Report on Linear Programming using Python and its Application in Fair Room Allocation

Master Thesis Project-2

by

Santosh Kavhar (22M0787)

under the guidance of

Prof. Swaprava Nath

Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai 400076

Contents

List of Figures

List of Tables

Abstract

Linear programming(LP) is being widely used in many engineering domains because of its wide use in resource allocation techniques. In this work we focus on the implementation of linear programming using python and its application in fair room allocation. By fair room allocation we mean certain parameters like best preference and envy-freeness. The problem of fair room allocation usually occurs when a group of tenants have different valuations for different rooms, that is the exact problem we will be trying to solve. For this we will solve the maximum weight bipartite matching problem using LP in python using PuLP library. Similarly we will solve the envy-freeness problem.

Acknowledgement

I sincerely thank my guide Prof. Swaprava Nath for his valuable guidance, encouragement, and patience. I would like to express my gratitude to all my teachers, and professors till date who have played a huge role in my academic life. I also wish to thank my family and friends for all their support in these trying times. I would also like to thank all the professors who have helped me learn the concepts of Computer Science and implement it. My friend Ram gave great feedback when it came to understanding this project. Thank you all and God bless you all.

Chapter 1

Introduction

There is a lot of learning in implementation and thus with this project we are learning the challenges to implement the solution of rental harmony. This project aims to find the best allocation of rooms with a reasonable rent.

1.1 Motivation

Solving the problem of Fair Rent Division will ensure harmony and cooperation amongst the tenants. It will also give a fair solution which will keep in mind the user valuations. Giving the fair solution means getting the best for one's money and thus this solution does justice to fair distribution. Also this solution is not like a black-box solution but its rather intuitive and the result is all based on Mathematics, so, we can find out why it is a fair solution every time. Also this problem is relevant in day-today life as per the huge response on the website like spliddit. It can be extended to have its application in other areas where fair division is needed.

1.2 Objectives

1. Develop a Fair Rent Division Algorithm Design and implement an algorithm that fairly divides rent among tenants in a shared living space, considering user valuations and preferences, to promote harmony and cooperation.

2. Consider User Valuations for Fairness Integrate a mechanism to capture and incorporate tenant valuations and preferences, ensuring the final rent division is in line with their perceived fairness and satisfaction.

3. Optimize for Best Value Optimize the rent division to ensure that each tenant receives the best value for their contribution, promoting equitable distribution and justifying the cost incurred.

4. Maintain Transparency and Intuitiveness Ensure the solution is transparent and intuitive, allowing tenants to understand how the rent division is determined. Provide clear insights into the mathematical basis of the solution, making it comprehensible and acceptable.

1.3 Structure

In rest of this chapter we will cover Background which has all the necessary terms and concepts defined ready to set us understand the project. The rest of this this report is organised as follows: In Chapter 2 we will cover the Related Works. In Chapter 3 we cover the implementation part of the project giving the model information as well. In Chapter 4 presents the analysis of one of the solution with an example. Finally in Chapter 5 named as Conclusion we cover the results, learnings and future steps for the project.

1.4 Background

In this section we will highlight the terms and their definition which will be useful to understand our work. We have three subsections, in the first we will review the relevant topics in LP then in second subsection we will review on python and PuLP and then finally the third subsection, where we will explain about MWBM and envy-free constraint.

1.4.1 Linear Programming(LP)

The goal of **linear programming (LP)** is to get the best possible allocation of decision variables which maximizes or minimizes the linear objective function. Decision variables are subject to linear constraints. The technique of linear programming is used in wide range of applications, including transportation, military, industry, agriculture, social science, economic, and healthcare system.

In linear programming, a **constraints** refers to a limitation or condition that restricts the possible solutions of a problem. These constraints are usually expressed as linear inequalities or equations that must be satisfied by the decision variables of the problem.

Optimization refers to getting the best or most favorable outcome or solution that can be obtained in a particular context or situation. An optimal solution is one that provides the best possible value for a given objective function, subject to a set of constraints.

The **feasible region** refers to the set of all possible values of the decision variables that satisfy all the constraints in a given problem. It is the set of all feasible solutions or points that can be considered as potential optimal solutions for the problem. If the feasible region is unbounded then we say that there is not any bounded solution to our problem.

The **objective function** is a mathematical expression that represents the quantity to be maximized or minimized in the optimization problem. It is a linear combination of the decision variables that describes the goal or the objective of the problem.

1.4.2 Python and PuLP

Python is a high-level, interpreted programming language that is widely used in a variety of applications and domains. Python is a versatile language that is used

in web development, scientific computing, data analysis, machine learning, artificial intelligence, and many other areas.

PuLP is a popular open-source linear programming (LP) modeling package in python. It provides an interface to create LP models in python, which can be solved by various LP solvers like GLPK, CBC, CPLEX, Gurobi, etc. PuLP allows users to create LP models using a more user-friendly syntax compared to other LP modeling tools.

We declare variables for our LP with help of *LpVariable*. To declare whether a problem is maximizing or minimizing we use *LpProblem*. In this the first parameter is the name of the problem and the second parameter is *LpMaximize* or *LpMinimize* depending on our problem. *LpProblem* returns us a variable in which we can append our LP constraints and maximizing or minimizing function. We can have only a single maximizing or minimizing function and rest have to be constraints. PuLP separates out the constraints from the function by looking at the equality condition that is input in case of constraints and is absent for the maximizing or minimizing functions. In the end we use *solve()* method to solve the LP problem. The *LpStatus* Option will print the type of solution like *Optimal, Not Solved, Infeasible, Unbounded, Undefined* etc.

We have also used functions like *lpSum* to get the sum of the elements from a list. This function helped us program at ease.

1.4.3 Utility, MWBM and Envy-Freeness

Utility is a way to measure how much someone values different options when deciding how to split the rent in a shared house. It helps each person choose a fair share that makes them the happiest and is fair to everyone.

Maximum Weight Bipartite Matching (MWBM) is a classical optimization problem in graph theory. It is a variant of the bipartite matching problem, where the edges between two disjoint sets of vertices are weighted, and the goal is to find a maximum-weight matching, i.e., a matching of maximum total weight.

Envy-freeness is a concept in social choice theory that refers to a situation where no individual in a group prefers the allocation of resources or goods received by another individual in the same group. In other words, an allocation is envy-free if everyone perceives their own allocation to be at least as good as someone else's allocation.

1.5 Related Works

We have referred many resources out of which the three are the most for our work are cited here.

- (Gal et al., 2016) talk about the fairest rent division problem and the first step in their algorithm is the welfare-maximizing assignment. It is from this article that we think of solving the matching using LP instead of traditional approach where we convert this problem into maximal flow problem which then can be solved by ford-fulkerson algorithm.

- (Chong and Zak, 2013) gives detailed chapters about LP from where we learned the concepts of LP, it also has a chapter on simplex algorithm as well as on concept of duality.
- (Mitchell et al., 2011) gives the documentation for PuLP which helped us use the functionality of PuLP. It is a self sufficient documentation for help.

Chapter 2

Implementation for Single House

2.1 Model

Finding the most optimum solution for a linear function (also known as the objective function) under a set of restrictions (a set of linear inequality constraints) is made possible by linear programming problems (LPP). The five basic requirements of linear programming are: constraints, objective function, linearity, non-negativity, finiteness. The most popular methods to solve LPPs are the simplex method and interior point methods. For the detailed explanations see (Chong and Zak, 2013).

A standard linear program is an optimization problem of form

$$\text{minimize} \quad c^T x$$

$$\text{subject to} \quad Ax = b$$

$$x \geq 0$$

$$\text{where } \mathbf{c} \in \mathbb{R}^{\mathbb{N}}, \mathbf{b} \in \mathbb{R}^{\mathbb{M}}, \text{and } \mathbf{A} \in \mathbb{R}^{\mathbb{M}*\mathbb{N}}, \text{ here } c^T = [c_1, c_2, ..., c_n],$$

$$x = [x_1, x_2, ..., x_n]^T \ ,$$

$$b = [b_1, b_2, ..., b_m]^T \ , \text{ and}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mn} \end{bmatrix}$$

x $\geq$ 0 means every component of x is non-negative. Instead of minimizing, we can maximize, or the constraints may be in the form of inequalities, such as Ax $\geq$ b or Ax $\leq$ b. We also refer to these variations as linear programs.

2.1.1 Example of LP

Example 1. *This example is adapted from (Chong and Zak, 2013). A manufacturer produces four different products: X_1, X_2, X_3, and X_4. There are three inputs to this production*

process: labor in person-weeks, kilograms of raw material A, and boxes of raw material B. Each product has different input requirements. In determining each week's production schedule, the manufacturer cannot use more than the available amounts of labor and the two raw materials. Every production decision must satisfy the restrictions on the availability of inputs. These constraints can be written down as

$$x_1 + 2x_2 + x_3 + 2x_4 \leq 20$$
$$6x_1 + 5x_2 + 3x_3 + 2x_4 \leq 100$$
$$3x_1 + 4x_2 + 9x_3 + 12x_4 \leq 75$$

As the negative production levels are not meaningful, we must impose the following non-negativity constraints on the production levels: $x_i \geq 0, i = 1,2,3,4$

Now, suppose that one unit of product X_1 *sells for ₹6, and* X_2, X_3, *and* X_4 *sell for ₹4, ₹7, and ₹5, respectively. Then, the total revenue for any production decision* (x_1, x_2, x_3, x_4) *is*

$$f(x) = 6x_1 + 4x_2 + 7x_3 + 5x_4$$

We can observe that m=3 and n=4. The problem can be written in compact form

$$\textit{maximize } c^T x$$
$$\textit{subject to } Ax \leq b$$
$$x \geq 0$$
$$\textit{where } c^T = [6,4,7,5],$$
$$x = [x_1, x_2, x_3, x_4]^T,$$
$$b = [20, 100, 75]^T \textit{, and}$$
$$A = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 6 & 5 & 3 & 2 \\ 3 & 4 & 9 & 12 \end{bmatrix}$$

The Optimal solution on solving this is on PuLP is $\mathbf{x}$*= [15, 0, 3, 0] and thus the Optimal value of profit is = 111₹.*

2.1.2 Maximum Weight Bipartite Matching(MWBM)

We can frame the MWBM in the form with some constraints.

$$\max \sum_{i \in N} \sum_{j \in M} v_{ij} \cdot x_{ij} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j \in M} x_{ij} \leq 1, \ \forall i \in N \tag{2.2}$$

$$\sum_{i \in N} x_{ij} \leq 1, \ \forall j \in M \tag{2.3}$$

$$x_{ij} \geq 0, \ \forall i \in N, j \in M. \tag{2.4}$$

From eq. (2.1) we can see that the MWBM is a maximization problem. Here v_{ij} refers to the weights of the edges of the graph between the source as node i and the destination as node j. For our example v_{ij} means the value of room i from the perspective of renter j. x_{ij} refers to the choice whether the source node i is connected with node j. If there were no constraints then all left hand side nodes(denoted by i) would have been connected with right hand side nodes(denoted by j). But we have three constraints. The first constraint, denoted by eq. (2.2), states that the number of nodes connecting to any node j (in right hand side) is less than or equals to one. Similarly the second constraint, denoted by eq. (2.3), states that the number of nodes connecting to any node i (in left hand side) is less than or equals to one. Because of these constraints we see that only the nodes which produce highest overall weight are considered for connection. The third eq. (2.4) gives us non-negativity condition. This constraint seems redundant as we are trying to maximize the eq. (2.1) but we will keep it for better clarity.

	Room 1	Room 2	Room 3
Renter 1	300	300	400
Renter 2	400	200	200
Renter 3	250	250	500

Table 2.1: Renters preference for rooms

Renter's preference for their choice of rooms is represented in table 2.1 . In this example, three renters prefer three rooms at different rates. This rent is achieved by means of maximum weight bipartite matching. The bold ones in table 2.2 are the assigned mapping such that maximum rent is achieved. This type of mapping is also called the maximum welfare assignment. For e.g in table 2.2 first renter is mapped to second room, second renter to first room, and third renter to third room.

2.1.3 Envy-freeness constraint

We need to focus on two main things in case of envy-freeness.

maximize R such that:

$$R \leq f_q(v_{1\sigma(1)} - p_{\sigma(1)}, ..., v_{n\sigma(n)} - p_{\sigma(n)}) \tag{2.5}$$

	Room 1	Room 2	Room 3
Renter 1	300	**300**	400
Renter 2	**400**	200	200
Renter 3	250	250	**500**

Table 2.2: Renter Allotted to rooms

$$v_{i\sigma(i)} - p_{\sigma(i)} \geq v_{ij} - p_j \quad \forall i, j \in [n] \tag{2.6}$$

Where f_q is a function which collects the minimum of all the values. σ refers to the allocated mapping, in case of table 2.1 $\sigma(1)$ means the room mapped to first renter. v_{ij} means the value of room i from the perspective of renter j. The variables $p_1, p_2, ...p_n$ denote the prices that will be set once these conditions are solved by linear programming.

Thus here we are trying to maximize the minimal profit earned by each renter by eq. (2.5).

By eq. (2.6) we are trying to make the allocation envy-free. The envy-free constraint for first renter will be satisfied when $v_{12} - p_2 \geq v_{13} - p_3$ and $v_{12} - p_2 \geq v_{11} - p_1$, which means his share of profit for the room he has been allocated is as good as anyone else. Similarly for other cases.

$$\sum_{i=1}^{N} p_i = Rent \tag{2.7}$$

By eq. (2.7) we make sure that sum of all the room prices calculated is equal to total Rent. With this added condition we would evoke the LP to give an Optimal solution.

2.2 Back-end

The problem as per the model is broken into 2 parts. Maximum Weight Bipartite matching and then Envy-freeness problem. Below functions show the implementation of both the parts. Complete code can be found on github.

Please note that the input is such that the total sum of all rent's room for a renter is equal to the total rent. This is true for all renters. In case of single hostel allocation we normalize the given input.

2.2.1 MWBM

```
1 def welfare_maximize(values, agent_list, room_list):
2
3     allocation = {}
```

```
    variables = {}

    # failure(agent_list)
    # failure(room_list)
    # failure(values)

    prob = LpProblem("Welfare_Maximization", LpMaximize)
    for a in agent_list:
        variables[a] = {}
        for r in room_list:
            variables[a][r] = LpVariable(f"x_{a}_{r}", 0, 1,
    LpBinary)

    # Objective is total welfare
    prob += lpSum(values[a][r] * variables[a][r] for a in
    agent_list for r in room_list)

    # Each agent assigned 1 room
    for a in agent_list:
        prob += lpSum(variables[a][r] for r in room_list) == 1

    # Each room assigned 1 agent
    for r in room_list:
        prob += lpSum(variables[a][r] for a in agent_list) == 1

    try:
        # print(prob)
        # Silence PuLP messages
        prob.solve(PULP_CBC_CMD(msg=False))
    except:
        failure("Error occured! Couldn't solve the LP problem!")
        return

    # Check the status of the solution
    if LpStatus[prob.status] != OPTIMAL:
        failure("Problem solving the LP!\t" + LpStatus[prob.status
    ])
        return

    # Get the allocation
    for a in agent_list:
        for r in room_list:
            if variables[a][r].value() == 1:
                allocation[a] = r

    return allocation
```

