Application of OR Models Using Python

Dr. Sunil Chinta

Sep 28, 2022

Agenda

- Modelling a Network rebalance problem
- Introduction to Solvers & OR-tools
- Solving network rebalance problem using OR-tools
- Pattern matching problem
- Nuggets from practise

Re-balance network inventory

- You received a call from an Inventory analyst that DC-B is soon going to be out of stock for a high selling seasonal item. Time is too short for a Vendor replenishment. Network inventory position revealed that DC-A is sitting on too much of inventory which will likely result in an expensive Markdown!

- You decided to transfer inventory from DC A to DC B. Transportation ops informed you that nodes A and B are not directly connected and inventory has to be flown through transshipment nodes. As this being a busy season and limited free capacity, the incremental flow between the nodes is limited. You can only transfer once due to time constraint.

- You need to determine the maximum you can transfer from DC A to DC B *(Formulate an LP)*

Origin	Destination	Capacity
A	I_1	20
A	I_2	15
I_1	I_3	5
I_1	B	10
I_2	I_3	5
I_2	B	20
I_3	B	10

1. What is the source & the sink

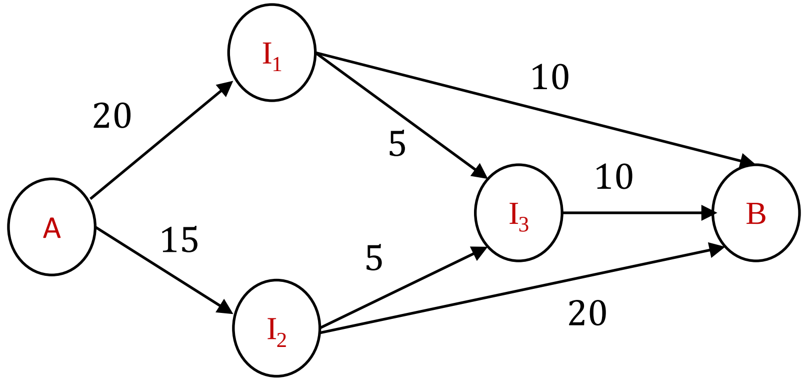
2. Plot network

3. What is the objective?

4. What are the decision variables?

Re-balance network inventory: formulation

Origin	Destination	Capacity
A	I_1	20
A	I_2	15
I_1	I_3	5
I_1	B	10
I_2	I_3	5
I_2	B	20
I_3	B	10



$$Max\ X_{A1} + X_{A2}$$

$$X_{A1} = X_{1B} + X_{13}$$

$$X_{A2} = X_{23} + X_{2B}$$

$$X_{13} + X_{23} = X_{3B}$$

$$0 <= X_{ij} <= U_{ij}$$

(i,j): valid arcs

U_{ij}: Arc capacities

Additional Info:

- These are Max flow problems
- *Ford-Fulkerson* method is an efficient algo. to solve this problem class
- Shortest path & Minimal spanning tree belong to same family
- Solving as an LP always yields an integer solution: Uni-modularity: Think of matrix determinant

OR-Tools

- Open source software suite for Optimization developed by Google
- For Linear programming problems, it has an inbuilt solver called glop *(Google's linear programming system)*
- It can be integrated with commercial or open-source solvers

Installation:
python -m pip install --upgrade --user ortools

Google AI

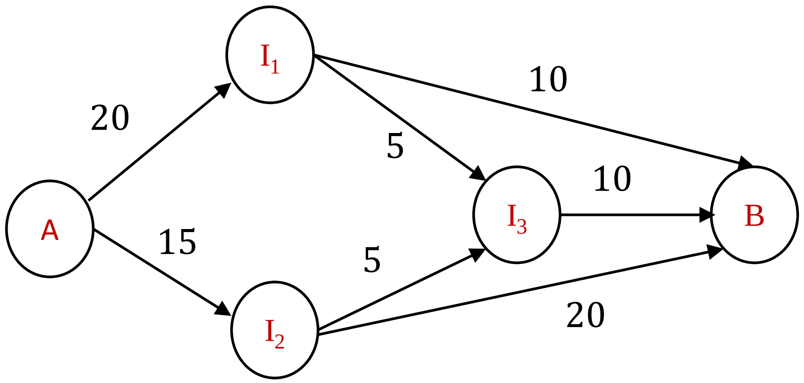
Solvers
Commercial:
- CPLEX: The best!
- Gurobi: Competes with CPLEX
- Local Solver: New Entrant
- FICO, LINDO, LINGO… *(Mid tier)*

Open Source (use with caution!):
- OR-Tools
- COIN-OR *(family of solvers)*

OR-Tools: Basic Syntax

- ortools.linear_solver. pywraplp : Python wrapper for or-tools lp
- pywraplp.Solver: Solver to be used
- NumVar(), BoolVar(), IntVar(): Continuous/Bool/Int variables
- Objective(): Objective function specification
- Constraint(), Add(): Addition of constraints
- Solve(): Optimizes the model
- Sum(): Summation expression over variables/constants

Network Rebalancing (revisited)



$$Max\ X_{A1} + X_{A2}$$

$$X_{A1} = X_{1B} + X_{13}$$

$$X_{A2} = X_{23} + X_{2B}$$

$$X_{13} + X_{23} = X_{3B}$$

$$0 <= X_{ij} <= U_{ij}$$

(i,j): valid arcs

U_{ij}: Arc capacities

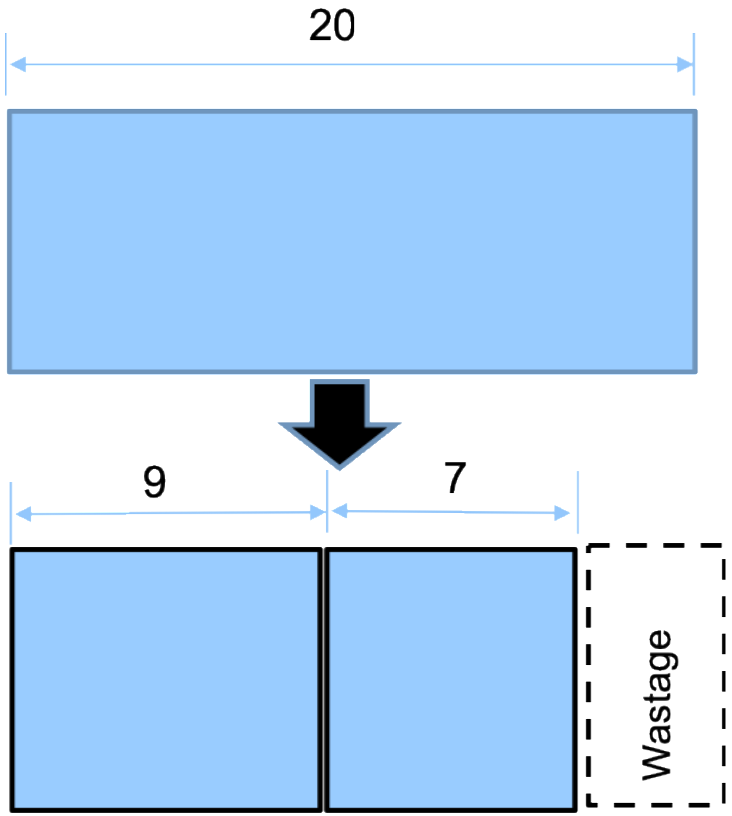
1. Write a Python code to solve this specific formulation
2. Programmatically prove that linear and integer programming yields same solution

```
# import solver
from ortools.linear_solver import pywraplp
#Instantiate a Glop solver
solver = pywraplp.Solver('SolveStigler',
                         pywraplp.Solver.GLOP_LINEAR_PROGRAMMING)
```

```
# Decision variables x_i_j: origin-destination
x_a_1=solver.NumVar(0.0, 20.0, 'x_a_1')
x_a_2=solver.NumVar(0.0, 15.0, 'x_a_2')
x_1_3=solver.NumVar(0.0, 5.0, 'x_a_3')
x_2_3=solver.NumVar(0.0, 5.0, 'x_2_3')
x_1_b=solver.NumVar(0.0, 10.0, 'x_1_b')
x_3_b=solver.NumVar(0.0, 10.0, 'x_3_b')
x_2_b=solver.NumVar(0.0, 20.0, 'x_2_b')

# Objective
objective = solver.Objective()
objective.SetCoefficient(x_a_1, 1)
objective.SetCoefficient(x_a_2, 1)

#objective sign
objective.SetMaximization()

#constraints: Balance constraints
solver.Add(x_a_1-x_1_b-x_1_3 == 0)
solver.Add(x_a_2-x_2_3-x_2_b == 0)
solver.Add(x_1_3+x_2_3-x_3_b == 0)

# solve
result_status = solver.Solve()
```

Apparel pattern making[1]

You are a Vendor from Bangladesh who supplies certain apparel styles to a leading retailer. Retailer orders you for different sizes of the same style and color. You get cloth as rolls with a specific length and width. Each size requires a different width of cloth but same length. You need to determine the number of rolls of cloth required such that the wastage is minimized



Size	Width	Order qty
XL	9	511
L	8	301
M	7	263
S	6	383

- *What is the objective?*
- *What are the decision variables?*

[XL,L,M,S]

[1,0,1,0]

Taken from reference 1

Apparel pattern making- formulation

Size	Width	Order qty
XL	9	511
L	8	301
M	7	263
S	6	383

Feasible/ Relevant patterns	Wastage	Pattern
[2,0,0,0]	2	1
[0,2,0,0]	4	2
[0,0,2,1]	0	3
[0,0,0,3]	2	4
[1,1,0,0]	3	5
[1,0,1,0]	4	6
[1,0,0,1]	5	7
[0,1,1,0]	5	8
[0,1,0,2]	0	9
[0,0,1,2]	1	10

Decision variables

x_j: Number of cuts with pattern j ($x_{1,}$ $x_{2,\ldots}$ $x_{10\ldots}$)

Objective function

Min Wastage

Min $2x_1 + 4x_2 + 0x_3 + 2x_4 + 3x_5 + 4x_6 + 5x_7 + 5x_8 + 0x_9 + x_{10}$

Constraints

$$\begin{bmatrix}2\\0\\0\\0\end{bmatrix} x_1 + \begin{bmatrix}0\\2\\0\\0\end{bmatrix} x_2 + \cdots . + \begin{bmatrix}0\\0\\1\\2\end{bmatrix} x_{10} \geq \begin{bmatrix}511\\301\\263\\383\end{bmatrix}$$

Pattern Making (Revisited)

1. Write a Python code to solve this formulation by taking only possible patterns as input and everything else programmatically computed

2. Programmatically prove minimizing wastage *(including excess)* minimizing cuts yield the same solution

Size	Width	Order qty
XL	9	511
L	8	301
M	7	263
S	6	383

Objective function

Min Wastage

Min $2x_1 + 4x_2 + 0x_3 + 2x_4 + 3x_5 + 4x_6 + 5x_7 + 5x_8 + 0x_9 + x_{10}$

Constraints

$$\begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} x_1 + \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \end{bmatrix} x_2 + \cdots . + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \end{bmatrix} x_{10} \geq \begin{bmatrix} 511 \\ 301 \\ 263 \\ 383 \end{bmatrix}$$

Python List Comprehension (Recap)

- Say, we have a List and want to increment each element by 1
- Traditional For-loop:

```
l = [3, 7, 9, 4]
new_l = []
for e in l:
    new_l.append(e+1)
print(new_l)
>>> [4, 8, 10, 5]
```

- List Comprehension:

```
l = [3, 7, 9, 4]
new_l = [e+1 for e in l]
print(new_l)
>>> [4, 8, 10, 5]
```

- Now say, we want to add respective elements from two lists

```
l1 = [3, 7, 9, 4]
l2 = [6, 2, 9, 12]
new_l = [e1 + e2 for (e1, e2) in zip(l1, l2)]
print(new_l)
>>> [9, 9, 18, 16]
```

Pattern Making (Continued)

```
# decision var
x_var = [solver.IntVar(0,solver.infinity(),"x_{}".format(i +1))
         for i in range(len(patterns))
         ]
```

```
solver.Minimize(solver.Sum([w * x for w, x in zip(wastage, x_var)]))
```

```
# Constraints
for i in range(len(sizes)):
    solver.Add(solver.Sum([p[i] * x
                           for p,x in zip(patterns,x_var)])
               >= demand[i],
              "cons_size_{}".format(sizes[i]))
```

Pattern Making (Continued)

```
for i in range(len(patterns)):
    print("number of cuts for pattern {}={}".format(
        x_var[i].name(), x_var[i].solution_value()))
```

Objective: num_cuts

```
solver.Minimize(solver.Sum([ x for w, x in zip(wastage, x_var)]))
```

Objective: Excess Wastage

```
expr1=solver.Sum([w * x for w, x in zip(wastage, x_var)])
expr2=solver.Sum(-1*demand[i]*sizes[i] for i in range(len(sizes)))
expr3=solver.Sum([p[i] *sizes[i]* x for p,x in zip(patterns,x_var) for i in range(len(sizes))])
print expr2
solver.Minimize(expr1+expr2+expr3)
```

Nuggets from Practise

- Modelling optimization problems has also an element of Art
- It’s paramount to tame the complexity. Problem can easily become intractable
- Decompose the problem wherever possible
- Analyse the trade-offs between solving a problem exactly vs heuristically
- Mathematical optima need not always translate to business optima
- Infeasibility is a big issue in practise. Debugging it is lot more harder
- There is always a trade-off between *“Exact solution of an approximate model”* and *“Approximate solution of an exact model”*

Git Repository

Slides and the associated code for the session can be downloaded from

chintasunny/vit-or-modeling-2022 (github.com)

References

1. OPERATIONS RESEARCH : PRINCIPLES AND APPLICATIONS, G. SRINIVASAN

2. OR-Tools | Google Developers