Android-Based Multi-Constraint Intelligent Scheduling for Amateur Sports Competitions

Submitted April 2025, in partial fulfilment of

the conditions for the award of the degree **BSc (Hons) Computer Science with Artificial Intelligence.**

20411175

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____GQY_____

Date __15__/__04__/__2025___

Word Count: 10840

Abstract

This project aims to provide an intelligent scheduling system to solve the problems of low efficiency and error-prone manual scheduling for amateur sports competitions. Compared with professional leagues, amateur sports often face the challenges of limited resources, uncertainty of the team number and available time, lack of advanced scheduling tools, etc. Therefore, an optimising Android app on mobile devices is implemented through this project combined with the simulated annealing algorithm to generate automatic solutions with high quality on the basis of satisfying multiple scheduling constraints. The system enables users to input the number of teams, venues, time slots, duration, and constraint preferences, improving user satisfaction and experience. The scheduling process includes initial solution generation and constraint-driven iterative optimisation. The objective function is evaluated and designed according to various soft constraints with a weighted penalty coefficient. In addition, the project passes all the tests for the pilot scenario and randomly generated scenarios, indicating its stability and extension. This study not only provides a convenient tool for grassroots event organisers but also has certain research and application value in the realization of mobile terminal optimization algorithms. Furthermore, the algorithm consistently generated high-quality feasible schedules within seconds, significantly outperforming manual scheduling in both speed and constraint satisfaction.

Contents

1 Introduction

This project explores the use of intelligent optimisation techniques, particularly simulated annealing, to generate efficient schedules for amateur sports scheduling problems. Not only does it guarantee fairness and optimal resource allocation, but it also tries to meet each constraint as much as possible judging by the objective function. It ensures a reasonable app reaction time while generating the best possible solution. The following sections provide background, motivation, aims, and the key objectives of the project.

1.1 Background

Amateur sports competitions are widely embraced across various groups, including educational institutions, industries, and communities. Unlike professional leagues, which are often faced with complex commercial constraints such as high viewer traffic hours and broadcast schedules, amateur sports scheduling is more limited by the availability of resources, for example, venues, time, and participants' preferences [3]. It is of great importance to schedule the event in an efficient way that meets each participant's preference and other constraints using limited resources. In the past, amateur sports were primarily scheduled manually because these events were typically small or medium in size, making it easier for people to organize them. However, as the number of teams and venues increased, along with multiple constraints at the same time, manual scheduling became complex and time-consuming, along with errors caused by humans. So far, automatic scheduling for amateur competitions has received only limited attention, while its occurrence may be far more frequent than that of professional games [10]. Therefore, there is a need for a more efficient way of scheduling. Through this project, these complex situations are handled easily and efficiently with the help of smartphones and intelligent optimization technology.

1.2 Motivation

This project is primarily motivated by the challenges encountered in scheduling amateur sports competitions, such as school leagues and community events. These competitions are closely related to our lives, but there is a lack of effective scheduling tools and platforms, resulting in inefficient manual scheduling and a high rate of constraint conflicts.

- Manual scheduling inefficiencies
  During school and community-level competitions, scheduling is often done by staff or managers with limited technical support. Therefore, there is a high rate of causing inefficient schedules, such as timetable overlapping, unfair distribution of time, and uneven use of the venues.
- Complexity of constraints
  Various constraints should be considered in the case of amateur sports scheduling, consisting of hard constraints and soft constraints. Hard constraints are those conditions that must be met without which the game cannot be played, for example, one team cannot play two games at the same time. Soft constraints can be evaluated using objective functions, which should be met as much as possible, including group constraint, date constraint, separation constraint, rest difference constraint, daily constraint, and team preferences, which will be discussed in detail in 2.3. Non-professional organisers often lack the appropriate tools to balance these constraints, so that manual scheduling often fails to find an optimal solution.
- Growing demand for mobile scheduling tools
  Rather than complex and costly desktop applications, lightweight and intuitive mobile devices increasingly draw the attention of organisers. A mobile app would be more interactive and easier for user input, enhancing accessibility and autonomy in the process.

1.3 Aims and Objectives

The main aim of this project is to develop a mobile app that facilitates the automated scheduling of amateur sports competitions, particularly in school or community-based contexts. The project targets non-professional events where manual scheduling is common due to limited resources, which aims to improve scheduling efficiency and quality and reduce human errors at the same time. Given that the application is deployed on mobile devices, the scheduling algorithm must also deliver results within a reasonable response time to ensure a smooth user experience and prevent delays or system unresponsiveness.

The key objectives of this project are:

- To investigate specific challenges encountered in scheduling amateur sports competitions, including limited tools, multiple constraints and scheduling efficiency.
- To conduct user research with typical users, such as event organisers, school teachers, and managers, to identify user requirements and necessary features.
- To design an intuitive and accessible user interface that enables a convenient way of inputting information about teams, venues, time, and constraints.
- To implement a scheduling algorithm based on heuristic search, for example, Simulated Annealing, that can effectively handle constraints and provide an optimal solution with a reasonable response time.
- To develop the app on the Android platform, ensuring compatibility with common smartphones and smooth user interaction.
- To evaluate the scheduling algorithm's effectiveness and efficiency through testing with the pilot problem and randomly generated problems.
- To collect and analyse user feedback to improve the interaction experience, algorithm performance, and user satisfaction.

The remainder of this dissertation is structured as follows. Section 2 describes a pilot problem scenario, possible constraints, and the evaluation methods of the result. Section 3 reviews the related work on sports scheduling problems and metaheuristic approaches. In section 4, the problem is formulated and the design of constraints calculation and the objective function used to evaluate the final schedule are described. Section 5 analyses the scheduling algorithm, including the mechanism of generating initial solution and implementing simulated annealing algorithm and randomly generated problems with pseudo-code. Furthermore, the languages and tools used in this project are declared, as well as the detailed implementation of the initial solution generation and the simulated annealing. Also, an overview of the app interfaces is shown in this section. Section 7 evaluates the performance of the system through various testing scenarios, while Section 8 discusses conclusions, project management, and possible future extensions. Finally, the Appendix provides a full scheduling example generated by the application.

2 Amateur Sport Competition Scheduling Problem

The scheduling problem considered in this project is modeled as a constraint-based optimisation task. The aim is to generate an optimal schedule for a group of teams over a fixed number of venues and days, while satisfying multiple constraints at the same time, including hard constraints and soft constraints. In the following sections, a pilot problem is introduced to simulate a realistic setting for testing the algorithm, along with definitions of possible constraints and the evaluation mechanism.

2.1 Pilot Scenario

The following is a description of the pilot scenario for an amateur sports competition problem: There is a soccer league between high schools with 10 teams, which will be played over two weeks. Event organizers must schedule the timetable for each match based on some constraints. To guarantee the fairness of the competition, the constraints should be adhered to as much as feasible.

Details of the event:

- Participating teams: 10 teams (named as A, B, C, D, E,,F, G, H, I, J)
- Competition rule: single round-robin (each team plays the other teams once)
- Venues: 2 venues (named as venue1 and venue2)
- Time slots per day: 9 a.m. and 1 p.m.
- Duration: 2 weeks (14 days)

Constraints considered in the pilot scenario:

- Hard constraints:
  - Only one match can be scheduled at one venue in one time.
  - Every team must play every other team once.
  - Each team could only have one match at the same time.
- Soft constraints:
  - Every team should only play at most one match in a day.
  - As much as possible to meet each team's preferred venues and playing time, morning or afternoon.

The first two hard constraints are inherently satisfied by the problem's assumptions and implementation. Specifically, as assumed in this project, there is only one possible match per field per day in any given time slot, ensuring that there are no overlapping matches.Similarly, the single round-robin mode guarantees that each pair of teams plays exactly once against each other. The last hard constraints should be checked in the scheduling algorithm, which is enforced both during initial solution generation and simulated annealing. Regarding the soft constraints, the first one aligns with the daily constraint setting in this project and the other corresponds to the team preference constraint.

Team preference constraint settings:

- Team A: prefers to play in the morning
- Team B: prefers to play in the afternoon
- Team C: prefers to play at venue1
- Team D: no preference
- Team E: no preference
- Team F: prefers to play in the afternoon at venue2
- Team G: no preference
- Team H: prefers to play at venue2
- Team I: no preference
- Team J: prefers to play in the morning

Example output timetable by manual scheduling:

	Venue 1	**Venue 2**
Day 1	9 a.m. A vs B 1 p.m. C vs D	9 a.m. E vs F 1 p.m. G vs H
Day 2	9 a.m. A vs C 1 p.m. I vs J	9 a.m. B vs D 1 p.m. E vs G
Day 3	9 a.m. A vs I 1 p.m. C vs E	9 a.m. B vs J 1 p.m. F vs H
Day 4	9 a.m. D vs F 1 p.m. G vs I	9 a.m. H vs J 1 p.m. A vs E
Day 5	9 a.m. B vs F 1 p.m. C vs G	9 a.m. D vs H 1 p.m. E vs I
Day 6	9 a.m. A vs F 1 p.m. B vs G	9 a.m. C vs H 1 p.m. D vs I
Day 7	9 a.m. E vs J 1 p.m. F vs I	9 a.m. A vs G 1 p.m. B vs H
Day 8	9 a.m. C vs J 1 p.m. A vs D	9 a.m. B vs E 1 p.m. F vs G
Day 9	9 a.m. H vs I 1 p.m. C vs I	9 a.m. B vs C 1 p.m. D vs J
Day 10	9 a.m. E vs H 1 p.m. F vs J	9 a.m. A vs J 1 p.m. B vs I
Day 11	9 a.m. C vs F 1 p.m. D vs G	9 a.m. G vs J 1 p.m. A vs H
Day 12	9 a.m. D vs E	

Table 1: Manual Scheduling Result of the Pilot Problem

This manual scheduling took about 30 minutes to complete. However, it is not an optimal schedule as there are many violations of constraints in this scheduler. For example, team J plays two games on day 9, which violates the daily constraint that each day should have a maximum of one game per day. Also, there are 27 cases of team preferences violated, including team A plays games in the afternoon on day 4,8, and 11, team

C plays at venue2 on day 6 and 9, etc. Therefore, an algorithm is needed to find a better solution to cope with multi-constraint.

2.2 Problem Constraints

The scheduling problem involves both hard and soft constraints, which are essential to ensure feasibility and quality of the generated schedules.

Some problem constraints regarding amateur sports scheduling are listed below:

- **Place constraint:** teams are restricted to home or away games during certain periods of time [14].

  Analysis: This constraint is not considered in my project as amateur sports often occur in schools, clubs, or neighbourhoods. There is less likely to have home or away games.

- **Group constraint:** limit the maximum number of games that can be played in a time period [14].

  Analysis: This constraint is considered in my project, for example, there should be no more than three matches during one specific time considering the limited resources available in real games, such as the limited number of volunteers, referees and administrators, or the limited amount of sports equipment.

- **Date constraint:** stipulate that certain games must be played during certain time periods, for example, a game between two teams must be played before or after a specific day [14].

  Analysis: This constraint is considered in my project, for example, two teams want to play earlier or later due to some emotional factors or time availability, thereby arrangements should be made according to their preferred dates.

- **Separation constraint:** limit the time between consecutive matches, for example avoiding teams playing games on two consecutive weekends [14].

  Analysis: This constraint should be considered in my project, for example, every team should not play games in two consecutive days to ensure that team members could have a rest and better prepare for the competition.

- **Fairness constraint:** ensure that the schedule is fair for all teams, for example the difference in home games between any two teams must not exceed a certain threshold [14].

  Analysis: This constraint is not considered in my project. As mentioned before, whether it is a home game or away game is not considered in amateur sports competitions.

- **Travel distance constraint:** consider geography and try to reduce the total travel distance, for example, balancing the number of near and far plays [14][7].

  Analysis: This constraint is not considered in my project. Since amateur tournaments rarely take place between cities, there is no need to give the trip distance a lot of weight. On the other hand, they always take place during school hours or between communities, which are very close to each team member.

- **Rest difference constraint:** the difference in game breaks between different teams is an important indicator of the fairness of the schedule. To achieve fairness, the schedule should minimize the difference in rest time between the two sides [13].

Analysis: This constraint is considered in my project as it guarantees the athletes physical and mental level of fairness. Although it would be a bit complex because in many cases not all two sides could have equal rest time, this constraint should still be taken into account and met as much as possible.

- **Game allocation constraint:** each team can only play one game in each round. There are a limited number of playing days per round, and matches need to be evenly distributed among these days [13].

  Analysis: This constraint is not considered in my project as the number of games each team should play in each round depends on the game rule, whether it is round-robin or knockout.

- **Team group constraint:** teams are grouped according to their age, gender or strength to ensure fair play [7].

  Analysis: This constraint is not considered in my project as in many cases it does not matter how old the players are. Amateur sports are typically targeted at specific groups, such as school students or members of local communities, though sometimes be open to the general public across all age groups.

- **Capacity constraint:** the capacity of each playground is limited, and it is necessary to avoid multiple teams playing at the same playground at the same time [7].

  Analysis: This constraint is considered in my project as a hard constraint, for example, there should be only one team playing in one playground at the same time.

