MODELS OF OPERATIONS RESEARCH

PROJECT GROUP 1

DEFAULT PROJECT - WORKFORCE MANAGEMENT

(WINTER - 2020)

PRESENTED BY-
SONAL BHOR (88659089)
HARSHITA SHIVARAMAKRISHNA (16365797)
SANJYOT THETE (48844546)
ADHEETH HUNDI (36920130)
SUMANTH POBALA (71652304)

INDEX

1. INTRODUCTION

Employee absenteeism is a significant concern for the UMD manufacturing company to maintain its assembly line. Because of unpredictable absenteeism, the UMD is facing a complex workforce management problem. Since the assembly line seriously understaffed has to be shut down, the UMD has to hire much more employees than needed actually. However, overstaffing wastes labor forces and charges higher labor costs. The UMD is currently engaged in multiple research projects for better understanding employee absenteeism and optimizing workforce planning and scheduling strategies. (Ref: Default Project PDF)

In order to successfully implement the scheduling strategy, the problem has been assigned to the below constraints:

(b) *Assignment priority.* It is an emergency if there are not enough showing-up employees to run all the workstations in the assembly line on a certain day. The manager can disobey all the rules below ((b) ~ (f)) to keep the assembly line running. For the emergency case, the manager could not assign employees to the positions of production leaders.

(c) *Team independence and collaboration.* Each team first assigns the members to complete its own tasks. If there are employees more than needed, unassigned ones can be lent to other teams.

(d) *Assignment preference.* If there are enough showing-up employees to run the assembly line, the manager would like to make an assignment according to his/her preferences given in the file all_team_prefer.xlsx.

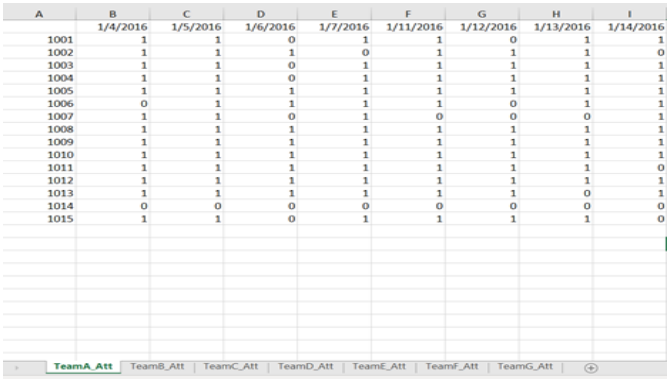
(e) *Go-home policy.* Every showing-up employee is fully paid $320 per day and absentee is not paid. If showing-up employees are more than needed, the manager could ask some unassigned employees to go home in order to reduce labor cost. The go-home employees are paid $160 that day.

(f) *Cross-training policy.* Cross-training is an alternative to the go-home policy. Cross-training can increase employees' versatilities so that more workstations can be operated by single skilled employees in the future assignment. The manager may prefer to provide cross-training courses for unassigned employees rather than send them back to home. A cross-training course requires a trainee to work with a skilled trainer for two days to learn a new job. A trainee will also be fully paid.

The UMD assembly line consists of seven sections (A ~ G), and each section is operated by a team. The data of daily operations are given in the three EXCEL documents -all_team_att.xlsx, all_team_skills.xlsx, and all_team_prior.xlsx.

Employee versatility. The file all_team_skills.xlsx tells the employees' versatile skills. Each workstation can be run either by an employee himself/herself who is labeled "1", or by two employees labeled "0" working together.

(all_team_att.xlsx): This EXCEL file records employees' daily attendances from 01/04/2016 to 09/05/2018. Workdays in a week are from Monday to Thursday.



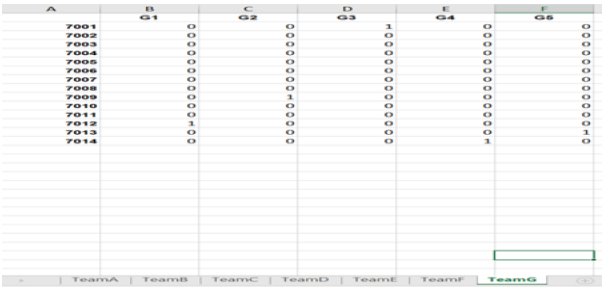
	1/4/2016	1/5/2016	1/6/2016	1/7/2016	1/11/2016	1/12/2016	1/13/2016	1/14/2016
1001	1	1	0	1	1	0	1	1
1002	1	1	1	0	1	1	1	0
1003	1	1	0	1	1	1	1	1
1004	1	1	0	1	1	1	1	1
1005	1	1	1	1	1	1	1	1
1006	0	1	1	1	1	0	1	1
1007	1	1	0	1	0	0	0	1
1008	1	1	1	1	1	1	1	1
1009	1	1	1	1	1	1	1	1
1010	1	1	1	1	1	1	1	1
1011	1	1	1	1	1	1	1	0
1012	1	1	1	1	1	1	1	1
1013	1	1	1	1	1	1	0	1
1014	0	0	0	0	0	0	0	0
1015	1	1	0	1	1	1	1	0

TeamA_Att | TeamB_Att | TeamC_Att | TeamD_Att | TeamE_Att | TeamF_Att | TeamG_Att

Figure 1: all_team_att.xlsx file

- Row names are employee IDs
- Column names are dates.
- If an employee shows up before the starting time 7:00 a.m. on a certain day, s/he is assigned a value "1", otherwise "0".

(all_team_prefer.xlsx): This EXCEL file gives a plant manager' preference for job assignments.