Listing 2.1: Python MWBM house code

2.2.2 Envy-freeness

```
def envy_free_prices(values, agent_list, room_list, allocation):

    prices = {}
    price_variables = {}
    rev_allocation = {y: x for x, y in allocation.items()}
```

```
    # Calculate the rent from the first row values
    rent = calculate_rent(values[0])

    prob = LpProblem("Envy_Freeness", LpMinimize)

    for r in room_list:
        # lower limit could be negative as well
        # If it is 0 then it is not a maximin utility solution
        price_variables[r] = LpVariable(
            f"p_{r}", -1 * rent, values[rev_allocation[r]][r]
        )

    # Objective is maximize minimum utility (or minimize negative
of minimum utility)
    min_utility = LpVariable("y", 0)
    prob += min_utility

    # Ensure prices sum to rent
    prob += lpSum(price_variables[r] for r in room_list) == rent

    # print(agent_list, values, allocation)
    # Ensure envy-free
    for i in agent_list:
        for j in room_list:
            # Sample Envy example:
            # 12,0,0
            # 12,0,0
            # 3,5,4
            # For envyness uncomment below 2 lines
            # if i == j:
            #     continue
            prob += (
                price_variables[j] - price_variables[allocation[i]]
                >= values[i][j] - values[i][allocation[i]]
            )

    # Bound minimum utility
    for i in agent_list:
        prob += min_utility >= values[i][allocation[i]] -
price_variables[allocation[i]]

    try:
        # print(prob)
        prob.solve(PULP_CBC_CMD(msg=False))
    except:
        failure("Error occured! Couldn't solve the LP problem!")
        return

    # print(agent_list, room_list, LpStatus[prob.status])

    if LpStatus[prob.status] != OPTIMAL:
        failure("Problem solving the LP!\t" + LpStatus[prob.status
])
        return None
```

```
58
59      for i in agent_list:
60          room = allocation[i]
61          prices[room] = price_variables[room].value()
62
63      return prices
```

Listing 2.2: Python Envy-free house code

2.3 Front-end

Only the UI will be shown and not its code. The Front-end or UI was built using React library. Complete code can be found on github.

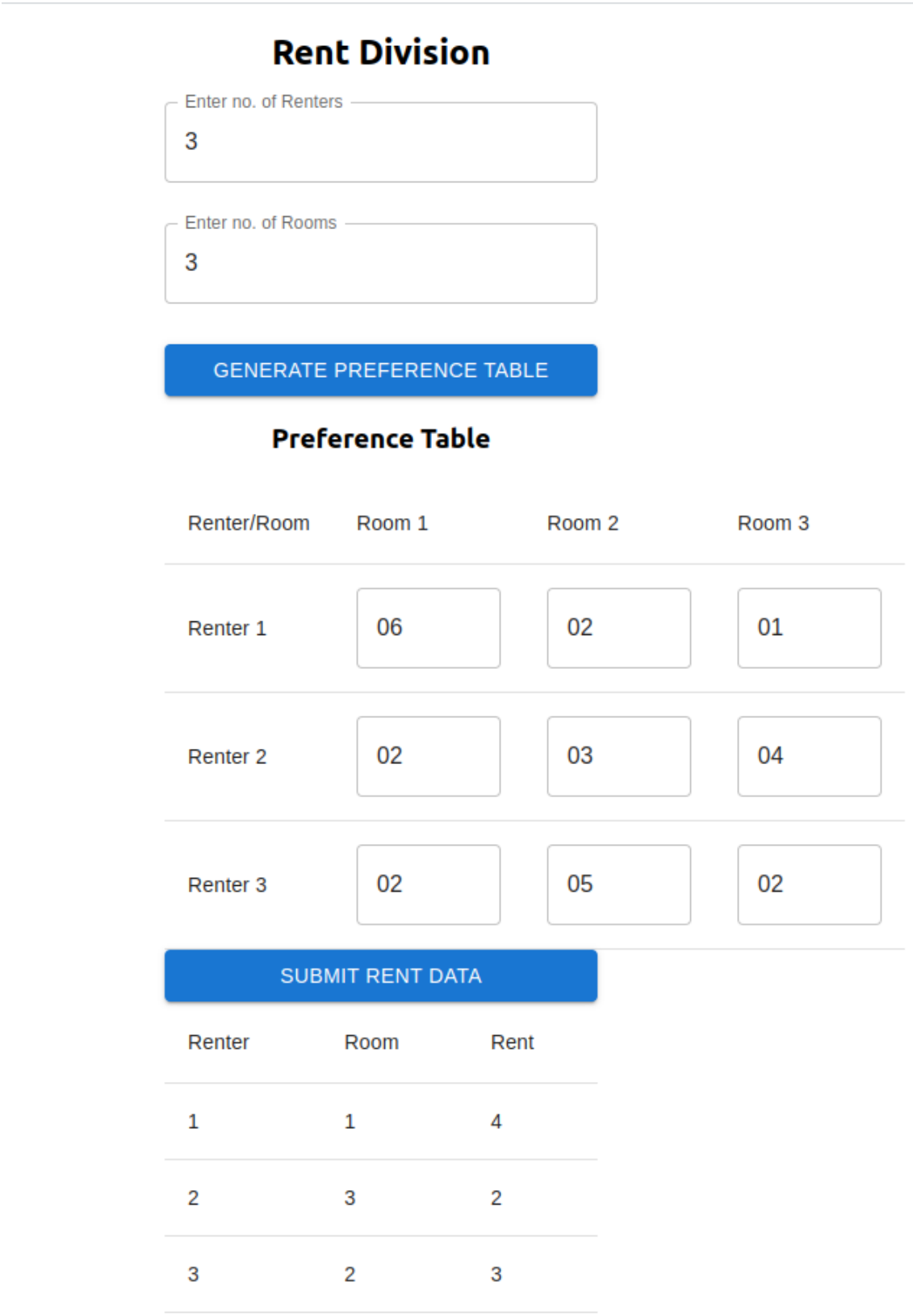

Figure 2.1: UI for Rent Division

Chapter 3

Analysis for Single House

Let us analyze and example to understand how the allocation is envy free.

3.1 Analysis with an Example

We will analyze a simple rent division problem, the same that was in the fig. 2.1.

3.1.1 The Problem

In the table 3.1 given below we want to get a best solution. Please note that the sum of all the three rows is equal to 9. The first step will be finding MWBM and the second step will be Envy-freeness.

	Room 1	Room 2	Room 3
Renter 1	6	2	1
Renter 2	2	3	4
Renter 3	2	5	2

Table 3.1: Renter's preference for rooms

3.1.2 The Output

Clearly we have a MWBM output as in table 3.2. This gives us information about the final allocation of rooms. We can verify that this is the best matching as the given values are already the highest in their own rows.

The final output after envy-freeness and maximin utility is as given in table 3.3:

3.1.3 Analysis

We will only analyze the envy-freeness property of the solution as MWBM has already been explained in previous section. Consider the analysis by solving for LHS(Left Hand Side) and RHS(Right Handf Side) for the equations given.

	Room 1	Room 2	Room 3
Renter 1	**6**	2	1
Renter 2	2	3	**4**
Renter 3	2	**5**	2

Table 3.2: Renter's Allocation for rooms

Renter	Room	Rent
1	1	4
2	3	2
3	2	3

Table 3.3: Final Rent assigned

- Consider Renter 1 who is assigned Room 1 for 4₹.
  Renter 1 will have to be satisfied by eq. (3.1):

$$v_{1\sigma(1)} - p_{\sigma(1)} \geq v_{1j} - p_j \quad \forall j \in [n] \tag{3.1}$$

  LHS = 6 - 4 = 2
  For RHS
  - When j=1 RHS = 6 - 4 = 2
  - When j=2 RHS = 2 - 3 = -1
  - When j=3 RHS = 1 - 2 = -1

  LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 1.

- Consider Renter 2 who is assigned Room 3 for 2₹.
  Renter 2 will have to be satisfied by eq. (3.2):

$$v_{2\sigma(2)} - p_{\sigma(2)} \geq v_{2j} - p_j \quad \forall j \in [n] \tag{3.2}$$

  LHS = 4 - 2 = 2
  For RHS
  - When j=1 RHS = 2 - 4 = -2
  - When j=2 RHS = 3 - 3 = 0
  - When j=3 RHS = 4 - 2 = 2

  LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 2.

- Consider Renter 3 who is assigned Room 2 for 3₹.

  Renter 3 will have to be satisfied by eq. (3.3):

$$v_{3\sigma(3)} - p_{\sigma(3)} \geq v_{3j} - p_j \quad \forall j \in [n] \tag{3.3}$$

  LHS = 5 - 3 = 2

  For RHS

  - When j=1 RHS = 2 - 4 = -2
  - When j=2 RHS = 5 - 3 = 2
  - When j=3 RHS = 2 - 2 = 0

  LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 3.

Thus, we find that the allocation is envy-free for all the renters. This can be found out for other solutions as well in a similar manner.

Chapter 4

Implementation for Hostel

4.1 Terminology

4.1.1 Floor

Floor refers to the floor of the building. The user has to input number of floors in total. This means if you have 3 wings in a hostel each of which contain 4 floors then you should input 3*4 = **12** floors as the input. Getting floors this way avoids the problem that arises when any floor's wing is to be conditionally left empty. You can edit the floor names in the input itself. By default the values will be like : "Floor 1", "Floor 2"., etc. Please refer fig. 4.2 , which has second input row as *Floor Names*.

4.1.2 Capacity

Capacity refers to the maximum capacity of the floor. It may be different for each floor. It is also taken as input from user along with Floor information. Please refer fig. 4.2 to see that capacity is first row of the input.

4.1.3 Normalization

In the house allocation we didn't do any normalization, but in case of hostel allocation we must do the normalization. This is because each floor should be valued as per the capacity parameter as well. We are normalizing the inputs with respect to the rent and capacity.

Please refer fig. 4.1 for the normalization of values entered as input to fig. 4.2. We will calculate how these have come into the picture. We will do it for Renter 1 only.

Solve for x which is normalization constant for renter$_i$, such that

$$\sum_{j=1}^{N} (Renter_i's\ value\ for\ floor_j) * (floor_j's\ capacity) * x = Rent \tag{4.1}$$

Here **N** stands for number of floors. Renter$_i$'s value for floor j means value of Rent of the floor j according to the renter$_i$. Normalised value thus finally is

$$(Renter_i's\ value\ for\ floor_j) * (floor_j's\ capacity) * x \tag{4.2}$$

```
"normalized_rent_data": [
    [
        9.411764705882353,
        10.588235294117647
    ],
    [
        8.0,
        12.0
    ],
    [
        10.32258064516129,
        9.67741935483871
    ],
    [
        9.411764705882353,
        10.588235294117647
    ]
],
```


Figure 4.1: Normalization for Rents

Example

Let us take an example for Renter 1 from the data of fig. 4.2.
Rent = 20.
Number of floors (N) = 2.
Renter= 1st.
Thus the equation for normalization becomes:

$$\sum_{j=1}^{2} (Renter_1's\ value\ for\ floor_j) * (floor_j's\ capacity) * x = 20 \tag{4.3}$$

Thus, this equation is expanded as :

$$\begin{aligned} &(Renter_1's\ value\ for\ floor_1) * (floor_1's\ capacity) * x \\ &+ (Renter_1's\ value\ for\ floor_2) * (floor_2's\ capacity) * x = 20 \end{aligned} \tag{4.4}$$

$$2 * 4 * x + 3 * 3 * x = 20 \tag{4.5}$$

Thus, x = 20/17
Thus $Renter_1$'s normalized value for $Floor_1$ is given by:

$$(Renter_1's\ value\ for\ floor_1) * floor_1 * x = 2 * 4 * 20/17 \approx 9.411 \tag{4.6}$$

Similarly $Renter_1$'s normalized value for $Floor_2$ is given by:

$$(Renter_1's\ value\ for\ floor_2) * floor_2 * x = 3 * 3 * 20/17 \approx 10.588 \tag{4.7}$$

Similarly we could do it for other 3 renters.

4.2 Model

We will see how hostel allocation is different from Room allocation in this section. Let us consider it by means of Maximum Weight Bipartite Matching(MWBM) and Envy-freeness constraint.

4.2.1 Maximum Weight Bipartite Matching(MWBM)

Let us take a look at the equations for Maximum Weight Bipartite Matching in case of Hostel allocation.

$$\max \sum_{i \in N} \sum_{j \in M} v_{ij} \cdot x_{ij} \tag{4.8}$$

$$\text{s.t.} \quad \sum_{j \in M} x_{ij} \leq 1, \ \forall i \in N \tag{4.9}$$

$$\sum_{i \in N} x_{ij} \leq C_j, \ \forall j \in M \tag{4.10}$$

$$x_{ij} \geq 0, \ \forall i \in N, j \in M. \tag{4.11}$$

Here eq. (4.8) is same as eq. (2.1), similarly eq. (4.9) is same as eq. (2.2) but eq. (4.10) is different from eq. (2.3). This equation tells us that the maximum number of renters on a floor j is at max C_j.

4.2.2 Envy-freeness constraint

maximize R such that:

$$R \leq f_q(v_{1\sigma(1)} - p_{\sigma(1)}, \ldots, v_{n\sigma(n)} - p_{\sigma(n)}) \tag{4.12}$$

$$v_{i\sigma(i)} - p_{\sigma(i)} \geq v_{ij} - p_j \quad \forall i, j \in [n] \tag{4.13}$$

Where f_q is a function which collects the minimum of all the values. σ refers to the allocated mapping, in case of table 2.1 $\sigma(1)$ means the room mapped to first renter. v_{ij} means the value of room i from the perspective of renter j. The variables $p_1, p_2, \ldots p_n$ denote the prices that will be set once these conditions are solved by linear programming.

4.3 Back-end

Here we will see the coding part of the code for Maximum Weight Bipartite Matching(MWBM) as well as Envy-freeness.

4.3.1 MWBM

```
def welfare_maximize(values, agent_list, room_list, capacity):

    allocation = {}
    variables = {}

    # failure(agent_list)
    # failure(room_list)
    # failure(values)
```

```
    prob = LpProblem("Welfare_Maximization", LpMaximize)
    for a in agent_list:
        variables[a] = {}
        for r in room_list:
            variables[a][r] = LpVariable(f"x_{a}_{r}", 0, 1,
    LpBinary)

    # Objective is total welfare
    prob += lpSum(values[a][r] * variables[a][r] for a in
    agent_list for r in room_list)

    # Each agent assigned 1 room
    for a in agent_list:
        prob += lpSum(variables[a][r] for r in room_list) <= 1

    # Each room assigned 1 agent
    # For floor assigned capacity agents for that floor
    for r in room_list:
        # Default capacity is 1
        prob += lpSum(variables[a][r] for a in agent_list) <=
    capacity[r]

    try:
        print(prob)
        # Silence PuLP messages
        prob.solve(PULP_CBC_CMD(msg=False))
    except:
        failure("Error occured! Couldn't solve the LP problem!")
        return

    # Check the status of the solution
    if LpStatus[prob.status] != OPTIMAL:
        failure("Problem solving the LP!\t" + LpStatus[prob.status
    ])
        return

    # Get the allocation
    for a in agent_list:
        for r in room_list:
            if variables[a][r].value() == 1:
                allocation[a] = r
    success(allocation)
    return allocation
```

Listing 4.1: Python MWBM hostel code

4.3.2 Envy-freeness

```
# floor_list is the same as room_list
def envy_free_prices(values, agent_list, room_list, allocation):

    prices = {}
    price_variables = {}
    rev_allocation = {y: x for x, y in allocation.items()}

```

```
    # Calculate the rent from the first row values
    rent = calculate_rent(values[0])

    prob = LpProblem("Envy_Freeness", LpMinimize)
    warning(room_list)
    warning(agent_list)
    warning(allocation)
    warning(values)
    warning(rev_allocation)

    for r in room_list:
        # warning(values[rev_allocation[a]][a])
        # lower limit could be negative as well
        # If it is 0 then it is not a maximin utility solution
        price_variables[r] = LpVariable(
        f"p_{r}", -1 * rent, values[rev_allocation[r]][r]
        )

    # Objective is maximize minimum utility (or minimize negative
    of minimum utility)
    min_utility = LpVariable("y", 0)
    prob += min_utility

    warning(price_variables)
    # Ensure prices sum to rent
    prob += lpSum(price_variables[r] for r in room_list) == rent

    # print(agent_list, values, allocation)
    # Ensure envy-free
    for a in agent_list:
        for r in room_list:
            # Sample Envy example:
            # 12,0,0
            # 12,0,0
            # 3,5,4
            # For envyness uncomment below 2 lines
            # if a == r:
            #     continue
            prob += (
                price_variables[r] - price_variables[allocation[a]]
                >= values[a][r] - values[a][allocation[a]]
            )

    # Bound minimum utility
    for a in agent_list:
        prob += min_utility >= values[a][allocation[a]] -
    price_variables[allocation[a]]

    try:
        print(prob)
        prob.solve(PULP_CBC_CMD(msg=False))
    except:
        failure("Error occured! Couldn't solve the LP problem!")
        return

```

```
61      # print(agent_list, room_list, LpStatus[prob.status])
62
63      if LpStatus[prob.status] != OPTIMAL:
64          failure("Problem solving the LP!\t" + LpStatus[prob.status
    ])
65          return None
66
67      for a in agent_list:
68          room = allocation[a]
69          prices[room] = price_variables[room].value()
70
71      return prices
```

Listing 4.2: Python Envy-free hostel code

4.4 Front-end

Only the UI will be shown and not its code. The Front-end or UI was built using React library. Complete code can be found on github.

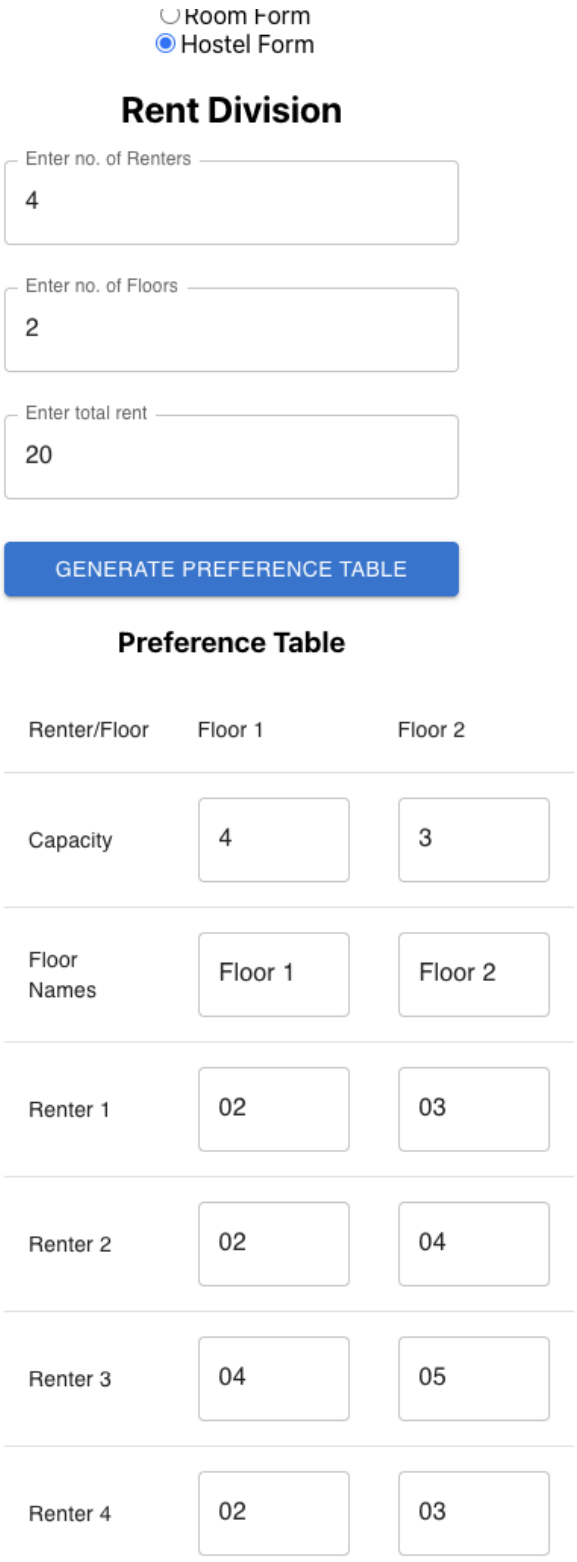

Figure 4.2: UI for Hostel Rent Division

Chapter 5

Analysis for Hostel Allocation

Here we will analyze the problem of hostel allocation with an example.

5.1 Analysis with an Example

The example will be the fig. 4.2. We will solve this example and then check for MWBM as well as envy-freeness constraints.

5.1.1 The Problem

Please refer fig. 4.2 for the problem. The rent is given as 20. We have 2 floors and 4 renters. Capacity and rent values for each floor is given in below table 5.1

	Floor 1	Floor 2
Capacity	4	3
Floor Names	Floor 1	Floor 2
Renter 1	2	3
Renter 2	2	4
Renter 3	4	5
Renter 4	2	3

Table 5.1: Renter's preference for hostel floor

We need to convert the rent values given into normalized values. For normalised values please refer fig. 4.1. We will get output based on these normalized values. Normalized values are put into table as given table 5.2. These are approximated values.

5.1.2 The Output

Once we apply MWBM and Envy-freeness constraints on the problem which is normalized we get the output as in the following fig. 5.1

Renter/Floor	Floor 1	Floor 2
Renter 1	9.411	10.588
Renter 2	8	12
Renter 3	10.322	9.677
Renter 4	9.411	10.588

Table 5.2: Renter's normalized values for hostel floors

5.1.3 Analysis

MWBM

Let us check the allocation for Maximum Weight Bipartite Matching(MWBM). Please note that these are only approx values, we will use the complete values in calculation for envy-free.

From the table 5.3 we can clearly that the matching is MWBM. Now we will prove envy-freeness.

Envy freeness

Now let us check the solution for envy-freeness.

- Consider Renter 1 who is assigned Floor 2 for 5.2941176264705888₹.

Renter	Floor	Rent
1	Floor 2	5.294117626470588
2	Floor 2	5.294117626470588
3	Floor 1	4.117647120588236
4	Floor 2	5.294117626470588

Figure 5.1: Solution for Hostel Rent Division

Renter/Floor	Floor 1	Floor 2
Renter 1	9.411	**10.588**
Renter 2	8	**12**
Renter 3	**10.322**	9.677
Renter 4	9.411	1**0.588**

Table 5.3: MWBM for hostel

Renter 1 will have to be satisfied by eq. (5.1):

$$v_{1\sigma(1)} - p_{\sigma(1)} \geq v_{1j} - p_j \quad \forall j \in [n] \tag{5.1}$$

LHS = 10.588235294117647 - 5.294117626470588 = 5.2941176676470585

For RHS

- When j=1 RHS = 9.411764705882353 - 4.117647120588236 = 5.294117585294117
- When j=2 RHS = 10.588235294117647 - 5.294117626470588 = 5.2941176676470585

LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 1.

- Consider Renter 2 who is assigned Floor 2 for 5.294117626470588₹.
  Renter 2 will have to be satisfied by eq. (5.2):

$$v_{2\sigma(2)} - p_{\sigma(2)} \geq v_{2j} - p_j \quad \forall j \in [n] \tag{5.2}$$

LHS = 12 - 5.294117626470588 = 6.705882373529412

For RHS

- When j=1 RHS = 8 - 4.117647120588236 = 3.882352879411764
- When j=2 RHS = 12 - 5.294117626470588 = 6.705882373529412

LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 2.

- Consider Renter 3 who is assigned Floor 1 for 4.117647120588236₹.
  Renter 3 will have to be satisfied by eq. (5.3):

$$v_{3\sigma(3)} - p_{\sigma(3)} \geq v_{3j} - p_j \quad \forall j \in [n] \tag{5.3}$$

LHS = 10.32258064516129 - 4.117647120588236 = 6.204933524573054

For RHS

- When j=1 RHS = 10.32258064516129 - 4.117647120588236 = 6.204933524573054
- When j=2 RHS = 9.67741935483871 - 5.294117626470588 = 4.383301728368122

LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 3.

- Consider Renter 4 who is assigned Floor 2 for 5.2941176264705888₹.

  Renter 4 will have to be satisfied by eq. (5.4):

$$v_{4\sigma(4)} - p_{\sigma(4)} \geq v_{4j} - p_j \quad \forall j \in [n] \tag{5.4}$$

  LHS = 10.588235294117647 - 5.294117626470588 = 5.2941176676470585

  For RHS

  - When j=1 RHS = 9.411764705882353 - 4.117647120588236 = 5.294117585294117
  - When j=2 RHS = 10.588235294117647 - 5.294117626470588 = 5.2941176676470585

  LHS $\geq$ RHS is satisfied, hence allocation is envy-free for Renter 4.

Chapter 6

Conclusion

Linear programming has wide variety of applications and in here we use it to help us solve the fair room allocation with help of PuLP library in python. We solve the fair allocation problem in two steps. In first we find the maximum weight bipartite matching and in the second we solve envy-freeness. As per the observations from all the test cases we can say that we have got an efficient solution. The rent division algorithm defined in (Gal et al., 2016) is at the kernel of our application.

6.1 Results

Finding a fair rent can be quite challenging and thus the concept of Game Theory is for help, with the results that we got it is for sure that the allocation were envy-free with maximizing the utility of each player. There could be many possible envy-free solutions but given solution is better off as it offers fair utility to all the players also taking care of envy-freeness.

6.2 Our Learnings

The approach derived by the authors was interesting and intuitive. Just with help of some maths we got a solution which is far fairer. The best part is this solution is calculated in Linear Time thus making it efficient to go with.

6.3 Future Steps

We have implemented the maximum welfare matching and envy-freeness property using linear programming. We would like extend on this concept to build a web application like spliddit which was used for dividing things fairly such as goods and rent. Now that spliddit is no more online we could plan a project like it to be hosted on the internet.

Bibliography

Edwin KP Chong and Stanislaw H Zak. *An introduction to optimization*, volume 75. John Wiley & Sons, 2013.

Ya'akov (Kobi) Gal, Moshe Mash, Ariel D. Procaccia, and Yair Zick. Which is the fairest (rent division) of them all? In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, page 67–84, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450339360. doi: 10.1145/2940716.2940724. URL https://doi.org/10.1145/2940716.2940724.

Stuart Mitchell, Michael OSullivan, and Iain Dunning. Pulp: a linear programming toolkit for python. *The University of Auckland, Auckland, New Zealand*, 65, 2011.