- **Compactness Constraints:** the number of rounds is minimized to make the competition schedule more compact [15].

  Analysis: This constraint is not considered in my project as the number of rounds depends on the game rule during the planned period, sometimes people would not be in such a hurry to arrange games closely.

- **Team Preference Constraints:** the preference of each team should be taken into consideration, including preferred venues and time.

  Analysis: This constraint is considered as an important part of my project as the schedule should meet as many team preferences as possible to maximise convenience.

In summary, this project considers several appropriate constraints that are both necessary for amateur sports competitions and easy to implement, especially considering the scenario of holding competitions in schools or communities. The implemented constraints include group constraints (e.g., limiting the number of matches in a given time slot), date constraints (team-specific time preferences), separation constraints (avoiding consecutive-day matches), rest difference constraints (fairness in rest time), capacity constraints (one match per venue per time slot), and team preference constraints (accommodating venue and time preferences).

Some other constraints, such as home/away balancing, travel distance minimization, team grouping by demographics, or compactness of rounds, are not considered. These are more commonly applicable to professional or large-scale events where travel and structural fairness are more critical. In contrast, amateur competitions typically involve geographically close venues and a more flexible participant structure, reducing the relevance of such factors.

The constraint considered in this project is therefore tailored to reflect the nature and limitations of non-professional tournaments, prioritizing feasibility, simplicity, and fairness within practical conditions.

2.3 Evaluation of a Schedule

The quality of a schedule is evaluated both from the perspectives of feasibility and optimisation. The evaluation is represented using the objective value given by the objective function, which gives a penalty for each constraint violation. The smaller the objective value, the better the schedule.

Feasibility
Firstly, a schedule is only considered valid if it satisfies all hard constraints. This includes ensuring that no team is assigned more than one match in the same time slot, no venue contains more than one match at the same time, and the schedule adheres to the round-robin structure where every team plays exactly once against every other team. These checks are performed during both initial solution generation and during each iteration of the simulated annealing process.

Objective function
In order to evaluate the soft constraints, an objective function is designed. This function penalties when each soft constraint is violated, including group constraint, date constraint, separation constraint, rest difference constraint, daily constraint, and team preference constraint. The goal of the scheduling algorithm is to minimise this objective value, which will subsequently provide a feasible and efficient solution with high quality.

The penalties applied for the violation of each type of soft constraint are shown below:

Type of Constraints	Penalty Coefficient
Group Constraint	2500
Date Constraint	5000
Separation Constraint	2500
Rest Difference Constraint	250
Daily Constraint	5000
Team Preference Constraint	250

Table 2: Penalty Coefficients of Different Soft Constraints

User validation
In addition to evaluating the algorithm itself, the schedule is also evaluated by the users as it is presented on the app for review. The users could visually check the allocation of each match and judge whether there is any inadequacy. The user feedback is an important part of improving the efficiency and quality of the scheduling app.

3 Related Work

The problem of sports scheduling, particularly for amateur sports competitions, has drawn increasing academic attention due to its complexity and frequent occurrence in daily life. This section reviews the existing literature and solutions from different perspectives, including foundational studies on sports scheduling, optimising algorithm approaches, existing applications, constraints frameworks, and the previous dissertation. The goal is to identify challenges and inadequacies in this field and establish a solid theoretical background for this project.

3.1 Previous Researches on Sports Scheduling

Sports scheduling has long been an important topic of numerous studies about optimisation. While most existing literature focuses on professional leagues, aiming to improve fairness, minimise travel distances, and

optimise broadcasting value, amateur sports present a set of complex challenges that are less frequently addressed.

In some studies, the common goals of scheduling professional competitions are discussed. For instance, Goossens and Spieksma [5] studied the scheduling of the Belgian professional football league, focusing on fairness and audience impact. In contrast, amateur sports contain more complex and diverse constraints originating from real life. As Knus [6] pointed out, amateur participants are often influenced by multiple roles and responsibilities outside the sports, such as work, study, and family, making scheduling more challenging as there may be more time conflicts for each participant. Additionally, public venues for amateur competitions usually need to be shared with other activities, which increases the complexity of scheduling by adding more restrictions to venues and time.

Manual scheduling under this condition is inefficient and time-consuming. As a result, the demand for intelligent scheduling is increasing in order to improve the quality of the schedule and reduce the burden at the same time. To achieve this goal, Capua et al. [3] proposed a timetabling and field allocation model for youth football teams in amateur leagues, addressing field constraints and time fairness. Besides, Schönberger et al.[10] showed the potential of hybrid techniques in amateur situations by introducing a memetic algorithm created for non-commercial sports leagues.

Recently, researchers studied deeper into practical applications. Van Bulck et al.[14] implemented RobinX, which is a constraint-driven framework for round-robin competitions. Tuffaha et al.[13] focused rest intervals between games for each team, balancing the physical fairness of the game and ensuring participants' well-being. Furthermore, Li and Goossens [7] investigated grouping and scheduling across multiple leagues, while Doornmalen et al. [15] used integer programming to study compactness in tournament design.

However, the majority of these studies are designed for desktop systems, which could provide great and stable performance without considering the response time and interaction experience as much. The application on mobile devices is quite limited, especially for amateur sports competitions. Therefore, this project is targeted at addressing this gap.

3.2 Review of Metaheuristics Approaches

There are various types of commonly used metaheuristics approaches in the literature, including simulated annealing, genetic algorithms, tabu search, and memetic algorithms. Among all the approaches, simulated annealing has been widely used for sports scheduling optimisation. This section provides a brief introduction to the simulated annealing based on a literature review.

As summarized by Busetti [2] and Talbi [12], simulated annealing mimics the physical process of annealing in metallurgy and introduces a probabilistic acceptance criterion to allow worse solutions during the search. This helps the algorithm avoid becoming trapped in local optima, which is especially useful for problems with a rugged solution landscape, such as sports scheduling. In Busetti's overview, simulated annealing involves temperature initialisation, neighbourhood generation, acceptance probability, and termination conditions. Parameter settings are crucial in simulated annealing as they determine the speed of exploring by controlling the trend of the temperature, which will influence the overall performance. There is a cooling rate to control the balance between exploration and exploitation. Normally, there will be a reheating mechanism when the temperature is lower than a specific value or there is no improvement for a long time. The following is the pseudo-code for simulated annealing.

Algorithm 1 Pseudo-code for the Simulated Annealing

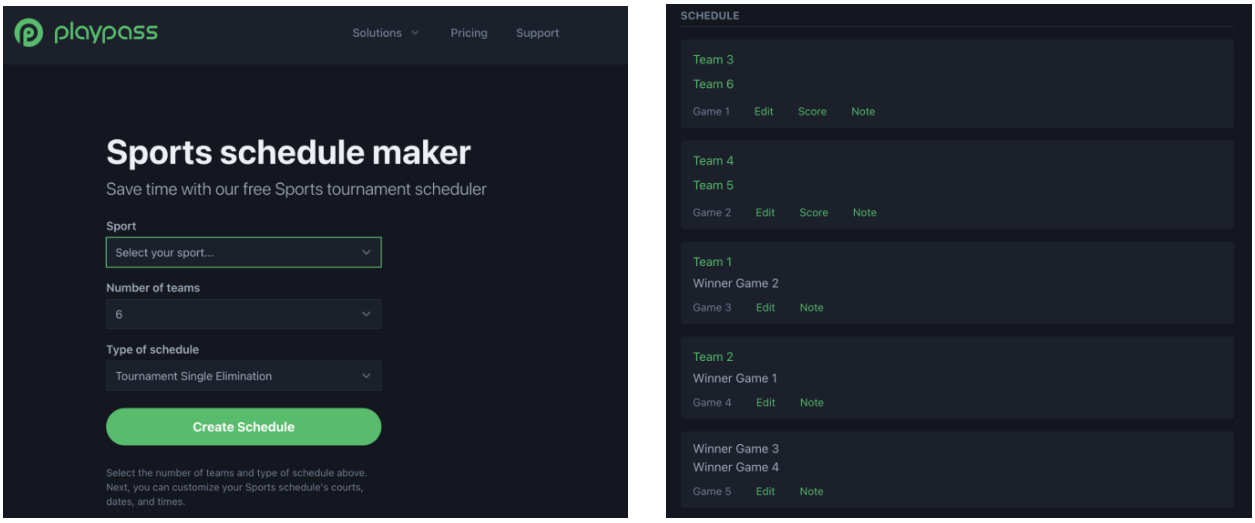
```
s ← Initial solution
T ← Initial Temperature
repeat
    repeat/* At a fixed temperature */
        Generate a random neighbor s'
        ΔE ← f(s') − f(s)
        if ΔE ≤ 0 then
            s ← s' /* Accept the neighbor solution */
        else
            Calculate acceptance probability p = e^(−ΔE/T)
            if p > random number in [0, 1] then
                s ← s' / *Accept the neighbor solution */
            end if
        end if
    until Equilibrium condition /*A given number of iterations at each temperature*/
    T ← g(T) /* Update T according to cooling schedule */
until stopping condition is met /* e.g. T < T_min */
```

Acceptance Probability Function
The Boltzman probability is often used in the simulated annealing: $P(\Delta, T) = e^{\frac{-\Delta E}{T}}$, where $\Delta E = fitness(s') - fitness(s)$ and T is the current temperature. As the temperature decreases, the probability of accepting worse solutions gradually reduces, guiding the algorithm toward convergence.

3.3 Existing Applications

There are several existing applications that support basic sports scheduling, such as Playpass [9], 4League [1], and Winner [16]. These tools typically generate random matchups and may provide options to avoid certain common issues, such as back-to-back games. However, they do not support complex scheduling constraints required in amateur sports settings, such as venue availability, preferred time slots, and fairness in match distribution.

For instance, as shown in Figure 1, Playpass offers users only three input fields: the type of sport, the number of teams, and the type of schedule. There is no option to include constraints such as venue limits or team preferences. Therefore, it is difficult to schedule amateur competitions, particularly in schools or communities, where complex restrictions need to be taken into consideration.



(a) Homepage of Playpass

(b) Schedule Interface of Playpass

Figure 1: Screenshots from Playpass demonstrating limited scheduling input fields.

Besides, the algorithmic nature of these technologies is another significant drawback. They fail to offer any optimisation or handle several constraints at the same time. This significant gap should be filled as amateur schedulers often have to cope with a range of preferences, resource constraints, and fairness issues, which require a flexible and constraint-aware algorithm.

Therefore, there is a clear need for developing a mobile-based application that could handle multiple constraints and provide an optimal solution within a short response time. This also highlights the importance of selecting a suitable metaheuristic approach, such as simulated annealing, which can accommodate a variety of constraints and give near-optimal results efficiently.

3.4 Review of Constraints

The International Timetabling Competition (ITC) is a benchmark initiative that aims to advance research in complex real-world scheduling problems. The ITC2021 problem [4] particularly focused on scheduling sports competitions, targeting at generating feasible and high-quality solutions under real constraints. In the ITC2021 competition scheduling problem, there are nine constraints mentioned, dividing into Hard Constraints (H) and Soft Constraints (S). The following five groups are used to further categorize these restrictions:

- Capacity Constraints (CA)
  - Decide which teams will play at home or away.
  - Limit the total number of games played by each team or team group.
- Game Constraints (GA)
  - Prohibit or enforce the play of specific matches in specific rounds.
- Fairness Constraints (FA)
  - Avoid disparities in the allocation of home and away or travel distances or match scheduling.
  - For instance, the amount of home and away games for each team should be about equal.

- Break Constraints (BR)
  - Limit the number of consecutive home or away games in a match.
  - A team is termed to have a "Break" if it plays two straight home games or two straight away games.
- Separation Constraints (SE)
  - Specifies the number of rounds that must be spaced between two consecutive games of the same team.

These constraints collectively capture the practical, competitive, and logistical aspects of real-world sports competitions, making ITC2021 a realistic and challenging benchmark for scheduling algorithms.

3.5 Review of the Past Dissertation

A relevant past dissertation by Teng Ma (2015) [8] focused on building a simple Android-based sports scheduling tool for single round-robin games using simulated annealing, which has similar aims to this project, but different ways of handling and modelling of constraints.

Although his project was notable for its practical implementation of scheduling on mobile devices with different constraints handling, which greatly inspired me, the treatment of constraints in his project, especially soft constraints, was simplified. Firstly, all soft constraints in the project were assigned fixed penalty weights, and the evaluation function treated all the constraints with an equal weight with no regard for the practical importance in real life. Furthermore, a more important point is that complex soft constraints such as fairness and preferences are not involved in his project. The interface developed by Ma did not provide any mechanism for users to customise constraints or define preferences, which limits its adaptability in more complex or real-world scenarios.

In contrast, this project targeted implementing more complex constraints handling and user customisation to cater for various scenarios in real life. By leveraging ITC2021-style classifications, more humanized constraints are set up in this project to reproduce the problem scenes in reality, which not only improves the quality of the model but also widens the application in the field of amateur tournament scheduling.

In addition to Teng Ma's dissertation, another relevant student project is the master's thesis titled "Scheduling in Sports – A Case Study" submitted to BI Norwegian Business School (2018) [11]. This study investigates sports scheduling from an operations research perspective, focusing on formulating the scheduling problem with constraints and objectives for a specific case company.

