Exercises - Section 4: Lecture 12 – Constraints with Two Subscripts – Solutions - Part 2

Suppose we have the following notation for a rental car fleet planning model:

DESCRIPTION	MATH	GUROBIPY
Car types: i=1 (small), i=2 (midsize), i=3 (luxury), i=4 (SUV), i=5 (minivan)	i	`i`
Rental locations (100 locations, so j=1,2,…,100)	j	`j`
Forecasted annual rental demand for car type i at location j (forecasted data)	D_{ij}	`D = {(i, j): d for i in range(1, 6) for j in range(1, 101)}` *NOTE: This assumes d is defined elsewhere in the code.*
Cost per car for car type i (known data; same cost regardless of location)	C_i	`C = {i: c for i in range(1, 6)}` *NOTE: This assumes c is defined elsewhere in the code.*
Number of cars of type i to purchase for location j (variable)	x_{ij}	`x = model.addVars([(i, j) for i in range(1, 6) for j in range(1, 101)], name="x")`

NOTE: You might have noticed that in the table above, the gurobipy code is not color-coded like it could be in some editors. We want you to get comfortable with different views of the code, so sometimes you'll see color and sometimes not; it doesn't affect the execution of the code. Similarly, you'll sometimes see different ways of doing the same thing in gurobipy, and different allowable syntaxes. For example, in the table above the name of the variables is defined in double quotes and in the table below it's defined in single quotes. There isn't an execution difference, so we want you to see and be comfortable with both ways.

Fill in the table below, using the notation above to write each of the following constraints mathematically and in Python.

CONSTRAINT	MATH	GUROBIPY
1. Must purchase at least 200 luxury cars across all locations	$\sum_{j=1}^{100} x_{3,j} \geq 200$	`model.addConstr(gp.quicksum(x[3, j] for j in range(1,101)) >= 200, name = 'luxuryMin')`

2. Number of luxury cars purchased across all locations can be no more than 10% of all vehicle purchases	$\frac{\sum_{j=1}^{100} x_{3,j}}{\sum_{i=1}^{5}\sum_{j=1}^{100} x_{ij}} \leq 0.1$ *or* $\sum_{j=1}^{100} x_{3,j} \leq 0.1 \sum_{i=1}^{5}\sum_{j=1}^{100} x_{ij}$	`model.addConstr(gp.quicksum (x[3, j] for j in range(1,101)) <= .1 * gp.quicksum(x[i, j] for i in range(1,6) for j in range(1,101)), name = 'luxuryFrac')` *NOTE: Because gurobipy doesn't accept expressions with variables in the denominator, only code for the second mathematical expression can be written in gurobipy; the first one won't work.*
3. Number of midsize cars purchased at location 4 must be at least 1% of the forecasted annual demand for midsize cars at that location	$x_{2,4} \geq 0.01 D_{2,4}$	`model.addConstr(x[2,4] >= .01 * D[2,4], name = 'midLoc4Min')`
4. Number of midsize cars purchased *at each location* must be at least 1% of the forecasted annual demand for midsize cars at that location	$x_{2,j} \geq 0.01 D_{2,j}$ for each location j *or* $x_{2,j} \geq 0.01 D_{2,j}$ $\forall j = 1, \ldots, 100$ *or* $x_{2,j} \geq 0.01 D_{2,j}$ $\forall j \in \{1, \ldots, 100\}$ *or* $x_{2,j} \geq 0.01 D_{2,j}$ $\forall j$ *NOTE: This last version is a little sloppier, not specifying what values of j there are, but it's often used as shorthand.*	`model.addConstrs((x[2,j] >= .01 * D[2,j]) for j in range(1,101))`

5. Number of vehicles of each type purchased at location 4 must be at least 1% of the forecasted annual demand for that type of vehicle at location 4	$x_{i,4} \geq 0.01D_{i,4}$ for each vehicle type i *or* $x_{i,4} \geq 0.01D_{i,4} \ \forall i$ *or* $x_{i,4} \geq 0.01D_{i,4} \ \forall i = 1, \dots, 5$ *or* $x_{i,4} \geq 0.01D_{i,4} \ \forall i \in \{1, \dots, 5\}$	`model.addConstrs((x[i,4] >= .01 * D[i,4]) for i in range(1,6))`
6. Number of vehicles of each type purchased at each location must be at least 1% of the forecasted annual demand for that type of vehicle at the location	$x_{ij} \geq 0.01D_{ij}$ for each vehicle type i and each location j *or* $x_{ij} \geq 0.01D_{ij} \ \forall i, \forall j$ *or* $x_{ij} \geq 0.01D_{ij} \ \forall i, j$ *or* $x_{ij} \geq 0.01D_{ij} \ \forall i = 1, \dots, 5, \forall j = 1, \dots, 100$ *or* $x_{ij} \geq 0.01D_{ij} \ \forall i \in \{1, \dots, 5\}, \forall j \in \{1, \dots, 100\}$ *NOTE: All of these types of notation are commonly used, but from here on out this course will usually use either the first (more descriptive) or the last*	`model.addConstrs((x[i,j] >= .01 * D[i,j]) for i in range(1,6) for j in range(1,101))`

<table>
<tr><td></td><td>(more mathematically precise).</td><td></td></tr>
<tr><td>7. At location 71, the number of cars of each type (small, midsize, luxury) purchased must be at least 1% of the forecasted annual demand for that type of vehicle at the location</td><td>$x_{i,71} \geq 0.01D_{i,71}$ for each vehicle type i
or
$x_{i,71} \geq 0.01D_{i,71} \ \forall i \in \{1, \dots, 3\}$</td><td><pre>model.addConstrs((x[i,71] >= .01 * D[i,71]) for i in range(1,4))</pre></td></tr>
<tr><td>8. Free upgrades are allowed. When purchasing cars (small, midsize, luxury) at location 71, enough cars of each type need to be purchased to meet at least 1% of the forecasted annual demand with a car of that size or an upgrade. (So, for example, demand for midsize cars could be met by having midsize and luxury cars available.)</td><td>All three of the following constraints are needed:
$x_{3,71} \geq 0.01D_{3,71}$
$\sum_{i=2}^{3} x_{i,71} \geq 0.01 \sum_{i=2}^{3} D_{i,71}$
$\sum_{i=1}^{3} x_{i,71} \geq 0.01 \sum_{i=1}^{3} D_{i,71}$
or
[A more-compact way to write the three constraints together]
$\sum_{i=k}^{3} x_{i,71} \geq 0.01 \sum_{i=k}^{3} D_{i,71} \ \forall k \in \{1,2,3\}$</td><td><pre>model.addConstr(x[3,71] >= .01 * D[3,71])

model.addConstr(gp.quicksum(x[i, 71] for i in range(2,4)) >= .01 * sum(D[i, 71] for i in range(2,4)))

model.addConstr(gp.quicksum(x[i, 71] for i in range(1,4)) >= .01 * sum(D[i, 71] for i in range(1,4)))</pre>
or

[A more-compact way to write the three constraints together]

<pre>model.addConstrs((gp.quicksum(x[i, 71] for i in range(k,4)) >= .01 * sum(D[i, 71] for i in range(k,4))) for k in range (1,4))</pre></td></tr>
<tr><td colspan="3">Notice how the three constraints work together. The first makes sure there are enough luxury cars to meet luxury demand. The second constraint by itself could allow the company to buy all midsize cars and no luxury cars – but the first constraint already disallowed that possibility. So, given the first constraint requiring enough luxury cars, the second constraint requires demand for midsize cars to be met by either midsize or luxury cars. The third constraint has the same sort of effect for small car demand.
This is an example of how modeling is an art; the other constraints we've seen so far have been straightforward mathematical and Python interpretations of the English, but this</td></tr>
</table>

	set of constraints requires more thought about how to get them to interact correctly to get the desired effect.	
9. Free upgrades are allowed. When purchasing cars (small, midsize, luxury) at each location, enough cars of each type need to be purchased to meet at least 1% of the forecasted annual demand with a car of that size or an upgrade. (So, for example, demand for midsize cars could be met by having midsize and luxury cars available.)	*All three of the following constraints are needed:* $x_{3,j} \geq 0.01 D_{3,j}$ $\forall j \in \{1, \dots, 100\}$ $\sum_{i=2}^{3} x_{ij} \geq 0.01 \sum_{i=2}^{3} D_{ij}$ $\forall j \in \{1, \dots, 100\}$ $\sum_{i=1}^{3} x_{ij} \geq 0.01 \sum_{i=1}^{3} D_{ij}$ $\forall j \in \{1, \dots, 100\}$ *or* *[A more-compact way to write the three constraints]* $\sum_{i=k}^{3} x_{ij} \geq 0.01 \sum_{i=k}^{3} D_{ij}$ $\forall k \in \{1,2,3\}$, $\forall j \in \{1, \dots, 100\}$	`model.addConstrs(x[3,j] >= .01 * D[3,j] for j in range(1,101))` `model.addConstrs( gp.quicksum(x[i, j] for i in range(2,4)) >= .01 * sum(D[i, j] for i in range(2,4)) for j in range(1,101))` `model.addConstr(gp.quicksum (x[i, j] for i in range(1,4)) >= .01 * sum(D[i, j] for i in range(1,4)) for j in range(1,101))` *or* *[A more-compact way to write the three constraints]* `model.addConstrs(( gp.quicksum(x[i, j] for i in range(k,4)) >= .01 * sum(D[i, j] for i in range(k,4)) for j in range(1,101)) for k in range (1,4))`

10. The average cost of all vehicles purchased at location 12 can be no more than $40,000	$\frac{\sum_{i=1}^{5} C_i x_{i,12}}{\sum_{i=1}^{5} x_{i,12}} \leq 40{,}000$ *or* $\sum_{i=1}^{5} C_i x_{i,12} \leq 40{,}000 \sum_{i=1}^{5} x_{i,12}$ *NOTE: In the first version, there's a fraction with variables in the numerator and denominator. The second version just multiplies both sides by the denominator to get a linear equation. As you'll see later on in the course, linear functions are preferable, so that version will be used from here on.*	`model.addConstr(gp.quicksum (C[i]*x[i, 12] for i in range(1,6)) <= 40000 * gp.quicksum(x[i, 12] for i in range(1,6)))`
11. The average cost of all vehicles purchased <u>across all locations</u> can be no more than $40,000	$\sum_{i=1}^{5}\sum_{j=1}^{100} C_i x_{ij} \leq 40{,}000 \sum_{i=1}^{5}\sum_{j=1}^{100} x_{ij}$	`model.addConstr(gp.quicksum (C[i]*x[i,j] for i in range(1,6) for j in range(1,101)) <= 40000 * gp.quicksum(x[i,j] for i in range(1,6) for j in range(1,101)))`
12. The average cost of all vehicles purchased <u>at each location</u> can be no more than $40,000	$\sum_{i=1}^{5} C_i X_{ij} \leq 40{,}000 \sum_{i=1}^{5} x_{ij} \quad \forall j \in \{1, \ldots, 100\}$	`model.addConstrs( (gp.quicksum(C[i]*x[i,j] for i in range(1,6)) <= 40000 * gp.quicksum(x[i,j] for i in range(1,6))) for j in range(1,101))`

13. The cost of all vehicles per forecasted rental can be no more than $100	$\frac{\sum_{i=1}^{5}\sum_{j=1}^{100} C_i x_{ij}}{\sum_{i=1}^{5}\sum_{j=1}^{100} D_{ij}} \leq 100$ *or* $\sum_{i=1}^{5}\sum_{j=1}^{100} C_i x_{ij} \leq 100 \sum_{i=1}^{5}\sum_{j=1}^{100} D_{ij}$ *NOTE: In contrast to the constraint on average vehicle cost where there were variables in the denominator, here it's not necessary to multiply both sides by the denominator because the denominator only contains data. It's okay to multiply both sides by the denominator, but it's not necessary.*	`model.addConstr(gp.quicksum (C[i]*x[i,j] for i in range(1,6) for j in range(1,101)) / sum(D[i,j] for i in range(1,6) for j in range(1,101)) <= 100)` *or* `model.addConstr(gp.quicksum (C[i]*x[i,j] for i in range(1,6) for j in range(1,101)) <= 100 * sum(D[i,j] for i in range(1,6) for j in range(1,101)))`

NOTES: