– Independent Work Report Fall 2020 –

Optimizing Group Socialization Using a Constraint Solver

Hillel Koslowe
Adviser: Andrew Appel

1-6-21

Abstract

Constraint solvers are powerful tools that solve optimization problems given clearly-defined constraints. This project deals with one specific optimization problem – given a group of individuals, their personal availabilities, and several other constraints, find the best possible scheduling to maximize group socialization over a period of time. In this project, I dealt with two possible maximization objectives: (1) maximizing the total number of people in attendance, across all meetings, and (2) maximizing the number of unique individuals met, across all individuals. I developed a tool that produces the optimal scheduling according to both of these maximization objectives, and found that maximization objective (1) is significantly more efficient, especially for larger problems.

1. Introduction

The tool developed by this project has broad applications for many different types of scheduling problems. That being said, perhaps the best way to describe what this tool can accomplish is by describing one real-life case this project addresses: the Princeton Computer Science Department has several dozen faculty members, and they want to socialize with each other. As computer scientists, the faculty members do not merely want to socialize, but they want to socialize in the *best possible way.*

More specifically, they have decided that group lunch dates are a good way to socialize. There are several different possible lunch times each day of the week, and over the course of the semester the faculty members would like to have an optimal group lunch date scheduling.

Of course, professors have individual availability constraints (for each professor, some times and days work, and other don't), and there may be group-wide constraints as well (perhaps the group has decided that lunch dates should have between 3 and 6 people).

Given these constraints – both for the individuals and for the group – the group would like to be able to find a semester-wide schedule that produces the *best possible socialization*. Now, what do we mean by *best possible socialization*? In this paper, we will develop two models to answer this question, and each model has its advantages and disadvantages.

Obviously, this problem – a group that wants to meet in an optimal way over a period of time – has many applications beyond the Princeton Computer Science Faculty. This paper will explain not only how the tool functions, but how the tool *works* – i.e. how this specific problem uses a constraint solver to produce the optimal group scheduling.

2. Problem Background and Related Work

2.1. Constraint Solvers and Google's CP-SAT Tool

A constraint solver is a tool which takes constraints – more specifically, constraints upon variables (Boolean, Integer, Float, etc.) – and produces a feasible or optimal solution, depending on the specific problem. There are many constraint solvers; see, for example, MiniZinc [3], which compares various constraint solvers.

The constraint solver used in this project is Google's CP-SAT Tool [1], which allows for different types of variables, constraints, and optimization objectives. This tool is well-explained [1], easy-to-use, versatile, open-source, and available in Python, all of which made it appropriate for this project.

2.2. Related Work

There are *many* scheduling and broader optimization softwares available to the public. Perhaps one well-known program that performs a somewhat similar task to this project's task is when2meet.

Figure 1: when2meet – individual's availability on the left, group's availabilities on the right

However, there are two key distinction between `when2meet` and this project:

1. `when2meet` finds *one* best time, whereas this project finds an entire scheduling over a potentially large period of time.
2. `when2meet` simply maximizes (or displays) the number of availabilities, whereas this project allows for other objectives as well.

In short, `when2meet` is a one-time, simple-maximization program, whereas this project deals with long-term maximizations and allows for different objectives to be maximized.

3. Approach

Constraint solvers in general, and the Google CP-SAT Tool in particular, can only address specific problems, and require problems to be defined carefully and precisely. While humans may understand *maximize the total number of people in attendance, across all meetings*, or *maximize the number of unique individuals met, across all individuals*, these types of objectives must be translated into our constraint solver's *language* and *syntax* in order to solve our problem.

Before translating our problem from English into the constraint solver's language, let us take the intermediate step of mathematically and logically defining our problem, its

constraints, and our objectives.

3.1. Terminology and Basic Rules

There are	n individuals
over the course of	d days,
where each day has	m potential meeting times.
Each non-empty meeting must have at least	g_{min} number of people
and must have at most	g_{max} number of people.
Each individual can attend at most	1 meeting per day.

3.2. Constraints

1. Each individual can attend at most 1 meeting per day.
2. Each meeting can have p people, where $p \in \{0, g_{min}, g_{min}+1, \ldots, g_{max}-1, g_{max}\}$ — in other words, meeting slots must be empty (0 individuals) or have between g_{min} and g_{max} people.
3. Individuals can attend meetings only if they are available at the meeting time.

3.3. Optimization Goals

As stated earlier, our two seperate maximization objectives are:

1. **Total People** – maximizing the total number of people in attendance, across all meetings
2. **Total New People** – maximizing the number of unique individuals met, across all individuals

The **Total People** maximization objective is simpler to understand and implement, so we will begin with it.

3.4. Maximizing Total People

One may be tempted (as I was initially) to simply treat the problem as follows: if there are n individuals over the course of d days, where each day has m potential meeting slots, then we can create $n \times d \times m$ Boolean variables in our CP-SAT model, where each of the $n \times d \times m$ variables represents an individual's attendance (=1) or non-attendance (=0) for a specific day and time.

This approach is quite intuitive (and not *that* far off from the final implementation) – after all, we can simply create our $n \times d \times m$ Boolean variables and then apply the constraints as follows:

1. **Constraint:** Each individual can attend at most 1 meeting per day.

   Implementation: The sum of the Boolean variables (1 indicates attendance, 0 indicates non-attendance) for each day's meetings, for each individual, for each day, must be ≤ 1.
2. **Constraint:** Each meeting can have p people, where $p \in \{0, g_{min}, g_{min}+1, \ldots, g_{max}-1, g_{max}\}$ — in other words, meeting slots must be empty (0 individuals) or have between g_{min} and g_{max} people.

   Implementation: Let the sum of the Boolean variables for each meeting, for each individual, for each day be denoted by s. Then, either $s = 0$ or $g_{min} \leq s \leq g_{max}$.
3. **Constraint:** Individuals can attend meetings only if they are available at the meeting time.

   Implementation: For each meeting for each individual for each day, if the individual is unavailable, then that Boolean Variable must $= 0$.

This is *mostly* correct. However, there is one key insight that requires us to slightly modify our approach. While it may seem at first that $n \times d \times m$ Boolean variables suffices – one for each person, for each day, for each meeting time – in actuality, we require more Boolean variables. Why? Because while it is true that each individual may attend only 1 meeting per meeting time, it is also the case that *we can have more than 1 concurrent meeting during a given meeting slot.*

Before describing in mathematical terms just how many concurrent meetings we may need at a given time, let us demonstrate the need for potentially having multiple concurrent meetings at the same time with a small example. Suppose, for example, that we have a group of 20 people ($n = 20$) that wants to socialize in an optimal way over 1 day ($d = 1$) with only 1 meeting time ($m = 1$). Now, suppose also that the group has decided that meetings should only have between 3 ($g_{min} = 3$) and 5 people ($g_{max} = 5$).

The problem with our initial implementation – (only) $n \times d \times m$ Boolean variables – is that we incorrectly assume that there must only be 1 meeting per meeting slot, when in reality we may be able to maximize **Total People** by having, in our example scenario, 4 concurrent meeting, each with 5 individuals. Without the possiblity of concurrent meetings, we would be limited to only 1 meeting with 5 individuals in attendance, when in reality we would like to have 4 concurrent meetings, each with 5 individuals, thereby yielding 20 individuals in attendance.

Now that we have explained the need for potentially having concurrent meetings, let us calculate the number of possible concurrent meetings for each day. In our previous example, we were able to maximize **Total People** with 4 meetings. However, meetings can have as few as g_{min} individuals in attendance. Thus, we may require

$$\left\lceil \frac{n}{g_{min}} \right\rceil$$

concurrent meetings to guarantee that we are not *missing out* on increasing our maximization objective due to having too few possible concurrent meetings. And therefore, rather than simply having $n \times d \times m$ Boolean variables, our CP-SAT model requires

$$\left\lceil \frac{n}{g_{min}} \right\rceil \times n \times d \times m$$

Boolean variables to properly maximize **Total People**.

With this in mind, we can add a fourth constraint, as follows:

4. **Constraint:** Each individual can attend at most 1 of the concurrent meetings running at the same time.

   Implementation: The sum of the Boolean variables for each concurrent meeting, for each meeting, for each person, for each day, must be ≤ 1.

Now that we have properly set up our **Total People** model, we can define the simple objective function as follows:

$$\max \sum_{i=0}^{\lceil \frac{n}{g_{min}} \rceil \times n \times d \times m} \text{Boolean Variable}_i$$

In other words, after implementing all the constraints, we want to assign each of the $\left\lceil \frac{n}{g_{min}} \right\rceil \times n \times d \times m$ Boolean Variables either 0 or 1 such that we maximize $\sum_{i=0}^{\lceil \frac{n}{g_{min}} \rceil \times n \times d \times m} \text{Boolean Variable}_i$. This will solve for our **Total People** objective – maximizing the total number of people in attendance, across all meetings – and indeed, this is a task which our CP-SAT Solver can do.

3.5. Maximizing New People

The **Total New People** objective has many similarities to the **Total People** objective. Indeed, all the constraints for the **Total People** implementation are also neccessary for the **Total New People** implementation, since we still have the same 4 constraints (only 1 meeting per individual per day; each meeting has size constraints; individuals have availability constraints; and each individual can attend only 1 meeting within a group of concurrent meetings). However, to implement the **Total New People** implementation, we will require more Boolean variables.

To give some intuition for the need for more Boolean variables in the **Total New People** implementation, let us try to explain why the **Total New People** adds more complexity to the problem.

Total People	**Total New People**
More is more – it is *always better* to have an individual in attendance rather than not in attendance, as this will neccessarily increase the maximization objective.	*More is not neccessarily more* – if individuals A and B are already meeting at some point, then it is *no better* for them to meet a second time; after all, we do not care about total attendees, but rather maximizing individuals meeting each other.
Group/Social dynamics are of no importance – all that matters is *the individuals'* attendance or non-attendance; as such, we do not need to keep track of *internal group dynamics*.	*Group/Social dynamics are important* – while attendance or non-attendance is neccessary to keep track of, it is not the only thing to keep track of. We also must keep track of *who attends a meeting with whom*, or, more precisely, *if person* A *attends a meeting with person B*.

As indicated by the chart above, we must keep track of *who attends a meeting with whom*, or, more precisely, *if person* A *attends a meeting with person B* for **Total New People**. To do this, we can add a `hasMet` variable for every individual in the group. Mathematically, we can define our `hasMet` Boolean variables as follows:

$$\texttt{hasMet}_{i,j} = \begin{cases} 1 & i \text{ and } j \text{ have met} \\ 0 & \text{otherwise} \end{cases}$$

for $i, j \in \{0, 1, \ldots, n-1\}$, where n refers to the number of individuals in a group.

Now, we can define our objective function as follows:

$$\max \sum_{0 \le i,j \le n-1} \texttt{hasMet}_{i,j}$$

This is a more complicated objective for the CP-SAT Tool to solve than the objective

function for **Total People**; nonetheless, this objective function and implementation can be solved using our CP-SAT constraint solver Tool.

4. Implementation

Because this project is not only theoretical but also implemented as a functioning tool that performs the scheduling problem outlined in this paper, I will now demonstrate some of the more crucial parts of the actual tool, as implemented in Google's CP-SAT Tool (using Python). I will only share concise snippets of code that are either critical to the CP-SAT Tool or that directly relate to the constraints and objective functions.

4.1. Initializing the Tool

Our tool is given the following inputs: n (number of people), d (number of days), m (number of meetings per day), g_{min} (minimum number of individuals per meeting), and g_{max} (maximum number of individuals per meeting). Additionally, the tool is given each individuals' availabilities. Beyond these given inputs, the first calculation we must perform before even using our constraint solver is the *possible number of concurrent meetings* calculation. As explained earlier, the number of concurrent meetings is $\left\lceil \frac{n}{g_{min}} \right\rceil$. The following is the corresponding Python code:

```
def numberOfConcurrentMeetings(gMin, n):
  return math.ceil(n / gMin)
```

Now, our tool can create a model and a Boolean variable corresponding to each `person, day, meeting time, concurrent meeting number`. Indeed, we require these $\lceil \frac{n}{g_{min}} \rceil \times n \times d \times m$ Boolean variables for both our **Total People** and **Total New People** models. (For the **Total New People** model, we will require additional Boolean variables, as explained earlier, and as we will demonstrate later.)

To instantiate our model and create our Boolean variables, we use the following code:

```
possibleConcurrentMeetings = numberOfConcurrentMeetings(gMin, n)
all_people = range(n)
```

```
all_shifts = range(m)
all_days = range(d)
all_concurrent_meetings = range(possibleConcurrentMeetings)

model = cp_model.CpModel()

shifts = {}
for n in all_people:
  for d in all_days:
    for s in all_shifts:
      for c in all_concurrent_meetings:
        shifts[(n, d,
                s, c)] = model.NewBoolVar('shift_%i%i%i%i' % (n, d, s, c))
```

4.2. Implementating the Constraints

Now that our model is inditialized, we can begin implementing the constraints onto the Boolean variables. We will begin with the constraints that apply to both the **Total People** and **Total New People** models.

Our first constraint, which applies to both **Total People** and **Total New People**, is the following:

Constraint: Each individual can attend at most 1 meeting per day.
Implementation: The sum of the Boolean variables (1 indicates attendance, 0 indicates non-attendance) for each day's meetings, for each individual, for each day, must be ≤ 1.

To implement this constraint into our constraint solver model, we use the following lines of code:

```
for n in all_people:
  for d in all_days:
    model.Add(sum([shifts[(n, d, s, c)] for s in all_shifts for c in
                                          all_concurrent_meetings]) <= 1)
```

Our second constraint, which also applies to both **Total People** and **Total New People**, is the following:

Constraint: Each meeting can have p people, where $p \in \{0, g_{min}, g_{min}+1, \ldots, g_{max}-1, g_{max}\}$ — in other words, meeting slots must be empty (0 individuals) or have between g_{min} and g_{max} people.

Implementation: Let the sum of the Boolean variables for each meeting, for each individual, for each day be denoted by s. Then, either $s = 0$ or $g_{min} \leq s \leq g_{max}$.

To implement this constraint into our constraint solver model, we use the following lines of code:

```
possibleValues = []
possibleValues.append([0])
for i in range(gMin, gMax + 1, 1):
  possibleValues.append([i])

for d in all_days:
  for s in all_shifts:
    for c in all_concurrent_meetings:
      thisSum = model.NewIntVar(0, gMax, 'sum%i%i%i' % (d, s, c))
      model.Add(thisSum == sum(shifts[(n, d, s, c)] for n in all_people))
      model.AddAllowedAssignments([thisSum], possibleValues)
```

Our third constraint, and the final constraint which applies to both **Total People** and **Total New People**, is the following:

Constraint: Individuals can attend meetings only if they are available at the meeting time.

Implementation: For each meeting for each individual for each day, if the individual is unavailable, then that Boolean Variable must $= 0$.

To implement this constraint into our constraint solver model, we use the following lines of code:

```
for n in all_people:
  for d in all_days:
    for s in all_shifts:
      for c in all_concurrent_meetings:
        if specificT[n][d][s] == 0:
          model.Add(shifts[(n, d, s, c)] == 0)
```

This code relies on a `specificT` list, a required input into our model which contains each individuals' availabilities for each shift for each day. (`specificT` is stored internally as an $n \times d \times m$ Python list for indexing convenience.)

4.3. Solving for Objective #1 – Total People

After implementing the 3 constraints above, all that is left for the **Total People** model is to actually solve for our objective function, given all the constraints. This is a computationally difficult task, but one which the Google CP-SAT Tool can solve in only a few lines of code.

As justified earlier, our maximization objective is

$$\max \sum_{i=0}^{\lceil \frac{n}{g_{min}} \rceil \times n \times d \times m} \text{Boolean Variable}_i$$

This is implemented in our code as follows:

```
model.Maximize(
      sum(shifts[(n, d, s, c)] for n in all_people
          for d in all_days for s in all_shifts
          for c in all_concurrent_meetings))
```

4.4. Implementing the Boolean Variables for Objective #2 – Total New People

In addition to the constraints already implemented, our **Total New People** model requires additional Boolean variables and constraints. Because the **Total New People** model maximizes the number of unique individuals met, across all individuals, we need to create `hasMet` Boolean variables for each individual-pairing as follows:

$$\texttt{hasMet}_{i,j} = \begin{cases} 1 & i \text{ and } j \text{ have met} \\ 0 & \text{otherwise} \end{cases}$$

for $i, j \in \{0, 1, \ldots, n-1\}$, where n refers to the number of individuals in a group.

To add these `hasMet` variables to our **Total New People** model, we use the following code:

```
aHasMetB = {}
for i in all_people:
  for j in range(i + 1, people, 1):
    aHasMetB[(i, j)] = model.NewIntVar(0, 1, 'met_%ii%ij' % (i, j))
```

Now that we have created a `hasMet` variable for each pair of individuals, we need the `hasMet` variables to indicate what they are set out to do: namely, to equal 1 if the individuals have met, and 0 otherwise.

To do this, we require a new set of Boolean variables. Not only do we require $\texttt{hasMet}_{i,j}$ variables, but we also require $\texttt{hasMet}_{d,s,c,i,j}$ variables, which indicate whether individual i and individual j meet at day d, shift s, concurrent meeting c. Once we have these variables instantiated, we can then set

$$\texttt{hasMet}_{i,j} = \bigvee \texttt{hasMet}_{d,s,c,i,j}$$

for each individual pair i and j, across all d, s, and c, representing days, shifts, and concurrent meetings, respectively. Applying $\bigvee$, or logical disjunction (**OR**), across all possible meetings between individuals i and j effectively sets $\texttt{hasMet}_{i,j} = 1$ if they meet *at least once*, and only if they never meet will $\texttt{hasMet}_{i,j} = 0$.

To implement this process into our CP-SAT model, we use the following code:

```
for d in all_days:
  for s in all_shifts:
    for c in all_concurrent_meetings:
```

```
      for i in all_people:
        for j in range(i + 1, people, 1):
          aHasMetB[(d, s, c, i, j)] = model.NewIntVar(0, 1,
                                      'met_%i%i%i%i%i' % (d, s, c, i, j))
          model.AddMultiplicationEquality(aHasMetB[(d, s, c, i, j)],
                                          [shifts[(i, d, s, c)],
                                          shifts[(j, d, s, c)]])

for i in all_people:
  for j in range(i + 1, people, 1):
      model.AddMaxEquality(aHasMetB[(i, j)], [aHasMetB[(d, s, c, i, j)]
                           for d in all_days for s in all_shifts
                           for c in all_concurrent_meetings])
```

4.5. Solving for Objective #2 – Total New People

Now that we have implemented our Boolean variables for **Total New People**, we need to solve for our maximization objective, which we previously defined as

$$\max \sum_{0 \leq i,j \leq n-1} \texttt{hasMet}_{i,j}$$

We implement this maximization objective as follows:

```
model.Maximize(
    sum(aHasMetB[i, j] for i in all_people
                       for j in range(i + 1, people, 1)))
```

