

IT UNIVERSITY OF COPENHAGEN

# Group Formation Algorithm

Optimizing fair teams for an annual Church Camp

**Joshua James Calba**  
&  
**Tan Dang**

A thesis presented for the degree of  
Bachelor of Science in Software Development

Supervisor: Riko Jacob  
IT-University of Copenhagen  
Denmark  
Date: May 15, 2025

## Abstract

Forming fair and balanced teams for an annual Church Camp can be a time-consuming and subjective process. This project presents a Group Formation Algorithm that automates team assignments based on camper form data, aiming to distribute key skills such as Bible knowledge, creativity, physically fit and musical abilities across teams. Implemented in Python and solved using a Mixed Integer Programming model with the Gurobi optimizer, the algorithm ensures quick and fair team formation. A graphical user interface built with a UI framework named NiceGUI allows users to visualize and manually adjust team constraints, offering a balance between automation and human oversight. The solution demonstrates how optimization techniques can improve fairness and efficiency in group formation for the Church Camp.

## Contents

|          |                                                                     |           |
|----------|---------------------------------------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                                                 | <b>2</b>  |
| 1.1      | Setup and preparation . . . . .                                     | 2         |
| <b>2</b> | <b>Technical Description</b>                                        | <b>2</b>  |
| 2.1      | User Interface . . . . .                                            | 2         |
| 2.2      | Mixed Integer Programming (MIP) . . . . .                           | 3         |
| 2.2.1    | Relation constraints . . . . .                                      | 4         |
| 2.3      | Optimization Solvers . . . . .                                      | 7         |
| 2.4      | NP-Hardness of the Problem . . . . .                                | 7         |
| 2.5      | Data Collection and Pre-processing . . . . .                        | 8         |
| <b>3</b> | <b>How to run the Application</b>                                   | <b>8</b>  |
| <b>4</b> | <b>Discussion</b>                                                   | <b>11</b> |
| 4.1      | Software Engineering Decisions . . . . .                            | 11        |
| 4.2      | Getting the Right Answers Starts with the Right Questions . . . . . | 13        |
| 4.3      | Fair - What does it mean? . . . . .                                 | 14        |
| 4.4      | Game Theory in Group Formation . . . . .                            | 14        |
| 4.5      | Transparency of the Group Formation Process . . . . .               | 15        |
| 4.6      | Evaluating the Effectiveness of the Algorithm . . . . .             | 15        |
| 4.7      | Future Development and Improvements . . . . .                       | 16        |
| <b>5</b> | <b>Conclusion</b>                                                   | <b>17</b> |
| <b>A</b> | <b>First version of the form</b>                                    | <b>18</b> |
| <b>B</b> | <b>Current version of the form</b>                                  | <b>19</b> |
| <b>C</b> | <b>Benchmark Table</b>                                              | <b>21</b> |

# 1 Introduction

Annually, the Copenhagen Bethel Missionary Baptist Church and its missions hold their Church Camp, gathering people from all around Europe. Every year, it takes hours or even days for the Camp Committee to create fair teams, designated by colors, that will compete in various categories such as outdoor games, Bible quiz, and biblical drama competition during the camp.

To address this, this paper explores how we can develop a Group Formation Algorithm that will generate five seemingly fair teams within seconds by balancing the skill levels of each camper based on their responses to a form (see Appendix B).

Our goal is to create an algorithm that takes the campers' form responses and produces five teams where the total skill level of each team is as balanced as possible.

## 1.1 Setup and preparation

Before we could design and implement the Group Formation Algorithm, we needed relevant data to work with. Although the church had data from previous camps, this information only included campers' names, dates of birth, countries, and the churches they are from, without any records of their preferences, skills, or relationships. Therefore, we did not have sufficient data to model team formation constraints effectively.

Initially, we used last year's participant list to manually simulate responses for each camper based on what we knew about them. This process was time-consuming and only served as trial data to test the algorithm's basic functionality. At the same time, we began preparing an official form for campers to submit their actual information of their skill level.

In the early phase of data collection, participation was limited, as the official camp registration had just opened. The initial dataset included only 14 responses, far below the expected number of over 120 participants. Nonetheless, this early data allowed us to proceed with testing and development.

Over the following months, as more people registered for the camp, we gradually replaced our manual generated data with real responses from campers. This transition enabled us to work with more realistic and accurate input for the group formation model.

Throughout the data collection process, we coordinated with the Camp Committee to ensure compliance with privacy regulations, including GDPR. Campers were informed about how their data would be used, and a privacy policy was published on the official website to address these concerns [4].

## 2 Technical Description

This section provides an overview of the technical foundation of the project, explaining the key technologies and methods used to develop the prototype. The main objective was to create a system capable of forming balanced teams based on skills and constraints.

To achieve this, we chose to develop the project in **Python**, as it offers a wide range of libraries suitable for mathematical modeling and optimization. We formulated the group formation problem using **Mixed Integer Programming (MIP)**, which allows us to model assignment problems with constraints in a precise mathematical form.

For solving the optimization model, we used **Gurobi**, a commercial solver known for its speed and efficiency in handling large-scale integer programming problems. Gurobi's academic license made it accessible for our project, while providing the performance needed for practical use.

Additionally, to create a user-friendly and visual interface for interacting with the data and results, we implemented the front-end using the **NiceGUI** framework.

The following subsections will describe these technical components in more detail.

### 2.1 User Interface

To provide a visual presentation of the solution and offer a more user-friendly way to manage relations through User Interface components, we chose an open-source framework called **NiceGUI** [11]. NiceGUI allows building user interfaces that run directly in the browser and offers a very gentle learning curve, which was important for us and the Camp Committee to be able to visualize the results and interact with the data, we wanted a framework that was easy to learn and quick to use, without

spending excessive time on learning complex front-end technologies. Additionally, NiceGUI supports Tailwind CSS for styling components and provides excellent documentation, making it a practical choice for our needs.

## 2.2 Mixed Integer Programming (MIP)

Mixed Integer Programming (MIP) is widely used for solving optimization problems with both integer and continuous variables [8]. In the context of this project, MIP provides a powerful framework for modeling the group formation problem by capturing both the discrete and continuous aspects of the task. For example, assigning a camper to one of the five teams is a binary (integer) decision, either a camper is on a team or not. Similarly, relational constraints such as two people needing to be together (or not) are also represented using binary variables (1 for true, 0 for false). On the other hand, the skill levels of each camper in categories like leadership, Bible knowledge, and creativity are treated as continuous variables, which allow the algorithm to calculate and balance the total skill distribution across all teams. This combination of variable types enables the algorithm to make optimal decisions that values relational preferences while ensuring each team is fairly balanced.

**Decision Variable:**  $x_{c,t}$

The core decision variable in the group formation model is denoted as  $x_{c,t}$ , where:

- $c \in C$  represents a camper from the set of all campers  $C$ ,
- $t \in T$  represents a team from the set of all teams  $T$ .

The variable  $x_{c,t}$  is a binary variable defined as:

$$x_{c,t} = \begin{cases} 1, & \text{if camper } c \text{ is assigned to team } t \\ 0, & \text{otherwise} \end{cases}$$

This variable forms the foundation of the Mixed Integer Programming model, as it captures the assignment decisions of campers to teams. All constraints and the objective function are expressed in terms of  $x_{c,t}$ , making it the central component of the optimization problem.

The solver's task is to determine the optimal values of  $x_{c,t}$  for all campers and teams, while satisfying the defined constraints and minimizing the objective function.

The optimization is subject to the following constraints:

- Each camper is assigned to one and only one team
- Each team has approximately the same number of members.
- The total skill levels of each team are as balanced as possible.
- Additional relations constraints can be added (either manually or from the suggestion of the form answers), such as keeping or separating specific campers.

### Each camper is assigned to one and only one team

To ensure that each camper is placed in exactly one team, we add the following constraint for all campers. This guarantees that no camper is left out or assigned to multiple teams.  $x_{ct} \in \{0,1\}$ : binary decision variable indicating if camper  $c$  is assigned to team  $t$ .  $T$  is the set of all teams and  $C$  is the set of all campers.

$$\sum_{t \in T} x_{ct} = 1 \quad \forall c \in C \tag{1}$$

### Each team has approximately the same number of members

To ensure fairness in team sizes, we constrain the number of campers assigned to each team to be close to the average team size. Specifically, the number of members in a team is allowed to vary within a small range around the average.

Let  $C$  be the set of campers and  $T$  the set of teams. Let  $x_{ct} \in \{0, 1\}$  be a binary variable indicating whether camper  $c$  is assigned to team  $t$ . The average team size is computed as:

$$\bar{s} = \frac{|C|}{|T|} \quad (2)$$

We define a tolerance  $\delta \in \mathbb{N}$ , which allows a small deviation above or below the average. Then, for each team  $t \in T$ , the number of assigned campers must satisfy:

$$\bar{s} - \delta \leq \sum_{c \in C} x_{ct} \leq \bar{s} + \delta \quad \forall t \in T \quad (3)$$

This ensures that all teams have approximately the same size, with a maximum difference of  $2\delta$  campers between the smallest and largest team.

### The total skill levels of each team are as balanced as possible

For each camper  $c$ , we define a real number  $s_c^k$  representing their score in skill category  $k$  (e.g., leadership, creativity).

$$s_c^k \in \mathbb{N}_0 \quad \text{is the score of camper } c \in C \text{ in skill category } k \in K \quad (4)$$

The average score  $\bar{s}^k$  of skill category  $k$  is the total score of all campers divided by the number of teams.

$$\bar{s}^k = \frac{1}{|T|} \sum_{c \in C} s_c^k \quad (5)$$

To create an imbalance variable we define an imbalance variable  $\delta_t^k$  for each team and skill category to represent how far that team is from the average skill score. We constrain the imbalance variable to be at least the absolute difference between team  $t$  total skill  $k$  and the average.

$$\delta_t^k \geq \sum_{c \in C} s_c^k \cdot x_{ct} - \bar{s}^k \quad (6)$$

$$\delta_t^k \geq \bar{s}^k - \sum_{c \in C} s_c^k \cdot x_{ct} \quad \forall t \in T, k \in K \quad (7)$$

Similarly, we apply the same balancing approach to distribute campers from different age and demographic groups evenly across teams. This includes ensuring a fair representation of men, women, youth, children, and babies on each team. The imbalance variables and constraints follow the same structure as those used for skill categories, where the deviation from the average number of campers in each group per team is minimized.

#### 2.2.1 Relation constraints

##### Hard constraint

To ensure that two campers  $p$  and  $q$  are assigned to the same team, we enforce the following constraint for every team  $t$ :

$$\forall t \in T : x_{p,t} - x_{q,t} = 0 \quad (8)$$

And to ensure that two campers  $p$  and  $q$  are not assigned in the same team, we enforce the following constraint for every team  $t$ :

$$\forall t \in T : x_{p,t} + x_{q,t} \leq 1 \quad (9)$$

### Soft Constraints for Together Relations

The relation constraint ensuring that campers  $p$  and  $q$  are assigned to the same team can be relaxed into a **soft constraint** by introducing a penalty variable. Specifically, we define a non-negative *slack variable*  $\delta_{pq} \geq 0$  for each pair of campers  $(p, q)$ , representing the degree of violation of this relation constraint.

To model the preference that  $p$  and  $q$  should be in the same team, we replace the hard constraint with the following inequalities, applied for every team  $t \in T$ :

$$x_{p,t} - x_{q,t} \leq \delta_{pq} \quad \text{and} \quad x_{q,t} - x_{p,t} \leq \delta_{pq} \quad (10)$$

This formulation ensures that  $\delta_{pq}$  captures the absolute difference between the assignment variables:

$$|x_{p,t} - x_{q,t}| \leq \delta_{pq} \quad (11)$$

- If campers  $p$  and  $q$  are assigned to the **same team**:  $|x_{p,t} - x_{q,t}| = 0$ , hence  $\delta_{pq} = 0$ .
- If they are assigned to **different teams**:  $|x_{p,t} - x_{q,t}| = 1$ , hence  $\delta_{pq} = 1$ .

To discourage violations of this preference, we add a penalty term to the objective function:

$$\text{Objective} += w_{pq} \cdot \delta_{pq} \quad (12)$$

where  $w_{pq}$  is a penalty coefficient reflecting the importance of keeping campers  $p$  and  $q$  together. Higher values of  $w_{pq}$  strongly discourage violations, while lower values allow more flexibility.

### Soft Constraints for Separate Relations

Similarly, for relations where campers  $p$  and  $q$  should preferably **not** be assigned to the same team, we can formulate this as a soft constraint using a penalty variable.

To relax this condition, we introduce a non-negative slack variable  $\delta_{pq} \geq 0$  and modify the constraint as follows:

$$x_{p,t} + x_{q,t} \leq 1 + \delta_{pq} \quad \forall t \in T \quad (13)$$

- If  $p$  and  $q$  are in **different teams**:  $x_{p,t} + x_{q,t} \leq 1$ , so  $\delta_{pq} = 0$ .
- If they are assigned to the **same team**:  $x_{p,t} + x_{q,t} = 2$ , which implies  $\delta_{pq} = 1$ .

As before, the violation is penalized in the objective function:

$$\text{Objective} += w_{pq} \cdot \delta_{pq} \quad (14)$$

where  $w_{pq}$  controls the importance of keeping  $p$  and  $q$  apart.

While soft constraints offer flexibility, we chose to model relations as hard constraints only to keep the model simple and ensure strict enforcement. Adding soft constraints would increase complexity, which was beyond the scope of this project. Therefore, we focused on guaranteeing relation satisfaction through hard constraints.

### Relation Validation: Avoiding Conflicts and Redundancies

To ensure that the relation constraints between campers do not prevent the algorithm from finding a feasible solution, we implemented several validation mechanisms. These validations prevent logical conflicts and eliminate redundant constraints caused by transitive relations.

## Direct Conflict and Duplicate Detection

For each new relation between two campers  $p$  and  $q$ , the code first checks whether a constraint between these two already exists. Specifically:

- If a relation of the same type already exists between  $p$  and  $q$ , it is considered a duplicate and rejected.
- If a relation of the opposite type (e.g., **TOGETHER** vs **SEPARATE**) exists between  $p$  and  $q$ , it is flagged as a logical conflict and rejected.

Mathematically, given:

- $R_{\text{together}} = \{(i, j) \mid i \text{ and } j \text{ should be in the same team}\}$
- $R_{\text{separate}} = \{(i, j) \mid i \text{ and } j \text{ should not be in the same team}\}$

A new relation  $(p, q)$  of type  $r$  is valid only if:

$$(p, q) \notin R_{\text{together}} \cup R_{\text{separate}} \quad \text{or} \quad \text{type}(p, q) = \text{type}(r)$$

## Transitive Redundancy Avoidance

The **TOGETHER** relation is transitive by nature:

$$(p, q) \in R_{\text{together}} \quad \wedge \quad (q, r) \in R_{\text{together}} \quad \implies \quad (p, r) \in R_{\text{together}}$$

However, explicitly adding  $(p, r)$  as a new constraint would be redundant, as this relationship is already implied by transitivity.

To prevent this, we model the current **TOGETHER** relations as an undirected graph  $G = (V, E)$ , where:

- Vertices  $V$  represent campers.
- Edges  $E$  represent existing **TOGETHER** relations.

Before adding a new **TOGETHER** relation between  $p$  and  $q$ , we check if there exists a path connecting them in  $G$ . If such a path exists,  $(p, q)$  is already transitively connected and the new relation is rejected as redundant.

To efficiently determine whether a transitive connection already exists between campers  $p$  and  $q$ , we perform a **Breadth-First Search (BFS)** traversal on the graph  $G$ . The goal is to verify whether there is an existing path between  $p$  and  $q$  through existing **TOGETHER** relations.

## Algorithm Overview

1. Initialize an empty set **visited** to keep track of explored campers.
2. Initialize a queue with the starting camper  $p$ .
3. While the queue is not empty:
  - Dequeue the current camper.
  - If the current camper is  $q$ , a path exists, and the new relation is redundant.
  - For each unvisited neighbor of the current camper:
    - Mark the neighbor as visited.
    - Enqueue the neighbor for further exploration.
4. If no path to  $q$  is found after the traversal, the new relation is considered valid and can be added.

**Example** Consider the following existing relations:

$$(A, B) \in R_{\text{together}} \quad \text{and} \quad (B, C) \in R_{\text{together}}$$

Graphically:

$$A \longleftrightarrow B \longleftrightarrow C$$

Attempting to add  $(A, C)$  as a new **TOGETHER** relation triggers a BFS search starting from  $A$ . Since  $C$  is reachable via  $B$ , the relation  $(A, C)$  is identified as redundant and thus rejected.

### Conflict Detection for SEPARATE Relations

Adding a **SEPARATE** constraint between campers who are already transitively connected via existing **TOGETHER** relations introduces a logical contradiction. Specifically:

$$\text{If } \text{path}(p \rightarrow q) \in R_{\text{together}}, \quad \text{then adding } (p, q) \in R_{\text{separate}} \text{ is invalid.}$$

To ensure consistency, we apply the **same graph traversal technique** described in the transitive redundancy check. By verifying whether a path already exists between campers  $p$  and  $q$ , we can immediately detect and reject conflicting **SEPARATE** constraints.

### Practical Example

Consider the following existing relations:

- Camper A is **TOGETHER** with B:  $(A, B)$
- Camper B is **TOGETHER** with C:  $(B, C)$

Adding  $(A, C)$  as a new **TOGETHER** relation would be redundant, as the transitive connection  $A \rightarrow B \rightarrow C$  already exists. Furthermore, adding a **SEPARATE** relation between A and C would create an unsolvable conflict and is therefore rejected.

By applying these validation checks, we ensure logical consistency in the relation constraints and prevent the optimization model from being cluttered with redundant or conflicting rules. This helps maintain model feasibility and keeps the formulation efficient.

## 2.3 Optimization Solvers

While MIP allows us to define and model our optimization problem, we still need an optimization solver to actually compute the best solution. An optimization solver takes the mathematical model and finds an optimal assignment of variables that satisfies all constraints. There are many solvers available, both open-source and commercial. In this project, we chose to use **Gurobi** as our solver. According to Gurobi, their solver is up to **10,000 times faster** than some open-source alternatives [7]. In addition Gurobi provides free academic licenses, which allowed us to use it at no cost during the project.

## 2.4 NP-Hardness of the Problem

The group formation problem in this project is part of a class of problems known as *NP-hard*, which means they are very difficult to solve efficiently as they grow larger. Specifically, our problem is similar to classic NP-complete problems like the *Partition Problem* and the *Generalized Assignment Problem*, where items (or people in our case) must be assigned to groups while adhering to several rules [6, Introduction].

NP-hard problems are known for their complexity because the number of possible solutions grows extremely fast as the input size increases. In our case, that means as the number of campers and constraints grows, it becomes harder and slower to find the optimal solution.

To deal with this, we used Mixed Integer Programming (MIP), which lets us describe the problem in a clear mathematical way. But solving it still takes a lot of computational power, especially when the problem is large. That's why we used Gurobi [7], a high-performance solver that can handle the underlying complexity more efficiently than open-source alternatives.

In practice, our problem sizes are small enough (around 130 participants) that Gurobi can find optimal or near-optimal solutions within a few seconds. However, this theoretical complexity is important to understand, especially when considering future extensions or larger applications of the algorithm.

## 2.5 Data Collection and Pre-processing

Campers complete a survey designed by the Camp Committee to assess several key skills, including leadership, Bible knowledge, creativity, musical ability, and physical fit. The responses are collected in a CSV format and processed using a custom Python script. Each camper is represented as a structured object containing attributes corresponding to their skill levels.

## 3 How to run the Application

Here, we explain how users can run the application and navigate its interface. First, you need Python version 3.10. To start the program, run this in the root directory of the source code:

```
$ python3 main.py
```

A GUI (Graphical User Interface) will appear as shown in Figure 1. If nothing appears, please check the terminal output. It will display a URL in the format `http://localhost:xxxx`, where `xxxx` represents the port number (e.g., `http://localhost:8000`).

You can also add flags to the command to fit your preference of the program. To read more about it, refer to the `README.md` file in the GitHub repository [5]

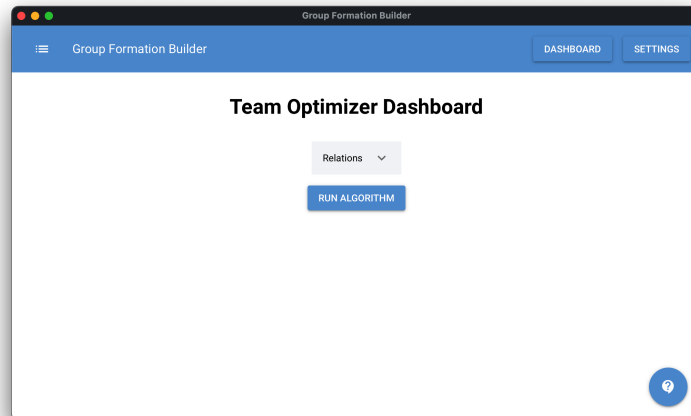


Figure 1: User interface displayed at program startup

In the top bar on the right you will see two navigation buttons. One leads to the dashboard page, which you already are in, and the other leads to the settings page, where for now the user can toggle the light and dark mode of the application. On the top left bar, there is a burger menu icon which toggles to show the side menu. There you can choose to quit the application by clicking the red button that says "Quit".

In the right bottom of the application there is a button with a question mark icon. When clicking on that, it will toggle to show a footer, where the footer content contains a GitHub icon that links to the repository as well as the current version number of the program.

When you click on *Relation*, it will expand and display two drop-down menus: one on the left for adding two people who should be together, and one on the right for adding two people who should not be together. Below each drop-down is an "ADD" button, which, when clicked, will add the selected pair with the corresponding relation. You can also choose to upload a file, save the current relations to a file, or clear all relations. Figure 2 shows the corresponding user interface. Figure 3 shows the relation table when you have added a relation.

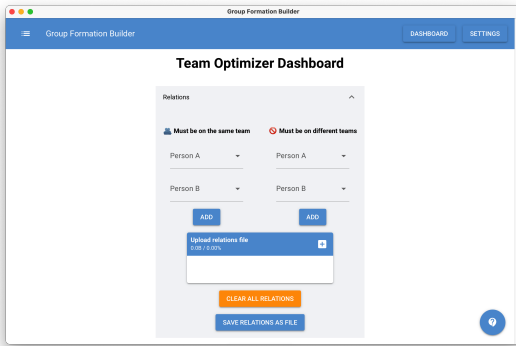


Figure 2: User interface when clicked on relation

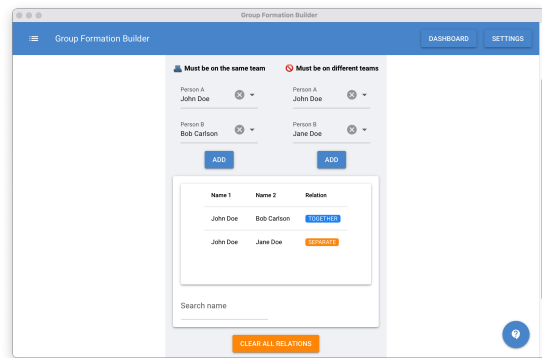


Figure 3: User interface when added a relation

When the *RUN ALGORITHM* button is clicked, a loading screen appears (Figure 4) to indicate that the algorithm is in progress. If no solution is found within the maximum time limit (default: 120 seconds), the user is notified with a message. Conversely, if a solution is successfully found, the interface will automatically scroll to the generated team results (Figure 5). Here you can choose to sort them in alphabetical order or by age and gender. Every team has an accordion that you can expand. Here each person is also given a list of emoji after their name that correspond to how they have answered the questions as shown in Figure 6.

The following emojis are displayed next to a camper's name when they answer *yes* to a corresponding question. If the answer is *maybe*, the emoji will be enclosed in parentheses. Please note that the emojis shown in Figure 6 are rendered using Apple's emoji set, as the application was run on macOS. The appearance of emojis will vary on other operating systems.

1. Crown - they said yes to be a team leader and maybe to be a co-leader
2. Bulb - they said yes to being creative
3. Music notes - They said yes to be good at playing guitar, piano or keyboard well
4. Book - They said yes to be knowledgeable about the Bible
5. Painting palette - They said yes to be able to create props for the drama competition
6. Dancing person - They said yes to be willing to take a speaker role for the drama competition
7. Running person - They said yes to be able to join physical game activities
8. Box labeled new - They said yes to never been in a Christian Camp before

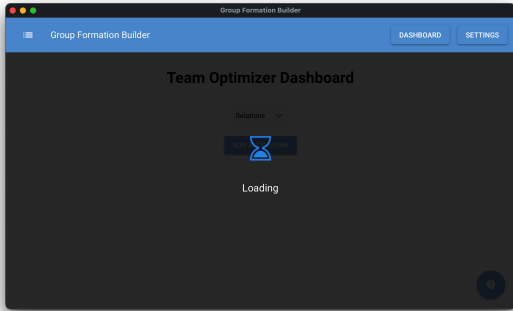


Figure 4: User interface when running the algorithm

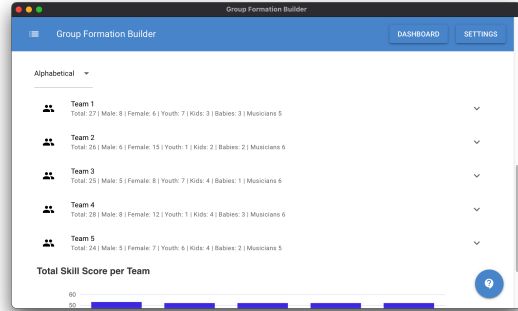


Figure 5: User interface when the program has output a solution displaying five teams

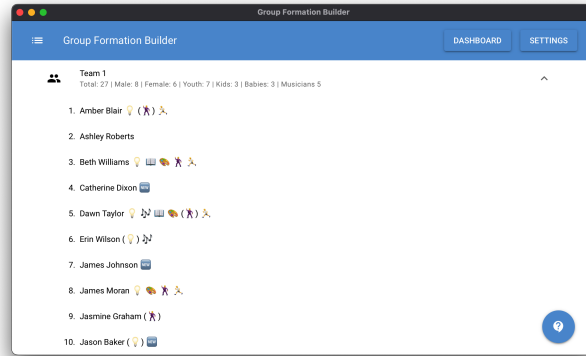


Figure 6: User interface when clicked on one of the teams will list the people in that team

When scrolling further down, the application displays charts showing the total scores for each category per team. Hovering over each chart reveals the exact total points for that category. The scoring system assigns three points for a *yes* answer, one point for a *maybe*, and zero points for a *no*. See Figure 7 for an example visualization.