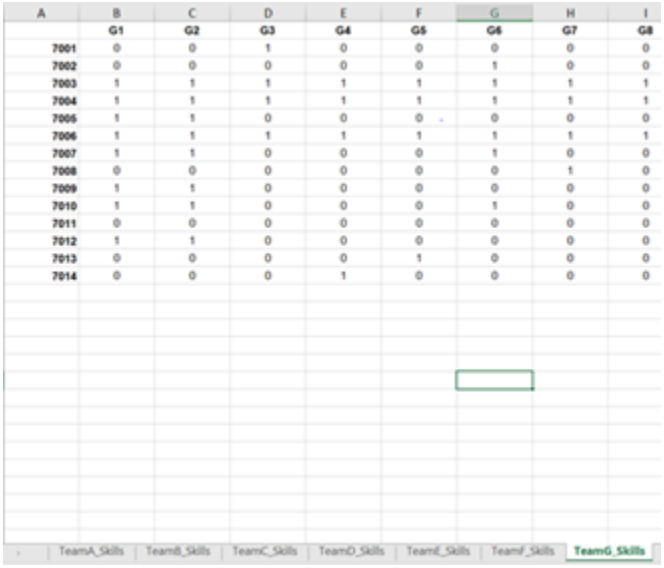
	G1	G2	G3	G4	G5
7001	0	0	1	0	0
7002	0	0	0	0	0
7003	0	0	0	0	0
7004	0	0	0	0	0
7005	0	0	0	0	0
7006	0	0	0	0	0
7007	0	0	0	0	0
7008	0	0	0	0	0
7009	0	1	0	0	0
7010	0	0	0	0	0
7011	0	0	0	0	0
7012	1	0	0	0	0
7013	0	0	0	0	1
7014	0	0	0	1	0

TeamA | TeamB | TeamC | TeamD | TeamE | TeamF | TeamG

Figure 2:all_team_prefer.xlsx

- Row names are employee IDs
- Column names are workstation titles.
- The plant manager prefers to assign employees to workstations valued "1" if possible.

(all_team_skills.xlsx): This EXCEL file gives intra-team versatility matrices.



	G1	G2	G3	G4	G5	G6	G7	G8
7001	0	0	1	0	0	0	0	0
7002	0	0	0	0	0	1	0	0
7003	1	1	1	1	1	1	1	1
7004	1	1	1	1	1	1	1	1
7005	1	1	0	0	0	0	0	0
7006	1	1	1	1	1	1	1	1
7007	1	1	0	0	0	1	0	0
7008	0	0	0	0	0	0	1	0
7009	1	1	0	0	0	0	0	0
7010	1	1	0	0	0	1	0	0
7011	0	0	0	0	0	0	0	0
7012	1	1	0	0	0	0	0	0
7013	0	0	0	0	1	0	0	0
7014	0	0	0	1	0	0	0	0

TeamA_Skills | TeamB_Skills | TeamC_Skills | TeamD_Skills | TeamE_Skills | TeamF_Skills | TeamG_Skills

Figure 3: all_team_skills.xlsx

- Row names are employee IDs
- Column names are workstation titles. For example, A1 is the first workstation in Section A of the assembly line.
- If an employee is capable of running a workstation independently, s/he is assigned a value “1”for this workstation, otherwise “0”.
- The last two columns in each sheet, filled in yellow, are production leader and tag relief, respectively. A production leader and tag relief should be able to operate all workstations in their section. The production leader is responsible for the operation of the section, monitoring any issues and ensuring production targets. The tag relief exchanges another employee when s/he needs a break.

2. PROBLEM STATEMENT - OUR INTERPRETATION

We observe the given problem as a composition of two optimization problems.

First problem deals with the skills of people and how that information can be leveraged to make an assignment of employees to the workstation which reduces the overall working cost, while ensuring that the operations are not shut down.

Second problem deals with the preferences of the manager to appoint specific employees to specific workstations. This objective is only considered if we are able to find enough employees based on skill matching to run a teams' operations.

A separate, third phase, of the workforce management problem deals with approaches which deal with the scenario where we have an excess of employees. We consider two approaches, exclusive to each other. First approach is a greedy one where we simply send the excess employees back, while still paying them half the working days wage. Second approach is a more long term approach, where we re-skill employees over time, leading to better workstation assignments in future.

In our interpretation, we do not consider the case where a team manager can interfere if we face a shortage of employees in a team on a particular day. We discuss the formulation of the case where a team with excess employees can share them with a team which needs more employees, in the future work section.

3. DEVELOPMENT ENVIRONMENT

Programming Language: Python
All our modelling is done with Python 3.

Execution IDE: Jupyter Lab and Google Colab
We developed the code in Jupyter Lab and Google Colab. Google Colab helped us collaborate over different parts of the problem simultaneously.

Open source packages:

1. **PuLP:** PuLP is an open-source linear programming (LP) package and comes packaged with many industry-standard solvers. It also integrates nicely with a range of open source and commercial LP solvers.
2. **Pandas:** Pandas is used for all the data reading, data manipulation and storing states of optimization results.
3. **Seaborn/Matplotlib**: Seaborn (with Matplotlib backend) is used to produce all the plots in this report.

We used Anaconda as our package manager.
To reproduce our results, our Google Colab notebook can be directly run here. It handles the installation of PuLP at the beginning and all other packages are already installed by default.

4. THE MODEL - MAIN PIPELINE

We are able to split the problem statement into four independent sub-problems, which we could distribute amongst ourselves and work parallelly. We will call these sub-problems, 4 stages, over which we model the complete problem statement. We also include an additional stage where we discuss the future work.

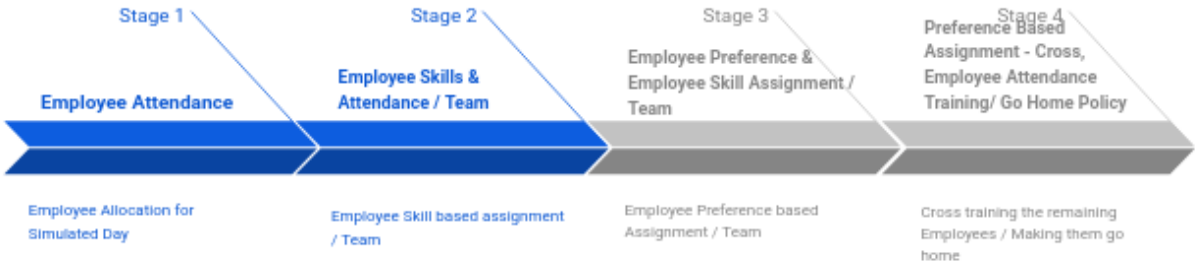


Figure 4: Formulated Model

1. Employee Attendance
   This stage deals with utilizing the historic attendance data to get attendance distribution for each employee and then simulate their attendance for new, hypothetical days.

