King's College London

6CCS3PRJ Final Year Individual Project Report

Author: Barnabas Szalai

Supervised by: Thomas Radzik

Student ID: 20046752

November 27, 2023

1 Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Barnabas Szalai	**November 27, 2023**

2 Abstract

2.1 Brief Summary

This report is about Invitely, which is a mobile application designed to allow medium to large sized teams to efficiently bulk schedule meetings. Through the app, users are able to join and manage teams, as well as construct and schedule meetings by inviting their colleagues. The major component of the app is the scheduling algorithm, which schedules multiple meetings at the same time, generating a more optimal solution than the traditional iterative approach. In addition, the report will also highlight, describe and evaluate the other components of the software system, which are mandatory for the proper functionality of the application.

2.2 Acknowledgements

I would like to express my sincerely gratitude for the work of my supervisor, Dr. Tomasz Radzik. He was extremely helpful and motivated throughout the entire year.

Contents

3 Introduction

3.1 Aims & Objectives

The aim of my project is to develop a meeting scheduling system that is able to generate efficient meetings based on some broad constraints specified by the host, and the preferences of each individual invited participant. The system will differentiate between individuals, based on their roles and priorities inside the team, and will make a decision on a wide range of considered factors. The system will consider the existence of previous meetings and will construct meetings that satisfy as many constraints and preferences as possible.

Even though there has been a lot of work on scheduling systems for individual meeting requests, the functionality that will make my algorithm different and innovative is its ability to schedule multiple meetings at the same time. The system will be able to consider as many meeting requests as required and schedule them at the same time, keeping in mind all of the hard and soft constraints of every stakeholder.

Scheduling multiple meeting requests at the same time allows the algorithm to maximise for the global optimum; thus, generating an objectively more advantageous result for the society (the environment where all the stakeholders coexist).

The aim of the project is not only to generate an algorithm that

will allow individuals to schedule meetings without the irritating need of sending out enormously complex group emails, but also to generate meetings in a way that benefits the most amount of people by disregarding the local objective of individual meeting requests and working towards maximizing global utility.

3.2 Preference Based Meeting Scheduler

A critical part of the functionality of my algorithm is taking into account individual preferences of the host and the invited participants when bulk scheduling the meetings.

To successfully develop an algorithm that generates meetings considering all of the relevant preferences and factors, I first had to determine what information should the system use. The first publication on group scheduling that is relevant to my project was carried out by Leysia Palen, at the University of Colorado in 1999. The paper mentions that the traditional, 'free/busy' response from invitees is way too basic to determine "users actual preferences" [7, p. 2]. Nawaz et al. came up with a system that they call the Semantic Meaning Scheduling System (SMSS). According to the paper, SMSS is a system that relies on Ontology in order to take into account the relevant stakeholders' preferences and constraints to represent the domain knowledge when scheduling meetings. The system that Nawaz describes relies on a web-based interface, where the users can eloquently input their preferences.

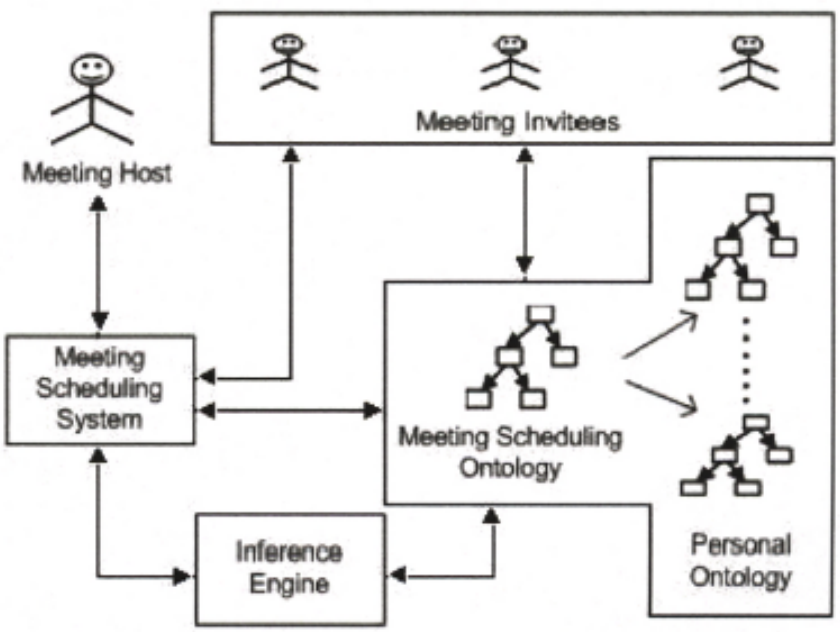

Figure 1: SMSS Structure (Nawaz et al.)

According to SMSS, the factors that determine the resulting generated meetings are derived from following:
1. The broad constraints defined by the host.
2. The Personal Ontologies of the attendees.

The personal ontology of a participant is made up of three main factors:
1. Roles (Director, Intern, etc ...) – each role has a priority defined by the team leader
2. Activity (Researcher, Marketing, HR, Student, etc ...)
3. Preferences (Temporal, Non-Temporal)

Temporal Preferences
Include the duration, time range and days
Non-Temporal Preferences
Include the density, participants, interests and overlaps

Both temporal and non-temporal preferences are prioritized by the participants. This allows each individual to declare their main priorities. For example, for one person, it is more important that the meeting is short than the number of meetings he or she has a day. When calculating efficient time slots, there are 5 main factors that are considered:

- Preference Weight (W)
- User Priority (U)

- Potential Time Slot T^n on Request Day
- Total # of Fulfilled Preference (m)
- Total # of Attendee for Requested Meeting (n)

$$\textbf{Weight of Time Slot } T^n = \sum_{i=1}^{n} \sum_{j=1}^{m} \mathrm{U}_i \mathrm{W}_{ijN} \tag{1}$$

When the weight of each time slot is identified, the algorithm will optimise for maximum utility and generate meetings for time slots that have the highest total weight. This algorithm is particularly useful for my code, as I will also take into account participants' individual preferences; however, there are some significant modifications that I will have to implement into my code to efficiently handle scheduling multiple meetings. One such alternation is the management of relationships between participants across multiple different meetings. In the single request algorithm (shown on Figure 1) the differences in user priorities and preference weights are only compared between participants who are invited to the same meeting. However, when scheduling multiple meetings, these weights and preference scores have to be observed for all participants, because in the event of a meeting with highly important participants, the scheduling of other meetings, with less important participants are not prioritized.

3.3 Advantages of Batch Scheduling

At this point in the report, I believe that it is important the describe why scheduling multiple meetings at the same time leads to a better

overall result, then scheduling meetings one-by-one after each other aiming to maximize the efficiency of the individual meetings.

Considering the global maximum instead of optimizing for the local maximum has the advantage of increasing the total utility of all the stakeholders. Even though the algorithm will schedule some meetings below their local potential; overall, the resulting schedule will lead to the highest benefit to society in terms of the predetermined advantages. (This also demonstrates the importance of giving value to the right attributes, and setting the optimisation scores and penalties properly). As an example, lets say meeting request #1 was scheduled at an interval where it does not overlap with any other meetings. However, meeting #5 and meeting #6 are more important for the same participant who also needs to attend meeting #1, but meeting #5 and meeting #6 got scheduled on top of each other because meeting #1 was scheduled first. In situations like these, it is imperative to consider all of the meetings to be scheduled at the same time, and prioritize scheduling meeting #5 and meeting #6 before deciding on meeting #1.

3.4 Global Optimization

In order to further understand the benefits of finding the most optimal solution that considers stakeholders across multiple meetings, it is imperative to define the meaning of global optimization. Global optimization is the method of searching for the best possible solution to an optimization problem over the entire feasible space, instead of

finding a local optimum. As it is clearly visible on Figure 2, finding the global optimal value can significantly improve to solution compared to the local results. Global optimization has importance if there are multiple local optima.

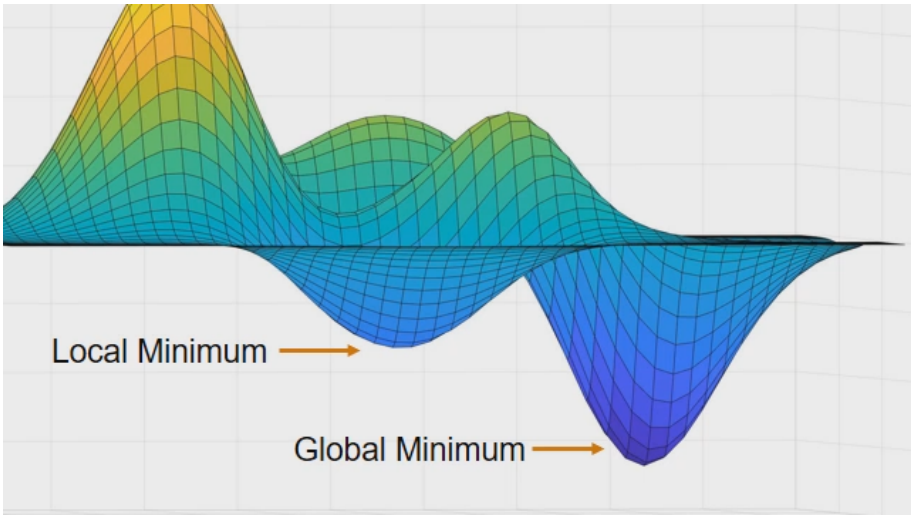

Figure 2: Global Minimum [16]

In order to set up a global optimization model, it is required to define a continuous objective function and continuous constraint functions. These assumptions are based on the Extreme Value Theorem, which states that a continuous function defined on a closed interval must have at least one global maximum and one global minimum. [16] Assuming that f(x) is a continuous function in a closed interval, there must exist two points c and d in the interval such that:

- f(c) is the maximum value inside the interval of f(x) and
- f(d) is the minimum value inside the interval of f(x)

Global optimization problems are considered NP-Complete, mean-

ing that they can be solve in non-polynomial time. As a result they are extremely challenging and the amount of time it takes to solve them (even with an excellent choice of algorithms) grows exponentially to the problem size.

3.5 Constrained Optimization

In order to find the global optimum in a given feasible space, it is imperative to outline well-defined constraints in addition to the objective function. Constrained optimization is the process of finding the optimal solution to a given problem. The objective function to be optimized can be either maximized or minimized, and the constraints can be either equality constraints (e.g., $x + y = 1$) or inequality constraints (e.g., $x \geq 0, y \leq 0$).

Constrained optimization problems can be classified into two types: linear and non-linear. In linear constrained optimization problems, both the objective function and the constraints are linear functions of the decision variables. Non-linear constrained optimization problems contain non-linear functions either in their objective function or in their constraints. Non-linear constraint optimization is considered challenging, as they deal with many local optima.

3.5.1 Interval Based Optimization

In scenarios when the upper and lower bounds of the solution set for the constraints can be calculated, interval based optimization can be utilized to take control over unwanted errors as a result of wrongly

used decimal digits [14]. Setting specific and narrow bounds for variables, it is possible to reduce the search space; thus, decreasing the possible solution set, which results in faster search times. Interval constraint optimization is a generalization of bound-constrained optimization, which considers the case where the input parameters have a known range of values. In interval constraint optimization, the input parameters are represented as intervals, which are a range of possible values. The goal of interval constraint optimization is to find the optimal solution that satisfies all of the constraints, regardless of the exact values of the input parameters. This can be done by considering all possible values of the input parameters within their respective intervals, and evaluating the objective function for each combination of values. The solution that produces the best objective function value across all possible combinations of input values is then considered the optimal solution. It is clearly visible, that searching for all combination of values can be extremely time consuming, especially when dealing with a large number of variables, with each having broadly defined constraints. As a result, it is imperative to define all known constraints to aid the solving algorithms by narrowing the search space.

In addition, it is also possible to take advantage of heuristics, that eliminate variable combinations that are infeasible. Heuristics work by iteratively exploring the solution space and refining the search based on the quality of the resulting solutions. Hybrid algorithms combine the strengths of different optimization techniques to solve complex optimization problems. They combine meta heuristics and

local search methods to achieve a balance between exploration and exploitation of the solution space.