Figure 7: User interface showing total scores per category for each team

## 4 Discussion

### 4.1 Software Engineering Decisions

Throughout the development of the Group Formation Algorithm and its accompanying application, several key software engineering decisions were made to ensure maintainability, performance, and usability.

#### Choice of Programming Language and Libraries

The system was implemented in Python due to its readability, wide adoption in both academic and industry settings, and its extensive ecosystem of scientific and optimization libraries. Python's support for rapid development and prototyping made it ideal for experimenting with different constraint models and interfaces.

The core optimization problem is solved using the Gurobi Optimizer, which was chosen for its performance, ease of integration with Python and availability of free academic licenses for educational purposes.

Additionally, our project supervisor's expertise in group formation algorithms and prior experience with Python ensured we had access to guidance and support for any programming challenges.

#### Modular Design

A modular architecture was adopted to improve separation of concerns and facilitate future extensions. The project is organized into logical modules, including:

- `csv_reader.py` – handles survey input and CSV data parsing.
- `formation.py` – contains the optimization model and logic for group formation.
- `gui/**/*.py` – responsible for the user interface and visual presentation of the application, built with NiceGUI.
- `main.py` – serves as the main entry point to run the application.
- `app_config.py`, `csv_schema.py`, `person.py`, `relations.py` – support files that manage configurations, data structures, and constraint logic.

This modular design ensures that each component of the system has a clear responsibility, making the source code easier to maintain, test, and extend. It also allows to adapt or enhance specific parts of the application without affecting the entire system.

#### GUI Framework Selection

For the graphical user interface, we selected NiceGUI, a modern Python-based GUI framework that leverages web technologies for a clean, responsive interface. It integrates seamlessly with backend logic and supports real-time updates and interactivity. The decision to use NiceGUI was driven by its simplicity, low learning curves and native support for input controls, file handling and visual layout.

#### Flexible Configuration for Testing and Development

The application includes configurable parameters, such as the number of teams and custom constraint inputs, which can be set through command-line arguments. This design choice was primarily made to simplify development and testing. By allowing parameters to be specified at runtime, we avoided hard coding preferences directly in the source code, making it easier to experiment with different files, variables, and configurations. This approach also improves usability for future developers or advanced users who wish to explore different optimization scenarios without modifying the source code.

## Version Control and Collaboration

The project was managed using Git for version control and hosted on GitHub. This allowed for collaboration between developers, issue tracking, and seamless integration with documentation like the `README.md` and example usage instructions.

## Data Privacy and Security Considerations

Given the sensitive nature of handling personal data, careful attention was given to data privacy and security throughout the project. Although the application does not store data permanently, it processes information such as full names, birthdays, and other personal attributes. To ensure privacy, all data handling is done locally on the user’s machine. The application does not upload or transfer any camper data to external servers or cloud services.

Since the project repository is publicly available on GitHub, we have taken precautions to ensure no sensitive data is ever exposed. This is enforced by using a `.gitignore` file that prevents actual camp data files from being pushed to the repository. All sensitive data used during development remains strictly local.

For demonstration and testing purposes, we created anonymized test datasets. These datasets are generated using common names from public libraries, ensuring no connection to real campers. Additionally, we replaced identifying attributes such as UUIDs, first and last names, birthdates, genders, and countries to fully anonymize the data. While the original skill-related answers remain unchanged for testing purposes, they are no longer linked to any identifiable person.

This approach ensures that no personal information from the camp is publicly accessible while still allowing the functionality of the application to be demonstrated and tested.

Lastly, all development work aligns with the data privacy guidelines outlined on the camp’s official website [3, 4], ensuring compliance with GDPR and the church’s own privacy policies.

## Scalability and Performance

To evaluate the scalability and performance of the group formation algorithm, benchmarking tests were conducted for increasing problem sizes. The results are summarized in Table 1. For each instance, the solver was executed four times, and the average runtime was computed. Detailed timing data is available in Appendix C.

In this benchmark, only the **total skill balance constraint** was considered. No relation constraints (such as **TOGETHER** or **SEPARATE** relations) were applied in order to focus solely on the core optimization problem.

The results show that Gurobi significantly outperforms CBC in all tested cases. For larger problem sizes, CBC failed to find a feasible solution within the given time limit of one hour. These timeouts are indicated with a dash (“-”) in the table.

The increasing runtime with problem size reflects the expected scalability challenges of the problem. Since this is a combinatorial optimization task involving team assignments and balancing multiple skills, the computational complexity aligns with typical **NP-hard** problem behavior, where solution time grows rapidly as input size increases.

Table 1: Solver Performance Comparison

| Problem Size   | Solver | Runtime (s) |
|----------------|--------|-------------|
| 10 campers     | CBC    | 4.16        |
| 10 campers     | Gurobi | 0.16        |
| 100 campers    | CBC    | -           |
| 100 campers    | Gurobi | 5.37        |
| 500 campers    | CBC    | -           |
| 500 campers    | Gurobi | 7.77        |
| 1000 campers   | CBC    | -           |
| 1000 campers   | Gurobi | 9.23        |
| 10 000 campers | CBC    | -           |
| 10 000 campers | Gurobi | 34.62       |
| 50 000 campers | CBC    | -           |
| 50 000 campers | Gurobi | 297.66      |

## 4.2 Getting the Right Answers Starts with the Right Questions

One of the key lessons learned from this project is the importance of how questions are presented to the user. Initially, two of the questions were not formulated well and therefore were changed early in the release of the form.

The first one was Question 7 (A), which originally asked if the camper had been part of a drama before. We assumed that if the camper answered "yes", they would want to have a speaking role in the drama because of their experience. However, we realized that just because a camper has experience, it does not necessarily mean they are willing to take a speaking role. The reason the question was formulated this way was because the Camp Committee wanted to ask indirectly. They believed that if asked directly, many campers would select "no" due to their previous camp experiences, resulting in very few willing to take a speaking role. From this, we learned that it is better to ask questions directly rather than indirectly, as indirect questions can lead to misrepresentation of the campers. Consequently, Question 7 was changed to directly ask if the camper is willing to take a speaking role in the drama (B).

The second question was Question 8. Initially, it was asked if campers were willing to help create props and designs for the drama (A). Later, we removed the word "help", as we wanted to know if the camper could create and design props independently. This allowed us to filter out campers who were willing to be helpful but lacked the necessary skills.

We also reviewed the first question of the form, which asked if the camper wanted to be a team leader, co-leader, or none of the above. Selecting "team leader" does not automatically make them a team leader, as there are only five team leader positions available. Moreover, the Camp Committee must approve whether a camper is fit for the role. This question serves only to indicate if the camper is willing to take a leadership role, and is used to assign their leadership score accordingly.

Questions 9 and 10, which are both text fields, are designed for campers to list full names of other campers they wish to be placed with in the same team (Question 9) or kept separate from (Question 10). Unfortunately, some campers did not understand the purpose of these questions and wrote unrelated responses. Many left the fields blank, and only a few followed the instruction to separate names by commas. We believe that the questions were formulated well enough, but perhaps they were too long for the average camper to read carefully. Some have assumed these fields were simply a message to the Camp Committee.

To avoid this issue in the future, we could implement a multiple drop-down selection menu. However, this would be difficult since we do not have a complete list of campers at the beginning of the registration process. An alternative solution would be to validate responses by checking if the answer follows the expected format, either using regular expressions or functions that detect if the camper is writing sentences instead of names by comparing with a dictionary. This remains a minor issue, as in the end, these relations are manually verified and approved by the Camp Committee.

Fortunately, the changes to Questions 7 and 8 were made early in the form's release, at a time when only 14 campers had submitted responses. This is not a problem, considering the total number of campers is expected to be around 120 or more, so only a small portion had filled out the outdated

version of the form. From this, we learned that questions should be asked clearly and directly to avoid confusion and ensure campers are accurately represented.

### 4.3 Fair - What does it mean?

Fairness was one of the main goals of this project, but defining what fair means is not always straight forward. In the case of our project, we approached fairness from both a mathematical and a human perspective.

#### Balancing Skills Across Teams

From a technical point of view, fairness meant distributing skills as evenly as possible across all five teams. Using form responses, we quantified each camper's strengths in categories such as leadership, creativity, Bible knowledge, and musical ability. The algorithm then assigned campers to teams in a way that minimized the differences in total skill levels between teams.

By ensuring that no single team had significantly more skilled members than the others, we tried to prevent situations where one team would dominate competitions or where another might feel disadvantaged. In this sense, fairness is closely related to balance. Especially the balance of visible and valuable contributions during camp activities.

#### Avoiding Manual Bias

Another aspect of fairness was reducing human bias. When teams are formed manually, there is always the risk, intentional or not, that certain people are favored or that preferences and friendships play too strong a role. By using an algorithm, we aimed to remove as much subjectivity as possible from the team formation process.

#### Perceived Fairness Is Important Too

Importantly, fairness is not only about numbers; it is also about how the outcome feels to the participants. Even if the teams are perfectly balanced from a mathematical perspective, campers may still feel that the process was unfair if they don't understand how teams were formed.

To address this potential issue, we designed the process to be transparent, where campers were informed that a group formation algorithm would be used and the main factors (the form questions) were known to all campers. However, we intentionally did not disclose exactly how their answers were weighted or how it would affect the results, as well as the specific relation constraints set by the Camp Committee are not shared with campers, in order to prevent people from trying to game the system or questioning the choice of the relation constraints.

### Limitations and Subjectivity

We understand that fairness can mean different things to different people. Even if the teams are balanced based on skills, some campers might still feel it's unfair if they are not with people they are close with, or if they feel like their team might be weak compared to the other teams.

These kinds of feelings are normal. Fairness is not just about numbers. It is also about how people experience the outcome. Our goal was not to make everyone completely satisfied, but to create a process that is consistent, balanced, and makes sense to most people.

### 4.4 Game Theory in Group Formation

Although our Group Formation Algorithm does not explicitly implement game-theoretic mechanisms, several concepts from game theory are relevant to how campers might interact with the system, especially in terms of strategic behavior, fairness, and coalitions.

In traditional game theory, agents (in this case, campers) act in their own self-interest, potentially strategizing to achieve a more favorable outcome [10]. For example, a camper might misrepresent their skills or preferences on the form if they believe it will increase their chances of being placed with friends or in a "better" team. This is an example of a game-theoretic problem known as *strategic misreporting* [9], which commonly arises in mechanism design.

To reduce incentives for such behavior, we deliberately designed the form and system to make it difficult to predict how specific answers affect team placement. Campers are not informed about how their responses are weighted or how relation constraints are used in the optimization process. This minimizes their ability to "game" the system. In doing so, we aim to follow the principle of *strategy-proofness* [9], where being honest becomes the best and safest strategy for all participants.

Fairness in allocation is also important and closely related to the concept of *envy-freeness* [1], which implies that no participant should prefer someone else's outcome over their own. While our algorithm does not guarantee envy-freeness mathematically, we try to minimize noticeable differences in team strength by balancing skill distributions as evenly as possible. This helps ensure that no team is perceived as significantly more favorable than others, which in turn reduces potential dissatisfaction or conflict.

Lastly, our system incorporates optional relational constraints (e.g., ensuring some campers are placed together or separated), which introduces a limited form of coalition dynamics. In theory, campers forming coalitions to influence their outcome is a concept from cooperative game theory [2]. To avoid unfair manipulation through such coalitions, we ensure that all relation constraints are subject to final approval by the Camp Committee.

### Example: Strategic Misreporting

Consider a camper named Alex, who is highly skilled in leadership and music. If Alex prefers to be on a team with their less-skilled friends, they might intentionally answer "maybe" or "no" on those skill questions. This lowers Alex's perceived skill level, increasing the chance they will be placed on the same team as their friends. This is a direct example of strategic misreporting [9, Chapter 9].

By keeping the internal mechanics of the algorithm hidden and avoiding published scoring criteria, we reduce the incentive and ability of campers to manipulate their outcome, reinforcing our goal of a strategy-proof process.

## 4.5 Transparency of the Group Formation Process

Campers are provided with information about the Group Formation Algorithm when they sign up for the camp, including its purpose and how their data will be processed [3]. However, the technical details of the project are not described on the website. To address this, we will refer this thesis report on the website, allowing interested campers to read about the technical aspects and other relevant sections not covered online. Additionally, the source code will be made publicly available [5]. However, understanding the code will require familiarity with programming, specifically in Python.

Regarding the input to the algorithm, campers are aware of the answers they have submitted through the form. However, they will not have access to the answers provided by other campers (unless shared personally) nor to the relational constraints indicating which campers should be placed together or kept separate.

The reason for not disclosing relation constraints is based on past experiences where campers questioned the fairness of team assignments and suspected bias from the Camp Committee, especially when assignments were done manually. Revealing these manually set relations would likely invite further complaints or unwanted discussions. Since this is the only part of the process still handled manually by us and the Camp Committee, maintaining its confidentiality helps prevent such issues.

We acknowledge that this lack of transparency regarding relations might reduce the perceived trust in the algorithm. However, after discussions with the Committee, it was agreed that limiting the visibility of relational constraints is preferable to avoid unnecessary conflicts and accusations of favoritism.

## 4.6 Evaluating the Effectiveness of the Algorithm

An important discussion point is whether the developed Group Formation Algorithm truly solves the problem faced by the Committee. From a technical standpoint, the algorithm performs well in terms of speed, efficiency, and objectivity. It distributes campers into five teams while balancing their skill levels as evenly as possible. Furthermore, it removes the human bias present in manual methods, except for relation constraints that are manually provided.

However, it is important to acknowledge that this solution is not intended to be the final, definitive team assignment. According to discussions with the Committee, the output generated by the algorithm will serve as a starting point rather than a final decision. Traditionally, teams have been formed through manual and arbitrary methods such as raffling names from a bucket or splitting lists alphabetically. While this algorithm provides a more systematic and fair initial team distribution, the Committee still retains the flexibility to make adjustments based on their insights.

There are several factors that influence why this solution cannot fully automate the group formation process. First, the accuracy of the input data depends on how campers evaluate themselves. Some may overestimate their skills, while others may underestimate their abilities out of humility. This subjectivity introduces uncertainty that the algorithm cannot control.

Additionally, the dynamic nature of camp attendance presents further challenges. Some campers may arrive late and miss specific activities, rendering certain skill evaluations irrelevant for those events. These edge cases were beyond the scope of the current solution and would require more complex handling in future iterations.

Another important consideration is the possibility of form responses being filled out by someone other than the camper. In some cases, friends or family members complete the form on behalf of campers, sometimes without fully consulting them. This can result in misrepresentations of the camper's actual skills, preferences, or relationships. Such cases introduce additional inaccuracies in the input data, further limiting the algorithm's ability to generate truly representative team assignments.

Despite these limitations, the algorithm significantly reduces the initial workload for the Committee by providing a well-balanced preliminary team distribution. It establishes a strong foundation from which the committee can refine the final teams, making the process faster and more structured than before.

In summary, while the algorithm does not solve every aspect of the group formation challenge, it offers a practical and scalable improvement over previous manual approaches.

## 4.7 Future Development and Improvements

While the current prototype successfully generates fair and balanced teams, there are several areas where the solution can be further developed to better support the Camp Committee's needs.

One key improvement is handling campers who will not attend the entire camp. Since some activities are scheduled for specific days, a camper's skill level may not be relevant if they are absent during those events. For example, a camper with high physical skills may not participate in outdoor games if they arrive late. To address this, the algorithm could be extended to consider attendance data linked to specific camp activities. However, this would require a more dynamic model, as the camp schedule changes every year based on feedback and new ideas.

Another area of improvement involves the user interface. Currently, the application is designed as a prototype for internal testing purposes. It allows us to load data from a CSV file and generate team formations, but it is not optimized for non-technical users. In the future, we aim to develop a more user-friendly interface tailored to the Camp Committee. This would include adjustable parameters, better visualizations of the generated teams, and interactive tools to fine-tune the results without requiring technical knowledge.

Beyond team formation, the Camp Committee also faces challenges in room assignments during the camp. Limited room capacity, family preferences, gender separation, and age considerations make room planning a time-consuming and complex task. For example, younger children may share beds with parents, while older campers require separate beds. Additionally, each room typically has a designated counselor responsible for its members, adding another layer of complexity.

Inspired by the team formation solution, a future extension of this project could focus on optimizing room assignments. This would involve developing a new form to collect relevant preferences and constraints for room sharing, and adapting the algorithm to handle these additional requirements. Such a solution could significantly reduce the workload for the Camp Committee and ensure a more organized and comfortable room distribution for all participants.

Overall, while the current algorithm serves as a solid starting point, there are many opportunities to expand its functionality and improve its usability for real-world camp planning scenarios.

## 5 Conclusion

This project set out to address the time-consuming and often subjective process of forming fair and balanced teams for the annual Church Camp organized by Copenhagen Bethel Missionary Baptist Church. Through the development of a Group Formation Algorithm, we aimed to streamline this task by automating the team assignment process while ensuring diversity and fairness based on campers' skills and relational constraints.

Using Mixed Integer Programming (MIP) and the Gurobi optimizer, we were able to model the group formation problem mathematically and generate team distributions within seconds. The algorithm efficiently balances skill levels across teams and eliminates initial human bias, providing the Camp Committee with a structured and data-driven starting point for team assignments. Additionally, a prototype user interface was developed using the NiceGUI framework, allowing for basic interaction with the system and visualization of the results.

However, this solution is not without its limitations. Factors such as the accuracy of self-reported skill levels, late registrations, and dynamic camp schedules present challenges that the current model does not fully address. Furthermore, while the algorithm generates a fair distribution, the final team assignments still require human adjustments to account for personal and situational considerations.

Despite these challenges, the project successfully demonstrates how optimization techniques can significantly reduce manual workload and bring more transparency to the team formation process. The collaboration with the Camp Committee ensured that practical needs and privacy concerns, including GDPR compliance, were considered throughout the project.

Looking ahead, there are several opportunities for further development, including handling partial attendance, improving the user interface for non-technical users, and extending the solution to optimize room assignments. These enhancements would bring the project closer to a fully integrated camp management tool.

It is also important to note that, at the time of writing, the actual Church Camp event has not yet taken place. Therefore, we are unable to evaluate how well the generated teams will perform in practice or gather feedback from the campers and the Camp Committee. As this paper will be submitted before the event, the real-world impact and reception of the algorithm remain to be seen.

In conclusion, while the algorithm is not a complete replacement for human judgment, it serves as an effective support tool that improves fairness, efficiency, and transparency in group formation. This project provides a strong foundation for future improvements and demonstrates the potential of optimization in solving real-world challenges.

## References

- [1] Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, 1996. Accessed: 2025-05-12.
- [2] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Computational Aspects of Cooperative Game Theory*. Morgan & Claypool Publishers, 2011. Accessed: 2025-05-12.
- [3] Copenhagen Bethel Missionary Baptist Church. Camp website - group formation algorithm, 2025. Accessed: 2025-05-15. URL: <https://camp.cbmbc.org/group-formation>.
- [4] Copenhagen Bethel Missionary Baptist Church. Privacy policy - cbmbc church camp, 2025. URL: <https://camp.cbmbc.org/privacy-policy>.
- [5] Joshua James Calba & Tan Dang. Group formation algorithm - bachelor project repository, 2025. Accessed: 2025-05-15. URL: <https://github.com/Josh0104/The-Group-Formation-Algorithm>.
- [6] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. Accessed: 2025-05-13.
- [7] LLC Gurobi Optimization. Open source vs commercial linear programming solver, 2023. Accessed: 2025-05-11. URL: <https://www.gurobi.com/resources/open-source-vs-commercial-linear-programming-solver/>.
- [8] Gurobi Optimization, LLC. Mixed integer programming (mip), 2024. Accessed: 2025-05-07. URL: <https://www.gurobi.com/faqs/mixed-integer-programming/>.
- [9] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007. Accessed: 2025-05-12. URL: <https://www.cs.cmu.edu/~sandholm/cs15-892F13/algorithmic-game-theory.pdf>.
- [10] Martin J. Osborne. *An Introduction to Game Theory*. Oxford University Press, 2004. Accessed: 2025-05-12. URL: [https://mathematicalolympiads.wordpress.com/wp-content/uploads/2012/08/martin\\_j\\_-\\_osborne-an\\_introduction\\_to\\_game\\_theory-oxford\\_university\\_press\\_usa2003.pdf](https://mathematicalolympiads.wordpress.com/wp-content/uploads/2012/08/martin_j_-_osborne-an_introduction_to_game_theory-oxford_university_press_usa2003.pdf).
- [11] zauberzeug GmbH. Nicegui - a python library to build web-based guis, 2025. URL: <https://nicegui.io/>.

## A First version of the form

Each question will list answers that the user can pick. Question 1 - 8 are required to be answered and can only pick one answer, while Question 9 and 10 are input text field that the user either can choose to leave blank or write the list of people's name separated by comma.

1. Are you willing to be a team leader on your team?
  - (a) Yes, I am willing to be a team leader
  - (b) No, but I can co-lead and help the team leader
  - (c) No
2. Would you consider yourself a creative thinker?
  - (a) Yes
  - (b) Maybe
  - (c) No
3. Do you consider yourself knowledgeable about the Bible?

- (a) Yes
  - (b) Maybe
  - (c) No
4. Are you physically fit and able to play in outdoor activities like outdoor games, obstacle courses, running, sports etc.?
- (a) Yes
  - (b) Maybe
  - (c) No
5. Can you play the guitar, piano or keyboard well?
- (a) Yes
  - (b) No
6. Have you been in a Christian camp before?
- (a) Yes
  - (b) No
7. Have you been part of a roleplay, skit or drama before?
- (a) Yes
  - (b) Maybe
  - (c) No
8. Will you be able to help create and design costumes and props for the drama?
- (a) Yes
  - (b) No
9. If you are entitled to be in the team with someone, please write their full name (for more than one person please separate the names by comma ',') e.g. kids needs to be with their parent/guardian or new camper needs to be with someone they know. **Please note that listing someone does not guarantee you will be placed on the same team. We will only honor these requests if they are reasonable and relevant. Unreasonable requests will be removed before the group formation process, so please answer truthfully. Leave blank if none,**
- (a) The user may leave the input blank or provide a list of names, separated by commas.
10. Please list the full names of people (separate by comma ',') who should not be on your team (e.g., spouse, siblings, roommates, people I see (almost) every day). This will help us create teams that encourage meeting new people, fostering new connections, and promoting teamwork with a more diverse group. Being on a team with familiar faces can limit these opportunities. **Leave blank if no**
- (a) The user may leave the input blank or provide a list of names, separated by commas.

## B Current version of the form

Question 7 and 8 are the only one that was changed

1. Are you willing to be a team leader on your team?
- (a) Yes, I am willing to be a team leader
  - (b) No, but I can co-lead and help the team leader
  - (c) No

2. Would you consider yourself a creative thinker?
  - (a) Yes
  - (b) Maybe
  - (c) No
3. Do you consider yourself knowledgeable about the Bible?
  - (a) Yes
  - (b) Maybe
  - (c) No
4. Are you physically fit and able to play in outdoor activities like outdoor games, obstacle courses, running, sports etc.?
  - (a) Yes
  - (b) Maybe
  - (c) No
5. Can you play the guitar, piano or keyboard well?
  - (a) Yes
  - (b) No
6. Have you been in a Christian camp before?
  - (a) Yes
  - (b) No
7. Are you willing to take a speaking role in the Biblical drama? (Ex. main role, supporting role, narrator etc.)
  - (a) Yes
  - (b) Maybe
  - (c) No
8. Will you be able to create and design costumes and props for the drama?
  - (a) Yes
  - (b) No
9. If you are entitled to be in the team with someone, please write their full name (for more than one person please separate the names by comma ',') e.g. kids needs to be with their parent/guardian or new camper needs to be with someone they know. **Please note that listing someone does not guarantee you will be placed on the same team. We will only honor these requests if they are reasonable and relevant. Unreasonable requests will be removed before the group formation process, so please answer truthfully. Leave blank if none,**
  - (a) The user may leave the input blank or provide a list of names, separated by commas.
10. Please list the full names of people (separate by comma ',') who should not be on your team (e.g., spouse, siblings, roommates, people I see (almost) every day). This will help us create teams that encourage meeting new people, fostering new connections, and promoting teamwork with a more diverse group. Being on a team with familiar faces can limit these opportunities. **Leave blank if no**
  - (a) The user may leave the input blank or provide a list of names, separated by commas.

## C Benchmark Table

Table 2: Detailed Runtime Results (averaged over 4 runs, no relation constraints, 5 groups)

| <b>Solver</b> | <b>Problem Size</b> | <b>Individual Runs (s)</b>     | <b>Average Runtime (s)</b> |
|---------------|---------------------|--------------------------------|----------------------------|
| CBC           | 10 campers          | 4.23, 4.13, 3.98, 4.30         | 4.16                       |
| Gurobi        | 10 campers          | 0.18, 0.20, 0.15, 0.14         | 0.17                       |
| Gurobi        | 100 campers         | 4.43, 1.49, 4.54, 11.00        | 5.37                       |
| Gurobi        | 500 campers         | 10.30, 5.09, 6.77, 8.95        | 7.77                       |
| Gurobi        | 1000 campers        | 10.11, 8.44, 6.97, 11.40       | 9.23                       |
| Gurobi        | 10 000 campers      | 28.96, 29.72, 34.34, 45.46     | 34.62                      |
| Gurobi        | 50 000 campers      | 325.31, 252.52, 294.63, 318.08 | 297.66                     |