Returnforce

Anurag Baddam

May 2020

1 Introduction

As the COVID-19 situation eases over the coming months, Salesforce, like so many other companies, will have to figure out how to bring our Ohana back to the offices in a safe manner that encourages social distancing. The goal of Returnforce was to build a tool that assigns each employee to a rotational day or shift that that they should attend the office to optimize for the physical distance between the desks of the employees that will be present in the office at the same time. To do this, I've created a linear program that assigns each employee to one of 4 days to attend the office. The solution maximizes the physical distance between the desks of the employees that will be at the office on any given day while ensuring that 6 feet of distance between employees working at their desks will always be maintained. Note that the algorithms in this paper are an initial pass and need to be further investigated and improved.

2 Technical Details

The linear program used to calculate the optimal seating arrangement is detailed in this section.

2.1 Configuration

All of the configuration variables used in the LP along with their values in the initial run of the program are listed below.

Variable	**Initial Value**	**Explanation**
NUM_SEATS	12	Total number of seats
ROW_SIZE	3	Seats per row
AISLE_FUDGE_FACTOR	5	An extra distance (in feet) applied to capture the distance b/w aisles
SEAT_DISTANCE	3	Distance (in feet) from one seat to an adjacent seat in same row/column
NUM_ROTATION_DAYS	4	Max number of days to partition across

2.2 Variables

The input to the LP is the matrix d, which is used to capture seat distances, where d_{ij} represents the Euclidean distance between seats i and j. This is all calculated prior to running the LP.

The LP is trying to solve for the matrix w, where w_{ij} being set to 1 indicates that seats i and j are assigned to the same day and 0 indicates that they are not. Each variable in the w matrix is of

type BINARY, so the LP will only ever set them to 0 or 1.

$$\begin{aligned} & d_{ij} \\ w_{ij} \in \{0,1\} & \end{aligned} \tag{1}$$

2.3 Objective function

The objective function that the linear program is trying to maximize is:

$$\max \sum_{ij,j>i} w_{ij} * d_{ij}$$

This will ensure that we are maximizing the total physical distance between all desks that are assigned to the same day.

2.4 Constraints

All of the constraints to the LP are detailed below.

2.4.1 Six feet constraint

In order to follow CDC guidelines, the LP will ensure that no two seats that are less than 6 feet apart are assigned to the same day. The is captured with the below constraint.

$$\forall_{ij,i!=j,d_{ij}<(6/\text{SEAT\_DISTANCE})} w_{ij} = 0$$

2.4.2 Total Number of Days Constraint

The maximum number of days to partition employees/seats across is set in the NUM_ROTATION_DAYS variable. If this variable did not exist, the LP would trivially assign each employee to a separate day, but this is unlikely to be practical. In order to enforce that the LP does not partition across too many days, we use the below methodology.

To enforce this constraint, we'll need a way to count the total number of days in any given assignment of w. First, we'll define a set of binary variables n where n_i will be set to 1 if seat i introduces a "new day", meaning that it has been assigned to a different day than any seat in the set $\{0, 1, ..., i-1\}$. Otherwise, if it is assigned to a day that has already been counted, not a new day, n_i will be set to 0. To do the counting, we'll store an array ss, where $ss_i = \sum_{j=0,1,...,i-1} w_{ji}/(i+1)$. Here, we are summing the weight of day i with all of the previous days and normalizing it over it's maximum value of $i+1$. We can now say that $n_i = \lfloor 1 - ss_i \rfloor$. This is because if ss_i is 0, that means none of the previous days matched day i so this is a new day and n_i will be 1. If ss_i is greater than 0, atleast one of the previous days did match, so n_i will be 0. Thus, we have the following variables:

$$\begin{aligned} & n_i \in \{0,1\} \\ ss_i = & \left(\sum_{j=0,1,...,i-1} w_{ji} \right) /(i+1) \end{aligned} \tag{2}$$

And the following constraints (note how we capture the *floor* function):

$$\begin{aligned} n_i &\geq (1 - ss_i) - .999 \\ n_i &\leq 1 - ss_i \\ \left(\sum_i n_i \right) &\leq \text{NUM\_ROTATION\_DAYS} \end{aligned} \tag{3}$$