Compared with my project which targets at amateur sports competitions, the BI thesis focuses more on business application, targeted at a real scheduling problem of a known organisation. The motivation of reducing human efforts and improving the quality of the schedule is quite similar. While his article is targeted at all sporting events, including home and away game considerations in the professional game, the multiple constraints he mentions are relevant to the amateur game and give me some inspiration.

However, his project focuses primarily on problem formulation at a conceptual level while my project emphasises algorithm implementation and practical experience by developing the Android app. This contrast highlights the novel part of my project - the mobile-based application with multi-constraints handling for scheduling amateur competitions.

4 Design and Methodology

In this section, the overall design of the code and approaches is introduced. A model of amateur sport scheduling is presented with clear key elements including team, venue, and time information. This is followed by the constraint encoding methods, which handle the inputs and processing of the algorithm, and the objective function, which is used to evaluate the performance of the schedule, pursuing reasonable and high-quality scheduling.

4.1 Representation of the Problem

Problem Background
The problem is to implement an algorithm using optimization methods to find an optimal solution for the amateur sports scheduling problem with multiple constraints.

Entities Identification

- Team: each team represents a participant. The properties of each team include team name, preferred venue and preferred time.
- Venue: each venue only contains its name as the property.
- Match: each match represents the team, venue, and time information, including the name of team 1, the name of team 2, the venue name, the time slot of the match, and the day.

Variables
The variables that should be obtained from the user input include team names, venue names, time slots per day, duration, the validity of each constraint and each constraint settings.

4.2 Constraints Encoding

In this project, the number of soft constraints violated was accumulated for each match according to the type of constraints and each number of violations is given a penalty coefficient, which is discussed in 4.3. The calculation methods of each soft constraint are shown below:

- Group Constraint:

  Definition:
  limit the maximum number of games that can be played in a time period. For example, up to 3 games at any one time.

  Assumption:

  - *maxGames* represents the maximise number of games that could be played in each time period.
  - L represents the length of the time list, which is equal to the multiplication of the duration and the number of time slots per day.
  - N_i represents the number of simultaneous matches in the i^{th} time period in the time list, which exceeds the *maxGames*.

  The total violations of the group constraint are calculated as:

$$V_{Group} = \sum_{i=1}^{L} (N_i - maxGames) \tag{1}$$

- Date Constraint:

  Definition:
  stipulate that certain games must be played during certain time periods.

  Assumption:

  - *maxGames* represents the maximise number of games that could be played in each time period.
  - N represents the team number.
  - N_{match} represents the total number of matches

$$N_{match} = \frac{N(N-1)}{2}$$

  - N_i represents the number of matches that violate the corresponding date preference constraints of the two participating games.

  The total violations of the date constraint are calculated as:

$$V_{Date} = \sum_{i=1}^{N_{match}} N_i \tag{2}$$

- Separation Constraint:

  Definition:
  limit the maximum number of consecutive days a team can play. For example, avoid having a team play three consecutive days.

  Assumption:

  - *separationDay* represents the maximise number of consecutive days that a team can play.
  - N_t represents the team number.
  - For team t, the set of consecutive playing segments is $\{B_{t,1}, B_{t,2}, \ldots, B_{t,k_t}\}$, where the length of each consecutive segment $B_{t,j}$ is denoted as $L_{t,j}$ (that is, the number of playing days within the segment).

  The total violations of the separation constraint are calculated as:

$$V_{Separation} = \sum_{t=1}^{N_t} \sum_{j=1}^{k_t} \max(L_{t,j} - separationDay, 0) \tag{3}$$

- Rest Difference Constraint:

  Definition:
  to achieve fairness, the schedule should minimize the difference in rest time between the two sides.

  Assumption:

  - N_m represents the number of matches in total.
  - $rest_{i,A}$ represents the number of days that team 1 of match i has rested.
  - $rest_{i,B}$ represents the number of days that team 2 of match i has rested.

The total violations of the rest difference constraint are calculated as:

$$V_{Rest} = \sum_{i=1}^{N_m} |rest_{i,A} - rest_{i,B}| \tag{4}$$

- Daily Constraint:

  Definition:
  limit the number of matches played by each team per day. Each team should only play at most one match in a day.

  Assumption:

  - D represents the number of days.
  - N_t represents the number teams.
  - $match_{t,d}$ represents the number of matches that team t has during day d.

  The total violations of the daily constraint are calculated as:

$$V_{Daily} = \sum_{d=1}^{D} \sum_{t=1}^{N_t} (match_{t,d} - 1) \tag{5}$$

- Team Preference Constraint:

  Definition:
  meet each team's preferred venues and time.

  Assumption:

  - N_m represents the number of matches.
  - $v_{i,1}$ is equal to 1 if the venue of team 1 of match i is its preferred venue; Otherwise, it is equal to 0.
  - $t_{i,1}$ is equal to 1 if the time of team 1 of match i is its preferred time; Otherwise, it is equal to 0.
  - $v_{i,2}$ is equal to 1 if the venue of team 2 of match i is its preferred venue; Otherwise, it is equal to 0.
  - $t_{i,2}$ is equal to 1 if the time of team 2 of match i is its preferred time; Otherwise, it is equal to 0.

  The total violations of the team preference constraint are calculated as:

$$V_{TeamPreference} = \sum_{i=1}^{N_m} (v_{i,1} + t_{i,1} + v_{i,2} + t_{i,2}) \tag{6}$$

4.3 Objective Function Design

To evaluate the degree of compliance to soft constraints, an objective function, which returns a value representing the total penalty of each constraint violation, is implemented. Each constraint has its own penalty coefficient according to its importance in real scenario.

The overall objective function value is calculated as:

$$f = 2500 \times V_{Group} + 5000 \times V_{Date} + 2500 \times V_{Separation} + 250 \times V_{Rest} + 5000 \times V_{Daily} + 250 \times V_{TeamPreference} \quad (7)$$

where V_x represents the total number of violations of constraint x.

5 Analysis on Scheduling Algorithm

In this section, the scheduling algorithm is analysed in detail, including the mechanism of generating initial solution, the implementation of simulated annealing algorithm, and the structure of the random problem generation.

5.1 Initial Solution Generation

To ensure that the scheduling algorithm is started based on a valid initial solution, the initial solution should distribute each match evenly and avoid all hard constraints. The main idea is to ensure that no team is scheduled to play multiple matches in the same day-and-time slot, thereby satisfying basic constraints. The following are some important steps during this process:

- **Time Slot List Setup:** A time list combining all the time slots and days is constructed. Each element in the time slot list is represented using a string in the format "day, venue name, time slot". For example, "1, venue1, 09am" means that there can be a match at venue1 at 9 a.m. on the first day.
- **Random Assignment:** The two teams are first selected in sequence to avoid having more than one game between two teams. Then, the time is selected randomly from the time slot list followed by checking whether there is one team playing multiple games on the same day at the same time, which ensures the satisfaction of hard constraints. The algorithm will keep changing the random time if finding a violation of hard constraints. If no violation occurs, all the match information will be saved in the solution list as the match type.

The pseudo-code to generate the initial schedule is shown below:

Algorithm 2 Pseudo-code for generating the initial solution

```
Initialise an empty schedule S
Construct a list T of all feasible time slots
for e ach pair (team_i, team_j) where i < j:
    Choose a random time slot t from T
    while t conflicts with previously assigned match in the solution S:
        t = pick another random time slot from T
    end while
    Schedule and store match between i, j at t
end for
```

5.2 Simulated Annealing Algorithm

The simulated annealing algorithm is implemented after generating the initial solution to further optimise the schedule. As discussed in section 3.2, simulated annealing has the advantages of escaping from the local

optima and accepting worse solutions with a certain probability during the search process, which enlarges the possibility of finding the optimal solution.

In this project, the main features of simulated annealing are implemented as follows:

- **Initial Temperature:** The temperature is set to be started from 8000, ensuring the optimal solution in the shortest possible time.
- **Cooling Method:** Both linear cooling and geometric cooling methods are tried. The linear cooling function is set as $t = t - i \times 30$ where t is the current temperature and i is the iteration number. The geometric cooling rate is set to be 0.95: $t = t \times 0.95$. After comparing all randomly generated testing problems, the solutions given by geometric cooling are always a little bit better than those generated by linear cooling. Therefore, geometric cooling with a cooling rate of 0.95 is reserved as the final cooling method.
- **Iteration at each temperature:** A constant number of 10 iterations is set for each temperature, maintaining a relatively short response time while ensuring the optimal schedule.
- **Accepting Criteria:** Given a current solution s and a neighbour s', the acceptance probability is: $P(\Delta, T) = e^{\frac{-\Delta E}{T}}$, where $\Delta E = fitness(s') - fitness(s)$ and T is the current temperature. If $\Delta E \leq 0$ or a randomly generated double number between 0 and 1 is less than the probability P, the neighbourhood solution is accepted.
- **Stopping Criteria:** The final temperature is set to be 1 as the cooling mode adapted in this project is geometric cooling. Therefore, after several iterations, the temperature will never be less than 0, thereby 1 is an appropriate destination. If the temperature goes down to 1 or the running time exceeds the maximum time, which is 60 seconds, the algorithm will be stopped.
- **Reheating Mechanism:** In this project, if there is no improvement for more than 25 iterations, the temperature is set to increase by multiplying the coefficient of 1.2, which could help the algorithm escaping from the local optima.
- **Cost Function:** The cost function (i.e. the objective function defined in section 4.3) is calculated as a sum of all weighted constraints:

  $$f = 2500 \times V_{Group} + 5000 \times V_{Date} + 2500 \times V_{Separation} + 250 \times V_{Rest} + 5000 \times V_{Daily} + 250 \times V_{TeamPreference}$$

  where V_x represents the total number of violations of constraint x.

5.3 Randomly Generated Problems

Randomly generated problems are implemented in this project as testing data as generating problem manually is time-consuming and sometimes can not be guaranteed to be comprehensive. Also, it is difficult to find a similar problem scenario online as all the settings like constraints settings vary in different projects. A total of 15 questions were generated, five of each small, medium and large size. These problems are generated based on the same seed value, ensuring the same problems in each test for convenient comparison. The random generation process in this project consists of:

- **Random Teams:** The number of teams varies within a given range (8-12 for small-size problems, 12-16 for medium-size problems, and 20 for large-size problems). For each team, a preferred venue and time slot are assigned with a certain probability.
- **Rancom Venues:** The number of available venues varies in each problem size: 2 venues in small-size problems, 3-4 venues in medium-size problems, and 5 venues in large-size problems.

- **Time Slots and Days:** A total number of days is defined, as 14 for small and medium problems and 30 for large-size problems. Each day has 2 time slots, which are combined as a list of possible "(day, time)" pairs.
- **Constraints Settings:** Each soft constraint is randomly selected.

Algorithm 3 Pseudo-code for generating random scheduling problem instance

Initialise `Random(seed)`
if problemScale == 1 **then**
    Set numberOfTeams $\in$ [8, 10], numberOfVenues = 2, totalDays = 14
else if problemScale == 2 **then**
    Set numberOfTeams $\in$ [12, 16], numberOfVenues $\in$ [3, 4], totalDays = 25
else
    numberOfTeams = 20, numberOfVenues = 5, totalDays = 30
end if
Initialise empty list `teams`
for $i = 1$ to numberOfTeams
    Generate team name (e.g., A, B, C...)
    Randomly assign preferredVenue (40% chance)
    Randomly assign preferredTime (60% chance: 30% "09am", 30% "01pm")
    Add team to `teams`
end for
Initialise `venues` as "venue1" to "venueN"
Set time slots list $\mathcal{T} = \{09am, 01pm\}$
Set maxDaily = 1, maxPerTime = 3, separationDay = 3
Initialise empty `datePreference`
for $i = 1$ to 3
    Randomly select two distinct teams t_1, t_2
    Randomly choose operator $\in \{\leq, \geq\}$ and dayLimit $\in [1, totalDays]$
    Append preference "t_1, t_2, operator, dayLimit" to `datePreference`
end for
return GeneratedProblem(teams, venues, times, totalDays, maxDaily, maxPerTime, datePreference, separationDay)

Finally, all these randomized parameters—teams, venues, times, constraints—are stored in a "GeneratedProblem" class, which is then fed into our scheduling algorithm. This helps us test various scales of the problem (small, medium, large) and gather performance metrics such as solution quality and computation time.

6 Implementation

During this implementation, the detailed process of each key module is stated, including the languages and tools, and the main realisation of the scheduling algorithm. The structure of the system focuses on three core parts: initial solution generation, simulated annealing optimization, and Android application interface development. This section aims to explain how the entire system works together to accomplish amateur sports scheduling tasks by introducing the logic and structure of each part of the algorithm.

6.1 Languages and Tools

This project uses Java 11.0 and Kotlin as the main programming languages. Kotlin provides concise and safe grammar, supporting the lambda expression, which is useful in judging whether a term follows a certain

format. Also, it reduces some common errors when developing, improving the readability and maintenance of the code. Gradle is selected as the structural tool and its powerful dependency management and flexible build scripts greatly simplify the project build and continuous integration process. In Gradle, the error-handling judgment is implemented through lambda expression, which makes the error-handling logic more concise and clear.