2. Skill-based assignment
   This stage handles the problem of assigning employees to workstations taking into account only the constraint of employee skills.

3. Preference-based assignment
   This stage improves upon the previous stage by incorporating manager preference into the LP modelling.

4. Cross-training/Go-home policy
   This stage manages the simulation of the two policies we have to deal with excess employees.

5. A Future work based on Inter Team Resource Sharing
   In this stage we discuss the approach to improving this modelling pipeline by formulating how excess employees from one team can be lent to a team with a shortage.

4.1. Employee Attendance

We are provided with the historical attendance of all the employees. This data can provide us an indication of how likely it is that an employee is present on any given day. We will assume that the probability of an employee being present on a given day is equal to the historical average of the employee's attendance.

We can run Monte Carlo simulation with these probabilities to simulate the employee presence. For each run of the simulation, we obtain a sample from this distribution, that indicates which employees are present on the current iteration. All other optimization stages downstream of this stage will utilize the samples we obtain from this stage.

With the data collected from multiple runs, we can come up with useful metrics for judging the health of our system, like the number of times an emergency situation was reached, number of times we are able to assign employees to their preference, etc.

Our pipeline for this stage looks like this:

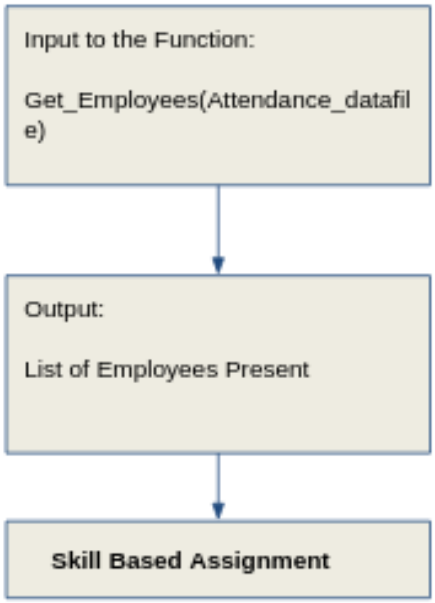


Figure 5

The code for generating attendance is shown in Figure 6.

```
# This function inputs the attendance sheet of current team, returns the employees who will be coming the next day.
def get_employees(att_df):

    att_avg = []
    emp_att = []

    for i in range(len(att_df)):
        att_avg += [((att_df.iloc[i,:].sum() / len(att_df.iloc[i,:]))*100).round(2)]   # Calculating the Average Attendance

    emp_dict = dict(zip(att_df.index, att_avg)) # Dictionary of employees and Probabilities

    for key in emp_dict:

        # Generating 100 random samples of Attendance according to Probabilities.
        one_count = int(round(emp_dict.get(key)))
        zero_count = 100 - one_count
        my_list = [0]*zero_count + [1]*one_count
        random.shuffle(my_list)
        emp_att += [my_list[0]]

    d = datetime.datetime.today()
    att_df[d] = emp_att      # Saving to dataframe by Date and time.
    emp_present = att_df.iloc[:,-1] # getting the last column generated.

    return emp_present
```

Figure 6: Employee attendance generation

```
employee_present = validate_employees(att,skills)
print(employee_present[employee_present == 1].index.to_list())
print('Total Number of Employees Present:',employee_present.sum())  # Total employees present
```

```
[3001, 3002, 3003, 3005, 3006, 3007, 3008, 3009, 3010, 3011, 3012, 3013, 3014, 3015, 3016, 3017]
Total Number of Employees Present: 16
```

Figure 7: Employee Allocation Results

Figure 7 shows the output of this function, where we have a list of 16 employees who are present on a given day out of a total of 17 employees from team C.

4.2. Skill-based Assignment

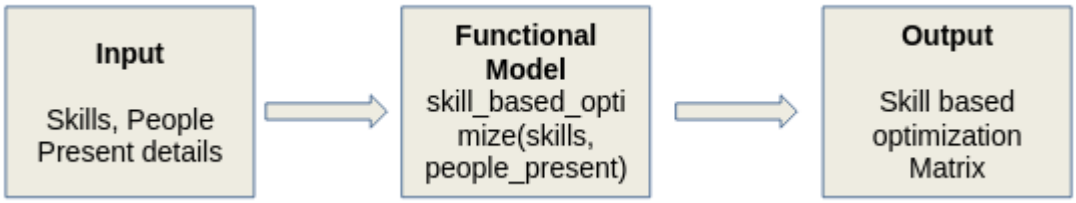


Figure 8: Skill based assignment workflow

As an input to this stage, we have complete information on which employees are present on this day. From the problem, we also have the employee skill set for each of the workstations.

Objective

In this stage, we attempt to minimize the total number of employees who are assigned to the workstations, while still keeping the assembly line working. We do that by defining an assignment matrix (which will be the parameter we optimize in this stage), where if an employee 'i' is assigned to the workstation 'j', we set the value Aij as '1', or '0' otherwise. Hence, the objective is,

$$\textbf{Minimize } \Sigma i \, \Sigma j \, Aij$$

Constraints

For the purpose of assigning either one skilled or two unskilled employees, we make a new skill matrix (as opposed to the one given in the problem) where employees who are skilled in a workstation have a value of *'1'*, otherwise the value is *'0.5'*.