2.4.3 Same Day as Yourself Constraint

Since you will trivially be assigned to the same day as yourself, we have the below constraint:

$$\forall_i w_{ii} = 1$$

2.4.4 Commutative Property

If seats i and j are on the same day, then seats j and i are also on the same day by the commutative property, giving us the below constraint:

$$\forall_{ij} w_{ij} = w_{ji}$$

2.4.5 Transitive Property

Due to transitive property, w_{ij} and w_{ik} can directly impact the value of w_{jk} as shown in the truth table below.

w_{ij}	w_{ik}	w_{jk}
0	0	$0 \cup 1$
0	1	0
1	0	0
1	1	1

For example, if i and j are on the same day, and i and k are on the same day, j and k must also be on the same day. The relationship in this truth table can be encoded with the following constraints:

$$\begin{aligned} w_{jk} &\geq w_{ij} + w_{ik} - 1 \\ w_{jk} &\leq |w_{ij} + w_{ik} - 1| \end{aligned} \tag{4}$$

The reason this works is best illustrated by expanding the truth table from above to include the effective forcing function of the constraint for each pair of values.

w_{ij}	w_{ik}	$w_{jk} \geq w_{ij} + w_{ik} - 1$	$w_{jk} \leq \lvert w_{ij} + w_{ik} - 1 \rvert$	w_{jk}
0	0	$w_{jk} \geq -1$	$w_{jk} \leq 1$	$0 \cup 1$
0	1	$w_{jk} \geq 0$	$w_{jk} \leq 0$	0
1	0	$w_{jk} \geq 0$	$w_{jk} \leq 0$	0
1	1	$w_{jk} \geq 1$	$w_{jk} \leq 1$	1

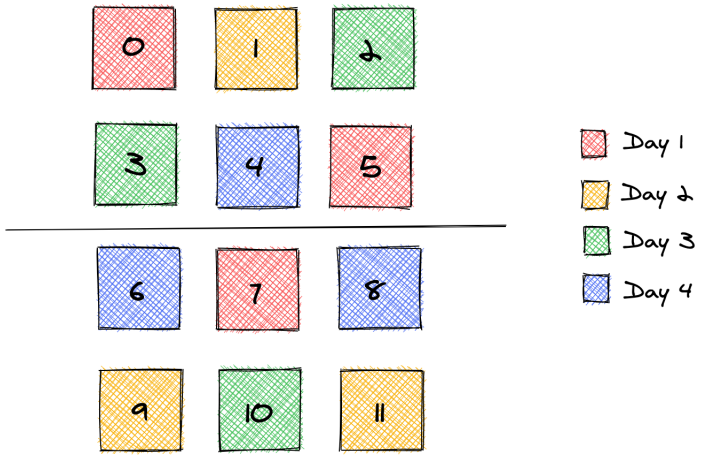
The final column in this truth table matches the final column in the truth above.
In order to represent an absolute value in a linear program, we introduce helper variables $t1$ and $t2$ to formulate the below constraints:

$$\begin{aligned} w_{jk} &\leq t1_{ijk} + t2_{ijk} \\ t1_{ijk} - t2_{ijk} &= w_{ij} + w_{ik} - 1 \\ t1_{ijk} &\geq 0 \end{aligned} \tag{5}$$

Essentially, $t1$ will capture the positive portion of $w_{ij} + w_{ik} - 1$ and $t2$ will capture the negative portion so adding them together works functionally as an absolute value operation.

3 Solution

Running the LP with the initial values given at the top of the paper partitioned the 12 seats (2 aisles) across 4 days in the following way:



The objective value was 66.7 feet of total social distancing. We cam see that 6 feet of distance is always maintained between any two seats assigned to the same day (assuming 3 feet b/w adjacent seats).

4 Future Work

- Integrate this feature with Google Calendar so that employees can easily see on what days they should attend the office.
- Test running the LP on more seats. Will it scale to all of the seats on a floor?
- How do we interpret the objective value? Could it help us decide the ideal number of shifts/days to partition across?

5 References

- Returnforce code
- Returnforce Demo Video