The IDE of the project is Android Studio 14.0, which provides various debugging tools, rich plug-in support and efficient Gradle integration, which provides great convenience for our Android development. During the testing period, MATLAB is used to plot the diagrams of each variable, directly displaying the trend of the algorithm, making the tuning more targeted and reliable. In the part of interface animation, we integrated Lottie, and realized smooth and vivid animation effect by loading JSON-formatted GIF files, bringing better interactive experience to users.

In summary, this project achieves high-quality sports scheduling by integrating modern language features, build tools, and testing methods, ensuing the efficiency of development and the stability of the application and user experience.

6.2 Implementation of the initial solution generation

An essential part of the optimization method is the creation of the initial solution, which creates a reliable starting point. The Initialization class in this project creates the initial schedule by using a random pairing technique while making sure that all hard constraints are satisfied.

To ensure that there is only one instance of the original solution and prevent inconsistency, the singleton pattern is used in the method's implementation. The key inputs it receives include teams, venues, time slots, total tournament days, and constraints such as the maximum number of matches played per day and the number of matches allowed per time slot.

The `generateInitialSolution()` method constructs all feasible combination of time and venues, and then randomly assigns each pair of teams a free time slot. To avoid invalid schedules, the algorithm ensures that:

- No team is assigned more than one game at the same time.
- Each match is assigned to a valid venue and time slot.
- Once a slot is used, it is removed from the available pool to avoid duplication.

The algorithm matching of time and venues are shown below:

Listing 1: Java Code for Combining Time and Venues

```
 for (int i = 1; i <= totalDays; i++) {
    for (int k = 0; k < venues.size(); k++) {
        for (int j = 0; j < times.size(); j++) {
            String slot = i + "," + venues.get(k).getName() + "," + times.get(j);
            timeSlot.add(slot);
        }
    }
}
```

Listing 1 constructs the pool of all feasible time slots by combining every day, venue, and time option. Each combination is of string type and saved in the timeSlot list for later assignment.

During the generation process, the violation of whether having one team playing multiple games on the same day at the same time is checked in each iteration. If no violation remains, the information of the match is saved in the solution. The detailed code is as follows:

Listing 2: Java Code for Generating Initial Solution

```
for (int i = 0; i < teams.size(); i++) {
    for (int j = i + 1; j < teams.size(); j++) {
        Team team1 = teams.get(i);
        Team team2 = teams.get(j);

        int rdm = random.nextInt(timeSlot.size());
        String[] slotArr = timeSlot.get(rdm).split(",");
        currentDay = Integer.parseInt(slotArr[0]);
        currentTime = slotArr[2];

        // Avoid having one team play multiple games on the same day at the same time
        String team_1_Schedule = team1.getName() + "," + currentDay + "," + currentTime;
        String team_2_Schedule = team2.getName() + "," + currentDay + "," + currentTime;
        while (teamSchedule.contains(team_1_Schedule) || teamSchedule.contains(
            team_2_Schedule)) {
            rdm = random.nextInt(timeSlot.size());
            slotArr = timeSlot.get(rdm).split(",");
            currentDay = Integer.parseInt(slotArr[0]);
            currentTime = slotArr[2];
            team_1_Schedule = team1.getName() + "," + currentDay + "," + currentTime;
            team_2_Schedule = team2.getName() + "," + currentDay + "," + currentTime;
        }
        teamSchedule.add(team_1_Schedule);
        teamSchedule.add(team_2_Schedule);

        for (Venue v : venues) {
            if (v.getName().equals(slotArr[1])) {
                currentVenue = v;
            }
        }

        solution.add(new Match(team1, team2, currentVenue, currentTime, currentDay));
        timeSlot.remove(rdm);
        numAssigned++;
    }
}
```

In Listing2, the first 9 lines show the process of selecting teams and time. Lines 11 to 21 demonstrate the iterations of checking whether having one team playing multiple games at the same time. The rest lines perform the storage of match information.

Through the above algorithm, the initial solution ensures a valid round-robin distribution and lays the basis for further optimisation using simulated annealing.

6.3 Implementation of the Simulated Annealing

The implementation of the simulated annealing is based on the pseudo-code in section 3.2. Not only does it contain the temperature-based acceptance criterion, but it also involves additional mechanisms for maintaining stable solutions and ensuring high-quality solutions. This algorithm begins with a feasible initial solution, exploring better schedules by iterative mutation and exploring new solutions.

The mutation mechanism in this algorithm is divided into two parts:

1. Swap teams A and B between two matches.
2. Move a match to an available empty timeslot.

These techniques are used to construct new candidate solutions for evaluation in each iteration. In the design of this algorithm, which mechanism to choose is determined by the proportion of assigned time

slots to the total time slots. If the random double is smaller than the proportion, swapping teams will be performed. Otherwise, a random match will be moved to a random free time slot followed by checking the hard constraints as well. A maximum of 30 swap attempts is allowed in each iteration. This ensures efficiency by avoiding excessive time spent searching for valid swaps when constraints are tight. The following code is the implementation of the mutation:

Listing 3: Java Code for Mutation to Empty Time Slot

```
while (swapAttempt < maxSwapTry && !swapped) {
private void moveToEmptyTimeSlot(List<Match> schedule) {
    int maxTry = 30;
    int attempt = 0;
    boolean move = false;

    while (attempt < maxTry && !move) {
        List<Match> tempSolution_empty_slot = new ArrayList<>();
        copySolution(tempSolution_empty_slot, schedule);

        // Randomly select a match
        int index = random.nextInt(tempSolution_empty_slot.size());
        Match match = tempSolution_empty_slot.get(index);
        int day = match.getDay();
        Venue venue = match.getV();
        String time = match.getTime();
        String oldSlot = day + "," + venue.getName() + "," + time;

        // Randomly select an empty timeslot
        int idx = random.nextInt(emptyTimeSlot.size());
        String[] slotArr = emptyTimeSlot.get(idx).split(",");

        // Assign new slot information to the match
        int newDay = Integer.parseInt(slotArr[0]);
        String newTime = slotArr[2];
        match.setDay(newDay);
        match.setTime(newTime);

        Venue newVenue = null;
        for (Venue v : Initialisation.getInstance().getVenues()) {
            if (v.getName().equals(slotArr[1])) {
                newVenue = v;
            }
        }
        match.setV(newVenue);

        if (isValidSchedule(tempSolution_empty_slot)) {
            // If valid, update emptyTimeSlot and return
            emptyTimeSlot.remove(idx);
            emptyTimeSlot.add(oldSlot);
            move = true;
        } else {
            attempt++;
        }
    }
}
```

The above function shows the process of moving random match to random time slot, which is called in the schedule() function. Lines before 35 show the procedure of randomly selecting teams and free time slot. The remaining lines proceed the validation of feasibility.

Listing 4: Java Code for Mutation of Swaping Teams

```
while (swapAttempt < maxSwapTry && !swapped) {
    double option = random.nextDouble();
    if (option < Initialisation.getInstance().getProportion() || temperature < 6000) {
        // Mutation: randomly swap the timeslots of two matches
        int index1 = random.nextInt(newSolution.size());
        int index2 = random.nextInt(newSolution.size());
        while (index1 == index2) {
            index2 = random.nextInt(newSolution.size());
        }
        List<Match> tempSolution = new ArrayList<>();
        copySolution(tempSolution, newSolution);

        Match match1 = tempSolution.get(index1);
        Match match2 = tempSolution.get(index2);

        Team tempTeamA = match1.getA();
        match1.setA(match2.getA());
        match2.setA(tempTeamA);

        Team tempTeamB = match1.getB();
        match1.setB(match2.getB());
        match2.setB(tempTeamB);

        // Check the validity of the tempSolution
        if (isValidSchedule(tempSolution)) {
            copySolution(newSolution, tempSolution);
            swapped = true;
        } else {
            // If invalid, then try another attempt
            swapAttempt++;
        }
    } else {
        // Randomly move one match to a free timeslot
        moveToEmptyTimeSlot(newSolution);
    }
}
```

Listing 5: Java Code for Simulated Annealing

```
// Move acceptance
double delta = newCost - currentCost;
double r = random.nextDouble();
if (delta < 0 || r < Math.exp((-delta) / temperature)) {
    copySolution(currentSolution, newSolution);
}

if (calculateCost(currentSolution) > calculateCost(bestSolution) * 1.5) {
    copySolution(currentSolution, bestSolution);    // Restore the optimal solution
}

// Update the best solution
if (calculateCost(currentSolution) < bestCost) {
    copySolution(bestSolution, currentSolution);
    bestCost = calculateCost(currentSolution);
    noImprovement = 0;
} else {
    noImprovement++;
}
if (noImprovement > 30) {
    noImprovement = 0;

    // Reheat the temperature
    temperature = temperature * 1.2;
    break;
}
```

Listing 4 is the implementation of random swapping by exchanging teams of two matches. Lines before 22 show the process of selecting two random matches and swapping their teams. The rest lines is the validation of the feasibility of the new solution.

Listing 5 is the key element of simulated annealing, the move acceptance (lines 2 to 6) and reheating mechanism (lines 20 to 26). Apart from the pseudo-code described before, there is a new mechanism of restoring the optimal solution when the cost of the new solution is more than 1.5 times that of the best solution (lines 8 to 10), which ensures that the algorithm will not accept a very terrible solution, further maintaining stability.

In summary, the simulated annealing implementation in this project combines probabilistic acceptance, swap-based mutation, and constraint-aware validation to iteratively refine the scheduling solution within the allowed time limit.

6.4 Implementation of the APP Interface

As for the front-end implementation, several user interfaces are designed. This section introduces the layout and function of each interface.

6.4.1 Implementation of the Home Page

The layout of the home page is shown below:

Figure 2: Screenshot of the Home Page.

In the center of the page, there is a vivid robot waving at the user, implemented by loading JSON-formatted GIF files integrated with Lottie, which improves user interaction experience. By clicking the "Start new scheduling" button, the new scheduling will start and will jump to the setting page.

6.4.2 Implementation of the Setting Page

The layout of the setting page is shown below:

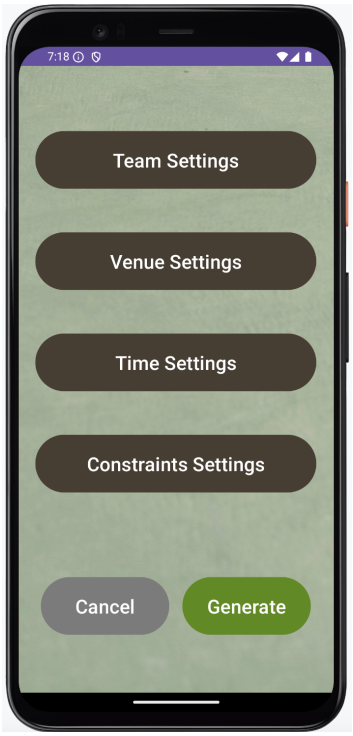
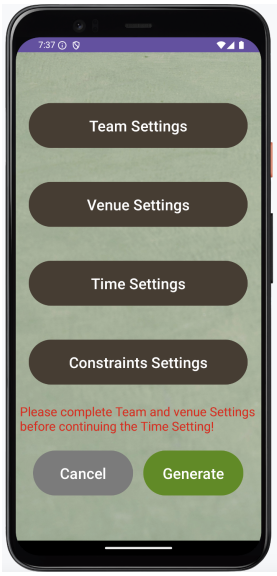


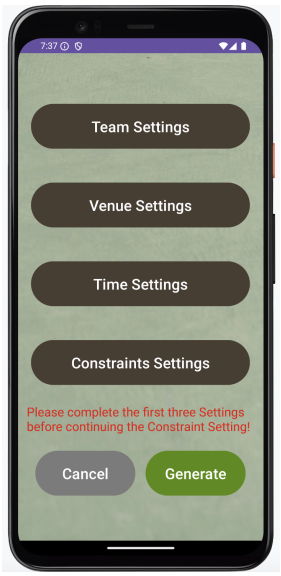
Figure 3: Screenshot of the Setting Page.

On the settings page, there are four buttons corresponding to their respective settings. Also, there is a complete system of error handling. The time settings button could only be accessed after finishing the first two settings as the calculation of duration involves team and venue information, which will be discussed in section 6.4.5 later. If not, there will be an error prompt in Figure 4(a). Similarly, the constraints settings button cannot be accessed unless the first three settings are all complete, as it allows users to input team, venue, and time information during constraint settings, which should be checked if it is the same as the early input. If not, the error message will be like Figure 4(b). Additionally, the generate button is only accessible after finishing at least the first settings as the constraint setting is optional. Without finishing the above settings, there will be a prompt shown in Figure 4(c) If the user clicks the cancel button, the schedule information will not be saved, and it will go back to the home page waiting for a new scheduling.

