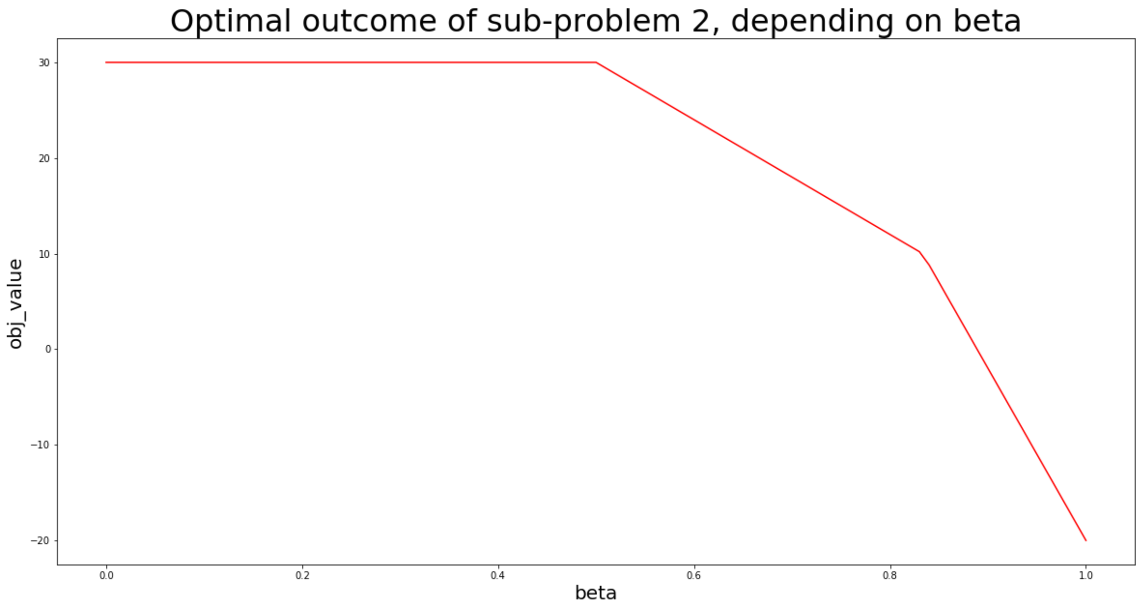
supplychaindataanalytics.com

Multi-objective LP with sub-problem weights - SCDA

Linnart Felkl M.Sc.

4-5 minutes



In my most recent post on linear programming I applied PuLP for solving below linear optimization problem, using two approaches. Approach #1 was based on solving a sub-problem with one objective only first, then adding the optimal outcome of that problem to a second sub-problem that considered objective two only. Approach #2 is based on combining all objectives into one overall objective function, combining single objectives with scalar weights.

In this post I want to show an alternative approach for solving below multi-objective linear optimization problem: I will use approach #1, but apply weights instead:

$$
\begin{aligned}
\max_{x_1,x_2} \quad & 2x_1 + 3x_2 \\
\max_{x_1,x_2} \quad & 4x_1 - 2x_2 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \leq 10 \\
& 2x_1 + x_2 \leq 15 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{R}
\end{aligned}
$$

The approach I now use is to solve the problem in two steps, where a sub-problem with one objective only is solved first and its optimal outcome is added to a second sub-problem with only the the second objective as a constraint. This approach has been demonstrated by before, but this time I will apply a scalar weight-factor to that constraint, considering the optimal outcome of sub-problem with 0-100%.

Starting with the first objective, the first sub-problem to solve would be the following:

$$
\begin{aligned}
\max_{x_1,x_2} \quad & 2x_1 + 3x_2 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \leq 10 \\
& 2x_1 + x_2 \leq 15 \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{R}
\end{aligned}
$$

The optimal objective value to above sub-problem is 30.

After having solved the first sub-problem, the second objective would be considered by a second sub-problem adding the optimal outcome to above problem as a weighted constraint:

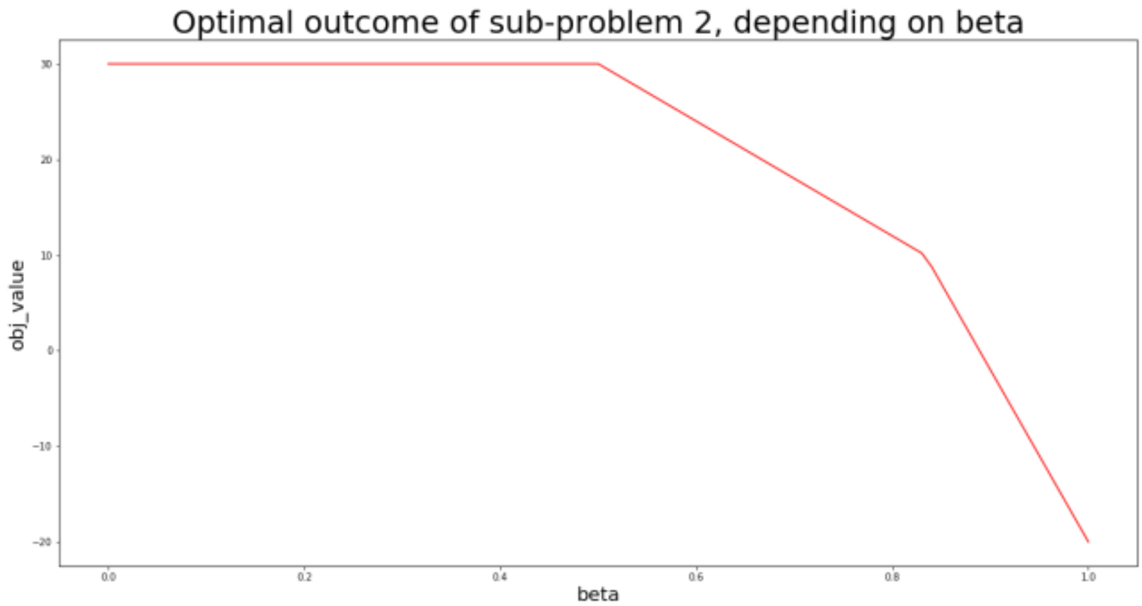
$$
\begin{aligned}
\max_{x_1,x_2} \quad & 4x_1 - 2x_2 \\
\text{s.t.} \quad & \\
& x_1 + x_2 \leq 10 \\
& 2x_1 + x_2 \leq 15 \\
& 2x_1 + 3x_2 \geq 30 * \beta \\
& x_1, x_2 \geq 0 \\
& x_1, x_2 \in \mathbb{R} \\
& \beta \in [0; 1]
\end{aligned}
$$

Here is the implementation in Python, using the PuLP module and applying a step-size of 0.01:

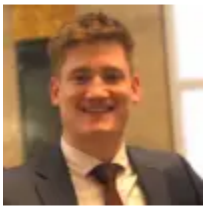
```
# import PuLP for modelling and solving problems
import PuLP

# import matplotlib.pyplot for visualization
import matplotlib.pyplot as plt

# import pandas and numpy for being able to store solutions in DataFrame
import numpy as np
import pandas as pd

# define step-size
stepSize = 0.01

# initialize empty DataFrame for storing optimization outcomes
solutionTable = pd.DataFrame(columns=
["beta","x1_opt","x2_opt","obj_value"])

# declare optimization variables using PuLP and LpVariable
x1 = PuLP.LpVariable("x1",lowBound=0)
x2 = PuLP.LpVariable("x2",lowBound=0)

# model and solve sub-problem no. 1
linearProblem = PuLP.LpProblem("First sub-
problem",PuLP.LpMaximize)
linearProblem += 2*x1 + 3*x2 # add objective no. 1
linearProblem += x1 + x2 <= 10 # add contraints from original
problem statement
linearProblem += 2*x1 + x2 <= 15
```

```
solution = linearProblem.solve()

# store optimal outcome of sub-problem no. 1 into variable
optimalObj1 = PuLP.value(linearProblem.objective)

# iterate through beta values from 0 to 1 with stepSize, and write
PuLP solutions into solutionTable
for i in range(0,101,int(stepSize*100)):
    # declare the problem again
    linearProblem = PuLP.LpProblem("Multi-objective linear
maximization",PuLP.LpMaximize)
    # add the second objective as objective function to this
sub-problem
    linearProblem += 4*x1-2*x2
    # add the constraints from original problem statement
    linearProblem += x1 + x2 <= 10
    linearProblem += 2*x1 + x2 <= 15
    # add additional constraint at level beta, considering
optimal outcome of sub-problem no. 1
    linearProblem += 2*x1 + 3*x2 >= (i/100)*optimalObj1
    # solve the problem
    solution = linearProblem.solve()
    # write solutions into DataFrame
    solutionTable.loc[int(i/(stepSize*100))] = [i/100,
                                                 PuLP.value(x1),
                                                 PuLP.value(x2),

PuLP.value(linearProblem.objective)]

# visualize optimization outcome, using matplotlib.pyplot
# -- set figure size
plt.figure(figsize=(20,10))
```

```
# -- create line plot
plt.plot(solutionTable["beta"],solutionTable["obj_value"],color="red")
# -- add axis labels
plt.xlabel("beta",size=20)
plt.ylabel("obj_value",size=20)
# -- add plot title
plt.title("Optimal outcome of sub-problem 2, depending on
beta",size=32)
# -- show plot
plt.show()
```



The results indicate that there is some range for beta where an increase in beta will not affect the optimal outcome of sub-problem 2, i.e. will not affect objective 2.



Data scientist focusing on simulation, optimization and modeling in R, SQL, VBA and Python

Post navigation