Optimising Conference Scheduling to Benefit Organisers and Delegates

Student Name: J.N. Roy
Supervisor Name: Prof. T. Erlebach.
Submitted as part of the degree of BSc Computer Science to the
Board of Examiners in the Department of Computer Sciences, Durham University

Abstract—Conferences are important events that take place all over the world. They serve as a platform for researchers to share their latest discoveries and advancements in technology and humanitarian efforts. However, organizing a conference can be a challenging and time-consuming task. There are several critical factors to consider, including the number of attendees, workshops and talks, refreshments, location, and more. One of the most complex and crucial aspects of conference organisation is creating a schedule that meets the needs and preferences of all attendees. This paper presents an automated and optimised approach to schedule any conference efficiently and effectively. The proposed solution involves building an online web application with a user-friendly interface that hosts two powerful automated scheduling algorithms that each ensures a schedule is optimal for all attendees. This application can be used by any organisation planning to host a conference. Conference attendees can sign up for the event through the web application, and the algorithm will create a schedule that considers their interests and preferences, among other factors. Two scheduling algorithms were developed - one that uses integer programming and graph theory techniques to create schedules quickly and another that uses genetic algorithms to produce higher quality schedules. The web application that was developed was user-friendly, however, it encountered some latency issues while serving clients, and at the same time running integer programming models and genetic workflows. It was discovered that the integer programming and graph visualization technique generated satisfactory results and was efficient and fast compared to the genetic algorithm approach which was slower but produced more effective solutions.

Index Terms—Conference Proceedings, Graph Algorithms, Integer Programming, User interfaces, Software Engineering for Internet Projects, Genetic Algorithms

1 Introduction

Conferences are large meetings of many people to engage in or share research on a particular activity or subject. Conferences facilitate the discussion of various contending or pressing issues on economy, peace, literature, religion or science. There are many conferences taking place all around the world, almost every day, helping to further the development of the world we live in. Usually, conferences require a substantial deal of planning which may involve, inviting speakers to share and present at the conference, marking out a date / set of dates in the year, securing a location, gathering registrations, and ensuring there are enough refreshments and resources to cater for all registrants.

1.1 Context

One of the most complex and difficult of aspects organising a conference is the scheduling of all the various talks and activities throughout the time of the conference in a way that ensures everyone can engage with and receive from the conference, or partake in parts of the conference they are more interested in. Often it can be difficult and time consuming to produce a conference schedule that does the bare minimum – ensuring every talk planned for a conference is scheduled with a time for it to take place – but even more so to produce a properly effective schedule that best optimised for all the attendees, delegates, speakers and organisers. An optimised schedule aims to ensure that all delegates are satisfied by the end of the conference, and whilst satisfaction can be an abstract measurement, this paper defines 'the satisfaction of delegates at a conference' as ensuring that all delegates gain, from the conference, that which they had initially planned to receive. Delegates may register to a conference with specific interests, and preferences in mind, and so they would prefer to engage in the conference in line with those interests – for example a registrant of a conference themed in a field of plant biology, who is particularly interested in hearing about the research in the developments of plant-based medicines, is assumed to want to hear talks or engage in workshops related to the theme of plant-based medicine. Thus, there is very little reason to assume that any delegate would not have enjoyed attending a conference if the schedule organising the conference was built to maximise the satisfaction of all delegates.

This project aims to propose a software solution to optimise the generation of conference schedules, in a way that maximises the satisfaction of delegates. Delegate satisfaction can be maximised by prioritising the schedule to ensure the presentation of desired talks, through understanding the different trends of delegate interests, whilst also catering to the availability of resources (and adhering to other constraints) set out by the conference organisers. While it would be useful to ensure that all delegates can see their desired talks, it is possible that the schedule creation can potentially create inconvenient conflicts. A conflict can arise when any delegate wishes to engage with certain talks / activities, but two or more of these activities are scheduled at the same time. Because it is impossible to be in two places at once, this delegate would only be able to attend one of these activities and miss out on the other. Ideally, this delegate should prefer to go to both talks, which would maximise their satisfaction of going to the conference. Therefore, a good conference scheduler should minimise conflicting interests for all delegates.

This paper proposes that software can be used to automate the process of schedule generation for not just one conference, but for any organisation that wishes to reduce the amount of time taken to plan the timings of their conference. As mentioned, conferences require a great deal of planning, and a lot of conference organisers spend much own time scheduling their conference, by themselves, which can be a tedious and time-consuming task, which could be spent on other organisational tasks. With the usage of an automated software application, the time spent on scheduling could be spent further planning, refining or re-affirming other logistics of a conference, such as sorting pricing and budgets, reviewing content to be presented, securing locations, organising accommodation services (where relevant) and the other crucial aspects.

1.2 Deliverables

This paper aims to combine different approaches to scheduling and combinatorial problems with software catered towards providing a good user experience, to produce an application that effectively assists conference hosts in organising their conferences and produce effective and useful schedules. The software will work alongside the process of planning, launching, running and ending a conference, but will focus on maintaining the schedule-generation aspect entirely. The application will be designed and developed to facilitate the scheduling of sessions (blocks of time that host one or several talks) by recording and processing conference details, timings and other relevant information. It is assumed that the conference organisers would decide on the topics / talks / sessions that need to be made available to the attendees well in advance of their registration. Once the conference is open for registrations, all interested attendees are expected to register within a given time, before the starting date or registration deadline. Subsequently, the application should allow delegates to indicate their choice of talks and activities of interest. With all delegates registered, the application is expected to schedule an optimised timetable for all activities, catered to all the different attendees, based on their choices or preferences. Once scheduled, it should provide necessary information to the delegates, to know where their talks are held, mitigating as many conflicts as possible.

Regarding the nature of the scheduler itself, there were initially a multitude of methods to choose from. There were three main methods that stuck out the most, each of differing assumed speed, effectiveness, and complexity, that would allow me to compare how well different methods perform, judge which is best for a commercial platform designed for conference scheduling (akin to the application described in this paper), and provide more of an informed judgement on the estimated performance of other methods not tested in this scenario. This project plans to make use of three main methods to create two types of schedulers, and aims to satisfy multiple functional and non-functional requirements. These have been split into mandatory requirements which will ideally be met by the end of the project lifespan, and some advanced requirements which will not be placed at the highest priority but can deliver some increased functionality.

Mandatory Requirements: The solution will be an online web application that facilitates the creation of profiles for conferences currently being planned. Conference profiles will be created by one of the two main user types, who can fill out forms and provide data to create an account for their conference on the system, which include all relevant dates, some relevant logistical information, and data on conference talks (activities). Conference talks can be described using special keywords called ”topics”. The users that can use the system in this way are called conference ”Hosts”. The other main user type are referred to as ”Delegates”, who can also log in to the system and register their attendance for a conference, after which they can preview the talks taking place in their chosen conference and rate each activity with their level of preference, indicating how much of a priority a certain talk is to that delegate. All of this information is fed to the *The Standard Scheduler* - one of the two scheduling approaches, chosen by the scheduling algorithm on the application server, to create conference schedules. This scheduler makes use of integer programming (IP) in conjunction with graph theory techniques to create schedules of talks and activities for a single-day conference. This describes the functional mandatory requirements; the non-functional mandatory requirements describe that users can interact with an intuitive graphical interface. The web application will be built using a client-server based application model, to handle accounts and provide an admirable user-interface for users. integrated with a CRUD database. Finally, the Standard Scheduler and the web application should facilitate single-day conferences.

Advanced Requirements: Building upon the same aforementioned web application, conference data will be given to another scheduling approach, more advanced the Standard Scheduler, called the *The Genetic Scheduler* which is built to employ heuristic genetic algorithms (based on Darwin’s ”Survival of the Fittest” theorems) to iteratively generate, evaluate, refine, and optimize various schedule combinations until an optimal arrangement is achieved based on the provided information. An multi-threading evaluation-simulator function will also be used to generate a metric that describes the success of the schedule - how well the generated schedule satisfies the preferences of delegates. Either scheduling approach used for a conference will be able to adapt to change conference details (until the start of the conference). This describes the functional advanced requirements; the non-functional advanced requirements describe that the solution can facilitate multi-day conferences. It would also be desirable to make use of recommender systems that can suggest conferences to delegates based on their preferences and registrations to previous conferences.

In this paper, we propose to optimise the generation of conference schedules using computer science methods. It aims to provide a solution to optimise conference scheduling to benefit organisers and delegates.

2 Related Work

2.1 Nature of CSP

Conference Scheduling Problems (CSP) involve creating an optimal a schedule for a host of a conference, while meeting certain objectives. These objectives may prioritise resources, expenses, people, or organisation. It is difficult to predict how much time it will take create a schedule for any conference, but this paper aims to present a software solution that can create a schedule for any conference in the most optimised way. The scope of this project will revolve around the context of a public conference organisation using a software solution to tackle CSP with regard to hosting sessions containing talks on papers published. A conference, hosted by an organisation, will host many ”talks” based on papers about topics related to the conference; each of these talks would be grouped into ”sessions”. Each of the sessions would take place at different points of the day at different ”locations” (commonly in lecture theatres). The main constraints around this problem are that many sessions need to be organised for a limited time frame, leaving no choice but to schedule multiple sessions of talks to take place at the same time - this is referred to have *parallel slots* of talks.

Without considering the delegates attending the conference, CSP in the context of conference scheduling would be an easy problem to solve heuristically. However the scope of this project includes factoring in the availability of delegates, which greatly increases the complexity and approach to the project. With regard to this, a solution to this problem needs to be designed that the number of delegates attending each session is maximised. Popular talks with similar sets of delegates attending them could be grouped together. This literature survey of related work will look a variety of approaches to solving CSP, to explore the different methods (and their effectiveness) used in previous research and how they might or might not be useful in this context.

First it is useful to understand the complexity and difficulty of CSP, as this gives solution designers an idea as to what makes a good algorithm - where an algorithm is a list of instructions to solve a problem, as described by Cook et al [11]. In an ideal world, a good algorithm, to a problem k, finds solution to an instance of k, in as little time as necessary, and that can be verified as correct. However, there all sorts of problems with different inputs, constraints and aspects; they may not always be a fast algorithm, or an algorithm that can be proved to produce an efficient solution (though the solution itself generated solving an instance of said problem can be proved to be correct). As such programmers need to classify the problem they are dealing with. In the branch of theoretical computer science that studies computational complexity, resources and computational problems, there are several classes of problem complexity:

- P *(Polynomial-Time)*: The class P, as described by Cook (et al) [11] denotes the set of all problems that can be solved by a polynomial time algorithm. A polynomial time algorithm solves a problem of size n in (worst-case) $O(nk)$ time, for some integer scalar value of k. An example of a problem that resides in the P class of complexity is the problem of sorting a list of elements; algorithms that can solve this problem include the widely known sorting algorithms ’Quick Sort’ and ’Merge Sort’.
- NP *(Nondeterministic Polynomial-Time)*: The class NP holds the set of problems that more complex that those in the P class. They are described by Cook (et al) [11] as the set of (roughly) all problems - for any input size that has a positive answer - for which a ’certificate’ solution can solve the problem, and the correctness of this certificate can be derived in polynomial time; the method to find this certificate may not be derived in polynomial time, but may require exponential time or even further beyond. Examples of problems that reside in this class include the ’Hamiltonian Circuit Problem’, or the ’Travelling Salesman Problem’. Such problems have solutions that can be verified quickly, but finding said solution is not necessarily easy or quick.
- $NP-Hard$ *(Nondeterministic Polynomial-Time Hard)*: This class of problems describe those problems for which there is no known polynomial-time algorithm. NP-hard problems are found by using ”reduction theory” - an algorithm for transforming one problem into instances of another to associate the two problems - to reduce problem A to $NP-Hard$ problem B, thereby proving problem A is also $NP-Hard$. Generally, these problems are, at the minimum, as hard as the hardest problems in NP. An example of an $NP-Hard$ problem is The Boolean Satisfiability Problem (SAT).
- $NP-Complete$ *(Nondeterministic Polynomial-Time Complete)*: This class describes problems that are both NP and $NP-Hard$; problems in this class are amongst the most difficult problems in NP as claimed by Cook et al [11]. Characteristically, it is theorised that if a polynomial-time algorithm could be found for any NP-complete problem, then it could be used to solve all NP problems in polynomial time. An example of an $NP-Complete$ problem is the Knapsack Problem.

From this set of problem classes, it is important to distinguish which class CSP would reside in - in order to correctly justify a possible solution. Quesnelle and Steffy [1] demonstrate a proof of the computational complexity of a slightly different version of CSP (compared to the one in this paper) by observing the *Timetable Design Problem* (TDD). The TTD problems aims to determine if there exists a feasible schedule for a set of time slots, a set of teachers coupled with available teaching hours, and a matrix that contains the courses which the teachers can teach. It is useful to observe TTD as a reduced model of CSP since aspects from both contexts are quite similar, and the objective of both problems are near the same. Both CSP and TDD share a resource in the variable set of time slots; the available teachers in TDD can be likened to set of available talks to be given in CSP (deliberately not the matrix of subject courses) as they arguably can be considered the resource of most importance. The set of available teaching hours, can be likened to available rooms in a conference (except that conference rooms are not bound exclusively to specific talks). The matrix of subject courses can be likened to the topic and themes associated with conference talks - even more so if multiple talks share a theme. However, there are some constraints in TTD that may not map in CSP. For example, certain variants of TTD include the constraint of the willingness of a teacher to teach, or the usage of multiple teachers to teach a course (which would affect the teach availability for some courses). Nonetheless, a justification can be made to reduce CSP to TDD. After creating a definition of objectives and constraints for TDD - and subsequently CSP with a change in one of the constraints - Quesnelle and Steffy go on to describe their CSP as being $NP-Complete$ via reduction from the Graph k-colourability problem [1]. Quesnelle and Steffy's approach was partly advisable in providing a good starting point in framing the requirements for the definition of CSP used in this paper, but it was clear that more insight was required.

Vangerven et al [2], describe an approach to conference scheduling that prioritises maximising participant satisfaction, based on an input of preferred presentations from said participant - which is quite similar to the aims and objectives of this paper. They describe a conference scheduling problem with a set of talks X, consisting of x talks, multiple sessions in a conference schedule consisting of k talks per sessions, all to be scheduled a set of time-slots T, with $|T| = \frac{|X|}{n}$. For every participant $p \in P$, there preference profile for that participant. The name for this problem is called CSP-n, where n refers to conference schedule with n active parallel sessions happening at any one time. They frame their problem's objective to specifically maximise attendance for every talk (assuming every conference room has infinite space). Vangervan et al [2] go on to state that CSP-2 is solvable in polynomial time, but declare that CSP-n is $NP-Hard$ for $n \geq 3$ - which they prove transforming an instance of the *Triangle Partition Problem* (TPP) into an instance of CSP-3.

This paper describes CSP in a manner more similar to that of Vangerven et al, where conferences schedules are made up of time-slots in which talks of different topics and themes are assigned (as described earlier). For every organisation that uses the web solution proposed here, there will be different instances of CSP. Like the problem descriptions proposed by Quesnelle and Steffy [1], and Vangerven et al [2], CSP aims to make heavy usage of the preferences, held by conference delegates, for different talks. However, unlike the aforementioned problems, solving CSP (in this paper) will be putting the priority on ensuring all delegates are able to see they talks they wish to see, as a pose to maximising talk attendance (which will made a lesser priority), or catering to the availability of speakers. It can be justified that CSP is NP-hard for any number of sessions when considering its computational complexity.

2.2 Approaches to scheduling

After determining the computational complexity of the problem, research was conducted to investigate the current approaches to creating a solution to the conference scheduling problem. One of the papers that helped to create the requirements for this project was a conference scheduling paper given at the PATAT-2016 conference [2]. The paper describes the context of scientific conferences scheduling lectures and sessions that talk about research papers. Vangerven (et al) describe using a hierarchical optimisation approach, sequentially optimising in three phases (each containing an integer programming model). The first phase aims to maximise total attendance at the conference, based on the preferences of all the guests that are registered - it is solved using the Constraint Satisfaction Problem. The second phase minimises session hopping, so that guests need not travel far between sessions. With more session hopping, there is greater inconvenience of moving between rooms, which can reduce overall satisfaction. The third phase aims to optimise the arrangement of presenters for all the sessions created in the second phase, ensuring the number of violated presenter availabilities is minimised. A violated presenter refers to a presenter who availability or preferences have not been met in the scheduling process. For example, a presenter is scheduled at two sessions simultaneously or if their preferred time slot or room assignment is not accommodated. This paper provided a lot of valuable guidance on a possible approach to creating schedules. It shows that integer programming can be a viable method for creating schedules for conferences. The solution in this paper does well to maximise total delegate attendance - which is a realistic objective for a scheduler, and a realistic pursuit for all conferences. A similar point can be made for attempting to reduce the level of violated presenter availabilities. Though in a realistic scenario, it might be hard to keep this as a top priority. The paper demonstrate that integer programming models can be used to model real scenarios, and moreover, that integer programming can be used simulate abstract constraints. This was part of the inspiration to use integer programming in final solution for this project.

Castro and Petrovic [3] describe a different scheduling problem context, but with a similar solution. In this context, 'pre-treatment' tasks, for radiotherapy in hospitals, require scheduling. In radiotherapy pre-treatment, each patient follows an ordered sequence of examinations, referred to as a pathway, consisting of several operations. The pathway is determined depending on the classification of the patient and site to be treated. These tasks help prepare hospital patients for radiotherapy treatment, and each patient requires many pre-treatment tasks to be completed for them. The hospital in question accommodates for many such patients, thus a scheduling system was required to organise the assignment and completion of all pre-treatment tasks, for all patients, in the most optimised way. The main solution for this problem was to form a mathematical program and create dispatching rules. Dispatching rules are used to generate an initial solution for the mathematical programming model. The performance of the dispatching rules is compared to the mathematical programming approach. This paper provided insight into the extent to which mathematical programming can be used to manage, organise different resources - this including not only time, but equipment, rooms, human resources and many others - however the complexity of a program such as this was deemed out of scope for this project. Despite this, this paper re-affirmed the sentiment that mathematical programming of some form could prove to be very useful.

Chan and Chung [4] describe that a multi-site production company's schedule can be modelled as distributed scheduling problem (which is NP-hard). There are various locations and entities (which are classified as nodes). Each of these entities have numerous tasks to perform; some of which may be shared with other entities, or be unique to a specific node instead. When modelling using distributed scheduling, it was noted that some constraints were that all available resources were pooled and shared among entities, which always allowed easy collaboration between entities. Interestingly, Chan and Chung investigated an artificial intelligence-based approach to solve the problem, unlike any other AI solution I have seen before. This was the multi-agent-based approach - aka the Mutli-agent system (MAS). In this method, multiple decision agents, created artificially, are trained and interact in a shared environment to achieve common or conflicting goals, creating a semblance of artificial intelligence. Each agent would manage its own handle and monitor its own schedule. A global schedule could then be created through negotiation and coordination of each agent. The context of this project by Chan and Chung is much different to this conference scheduling problem presented in this paper, neither can its solution be applied to solve or be used to model CSP in any way, but it presented the very interesting and potentially lucrative idea of leveraging machine learning to solve an $NP-Hard$ problem.

Another vastly different but interesting approach was produced by Zhou, Nee and Lee (in 2009). Zhou, Nee, and Lee describe the context of dynamic job shop scheduling problems. The authors investigate the algorithm's performance in different scenarios with varying machine utilization levels and processing times. Zhou, Nee and Lee describe each task in the job shop scheduling problem (JSSP) has some operations to be processed in certain technical sequence and each operation has a predetermined processing time. The main goal was to find a schedule to process all jobs in a manner to optimise given performance objectives. The paper's solution involves using a metaheuristic ant-colony algorithm as an optimisation solution (ACO). ACO was tested in various scenarios with different levels of machine utilisation, processing time distributions, and performance measures. It was compared with dispatching rules such as first-in-first-out, shortest processing time, and minimum slack time. According to the summaries of the experiments and tests conducted in research, ACO produced good optimisations and did better than other approaches, when producing schedules with lowest mean flow time, mean tardiness, and total throughput. However, it was noted that ACO performed better when the machine utilisation or the variation of processing times was not high compared to other approaches. It can be justified that the job-scheduling problem shares more aspects in common with CSP, than the multi-agent system project by Chan and Chung. Both the ACO model and the CSP problem deal with assigning events to different time-slots in a day regarding a single are, in the most optimal way (but this might one of the few things both contexts might share). Nevertheless, this paper provided some interesting ideas, proposing that the use of a metaheuristic algorithm, like the genetic algorithm, could be a possible approach to solving CSP.

3 METHODOLOGY

This section presents the solutions to the problems in detail. It describes the style and prerequisites of the input data, the functional requirements of the system's processing, and the style and usability of the output information.

3.1 The Web Application: ConferenceScheduler

In designing the solution for this problem, it was useful to consider what it might be like to host a conference.

3.1.1 Input: User Types

There are two main users for this web application:

- *Hosts* - Conference organisers can create accounts on the system. Using these accounts they can create profiles for conferences they are setting up. For any conference setup by any Host user, setting up a profile for said conference will involves providing the name for the profile (commonly the same as name of the conference itself), an optional *unique-resource-locator (URL)* for the conference, and a set of dates (which each mark the beginning and end of a phase in planning and running a conference) including:
    1) *Paper Review / Talk Finalisation Date*
    2) *Delegate Registration Deadline*
    3) *Conference Start Date*
    4) *Conference End Date*

  Each of which will be explained further in section 3.1.2. Creating the conference profile also includes describing the start and end time of each day the conference is active, the average length elapsed for a talk, and the maximum number of different talks that can happen at any one time-slot (this was a factor considering the scenario of a conference location being able facilitate multiple talks in a time-slot by having different talks in different rooms).
- *Delegates* - Conference attendees can create accounts for themselves on the system. Using these accounts, they can register to existing conferences created by hosts. Delegates can sign up to the web application using an email address, a user name, a secure password, and a date of birth, to uniquely identify their account. Using the web application, they can search for their desired conference, register to it, then preview the list of talks / activities decided to be presented during the time of the conference. Delegates can indicate their *preference* to see certain talks, by way of a measured score meter.

3.1.2 Input: Conference Interactions

The usage of the Conference Scheduler web application follows a singular main flow of processes (which can be observed in figure 1). The main process flow, can be repeated many times for many different conferences as they are created, or for the same conference as changes are made and edited, but the overall structure remains the same. The usage of the web application begins with the creation of user accounts (the types of accounts are described earlier in section 3.1.1) which will have slightly different behaviours depending on the user type. *Host* users can create, maintain, and terminate conferences - given the relevant resources - whenever they desire as long as they are able to establish the different *conference phases*, and create all the different *talks* involved in the conference - where a *talk* is a blanket term for any activity taking place during a conference, with the exception of breaks for refreshments, or lunch times. A talk may be described by the *speaker* presenting it, and *topics* associated with it - where a *topic* is an abstract keyword that has meaning in the context of a conference's theme (for example, 'The Gospels' in a Christianity conference) that can be used to describe the content of a talk.

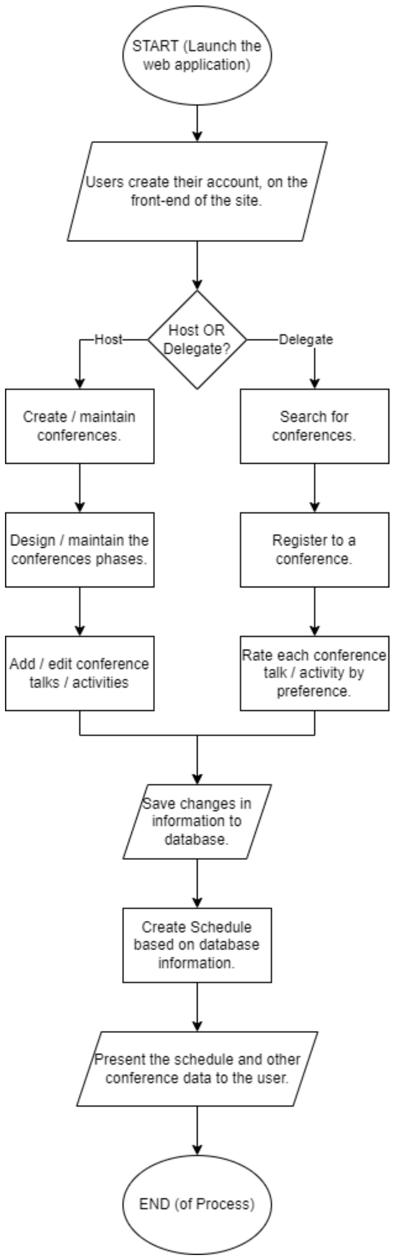

START (Launch the web application)
Users create their account, on the front-end of the site.
Host OR Delegate?
Host
Delegate
Create / maintain conferences.
Design / maintain the conferences phases.
Add / edit conference talks / activities
Search for conferences.
Register to a conference.
Rate each conference talk / activity by preference.
Save changes in information to database.
Create Schedule based on database information.
Present the schedule and other conference data to the user.
END (of Process)

Fig. 1. A Flowchart describing the holistic flow of using the web application.

The conference phases refer to periods of time leading up to, and during the running of the conference, and will be decided by the host by the time they create their conference. They help the Hosts keep a track of time, but also allow the web application and the scheduler run efficiently. The first phase starts from the creation of the conference and runs until the *Paper Review / Talk Finalisation Date*, which outlines the end of the period of time when all the talks, activities, workshops (and such) have been finalised (the exact date of which will often be months before the conference begins) and has been provided to the software system. Within the context of conference planning, this phase may involve conference hosts waiting on other individuals, parties, or organisations to express an interest in presenting a talk, or activity, or delivering some form of content at their conference. Alternatively, this phase might provide some time for the conference hosts themselves to produce a presentation, or set of presentations of their choosing. It tells the web application that not much computation is warranted until this phase is completed. It is assumed that upon passing the date for talk finalisation that all activities have been finalised, and the web application can proceed to start running the scheduler algorithm on the conference data. The web application is designed to operate on a continuously running server, so it was preferred that it should not have to waste computer resources running a heavy scheduler (at least in the context of the Standard or Genetic scheduler) on conference data that is not yet finalised. It is not a strict deadline as talk information can be still be provided as input during the next phase - as if the scheduler would immediately terminate the conference if no talk data is provided by the time this deadline arrives - but it acts a guide to the scheduler algorithm, as well as the delegates who may have an idea of when they shall know what content might be delivered at the conference, and time at which they might indicate their preferences.

The next phase runs until the *Delegate Registration Deadline* which depicts the phase in which delegates are able to register themselves to a conference. Whilst this phase can run in parallel with the Paper Review / Talk Finalisation phase, given the new considerations that will be introduced in the new phase, it was intended that this second phase takes place after the end of the of the Paper Review / Talk Finalisation Date. This new phase will allow the delegates of a conference to preview the (ideally) finalised list of talks and indicate what level of preference they might have toward each item - which the scheduler will use in its generation process. Within the context of conference scheduling, this phases takes place shortly before the start date of the conference, and all throughout its duration, the conference scheduler run until the conference finally begins. After this deadline has passed, new delegates will be unable to register to the conference.

The final phase begins with the *Conference Start Date* and ends with the *Conference End Date*. It describes the period in which the conference takes place. By the time the start of this phase is reached, all of the logistics will have been sorted out, all of the unique planning aspects will have been finalised, the conference follows through with presenting its registered talks, and delegates go to the conference seeing those talks that they put an interest in. This is where the results of the scheduler will be used. The web application will have finalised the itinerary of all talks and activities for each of the conference, using different combinatorial methods. During this phase, users registered to a conference can view the generated optimised timetable in the web application a mobile device. As they move from activity to activity, the generated solution will guide users such that they are able to visit the talks they wish to see. The scheduler will no longer run on a conference in this stage; it should finalise its choice of solution before this phase begins, lest (should it operate whilst a conference runs) it makes multiple inconvenient changes to the schedule, causing a confusion whilst a conference is running. Past the start date of the conferences, no changes would be made to a conferences profile, its talk information or its final schedule. A conference will be terminated, if reaches this final phase with any missing data in its profile. An example of a timeline for any set of conference phases can be seen in table 1.

TABLE 1
An Example Timeline of Conference Phases (in the year of writing: 2024)

Phase No.	Deadline for Phase	Deadline Date
1	START	
1	Paper Review / Talk Finalisation Date	25th May
2	Delegate Registration Deadline	15th July
3	Conference Start Date	20th July
3	Conference End Date	23rd July
4	END	

The scheduler algorithm should take various aspects of a conference into account when using combinatorial methods to create useful, optimised schedules. This includes the number of days elapsed during the running of the conference, calculated by finding the number of days between the conference start date and conference end date - each of these will be provided by the host when they create the conference. If a host sets up a conference to run for more than one day, this is used to ensure that the scheduler considers a wide availability of time-slots (in the common case that not all talks can be scheduled on one day). In conjunction with the number of days elapsed in a conference, the scheduler should also consider the *starting* and *ending* time for each day of the conference (which will be assumed to be the same each day). The scheduler will need to have a list of all available time slots in the conference with which to assign talks and activities, and this can be obtained by considering both the number of available days to run the conference, and the time period that will be available each day,

The schedule will also consider *sessions* when creating a conference profile. A session refers to a collection of talks in between two time periods, in a single day. Commonly, A day in a conference will often comprise of multiple sessions, separated by refreshments breaks or a lunch slot, with each session containing numerous talks. Some conferences may be able to facilitate the running of multiple sessions at the same time; two sessions may cover two different sets of talk within a shared set of time slots, for example. Sessions can be used to group similarly themed talks together - that is, talks that described using the same keyword topics - so that delegates experience a continuity of information as they progress through a conference. This is especially helpful if a delegate records a preference towards certain talks that are categorised with a certain theme; the usage of sessions promotes a continuity of information in the themes and topics that delegate is interested in, a method employed by Vangerven et al [2] during the development of their personalised conference scheduler. Thus, it was a requirement that talks that are i) recorded with association to certain topics, and ii) preferred by similar delegates with a preference to certain topics, are grouped together in sessions. In this solution, the arrangement of sessions in a conference schedule would be decided by the scheduler algorithms, but the host can inform the scheduler (by interacting with the user interface within the web application) of certain constraints on the nature of sessions. These include the maximum number of talks per session, the average length for all talks in the conference (and thus each session) and the maximum number of slots that can take place in parallel - which describes two or more sessions of the same size taking place during the same time slots. The maximum number of talks per session, and average length of talks allow the scheduler to create a timetable of slots to which different talks will be assigned - with adequate breaks being assigned between the end of one session and the beginning of another, as well as the accommodation of an extended lunch time break near twelve o'clock each day. As mentioned before, it is preferred that similar talks are scheduled together in sequence (such that n *similar* talks take place one of after the other in a single session); conversely parallel slots can be used to maximise the delegate attendance of talks by scheduling dissimilar talks at the same time, in parallel slots. The job of the scheduler is to ensure that any delegate A can attend their preferred talks without running into a conflict, and a conflict would arise if two preferred talks are scheduled at the same time (since a delegate cannot attend two or more talks at the same time), so the preferred talks should be scheduled at different time slots; the use of a parallel slot allows for a talk that is not preferred by delegate A(but perhaps preferred by other delegates B, C and D) to be scheduled at the same time as delegate A's preferred talks - which not only reduces the level of conflicts but serves to optimise the attendance and engagement from all delegates at a conference.

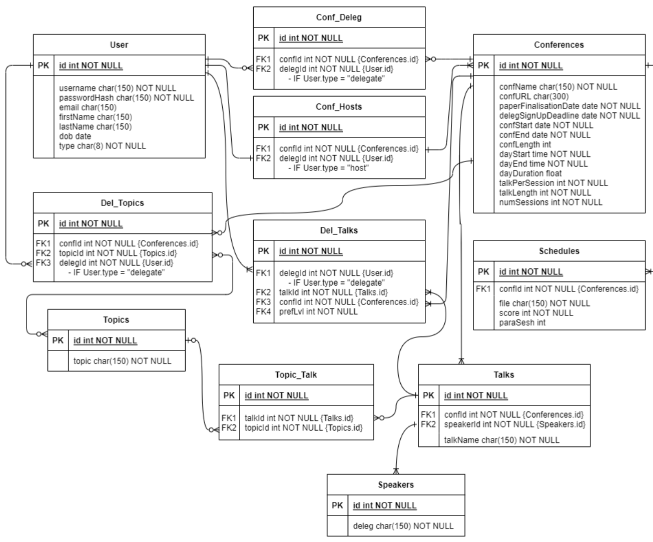
The final aspect of conference data that needs to be provided as input is to do with conference talks. After a host has provided the aforementioned information regarding the aforementioned deadlines for conferences phases, and the information regarding the nature of schedule sessions, the host will then be asked to provide the entire list of talks available for to be presented in the conference. A conference host may choose to fill out all talk information the moment they create a conference profile for the first time, or they might choose to not fill out any talk information (to complete it another time), or start with a preliminary list before ideally finalising the collection of talks before the start of phase two. Within the context of the web application, a talk (acting as a blanket term for all activities and content delivered in the conference) is comprised of a name, a speaker, and optionally a set of topics (understood by delegates) that describe the content of the talk. This is the data the scheduling algorithm will use to create a schedule. Each talk will be assigned to a slot, and the task of the scheduling algorithm will be to quickly, efficiently and thoroughly go through the permutations of talk assignments such that the best schedule for the conference is reached.

3.1.3 Functional Flow

The web application will be built using the Python programming language [5] using the lightweight Python web framework, Flask [6]. Flask will be used to build a server from which the web application is served to any client user, while the scheduling algorithm (and chosen scheduling approach) runs in parallel. Python was chosen for its large ecosystem of libraries and frameworks, that provide solutions to building all sorts of mathematical, combinatorial, and machine learning algorithms (as would be required by the Standard scheduler), whilst being able to easily maintain a robust web application. Python also lends itself for its having readable syntax, making it easy for developers to create and maintain programming code. Flask provides the essential functionalities for building web applications which provided a straightforward development experience. These functionalities include robust user authentication and authorisation, easy database support with the four essential Create, Read, Update and Delete (CRUD) operations [7], account-session management, and a usable functional parallelisation system. The functional parallelisation system was especially useful to be able to run both the web application's server program and the different scheduling algorithms in parallel - akin to as a nearly asynchronous system. However there were some notable trade-offs in using Python. One of these were the limited performance in processing speed, or optimisation. The Flask framework also lacks in scalability of features compared to comprehensive frameworks like Django (another web application framework in Python).

Once all the conference information has been provided to the web application, it will be saved into the web application's database; an Entity-Relationship Diagram for the database can be seen in figure 2. The database will hold all information on every user, conference, and schedule to be created and recorded whilst the web application is running. It will be developed using the python library SQLAlchemy [8] which has useful compatibility with python Flask; Flask supports SQLAlchemy out of the box. A multitude of entities will be used to organise and easily create, retrieve and maintain data but the main entities are the following:

- *Users* - this entity holds information on users, account details, unique identifiable details, and (most importantly) their account type - between that of host and delegate. Each user has their own unique identification number.
- *Conferences* - this entity holds all information on conferences, including key dates, timings, and session details. Two different join tables helps to distinguish which host user is responsible for which conference, and keep track of which delegates have been registered to each conference. Each conference has it's own unique identification number.
- *Talks* - this entity holds all information of talks from all conferences, storing each talk's name, speaker identification number (which can be cross-referenced with a table describing all conference speakers), and identification number of the conference it belongs to. Each talk also has it's own unique identification number. A join table connects each talk to the topics (keywords) that describe it. Another join table keeps a track of how any delegate user scores a talk in terms of preference, but only between a delegate and a talk that are associated with the same conference identification number (that to say, this delegate is registered to the conference for which the talk is being presented). Talks can be scored between 1 and 10, with a default value of 5 indicating no preference. A talk given a score of 6 or above is deemed to be *preferred* by the delegate that scored the talk that way, with a preference level of 8 or above indicating stronger preference.
- *Schedules* - this entity records the best evaluated schedule for each conference. Updates to this entity are made after a scheduler algorithm has concluded that it has generated a better schedule compared to what is currently prepared for a conference (under the condition the conference in question is in phase two of running (before the conference start date) and changes can still be made to conference details). The entity is updated by changing the specific schedule file saved to a conference.



Conf_Deleg
PK id int NOT NULL
FK1 confId int NOT NULL {Conferences.id}
FK2 delegId int NOT NULL {User.id}
- IF User.type = "delegate"
User
PK id int NOT NULL
username char(150) NOT NULL
passwordHash char(150) NOT NULL
email char(150)
firstName char(150)
lastName char(150)
dob date
type char(8) NOT NULL
Conferences
PK id int NOT NULL
confName char(150) NOT NULL
confURL char(300)
paperFinalisationDate date NOT NULL
delegSignUpDeadline date NOT NULL
confStart date NOT NULL
confEnd date NOT NULL
confLength int
dayStart time NOT NULL
dayEnd time NOT NULL
dayDuration float
talkPerSession int NOT NULL
talkLength int NOT NULL
numSessions int NOT NULL
Conf_Hosts
PK id int NOT NULL
FK1 confId int NOT NULL {Conferences.id}
FK2 delegId int NOT NULL {User.id}
- IF User.type = "host"
Del_Topics
PK id int NOT NULL
FK1 confId int NOT NULL {Conferences.id}
FK2 topicId int NOT NULL {Topics.id}
FK3 delegId int NOT NULL {User.id}
- IF User.type = "delegate"
Del_Talks
PK id int NOT NULL
FK1 delegId int NOT NULL {User.id}
- IF User.type = "delegate"
FK2 talkId int NOT NULL {Talks.id}
FK3 confId int NOT NULL {Conferences.id}
FK4 prefLvl int NOT NULL
Schedules
PK id int NOT NULL
FK1 confId int NOT NULL {Conferences.id}
file char(150) NOT NULL
score int NOT NULL
paraSesh int
Topics
PK id int NOT NULL
topic char(150) NOT NULL
Topic_Talk
PK id int NOT NULL
FK1 talkId int NOT NULL {Talks.id}
FK2 topicId int NOT NULL {Topics.id}
Talks
PK id int NOT NULL
FK1 confId int NOT NULL {Conferences.id}
FK2 speakerId int NOT NULL {Speakers.id}
talkName char(150) NOT NULL
Speakers
PK id int NOT NULL
deleg char(150) NOT NULL

Fig. 2. An entity-relationship diagram showing the structure of the web application's database.

As mentioned in the deliverables (section 1.2), ¡finalise this!¿ two/three types of scheduling algorithms were designed (all of which are discussed in further detail in section 3.2). Slightly different workings for the random and standard scheduler - talk about this. The Standard Scheduler will be implemented directly onto the back-end server of the web application, leaving the client-side of the web application to strictly operate as user interface. Through the user interface, host users and delegate users will provide the relevant information needed for a conference which will then be saved in the database (more information on user interfacing is discussed in section 3.1.4). The scheduler will iterate over all the conferences stored in the database, retrieving all information stored about the conference itself, the talks assigned to the conference, and the delegates registered to attending the conference as well as their preferences for each talk. For each conference, the scheduler will then use the gathered information to create a *master schedule* for every talk, every time-slot and every parallel slot, for every day the conference runs. This master schedule tracks every thing will happen in a conference, and would be primarily viewed by hosts on their account page on the web application, so that they may follow what is happening at all times. Delegates will be given a *derived schedule* to view on their account page on the web application, which contain the time-slot assignments for the talks the delegate indicated their preference for (in phase two, before the start of the conference). A derived schedule for any delegate will show fewer talks than the master schedule, only showing all the relevant talks in a single slot view (which is different to the master schedule showing all talks in a multi-slot view) since a delegate can only attend one slot at a time. This will be dependant on whichever talks were preferred by a delegate (whichever talks were scored higher in preference). An example of a master schedule can be seen in table 2, and an accompanying derived schedule for any conference delegate can be observed in table 3. Importantly, the scheduler algorithm will only create a master schedule; the web application's server will create the derived schedule directly from the master schedule as and when a delegate user logs into the system. This being said, a delegate may also decide to view the master schedule to see what else is taking place in their conference and choose to attend a talk, they had not previously given their preference toward. In any case, the master schedule created by the scheduling algorithm will be saved locally to the web application's server, in a formatted text file; a link to this text file will be then be created as a record and saved in the schedules entity within the database. Whenever the web application searches for the most up-to-date schedule for a conference, it searches for this record in the database which will provide the unique identification name for the file containing the schedule, before loading it into the user interface.

TABLE 2
An mock example of a master schedule, for a host user - the second day in a conference

Time Slot	Slot 1	Slot 2
11:00	Talk A by B. Charles	Talk D by E. Fyalta
11:30	Talk G by H. Islington	Talk J by K. Lemmings
12:00	Talk M by N. Oshunthanum	*No Talk Scheduled*
12:30	LUNCH + REFRESHMENTS	
13:30	Talk P by Q. Rogers	Talk S by T. Usher
14:00	Talk U by V. Willis	Talk X by Y. Zadje
14:30	Talk AA by B. Christian	*No Talk Scheduled*
15:00	—-BREAK—-	
15:15	Talk DD by E. Flemmings	*No Talk Scheduled*
15:15	Talk GG by H. Inya	Talk JJ by K Li-Sho

For a conference that does not have a schedule, the scheduling algorithm will first create a schedule and then evaluate its effectiveness by using a simulating function. This function rapidly simulates delegates (registered to the conference in question) going through the new schedule to see how many are able to attend all of the talks they record a preference for, looking out for *conflicts*. The function will keep a track of the number of delegates that are able to attend 65% of their preferred talks. A newly generated schedule will be accepted as a good solution if at least approximately 75% of delegates are able to attend at least 65% of their preferred talks. For conferences assigned a preexisting schedule, new schedules will still be generated to ensure the most optimised schedule is found; the scheduling algorithm will use the simulation function slightly differently to compare the performance of two schedules. This will be done by using the simulation function to generate a performance score for both schedules - a schedule is awarded a point for every delegate that can attend 65% of their preferred talks. The schedule with the higher performance score will be used as the new schedule.

TABLE 3
An mock example of a derived schedule, for a delegate *(based on the master schedule in table 2)* - the second day in a conference

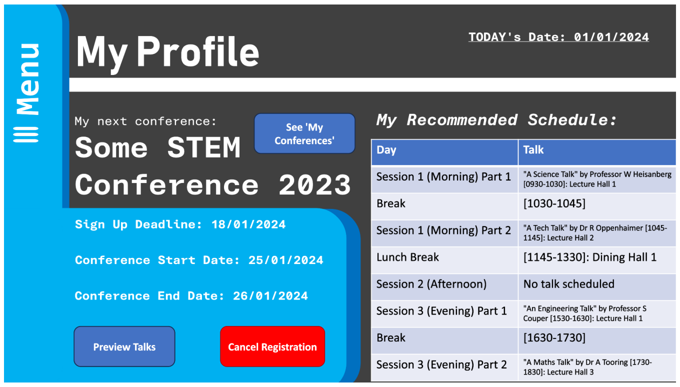
Time Slot	Talk Name	Speaker
11:00	Talk A	B. Charles
11:30	Talk G	H. Islington
12:00	*No Talk Scheduled*	N/A
12:30	LUNCH AND REFRESHMENTS	
13:30	Talk S	T. Usher
14:00	Talk U	V. Willis
14:30	Talk AA	B. Christian
15:00	—-BREAK—-	
15:15	*No Talk Scheduled*	N/A
15:15	Talk JJ	K Li-Sho

3.1.4 Output: User Experience

Upon first entering the web application, the user will be brought to the *login* page obliged to log in to an existing account (within the *login page*), or create a new one (within the *sign-up page*) - both in the style of a HTML form page. Users will need to create accounts on the system, entering information about their name, email, and date of birth. They should create a username, a password and specify their user type - which will then be saved to the web application's database. Upon logging into the system with an account that exists, the user is brought to their account home page (also known as the profile). The profile is a presentation of information for a user's next upcoming or current conference - if they are registered, or it will prompt the user to find one if the user has no upcoming conferences. If a host has created a conference, or a delegate has a registered itself to a conference, the web application will show the details of this conference, including the name, relevant dates, and the schedule in the form of a multi-slot timetable (if it has been created), alongside different buttons that allow navigation the list of talks for the conferences, all conferences associated with the user (a design for a delegate's profile can seen in figure 3), and options to cancel the registration, edit their profile details. delete their account, or search for more conferences - much of these options will kept in a navigation bar to the side or top of the profile page. If any user is associated to multiple conferences, the profile page will retrieve the details of the conference that is nearest in time.

Aside from viewing the profile page, a host can create a profile for a conference they are organising within the conference editor of the web application (importantly they do not create a conference here, but they add their pre-existing conference to the database of the web application). To create this profile, a host will be prompted for the conference information discussed in section 3.1.1 within a different HTML form page, with a save button that write this conference information to the database (within the conferences entity). Following this, another HTML form page will prompt the host for talk information (as discussed in 3.1.2) with the option to add any number of talks; a save talks button will save the talk information to the database within the talks, speakers and topics entities, with association to the correct conference newly created within the conferences entity. Concluding the conference profile creation, the host will be redirected back the profile page with the details of their newly created conference being shown. Where a schedule has not been created, a simple list of saved talks will be presented on the user interface in place of the timetable. A host can edit their conference details at any time.

Since much of the delegate functionalities would stem from being registered to a conference, a new delegate would be empty with a encouraging message for the user to find a conference using the *search* feature (to get to the search page). Entering the search page, the delegate will be met with a search bar and no results. There are two ways to search for a new conference: the delegate can enter a keyword, acronym (it is common for some conferences to have their name derived from a string of words by creating an acronym from each word's initial letter), or any relevant query to find a conference's name. The web application will take this search query and find conference names that are the same as, or similar to the search query (in terms of literal words), and then return any results back to the user on the search page (it is possible for no results to be returned if the database does not recognise the search query string). Alternatively, the user can click the show all conferences button to show a list of all created conferences stored in the database. Upon settling on a conference, the delegate can select the conference within the list of results to register to the conference if they are not already registered (attempting to register to an already registered conference, will redirect the user back the profile page, providing a message to the delegate informing them that they are already registered to that conference). In any case, a new registration will redirect the delegate back to the profile page to view the conference details of whichever conference is next approaching. Once a delegate registers for a new conference, the web application will send an alert message to the user's interface informing that they are now registered to the new conference, while simultaneously recording the registration to the database. Alongside the normal conference information (and schedule if one exists for this conference), a delegate can preview the list of available talks for the conference to indicate their level of preference for each talk, as well as revoke their registration should they desire. Once a delegate records their preference for different talks, the web application will send an alert message to the user's interface informing that



My Profile

TODAY's Date: 01/01/2024

My next conference:
Some STEM Conference 2023

Sign Up Deadline: 18/01/2024
Conference Start Date: 25/01/2024
Conference End Date: 26/01/2024

My Recommended Schedule:

Day	Talk
Session 1 (Morning) Part 1	"A Science Talk" by Professor W Heisanberg [0930-1030]: Lecture Hall 1
Break	[1030-1045]
Session 1 (Morning) Part 2	"A Tech Talk" by Dr R Oppenhaimer [1045-1145]: Lecture Hall 2
Lunch Break	[1145-1330]: Dining Hall 1
Session 2 (Afternoon)	No talk scheduled
Session 3 (Evening) Part 1	"An Engineering Talk" by Professor S Couper [1530-1630]: Lecture Hall 1
Break	[1630-1730]
Session 3 (Evening) Part 2	"A Maths Talk" by Dr A Tooring [1730-1830]: Lecture Hall 3

Fig. 3. A planned design for the Profile Page.

their preferences have been saved, while these preferences are recorded in the database.

3.2 How the Schedulers Work

The scheduler algorithm runs in parallel with the main web application system, so that it can make changes even whilst the web application serves client machines and interacts with users. It follows a process flow as seen in figure 4. It follows the cyclical pattern of retrieving conference information, creating the entire set of time-slots available based on conference information, assigning some slots for refreshments and lunch breaks, running one of the two/three schedulers, evaluate the generated schedule, and save better schedules when necessary, for every conference stored in the database. This set of processes will continue to run whilst the web application is running. Constraints on the scheduler will ensure that each talk is not scheduled more than once, and that each parallel-slot (at any time-slot) will hold one talk at a time. At this first iteration of the project, the preferences of speakers will not be observed here; it will be assumed that speakers in a conference may not be using this web application - at least from the role of a speaker - though this is often unrealistic. It will be the responsibility of the conference host to inform their speakers of the timings once the scheduling algorithm has come up with a schedule for all the talks, and the speakers who will be delivering them. Therefore speakers will not be able to input their timing preferences on when they would prefer to deliver talks. Speakers may still choose to register to the web application as a delegate, if they would like to attend talks presented by other speakers / presented on other topics, but their priority should remain on presenting talks as per the solution generated from the scheduling algorithm. An extension of this project (*to be discussed in the conclusion*) describes how an improved version of this web application might be able to handle speaker preferences. Talk attendance will not be the main priority of the algorithm. The main priority of the scheduling algorithm will be to minimise the number conflicts that arise in the schedule.

Each scheduling approach will use the same input data. The initial input is a dictionary containing time-slots. To create this dictionary, the algorithm first establishes another dictionary to represent each day of the conference, allowing the algorithm to organize each day's schedule. This

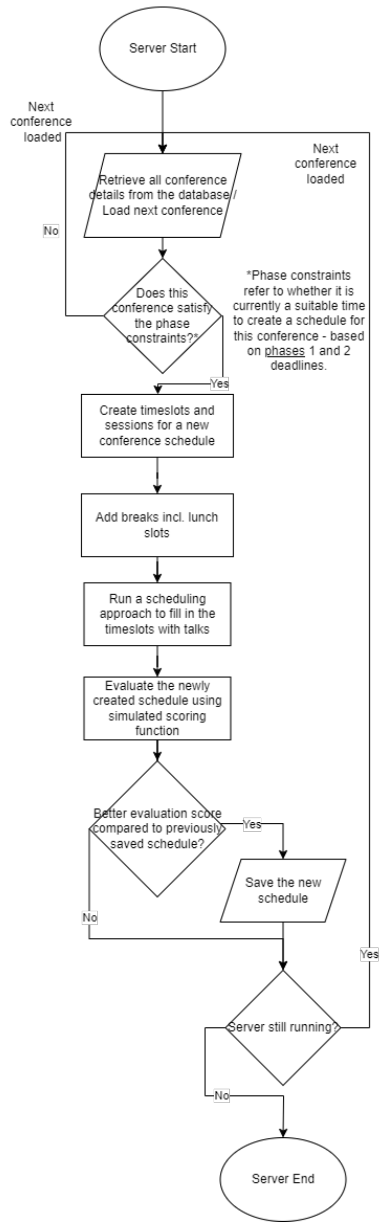

Server Start
Next conference loaded
Retrieve all conference details from the database / Load next conference
No
Does this conference satisfy the phase constraints?*
*Phase constraints refer to whether it is currently a suitable time to create a schedule for this conference - based on phases 1 and 2 deadlines.
Yes
Create timeslots and sessions for a new conference schedule
Add breaks incl. lunch slots
Run a scheduling approach to fill in the timeslots with talks
Evaluate the newly created schedule using simulated scoring function
Better evaluation score compared to previously saved schedule?
Yes
Save the new schedule
No
Yes
Server still running?
No
Server End

Fig. 4. A Flowchart describing the process flow of the scheduler algorithm.

inner dictionary, referred to as the ”agenda” for any given day, initially has empty lists as values to accommodate talks. The keys of this inner dictionary represent estimated timestamps, calculated by counting time intervals starting from the conference’s designated start time for each day. These intervals are determined by the length of each talk (specified during conference creation) and the number of talks per session. For instance, if each talk is 10 minutes long and a session consists of 4 talks, the algorithm starts at 9:00 AM (the designated start time) and increments by 10 minutes four times to create the timestamps for each session. After these slots are created, the algorithm inserts a 15-minute break unless the timestamp reaches 12:00 PM, in which case a 1-hour lunch break is inserted instead. Each timestamp becomes a key in the daily agenda, with empty lists as default values, ready to be filled with talks once a scheduling approach is applied. Figure 5 illustrates an example of these empty time-slots.

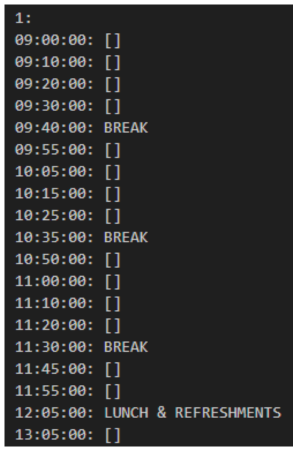

```
1:
09:00:00: []
09:10:00: []
09:20:00: []
09:30:00: []
09:40:00: BREAK
09:55:00: []
10:05:00: []
10:15:00: []
10:25:00: []
10:35:00: BREAK
10:50:00: []
11:00:00: []
11:10:00: []
11:20:00: []
11:30:00: BREAK
11:45:00: []
11:55:00: []
12:05:00: LUNCH & REFRESHMENTS
13:05:00: []
```

Fig. 5. An image example of the empty time-slots created before using a scheduling approach.

Following this, the scheduling algorithm will also retrieve all talk data related to a conference, including database identification numbers, names, speaker details, and associated topic details (also containing data identification numbers). It will also contain its average preference rating amongst all delegates in a conference, and a binary variable to distinguish whether the talk was popular or not; a talk is popular if its average preference rating is at a score of 7 or above. All talk data will be compiled into a multi-dimensional list. The final input will be a set of records that describe the delegate preference ratings for each talk stored in a multi-dimensional list - hereby referred to as the set of ratings.

Once one of the below scheduling approaches has created a schedule, the algorithm will use an evaluation-simulator function to simulate delegates participating in the conference, and going to talks as per the schedule (as recorded in the database for the conference in question). The evaluation-simulator will simulate whether a delegate is able to make at least 65% of their preferred talks (those they rated with a preference score higher than 6 out of 10). A score is generated (as mentioned in section 3.1.3) and compared to any existing schedule and its corresponding score to observe which is most optimal. The most optimal schedule is then saved into the database (overwriting the previous record of the most optimal schedule) so that it might be used when then conference finally starts.

In order to assign talks to slots, the scheduling algorithm chooses from one of the following two scheduling approaches:

3.2.1 Type A: The Standard Scheduler

The Standard scheduler is what is expected to be the main scheduler of choice as the web application is running, compared to the genetic algorithm that is considered to be a much more advanced algorithm. The standard scheduler uses integer programming techniques in conjunction with a weighted-graph visualisation to produce efficiently allocate the different talks to all the available time-slots. Because CSP has been established as $NP - Hard$, it can be justified that the hybrid-usage of integer programming and graph theory make for a valid heuristic approach to solve the problem and create optimal schedules. The Integer Programming (IP) model is defined with decision variables, an objective functions and some baseline, independent constraints. Let x_{dstp} be a binary decision variable representing whether, on day d at time-slot s, talk t is scheduled in parallel slot p.

$$x_{dstp} = 1$$

If talk t is scheduled, at time-slot s, parallel-slot p, on day d.

$$x_{dstp} = 0 \textit{ otherwise.}$$

The objective function is defined:

$$\textit{Maximise } Z = \sum_{d}\sum_{s}\sum_{y}\sum_{t}\sum_{p} x_{dstp} \cdot \text{preferenceScore}(y)$$
$$-\text{Penalty(t, t')}$$

Where Z represents assignment of all talks, and *preferenceScore(y)* is the preference score that delegate y gave talk t. The model is subject to the following constraints:

1) $\sum_d \sum_s \sum_p x_{dstp} = 1$, for all talks t.
   This constraint ensures every talk is scheduled exactly once in a schedule.
2) $\sum_t x_{dstp} \leq 1$, for all time-slots s, parallel sessions p, and days d.
   This constraint ensures that every parallel-slot, within each time-slot each day, holds at most one talk.
3) The Penalty function induces a value loss on the objective function any time talk t is scheduled within close temporal distance the previous most popular talk t', where t' is constantly updated to the minority proportion of talks that have the highest preference ratings according to delegates.

$$\text{Penalty}(t, t') = \frac{1}{\text{distance}(t,\, t')}$$

Built using the python module PuLP, this model works by creating a value formula for scheduling talks by processing delegate preferences. The idea of an optimised conference schedule, within the context of this project, is a schedule that maximises the satisfaction of the delegates that attend, the attendance of delegates in different talks, and networking opportunities while balancing diverse interests. These criteria were defined by observing the user experience of a delegate as they attend a conference. Some delegates that attend a conference may have a specific idea of the themes and topics they wish to engage, whilst others may not. For delegates belonging to the former category, it was thought they would be better satisfied if they were able to engage with the talks and activities that catered to their interests and preferences. Furthermore, the model clusters groups of such delegates together, by identifying talks with overlapping themes or topics, to facilitate networking opportunities by bringing together delegates who share common interests, thereby encouraging discussions, exchanges of ideas, and collaboration among like-minded individuals. Extending this, the model aims to cater to those delegates without a specific theme-motivation in mind by ensuring diversity and inclusivity by incorporating all talks into a schedule. This diversity not only enriches the conference experience for delegates but also fosters a vibrant and inclusive community where individuals from different backgrounds and disciplines can connect and engage with each other, which may help to improve the overall satisfaction felt by the attendees of a conference. It is preferred that the popular talks in a conference are scheduled away from each other in a conference; logistically it would be less optimised for all the popular talks to be scheduled in close proximity to each other. Therefore as the model works to solve the objective function, a penalty will be induced if it tries to schedule popular talks in neighbouring time slots (or in close proximity). For any popular two talks t_1 and t_2, the weight of the penalty is inversely proportional the temporal distance between these two talks. In this way, the model seeks to optimize session attendance by scheduling popular talks or sessions at times when they are likely to attract the highest number of attendees. By inferring and analyzing popularity from delegate preferences, the model attempts to find optimal temporal spacing for scheduling high-demand sessions, thereby maximizing engagement.

After the IP model creates a schedule, the standard scheduler models the talks using a graph visualisation, before then applying graph theory techniques and concepts to further improve the solution. A graph $G = (V, E)$ is constructed for all the talks in a conference, where V is the set of nodes in the graph, representing scheduled talks, and E is the set of edges in the graph that are established based on delegate preferences. If two talks are preferred by the same delegate(s), an edge is added between the corresponding vertices in the graph. This creates a connection between talks that share common preferences, indicating a potential relationship or similarity in content as perceived by the delegates. Delegate preferences play a central role in shaping the graph. The scheduler considers preferences expressed by delegates for certain talks and establishes connections between talks based on shared preferences. Once the graph is constructed and weighted, the scheduler applies a graph coloring algorithm to assign colors to different talks in a way that adjacent talks do not share the same color. An example of a graph representing delegate preferences of 20 talks can be observed in figure 6. The weights for edges

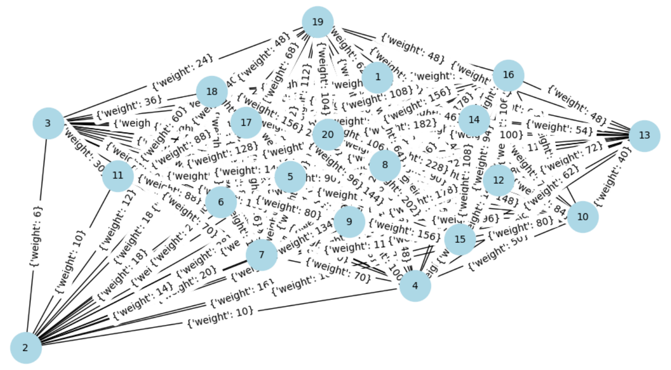


Fig. 6. A Weighted Graph of talks (as nodes), where each connecting edge represents the number of delegates that want to see those two talks. Generated with Python.

in the graph will allow the standard scheduler to identify and highlight the talks with the highest and lowest levels of interest from delegates. Within the python implementation, the standard scheduler uses a greedy colouring algorithm (originally based on work from Kosowski and Manuszewski [9] to produce a solution to the *Graph n-colouring problem*, implemented within the Python 'NetworkX' module [10]) to assign colours to nodes based on the weights between different nodes. By using the resulting colours of different graph nodes, the standard scheduler reorganises the solution obtained from the IP model (where necessary) to ensure that talks with the same color are not scheduled in parallel. This may involve adjusting the time slots of certain talks to avoid conflicts and optimise the overall schedule.

3.2.2 Type B: The Genetic Scheduler

The Genetic scheduler is an advanced scheduling approach compared to the Standard scheduler but would work in conjunction with the same web application. Genetic algorithms are meta-heuristic processes (with a big focus on data features) based on the Charles Darwin's theorems of natural selection and evolutionary optimisation. Genetic algorithms highlight and investigate which individuals in each generation of population have the required optimal genes that tend towards the most optimal solution. The main stages in creating an genetic algorihm involve, *Initialisation*, *Fitness*, *Selection*, *Crossover*, and *Mutation*.

The *initialisation* phase involves creating a set of random *individual* solutions, called a *population*, as described by Mallawaarachchi [13]. For any conference using this scheduling approach, an individual would be a schedule that contained all the talks in a conference in different time-slots. Mallawaarachchi goes onto describe that an individual is made up of variable parameters, called *genes*, which can be joined in a string known as a chromosome.

In this context, genes might take the form of talks and their assignment in schedules - in list of schedule time-slots, two different schedules (individuals) might have a differing gene if they schedule a different talk in the same time-slot. Using this structure, many individuals are generated randomly, in an initial generation, and are fed through the genetic selection process. More individuals are created as the genetic selection and evolution process takes place; the aim the genetic algorithm is create individuals that are prove to be increasingly more effective at solving the problem, until the most optimal solution is found - after creating x generations of individuals (for some number of generations x).

Before *offspring* or mutations occur, the *fitness* of different individuals, with respect to the aims and objectives of the problem, is measured. Kumar [14] describe fitness scoring to show the ability of an individual to compete for mating with other individuals (crossing-over with other solutions) to create better individuals in a subsequent generation. In this context, the fitness function will be designed to use the evaluation-simulator function to quantify how well an individual might perform, as a proper conference schedule.

Once fitness scores have been calculated, the *selection* stage involves choosing individuals from the current population to serve as parents for the next generation. The selection process favors individuals with higher fitness scores, increasing the likelihood of propagating favorable traits to the next generation. The selection mechanism behaves like a tournament selection to choose parents based on their fitness scores. Schedules with higher fitness scores have a higher probability of being selected as parents.

The *crossover* stage involves combining genetic material from two parent individuals (schedules) to produce offspring with characteristics inherited from both parents. Crossover promotes genetic diversity and exploration of the solution space. In conference scheduling, crossover typically involves selecting random points in the parent schedules and swapping genetic information between them to create offspring - using crossover points to extract different genes from each parent. This process combines elements of both parents' schedules to produce potentially better solutions.

The *mutation* stage introduces random changes to individual schedules to maintain genetic diversity and prevent premature convergence to sub-optimal solutions. Mutation operators involve random assignment changes such as swapping talks between time-slots, or moving talks to different spaces in the schedule. These changes help explore new regions of the search space and potentially improve the quality of solutions. Kumar [14] explains the key idea of inserting random genes in offspring to maintain diversity in the population and avoid premature convergence. There will be a 10% for any individual schedule to mutate.

There are a number of differences between the nature of the Standard Scheduler and the Genetic Scheduler, but the most prominent difference is related to one of the constraints applied to both scheduling approaches. Since the Genetic scheduler can only prioritise one constraint as part of its fitness functionality, it may schedule the same talk more than once in a schedule in order to ensure as many delegates are satisfied as possible, compared to the Standard Scheduler which can be forced to schedule every talk only once. This is because genetic algorithms operate on using probability based changes, which often will not account for the pre-existence of any gene in the individual.

Upon completing the genetic process for a number of generations, the Genetic scheduler cycles through the final population to find the individual with the best fitness, which in this case (for any conference) refers to the schedule that is evaluated to predict the highest proportion of delegates that are able to attend at least 65% of their chosen talks within said schedule - as calculated by the evaluation-simulator function.

4 Results

The following section describes the results of the project; it demonstrates how the final solution works (using both scheduling approaches) on two different test conferences. After the two initial conferences, the web application was test on several further test conferences (with randomly generated delegate data).

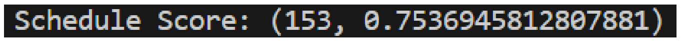
Upon running the server, a local-host link was generated for which the web application was accessed. The web application server log had also reported that the server had been launched and database initialised. A new account was created under the username "TheFinalHost" with the name of this paper's writer, fake email, and fake date of birth (see Appendix 1). Another new account under the username ”TheFinalDeleagte” was also created; the server log successfully recognised both account beings created. Logged in the ”TheFinalHost”, a set of test data was created in the form of a fictitious conference based on the successful STOC 2023 conference [12] by Saha and Servedio et al. The STOC 2023 conference was held in Orlando, Florida in the United States and engaged in topics regarding the theory of computing, algorithms, data structures, graph theory, combinatorics, and theoretical computer science (as well as many others). The testing conference ”STOC 2024” was created out of inspiration (the full details of which can be seen in Appendix 2), using a custom date for the paper submission deadline, but deriving the registration deadline date, and conference dates before pushing them forward a year as if they would take place in the year 2024. These dates were used as the deadline dates for each of the four conference phases: *Paper Review / Talk Finalisation Date*, *Delegate Registration Deadline*, *Conference Start Date*, and *Conference End Date*. The initial conference details were then finalised by setting the number of talks per session to be four, the average talk length to be fifteen minutes, and the maximum number of parallel sessions (slots) at any one time to be 3. After this, forty of the talks from STOC 2023 were added to the profile of ”STOC 2024”, where the names of papers were kept the same, where only one speaker was provided. The topics of each talk were based on keywords and phrases present in the title of the talk; some of the talks added to the database were given common topic names. All of this new conference data was saved to the web application’s database. Upon finishing all of the data entry, the host was redirected back to the dashboard page being able to see all the new conference data presented to them in a user interface very close in resemblance to the planned screen designs.

Upon confirming that the server acknowledged the creation of the new conference, the scheduler algorithm began to start retrieving data for the conference. It was prepared to create a schedule for this conference since it could seamless and accurately retrieval of all relevant conference information from the database, but noticed that there was no delegate information with which it could process. Whilst the scheduler algorithm acknowledged this, a small piece of code was written to generate two hundred random delegates (each with random, but consistent strings of characters for usernames, first-names, passwords and emails) and save them to the database - akin to an benign server-side SQL injection completely safe within a development environment. For each delegate, random preference scores were created for each talk, as a way of simulating delegate x rating talk y with a (random) score of z, and were saved to the database, associated with the conference STOC 2024, allowing the scheduler algorithm to start creating schedules.

The web application and scheduling algorithm were able to operate in parallel, as intended. The web application continued to serve the Host user through profile changes (through the profile-editing section) whilst creating the three main scheduling data structures - those described in section 3.2. Once the schedule timing slot data, talk information and delegate preference information was created, the server log denoted the scheduling algorithm starting its procedures on the STOC 2024 conference. The scheduling approach of choice for this conference was the Standard Scheduler, which started by creating each of the 10,440 decision variables the objective function, and constraints for the integer programming model. It took approximately 23 seconds for the model to be solved. The results of the model were in the form of assigning values of either 0 or 1 to the different decision variables, aiming to satisfy the constraints while maximizing the objective function. The resulting schedule was derived from the assignment of values to the decision variables. Each decision variable assigned a value of 1 indicates a scheduled talk. This assignment allowed for the identification of the specific day, time-slot, and parallel slot allocated to each talk; the server log reflected the creation of this preliminary schedule.

The schedule was then passed onto the graph visualisation implementation. The standard scheduler first created a graph data structure - following the description of a schedule graph mentioned in section 3.2.1 - resembling the which talks in the conference are popular and share similar levels of interest. After applying the greedy colouring algorithm to this graph, the resulting ”colouring” was used to judge which talks could or could not be scheduled together. The colouring result was used on the IP model’s preliminary solution to change the slots of talks that held the same colour within the coloured graph, refining the level of satisfaction the schedule produced.

The schedule was subsequently forwarded to the evaluation function. By simulating the experience of each randomly created delegate as they navigated through the schedule, a satisfaction score of 75.3% was computed. This score indicates that approximately 75.3% of delegates (153 delegates in total) participated in at least 65%, or more, of their preferred talks during the simulation (as seen in figure 7). Finally, the standard scheduler completed its task and saved the finalized schedule into a text file, and a reference to this file was stored in the web application’s database. Although there was not actually a previous schedule for STOC 2024, this action demonstrates that the scheduling algorithm is programmed to overwrite previous data if available or create a new record when necessary. It was noticed through the difference in the IP model solution and the final schedule, seen in the server log, that some talks were initially scheduled in parallel within the same time-slot but were then moved to be scheduled in adjacent time-slots.



```
Schedule Score: (153, 0.7536945812807881)
```

Fig. 7. The results from the evaluation score function on the ”test-use” STOC2024 conference and the schedule generated by the Standard Scheduler.

Finally after a few seconds taken to save the new schedule, it was loaded into the Host user’s dashboard allowing them to view the schedule generated in a multi-session view (see Appendix x).

To test the Genetic Scheduler, a new conference was created entirely from scratch (under a different host user account) over the course of four days. It was created with fewer talks than ’STOC 2024’, but had an increased number of time slots. During this testing, it was named ’METALCON 2024’ and had fictitious details regarding its deadlines for different conference phases, the number of talks per session, the average length of a talk and the number of parallel sessions happening during any time-slot. Like before, the web application and scheduling algorithm were able to operate in parallel. The web application continued to serve the Host user through conference changes for ’METAL-CON 2024’ whilst creating the three main scheduling data structures. Once the schedule timing slot data, talk information and delegate preference information were created, the server log denoted the scheduling algorithm starting its procedures on the ’METAL-CON 2024’ conference. It was given to the Genetic Scheduler to produce a schedule, which had been programmed to process populations of 75 individuals in each generation, for a total of 25 generations. It proceeded to take several minutes to go through the stages of selecting parents, creating offspring via gene crossover, and refining a new population through mutation. The algorithm had finished executing, describing the individual in the final population that had the best fitness, which provided the most optimal schedule for the conference. This schedule was a viable solution for the ’METAL-CON 2024’ conference, boasting a delegate satisfaction score of 100% - which was believed to be incorrect at first, but then verified by running the conference data again through the genetic algorithm multiple times and observing the process of calculating the delegate satisfaction score. The two schedulers had different use cases depending on the host user’s preferences and time availability. If a host user wanted to space out their conference talks across an extended period, they may benefit from using the Standard Scheduler. However, if a host user had a higher number of talks and less time availability, they may benefit from using the more advanced Genetic Scheduler (though it would take longer to produce a schedule). Both scheduling approaches were able to schedule conferences that lasted a single day or multiple days. It was also observed that both schedulers generated different schedules for different conferences, as talks were removed or delegate scores changed.

The rest of the appendix displays the various application pages that were finally implemented, as well as some charts showing scheduler results.

5 EVALUATION

Upon completion of the project, it was straightforward to evaluate the aspects of the project that were related to the functional processes involved in retrieving user data, producing information output and processing schedules. However, it was more difficult to evaluate the non-functional aspects of the project, such as the effectiveness of the user interface, the reliability of the web application’s framework, or the general robustness of the system. An evaluation of the functional aspects of the project will involve a check-list comparison between the original mandatory and advanced requirements and the delivered functionality. Evaluating the non-functional requirements will be done using a critical analysis and Nielsen’s 10 Usability Heuristics for Interface Design [15].

5.1 Functional Evaluation

The following text describes the successful implementation of a web application where users can create accounts and conference profiles. The application offers all normal account creation, edit and removal options required for the two different user types, ”host” and ”delegate”. The database supporting the web application was successfully implemented and could facilitate all standard database operations, including create, read, update and delete (CRUD), on all the information provided to it. The functionality was verified through numerous tests (akin to those described in the results section) and a database viewing interface, which showed that user data, conference data, talk data and schedule data could be easily created, edited and changed without any issues.

The Standard Scheduler implemented and produced good results but did not fully adhere to the model described in section 3.2.1. During development, it became technically troublesome to implement the penalty constraint of the model using PuLP, so it was omitted from the final model. Despite this, the Standard Scheduler produced good results that satisfied the baseline success benchmark defined in section 3.1.3. The integer programming model and graph theory techniques worked in tandem to produce successful working schedules - the scores generated for several test conferences can be seen in Appendix x. The Genetic Scheduler was also successfully implemented (exactly to its specification in section 3.2.2), and for the same set of test conferences, it produced better quality schedules - where much higher proportions of delegates were able to attend at least 65% of their preferred talks - but took far longer to create each schedule. For some of the conferences, it was noticed that the Standard Scheduler tended to produce schedules that increased the space between all talks as evenly as possible, across all available time-slots; the Genetic Scheduler instead created schedules where talks were skewed to a certain time frame in a conference. This suggested some different use cases for both schedulers: if a host user preferred to space out their conference talks across an extended period (such that the ratio of the number of talks to the number of available time-slots is quite large), they may benefit from using the baseline-proficient Standard Scheduler; if a host user was far more limited in time availability, with a higher number of talks (such that the ratio of the number of talks to the number of available time-slots is smaller) with enough talk locations available, they may benefit from using the more advanced Genetic Scheduler instead (though it might take far longer to produce a schedule). It was noticed that both scheduling approaches would generate different schedules for changing conferences (as different talks were removed, or as delegate scores changed).

However, there are some features that could have provided improved integrity and error-prevention, which were not implemented in the final solution. Whilst the user authentication was robust and secure, strict database record locking for conference data was not implemented. This is because the system was design that only one user could amend any record at any one time. Despite this, record locking could reinforce data integrity to reduce any the data risks of any rarely-occurring where multiple users write to any single record in the database.

5.2 Non-Functional Evaluation

It was clear to see that both schedulers had no issue scheduling conferences that lasted a single day or multiple days in length. This was verified in the web application's dashboard; by logging as both a host and a delegate, the user is able to view the current day's schedule but also tell the application to look at the schedule for a different day in the conference, adequately pulling the correct schedule data and presenting it in a easy-to-understand form.

To assess the user interface, the application was assessed by the Nielsen's 10 Usability Heuristics for User Interface Design [15]. It performed well in response to guidelines concerning:

- *"Visibility of System Status"* - The web application's design keeps the user informed about what they are doing, changes they are making, and how they are moving through the system, using good descriptions, various on-screen notifications and warnings.
- *"Match Between the System and the Real World"* - The terminology used in application will be familiar to conference organisers. And for the lesser known terms, the solution establishes mental models and provides some descriptions of meaning to some of its terms - like explaining the different conference phase meanings.
- *"Consistency and Standards"*
- *"Recognition and Recall"* - The dashboard presents a simple view of all conference timings, and phase details. All previously saved conference information is loaded into the edit-conference page to remind the user of the data they had provided beforehand, allowing them to recognise their conferences instead of purely remembering them.
- *"Help and Documentation"*

It could have improved at providing a response to:

- *"User Control and Freedom"* - While an "undo" or "cancel" option is supported in the system, by not aggressively saving changes once they are made - until they are confirmed - they are not obvious in the system.
- *"Error Prevention"* - The application has rigorously tested for a selection of scenarios with the goal of consistent usability and minimal failure, there is a possibility for the application to break at certain errors specifically during back-end schedule processing (if any data is not formatted correctly).
- *"Flexibility and Efficiency of Use"*
- *"Aesthetic and Minimalist Design"*
- *"Help Users Recognise, Diagnose, and Recover from Errors"*

The only non-functional requirement that was not satisfied, was the inclusion of a recommender system, to run in parallel within the web application, that provided conference suggestions to delegates, based on the previous history of conferences that said delegate participated in, or conferences they were currently registered to. This would have improved the user experience of finding conferences, making it more intelligent, but also easier and more accessible.

5.3 Some Possible Improvements

Overall the project was successful in answering the research question, as it demonstrates a the design of a web application built to assist in conference hosts in planning the schedules for their conference, by providing a platform to gain delegate information, and leveraging powerful computing-based heuristic methods to create useful and optimised schedules. However some other improvements could have been made to either scheduling approach to better refine their quality, and expand the impact of their usefulness. Both schedulers could have benefited from the use of speaker preferences and availability when processing talk information, and assigning talks to time-slots. It is common for talks to be established with specific speakers in mind, and it is equally common for a speaker to present multiple talks. Therefore, speakers would benefit greatly from having a schedule that accommodated for their availability to present talks, and their preferences for talks they would desire to see. Finally another improvement for both schedules could be a method to ensure maximum attendance in generated schedules; this could be accomplished using maximum-flow calculations. Maximising the attendance of talks in a schedule may indirectly help to create a even distribution of attendance for all activities, whilst further improving a delegate's satisfaction through better engagement with conference content.

6 Conclusion

This paper has successfully delved into the exploration of computer science methodologies for optimizing the generation of conference schedules, with the aim of enhancing the experience for both organizers and attendees. The integration of a user interface proved to be instrumental, facilitating the seamless creation of conference profiles. Through this interface, hosts and delegates could effortlessly input data, subsequently yielding an optimized schedule courtesy of the scheduling algorithm. While some issues were apparent, our findings demonstrated the effectiveness of the heuristic genetic scheduler and the efficiency of IP modelling and graph theory based scheduler. Incorporating this web application with the IP modelling and graph theory based scheduling approach could greatly benefit the planning of realistic conferences, streamlining the conference planning processes and maximising the quality of time utilised during such events, but could potentially miss out on producing better quality results compared to using a genetic algorithm (which would be slower in comparison). Moving forward, there are several potential avenues for extending this project. One possibility is providing host users with the ability to override specific parts of schedules could allow for greater flexibility and customisation, accommodating contextually specific needs. Furthermore, there is potential for commercial release of this application on live servers, enabling multiple organisations to utilise the web application simultaneously, thereby amplifying its impact and effect.

REFERENCES

[1] J Quesnelle, and D Steffy ”Scheduling a conference to minimize attendee preference conflicts” in *7th Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2015)*. Prague, Czech Republic. 2015.

[2] B Vangerven, et al. ”Conference scheduling - A personalized approach” in *“Proceedings of the 11th International Conference on Practice and Theory of Automated Timetabling (PATAT-2016)”*. vol 81. issue 0305-0483. pp38-47. Dec 2016.

[3] E. Castro, S. Petrovic, “Combined mathematical programming and heuristics for a radiotherapy pre-treatment scheduling problem”, *Journal of Scheduling*, 15, 2012, pp. 333–346, https://doi.org/10.1007/s10951-011-0239-8

[4] H. K. Chan, S. H. Chung., “Optimisation approaches for distributed scheduling problems” in International Journal of Production Research, 51, 2013, pp. 2571-2577. https://doi.org/10.1080/00207543.2012.755345

[5] Python. *Python*. (Version 3.8.8). Accessed: Oct 2023. [Online]. Available: https://www.python.org/downloads/release/python-388/

[6] Flask. *Flask Documentation (Version 3.0.2)*. (n.d.), Accessed: Nov 2023. [Online]. Available: https://flask.palletsprojects.com/en/3.0.x/

[7] A Sharif. (2024. Jan. 8) ”What is CRUD?”. *Crowdstrike*. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/observability/crud/

[8] SQLAlchemy. *SQLAlchemy (Version 2.0.29)*. (n.d.). [Online]. Available: https://www.sqlalchemy.org/, https://pypi.org/project/SQLAlchemy/ [Accessed: Jan, 2024].

[9] A Kosowski, K Manuszewski. ”Classical Coloring of Graphs” in *Graph Colorings*. pp2-19. 2004. ISBN 0-8218-3458-4.

[10] A A Hagberg, D A Schult, P J Swart. “Exploring network structure, dynamics, and function using NetworkX” in *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA. 2008. pp. 11–15.

[11] W J Cook, W H Cunningham, W R Pulleyblank, and A Schrijver. ”NP and NP-Completeness”. in *Combinatorial Optimization*, 1st Edition. R L Graham, J K Lenstra, R E Tarjan. Eds. New York. Wiley-Interscience Publications. 1997. pp309-325.

[12] B Saha, R Servedio et al. (2023, June. 20-23). *STOC 2023: 55th Annual ACM Symposium on Theory of Computing June 20-23, 2023 in Orlando, Florida*. [Online]. Available: http://acm-stoc.org/stoc2023/

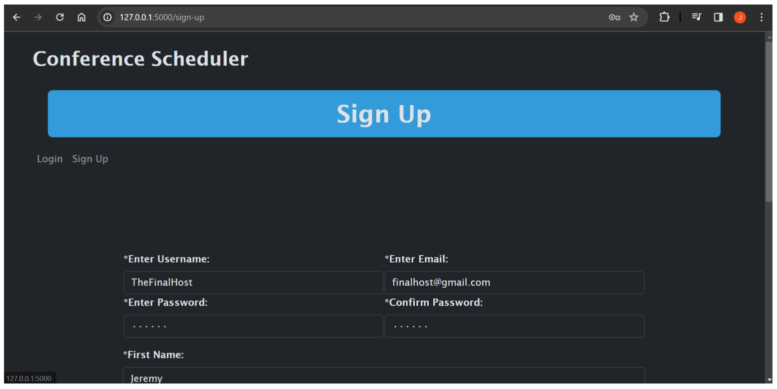
[13] V Mallawaarachchi. (2017, Jul. 8) ”Introduction to Genetic Algorithms — Including Example Code”. *Medium*. [Online]. Available: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[14] A Kumar. (2023, Mar. 8) ”Genetic Algorithms”, *Geeks for Geeks*. [Online]. Available: https://www.geeksforgeeks.org/genetic-algorithms/

[15] J Nielsen. ”Usability Heuristics for User Interface Design,” *Nielsen Norman Group*. [Online]. Available: https://www.nngroup.com/articles/ten-usability-heuristics/. [Accessed: April 12, 2024].

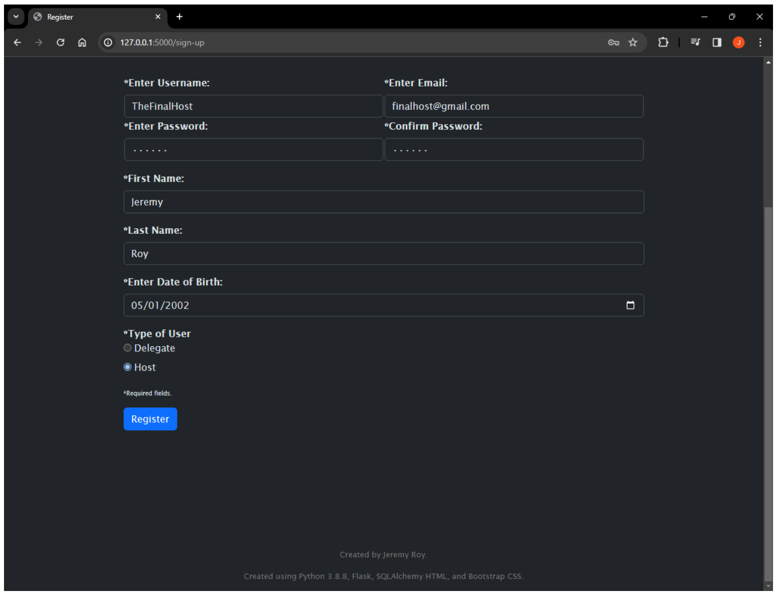
7 APPENDIX (INCOMPLETE)

7.1 Appendix 1: Host Account Creation



*Enter Username: TheFinalHost
*Enter Email: finalhost@gmail.com
*Enter Password: ······
*Confirm Password: ······
*First Name: Jeremy

Fig. 8. Part a) Creating a new account in the Sign Up page

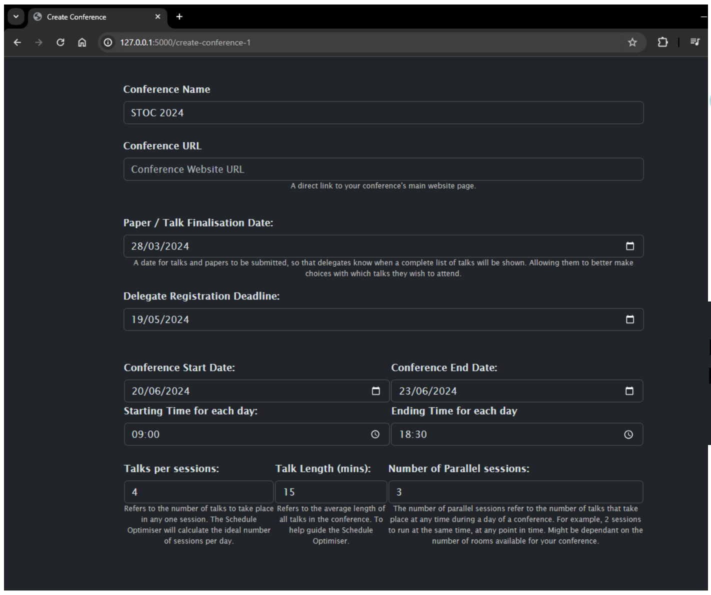


*Enter Username: TheFinalHost
*Enter Email: finalhost@gmail.com
*Enter Password: ······
*Confirm Password: ······
*First Name: Jeremy
*Last Name: Roy
*Enter Date of Birth: 05/01/2002
*Type of User
Delegate [✓ Host]
*Required fields.

Created by Jeremy Roy.
Created using Python 3.8.8, Flask, SQLAlchemy HTML, and Bootstrap CSS.

Fig. 9. Part b) Creating a new account in the Sign Up page

7.2 Appendix 2: STOC 2024 Creation



Conference Name

STOC 2024

Conference URL

Conference Website URL

A direct link to your conference's main website page.

Paper / Talk Finalisation Date:

28/03/2024

A date for talks and papers to be submitted, so that delegates know when a complete list of talks will be shown. Allowing them to better make choices with which talks they wish to attend.

Delegate Registration Deadline:

19/05/2024

Conference Start Date: 20/06/2024

Conference End Date: 23/06/2024

Starting Time for each day: 09:00

Ending Time for each day 18:30

Talks per sessions: 4

Refers to the number of talks to take place in any one session. The Schedule Optimiser will calculate the ideal number of sessions per day.

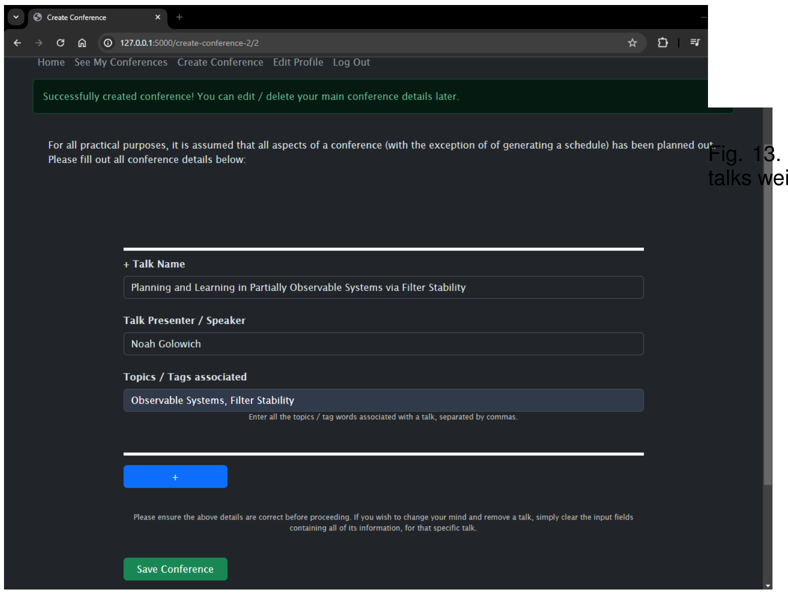
Talk Length (mins): 15

Refers to the average length of all talks in the conference. To help guide the Schedule Optimiser.

Number of Parallel sessions: 3

The number of parallel sessions refer to the number of talks that take place at any time during a day of a conference. For example, 2 sessions to run at the same time, at any point in time. Might be dependant on the number of rooms available for your conference.

Fig. 10. Part a) Creating a new conference based on STOC 2024 [12]



Successfully created conference! You can edit / delete your main conference details later.

For all practical purposes, it is assumed that all aspects of a conference (with the exception of of generating a schedule) has been planned out. Please fill out all conference details below:

+ Talk Name

Planning and Learning in Partially Observable Systems via Filter Stability

Talk Presenter / Speaker

Noah Golowich

Topics / Tags associated

Observable Systems, Filter Stability

Enter all the topics / tag words associated with a talk, separated by commas.

Please ensure the above details are correct before proceeding. If you wish to change your mind and remove a talk, simply clear the input fields containing all of its information, for that specific talk.

Fig. 11. Part b) Creating a new conference based on STOC 2024 [12]

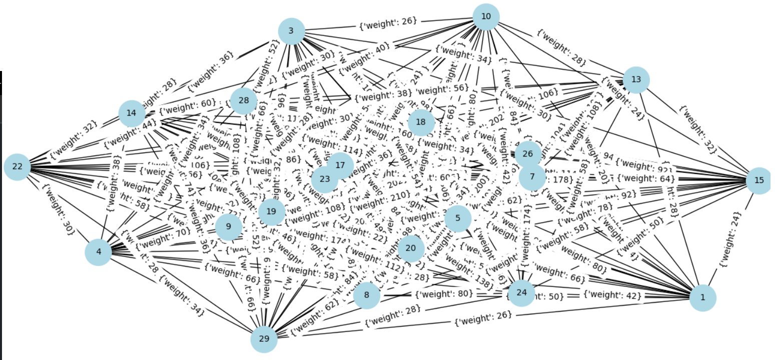


Fig. 12. Part A) A graph of conference talks weighted with delegate preferences for a test conference.

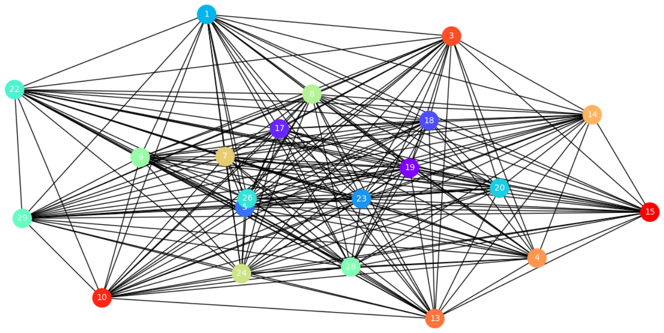


Fig. 13. Part B) A colouring applied to the same graph of conference talks weighted with delegate preferences.