Hence, if we constrain the sum of this new skill matrix per column(workstation) to **exactly** 1, we allow only two scenarios:

1. One skilled employee (1 employee with skill value '1')
2. Two unskilled employees (2 employees with skill value '0.5')

If S is the new skill matrix,

Sij = 1 if employee ‘i’ is skilled at workstation ‘j’

= 0.5 otherwise

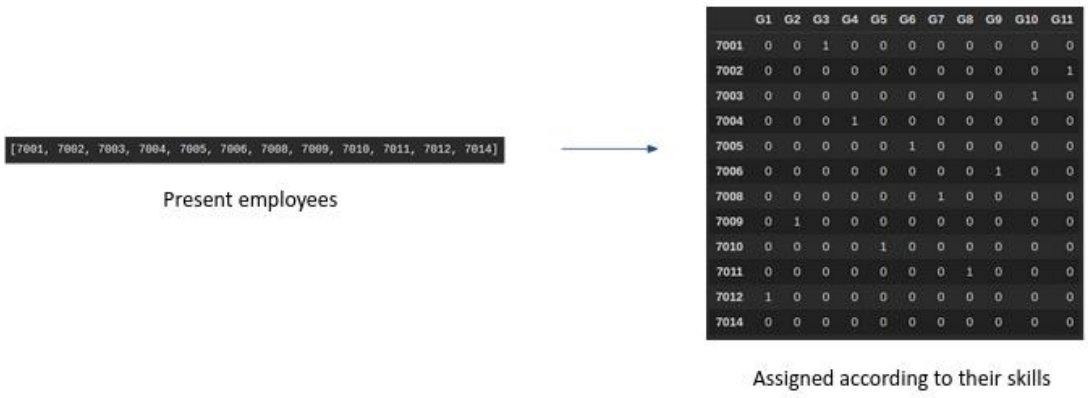
This constraint is then simply equating the sum of the product of employee assignment and the skill value of employee-workstation to exactly ‘1’.

Σi Aij * Sij = 1 for all workstations ‘j’

In this model the input is the Employees skills and the Employee attendance details from stage 1. We pass this into the function *skill_based_optimize( )* where it maximizes the objective and outputs a Skill Based optimization matrix.

```
def skill_based_optimize(skills, people_present):
    skills_present = skills[skills.index.isin(people_present)]
    skills_present = skills_present.applymap(lambda x: 0.5 if x == 0 else x)

    workstations = skills_present.columns.to_list()
    employees = skills_present.index.to_list()

    prob = LpProblem("assignment", LpMinimize)  # Minimization problem
    assignment = LpVariable.dicts('Assign', (employees, workstations), cat='Binary')  # Each value of matrix is a binary variable
    prob += lpSum([assignment[emp][wrk] for emp in employees for wrk in workstations])  # Objective, which is sum of assignment matrix

    for workstation in workstations:
        # for each workstation, sum(assignment * skill) over all employees = 1
        prob += lpSum([assignment[employee][workstation] * skills_present.loc[employee, workstation] for employee in employees]) == 1

    for employee in employees:
        # Each employee can be assigned at max 1 workstation
        prob += lpSum([assignment[employee][workstation] for workstation in workstations]) <= 1

    prob.solve()
    solve_status = LpStatus[prob.status]

    if solve_status == 'Optimal':
        # Do something
        result = pd.DataFrame(assignment).applymap(lambda x: x.varValue).T.astype(int)
        people_assigned = value(prob.objective)
        excess_people = result.sum(axis=1).loc[lambda x: x == 0].index.to_list()
        return True, result, excess_people  # First element tells if solution was found
    else:
        # Do something else
        return False, None, None
```

Figure 9: Skill Based assignment code



	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11
7001	0	0	1	0	0	0	0	0	0	0	0
7002	0	0	0	0	0	0	0	0	0	0	1
7003	0	0	0	0	0	0	0	0	0	1	0
7004	0	0	0	1	0	0	0	0	0	0	0
7005	0	0	0	0	0	1	0	0	0	0	0
7006	0	0	0	0	0	0	0	0	1	0	0
7008	0	0	0	0	0	0	1	0	0	0	0
7009	0	1	0	0	0	0	0	0	0	0	0
7010	0	0	0	0	1	0	0	0	0	0	0
7011	0	0	0	0	0	0	0	1	0	0	0
7012	1	0	0	0	0	0	0	0	0	0	0
7014	0	0	0	0	0	0	0	0	0	0	0

Figure 10: Skill Based Assignment Result matrix

4.3. Preference Based Assignment

As an input, we have skill-based assignment of employees from the second stage. We also have the manager's preference of employees to workstations, denoted by Pij (which is '1' or '0') where the employee 'i' is preferred for the workstation 'j'.

Objective

We attempt to maximize the sum of correct preferences matched, which is defined as the sum of the product of preference and actual assignment. That is,

$$\textit{Maximize } \Sigma i \, \Sigma j \;\; Pij * Aij$$

Constraints

This optimization is subjected to all the constraints we had for the skill-based assignment.

In addition, to conserve the number of employees that were finalized at the end of stage 2, We note the total employees assigned in each team at the end of stage 2 (obtained by Σi Σj Aij) by 'T' and we add the constraint,

$$\Sigma i \, \Sigma j \; Aij \; = T$$

Here is the overall workflow for this stage.

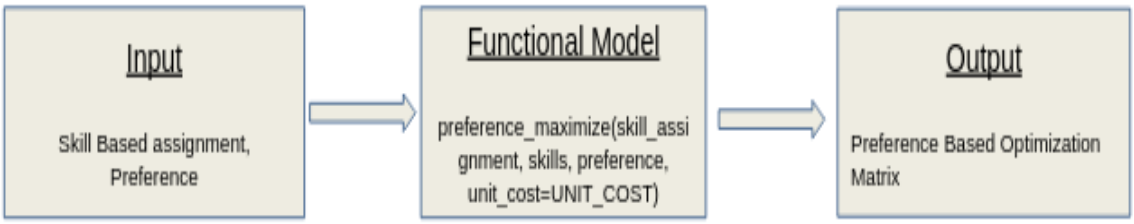


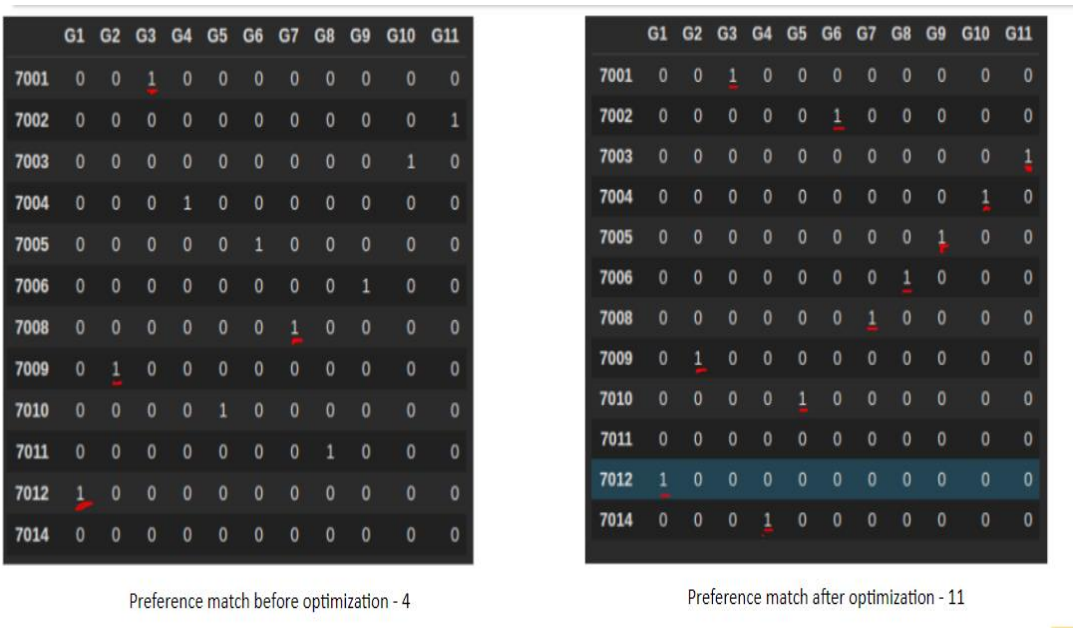
Figure 11:Preference Based Assignment execution timeline

In preference based assignment, we have the input as employee preference data and the skill based assignment output which we have obtained in the previous stage. We pass this input to a function *preference_maximize( )*, where the output will be the preference based optimization matrix.

```
[ ]  # Function for preference based optimization

    def preference_maximize(skill_assignment, skills, preference, unit_cost=UNIT_COST):
        people_present = skill_assignment.index.to_list()
        skill_assigned_count = skill_assignment.sum().sum()

        skills_present = skills[skills.index.isin(people_present)]
        skills_present = skills_present.applymap(lambda x: 0.5 if x == 0 else x)

        workstations = skills_present.columns.to_list()
        employees = skills_present.index.to_list()

        prob = LpProblem("preference", LpMaximize)
        assignment = LpVariable.dicts('Prefer', (employees, workstations), cat='Binary')

        prob += lpSum([preference.loc[emp, wrk] * assignment[emp][wrk] for emp in employees for wrk in workstations])

        for workstation in workstations:
            # for each workstation, sum(assignment * skill) over all employees = 1
            prob += lpSum([assignment[employee][workstation] * skills_present.loc[employee, workstation] for employee in employees]) == 1

        for employee in employees:
            # Each employee can be assigned at max 1 workstation
            prob += lpSum([assignment[employee][workstation] for workstation in workstations]) <= 1

        prob += lpSum([assignment[emp][wrk] for emp in employees for wrk in workstations]) == skill_assigned_count

        prob.solve()

        solve_status = LpStatus[prob.status]

        if solve_status == 'Optimal':
            # Do something
            result = pd.DataFrame(assignment).applymap(lambda x: x.varValue).T.astype(int)
            preference_matched_count = int(value(prob.objective))
    #         worsktation_people_count = result.sum()
            excess_people = result.sum(axis=1).loc[lambda x: x == 0].index.to_list()
            assigned_people_cost = result.sum().sum() * unit_cost

    #         print(f'Preferences matched for {preference_matched_count} out of {skill_assigned_count} people.\n')
            return True, result, excess_people, assigned_people_cost
        else:
            # Do something else
            return False, None, None, None
```

Figure 12: Preference Based Assignment snippet

In this model we are maximizing the preference matching and the LP Variable is the assignment of employee to workstation. We can see the output produced by the model in the below output snippet.



	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11
7001	0	0	1	0	0	0	0	0	0	0	0
7002	0	0	0	0	0	0	0	0	0	0	1
7003	0	0	0	0	0	0	0	0	0	1	0
7004	0	0	0	1	0	0	0	0	0	0	0
7005	0	0	0	0	0	1	0	0	0	0	0
7006	0	0	0	0	0	0	0	0	1	0	0
7008	0	0	0	0	0	0	1	0	0	0	0
7009	0	1	0	0	0	0	0	0	0	0	0
7010	0	0	0	0	1	0	0	0	0	0	0
7011	0	0	0	0	0	0	0	1	0	0	0
7012	1	0	0	0	0	0	0	0	0	0	0
7014	0	0	0	0	0	0	0	0	0	0	0

Preference match before optimization - 4

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11
7001	0	0	1	0	0	0	0	0	0	0	0
7002	0	0	0	0	0	1	0	0	0	0	0
7003	0	0	0	0	0	0	0	0	0	0	1
7004	0	0	0	0	0	0	0	0	0	1	0
7005	0	0	0	0	0	0	0	0	1	0	0
7006	0	0	0	0	0	0	0	1	0	0	0
7008	0	0	0	0	0	0	1	0	0	0	0
7009	0	1	0	0	0	0	0	0	0	0	0
7010	0	0	0	0	1	0	0	0	0	0	0
7011	0	0	0	0	0	0	0	0	0	0	0
7012	1	0	0	0	0	0	0	0	0	0	0
7014	0	0	0	1	0	0	0	0	0	0	0

Preference match after optimization - 11

Figure 13: Preference Based assignment output

We can clearly see the improvement in our employee assignment in Figure 12. On the left hand, we have the skill based assignment of employees from stage 2 and on the right we have the preference based assignment from stage 3.

We notice that in skill-based assignment, we only had 4 employees who were matched to their preference. When we optimize for maximum preference matching, we are able to match 11 employees to their preferences.

Considering Staffing fluctuations:

If any excess unassigned employees remain, we may decide to send them home for reduction of total cost or we can choose to cross-train in hopes of reducing future cost of overstaffing by upskilling employees with various workstations. There are two possibilities in going ahead from here which are-

1. Excess Employees are being sent to home to reduce the cost
2. We will ask employees to participate in a Cross-training program where they will be trained for 2 days and after that timeline they will be multiskilled and the new skills would be updated in the skill matrix so that they can be used in the future simulations.

