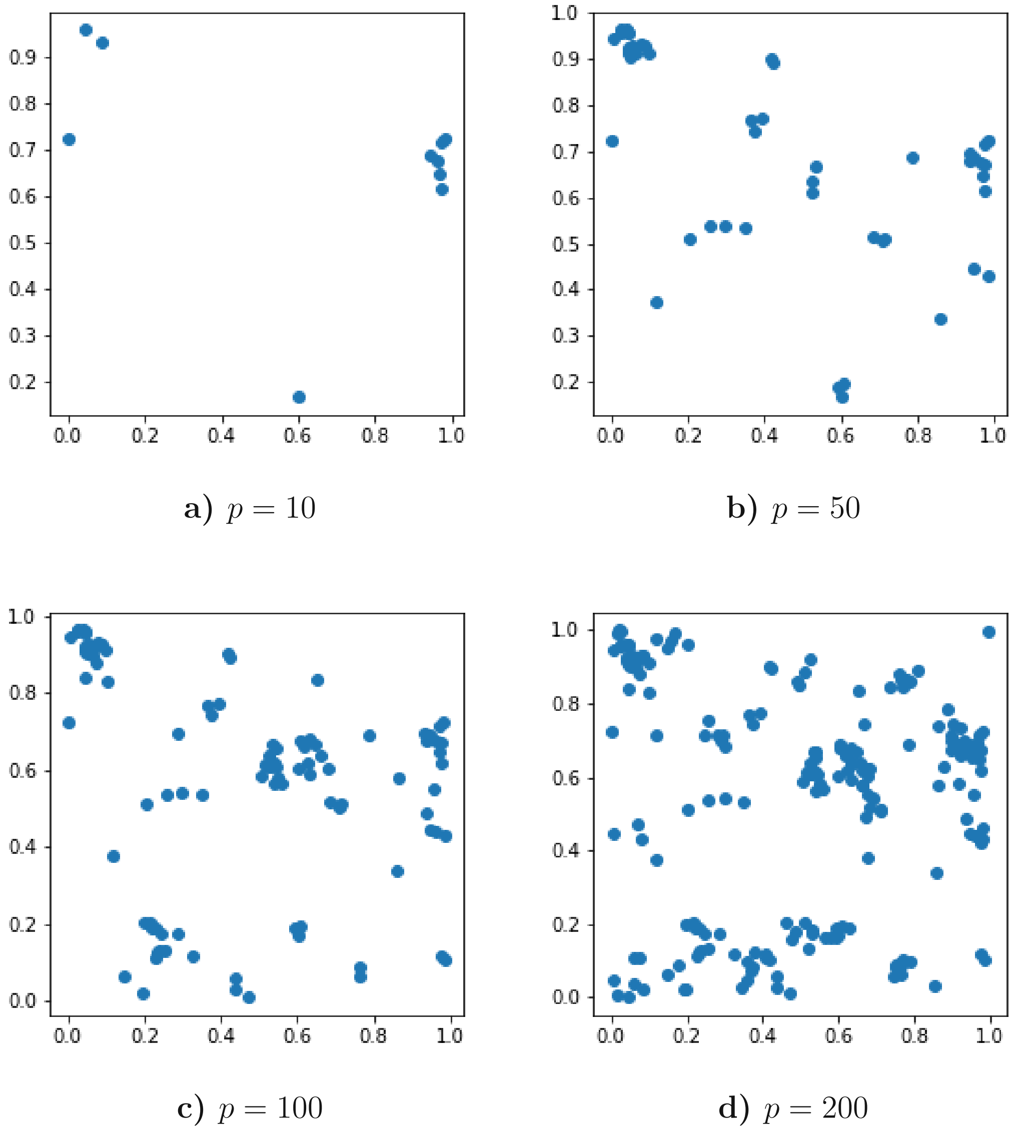
After each person is assigned a point on the plane we then choose a host for our meeting at random. Further attendees are then selected for the event based on the distance to the host such that people with closer points are significantly more likely to be chosen for the event.

5.3 Assigning Preferences

Finally, we have to assign preferences for the event to all of the attendees. Again we want the instance data to roughly reflect the situation of a corporate meeting. Unfortunately, thr previously cited paper [RN01] does not contain data to suggest when meetings usually take place. However, assuming that the people in our system work an office job we can consider a generic 9-to-5 workday schedule. We generate a preference table by combining values obtained from two independent functions:

- **Slot Preference**: This function is designed to roughly reflect the time available for a meeting on an average workday. For a given time of day (or respective slot) it returns a preference value. It only depends on the time slot and is independent of the day. Given that we assume a rough 9-to-5 workday, we usually have high preference values during that time, while returning a low value in the evenings. For example the preference for someone to take a meeting at 10am might be higher than for 2am. The exception to this rule would be a small lunch break at noon where our function returns a lower preference value.
- **Day Preference**: This function focuses on the preference for a given day while disregarding the slots. For a given day it returns a preference value. For this function the preference is meant to decrease over time. This should reflect that in many cases it is preferable to schedule meetings sooner rather than days or weeks into the future. Another possible function here would be a step-function that would indicate a significant decrease in preference after a given day. That could be used to represent a deadline.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



a) $p = 10$ **b)** $p = 50$

c) $p = 100$ **d)** $p = 200$

Figure 5.1: Generated datapoints on the 1×1 plane for p number of people.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

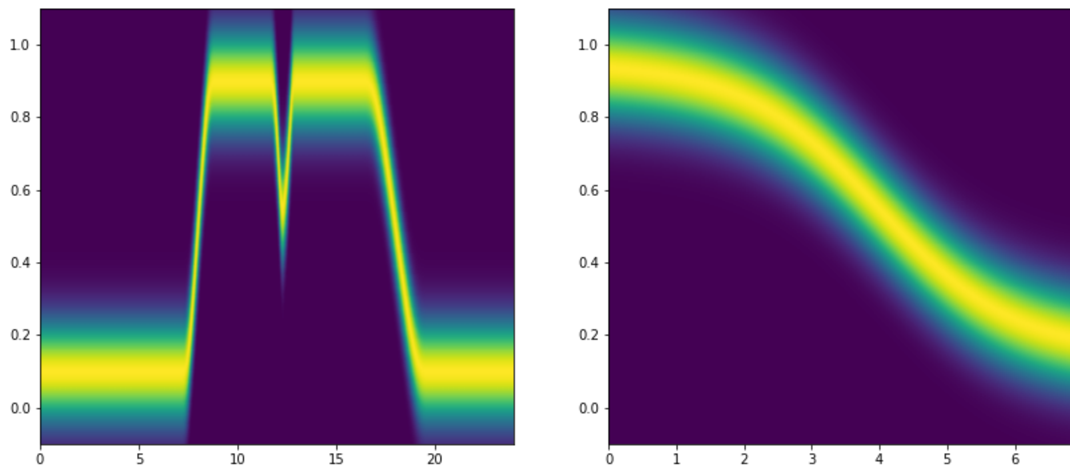


Figure 5.2: Heatmaps displaying the distributions used for generating preference data. The image on the left displays the distribution of preference throughout a workday. The image on the right displays the distribution of preference throughout a week.

For each user and event we then generate a preference by using these preference functions. Specifically the preference values are generated by calling both functions for each combination of slots and days and then multiplying the respective results. This way, however, every person would always have the same preferences for each slot. In order to add some randomness we therefore use a normal distribution with the aforementioned preference as mean and 0.1 as standard deviation and assign the result as the preference. Figure 5.2 displays what each of the preference functions looks like when the normal distribution is applied this way. When looking at the images we notice that due to the use of the normal distribution we might end up with preference values above 1 or below 0. For those cases we reject the values and repeat the previous step until the result lies in [0,1].

Finally, we need to simulate already scheduled meetings or other obstacles that would cause people to reject slots for a meeting. To do this, we simply blocked a random number (from a given range) of timeslots for each person given the event. This was simply achieved by setting certain slot preferences to 0. The actual slots to block this way were determined by the preference function for a day described above. I.e. a slot with a high preference is also more likely to get blocked.

Ultimately, additional slots were blocked in such a way that for each event only the 24 most valuable slots would remain available. This effectively limits R^* the maximum number of slots for any event and therefore, according to our thoughts in Section 4.4, should also improve our runtime.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 6

Evaluation

In this chapter we will look deeper into the actual application and evaluation of the algorithm described in Chapter 4. For that we briefly introduce the tools used to implement and evaluate the algorithm using the instance generator described in Chapter 5. We will also give some motivation for our choice of the used hyperparameters and tools. Furthermore, we implemented a few simple algorithms to use as a baseline to test our algorithm against. Beyond social welfare we will also look into other metrics such as fairness, the number of scheduled meetings and the number of rounds needed for the algorithm to terminate. Lastly, we are also going to examine the effects that some privacy-preserving methods can have on our results.

6.1 Reference Algorithms

In order to be able to evaluate the performance of our algorithm we implemented a few other algorithms for comparison.

6.1.1 Centralized Greedy Algorithm

The first algorithm implemented is a centralized algorithm that schedules meetings in a greedy fashion. The initialization works the same way as it does for our algorithm (described in Section 4.1). Afterwards, however, the SynchronizerAgent collects the preferences from all the EventAgents and then schedules the events in a greedy manner. I.e. we select the event with the highest possible preference and then try to assign it to the corresponding slot. If that is possible, we schedule and then remove that event from consideration. If scheduling is not possible, we instead remove that slot from consideration for the selected event. We repeat this process until no more meetings can be scheduled.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.1.2 ILOG CP Solver

The second algorithm used for comparison is also centralized. Here we used the ILOG CPLEX Optimization Studio [CPS] and the corresponding IBM ILOG CP Optimizer [CPO] [1] to formulate and solve the meeting scheduling problem as a CSP. In order to save time the previously yielded result from the greedy approach was used to warm-start the solver. Since that approach still takes very long for larger instances, we also limited the available amount of time to t minutes. For smaller problem instances this usually yielded the optimal solution, while for larger instances it often did not improve the original greedy result a lot. One benefit of this solver is that it also yields an upper bound for the optimal solution.

6.1.3 Distributed Greedy Algorithm

The next algorithm used for comparison is a distributed greedy algorithm. Once again the initialization works as described in Section 4.1. However, in order to preserve the decentralized nature of the problem we do not collect the preferences in one place. Instead a random EventAgent is chosen and gets to schedule its event for the slot with the highest possible preference that is still available. This step is repeated until all EventAgents had their turn.

6.1.4 MSRAC-based Algorithm

The last algorithm used for comparison is based on the MSRAC algorithm presented in [BH07]. The authors used that algorithm to solve a meeting scheduling problem similar to the formulation in Section 3.1. The key differences are:

- The version of the problem they considered does not assign a length to each event and therefore each event has a given length of 1 slot.
- Each person only has one preference per timeslot. Unlike our formulation, where a preference for each timeslot and event is allowed.
- Each event is assigned an importance value in order to give priority to more important events.

These differences are mostly limitations from our problem formulation. We have already discussed in Section 3.4 how such importance values could be encoded into the users' preference data. Despite the limitations of the problem formulation, the MSRAC algorithm also has a few advantages over our ALMA-based approach. For one, it works in a dynamic way, i.e. it allows the addition of new events at any time and can change the calendar accordingly whereas our algorithm has to schedule all events at once. Another advantage is that it works asynchronous and therefore eliminates the need for centralized synchronization.

[1] The IBM ILOG CP Optimizer is a state-of-the-art tool to solve constraint satisfaction problems.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Our implementation for a MSRAC-based algorithm to solve our formulation of the meeting scheduling problem works as follows. The initialization of the MSRAC algorithm works similar to our previous approach. If an agent wants to schedule an event he asks all participants for the relevant preference information. Once he receives said information the preferences are summed up and combined into a list of slots sorted by preference. Unavailable slots, as well as slots occupied by more important events, are dropped. The paper further proposes to relay only a ranking of slots instead of the complete preference data in order to increase privacy. The agent that collects that information would then have to generate a new implicit scale based on the rankings and consider that one as the preference. If in the sorted list entries have the same preference value the authors further recommend to choose the ordering such that the maximal distance between users' preferences is minimized.

Now the EventAgent proposes the first slot in its sorted list to its participants. Each participant then looks at the proposed events for that slot and keeps the one with the highest importance value. More specifically:

- If the slot is free, accept the proposal.
- If the slot is occupied by a less important event, accept the proposal and invite the agent of the less important event to reschedule.
- If the slot is occupied by a more important event, reject proposal and invite the agent to propose new slot.
- If the slot is occupied by an equally important event the authors propose a few ways of handling this case. The easiest one would be to simply choose to keep the event with a higher utility and reject and reschedule the other one.

If any EventAgent now receives a message to reschedule, it deletes the current proposal and proposes the next slot to its participants. This process is repeated until a stable state is reached.

With this we can now see that the MSRAC algorithm can also handle inputs where events are longer than 1 slot and people have more specific preferences. However, this will invalidate some theoretical considerations made by the authors and is therefore expected to worsen results. In order to yield an importance value for events, we simply decided to take the average preference over all participants. However, since meetings with more participants tend to be more important and they also contribute more to social welfare, we decided to additionally multiply this value by the number of participants. This choice for the importance value also makes it highly unlikely that two events have the same importance value, which is why we did not further analyze the effects of the different ways to handle collisions of equally important events that were originally proposed by the authors.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.2 Testing

In this section we will be using the instance generator from Chapter 5 to evaluate our ALMA-based algorithm and compare it with the algorithms introduced in the previous section.

6.2.1 General Instances

For the first experiment we want to investigate the performance of our algorithms on a set of random instances. We keep the instances general by not imposing any further restrictions, beyond the ones already discussed in Chapter 5. That allows for a good initial overview of the performance.

Setup

For the first set of experiments we used a calendar with 7 days and 24 slots per day. The maximum number of slots to block each day b was chosen to be 4. The number of events to schedule was chosen from $\{10, 15, 20, 50, 100\}$ and the number of people in the system from $\{10, 20, 30, 50, 100\}$. For each of those combinations an instance was generated and evaluated.

The distributed algorithms were implemented using the JADE (Java Agent DEvelopment) Framework [BBCP05, jad]. As the name suggests JADE is a framework for Java to assist with building multi-agent systems. For the CPLEX CP optimizer a time limit t of 20 minutes was set. Furthermore in both our ALMA-based algorithm as well as for the MSRAC approach we allowed the UserAgents to relay their true preference data in order to improve comparability.

Other parameters used in the following evaluation of our ALMA-based algorithm such as the back-off function f and its respective parameters, the number of slots to consider for the loss function, the preference scaling as well as possible discussed modifications to the strategy (see Section 4.2.3) were chosen with the help of the SMAC tool [HHLB11, sma]. SMAC (sequential model-based algorithm configuration) is an automated algorithm configuration tool that uses bayesian optimization at its core to find good hyperparameters for an algorithm.

Since hyperparameter tuning takes a long time we had to come up with a way to speed up the process. For that we started with a large configuration space for the parameters. Then we ran SMAC with an evaluation function that only evaluated each configuration on a single instance (with 50 events and 30 people). The results from that were then used to narrow down the configuration space. Afterwards SMAC was run again. This time we used 10 instances ($\{50, 100\}$ events each for $\{10, 20, 30, 50, 100\}$ people) to evaluate the parameters. Both optimizations were performed for several hours in order to yield adequate hyperparameters.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This resulted in the choice of f as the function motivated by the logistic function in (4.1) with $\gamma = 15.72$. The number of resources to consider for the loss k was chosen as 13 and we scaled the loss using the highest global preference. Furthermore all modifications mentioned in Section 4.2.3 were applied. I.e. after backing off, an agent would continue to monitor the same slot in the next round and only move on to monitor the next slot in the following round. We would also not remove slots from consideration after backing off unless they were already occupied.

Additionally we also ran tests for ALMA-Learning as described in 4.2.4. We used the parameters above for the initialization of the basic ALMA at the core of ALMA-Learning. Motivated by the parameters in the original paper [DF20], the length of the reward history L was chosen as 20, the learning rate α as 0.1 and the number of iterations for the algorithm as 512. Trying to use SMAC in order to find better values for these parameters was not very efficient due to long runtimes. However, after a small number of rounds no significantly better alternatives to the values above were found.

Social Welfare

Figure 6.1 displays the resulting social welfare relative to the CPLEX result. The decentralized and non-deterministic algorithms "ALMA", "decentralized greedy" and "MSRAC" ran 10 times for each instance. "ALMA-learning" also ran 10 times per instance, however, due to long runtimes this was reduced to 5 and eventually 3 runs for larger instances. The values on the figures represent the means and the error bars display the standard error of the mean. The dashed line displays the upper bound as determined by CPLEX and the blue area the possible values for the optimum.

The results show that our ALMA-based approach can consistently outperform the decentralized greedy algorithm and the MSRAC based algorithm most of the time. For smaller instances it stays roughly within 95% of the optimum. For larger instances we lie within 90% of the CPLEX result. Furthermore the results yielded from the ALMA-Learning approach were even better. They outperformed the centralized greedy result in most cases and mostly stayed within 95% of the CPLEX result.

Number of scheduled meetings

While we mainly focused on optimizing social welfare, we also wanted to take a look at the number of scheduled meetings that resulted from these tests. That is displayed in Figure 6.2. There, we can see the number of meetings scheduled for a given number of people in the system. Again, for the decentralized algorithm the points visualize the mean number of scheduled meetings and the error bars visualize the standard error. We only look at instances with 100 events as most other instances have been fully scheduled by all algorithms. The figure shows that both ALMA algorithms perform rather similar to the centralized algorithms. The decentralized greedy algorithm slightly outperforms the rest and the MSRAC algorithm generally yields the worst results for these instances.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

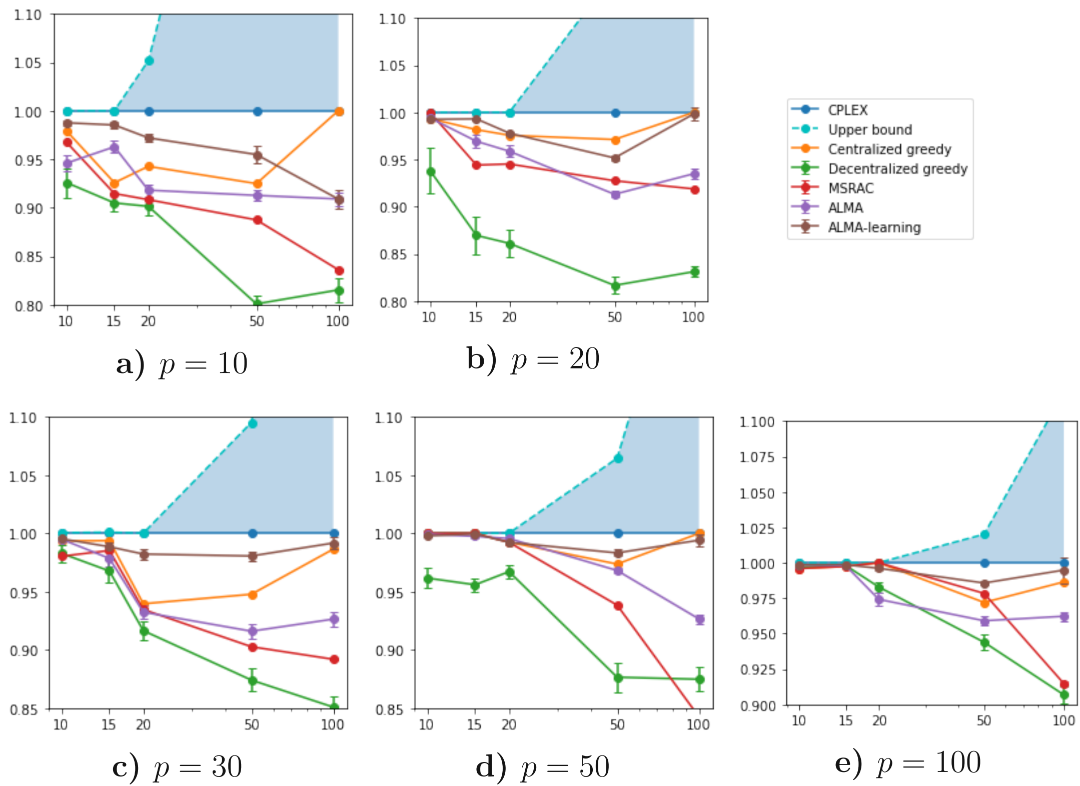


Figure 6.1: Evaluation of the algorithms using generated test data. p describes the number of people for each simulation. The number of events to be scheduled is displayed on the x-axis while the social welfare value relative to the CPLEX result is displayed on the y-axis.

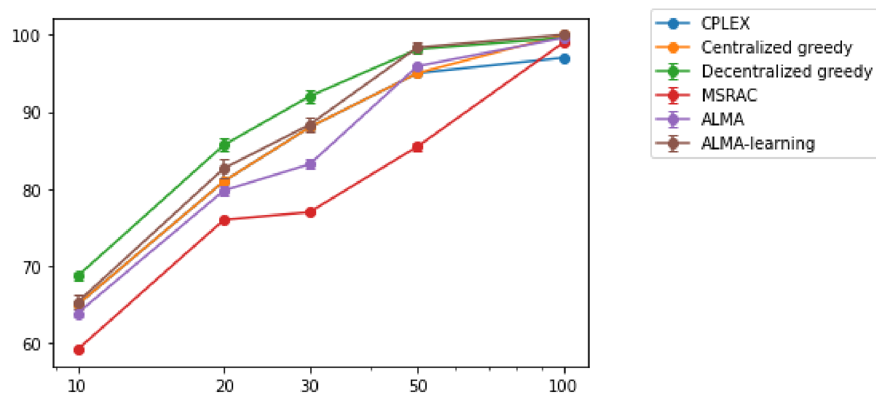


Figure 6.2: Count of the scheduled meetings of the algorithms using generated test data. The number of people in the system is displayed on the x-axis while the number of actually scheduled meetings is displayed on the y-axis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Fairness

One other metric we want to consider is fairness as measured by the Gini coefficient. The Gini coefficient is a fairly common measure used to demonstrate the (in)equality in a system. It was first introduced by Gini in [Gin12]. In order to calculate the Gini coefficient for our system we looked at the obtained results for all algorithms and for each result we took the utility for every person in the system. We then divided those utilities by the number of events a given person was originally assigned to and used those values to calculate the Gini coefficient as half of the relative mean absolute difference.

For the Gini coefficient, in general, a lower value indicates higher fairness. We look at this metric in order to investigate the fairness of our algorithm for each individual in the system while also taking the number of events each person may attend into consideration. After initially evaluating the Gini coefficient we could observe two interesting things about the fairness in connection with our data.

First, we could see that for a larger number of people and a low number of events the allocations, regardless of the algorithm, would yield high Gini coefficient values and would, with a growing number of events, converge towards a fairer allocation. For example for 100 people the Gini coefficient dropped from roughly 0.18 down to 0.08. This is expected since with a low number of events it is hard to meet the preferences of a large number of people. However, once the number of events grows, that effect will even out, eventually resulting in a fairer state.

Secondly, we observed that the Gini coefficient values, once converged, were rather low indicating very fair allocations. The reasons for that are not entirely clear and could be connected to the dataset or the structure of the problem itself. We theorize that this may at least in part be due to the nature of our generated dataset. Since all preferences for all people are essentially generated using the same distribution, it is very likely that a slot which satisfies one person's preferences might do so for most attendees. That would, in theory, lead to similar utility values among all people in the system and therefore result in a high degree of fairness. We also experimentally verified this by evaluating the fairness of datasets with more randomized data. Doing this resulted in lower fairness i.e. higher Gini values at around 0.15.

Since our goal is to compare the fairness between the algorithms, we calculated and then plotted the Gini values for the results of all algorithms for the given instances. These results can be found in Figure 6.3. Again, all Gini values are relative to the corresponding results from the ILOG CP Solver (CPLEX). The results show that while fairness was never directly considered in the design of our algorithm, the ALMA and ALMA-Learning approaches generally outperform the other algorithms for larger instances. Even the centralized approaches yield less fair results when given instances with a high number of events to schedule. Furthermore ALMA-Learning also outperforms the ALMA w.r.t. fairness in most cases.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

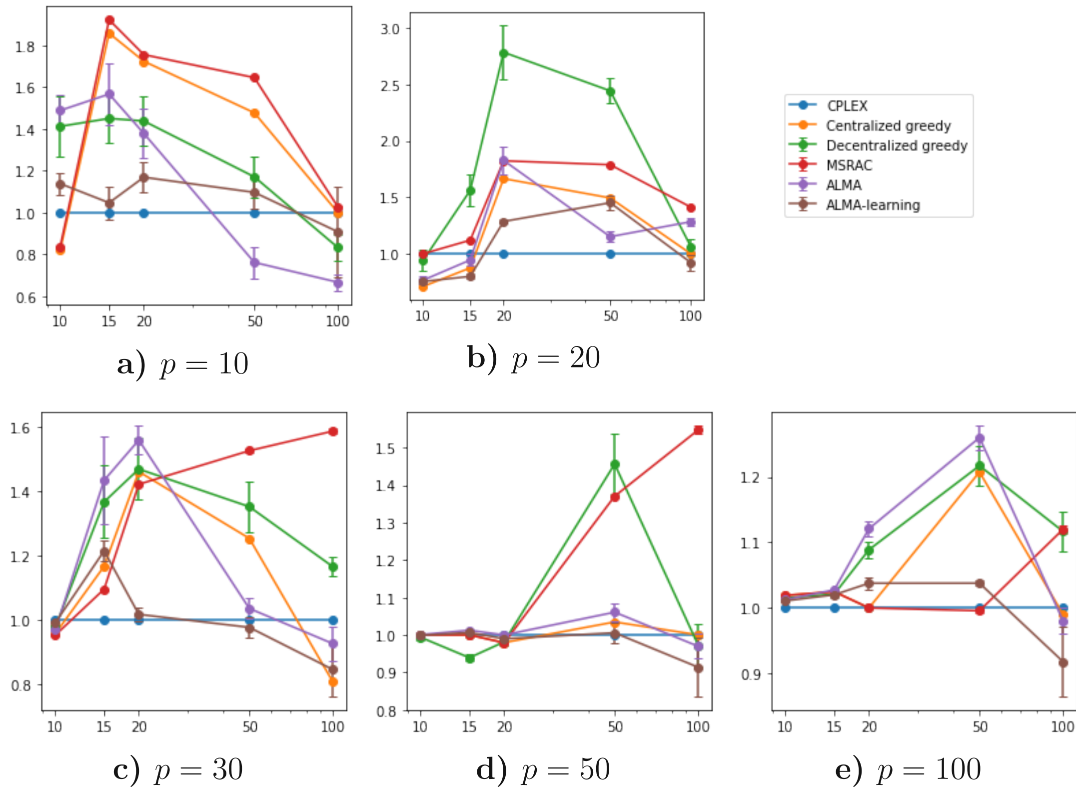


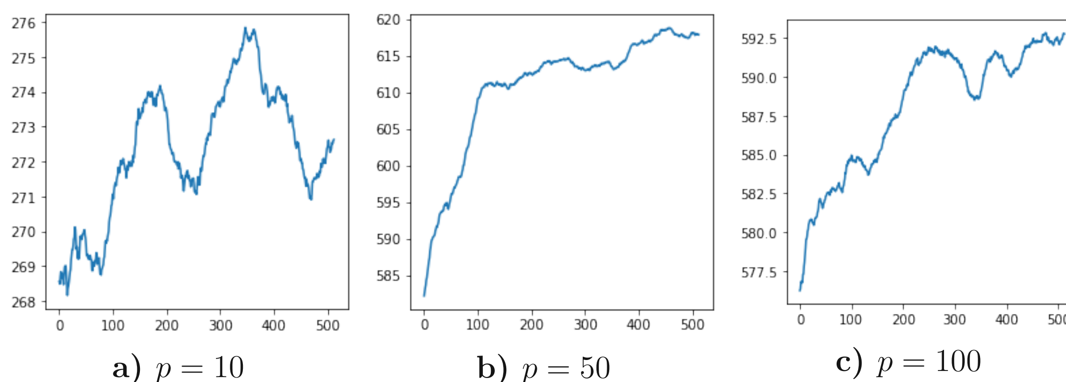
Figure 6.3: Evaluation of the fairness using generated test data. p describes the number of people for each simulation. The number of events to be scheduled is displayed on the x-axis while the Gini coefficient relative to the CPLEX result is displayed on the y-axis.

Learning rounds

Lastly, we also wanted to use the dataset to investigate the effect the number of rounds we use for ALMA-Learning has on our result. That is relevant since a high number of learning rounds significantly increases the runtime of our algorithm and therefore we would like to keep the number of learning rounds as low as possible. On the other hand we still need enough rounds to gain as much improvement as possible over the basic ALMA approach. For the evaluation we captured all intermediate social welfare results after each round. Figure 6.4 displays those results for a few instances.

From these plots we can see that, especially for a small number of people, the algorithm may not fully converge towards a value and that at some point larger rounds may not improve and can even worsen the result. The other plots show that while we achieve the most benefit in the first 100-200 rounds, a higher number of learning rounds becomes increasingly necessary for a large number of events. Looking at the case with 100 events we can see that we might even have gotten slightly better results by continuing to run the algorithm for a few extra rounds.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub



a) $p = 10$ **b)** $p = 50$ **c)** $p = 100$

Figure 6.4: Evaluation of how the number of learning rounds improves the result of ALMA-Learning. p describes the number of people for each simulation. We used 100 events in all simulations. The number of rounds is displayed on the x-axis while the moving average social welfare (using 32 datapoints) is displayed on the y-axis.

6.2.2 Larger Datasets

A big issue with our results so far is that we need to look at larger instances in order to make stronger statements about the performance of our algorithm. The reason why we did not consider larger instances can be found in Figure 6.1. There, we can see that the bound for the optimal value is simply too high for larger instances to make any statements about the performance relative to the optimum. Therefore in this section we want to look at a slightly different set of instances that ultimately allow us to give tighter bounds for larger problems.

Setup

For this test we wanted to create instances that are somehow easier to solve for the ILOG CP solver or at least allow for tighter upper bounds than the general problem instances in Section 6.2.1. The idea was that we start by solving two different problem instances with a low number of events, each for one week. We can then combine the two by simply duplicating the preferences for each week. This would then result in an instance with double the number of events to schedule over two weeks. The fact that the preferences in that case are periodically repeating is not too far fetched as it would be reasonable for someone to have the same preference to hold a meeting at a given day and time no matter the week. Furthermore we can combine the previous results from solving the two smaller instances to yield a good starting point for the combined instance.[2] Given the simplification of the instance and the starting point, we were able to find better upper bounds for larger instances.

[2]For this, we additionally need to make sure that no event exceeds the given calendar. Otherwise the combined schedule might not be valid.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

For this specific test we wanted to keep the calendar similar to the one in the previous section. Therefore we used the same parameters for the instance generator to generate 7 1-day long sub-instances. We then combined them into one larger 1-week long instance. This is also to keep the ratio of events to schedule to the size of the calendar comparable to the previous experiment. For this experiment we focused on instances where there were 100 people in the system. This way 4 instances were generated with sub-instances of 10, 20, 30 and 40 events each, resulting in combined instances of 70, 140, 210 and 280 events respectively. For the algorithms the parameters were chosen as in the previous experiment.

Results

Figure 6.5 displays the results of the evaluation on the new dataset. Just as wanted, the upper bound is significantly tighter. Especially when considering that we are dealing with much larger problem instances here as compared to Section 6.2.1. Here we can see that even for the new instances our approach outperforms the other decentralized algorithms. We can also see that we are still in a 90% range of the CPLEX results which would put us at roughly 85% of the optimal solution in the worst-case. For ALMA-Learning the results are even better with it being at worst at 90% of the optimal solution and coming close to the centralized greedy approach. It is, however, important to note that due to the simplified structure of these instances the results are not necessarily comparable to the ones from Section 6.2.1. When comparing the results from the previous experiments with 50 events to the significantly better results here with 70 events that becomes even more apparent.

Figure 6.6 again displays the relative fairness using the Gini coefficient. Once more we can see that ALMA-Learning is consistently fairer than the base version of ALMA and that it is at least as fair as the centralized greedy algorithm. However, we can see that the centralized ILOG CP solver now outperforms the other algorithms w.r.t. fairness.

6.2.3 Complexity experiments

In this section we want to experimentally verify our thoughts from Section 4.4. For that we want to look at the number of rounds that our ALMA-based approach needs to terminate for given instances. Here, we performed two sets of experiments. In the first we used a fixed schedule with 7 days and 24 hours and varied the number of people and events. For the second experiment we used a fixed number of 100 people and instead varied the number of days in our calendar as well as the number of events. We also adjusted the preference assignment depending on the number of days such that later dates would be more acceptable if the calendar was longer. For each given combination of parameters we then generated 10 random instances to solve.

Figures 6.7 and 6.8 display the results of these experiments. They show the mean number of rounds it took to solve these instances as well as the corresponding standard error. We can see that the number of people and the number of days do not significantly affect our

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

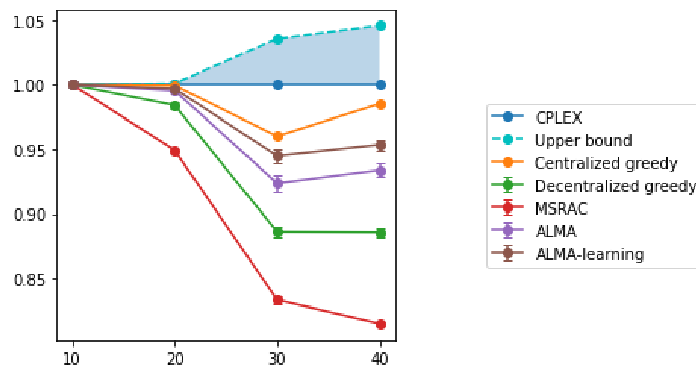


Figure 6.5: Evaluation of the algorithms on the new dataset. The number of people in the system is 100. The number of events to be scheduled per day is displayed on the x-axis while the social welfare value relative to the CPLEX result is displayed on the y-axis.

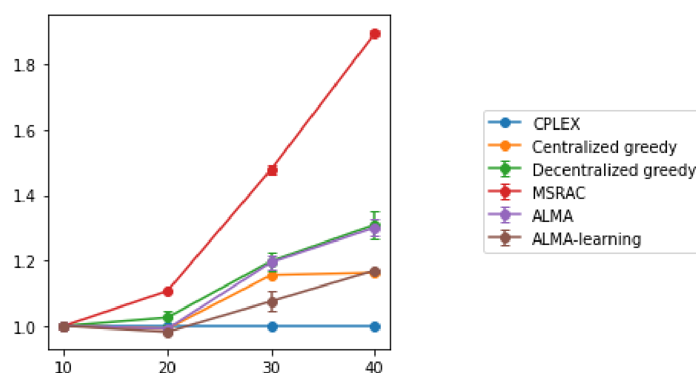


Figure 6.6: Evaluation of the fairness using the generated large testset. The number of people in the system is 100. The number of events to be scheduled per day is displayed on the x-axis while the Gini coefficient relative to the CPLEX result is displayed on the y-axis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

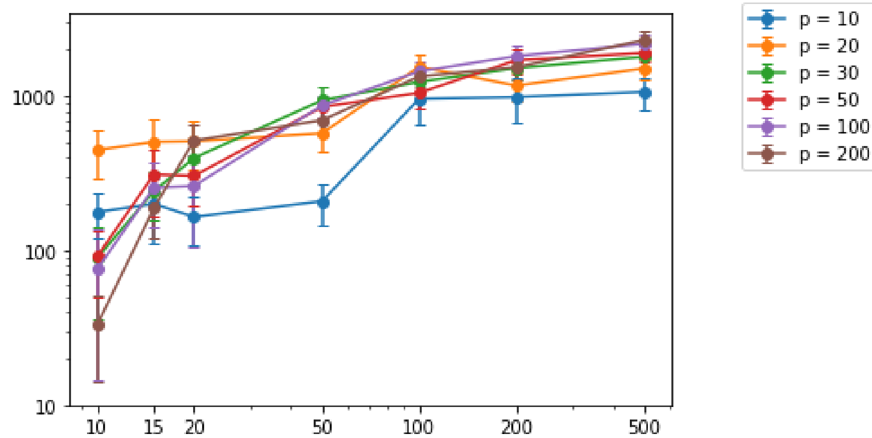


Figure 6.7: Number of rounds needed for the algorithm to terminate. p describes the number of people for each simulation. The number of events to be scheduled is displayed on the x-axis while the number of rounds is displayed on the y-axis.

complexity which is consistent with the observations from Section 4.4. Both plots also show a slowing growth in complexity with more events. This most likely stems from the clustering of our attendees (see Section 5.2) decreasing the number of possible conflicts and the punishment of high round numbers (see end of Section 4.4).

In both cases we can, however, still see a small increase in complexity with a growing number of events for any fixed number of people and fixed calendar. That could be due to the fact that simply clustering people does not completely limit the number of events competing for a timeslot with overlapping attendee sets N^* (see Section 4.4). Given that with a fixed number of people and a fixed calendar the density of events per slot and events per person has to grow, we expect more collisions and therefore a higher runtime or number of rounds.

6.2.4 Privacy Experiments

For the last experiment we wanted to look at the privacy of users. As mentioned before, we are already given a certain degree of privacy since there is no direct communication of preferences between EventAgents. However, in the previous experiments we still shared all preference data within an event group. As we also mentioned it is possible to further improve privacy by relaying noisy preference data or even just sending a ranking of slots to the EventAgent. One would, however, expect these measures to have a negative effect on the overall social welfare. In this section we investigate the effect those measures can have on the quality of our results.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

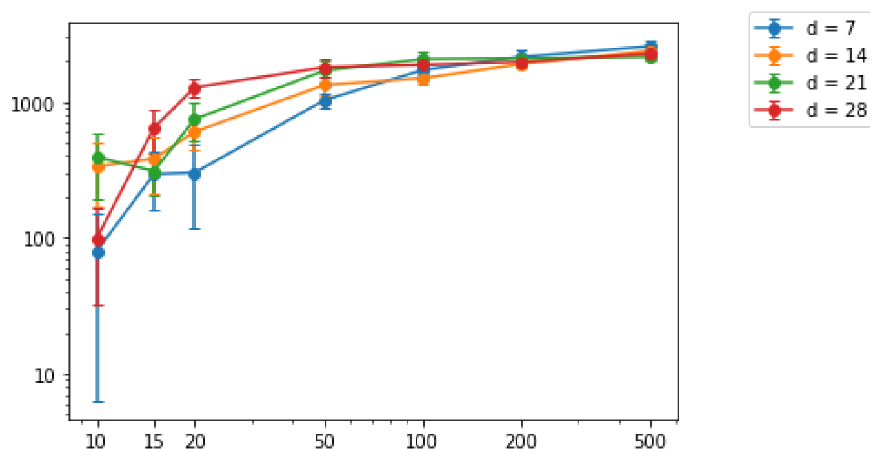


Figure 6.8: Number of rounds needed for the algorithm to terminate. d describes the number of days for each simulation. The number of events to be scheduled is displayed on the x-axis while the number of rounds is displayed on the y-axis.

Setup

For this experiment we looked at the following ways of obscuring the preference of a RepresentationAgent to improve privacy:

- **Ranking:** In this scenario the RepresentationAgents relay a ranking rather than their true preferences. This method was already proposed in [BH07] in order to provide privacy for the users. The EventAgent then creates a pseudo scale for each slot by assigning it a preference of $(numberOfSlots - rank + 1)/numberOfSlots$. We even add unavailable slots to that ranking as we can simply refuse to schedule them later on.

- **Gaussian Noise:** In this scenario the RepresentationAgents add gaussian noise with a standard deviation of σ to its preferences. This is a common method to improve privacy (see e.g. [BW18]). For the following experiments the considered values of σ were $\{0.05, 0.1, 0.2\}$. Any preference values exceeding our original interval $[0, 1]$ were rounded accordingly. Again we allow assigning nonzero preference values to unavailable slots and then refuse to schedule them if necessary.

Otherwise using the same parameters as in Section 6.2.1 we then evaluated the algorithm on the same dataset as in 6.2.1 once for ALMA and once for ALMA-Learning. For the latter algorithm, however, we chose slightly different parameters. In particular we selected the back-off function f to be the function based on a normal distribution ((4.3) in Section 4.2.2). The corresponding parameters for f were chosen to be $\mu = 0.46$ and $\sigma = 0.2$ with the number of resources to consider $k = 24$. These parameters are also the result of a SMAC hyperparameter search. While their performance, in terms of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

utility, were only marginally worse[3], they considerably decrease the number of rounds and therefore our runtime, which is greatly beneficial for ALMA-Learning. We used 5 evaluations per instance for ALMA-Learning.

Results

Figure 6.9 displays the results of each considered alternation to the ALMA algorithm as well as for ALMA-Learning. Focusing on the base version of ALMA we can see a decrease in performance, especially for high σ. We can also see that for small values of σ we got results within 90-95% of the baseline ALMA implementation. While the ranking approach worked rather well for a small number of events, it performed significantly worse for a large number of events. We can also see that in some rare cases with a small number of meetings we even managed to increase social welfare.

Focusing on the results of ALMA-Learning we can observe that the different privacy enhancing techniques have a similar effect as for ALMA. In fact, we can see that while in some cases ALMA-Learning still outperforms normal ALMA, sometimes the techniques used here cause ALMA-Learning to underperform as well. This is no surprise, as changing the preference values in the system also distorts the values we use to learn.

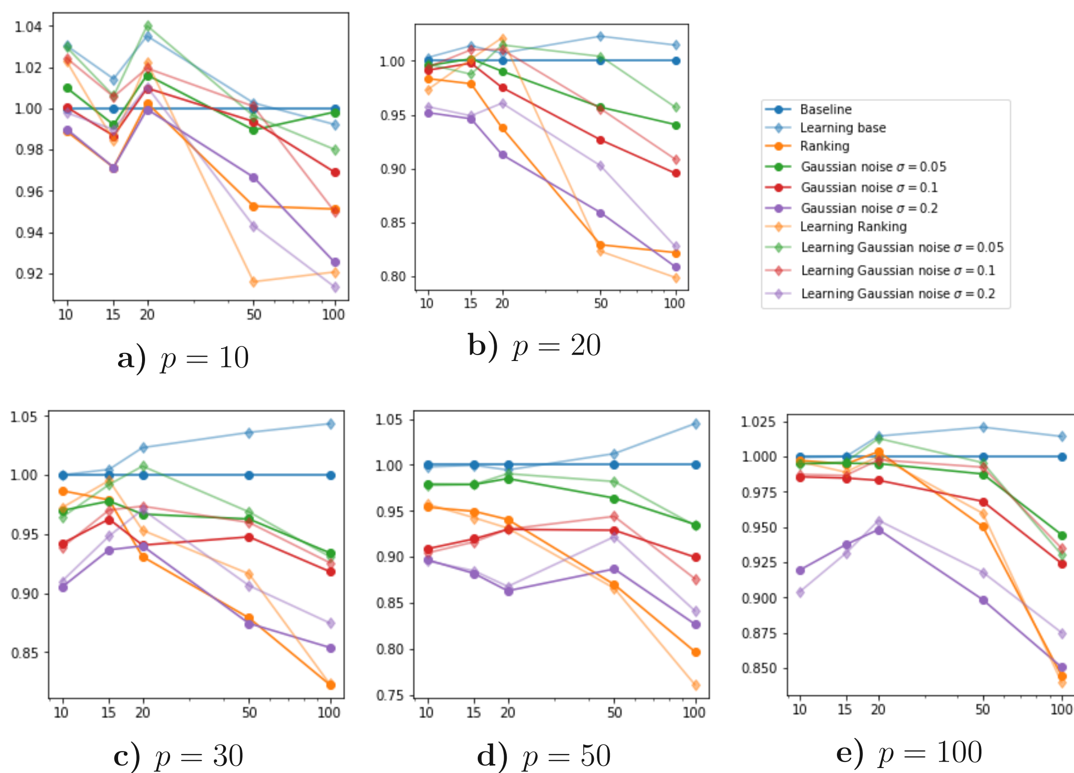
Fairness

Finally, we also want to visualize the effect these privacy improvements have on the fairness of our results. Figure 6.10 shows the Gini coefficient for all relevant scenarios relative to the one from an unobscured ALMA. There we can again see that, with exception of the ranking method, ALMA-Learning improves fairness when compared to their respective non-learning ALMA variants most of the time. Rarely we can observe an increase in fairness in our scenarios, however, most of the time, we can see that it leads to unfair results. Again, as expected, the effect is worst for high gaussian noise ($\sigma = 0.2$) and the ranking method.

These results again underline our previous thoughts about how there is always a tradeoff between the quality of the solution and the privacy of the users. We have, however, seen that under the right circumstances the solutions can still be good.

[3] When compared to the performance of ALMA-Learning with the parameters from Section 6.2.1 and the dataset from the same section the new parameters result on average in a utility 99.3% of the originally used one.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



a) $p = 10$ **b)** $p = 20$

c) $p = 30$ **d)** $p = 50$ **e)** $p = 100$

Figure 6.9: Evaluation of the ALMA and ALMA-Learning algorithms with obscured preferences. p describes the number of people for each simulation. The number of events to be scheduled is displayed on the x-axis while the social welfare value relative to the mean of the unobscured ALMA result is displayed on the y-axis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

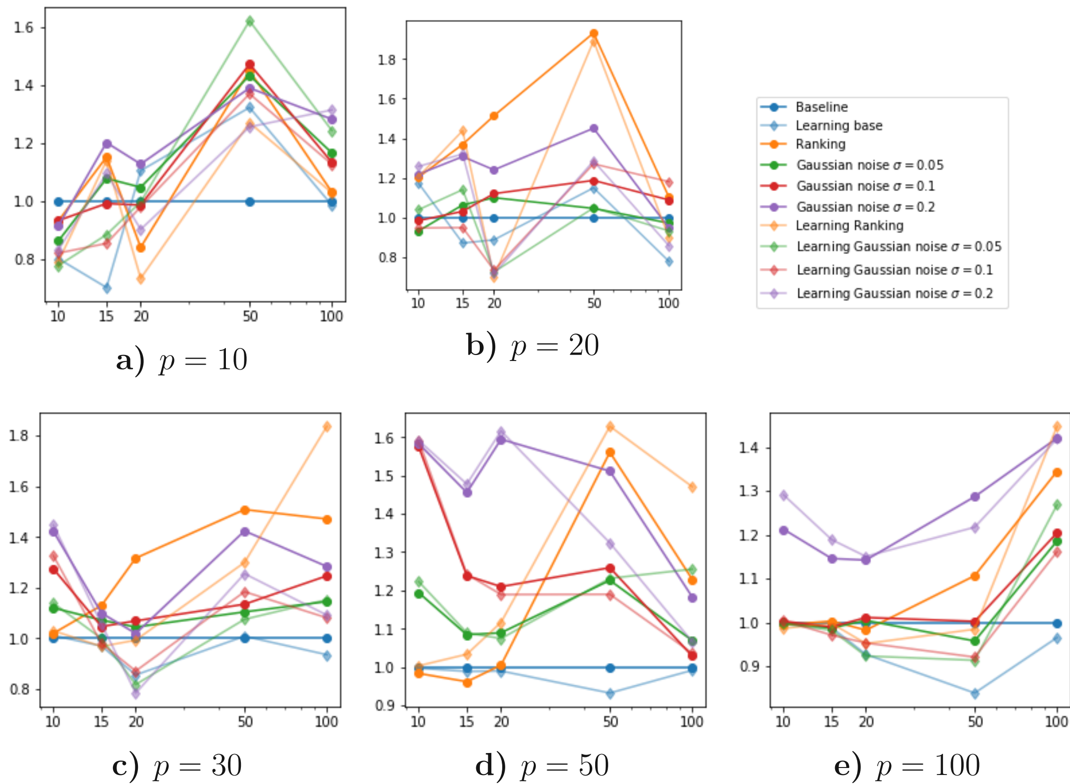


Figure 6.10: Evaluation of the relative fairness of ALMA and ALMA-Learning algorithms with obscured preferences. p describes the number of people for each simulation. The number of events to be scheduled is displayed on the x-axis while the Gini coefficient relative to the mean of the unobscured ALMA value is displayed on the y-axis.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 7

Conclusion

In this thesis we looked at the meeting scheduling problem in a distributed setting. We defined the problem by formulating it as a Constraint Satisfaction Problem. In addition we also investigated the complexity of the problem as well as the possibilities that our formulation provides.

Using a multi-agent system we then developed a distributed algorithm for solving the meeting scheduling problem. The algorithm is based on novel heuristic approaches and is later also improved by using a learning algorithm. This allows us to solve large instances of the problem within reasonable runtime. We also performed some theoretical analysis of the proposed algorithm. For that we focused on the convergence and runtime as well as privacy in order to underline the benefits of our algorithm. The analysis also served as a motivation for potential alternations to the algorithm that could improve the solutions even further.

To be able to properly test the algorithm an instance generator was designed with the goal of generating test instances that properly portray a real-life scenario for meeting scheduling in a corporation. In order to do that, the instance generator is based on data from real-life studies.

Finally, the algorithm was evaluated using the generated instances. Our approach showed promising results by staying close to the optimal social welfare in many scenarios while also scaling well for larger problem instances. We could observe even better social welfare results when using a learning approach. Other metrics such as the fairness as indicated by the Gini coefficient also showed that our algorithm provided highly fair solutions.

While this thesis discussed and studied the meeting scheduling problem and the possibilities of our algorithm, it is by no means exhaustive. Future work could further investigate privacy enhancing methods (e.g using PALMA) and their effects on solutions. Alternatives to ALMA-Learning, such as Bayesian Networks, could be explored to infer knowledge about other users in the system. Again one would have to consider the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

tradeoff between privacy and solution quality. Another aspect that might be interesting to look at, is the handling of byzantine agents. Namely, how an agent with knowledge of the algorithm could manipulate it by relaying specific preference values, in order to maximize its own personal preference and how the algorithm could be changed to prevent or discourage such behaviour.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bibliography

[BBCP05] Fabio Bellifemine, Federico Bergenti, Giovanni Caire, and Agostino Poggi. *Jade — A Java Agent Development Framework*, pages 125–147. Springer US, Boston, MA, 2005.

[BH07] Ahlem BenHassine and Tu Bao Ho. An agent-based approach to solve dynamic meeting scheduling problems with preferences. *Engineering Applications of Artificial Intelligence*, 20(6):857–873, 2007.

[BW18] Borja Balle and Yu-Xiang Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *International Conference on Machine Learning*, pages 394–403. PMLR, 2018.

[CPO] Ibm ilog cp optimizer. `https://www.ibm.com/analytics/cplex-cp-optimizer`, last accessed on 14.12.21.

[CPS] Ibm ilog cplex optimization studio. `https://www.ibm.com/products/ilog-cplex-optimization-studio`, last accessed on 14.12.21.

[DF20] Panayiotis Danassis and Boi Faltings. Learning to persist or switch: Efficient and fair allocations in large-scale multi-agent systems. In *ALA 2020: Adaptive Learning Agents Workshop*, 2020.

[DFRF19] Panayiotis Danassis, Aris Filos-Ratsikas, and Boi Faltings. Anytime heuristic for weighted matching through altruism-inspired behavior. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 215–222. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[DKJ18] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018.

[dMGS18] Rodrigo Rodrigues Pires de Mello, Thiago Ângelo Gelaim, and Ricardo Azambuja Silveira. Negotiation strategies in multi-agent systems for meeting scheduling. In *2018 XLIV Latin American Computer Conference (CLEI)*, pages 242–250. IEEE, 2018.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[DTF21] Panayiotis Danassis, Aleksei Triastcyn, and Boi Faltings. Differential privacy meets maximum-weight matching. In *ALA 2021: Adaptive Learning Agents Workshop at AAMAS*, 2021.

[DWF21] Panayiotis Danassis, Florian Wiedemair, and Boi Faltings. Improving multi-agent coordination by learning to estimate contention. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization, 2021.

[FFRW02] Maria Sole Franzin, EC Freuder, F Rossi, and R Wallace. Multi-agent meeting scheduling with preferences: efficiency, privacy loss, and solution quality. In *Proceedings of the AAAI Workshop on Preference in AI and CP*, 2002.

[Gin12] Corrado Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche. [Fasc. I.]*. Studi economico-giuridici pubblicati per cura della facoltà di Giurisprudenza della R. Università di Cagliari. Tipogr. di P. Cuppini, 1912.

[GM86] Fred Glover and Claude McMillan. The general employee scheduling problem. an integration of ms and ai. *Computers & Operations Research*, 13(5):563–573, 1986. Applications of Integer Programming.

[GPR+17] Mirco Gelain, Maria Silvia Pini, Francesca Rossi, Kristen Brent Venable, and Toby Walsh. A local search approach for incomplete soft constraint problems: Experimental results on meeting scheduling problems. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, pages 403–418, Cham, 2017. Springer International Publishing.

[GS96] Leonardo Garrido and Katia Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the Second International Conference on Multiagent Systems*, pages 95–102, 1996.

[HHLB11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization*, pages 507–523, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[jad] Jade. `https://jade.tilab.com/`, last accessed on 14.12.21.

[Kar72] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32, Mar. 1992.

[Mon74] Ugo Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information sciences*, 7:95–132, 1974.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[MTB+04] Rajiv T Maheswaran, Milind Tambe, Emma Bowring, Jonathan P Pearce, and Pradeep Varakantham. Taking dcop to the real world: Efficient complete solutions for distributed multi-event scheduling. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 310–317. IEEE Computer Society, 2004.

[NS20] Archana Nigam and Sanjay Srivastava. Oddms: Online distributed dynamic meeting scheduler. In Simon Fong, Nilanjan Dey, and Amit Joshi, editors, *ICT Analysis and Applications*, pages 199–208, Singapore, 2020. Springer Singapore.

[ODF12] Brammert Ottens, Christos Dimitrakakis, and Boi Faltings. Duct: An upper confidence bound approach to distributed constraint optimization problems. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[RN01] Nicholas C Romano and Jay F Nunamaker. Meeting analysis: Findings from research and practice. In *Proceedings of the 34th annual Hawaii international conference on system sciences*, pages 13–pp. IEEE, 2001.

[SD91] Sandip Sen and Edmund H. Durfee. A formal study of distributed meeting scheduling: Preliminary results. *SIGOIS Bull.*, 12(2–3):55–68, October 1991.

[sma] Smac. `https://www.automl.org/automated-algorithm-design/algorithm-configuration/smac/`, last accessed on 10.11.21.

[YDIK98] M. Yokoo, E. H. Durfee, T. Ishida, and K. Kuwabara. The distributed constraint satisfaction problem: formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, Sep. 1998.

[ZC09] Alejandro Zunino and Marcelo Campo. Chronos: A multi-agent system for distributed automatic meeting scheduling. *Expert Systems with Applications*, 36(3):7011–7018, 2009.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub