Hey what’s going on everyone. Today we’re going to learn about how to solve another linear programming problem by using LpMinimize. Let’s read the problem (read the problem).

(create new cell)

First let’s import pulp (type):

import pulp

(run it)

As you can see here, I only typed “import pulp”. From other problems, we had written “from pulp import \*”. This allowed us to write pulp methods without having to write “pulp” before each method from the library. In this example, we will preface each method with “pulp”.

Now from the problem let’s add the variable and their values of max tables, max table size and guests (type):

(create new cell)

max\_tables = 5

max\_table\_size = 4

guests = ‘A B C D E F G I J K L M N O P Q R’.split()

The split() method in Python breaks a string into a list, using whitespace (by default) as the delimiter. Each sequence of characters between spaces becomes an individual element in the resulting list. Let’s see how this looks (type):

print(guests)

(run the cell)

Now let’s define the happiness function that calculates a "happiness score" for a given table by finding the maximum alphabetical distance between the first and last guest's names at the table. (type):

(create new cell)

def happiness(table):

return abs(ord(table[0]) - ord(table[-1]))

(run cell)

ord(character) converts a character (like 'A') into its Unicode code point. For example: (type):

(create cell)

print(ord('A'))

print(ord('C'))

(run cell)

It subtracts the Unicode values of the first (table[0]) and last (table[-1]) guests at the table. abs() ensures the result is positive. This happiness function is used to evaluate the desirability of seating arrangements in the wedding seating model. Tables with higher differences between the first and last guest's alphabetical order are considered "happier."

Let’s create a list of all possible table seatings (type):

(create cell)

possible\_tables = [tuple(c) for c in pulp.allcombinations(guests, max\_table\_size)]

print(possible\_tables)

(run cell)

This code generates all possible combinations of guests for tables, with a maximum size determined by max\_table\_size which we established before is 4 guests. tuple(c) converts each subset into a tuple, ensuring immutability and compatibility for further processing. pulp.allcombinations is a method from the pulp library used to generate all possible subsets of a list, up to a specified maximum size.

Next let’s initialize the linear programming model. The goal is to minimize an objective function. (type):

(create cell)

seating\_model = pulp.LpProblem(‘Wedding\_Seating\_Model’, pulp.LpMinimize)

print(seating\_model)

(run cell)

Next we need to create a dictionary of binary decision variables, where each variable corresponds to whether a specific table (combination of guests) is selected (1) or not (0). (type):

(create cell)

x = pulp.LpVariable.dicts(‘table’, possible\_tables, lowBound=0, upBound=1, cat=pulp.LpInteger)

print(x)

(run cell)

pulp.LpVariable.dicts is a function from the PuLP library to create multiple decision variables efficiently. The variables are stored in a dictionary with keys from possible\_tables. "table" is the prefix for naming the variables, which will appear as "table\_<combination>". lowBound=0 is the minimum value for each decision variable. Here, it’s 0, meaning the table is not selected. upBound=1 is the maximum value for each decision variable. Here, it’s 1, meaning the table is selected. cat=pulp.LpInteger specifies the variable type as an integer. Since the range is 0 to 1, these variables are binary.

Now we need to add an objective function to the optimization problem. (type):

(create cell)

seating\_model += pulp.lpSum([happiness(table) \* x[table] for table in possible\_tables])

print(seating\_model)

(run cell)

The goal is to maximize happiness by assigning guests to tables in a way that maximizes the total happiness score. happiness(table) represents the happiness score of seating a specific combination of guests at a given table. x[table] is a decision variable, indicating whether the specific table arrangement is used (1) or not (0). pulp.lpSum() aggregates the weighted happiness values for all possible table configurations. The summation ensures that only configurations with x[table] = 1 (active tables) contribute to the total happiness.

After establishing the objective function, let’s add a constraint to the seating model, ensuring the total number of tables used does not exceed a specified maximum. (type):

(create cell)

seating\_model += (

pulp.lpSum([x[table] for table in possible\_tables]) <= max\_tables,

‘Maximum\_number\_of\_tables’

)

print(seating\_model)

(run cell)

(in output scroll down to SUBJECT TO)

pulp.lpSum([x[table] for table in possible\_tables]). This calculates the total number of tables used by summing the decision variables x[table]. Each x[table] can only be 0 (table not used) or 1 (table used), since it's defined as a binary variable. The value of our max\_tables variable that we established earlier is 5.

The next set of constraints that we must create, will define the set partitioning problem by guaranteeing that a guest is allocated to exactly one table. (type):

(create cell)

for guest in guests:

seating\_model += (

pulp.lpSum([x[table] for table in possible\_tables if guest in table]) == 1,

f"Must\_seat\_{guest}"

)

(run cell)

For a given guest, this sums up all the binary decision variables x[table] for tables where the guest is included. If a guest is assigned to a table, x[table] will be 1; otherwise, it is 0.

Now we can solve the problem as well as show the solution (type):

(create cell)

seating\_model.solve()

print(f"The chosen tables are out of a total of {len(possible\_tables)}:")

for table in possible\_tables:

if x[table].value() == 1.0:

print(table)

(run the cell)

So here we se out of a total of 4047 tables, these are the chosen tables after evaluating all constraints and conditions.

So that’s a wrap for this video, hope you learned something and thanks for watching.