5. Results and Evaluation

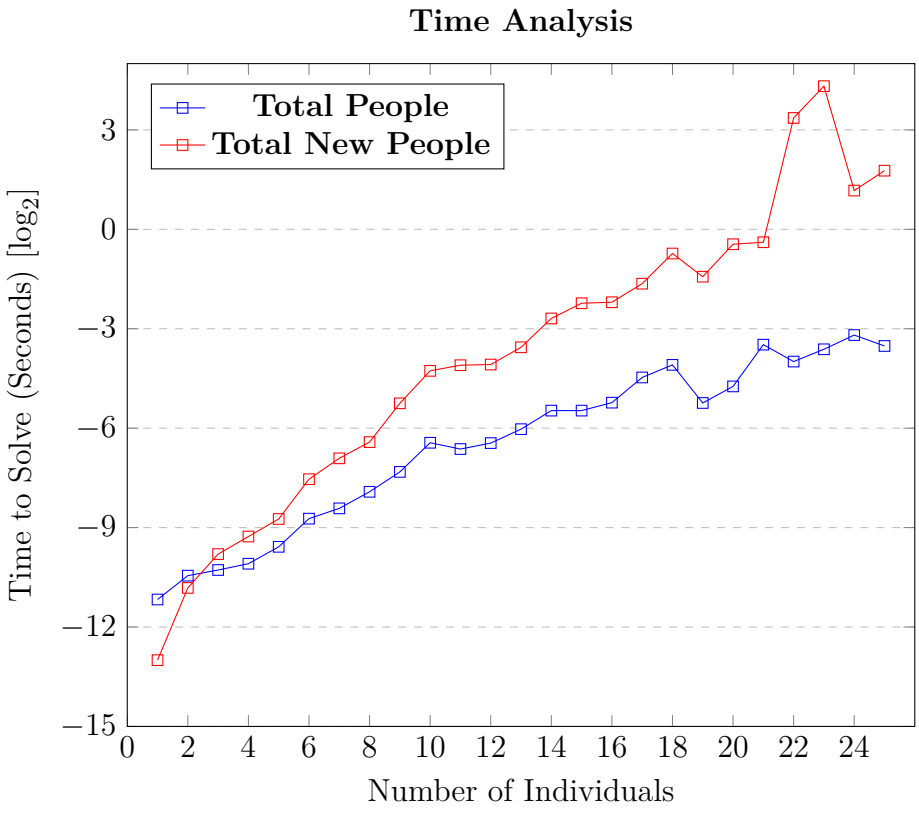
5.1. Time Analysis

Now that we have demonstrated mathematically and techincally how to solve our socialization problem for the **Total People** and **Total New People** objectives, let us analyze and compare

the two different models.

Before we begin a proper analysis of our two models, we will demonstrate the difference between the two models' performances using a graph. Then, we will not only explain the trends in the graph but explain *why* the plots are as they are.

The graph below was generated by randomly generating each individuals' availabilities. This particular graph has n (number of individuals) ranging from 1 to 25, and assumes throughout that $d = 5$ $m = 2$, $g_{min} = 4$, and $g_{max} = 15$.



Because the y-axis is plotted on a log_2 scale, it is clear from the graph that both models are *exponential growth*. This should come as no surprise, since our project is based on Google's CP-SAT Tool, and the general SAT problem is an **NP**-complete problem, with known algorithms taking exponential time [2].

5.2. Number of Boolean Variables for Each Model

To explain why **Total New People** takes longer to solve, let us compare the number of Boolean variables each model uses. The purposes for the Boolean variables have already been explained, and the number of Boolean variables is clear from the Python code and the corresponding mathematical analysis; however, we present the number of Boolean variables here in a concise and organized manner.

For the **Total People** model, there is only 1 *batch* of Boolean variables.

Purpose: Attendance or non-attendance for each concurrent meeting for each shift for each day for each individual.
Number of Boolean Variables: $\left\lceil \frac{n}{g_{min}} \right\rceil \times n \times d \times m$

However, for the **Total New People** model, there are two more *batches* of Boolean variables required.

Purpose: Have individuals i and j met at a specific day, shift, concurrent meeting slot?
Number of Boolean Variables: $d \times m \times \left\lceil \frac{n}{g_{min}} \right\rceil \times \frac{n(n+1)}{2}$

Purpose: Have individuals i and j met?
Number of Boolean Variables: $\frac{n(n+1)}{2}$

Indeed, there are significantly more Boolean variables required for the **Total New People** model, which accounts for the larger solving time. And because of the exponential order-of-growth, the **Total New People** model becomes impractical to use for larger groups, whereas the **Total People** model is still practical for large groups.

5.3. Using the Tool

While both models can be used via Python scripts, I have also developed a basic script that converts a `.csv` / Microsoft Excel file into the desired scheduling.

For example, the following could be the input file, describing the group's preferences and individuals' availabilities.

Number of People	Number of Days	Meetings Per Day	Minimum Group Size	Maximum Group Size						
7	5	2	2	6						
Name	11/2/2020		11/3/2020		11/4/2020		11/5/2020		11/6/2020	
	12:00-1:00pm	12:30-1:30pm	12:00-1:00pm	12:30-1:30pm	12:00-1:00pm	12:30-1:30pm	12:00-1:00pm	12:30-1:30pm	12:00-1:00pm	12:30-1:30pm
1 Aarti Gupta	1	1	1	1		1		1		1
2 Alan Kaplan		1			1	1	1			
3 Andrew Appel										
4 Arvind Narayanan	1	1	1	1	1					1
5 Ben Raphael					1	1	1	1		1
6 Brian Kernighan		1		1			1		1	
7 Edward Felten			1							

Figure 2: Group Preferences and Individuals' Availabilities

Then, the following could be the two output files.

Date	Time	Aarti Gupta	Alan Kaplan	Andrew Appel	Arvind Narayanan	Ben Raphael	Brian Kernighan	Edward Felten
11/2/2020	12:30-1:30pm	1					1	
11/2/2020	12:30-1:30pm		1		1			
11/3/2020	12:00-1:00pm	1						1
11/3/2020	12:30-1:30pm				1		1	
11/4/2020	12:00-1:00pm				1	1		
11/4/2020	12:30-1:30pm	1	1					
11/5/2020	12:00-1:00pm		1				1	
11/5/2020	12:30-1:30pm	1				1		
11/6/2020	12:30-1:30pm	1			1	1		

Figure 3: Total People Scheduling

Date	Time	Aarti Gupta	Alan Kaplan	Andrew Appel	Arvind Narayanan	Ben Raphael	Brian Kernighan	Edward Felten
11/2/2020	12:30-1:30pm	1			1		1	
11/3/2020	12:00-1:00pm	1			1			1
11/4/2020	12:30-1:30pm	1				1		
11/5/2020	12:00-1:00pm		1			1		

Figure 4: Total New People Scheduling

6. Conclusion and Future Work

In this project, we studied a well-defined scheduling problem: finding the best long-term scheduling for a group of individuals who want to socialize in the optimal way. We defined the *best* way to socialize in two different ways: (1) maximizing the total number of individuals in attendance over the entire period of time, and (2) maximizing the total number of new individuals met over the entire period of time.

Not only were we able to mathematically define these problems, but we were also able to produce a working tool that produces actual schedules for these two optimizations using Google's CP-SAT Tool in Python.

And beyond solving these problems, we analyzed and explained their run-times. Because **Total New People** has a larger run-time, it is impractical to use for larger groups, whereas **Total People** is a more scalable maximization objective.

One way to improve this tool is to produce a more user-friendly interface. While the `.csv` / Microsoft Excel tool is *better* than a mere Python script, it still requires a Python script to run, which requires users to either (1) send their `.csv` / Microsoft Excel file with their group's specifications to someone who can run the program, or (2) download the program and its dependencies. Perhaps, an interface similar to `when2meet`'s can be created, which would allow this tool to be more user-friendly for a wider audience.

7. Acknowledgements

I would like to thank Professor Andrew Appel for his guidance and support. Professor Appel's Functional Programming course introduced me to the intersection between Computer Science and Formal Logic, which sparked my interest in not only this project, but also in other projects and coursework as well.

Professor Appel's ability to distill complicated problems down to their core, combined with his lighthearted nature, made working with him not only interesting and enlightening but also an absolute pleasure.

I would also like to thank my family and friends for allowing me to share my progress with them throughout the semester, listening to my ideas, offering suggestions and improvements, and helping me communicate my problem in everyday English.

References

[1] "Constraint optimization." [Online]. Available: https://developers.google.com/optimization/cp

[2] "Implications of the exponential time hypothesis." [Online]. Available: http://cseweb.ucsd.edu/~adhayal/researchexam.pdf

[3] "The minizinc challenge." [Online]. Available: https://www.minizinc.org/challenge.html