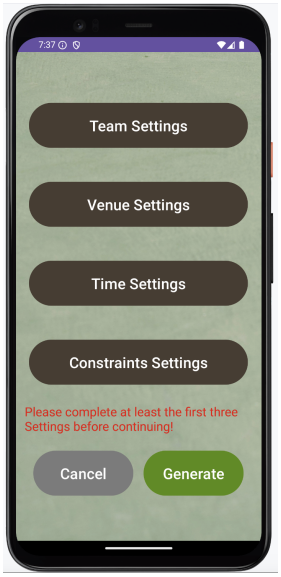
The error prompts are shown below:



(a) Error Prompt of the Time Settings Button



(b) Error of the Constraints Setting Button



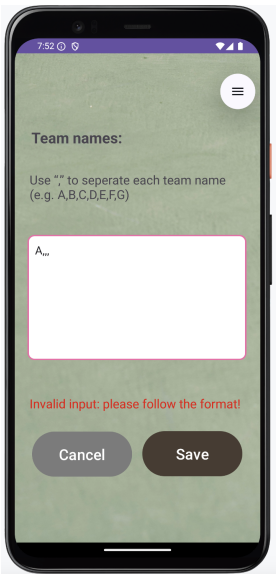
(c) Error Prompt of the Generate Button

Figure 4: Screenshots of Error Prompts of the Setting Page.

6.4.3 Implementation of the Team Setting Page

The layout of the team setting page is shown below:

(a) Screenshot of the Team Setting Page.



(b) An Example of Wrong Input in Team Setting.

Figure 5: The Team Setting Page

In the team setting page, users are required to input each team name separated by comma following the

format given as the hint. If not following the format, there will be an error prompt asking for re-input. If clicking the cancel button, the input will not be saved. If successfully clicking the save button, the input will be saved, and it will redirect to the setting page for other settings.

The input information is saved in SharedPreferences, which is a lightweight local storage mechanism provided by Android. Here is the code for saving the information into SharedPreferences:

Listing 6: Saving Input in SharedPreferences

```
SharedPreferences prefs = getSharedPreferences("InputPrefs", MODE_PRIVATE);
SharedPreferences.Editor editor = prefs.edit();
editor.putString("team_names_input", text);
editor.apply();
```

The following code is typical life cycle processing logic in Android and is used to save input when the user leaves the current page (onPause) and resume input when the user returns to the page (onResume):

Listing 7: Saving Input and Resuming Mechanism

```
@Override
protected void onPause() {
    super.onPause();
    // Save the input content
    SharedPreferences prefs = getSharedPreferences("InputPrefs", MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putString("team_names_input", editText.getText().toString());
    editor.putString("team_finish", "true");
    editor.apply();
}

@Override
protected void onResume() {
    super.onResume();
    // Restore the input content
    SharedPreferences prefs = getSharedPreferences("InputPrefs", MODE_PRIVATE);
    String savedText = prefs.getString("team_names_input", "");
    editText.setText(savedText);
}
```

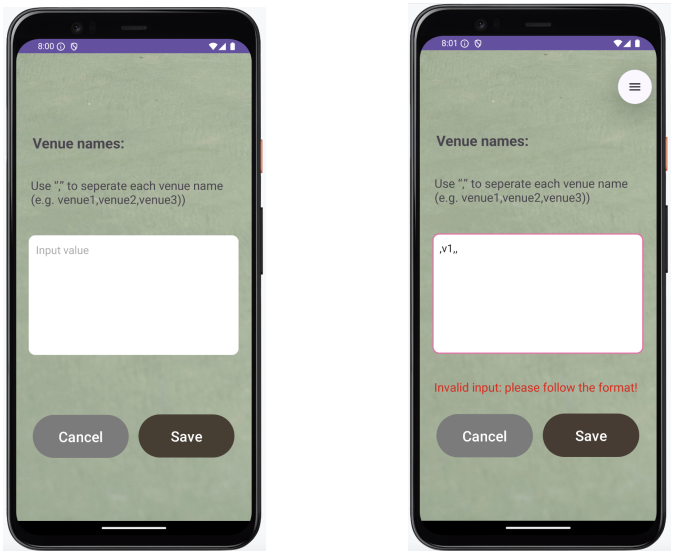
When we need to call the input of the team name in other pages or classes, we can also retrieve the data from SharedPreferences.

Listing 8: Retrieving Input from SharedPreferences

```
SharedPreferences prefs = getSharedPreferences("InputPrefs", MODE_PRIVATE);
String teamNames = prefs.getString("team_names_input", "");
```

6.4.4 Implementation of the Venue Setting Page

The layout of the venue setting page is shown below:



(a) Screenshot of the Venue Setting Page.

(b) An Example of Wrong Input.

Figure 6: The Venue Setting Page

The venue setting page is quite similar to the team setting page, with the same format of input and error-handling.

6.4.5 Implementation of the Time Setting Page

The layout of the time setting page is shown below:

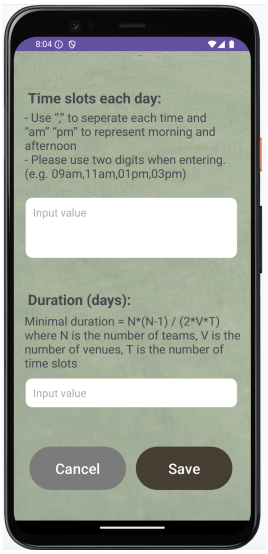
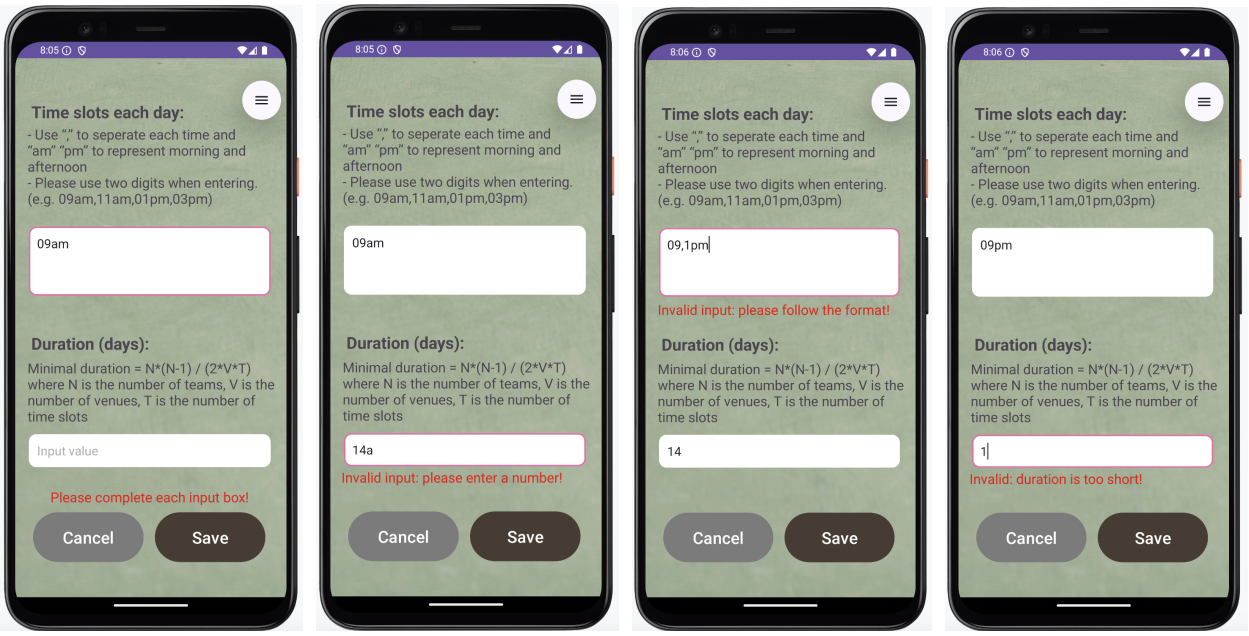


Figure 7: Screenshot of the Time Setting Page.

There are two settings in the time setting page, time slots per day and the duration. Both the two settings should be complete before continuing, otherwise there will be an error prompt shown in Figure 8(a). The type of the duration input should only be a number, otherwise an error message will appear as Figure 8(b). The format of the time slot should be two digits followed by am or pm, separated by comma, which is given as a hint on the page. If not following this format, the error prompt occur like Figure 8(c). Additionally, there is a function that tell users how to calculate the minimal duration. Assume N is the team number, V is the venue number, and T is the time slots number, then the minimal duration is :

$$Duration_{min} = \left\lceil \frac{N(N-1)}{2VT} \right\rceil$$

If the input number is smaller than $Duration_{min}$, there will be an error hint as shown in Figure 8(d).



(a) Error: duration cannot be empty

(b) Error: wrong duration format

(c) Error: wrong time slots format

(d) Error: duration is too short

Figure 8: Screenshots of Error Prompts of the Setting Page.

6.4.6 Implementation of the Constraints Setting Page

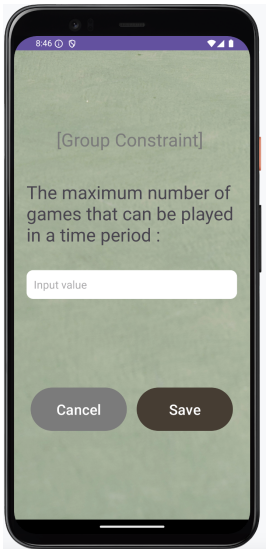
The layout of the constraints setting page is shown below:

Figure 9: Screenshot of the Constraints Setting Page.

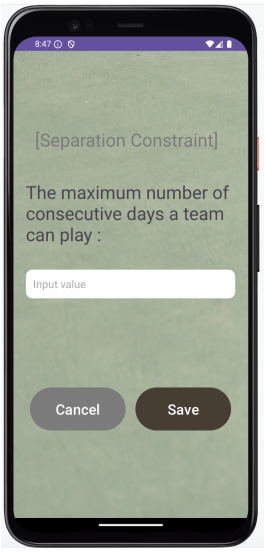
In the constraints setting page, each constraint appears as a checkbox. It will direct to the corresponding parameter setting page by clicking.

6.4.7 Implementation of the Group Constraint and Separation Constraint Setting Pages

The layout of these two constraint setting pages are shown below:



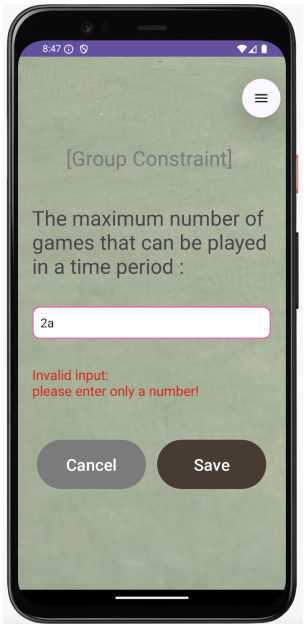
(a) The Group Constraint Setting Page.



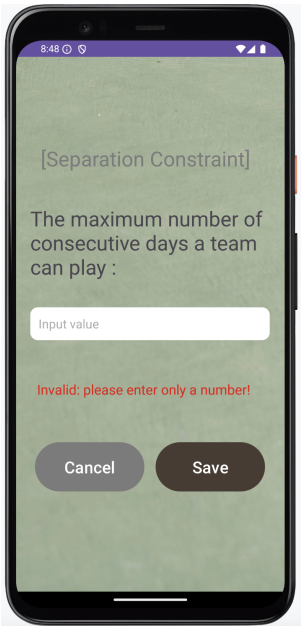
(b) The Separation Constraint Setting Page.

Figure 10: The Group Constraint and Separation Constraint Setting Page

The type of inputs in these two settings are both numbers. If the input is not a number, there will be an error prompt shown as below:



(a) An Example of Wrong Input of Group Constraint.



(b) Wrong Input of Separation Constraint.

Figure 11: Examples of Wrong Input in Group Constraint and Separation Constraint Settings

6.4.8 Implementation of the Rest Difference Constraint and Daily Constraint Setting Pages

These two constraints do not have any parameters that need to be set, so the user only needs to click those checkboxes.

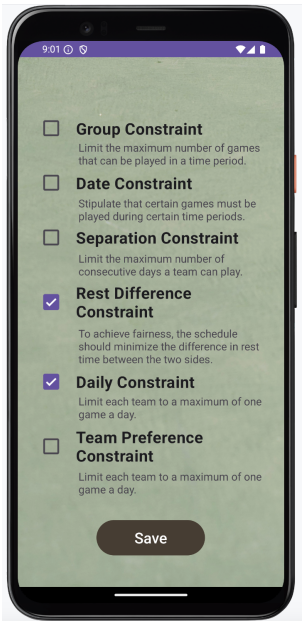


Figure 12: Screenshot of the Rest Difference Constraint and Daily Constraint Setting Page.