Cross-training and Go-home implementation:

Before we implement cross-training, we have as an input. the total excess employees for a given day. We first check if these employees have already been training at a workstation before this day. If that is the case, we assign that employee to their previous training workstation so that they are able to complete their 2 days training. On the other hand, if an employee is new for training, we randomly allot them a workstation which is available for training, making sure that the employee is not already trained in the workstation we are going to assign them to.

We track the total number of days an employee has been training on a given workstation in a Python dictionary. Each day we scan this dictionary for cases where an employee has completed their 2-day training for a given workstation. In this case, we remove that employee's entry from the training dictionary and make their workstation available for training other employees. We also update this employee's new skill in the skill matrix, which then is used for optimizations from the next day onwards.

Go-home policy implementation is relatively trivial. We note the total number of excess employees we have for the day and multiply that number with half the normal working wage of an employee. This denotes that the excess employees have been sent to their home with half their wage.

4.4. Cross-Training

Cross-training involves teaching an employee who was hired to perform one job function the skills required to perform other job functions. Employees involved in cross-training programs become skilled at tasks outside the usual parameters of their jobs and thus become greater assets for the company while gaining knowledge and skills that benefit them personally. Cross-training programs are a way to more formally organize the process of getting employees prepared to be able to do more than a single job. These programs offer a wide variety of benefits for businesses.

- Great return on investment
- Better collaboration
- Increases employee motivation
- Increases workforce sustainability
- Improves efficiency
- Makes your company more agile

To be effective, a cross-training program must be carefully planned and organized. It cannot be implemented all of a sudden during a crisis. There are a number of decisions that a company must make before the program can get started. It is important, for example, to decide who will be eligible for training, whether the training will be mandatory or voluntary, whether the training will be restricted within job classifications or open to other classifications, and whether it will be administered internally or externally. Prior to implementation, it might be helpful to set up a task force consisting of both management and employees to research the pros and cons of cross-training for the business, assess the feasibility of setting up a program, work out the implementation issues, and set up a realistic schedule for each position.

Block Diagram :

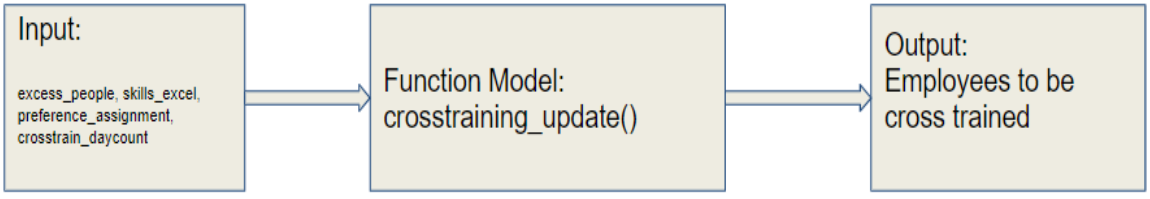


Figure 14: Cross training model

Output:

	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11
7001	0	0	1	0	0	0	0	0	0	0	0
7002	0	0	0	0	0	1	0	0	0	0	0
7003	0	0	0	0	0	0	0	0	0	0	1
7004	0	0	0	0	0	0	0	0	0	1	0
7005	0	0	0	0	0	0	0	0	1	0	0
7006	0	0	0	0	0	0	0	1	0	0	0
7008	0	0	0	0	0	0	1	0	0	0	0
7009	0	1	0	0	0	0	0	0	0	0	0
7010	0	0	0	0	1	0	0	0	0	0	0
7011	0	0	0	0	0	0	0	0	0	0	0
7012	1	0	0	0	0	0	0	0	0	0	0
7014	0	0	0	1	0	0	0	0	0	0	0

Figure 15: output of cross training model

After preference based assignment we observe that 7011 has no workstations assigned. Hence, we send 7011 to be cross trained.

4.5. Go-Home Policy

In the case of overstaffing or inefficient staffing, the company may decide to send some of the employees home on the basis of preference, skill and inefficiency. This may either lead to a reduction in the overall skill of teams and incur extra costs to hire new employees to fill in the skill void.

Block diagram :

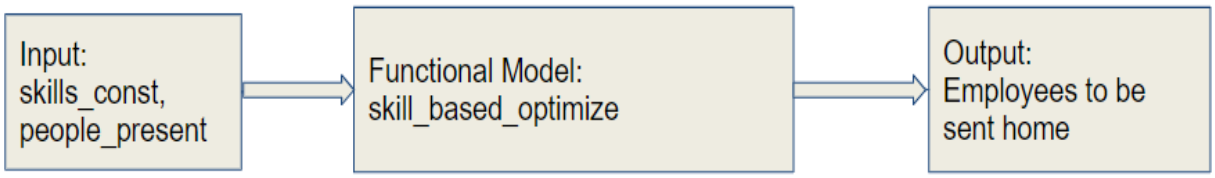


Figure 16: Go-home policy model

Employees who couldn't be cross trained are filtered out based on their original skill set and sent home to reduce costs.

5. PROJECT RESULTS

5.1. Company Cost for cross training policy

The below visualization represents the company total cost statistics on a 30 day rolling mean basis against a simulation for implementing cross training policy for a 1000 days' work period . It is observed that the total cost of the company is declining as the cross training policy is put to effect but encounters considerable inconsistent fluctuations. As the number of working days increase, a moderate cost variation is observed. This plot shows that there are significant initial investment costs encountered by the organization while training the employees as the employees get trained and when re-skilled employees start working for regular work, the total cost lessens.