Even though Interval Based Optimization outlines excellent potential solutions for scheduling problems, it is not the most optimal method for my problem due to the fact that it deals with real numbers. When scheduling meetings, it is perfectly fine to work and express every variable in minutes, which are integers. As a result, it is possible to express the meeting scheduling optimization problem as a combinatorial problem.

3.6 Constraint Programming

3.6.1 Combinatorial Problems

Combinatorial problems are optimization problems that deal with finding the most efficient solution from a discrete solution set. It works by choosing a combination or permutation of elements from a finite set of choices, subject to a set of constraints and objectives. The majority of combinatorial optimization problems are NP-hard, meaning that the amount of time it takes to solve it grows exponentially with the size of the problem. It is connected to the Constraint Satisfaction Problem (CSP), which also works with a discrete decision set and deals with finding a solution that satisfies all determined constraints. However, CSPs do not intend to find an optimal solution to a particular problem, therefore they do not require the specification of an objective function.

A traditional example of a combinatorial optimization problem is

the Traveling Salesman Problem (TSP). TSP aims to calculate the shortest route that visits a set of cities exactly once and ends up at the starting point. Another optimization problem worth mentioning is the knapsack problem, which involves selecting a number of items from an item list that needs to be packed in a knapsack, given a weight limitation and a value function.

Constraint Programming (CP) is a paradigm that is used for efficiently solving combinatorial problems. When using constraint programming, one must clearly define the bounded discrete variables, constraints on those variables and an objective function. Then the solver will find the most optimal solution from the feasible solution set. CP is an adequate technique for problems that can be illustrated at an optimal mapping of an ordered set to another ordered set, where variable relationships between the sets can be formulated as mathematical expressions that are consequential to the objective function [11]. My problem, of determining efficient meeting schedules, can be formulated as such mapping.

A key advantage of CP is its utilization of domain-specific inference methods, called constraint propagation. These techniques reveal information in the problem that help the algorithm reduce the search space, which is often extremely large. [17]. One such method is called inference, which deals with deriving desired implications from a set of constraints. Inference reduces the search space through reducing the domain of the solutions set [15]. There are many methods that are currently used to achieve inference. One of the most im-

portant algorithms is arc-consistency, which makes sure that each and every value in a variable's domain stays consistent with at least one value in the domains of its neighbouring variables. Another imperative and useful algorithm, which I will also be applying is the nogood learning method. This algorithm involves learning from past failures during the search and using that information to avoid future failures. As the elimination of potential variable assignments occurs during the search process, this method can be particularly efficient, by only generating reductions that are actually meaningful and efficient. I will discuss nogood learning in greater detail later in the report where I explain the functionality of the algorithms I used. Furthermore, I will also demonstrate the advantages of the mentioned techniques, when I compare constraint programming optimization for batch scheduling with a greedy approach.

3.6.2 Relaxation

Another tool that CP solvers use is called relaxation, which is a simplification technique that is used to loosen or 'relax' some hard constraints [15]. By temporarily ignoring some of the predefined constraints that would otherwise be challenging to satisfy, a feasible solution can be found that meets the majority of the remaining constraints. Treating such constraints as soft constraints by introducing slack variables, it is possible to find solutions that are close to the original optimum. Slack variables can be used to represent relaxation by allowing some degree of violation or deviation from the original constraints. For example, if we have a constraint that

requires a particular variable to take on a specific value, we can introduce a slack variable to allow the variable to deviate from this value within some predefined tolerance. In case of a minimization function, the slack variable defines the elasticity of the constraints from the minimum towards upwards, because there can be no better solution than the optimal minimum.

To demonstrate relaxation with an example, imagine a problem where we need to assign three tasks to three workers, but each worker has a different skill level and each task requires a specific level of skill. The objective is to minimize the total duration it takes to complete the tasks. To represent this problem, the following constraints are defined:

- The skill level of the assigned worker must be greater than or equal to the task's required skill level
- Each task must be assigned to exactly one worker.
- At most one task can be assigned to each worker

In reality, it is often not possible to find a solution that is inside all of these constraints; therefore, a slack variable is introduced to the first constraints (lets say 20%) that will allow the skill levels of some workers to be maximum 20% less than the required. However, it is possible to minimize the total difference that is added up from the cumulative differences of the required skill levels.

Relaxation has the potential to find a feasible solution when it would

be impossible to find one otherwise. On the other hand, it is imperative to be cautious with slack variables, as they may allow the algorithm to generate solutions that do not align with the requirements.

Soft constraints will be an imperative element of my scheduling algorithm, as they will allow me to schedule meetings when participants' calendars are extremely busy. When there are many meeting requests and most invited participants have a busy schedule, it might be infeasible to create a schedule which will work for every individual. Therefore, it will be imperative to introduce slack variables, that will allow the meetings to not work for some participants, while making sure that it works for as many people as it is possible.

3.6.3 Search Algorithms

CP search algorithms use the solution space (defined by the variable constraints) to identify feasible and optimal solutions for a particular problem. These algorithms utilize a combination of algorithms, such as constraint propagation, to systematically narrow down the possible solutions to find solutions that do not violate hard constraints. Tree search is a frequently used algorithm for finding feasible CP solutions. Tree search algorithms operate by building a search tree, where each node represents a partial assignment of values to the variables and each edge represents a possible value for the next variable. There is a lot of research on tree search algorithms; however, A* is probably the most notable one. A* Search utilizes the heuristics of Best-First Search (BFS) with a cost function that considers

the cost of the current partial assignment as well as the predicted cost of the remaining variables. A* Search is especially effective when used for large search spaces.

3.6.4 CP Solvers

Constraint programming solvers are tools that are designed to solve complex optimization problems. There are many open-source and commercial softwares that utilize a combination of search algorithms to aid with the solving of problems with extremely large search spaces. Such solver tools include BM ILOG CPLEX, Gecode, Choco and MiniZinc. However, I will be using Google's Or-Tools software suite, which was developed to solve constraint programming problems. It was developed in C++, but has wrappers for python, which I will use to define my problem set and access their solvers.

3.6.5 Lazy Clause Generation

Lazy Clause Generation (LCG) is complex optimization algorithm that utilizes the combination of Boolean Satisfiability Problem (SAT) and CP. It works by generating conflict clauses on-demand, thus the term 'lazy'. The algorithm differs from traditional SAT solving algorithms in that clauses are not generated upfront (CNF), or through conflict-driven clause learning (CDCL), rather they are only generated when their existence is mandatory. LCG generates clauses at the moment when a conflict occurs during the search process.

When initial clauses are generated, they are used to produce an implication graph, which is a data structure that aids the search pro-

cess by providing relationships between assigned variables and the propagation of these assignments through constraints [13]. The implication graph is primarily utilized in the event of conflict analysis. The graph's directed edges guide the propagation of assignments as a result of constraints [13]. An edge from node X to node Y demonstrates that the assignment in node Y resulted from the assignment in node X through constraint propagation. Upon a conflict (inconsistency in the assignments), the implication graph is utilized to perform conflict analysis, which aims to discover the conflict's reason and produce new conflict clauses that represent that particular reason [13]. This results in more precise and relevant conflict clauses that constantly reduce the search space.

An example that Peter J. Stuckey, a professor at Monash University, used to explain this continuous efficient clause generation is the following:

$$\text{[13]} \quad \begin{array}{c} \text{consider x(1, 20) + y(5, 10)} \leq \text{z(10, 30)} \\ z \leq 24 \text{ does not propagate} \\ x \geq 10 \text{ propagates } y \leq 14 \\ \textbf{stored as } [x \geq 10] \ \textbf{\&\&} \ [z \leq 24] \ \textbf{implies} \ [y \leq 14] \end{array} \tag{2}$$

This propagation demonstrates both the advantage and the disadvantage of LCG. LCG generates additional clauses during search, when it is able to exclude areas of the search space. In order to

efficiently generate the additional clauses, it has to explain precisely every step of the propagation. This explanation adds complexity to the algorithm; however, in large problem spaces, it becomes extremely efficient.

In addition to generating additional explanations, LCG also takes advantage of a technique called nogood learning, which I have mentioned briefly already. Nogood creation takes place through analyzing conflicts and examining the assignments that led to the conflict. As a result, the algorithm is than able to formulate a logical expression that portrays the negation of the two assignments. This representation defines value assignments that should not occur together in future solutions [13]. Determining potential assignments to be included in the event of conflict is carried out by an approach, called First Unique Implication Point (1UIP). This technique decides about the assignments that led to the conflict through tracing back preceding assignments. 1UIP aims to identify the first unique implication point, which is the assignment that is exclusively accountable for the conflict, which is used to generate a nogood from itself to the conflict [12].

Let's assume the following:

$$\begin{gathered} x_2 \leq x_5 \\ [12][x_1, x_2, x_3 \textbf{ and } x_4] \text{ are all different} \\ x_1 + x_2 + x_3 + x_4 \leq 9 \end{gathered} \tag{3}$$

As it is clearly visible on Figure 3, $x_2 = 2$ is the 1UIP, as it is

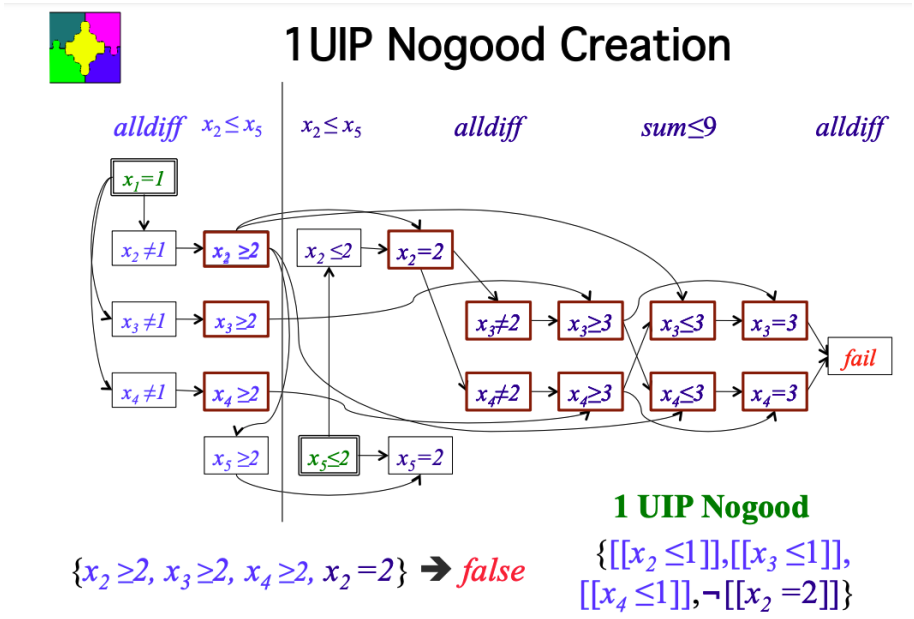

Figure 3: 1UIP NoGood Creation [12]

responsible for the conflict; therefore, a nogood is created in the form of:

$$[12]\{[[x_2 \leq 1]], [[x_3 \leq 1]], [[x_4 \leq 1]], \neg[[x_2 = 2]]\} \tag{4}$$

Even this basic representation of the mechanics behind the Lazy Clause Generation method illustrate how effective it can be to generate inconsistencies during search, and thus significantly removing the search space in the areas that are the currently most relevant.

3.7 Economic, Commercial, and Customer Context

Understanding different approaches to scheduling is imperative in the design phase of a new software. Search methods do not only

impact the usability of applications through their search time and space complexities, but will also give different results based on the desired outcome of each algorithms. Personally, in order to build a meeting scheduler that is able to efficiently and sufficiently produce calendars that provide a true value to my users, I have to include algorithms that are not only capable of collecting, combining and utilizing all of the invited users' data, but can do it with precise accuracy and efficiency. Optimizing meetings across the calendars of multiple participants, across multiple days requires calculations in non-polynomial time. As a result, I made sure to identify algorithms, such as the Lazy Clause Generation method, that are able to deal with such complexities, and are capable of managing enormous problem spaces.