6.4.9 Implementation of the Date Constraint Setting Page

The layout of the date constraint setting page is shown below:

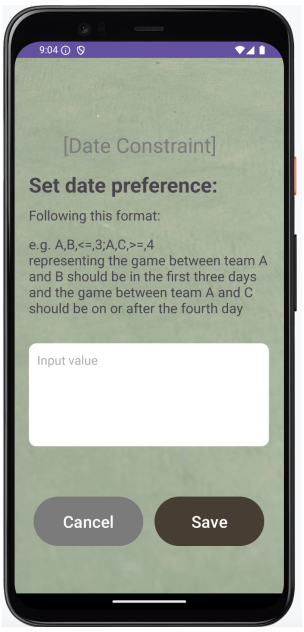
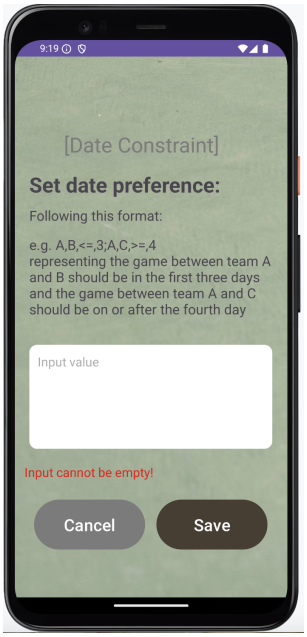
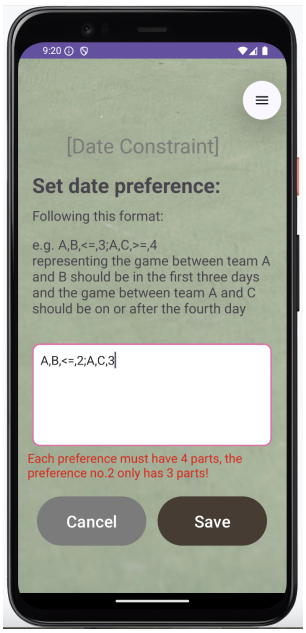


Figure 13: Screenshot of the Date Constraints Setting Page.

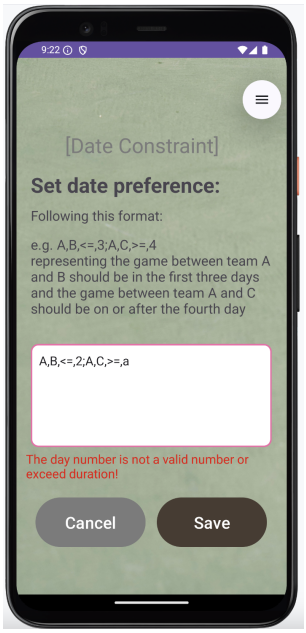
The input format of date preference should contains 4 parts, including team name 1, team name 2, the operator, and the day. The input string is split to be an array and each element is extracted and validate. Here are some situations of wrong input:



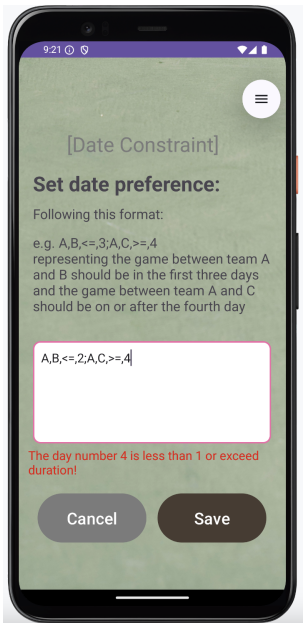
(a) Error: input cannot be empty



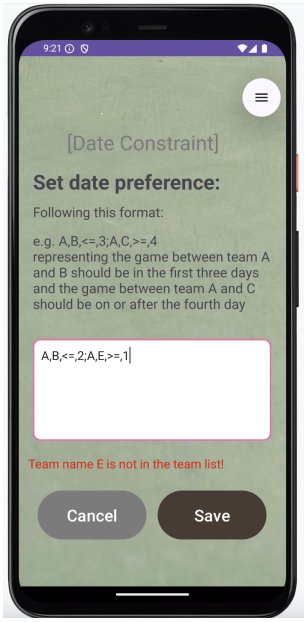
(b) Error: preference should have 4 parts



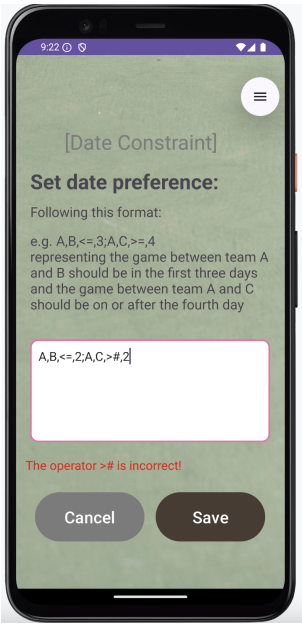
(c) Error: day should be a number



(a) Error: day cannot exceed duration



(b) Error: team name not in the team list

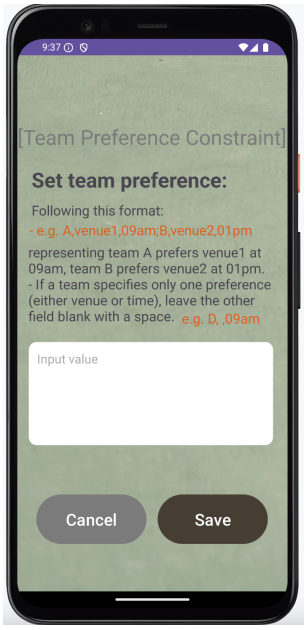


(c) Error: operator is incorrect

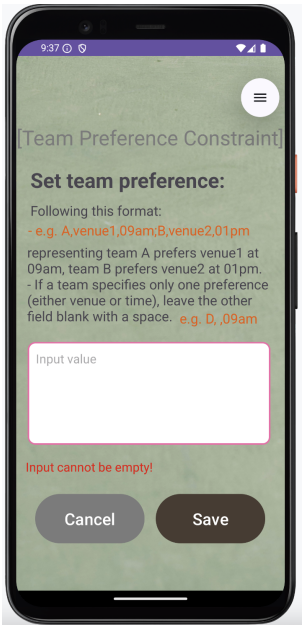
Figure 15: Examples of Wrong Input in Daily Constraint Settings

6.4.10 Implementation of the Team Preference Constraint Setting Page

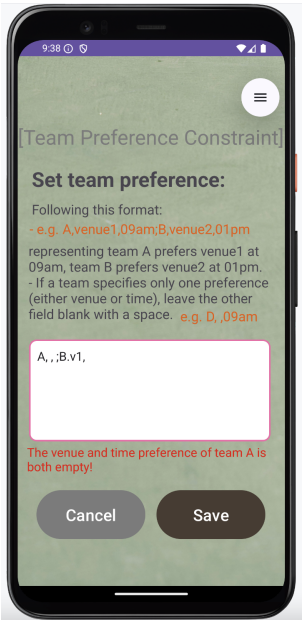
The layout of the team preference constraint setting page is shown below:



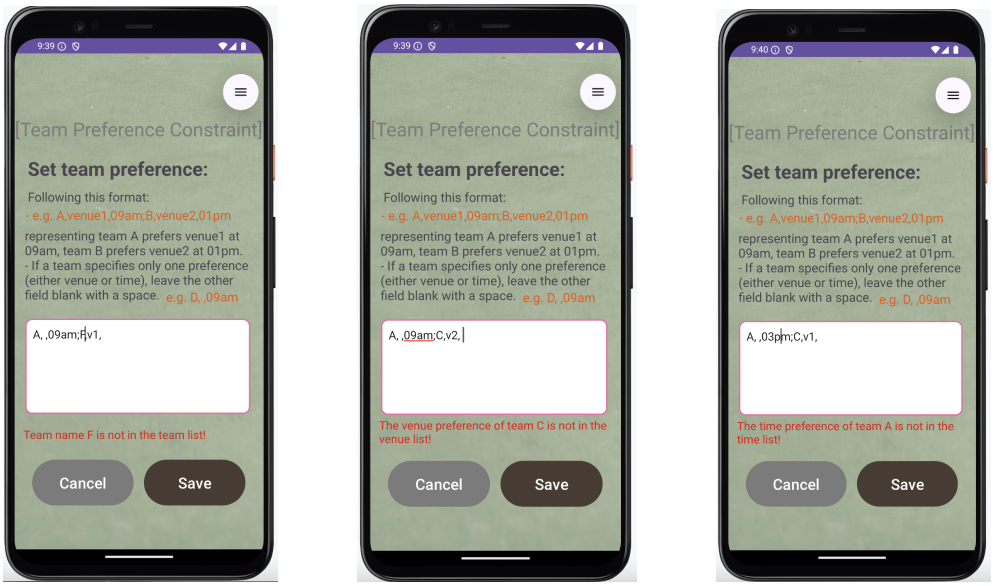
(a) The Team Preference Constraint Setting Page



(b) Error: input cannot be empty



(c) Error: venue and time cannot both be empty



(a) Error: team name not in the team list

(b) Error: venue name not in the venue list

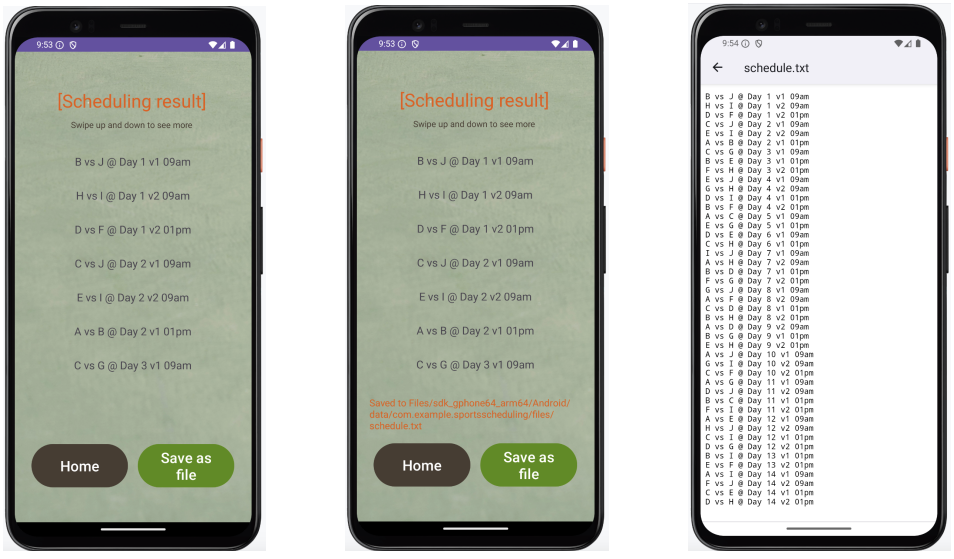
(c) Error: operator is incorrect

Figure 17: Error: time slot not in the time list

The above are some common errors when inputing team preference. Each preference should contain three parts: team name, venue name, and time slot, separated by comma. The separating punctuation between preferences is semicolon. The venue name and time slot cannot be both empty at the same time. Also, the team name cannot be empty. These error prompts give users instructions and improve the overall experience.

6.4.11 Result Page

The layout of the result page is shown below:



(a) The Result Page.

(b) Saving the result.

(c) The result file.

Figure 18: Screenshots of the Result Page.

The scheduling result is shown in the format of "team1 vs team2 @ Day venue time". The result could be swiped up and down to see more information. Additionally, the result can be saved locally on the phone in the path of "Files/sdk_gphone64_arm64/Android/data/com.example.sportsscheduling/files/schedule.txt" implemented using the file writer.

7 Evaluation

This evaluation section aims to assess the effectiveness and practicality of the amateur sports scheduling system. The main perspectives of the evaluation are to test the performance of the pilot problem and the robustness of the randomly generated problems, which could validate whether the algorithm could adapt to complex and random scenarios. Additionally, the interface should also be tested in order to guarantee user experience. Moreover, the advantages and limitations are discussed in this section. By comprehensive analysis, the future goals for improvement and support for subsequent optimization and expansion become clear.

7.1 Test of the Pilot Problem

The pilot problem consists of 10 teams, 2 venues, 2 time slots per day, and 14 days as the duration as described in detail in section 2.1. The soft constraint considered in the pilot problems are the daily constraint and the team preference constraint.

Evaluation

For each team, the maximal violations of daily constraints is $C_9^2 = 36$. Therefore, the total number of potential violations of daily constraints in the pilot problem is $36 \times 10 = 360$.

For the preference constraints, the maximal violations of:

- team A: 9
- team B: 9
- team C: 9
- team D: 0
- team E: 0
- team F: 18
- team G: 0
- team H: 9
- team I: 0
- team J: 9

Therefore, the total number of potential violations of team preference constraints in the pilot problem is $9 \times 5 + 18 = 63$.

There is a manual scheduling timetable listed in section 2.1. The Table 3 below is the solution generated by the algorithm.

	Venue 1	**Venue 2**
Day 1	9 a.m. B vs J 1 p.m. -	9 a.m. H vs I 1 p.m. D vs F
Day 2	9 a.m. C vs J 1 p.m. A vs B	9 a.m. E vs I 1 p.m. -
Day 3	9 a.m. C vs G 1 p.m. B vs E	9 a.m. - 1 p.m. F vs H
Day 4	9 a.m. E vs J 1 p.m. D vs I	9 a.m. G vs H 1 p.m. B vs F
Day 5	9 a.m. A vs C 1 p.m. E vs G	9 a.m. - 1 p.m. -
Day 6	9 a.m. D vs E 1 p.m. C vs H	9 a.m. - 1 p.m. -
Day 7	9 a.m. I vs J 1 p.m. B vs D	9 a.m. A vs H 1 p.m. F vs G
Day 8	9 a.m. G vs J 1 p.m. C vs D	9 a.m. A vs F 1 p.m. B vs H
Day 9	9 a.m. - 1 p.m. B vs G	9 a.m. A vs D 1 p.m. E vs H
Day 10	9 a.m. A vs J 1 p.m. -	9 a.m. G vs I 1 p.m. C vs F
Day 11	9 a.m. A vs G 1 p.m. B vs C	9 a.m. D vs J 1 p.m. F vs I
Day 12	9 a.m. A vs E 1 p.m. C vs I	9 a.m. H vs J 1 p.m. D vs G
Day 13	9 a.m. - 1 p.m. B vs I	9 a.m. - 1 p.m. E vs F
Day 14	9 a.m. A vs I 1 p.m. C vs E	9 a.m. F vs J 1 p.m. D vs H

Table 3: Scheduling Result of the Pilot Problem Generated by the Algorithm

For manually generated solution:

- The number of violation of daily constraint: 1.
- The number of violation of team preference constraint: 27.

By using the objective function defined in section 4.3, the cost of the manual schedule is

$$f = 5000 \times V_{Daily} + 250 \times V_{TeamPreference} = 5000 \times 1 + 250 \times 27 = 11750$$

For algorithm-generated solution:

- The number of violation of daily constraint: 0.
- The number of violation of team preference constraint: 6.

The cost of the manual schedule is

$$f = 5000 \times V_{Daily} + 250 \times V_{TeamPreference} = 5000 \times 0 + 250 \times 6 = 1500$$

The process of manual scheduling includes listing all teams, manually pairing them with available time slots, ensuring that there are no violations of hard constraints, and then adjusting them to avoid the soft constraints as much as possible by backtracking.

From the perspective of cost value, the solution generated by the algorithm is far superior to the solution generated manually. It took about 30 minutes to generate this solution manually, but only 8.61 seconds to generate this solution by the algorithm. Additionally, compared with the total number of potential violations, the number of violations in the algorithm-generated schedule is only 1.4 percent. Therefore, regarding the pilot scenario, the schedule generated by the algorithm is both optimal and time-reducing, which means a huge success.

Apart from the objective value of the solution, the algorithm performance is also an important aspect of evaluation.

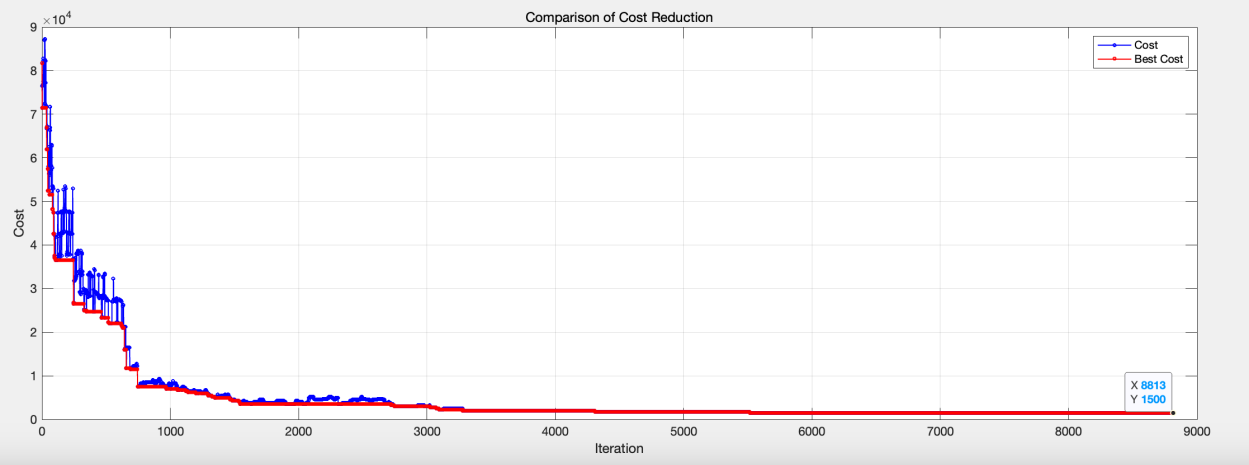


Figure 19: The Trend of the Cost and Best Cost.

Figure 19 shows the trend of cost reduction in the simulated annealing algorithm. The overall trend is clearly declining, which aligns with the expectations of the convergence trend of simulated annealing. In the first few hundred iterations, the cost fluctuates significantly, because the initial temperature is high, and the probability of accepting the "poor difference solution" is also high, which is conducive to jumping out of the local optimal. As the temperature decreases, the solution becomes more stable and tends to be globally optimal. In addition, the final optimal cost (the red line) converges earlier and remains stable, indicating that the algorithm quickly finds the optimal solution and avoids repeated fluctuations. In conclusion, this figure effectively illustrates that the simulated annealing algorithm has good optimisation ability and can gradually reduce and converge the cost in this project.

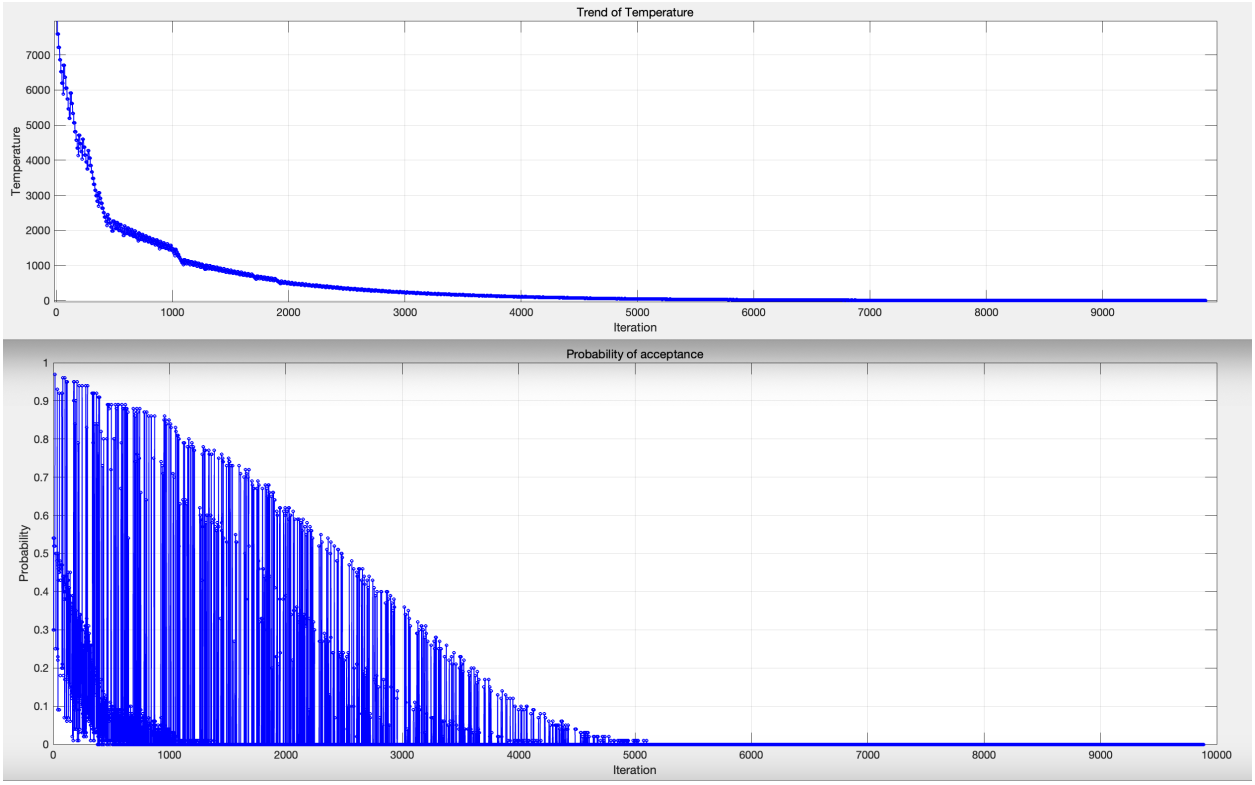


Figure 20: The Trend of Temperature Cooling and Probability Decline.

Figure 20 illustrates the trend of declining acceptance probability along with the reduction in temperature. The temperature in the figure shows a typical exponential downward trend, which is consistent with the design of the cooling mechanism. As the temperature decreases, the algorithm gradually transitions from the exploration phase to the exploitation phase. During the initial exploring stage, there is a high probability of accepting the suboptimal solution close to 1, which helps to jump out of the local optimal. As the temperature decreases, the acceptance probability approaches 0, indicating that the algorithm is more inclined to accept the optimal solution in the later stage, thus enhancing stability and convergence.

To sum up, these diagrams fully reflected the implementation effect of simulated annealing and the feasibility of the algorithm.

Tests with other seed values

seed value	**best cost**
1234	2000
12345	2000
123456	1500
1234567	1500
12345678	2000
123456789	1750

Table 4: The Best Costs with Different Seed Values

From Table 4, we could see that this algorithm is stable, providing similar optimal solution in different random situations. The best cost values obtained remain close, mostly ranging from 1500 to 2000. This

consistency indicates that the performance of the algorithm does not heavily rely on the specific seed value used, which is a desirable property for real-world applications.

7.2 Test of the Randomly Generated Problems

To evaluate the robustness and adaptation of the algorithm, several randomly generated scheduling problems are tested. These problems vary in the number of teams, venues, time, and the types of constraints being considered, which simulate complex scenarios.

The results demonstrate that the algorithm could perform well in various situations and produce stable scheduling results, suggesting that the system is generalisable and not limited to a specific hand-crafted scenarios. In addition, the trend of the diagrams plot in these random problems is similar to that of the pilot scenario, showing consistency and stable convergence behavior under different inputs.

Comparison of Two Cooling Methods Under Random Problems

In order to further explore the influence of different cooling strategies in simulated annealing algorithm on the results, this project conducted comparative experiments between geometric cooling and linear cooling in several randomly generated problem examples. The following formula is used for the two strategies:

- Geometric Cooling: $T = T \times 0.95$
- Linear Cooling: $T = T - i \times 30$

The resulting scheduling solution costs recorded at different problem sizes (i.e., scale = 1, 2, 3) are as follows:

Problem	Scale = 1					Scale = 2					Scale = 3				
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
Geometric	0	2000	0	2500	2250	0	5500	0	6750	7750	0	192750	270000	107500	31500
Linear	0	3500	0	2500	2250	0	6000	5000	6750	7750	0	192750	270000	107500	31500

Table 5: Comparison of Geometric and Linear Cooling under Different Scales and Problems

It can be seen that the two methods can complete the scheduling task well on the small-scale problem, but in some cases, the geometric cooling achieves better objective function value. However, in large-scale problems, the performance of the two is closer. In the overall trend, geometric cooling method has a stronger ability to converge to a better solution in a shorter time, which is suitable for mobile applications that have certain requirements for solving speed and quality.

Overall, geometric cooling strategy is superior to linear cooling strategy in the test scenario of this project, so geometric cooling is chosen as the default temperature updating mode in the final implementation.

7.3 Advantages

The proposed amateur sports scheduling system offers several advantages:

- **Automation:** The scheduling process is fully automated, reducing manual effort and potential for human error.
- **Constraint Handling:** Both hard and soft constraints are considered in the optimisation, increasing the practicality of the output schedule.

- **Mobile Compatibility:** The system is implemented as an Android app, providing convenience and portability for users.
- **User-Centered Design:** The interface allows input of preferences, such as preferred dates and venues, enhancing user satisfaction.
- **Robustness:** The simulated annealing algorithm has demonstrated robustness and adaptability across various problem instances.

7.4 Limitations

Despite the system's effectiveness, several limitations still exist:

- **Scalability:** With the increase in the number of teams and constraints, the responding time may increase significantly, which will affect the efficiency of solving large-scale problems.
- **Constraint Flexibility:** Only six types of soft constraints are considered in this project, which cannot meet the extension requirements of the public.
- **Single Mode:** In this project, only the round-robin mode is considered without any space for users to define the mode they like, lacking personalisation in this aspect.
- **Adjustment:** The current system does not allow users to adjust matches by themselves. If the algorithm does not find an optimal solution, or users want to personalise some matches, it would be convenient for them to adjust manually in order to give a better schedule.
- **Interaction:** The current mode of entering information following the hint is a bit complex and time-consuming. The form of the user manually entering the string is likely to be wrong and should be changed to a better way of exchanging information in the future.

8 Conclusion and Future Works

This section summarised the research outcomes, management, personal contributions and future improvement of this project. During the process of implementing the amateur sports scheduling problem, this project is oriented to the actual demand in real life, realising satisfying multiple constraints including both hard and soft constraints at the same time through simulated annealing, which has good practicability and expansibility.

8.1 Conclusion

This project builds a structure of amateur sports scheduling system, which could adapt to various complex situations, providing efficient scheduling. This project shows high performance in satisfying multiple constraints, which could give high-quality solutions while reducing human efforts at the same time. The main process of this project includes generating an initial solution, mutation between neighbourhoods, designing a simulated annealing algorithm and the objective function that deals with penalties of constraint violations. Besides, the testing results of both the pilot problem and random problems illustrate its robustness and feasibility. Although there are some aspects that could be improved in the future, this project is overall a success, achieving the initial aims.

8.2 Project Management

In terms of project management, the whole process follows the plan strictly, from initial requirement analysis, literature review, and problem definition, to optimisation programming and interface development. Git is used in this project as the version control tool, making the whole process transparent and controllable. By utilising git branches, git issues and milestones, the project keeps progressing with high efficiency and low errors. Additionally, meetings play an important role in monitoring progress and resolving potential issues in a timely manner.

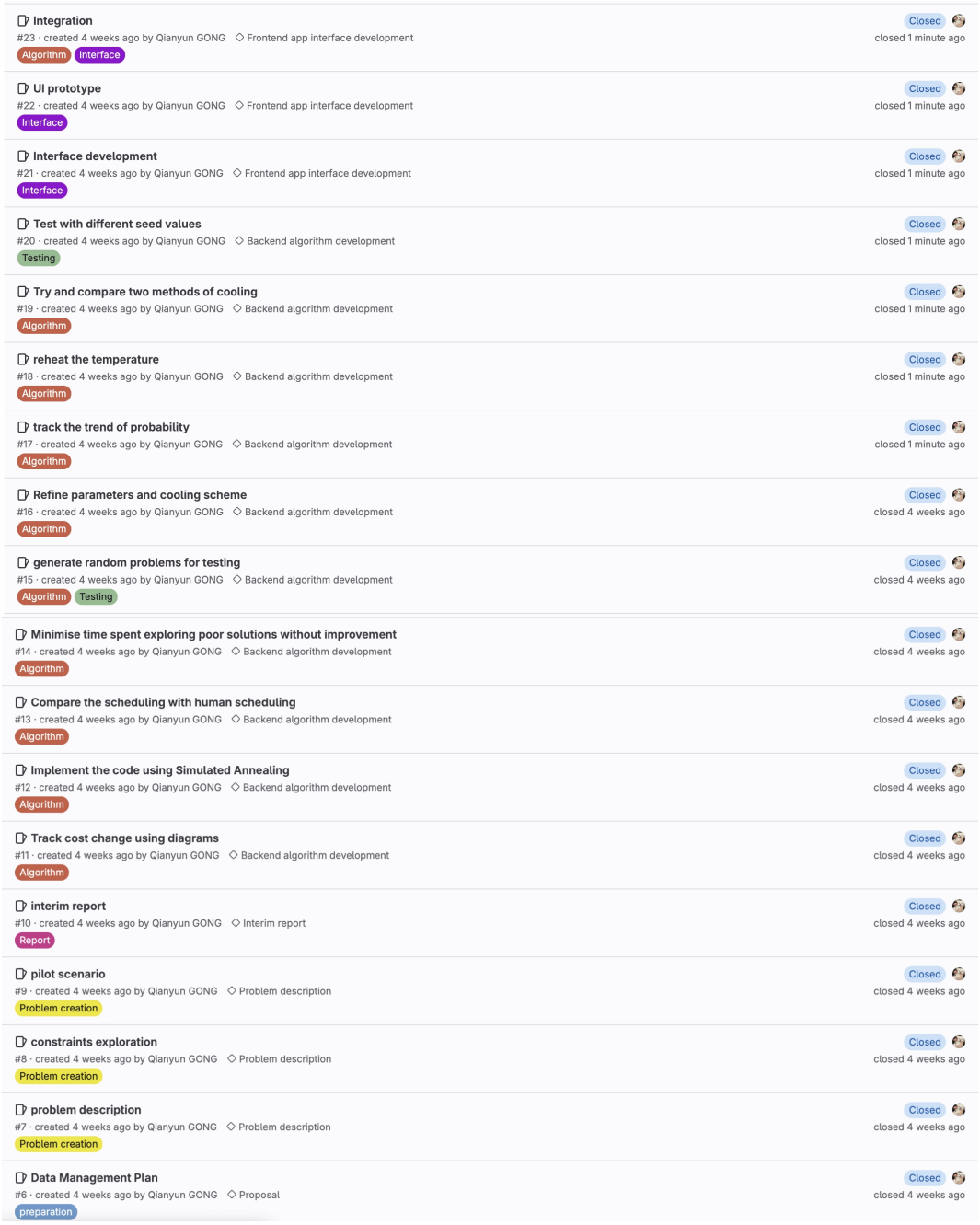


Figure 21: Issues of My Project in GitLab.

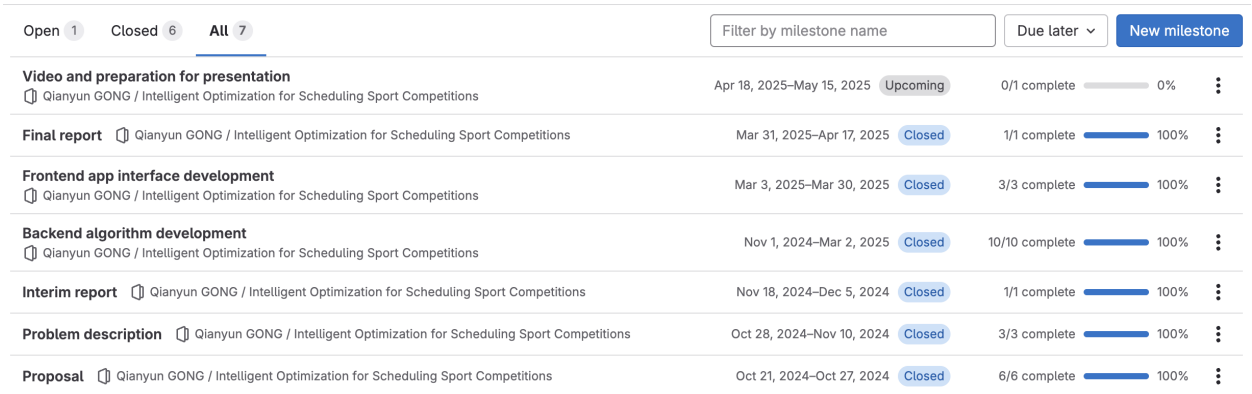


Figure 22: Milestones of My Project in GitLab.

The above figures show my project management though GitLab with each issues finished on time.

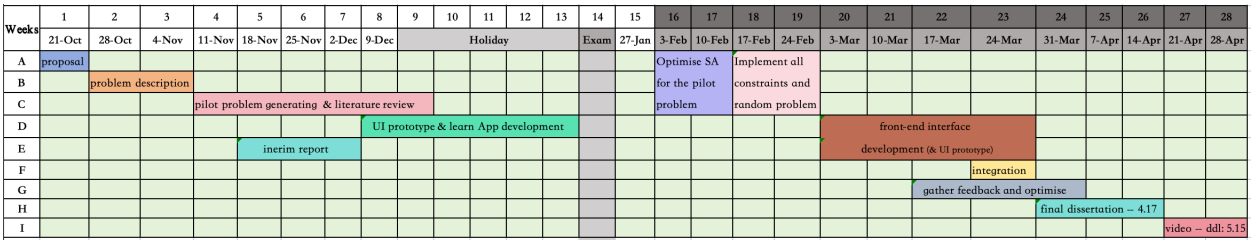


Figure 23: Project Plan.

As shown in the project plan, during the first semester, I mainly focused on the problem description, pilot problem generating, and literature review in order to identify the possible constraints and figure out the need for amateur sports scheduling. After finishing the interim report, I began implementing my algorithm, developed the whole optimisation process and integrated it with the interface, enabling a stable performance on mobile devices. Finally, I conducted tests to further improve the system, aiming at finding the optimal solution and improving user satisfaction at the same time.

8.3 Contributions

In this project, I undertook the overall design, implementation and evaluation of the system. My main contributions include:

- **Literature Review:** I conducted a literature review to figure out the potential needs and challenges for amateur sports scheduling. Also, I created the problem description and the pilot problem for representation and testing.
- **Algorithm design and implementation:** I was responsible for designing a scheduling model and implementing the simulated annealing algorithm in Java.
- **App front-end development:** I used Android Studio to complete the interface development of the mobile app, enabling users to input team information and preferences, and display the generated results.
- **Constraints Identification:** The most significant contribution I made is defining an algorithm that satisfies multiple constraints customised by users, which is meaningful for promoting amateur sports scheduling.

During this project, I have a deeper understanding of the simulated annealing algorithm by conducting exploration and figuring out the problems of slow convergence and sensitive parameter settings during tuning. Reviewing the whole development process, I not only improved my programming skills and algorithm design ability, but also further exercised my comprehensive literacy of demand analysis and system planning.

8.4 Laws, Social, Ethical and Professional Issues (LSEPI)

This project not only focuses on technical implementation and algorithm optimisation but also fully considers relevant legal, social, ethical and professional code issues to ensure that the development and application of the system meet the responsibilities and norms of the real society. The following is a discussion of several dimensions:

- **Intellectual Property:** The source code, scheduling algorithm design, and interface implementation involved in this project are all original achievements and belong to the developer. Further, open source or commercial use of the project will be considered using an appropriate open source license or applying and declaring in accordance with the university's IP policy.
- **Research Ethics:** The project does not involve human experimentation or user data collection, so ethical approval is not required. However, when simulating user behavior or preferences, ethical practices were followed to ensure that no false or misleading Settings were involved, and that the test data was entirely fictitious or anonymously generated.
- **Data Protection:** Data used in this project is only stored in a local phone chosen by the user. No data will be stored in cloud memory, avoiding the risks of remote transmission.
- **Other Legislation:** The project is deployed on the Android platform and was developed in compliance with the developer policies of Google Play and the requirements of the Computer Misuse Act, and did not introduce any potentially malicious components or features.
- **Social Influence:** This project aims at improving the efficiency of the organisers and reducing human errors, which has positive social meaning. Also, fairness is taken into consideration in this project to avoid unfair scheduling as much as possible.

8.5 Future Works

Targeting some inadequacy in this project, the following aspects could be improved in the future:

- Introduce parameter automatic adjustment mechanism to enhance the adaptability of the algorithm, especially toward large-scale problems.
- Introduce more types of constraints to cater for users' personalised requirements.
- Modify the interaction mode to make it more convenient for users to input necessary game information.
- Allow users to modify the generated scheduling if there is any error or less satisfaction.
- Implement multiple game modes other than round-robin for users to customise.

References

[1] 4League. 4league (version 1.0) [mobile app]. `https://4league.app/`, n.d. Accessed on 2025-04-14.

[2] Franco Busetti. Simulated annealing overview. World Wide Web, 2003. Accessed on April 12, 2003.

[3] R. O. Capua, S. L. Martins, and C. C. Ribeiro. Timetabling and field assignment for training youth football teams in amateur leagues. In *Proceedings of the 9th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2012)*, pages 352–363, Son, Norway, 2012.

[4] G. H. G. Fonseca and T. A. M. Toffolo. A fix-and-optimize heuristic for the itc2021 sports timetabling problem. `https://www.patatconference.org/patat2020/proceedings/papers/39.ITC2021_paper5.pdf`, n.d. Accessed on 2025-04-14.

[5] Dries Goossens and Frits Spieksma. Scheduling the belgian soccer league. *Interfaces*, 39(2):109–118, 2012.

[6] Sigrid Knust. Scheduling non-professional table-tennis leagues. *European Journal of Operational Research*, 205(2):358–367, 2010.

[7] Meng Li and Dries Goossens. Grouping and timetabling for multi-league sports competitions. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2022)*, page Paper 20, Leuven, Belgium, 2022.

[8] Teng Ma. Android-based sports competition scheduling system. Msc dissertation, University of Nottingham, School of Computer Science, 2015. Submitted in partial fulfilment of the requirements for the degree of MSc.

[9] Playpass. Tournament scheduler, n.d. Accessed on 2025-04-14.

[10] J. Schönberger, D. C. Mattfeld, and H. Kopfer. Memetic algorithm timetabling for non-commercial sport leagues. *European Journal of Operational Research*, 153(1):92–101, 2004.

[11] Ingvild Stensrud. Scheduling in sports – a case study. Master's thesis, BI Norwegian Business School, 2018. Accessed: 2025-04-17.

[12] El-Ghazali Talbi. *Metaheuristics: From Design to Implementation*. John Wiley & Sons, 2009.

[13] Tala Tuffaha, Burak Çavdaroğlu, and Tarik Atan. Timetabling round robin tournaments with the consideration of rest durations. In *Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2020)*, page Paper 65, Bruges, Belgium, 2020.

[14] Dirk Van Bulck, Dries Goossens, Jörg Schönberger, and Mario Guajardo. Robinx: An xml-driven classification for round-robin sports timetabling. In *Proceedings of the 11th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2018)*, page Paper 39, Vienna, Austria, 2018.

[15] Joris Van Doornmalen, Christopher Hojny, Roel Lambers, and Frits C. R. Spieksma. Integer programming formulations for compact single round robin tournaments. In *Proceedings of the 13th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2022)*, page Paper 22, Leuven, Belgium, 2022.

[16] Winner. Winner [web app]. `https://winner-9bee4.firebaseapp.com/`, n.d. Accessed on 2025-04-14.