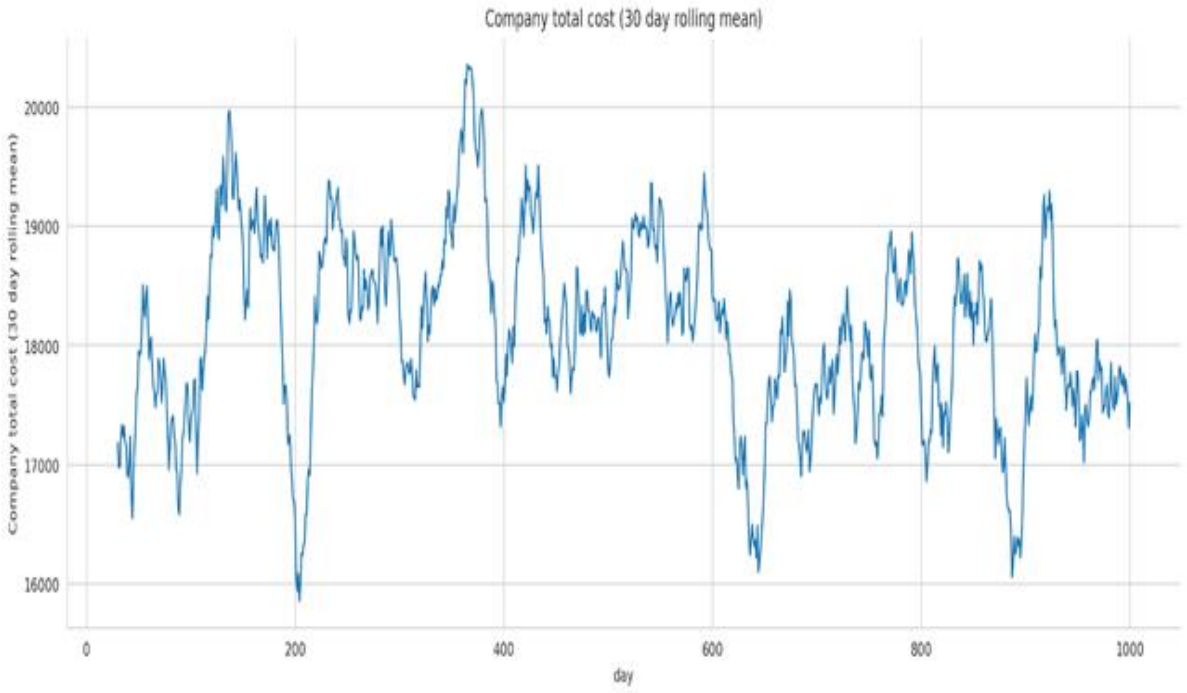


Figure 17: Company Cost for cross training policy

5.2. Company cost for Cross-Training policy (Split by training and working cost)

The below visualization represents the company total cost variation on a 30 day rolling mean basis split further into Training cost on the left axis and the Working cost on the right axis for a simulation which implements cross training policy for a 1000 day work period. If we consider the Training cost of the company, we observe downward slope. This reaffirms the observation in the previous result that there is a huge initial investment to train the employees and as the number of days progress, a steady drop is observed. We also observe here that when most of the training is finished, which can be seen by a drop of training cost to a negligible value, our working cost starts to decrease slowly since more re-skilled employees are now present in the workforce who can be assigned efficiently.

Figure 18: Company cost for cross training policy

5.3. Employee Skills

The below visualization represents the total skills of the team's performance statistics against a simulation for implementing cross training policy for a 1000 days' work period. Over time it is observed that each team's overall skills improve. Teams C, B and F see the maximum improvement in their net skill level.

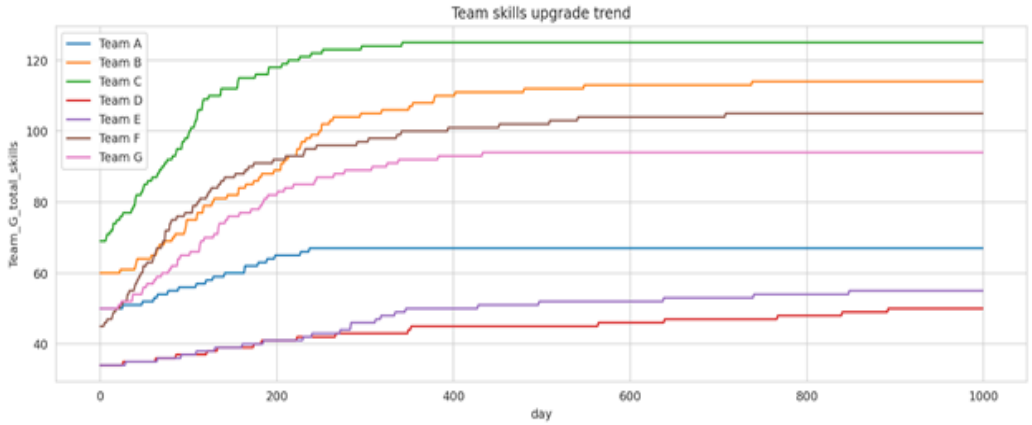


Figure 19: team skills upgrade trend

5.4. Cross- Training vs Go-Home Policy

The below visualization represents a comparison between Cross training and Go-Home policy implementation for a 1000 day work period simulation. We compare the number of teams who are able to find a feasible solution on a 30-day rolling average basis for cross training as well as go home policy. We observe that during cross training (in blue) as a result an average of around 5 teams seem to successfully find a feasible solution due to the increase in the teams' overall skillset. In comparison, an average of around 4 teams seem to arrive at a feasible solution when the go-home policy (in orange) is implemented.

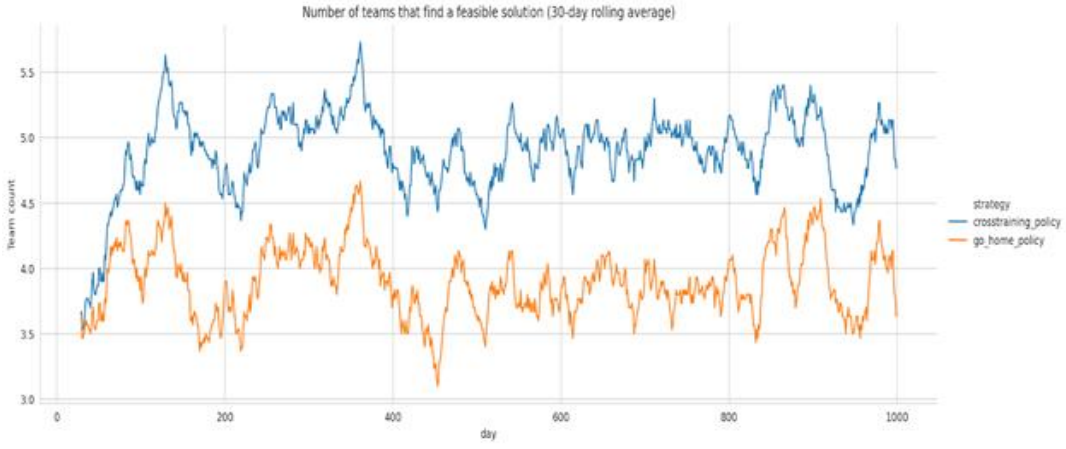


Figure 20: Number of teams that find a feasible solution

5.5. Cross- Training vs Go-Home Policy

The below visualization represents a comparison between Cross training and Go-Home policy implementation for a 1000 day work period simulation. The comparison is based on the company overall cost while implementing cross training and go home policy. It is observed that while implementing cross training (in blue) initial investment costs seem to be high but as the days progress the cost reduces to a bare minimum. This is because the initial investment is to cross train the employees, once the employees are out of training and start working a regular schedule the company does not have to incur additional training or maintenance costs. In comparison, while implementing go-home policy the cost seems to not change much with the increase in the number of working days. As, when employees are sent home, there is no change in their skill-set and hence their long term value remains constant, unlike the people who are cross-trained.

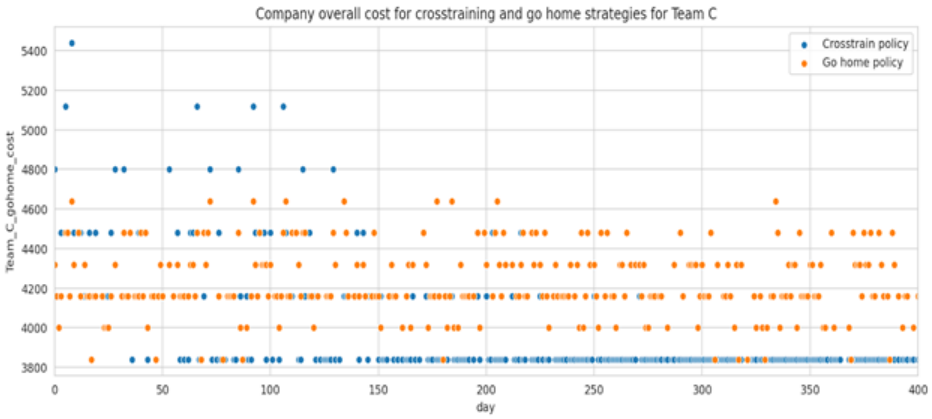


Figure 21: Company Overall Cost for cross train and go home

5.6. Improvement in preferences matched after the preference matching stage

We observe that there is, on an average, a significant improvement in the preferences matched due to the optimization we have done in the preference matching stage. Y-axis here represents the increase in the number of employees who are able to work at their preference after the preference matching stage. Teams C, F and B have benefitted the most with an approximate improvement of around 6.7, 6.9 and 5.6 respectively.

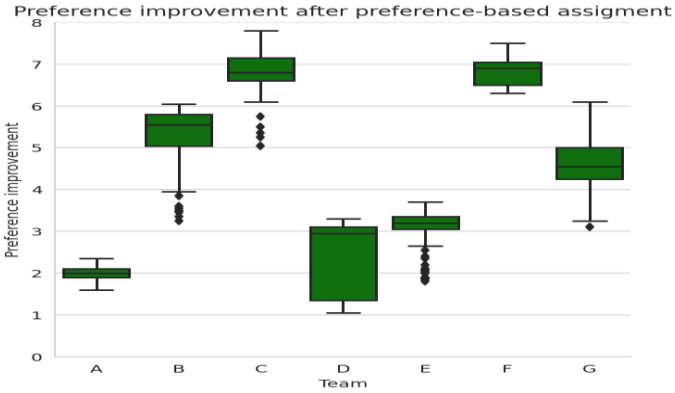


Figure 22: Improvement in preferences matched after the preference matching stage

6. FUTURE WORK

As an extension to this problem, future work and approach could be to implement Inter-Team Resource sharing. Team Resource Management strives to fully utilize all available resources, information data, tools/equipment/aids and humans appropriately, in order to achieve the optimal efficiency. We attempted to formulate inter-team resource sharing in scenarios where some teams have excess employees while other teams are not able to find a feasible solution. Since we are tracking the excess employees already in our simulation, we can implement following algorithm:

1. Assign one of the excess employee randomly to a failed team

2. Run the skill-based simulation for the updated team

3. Repeat the process (points 1 & 2) until a solution is found

4. Repeat workflow above (points 1, 2 & 3) till all the teams find a feasible solution or we run out of excess employees.

This approach should efficiently utilize the excess employees because it re-computes the LP after a new employee is assigned to a failed team. This optimization can be done right after the skill-based assignment stage is over, and right before the preference matching is done.

7. CONCLUSION

We study and model the operations of a fictional firm and simulate the organizations' working for a better understanding of our modelling approaches. We are able to break down entire operations in the problem description into 5 stages: generating attendance, skill-based assignment, preference-based assigned, cross-training/go-home policy and inter-team resource sharing (future work). The stages are successfully implemented in Python with the help of an open-source LP library, PuLP. All the stages are then simulated in their respective order. We track all the necessary metrics during the simulation and monitor them after. We find significant general conclusions about the modelling strategies we implement, notably the improvements of skill-based assignment and preference based assignment and note the dichotomy between cross-training and go-home policy approaches, in terms of cost, existence of feasible solutions and overall skill improvements.

8. PROJECT FILES

Colab Notebook
Simulation output data

9. CONTRIBUTION

WORK STRUCTURE	TEAM MEMBER
Employee Allocation	Meher Adheeth Hundi
Skill-Based Assignment	Sonal Bhor
Preference-Based Assignment	Sanjyot Thete
Cross-Training and Go home Policy	Sumanth Pobala
Graph Results	Harshita Shivaramakrishna