4 Background

4.1 History of Meetings

Meetings have always been an imperative element of our lives. Large corporations, educational institutions, hospitals and many other industries utilize meetings to discuss ideas and come up with solutions to current problems. Before technology was widely utilized, the date and time of the occurrences of these meetings were discussed in-person prior to the meeting, or they were agreed on during the group's previous meeting. In Ancient Rome, meeting etiquette was developed, and a common procedure for meetings was agreed on, which is the foundation of meetings today.

Back than, meetings were held in communal spaces, and philosophical ideas were discussed by the people of the town. However, when offices were introduced in the 20th century, the meetings started to follow a hierarchical structure, where the topic, date and time of the meeting was idealized by the leader of the group [9]. The spread of technology in the 21st century allowed offices and people to hold meetings online, which made communication inside a team more available to everyone. Furthermore, as a result of the pandemic, there was a major shift to remote working and online meetings, which many believe made communication even more available to everyone. Today, it takes 1 minute to send somebody a Zoom invite and start having a conversation with them.

Even though a lot has changed in the format and availability of meetings, the overall concept and principle did not change since the Romans. The topic, date and time of the meeting is discussed prior to the event (usually by the team or project leader), and during the meeting, people raise their hands, comment on ideas and hold voting.

As I indicated above, the traditional way meetings have always been scheduled is extremely unidirectional. Person A, sends person B, C and D a Microsoft Teams invite - which already includes the meeting title, date and time - which person B, C and D can decide to either accept or decline. This format of scheduling requires person A to have complete and perfect knowledge on the preferences of person B, C and D. Even though person A could ask person B, C and D - either in-person, or by sending them a message - about their preferences, this usually does not happen, and if it does, it is rather challenging to decide for person A on the best option for the meeting if either person B, C or D has different preferences to his/her initial idea about the meeting. Additionally, scheduling meetings and collecting individual attendee preferences in the traditional way is almost impossible in a large number of invitees. Accumulating the preferences of 25 people, evaluating all of them and deciding on which date and time is the most appropriate for the meeting is almost impossible.

4.2 Corporate Meetings Today

As of now, in most companies, usually an email is sent out with every invitee CCd, asking them whether a particular date and time is plausible for them. After, each participant replies whether they are happy with it, or whether they would prefer a different time. This way of scheduling is extremely time consuming. Considering the priority of each individual in the team, identifying people who work on the same topics, and making sure to have a representative of each sub-department, and many more imperative factors is technically unfeasible. Even though there are there several apps that allow users to identify time slots where the most people are available or hold votings on preferred time slots, current software fail to provide the ability for teams to schedule efficient meetings on a macro level. Finding the best time slot for a meeting request is important; however, if that meeting is not as important in general, it is more advantageous to prioritize scheduling other meetings first.

4.3 Existing Work

There are many scheduling applications on both the App Store and Google Play. There are also many applications that provide a platform for hosting and carrying out meetings, such as Microsoft Teams and Zoom, to name the most famous ones, which contain some sort of scheduling functionalities. In order to identify the strengths and weaknesses of currently available applications, I constructed a benchmark for the ones that I believe are most relevant to the app that I am developing.

Calendly

Calendly is a 'user-friendly scheduling system' [18]. At the most basic level, the app is extremely simple to use. Users can create their own calendars, and share an invite link. Using this link, individuals can schedule 1:1 meetings with the person, and it automatically adds it to both of their calendars, and changes their availabilities. It also includes a team scheduling option, where individuals can select who they would like to have a meeting with inside the team. It is efficient, because the entire team uses the same invite link. Even though this application is widely used and extremely user-friendly, it is very different from my planned application. The only elements that I might be able to benefit from are their UX design choices for displaying meeting options and calendars.

Doodle

A scheduling application that allows users to organize meetings of many different types. The main focus of the app is sending out 'group polls' with a list of meeting date/times. Invitees than vote on which meetings work for them, and the time that most people voted for is selected by the system. Doodle's main objective was to eliminate the 'endless email chains', which is also an imperative priority of my application. Doodle is one of the most widely used meeting schedulers with more than "30 millions users per month" [5]. Even though Doodle is probably the most similar application to the one I am planning on developing, it lacks many features that

will make my software unique. First of all, in Doodle, the host actually specifies some meeting times that the invitees are able to choose from, which is not as bidirectional and democratic as my application will be with the loosely defined initial constraints. Furthermore, even though Doodle has an option to create teams and organize team meetings, it doesn't specify team roles and activities that differentiate between individual team members, and also it doesn't establish relationships between team members that can be utilized to generate accurate predictions. As a result of Doodle's feature of specifying a number of meeting time options, the recommendation that is generated after the invitees responded is also weaker than it will be in my system. My application will be able to generate recommendations that were not even considered by the host when he/she sent out the invites. Lastly, the most notable difference will be visible in my application's batch scheduling system. Scheduling multiple meeting requests at the same time will make event scheduling inside teams more efficient than it was possible on any other platform so far that only considers individual meetings.

SavvyCal

This modern scheduling application is very similar to Doodle. The biggest difference between the two apps is present in the way it schedules 1:1 meetings, which is a feature that I am not implementing. The only difference between SavvyCal and Doodle in scheduling team meetings, is that is displays the users calendar when they are to choose from the proposed meeting times in the meeting polls. One

thing that this application does extremely well, is that it always shows the users their current schedule, which helps them identify their available times and minimizes the chance of double-booking. It also includes a feature called 'collective scheduling' [6], where organizers can see time periods when multiple users in the same team are free at the same time. My focus will be to develop a truly democratic meeting scheduler; therefore, I will make sure that the users are aware of when their teammates are available, so that they can have an influence on selecting time periods that benefit everybody.

4.4 Scope

4.4.1 Mobile Application

In order to construct a system that is both efficient and useful, I decided to develop a mobile application. To be able to reach as many people as possible, without having to build the application twice, I decided to use React Native as part of my software stack, in order to be able to publish the app to both the App Store and Google Play. Furthermore, in order to support the instantaneous synchronization of the users, meetings and calendars, I decided to use Amplify, which is a collection of services, tools and functions provided by Amazon Web Services. The services allow mobile and web developers to build, host and test scalable and secure cloud-powered applications with a usable interface. Some of its features include storage, analytics, predictions, APIs and authentication. AWS Amplify will allow me to not only provide a database for my application, but also utilize its extensive hosting solution. It provides easy integration into

React Native front-end components with its developed 'cloud connected backend' [8].

In addition, I will be hosting an extra backend on Django, which will be accessed by AWS through API calls. Django is one of the most popular python-based backend frameworks. I will utilize Django to build my backend logic, because it allows me to build highly scalable and secure applications in python. Both AWS and Django provide backend services, however they are useful for different purposes. Django is the best in the field for building highly modular and customizable systems. It provides features, such as ORM, URL routing, template rendering and can easily manage dependencies and packages. These are all essential features that I will need to take advantage of, and the AWS Amplify platform does not sufficiently provide them. On the other hand, it was imperative for me to utilize AWS Amplify as well, as it provides cloud services that are essential for the proper functionality of a mobile application, such as storage, authentication components, or data streaming.

4.4.2 Web Demo

In addition to the mobile application, I decided to develop a web-based application, which will be able to demonstrate the usability, functionality and advantages of the batch scheduling algorithm which I developed. In the mobile application, it is challenging to showcase the benefits of the algorithm, as it is not clearly visible why meetings got scheduled to that exact date and time interval. As a result, I developed an application, where users can create many

participants and meeting requests at ease. The user specifies the constraints of the meetings and the preferences of the participants, and the algorithm generates the schedule for all of the input participants in a calendar-like format under each other. As a result, meeting intervals are clearly visible, which allows the user to inspect the reasons behind the scheduled dates and times as well as the meeting collisions if they exist. The demo allows users to quickly test the algorithm in a controlled environment without having to download or install the mobile app. In addition, it also allows me to run tests by providing large quantities of inputs, with many collisions, aiming to break the code. Lastly, it is a great marketing tool that can generate interest and valuable user feedback for my application by providing users with an easily accessible interface to test its functionalities, without the need to visit the App Store.

4.5 Security

In order to identify common safety issues, I researched some recent mobile application security breaches. According to HackerRead, OyeTalk - a famous voice chat application reaching over 5 million downloads - accidentally leaked their Firebase data [1]. Firebase is one of the most popular cloud platforms, with similar features and functionalities to AWS. Even though leaving some hard coded information (API keys) in the front-end was a careless practice from the developers, the owners also failed to take the security breach seriously, resulting in further adverse effects. Learning from OyeTalk's mistake, I will ensure that I do not access any of my APIs (AWS and

Django) through hard coded keys in the code, as well as establishing a procedure to immediately shut down all of my services in case of the event of a security breach.

5 Requirements and Specifications

5.1 Brief

Identifying user stories before starting to develop a major software engineering project is imperative to adequately meet users' needs, understand the project's scope and efficiently manage time.

5.2 Stakeholders

Stakeholders are users who have some sort of interest in my project, and whose life may be affected by it. In order to develop a software system that adequately meets requirements, it is imperative to identify who are the individuals whose needs have to be satisfied. In case of my project, the target audience is employees inside medium to large corporations, as well as the managers and leaders of those employees. In order to precisely identify my requirements, I will constantly keep in mind my target audience.

5.3 Requirements

5.3.1 Functional Requirements

Mobile Application

- The user should be able to view his/her calendar day view on the homepage
- The user should be able to create, login, log-out, edit and delete their account
- The user should be able to create, edit and delete teams
- The user should be able to invite and remove other users from his/her team
- The user should be able to view the details of the teams where he/she is a member of, such as its description, headquarter location, other members and upcoming and past meetings
- The team leaders should be able to create meeting requests using some broad constraints, and invite participants
- The user should be able to select his/her preferred time of the day to attend meetings
- The user should be able to edit his/her non-temporal meeting preferences in the account page (e.g. maximum number of meetings per day, interests
- The user should receive notifications about relevant events (e.g. meeting invites, team invitations)
- The user should be able to enter search requests, which should generate a list of options for the user to navigate to (e.g. teams, meetings, preferences)

- The system should utilize constraint programming to generate meetings based on the hard constraints specified by the host and the preferences of the invited participants

Web Demo

- The user should be able to select between the algorithms, which the system will use to generate the meetings
- The user should be able to input and remove participants into/from the system
- The user should be able to edit the details and preferences of the inputed participants
- The user should be able to input and remove meeting requests into/from the system
- The user should be able to edit the meeting length and the invited participants of the
- The user should be able to generate scheduled meetings based on the input meeting requests
- The user should be able to view the list of generated meetings, where each row represents the calendar of a participant

5.3.2 Non-Functional Requirements

Mobile Application

- The application should be available on both IOS and Android, and it should be device independent

- The application should utilize an authentication system, which automatically hashes users' passwords and provides access based on permissions
- The authentication system should send confirmation emails to registered users

AWS

- The system should provide a database to persistently store users' data
- The system should send API requests to the Django back-end to get scheduled meetings
- At every hour, the system should automatically determine which meetings have to be scheduled using the meeting requests' 'schedule_by' attribute
- The system should provide a defined data schema which can be queried using GraphQL HTTP requests
- The system should provide real-time updates, that alerts React Native when there is a change in a data
- The system should provide a conflict resolution system that prevents concurrency conflict when multiple users access and edit the database at the same time

Django

- The system should be able to accept API requests and send responses

- The system should be able to calculate scheduled meetings based on multiple algorithms

Web Demo

- The system should be able to make API requests to the Django back end, and receive data as a response
- The system should be accessed through a public web URL

5.3.3 Constraint Requirements

- The mobile application should support the minimum version of Android 7.0 and iOS 11 or later.
- The entire system (mobile application + AWS, Django Back-end) should stay stable even with thousands of concurrent users
- The meeting length must be at least 15 minutes and no more than 4 hours
- The system must support a maximum of 3 meeting requests per user per day
- The system must use industry-standard encryption (AES 256) to secure users' data

5.3.4 Performance

5.3.5 Limitations

6 Design

6.0.1 React Native

Mobile application development involves creating programs that are capable of running on mobile devices, such as on smartphones and tablets. There are a variety of platforms to consider when building applications, which all have unique hardware and software requirements. Popular operating systems include IOS, Android and Windows. According to BankMyCell, 86.29% of the population has a smartphone device, which is around 7.33 billion people [3]. As a result, it is imperative to utilize frameworks that allow cross-platform development. 71.45% of the devices run on android, and 27.83% run on IOS, which is more than 99% of all phones [2]. However, 67% of the global app spending was through the App Store, which conveys that IOS users are more willing and able to spend money on apps [10]. With the knowledge of these numbers, I was certain that I will want to develop an application that will run on both Android and IOS. There are several platforms that I evaluated when making the decision on which framework to use, but the two major ones were Flutter and React Native. I decided to use React Native because of several reason. Firstly, I believe that JavaScript is a more efficient and more widely utilized programming language than Dart (made by Google), which is the programming language used by Re-

act Native. Secondly, React Native is a more established software with a larger community than Flutter, therefore it has more online resources available. Finally, React Native performs significantly better at rendering UI components because of the way it renders them as a tree of nodes.

React Native is an open-source mobile application development framework, which was created by Facebook. It allows developers to take advantage of JavaScript and build cross-platform mobile applications. It will render the applications using 'native' elements by converting the JavaScript code into Objective-C (Swift) for IOS and Java for Android. In addition, it takes advantage of React's component based UI management, which encourages efficient code reusability.

6.0.2 Amazon Web Services

AWS services are an integral part of my mobile application. A sucessful app should be backed up with a secure, robust and flexible cloud infrastructure, that allow data to be stored and managed efficiently. Cloud enables apps to function properly, because it provides features that increase performance and improves scalability and reliability. AWS provides apps the crucial resources and infrastructure that enable them to operate efficiently and scale vertically in a flexible manner.

AWS is the leading cloud computing platform on the market, pro-

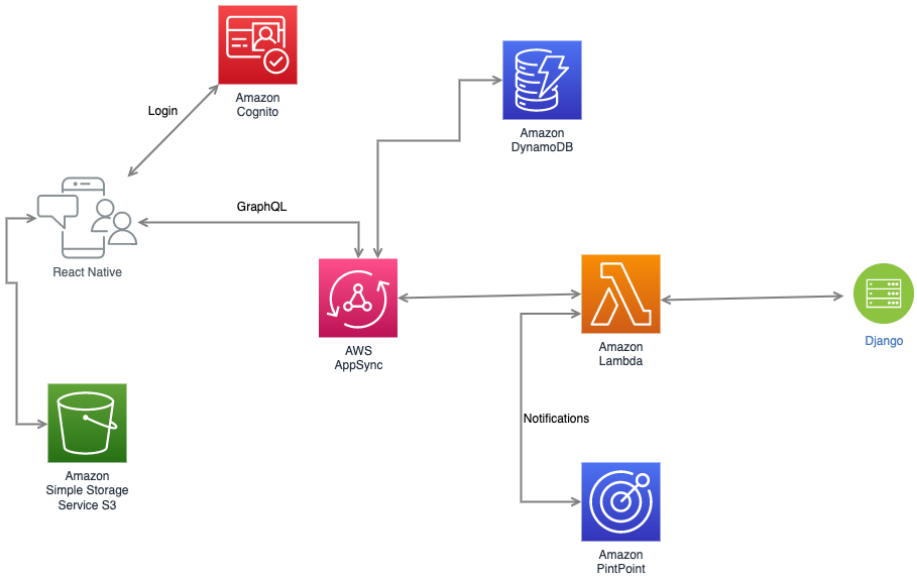

Figure 4: AWS Architecture

viding over 200 services. The majority of the services that AWS provides are microservices, meaning that they run independently from each other. As a result, it is possible to add, remove or alter services without having to restructure the entire architecture. As I mentioned before, AWS provides hundreds of services; therefore, my first challenge was identifying the ones that I will require to develop a successful scheduling application.

As it is clearly visible on Figure 4, I am utilizing 6 services in total, excluding my React Native application and my Django back-end. In order to make sure that the user has a smooth experience, it is imperative to make sure that the individual services are able to properly communicate with each other. Below I list and explain the 6 AWS services, and how they are implemented to provide efficient

cloud functionality to my app.

It is also notable to mention that AWS services are considered 'serverless' functions, meaning that they are distributed across multiple Amazon servers. As a result, there is no dedicated server that the user can manage. This abstracts the entire architecture, and makes it more secure by running services across many different computers, making DDOS attacks inherently impossible.

6.0.3 Web Demo

To demonstrate the functionality of my different scheduling algorithms, I decided to develop a React website, where the user can bulk input a lot of data, and schedule meetings with ease. In addition, I wanted to ensure that the visual representation of the scheduled meetings demonstrates the efficiency of the algorithms through displaying all of the users' calendar in a way that allows the user to easily compare them. The web demo communicates with the Django backend only, because there is no need for it to utilize any of the AWS services, such as database storage, authentication or file storage. This instance is also a perfect example for why disjointing different backend services can be advantageous to make the entire software system more independent, efficient and open for modification. This distribution of the backend services is often called 'microservices', which occurs when the software system is broken down into smaller loosely-coupled services that are independent of each

other. Such an architecture encourages modularity, scalability and flexibility.

7 Implementation

7.1 Scheduling Algorithms

7.1.1 Overview

As I mentioned before, the implementation of my algorithms will be in a Django backend. Therefore, both the frontend, and other backends (AWS Amplify) will be able to call the scheduling algorithms. I included multiple algorithms inside my Django backend, and they can all be called under different API URLs; therefore, their functionality and results can be compared. Another reason besides the ones I already mentioned for my choice of implementing the scheduling system in a separate Django backend, is that I wanted it to be callable from both the AWS Amplify backend based on event listeners - such as when the current time and day reaches the time and day specified for a meeting to be scheduled by - as well as from my Web Demo, which was developed with a React front-end.

Below, I will outline and explain the functionality of each of the different algorithms that I developed for the same purpose of scheduling meeting requests.

7.1.2 Boolean Satisfiability (SAT) Algorithm

In order to construct a system that beats complex algorithms that are currently on the market, I decided to use Google's Or-Tools library. As I briefly mentioned before, Google Or-Tools is an open-source software suite that allows users to efficiently solve optimization problems using multiple different techniques. I will be utilizing its constraint programming features that provide algorithms for finding feasible solutions to problems that the user can define through variables, constraints and an objective function.

CP-SAT

CP-SAT (Constraint Programming-Satisfiability) is one of the solvers that Google provides with its Or-Tools library. It is particularly efficient, because it combines constraint programming and boolean satisfiability techniques to solve combinatorial problems. It takes advantage of SAT techniques to solve boolean constraints and CP techniques to solve all of the other constraints.

Because of the fact that it solves boolean constraints extremely efficiently, I wondered whether there is a way to define my problem as an SAT. To understand the constraints and preferences of each participant, I knew that I had to define the problem in terms of the calendar of each participant. Therefore, I realized that using a three dimensional data strucutre, I can describe the problem as boolean values.

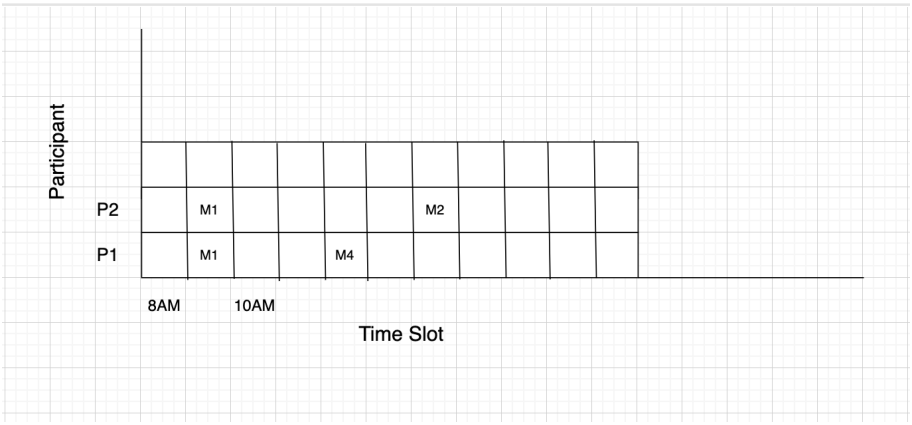

Figure 5: SAT 3D Data Structure

As it is clearly visible on Figure 5, this problem can be defined on fixed time intervals on a 3D data structure. For each participant, for each time slot and for each meeting it is possible to set a true or a false value. Using a for loop, I created all of the variables that the solver requires to calculate the final result, and placed them in a 3D array.

$$availabilities[(p, slot, m)] = model.NewBoolVar(f"availabilities\_\{p\}\_\{slot\}\_\{m\}") \tag{5}$$

As I mentioned previously, by only utilizing boolean variables, OR-Tools only employs its SAT solvers, which is more efficient than having to consider variables of integer types and exploiting constraint programming techniques as well. The SAT solver transforms the given problem instance into a sequence of logical expressions. In order to come up with a solution, the algorithm finds an assign-

ment of 0 or 1 to all of the variables that satisfy all constraints. The technique is similar to CSP; however, taking advantage of the soft constraints, the program is able to disobey some constraints to generate a solution that is feasible and as close to the optimal one as possible.

If we consider participants from index 0, time slots from index 8, and meetings from index 0, we can define meeting #6 happening on participant #2's calendar from 8-9AM as array[3][8][7] = true. Defining the calendar of each participant in this format is extraordinarily efficient, because it is possible to define constraints between participants rather easily. The most important constraint in this algorithm is that participants attending the same meeting, must have that meeting in their calendar at the same time. Therefore, if both participant #2 and participant #3 are attending meeting #6, if array[3][12][7] == true, that must mean that array[4][12][7] = true.

Another constraint is that participants who are invited to a meeting, must attend that meeting exactly once during the defined time interval. Therefore, if participant #3 is invited to meeting #3, array[4][...][4] should be true exactly once.

Similarly to constraints, in a 3D array format, it is also reasonably plausible to define an objective function. One of the main criteria when optimizing the algorithm is to minimize the number of collisions. A collision occurs when one participant gets scheduled

multiple meetings for the same time; for example, array[3][8][7], and array[3][8][9] are also true. Even though it would be possible to state that one participant can attend at most one meeting at the same time, in some over constrained scenarios when this is not feasible, the algorithm might not give any result. As a result, minimizing the number of cases when a participant attends two meetings at the same time slot is required. Another way to achieve the same result is to minimize the total number of empty slots across all the time slots for a given participant, which will encourage the scheduler to place meetings on empty slots, instead of placing them on slots where there are already meetings scheduled for.

$$\begin{aligned} &\text{collisions} = ([\text{availabilities}[(\text{p, slot, m})] \text{ for m in all\_meetings}]) \\ &\quad \text{AddMaxEquality}(\text{participant\_collisions}[(\text{p, slot})], \text{collisions}) \end{aligned} \tag{6}$$

As it is clearly visible on Equation 6, looking at the problem from a global perspective, minimizing collisions can be achieved through encouraging meetings to be scheduled where there is an empty slot. As values can be either 0 or 1, AddMaxEquality will always assign participant_collisions 1 if there is already at least 1 meeting scheduled for a particular slot. Therefore, through counting the number of slots where there is at least one meeting occuring (see Equation 8), it is possible to set up a minimization function, that will encourage meetings to be scheduled on empty slots.

$$\text{sum}([\text{participant\_collisions}[(p, \text{slot})] \text{ for p in all\_participants} \\ \text{for slot in availableTimes}]) \tag{7}$$

Finally, I included minimizing the number of collisions with participants' previous unavailabilities into my objective function. Previous unavilabilities are meetings that are already scheduled for the participants, which can be meetings from the app's database, or individual synchronised meetings and events. The technique for minimizing such collisions is similar to the one described above. Because there are two separate objectives, in order to represent the final objective function, it is imperative to discuss the notion of penalties.

Penalties represent costs for violating individual soft constraints. They are extremely important as they allow the me to assign priorities to the individual objectives that make up the objective function. By multiplying the participant_collisions by a penalty of 5 and multiplying unavailability_collisions by a penalty of 2, I decided to prioritize avoiding collisions between the newly scheduled meetings over collisions with past meetings and events. I made this decision, because in the advanced algorithm, that I will discuss later in the report, I will consider previous meetings scheduled through the app as unavailabilities and I will be able to determine importances by attaching priorities to the meetings. However, in this version, I assume that there is no knowledge over the inputed unavailabilities;

therefore, the meeting requests are considered more important than events in the calendar.

In order to efficiently solve the SAT, Or-Tools takes advantage of techniques such as the lazy clause generation and nogood learning that I mentioned in the introduction of the report. In order to minimize the search space, it is advise to include as many constraints as possible, even if the current constraints describe the problem sufficiently already.

7.1.3 Advanced Algorithm

Even though the previous algorithm, which relies on SAT techniques to solve my optimization problem, is efficient, it fails to account for different time intervals for meetings. As it only works with boolean variables, it can only express whether there is a meeting scheduled for a particular time period; however, it is not capable of specifying the length of that period. One solution to that problem would have been to generate a 3D array, where each time slot represents a 1-minute or a 15-minute block; and, specifying the length of the meeting could have been achieved through scheduling a particular meeting over multiple consecutive time slots. However, considering calendars over multiple days would have required the generation of insanely large arrays, which would have been extremely challanging to produce.

As a result, I decided to write a new algorithm from scratch, which mainly utilizes constraint programming techniques, where the meet-

ings' intervals are defined as integer values (start_time - end_time). In addition, unlike the previous algorithm, this one is also capable of handling the scheduling of meetings over multiple days, which is an imperative feature that I would like to describe now.

Because of the fact that constraint programming works with integer values, it is more advantageous to define the calendar space where meetings can be scheduled in minutes, which is an integer number. However, meeting requests that users created are defined in date intervals, such as 'I would like the meeting to take place between May 5th and May 10th. Therefore, in order to define the domain of the illustrated meeting request, time 0 will represent the start of the work day on May 5th (which is defined by the team leader, but is 8AM by default). The upper end of the time domain is the last working hour on the 10th of May, which is 8PM by default. Therefore, each point in time in this 6-day interval example can be represented as a minute integer. For example, a meeting taking place on the 5th of May from 14:00 - 14:30 is represented as (360 - 390).

However, an additional difficulty in this problem is retrieving which day a particular time index is on, and the time in the day. We can utilize the division and modulo operations to calculate these values. For this calculation, we need to know the length of the day, which is 720 by default (8AM-8PM).

$$\begin{aligned} \text{x } \% \text{ 720} &= \text{current day} \\ \text{x mod 720} &= \text{current time in the day} \end{aligned} \tag{8}$$

Defining each meeting's domain in this format is efficient, because all of the requests can be considered together efficiently. Even though different meetings will have different domains, by considering a common predefined interval, each meeting's domain can be represented as integer values. For example, lets say that the algorithm is asked to schedule 2 meetings. Meeting Request #1 has to be scheduled between May 4th and May 6th, and Meeting Request #2 between May 3rd and May 7th. The total interval to consider is from May 3rd - May 7th, and the domain of #1 can be defined as (720 - 2160) and #2 as (0 - 3600). This format allows efficient comparison of the meetings, which will be particularly useful when identifying collisions.

The most important constraint that must be defined when defining the scheduler in this format is on the start and the end times. The difference between the end time and the start time of a meeting has to equal the specified duration of that meeting. In addition, the minimum start time of each meeting must be greater than or equal to the start time of its domain and smaller than or equal to the end time of its domain.

After specifying the basic hard constraints, most of the additional

constraints will be soft ones. I decided to identify three main objectives to minimize in descending order in terms of priorities: newly created meeting overlaps, unavailability overlaps and participant preference overlaps. These are all relaxation techniques that I mentioned in the introduction section of the report. When calculating overlaps, relaxation is utilized to allow the solver to accept some collisions in order to avoid not receiving any feasible results at all.

Meeting Overlaps

I decided to minimize the total length of overlaps, instead of the number of overlaps. To identify overlaps, I iterated through all of the meeting requests, and compared each meeting request to every other one-by-one for each participant. I created an integer variable, overlap, to represent the overlap duration in minutes. A boolean variable m1_before_m2 represents whether the first meeting occurs before the other meeting. 'overlap' is a value that is greater than or equal to the difference between m1's end time and m2's start time if m1_before_m2 is true. If m1_before_m2 is false, 'Overlap' is a value greater than or equal to the difference between m2's end time and m1's start time.

if m1_before_m2

overlap $\geq$ (meeting_start_vars[m1] + m1_duration - meeting_start_vars[m2])

else

overlap $\geq$ (meeting_start_vars[m2] + m2_duration - meeting_start_vars[m1])

(9)

Looking at Equation 9, it is clearly visible that the overlap duration between two meetings for a particular participant can be represented as an inequality formula. The total overlap across all participants for each and every pair of meeting is the sum of all overlaps represented on Equation 9. One additional factor in calculating the total overlaps is participants' individual importances. The overlaps of participants with higher importance should be taken more seriously. Therefore, I created a variable 'weighted_overlap which is the product of 'overlap' and the participant's importance. As a result, an overlap of 2 minutes for a participant with an importance score of 2 will be considered as unsatisfactory as an overlap of 4 minutes for a participant with an importance score of 1.

Unavailability Overlaps

To minimize unavailability collisions, I decided to include a slack variable. As I mentioned in section 3.6.2, through the use of slack variables, developers are able to convert inequality constraints into equality. With the slack variable, I defined a maximum collision (in minutes) that can occur, in order to convert the hard constraint of collision to a soft one. Even though I allow the program to violate the collision constraint, I ensured that it is avoided as much as possible by including the sum of all slack variables into my minimization function.

Furthermore, the intervals for unavailable slots are specified for given days; therefore, the equation below is used to calculate the exact interval in the required integer format, where total_minutes represents the length in minutes of a work day.

$$\text{start\_day\_minutes} = (\text{day} - 1) * \text{total\_minutes} \tag{10}$$

Participant Preference Overlaps

Finally, participants have the ability to specify their preferences for which time of the day they would prefer to have their meetings. The three options are: 'morning', 'afternoon', 'evening', which are all intervals relative to the specified length of a working day. Even though it will almost be impossible to fit all of the meetings of a participant in the 3-4 hour preferred window, the algorithm aims to fit as many meetings as it possibly can. Preferences are specified in terms of hours in ISO 8601 format; therefore, the equations below calculate the preferences in terms of integer values.

$$\begin{aligned}\text{start\_pref\_minutes} &= (\text{start\_pref.hour} - \text{start\_time.hour}) * 60 + \text{start\_pref.minute} \\ \text{end\_pref\_minutes} &= (\text{end\_pref.hour} - \text{start\_time.hour}) * 60 + \text{end\_pref.minute}\end{aligned} \tag{11}$$

Preference collisions are calculated in the usual way; however, the penalty for not scheduling a meeting to which a particular participant is invited to inside that participant's preferred time zone, is set to the total duration of that meeting.

The final objective function is defined as the sum of the total_overlap, total_unavailability_penalty and the total_preference_penalty. I assigned the objective function to the solver as a minimization task;

therefore, it will aim to minimize the total sum, taking into account the weight of the individual components.

Lastly, specifying the format of the data to be returned by the function was imperative to make sure that the API response matches the required format. I created a print_best_solution function inside my printer class (which I used to output the results for debugging purposes), that returns the resulting scheduled meetings in the proper format - in terms of a list of participant calendars.

7.1.4 Greedy Algorithm

To demonstrate the efficiency of batch scheduling, I developed an algorithm that solves the same problem using a greedy approach. The greedy algorithm that I developed is the iterative scheduling approach for my SAT algorithm, which I described in chapter 7.1.2. However, in order to be able to compare and demonstrate the pure strength of the batch scheduling algorithm, I decided not to include the participants' previous unavailabilies. Therefore, using my web demo, it is possible to compare the two different scheduling algorithms (SAT and greedy) with the same input data and inspect the number of collisions for each output.

The scheduleGreedyMeetings function determines the best possible time slot to schedule each of the input meetings in an iterative way. However, the order in which the meetings are received greatly contributes to the outcome, as there is no back propagation in the algorithm. As a result, it is only capable of determining the most

efficient slot for the current meeting that is to be scheduled, but does not have to ability to rearrange previous meetings in order to maximize total utility across all meetings and participants. To calculate the total overlap at each time slot, the following method is used:

```
overlap = sum([1 for s, m, p in scheduled_meetings if
p.intersection(meeting_participants) and s == slot])
```

A list comprehension is used to create a list of 1s for each meeting that would overlap with the meeting if it was scheduled at 'slot'. With this knowledge, it is possible to determine which time slot would yield the least amount of collisions.

The greedy algorithm demonstrates the efficiency of the batch scheduling system even with the only consideration of minimizing the the total number of collisions. Considering the numerous other objectives that I included in my more advanced algorithms, it is apparent that it would generate results that absolutely don't benefit as many stakeholders as the potential. In order to put into perspective the difference in the efficiencies between the algorithms, I will include quantitative comparisons between the results of each algorithm with identical input in the Evaluation section of my report (Section 8).

7.1.5 Comparison of Algorithms

Even though it would have been adequate to only develop the interval batch scheduling algorithm for my mobile application, I was

motivated to construct multiple algorithms to explore further options and compare efficiencies. I will demonstrate the differences in the performances of the algorithms in the Evaluation section of my report (Chapter 8); however, I believe that it is imperative to mention that the batch scheduling algorithm is significantly more efficient in avoiding collisions. Through the use of back-propagation and other optimization features, it is able to identify optimal meeting times that result in the most efficient outcome considering the total utility of all of the stakeholders.

7.1.6 REST API

To allow my mobile application and the web demo to access the meeting scheduling algorithms, I constructed a REST API. Representational State Transfer is an architecture that allows developers to build web service back-ends. It uses HTTP requests to establish communication between the client and the server. Through calling the API, the client is able to send data to the back-end and receive a response. When the server receives the data, it is able to perform operations on it and send an HTTP request as a response. It is imperative to mention that REST is stateless; therefore, it does not maintain any previous information from the client [7]. Each and every information that is sent from the client is sufficient to establish the communication, thus providing a safe communication channel and flexibility [7].

7.2 React Native

7.2.1 Overview

7.2.2 Homepage

Keeping up with the trend of designing and implementing minimalist applications, I decided to place a simple calendar on the homepage of my app, which displays the user's meetings and events taking place on the current day. It is a vertical view, and contains a horizontal red line which moves vertically displaying the current time. In order to make the app dynamic and refresh its content without having to reload the screens or the entire app, I took advantage of hooks, which were introduced in version 16.8 and allow developers to re-render individual components when specific variable values change.

```
const [currentTime, setCurrentTime] = useState(new Date());
```
(12)

The code above (Equation 12) demonstrates the implementation of a useState hook, which defines a value and a function that updates that value. Whenever there is a change in the value of currentTime, any component that displays or uses that value will re-render. As a result, my time indicator constantly updates its position and displayed text whenever the current time (hour:minutes) changes.

7.2.3 Teams

Teams are an imperative part of my application. The batch scheduling of meetings is only possible between participants of the same team. Every user is able to create a new team, and invite other users to that team. Under the teams menu section, the user is able to view a list of every meeting that he/she has a membership to. Under each team, the number of team members and the number of new meeting request notifications is visible. To create a team, the user only has to specify the name of the team, the headerquarter and its description. A user can create a maximum of 5 teams, in order to avoid spamming the database with unused teams.

GraphQL

One feature that the team accumulator page utilizes is GraphQL, which is a data query language for APIs. It provides a more efficient way for the client and the server to communicate than the traditional REST API. GraphQL has the advantage of returning only the data from the database that the user requires in a single request, which reduces the amount of data that has to flow over the network, thus increasing security and reducing latency. When the user creates a team inside my mobile app, a new team object is created with the input details. React Native sends a request to the AWS DynamoDB database through a GraphQL request, and receives a response. One special feature of GraphQL is called subscription, which allows the client (React Native) to listen and receive real-time updates from the server when a defined data changes. In

order to generate such subscriptions, there is an established connection between the database and the client through GraphQL using WebSockets. WebSocket API is an innnovative technology that allows two-way communication between the client and the server over a singular TCP connection. As a result, every time a user creates a team, or joins a team, his/her team list page updates in real-time.

Employee management can be accessed inside the 'Employees' section of the team page. Every team member is able to view the list of members; however, team leaders have the option to edit, remove and invite new members. To invite a member, the user can view a list of all of the users of the application. In addition, the user can filter the results, through a search bar at the top of the screen. Because of the fact that users' name details are stored in the database as separate attributes, the following code (Equation 13) allows me to search for users by their first name and last name as well. It also makes sure that uppercase and lowercase letters are omitted to allow for a smoother search.

$$\begin{aligned} &\text{participants.filter((participant)} => \\ &\text{(participant.first\_name} + \text{ " " } + \text{participant.last\_name).toLowerCase()} \\ &\text{.includes(search.toLowerCase())} \end{aligned} \tag{13}$$

In additional to employee management, team leaders have the ability to edit and delete teams by clicking on the settings icon in the

top right of the team page. When the user deletes a team, it automatically places his/her screen to the team accumulator page, and removes the deleted team from the list. When a team is deleted, all of the memberships that are connected to that team are deleted from the database.

AWS Versioning And Deletion Protection

Even though I will describe the functionality of the AWS services I am utilizing, I wanted to quickly mention versioning and deletion protection because it is an imperative feature to consider when fetching and updating data to/from the database from React Native. Versioning is an AWS feature that allows users to identify and track changes on resources. When versioning is enabled, AWS automatically creates a new version each time the resource is updated, rather than overwriting item with the previous version. This allows users to keep a record of all changes made to the resource and revert to a previous version if necessary. This is extremely important, because it prevents multiple users to make changes to the same database item. Versioning allows database transactions to be isolated, meaning that the data stays consistent when multiple users try to access and modify it at the same time. Isolation is an ACID principle, which is the basis for a secure and reliable database transaction. When one user updates a data item, he/she has to specify the version which he/she wants to update. Therefore, if another user retrieves the same item while the first user works on it, when the second user wants to send the changes to the database for the retrieved version, it will not update it because its version already

changed. As a result, concurrent database transactions are executed serially, meaning that their outcome becomes independent of each other. In order to achieve versioning, a _version attribute is specified for each database item, which keeps track of its current version.

In addition to versioning, deletion protection is another crucial feature that I decided to utilize to achieve ACID database transactions. Deletion protection prevents the accidential deletion of items from the database by adding an additional boolean attribute (_deleted), which keeps deleted items in a backup TTL (time-to-live) database. AWS provides a service for permanently deleting softly removed items after 30 days. Deletion protection ensures integrity and availability, which are essential features of secure databases. In order to only fetch items from the database that haven't been deleted by the user, I needed to filter table queries, and exclude every item that has been deleted. Even though it is clearly visible that deletion protection increases the amount of data that has to be queried, I still decided to take advantage of it in order to seccurely maintain the storage of my users' data.

7.2.4 Meetings

Scheduling meetings is the main functionality and the reason for being of my application. I wanted to implement meeting creation in a way that explicitly takes the pressure - related to providing data that has an effect on his/her colleagues - off the team leader's shoulders. As a result, the user is only asked to provide some basic data, such as the title of the meeting, description, earliest and latest

date for the meeting to take place, earliest and latest start and end times which define the day interval I described in section 7.1.3. In addition, the user is asked to select the days when the weekend can be scheduled on (Monday - Friday by default). Finally, the user has the option to enable or disable bulk scheduling. When this option is not selected, the meeting will be scheduled instantly, and the meeting invite will be sent to the invited team members. However, when this option is selected, the user has the option to specify by when the meeting should be scheduled, which will send the invites to the invited members on that day. An important concept when entering the 'invites by' date is that it must be at least 24 hours before the earliest date, so that the invited participants have enough time to learn about the whereabouts of the meeting. I decided to allow the user to choose whether the meeting should utilize the bulk scheduling option, is to allow meetings to be scheduled for whenever the user wants (for example the next day), and there is not enough time to take advantage of the batch scheduling feature.

When all of the basic details are inputed, the user is redirected to the next screen, which requires the invitation of at least 1 participant from the team which the scheduling of the meeting request was initiated from. Invited participants can be selected by clicking on their profile once. By clicking on their profile one more time, the user is able to set their presence as optional, which will weight their absence or collision less than for mandatory participants. The 'toggleSelection' method defines the logic behind which participants are selected for mandatory attendance and which are for optional.

Is uses two useState arrays to store the two types of invitees. When a user icon is pressed, it first checks if the person is already present in the mandatory array. If present, the participant is removed and placed into the optional array.

```
setSelectedPeople(selectedPeople.filter((p) =>
p !== person))
setSelectedOptionalPeople([...selectedOptionalPeople,
person])
```

After, it checks the optional array and does a similar version of the above operations. Finally, when the code arrives at the base case - the participant is not selected at all - it adds that person to the mandatory array.

Once the 'Create Meeting Request' button is pressed, the meeting request is sent to the database, and it stays there - untouched - until it is ready to be scheduled which will depend on its 'invites_by' date attribute.

7.2.5 Profile

The profile page displays just two basic features to the users. First, the user can specify his/her preferred time of the day to attend meetings. Selecting one of these options, and pressing the 'Save Settings' button will update the user's preference in the database, and will be used by the meeting scheduler. In addition, pressing the button also displays an alert to the user, which ensures him/her

that the changes have been recorded.

```
Alert.alert('Settings Saved', 'Updated to ${selectedTime}.');
```

In addition, The user has the ability to sign out from the account using the 'Sign Out' button. This action will take the user to the sign in/sign up page.

7.2.6 Navigation

Routing and navigation is implemented using React Navigation, which is a collection of libraries that allow developers to manage and navigate between components and screens inside React Native. The library provides an extensive array of navigation patterns, such as tab and stack navigations which allow for the development of flexible and customizable systems.
I decided to utilize a stack navigator, which adds newly opened screens on top of a global stack. This provides an easy way of going back to previous screens, which feels natural to most users. To define the stack, I created a 'Navigator' class, and added it as a global variable:

```
const Stack = createNativeStackNavigator();
```

Then, I defined each of the screens that can be added to the stack, such as:

```
<Stack.Screen name='Meeting' component={MeetingScreen}/>
```

I added all of my 10 screens to the Stack Navigator. To navigate between the screens, I passed on the navigation variable across all screens, which included the information about the global stack. To instantiate a new screen instance of the previously described MeetingScreen, I called the following code, which also adds the screen to the top of the stack:

```
navigation.navigate('CreateNewTeam')
```

To pop the top (current) screen from the navigation stack and go back to the previous screen, I utilized the following function:

```
navigation.goBack();
```

Another imperative element of the navigator, is the bottom navigation menu. I decided to utilize the 'BottomTabNavigator' library from the same library collection, and defined the global tab as:

```
const Tab = createBottomTabNavigator();
```

To make the tab navigator visible on all of the screens, I decided to include it on the bottom of the screen in the App.js file, which is the equivalent of the 'main' function.

7.2.7 Data Context

Context API is a library that allows data in a React Native application to be shared across all components, without the need to pass it as part of the props through multiple levels in the component tree. A

data context is a global object defined inside a JavaScript file, which can be imported from any component. It also has an update function, which can also be called from any component. Even though context is a great tool to utilize global variables across components easily, they are rather challenging to maintain and debug and can decrease performance. As a result, I decided to use the Context library for values that I would otherwise had to pass on between components a numerous number of times (current user, currently scheduled meeting request object).

7.2.8 Authentication

Even before I started developing the application, I decided that the security of my users' data will be the most important feature that I must implement. As a result, I decided to implement an authentication system that takes advantages of the most advanced cryptography and systems security techniques. AWS Cognito provides a fully managed authentication service, which also integrates perfectly with other AWS services that I am using. In addition to authentication, AWS Cognito allows me to acces features, such as user profile data storage and multi-factor authentication (MFA). To install and set up AWS Cognito with my React Native application, I was required to utilize the AWS Amplify CLI, which allows the management of the resources of AWS services.

One difficulty with setting up the account system through AWS Cognito, is that it doesn't automatically sync the data with the database (dynamoDB). As a result, every time a user logs into my applica-

tion, I have to check whether the user exists in the database, and update its profile based his/her authentication profile. To achieve this, I can request the user's authentication data from AWS Cognito through:

```
const authUser = await Auth.currentAuthenticatedUser({bypassCache: true});
```

Next, I check whether the user is already present in the database:

```
const userData = await API.graphql(graphqlOperation(getUser,
{ id: authUser.attributes.sub }));
```

Finally, if the user is not in the database, I can create a user entry in the DB with the user's authenticated data. To comply with user data protection laws, such as GDPR, I decided to provide full transparency and only collect basic information from users that are necessary to determine a user's identity (email, password).

7.3 Web Demo

On the initial screen of my application, the user has the ability to select from all of the available scheduling algorithms. As a result of retrieving the results of the scheduling algorithm through API, I achieved modularity for my entire system. Therefore, every time I want to add another algorithm, I am able to specify a new API URL for that algorithm, and extend any frontend code to call that URL. Therefore, even in the web demo, I am able to add as many different views as I want to demonstrate the functionalities of different algorithms.

7.3.1 Setup Screen

Once the preferred algorithm is selected, the user gets navigated to the setup screen, which requires the input of information for the disred meeting requests. It is possible to enter a team name for aesthetics purposes, which also demonstrates that meeting requests are batch scheduled inside individual teams, and their capabilities do not extend beyond the team.

Next, the user is able to input as many participants(team members) as it is preferred. I wanted to make sure that my front-end view does not limit the user with the input in any way, therefore I developed the system so that it is able to grow vertically infinitely. Once the name of a new participant is entered, it creates a new person item in the list, and automatically opens the editor for that participant, which can also be achieved through the green edit button. It is imperative to manage the added, edited and removed participants in real-time, because the user has to option to navigate between the screens of the apps easily, therefore, the data must stay consistent all the time. The delete button removes the user useState list variable using the following code:

```
setItems(items.filter((_, i) => i !== index))
```

As it is visible, the code takes advantage of the update variable of the state hook, and updates the list by removing the desired item at the selected index. Another important React hook that my demo utilizes is the useEffect hook, which allows components to listen to lifecycle events, such as mounting and unmounting. Using

'useEffect', I was able to run code every time a variable changed state, or the screen was rendered. As an example, every time a particular participant is selected to be edited (which automatically happens when a new one is created), the code inside a React hook in the 'ParticipantInfo' class is called, which is responsible for editing the information of the participant. By using 'useEffect', I was able to automatically display the edit component, which allows me to update the screen without having to render a new one and passing on every information that the user already inputed into a new screen.

Depending on the selected scheduling algorithm, the user is able to edit all of the information that the particular algorithm requires about invited participants. Once the user is happy with all of the edited information, the 'Save' button saves the information of the participant into a global DataContext variable, which can be accessed from every screen. Furthermore, every time the edit component is opened for a particular participant, it must update its content with the information that the participant already inputed previously, such as the list of unavailabilities. To achieve this, this information has to be retrieved from the global Context variable of the user.

```
alreadySelectedSlots = (addedParticipants.filter((i) =>
i.name==selectedParticipant [0].unavailableSlots
```

Then the information has to be passed to the hook states of the edit component, which will automatically display the current participant information.

```
const addedSlots = []
alreadySelectedSlots.forEach(function (item, index) {
    setTimeSlots([...addedSlots, item]);
    addedSlots.push(item)
});
```

The 'addedSlots' variable keeps track of variables that are already added in with the foreach loop. This is required because in React, state updates are asynchronous, meaning that the new state may not be immediately available. Thus relying on the old state of the 'timeSlots' variable would lead to unreliable results.

In addition to participants, the user also has the option to input as many meeting requests as he/she finds suitable. Similarly to editing participants, it is possible modify the data of each meeting. Again, this also depends on the selected scheduler, but the main editable attribute is the list of invited participants to the meeting. It is possible to choose any participant from the list that the user already entered as a participant. The user can select and unselect users to be invited to the particular meeting by clicking on the participant's name.

7.3.2 Results Screen

The results screen is one of the paramount feature of my entire project. This screen allows the user to get an insight into the functionality of the optimization scheduling systems, but getting a visual representation of its output based on custom inputs. The user is

able to inspect the individual calendar of every inputed user. Every meeting that the scheduler returns is displayed on the screen. The same meetings share the same colour; therefore, it is easy to see that the same meetings get scheduled for the invited participants for the same time intervals. In addition, the participants' previous meetings and events that were inputed in the 'unavailabilities' section is also visible on the calendars. The user has the ability to go back and forth between the setup screen and the results screen. As a result, it is possible to add or edit the input attributes and visually observe the change in the output values. One notable observation is the fact when a new meeting is inputed, the entire calendar for every individual gets rescheduled. This demonstrates the advantage of batch scheduling. Instead of keeping the old calendar the same, and adding the new meeting to the most most efficient local optimum, it has the ability to rearrange all of the meetings, and generate an entirely new calendar that maximizes the total utility of all the stakeholders.

The number of slots in a given day is generated dynamically; therefore, the code is extendable as much as required, and the day start and end times can easily be changed. In addition, it is also possible to specify the number of day intervals the meetings can be scheduled for. As I mentioned before, the number of consecutive days the meetings can be scheduled for depend on the inputed earliest and latest date in the mobile application. This feature is not included in the demo as it is not essential for the demonstration of the functionality of the scheduling system; however, it is an imperative feature

of the day-to-day functionality of the app.

When the 'Generate Schedules' button is pressed, all of the input data is validated, and the scheduler system in the Django backend is called through an API request:

```
const dataToPassApi = {participants: addedParticipants,
    meetings: addedMeetings}
useEffect(() => {
    axios
    .put("http://localhost:8000/scheduler/basic_meetingrequest/",
        dataToPassApi)
    .then((res) => setParticipants(res.data.data.result),
        setMeetings(res.data.data.allMeetings))
    .catch((err) => console.log(err));
}, [])
```

As is it is clearly visible, depending on the selected scheduling algorithm, the API with the correlating URL is called. The different scheduling algorithms are available on the different scheduler subdomains. Thus, the demo application can be extended to be able to demonstrate the functionality of as many algorithms as it is required.

The different results screen require the presence of different components. For the SAT results, the meetings are fixed in lengthm, therefore their rendering functionalities are quiet simple. However, it was rather challenging to implement the rendering of the interval

schedules. One difficulty was calculating the start and end times of the meeting blocks. To calculate the positions for the system that displays the calendar from 8AM to 8PM, the following code is utilized:

```
const start = Math.max(meeting.start - offset, hour);
const end = Math.min(meeting.end - offset, hour + 1);
const left = start - hour;
const width = end - start;
```

An additional difficulty was representing overlapping meetings. When meetings overlap, their height has to be adjusted, so that both meetings are visible under each other. To achieve this I was required to check whether any meetings are overlapping.

```
const allOverlappingMeetings = meetings.filter(
    (m) => m.start < meeting.end && m.end > meeting.start
);
```

To set the style of the meeting block based on the 'allOverlapping-Meetings' information, I was able to adjust the React style using:

```
const overlappingIndex = allOverlappingMeetings.findIndex(
    (m) => m.id === meeting.id
);
const top = overlappingIndex * (100 / allOverlappingMeetings.length);
style={{
top: `${top}%`,
height: `${100 / allOverlappingMeetings.length}%`,
}}
```

Next, I was also required to check whether there is any overlap with an unavailability, which is also a meeting component, but is generated from the input data rather as a scheduled meeting response from the API.

```
const isOverlap =
participantUnavailableBlocks &&
participantUnavailableBlocks.some(
(block) =>
    block.day === selectedDay &&
    timeToDecimal(block.startTime) < meeting.endTime &&
    timeToDecimal(block.endTime) > meeting.startTime
);
```

The code above displays an algorithm that checks whether two time intervals are overlapping. If there is an overlap between the meeting object to be displayed and any unavailability, the value of isOverlap becomes True.

Lastly, I was required to come up with an algorithm that is able to convert the time of the scheduled meetings from a string ('hh:mm') format to a decimal format. To achieve that, I used the following algorithm:

```
const timeToDecimal = (time) => {
    const [hour, minute] = time.split(':');
    return parseInt(hour, 10) + parseInt(minute, 10) / 60;
};
```

As it is clearly visible, the praseInt JavaScript method converts the string variables into an integer using a radix (base) value, which indicates the preferred type of numeral system to be used for the conversion.

7.4 AWS Services

7.4.1 Overview

As I mentioned in Section x, AWS services provide the basis of the cloud infrastructure for my mobile application. In order to serve the entireity of the functional, security and visual reuqirements of my users, I was required to utilize a list of inter-connected services (Figure 4).

AWS AppSync

AWS AppSync provides developers the ability to build scalable APIs for mobile applications. It takes advantage of GraphQL, and allows React Native to connect to (fetch, modify and create data) other AWS services, such as DynamoDB and Lambda. This is the most essential feature as the communication between React Native and AWS is established through AppSync.

Amazon DynamoDB

DynamoDB is a NoSQL database service provided by AWS. It is considered one of the most efficient databases on the market, as it provides services such as automatic replication, failover capabilities, conflict detection and versioning. Conflict detection - as I briefly

mentioned in section 7 - is a feature that protects user data by ensuring that no data is accidently deleted or modified in an undesired way. AppSync increments the versions of items when they are modified, and it also assigns the _deleted attributes to removed ones. This also demonstrates the inter-connectivity of the different AWS services, and their efficiency in providing an efficient way of storing and managing users' data.

Amazon Cognito

Amazon Cognito is a service that provides user management, authentication and authorization to my mobile application. As I briefly mentioned, when describing this service in section 7.2.8, Cognito provides Two-factor authentication and hashing functions to securely store and manage my users' data. It is able to integrate with other AWS services; therefore, I am able to grant users access to other services through permissions. I utilized this feature when deciding on which user has the ability to create teams, meetings and memberships. Below, I outline and describe the various services I configured and implemented.

Amazon Simple Storage (S3)

Simple Storage is a fully managed storage service that allows objects to be reserved in a scalable manner. It provides functionalities that allow the objects to be retreived and stored in an efficient and secure way. Data that I decide to store in S3 are avatars and profile icons

of teams and users respectively. S3 provides automatic replication, which ensures that users' data is securely stored. However, I wanted to make sure that I do not store any data or image that the user does not want us to keep. Therefore, I decided to implement a feature that permanently deletes files that the user specifically decides to delete, such as when his/her account is deleted. Thus, files are only kept when there is a chance that the file was not purposefully deleted, such as when the user changes his/her profile photo.

Amazon Lambda

Lambda is a computing service that allows for the execution of code on cloud servers without the need for managing dedicated servers. Even though they have some advantages, I am only using them to communicate with DynamoDB and call the Django API. Because AWS services are inter-connected, I am able to retrieve and update data to and from DynamoDB, which increases efficiency. In addition, it is also possible to execute Lambda code when a particular data in the DynamoDB changes. These event listeners allow me to run Lambda functions at specific times (when the meetings are ready to be scheduled), thus effectively managing the life-cycle of the scheduling of meetings.

Amazon PinPoint

Amazon Pinpoint is a customer engagement service that allows applications to send targeted messages to users (such as emails, SMS

and push notifications). I decided to utilize their push notification services to alert customers when they receive a meeting or team invite. PinPoint is an excellent tool for identifying data that are relevant for specific customers, and also for sharing that data with the front-end, which is capable of displaying it to the user.

7.4.2 Configuration

After setting up the individual AWS service components, I utilized an additional AWS feature - Amplify - that allowed me to access and manage the individual services through a Command Line Interface (CLI). CLI is a text-based interface that allows developers to establish communication with a software system. Using Amplify's CLI, I was able to install, test, customize and manage the configuration of the AWS services. Furthermore, I was also able to generate and customize code that efficiently integrates these services with my React Native application.

7.5 Implementation Issues

7.5.1 SAT Algorithm

Even though minimizing the number of empty slots for a given participant will encourage meetings to be scheduled to empty slots over slots where there is already a meeting, it will not encourage collisions to be minimized when there are already 2 or more meetings scheduled for the same; therefore, it will not aim to schedule only 2 meetings at a given time slot over 3 or more meetings.

7.6 Testing

Testing is an essential step in the software development lifecycle, in order to make sure that the application not only compiles and works, but also functions in the way the developer intended it to work. In the following subsections, I will outline and explain how I utilized a range of testing techniques to make sure that the different sub-systems of my application work as intended.

7.6.1 Scheduling Algorithm

In order to efficiently test my python code, for every algorithm I developed, I wrote an extensive print function. Optimization functions have the capability of returning a a lot of information about the results of the algorithm, which has to be filtered, selected and managed. Once Google Or-Tools' solver finishes with the calculations, it can return the objective value, which is the final result of the total minimization function, as well as the individual values of all of the defined model variables. In my basic SAT solver, I iterate through the 3D array, and append every meeting that takes place in each time slot for every participant in a 2D array as a string.

```
print(f"Participant {p}, Time slot {slot},
Meeting: {meetingHappeningAggregator}")
```

Printing the solution in the interval solver was a more challenging task, because it generates many different solutions, and it is required to manually check for the best solution. I achieved this through creating a print_best_solution function, which prints the most optimal

solution that is identified through checking whether its result for the objective function is smaller than the previous results'.

```
if current_overlap < self.best_overlap:
self.best_overlap = current_overlap
self.best_solution = {meeting: self.Value(start_var)
for meeting, start_var in self._meeting_start_vars.items()}
```

In addition to print statements, I utilized PyCharm Debugger, which provides a user interface to easily debug errors by allowing the developer to visually inspect the code by setting breakpoints and stepping through it while also looking at the values of all of the defined variables.

7.6.2 React Native

Developing the mobile application in JavaScript was extremely challenging, because of the sheer amount of code I had. In total I had 16 screens and more than 10 reusable components. Even though I spent a lot of time and effort to maintain high modularity, it was required to include dependencies between the screens and components, which ultimately led to many errors and bugs throughout my development process. To debug my code, I utilized console.log() numerous times, which is similar to print statements in other programming languages. In addition, I took advantage of React Native Debugger, which allowed me to constantly inspect logs and errors through a web interface, which is similar to the debugging process of web development. In addition, I utilized Expo's simulator, which

allowed me to inspect a live demo of my application on my phone in real-time, without having to deploy the software to the App Store.

7.6.3 Web Demo

The debugging process of the Web Demo was similar to the development of the Mobile Application, as it is also based on React and JavaScript.

7.7 Verification

In order to verify my application before completing in-depth functional testings, I decided to carry out code walkthroughs, which is the process of running and inspecting my code line-by-line. I completed this procedure several times for my scheduling algorithms, so make sure that every constraint and objective performs adaquately. In addition, I carried out integration testings, by verifying that the communication between the clients and servers through the APIs properly function by closely examining response logs.

7.8 Future Maintenance

Throughout the entire development of my project, I ensured that each module is available for future maintenance. In order to ensure that the code can be modified by other developers as well in the future, it is imperative to follow the ACID principles. Primarily, I made sure that every function I created was open for extension without the need to modify the code. In addition, I designed the entire architecture of the code - including the React Native app,

Django, and AWS - in a way where the individual components can be edited, replaced or entirely removed without it having a major effect on other components. In addition, I made sure to continously utilize concise and informative comments to make sure that me or other future developers understand what individual functions are doing. I made sure that each function has exactly one purpose and also that it fulfills that purpose entirely. Lastly I consistently followed coding standards to make sure that my code is becomes familiar to other developers as well. In order to achieve this, I utilized comprehensible and descriptive naming conventions and handled exceptions properly, by providing descriptive explanations for caught errors.

8 Evaluation

8.0.1 Experimentation

In order to demonstrate the efficiency of the batch scheduling algorithm, I decided to run both the greedy and my SAT algorithms with the same input data, and calculate the difference in the number of collisions. A collision is when a particular participant has to attend multiple meetings at the same time. In order to adequately observe the differences, I only considered collisions, which provided me with an integer value that is easy to compare. I ran the tests with many different input meetings and participants. The results of my testing are shown below.

As it is clearly visible on Figure 6, the batch scheduling algorithm

# of Meetings	# of Participants	# Collisions (Greedy)	# Collisions (Batch)
2	2	0	0
6	2	0	0
8	4	2	1
10	4	4	3
12	6	6	5
16	6	9	5
20	6	19	15

Figure 6: Comparison of 2 Scheduling Algorithms

consistently performed better on both small and large input sizes. By maximizing the global maximum, instead of considering local objectives iteratively, the scheduling of meetings is significantly more efficient.

8.0.2 Limitations

Scheduling Algorithms

Even though I ended up only utilizing one of my three algorithms, I believe that it was a smart idea to implement all three in order to demonstrate why the one I developed is as efficient as it is. Even though my advanced algorithm (described in Section 7.1.3), which I am utilizing for my mobile application has significant strengths in terms of efficiency and the amount of data it can handle, it still has many limitations.

Firstly, the algorithm is not capable of identifying the importance's of the different meetings to be scheduled. When meeting requests are generated with important invitees, it will aim not to cause any collisions for the important participants. However, it is not capable of identifying priorities between the different meetings. In order to include this feature, participants should be able to identify topics

that interest them and a list of people that they consider important. The prioritized topics, and the relative importance's of the people to each other would enable for the development of a system that is able to prioritize the minimization of collisions on meetings that are considered more important for a participant.

In addition, the algorithm does not include the feature of allowing for specifying prerequisites for meetings. Batch scheduling meetings is a beautiful feature; however, in some instances it is possible that there are multiple meeting requests in the batch pool that should have a specific order. As an example, "meeting #1 should take place at least 5 hours before meeting #2". If a participant decides to schedule multiple meeting requests, he/she should be able to specify some prerequisite constraints on these meetings. This additional objective could be included in the objective function to enable for a more complex system.

Mobile Application

Even though I believe that the final version of my mobile application is a useful and complete software, I am aware that it has a list of limitations. One such limitation is the invitation system. The leader of a team has the permission to invite other members to the team. However, the invited members are instantly given a membership to that team, and they are not required to accept the request. Implementing this feature would provide a more efficient membership system and would allow individuals to have full contorl over their memberships.

Also, there are very few user preferences available. The user only has the ability to sign out of his/her account and change his/her meeting time preference. When I designed the application, I was planning on including other options, such as setting the maximum number of meetings per day, the amount of break between individual meetings as well as conveying the preference between small and large gaps between meetings. Even though these additional preferences would have provided more control for users over their meeting schedules, I believe that the application still gives an adequate amount of attention to user preferences.

Web Demo

When I decided to implement a web interface to test the functionality of my different algorithms, I did not plan to make it nice and user friendly. However, I understand that it would have been advantageous to pay more attention to the details of the software. As an example, the user is able to provide false data in many places, which have the ability to break the algorithm. Even though the website is not connected to the database, and has no ability to affect on any user data, it would have still been a nice software engineering practice to validate every user input.

8.0.3 Summary

Even though my software system has a list of limitations, which could have significantly improved its quality, I believe that the final result of my project is a complete system, which allows users to schedule meetings, test the functionality of the scheduling algo-

rithms and enjoy quality user experience.

9 Legal, Social, Ethical and Professional Issues

Following ethical guidelines and constantly keeping in mind ethical considerations are practices that a programmer must do in order to develop applications that are in compliance with data protection regulations and promote equality. The British Computer Society published a Code of Conduct, which lays out a list of guidelines that aim to provide a framework for professional and ethical behaviour in informatics [4]. In order to make sure that I am ethically correct during the entire cycle of my development process, I followed these guidelines. Another advantage of following such guidelines, is that the programmer might not be able to constantly question and argue about the ethics of every aspect of his/her developed system. Therefore, to ensure that no moral principles are overlooked, a guideline like the BCS Code of Conduct can come in handy.

In order to adhere to 'Professional Competence and Integrity', I made sure to use open-source libraries and develop my code using modular practices so that future developers will be able to contribute to it if necessary. In addition, I made sure to use the latest cryptography techniques to hash user data and maintain high security standards.

Finally, in order to make sure that I positively contribute to society and do not exclude any group of demographics, I developed a system

that is available to the most amount of people. As I mentioned in the report previously, my application is available on both IOS and Android, and the scheduling algorithm can be tested through a web interface in a user-friendly manner. As a result, I believe that my software system does not discriminate any individual or group, and is available to users independent of their location, gender and sex. Furthermore, I do not collect any user data, other than the ones necessary to identify individuals for the account system.

10 Conclusions and Future Work

10.0.1 Conclusion

Even though I successfully completed every component of my software system that I planned, I encountered many difficulties, which required me to alter from the initial plan. However, I believe that looking at the entirety of my software system, which consists of a mobile application, a website, an AWS cloud backend and a Django API for scheduling the meetings, I am positive that I constructed a meaningful system from which my users will actually benefit from. I decided to give a name to my application at the end of my development process, which is Invitely.

I believe that my mobile application, developed using React Native, has a beautiful user interface and strong functionalities. I never used React Native or JavaScript before; therefore, I was required to learn both in order to start the development. I was also forced to learn the functionalities of many libraries, such as Context or

SafeAreaView. Even though I generated a plan for my app, because of my lack of knowledge on what is possible with React Native, I had to constantly adopt new skills and practices to implement the desired functionalities.

Deciding on implementing a website to demonstrate the functionality of my scheduling algorithms was a beneficial choice, because it also allowed me to constantly test them without the need to generate the data inside my mobile application. Overall, I believe that I spent more time developing the demo, which was a huge positive because it allowed me to completely customize the scheduler.

Learning and working with Google Or-Tools was a fantastic opportunity. It not only allowed me to develop a meaningful algorithm that is comparable to many of the competitor apps' difficulty, but it also provided me the chance to learn a new skill, which I will be able to transfer to solve other problems in the future. Or-Tools is an extremely complex library, with thousands of functions and features. I spent many days learning about it and its wide range of features before actually starting to implement basic systems with its constraint programming solvers. I believe that Or-Tools allowed me to develop a scheduling algorithm that is greatly open for extensions, and I am positive that I will continue working on it in the future.

Lastly, I would like to mention my learning process on cloud computing. I have never worked with AWS before, and I have never

developed backend APIs. Learning the basics of cloud computing and the functionalities of all of the AWS services which I ended up using took me a long time, but I believe that it is a marketable skill that I will definitely benefit from in the future. Building the Django API using the REST framework allowed me to understand the connection between the different parts of a software system ,and provided me with the skill of building powerful and structured backends.

10.0.2 Future Work

Even though Invitely is a working MVP, there are many implementation and extension opportunities that could significantly improve its performance, usability and functionality. Below are a list of features that would definitely make my software system more thorough and efficient.

Detailed Account System

One significant extension that could improve the functionality and user experience of my application is a more detailed account system. As of now, users are only asked to enter their email address and username to the system. Even though this ensures compliance with data protection policies, more information about users would enable for the development of more sophisticated social features inside the app. Knowledge about the users' age, location, preferences, previous events, occupations and many more would allow the app to generate more precise recommendations, schedule more reliable and optimal meetings and allow for the implementation of a social network inside

the app.

Calendar Sync

Enabling users to synchronize all of their calendars to the app would significantly improve user experience. It would automatically include the unavailabilities to the meeting schedulers, which would be able to generate more efficient results to the users. As of right now, the only unavailabilities that the scheduler is able to utilize are the meetings that were scheduled through the app. Due to the fact that there are many calendar systems available on the market (Google Calendar, Microsoft Outlook, Apple Calendar), it would require a lot of work to implement a synchronization option for all of them.

App Search

Even though I created a screen inside my application that could allow to search for content inside the app, I ended up not implementing this feature. Using this feature, users could search for any data inside the app, such as teams, meetings and other users. In order to implement this, a more details account system, which I mentioned above, is required, so that the app has more information about the users.

Native Development

Even though React Native allowed me to deploy the same application to both IOS and Android, it has some limitations when it comes to native elements. Some elements, especially the ones that were recently released (3D touch, system colour), are not customizable

through React Native. Therefore, developing the same application on native platforms, such as Xcode and Android Studio would allow for the customization of native elements.

User and Team Avatar Selection
Another idea I came across during the development of the application was the implementation of a feature that would allow users and teams to select from a wide range of icons to use as their profile avatar. Even though this seems like an easy implementation, the most significant issue with this is acquiring the license to the tens or hundreds of icons. Another option is to hire a digital designer to create them, which would also come with a significant investment.

11 User Guide

Finally, I would like to explain how the entire software system comes together, and a step-by-step procedure to run and test it.

Django Back-end The Django back-end is deployed on Heroku. The file, which I provided includes the setup instructions and settings, which are required for the deployment. As a result, it is not required to host it locally. The back-end API can be accesed through https://prj-backend.herokuapp.com/, which does not contain any front-end code. Every other software accesses this backend through the mentioned link.

Web Demo The web demo is not deployed; therefore, it is required to host the React app locally. To do this, enter the project folder,

web_demo, and install the required dependencies:
npm install
Then, start the react server: npm start
The web app should be accessable on localhost:3000

Mobile Application

The mobile app is also not deployed, as it requires a lot of investment to create an app store account. In order to test the app locally, the user should have either a simulator on their computer device (Xcode Simulator, or Android Studio), or download the Expo Go app on their smartphone. Once this is done, enter the folder 'MeetingScheduler' and install the required dependencies:
npm install

Next start the server: npm start

The app should be accessible through either using a simulator, or by scanning the displayed QR code using the Expo Go app.

AWS Amplify

In order to manually schedule the algorithms, the following should be executed:
amplify mock function testBackend

In order to do that, the user must connect to my Amplify account through the CLI:
amplify configure project

For my AWS credentials, please get in touch with me as it would breach safety security measures if I included the tokens here.

12 Bibliography

References

[1] Android voice chat app with 5m installs leaked user chats.

[2] Android vs ios: Mobile operating system market share statistics you must know.

[3] April 2023 mobile user statistics.

[4] Bcs code of conduct.

[5] Doodle.

[6] Savvycal.

[7] What is a rest api?

[8] Amazon. Amplify, 2015.

[9] Michael Clarke. A brief history of meetings.

[10] Mansoor Iqbal. App revenue data (2023).

[11] John J. Kanet. Constraint programming for scheduling.

[12] Peter J. Stuckey. Lazy clause generation.

[13] Peter J. Stuckey. Search is dead, long live proof.

[14] Pal Laszlo. Global optimization algorithms for bound constrained problems.

[15] Greger Ottosson. Integration of constraint programming and integer programming for combinatorial optimization.

[16] Janos Pinter. Global optimization.

[17] Gerrit Renker. An introduction to interval-based constraint processing.

[18] Olga Tosic. Calendly reviews: Is it really worth it to use this booking app?, Sep 2022.