DAT600: Algorithm Theory - Assignment 4

Antón Maestre Gómez - Daniel Linfon Ye Liu

Task 1 Exam Problem:

Information

The task is to determine the **optimal weekly production quantities** of products **X** and **Y** to **maximize profit**, given:

- Time Requirements (in minutes):

	Machine Time	**Craftsman Time**
Product X	15	20
Product Y	20	30

- Resource Availability (per week):
  - Machine time: **40 hours = 2400 minutes**
  - Craftsman time: **35 hours = 2100 minutes**
- Costs:
  - Machine: **£100/hour** = **£1.667/minute**
  - Craftsman: **£20/hour** = **£0.333/minute**
- Revenue:
  - Product X: **£200**
  - Product Y: **£300**
- Constraint:
  - At least 10 units of **Product X** must be produced per week.

LP Model Formulation

Let:

- x = number of units of **Product X** produced.
- y = number of units of **Product Y** produced.

The final objective is to maximize the profit, which can be calculated as:

Profit = Revenue - Cost

Revenue = $200x + 300y$

Costs:

- Machine: $15x + 20y$ minutes → $(15x + 20y) \times 1.667$
- Craftsman: $20x + 30y$ minutes → $(20x + 30y) \times 0.333$

Total Profit:

$$\text{Profit} = 200x + 300y - 1.667(15x + 20y) - 0.333(20x + 30y)$$

$$= 200x + 300y - (25.005x + 33.34y) - (6.66x + 9.99y)$$

$$= (200 - 25.005 - 6.66)x + (300 - 33.34 - 9.99)y = 168.335x + 256.67y$$

So we need to maximize:

$$\text{Profit} = 168.335x + 256.67y$$

However, it must be taken into account that:

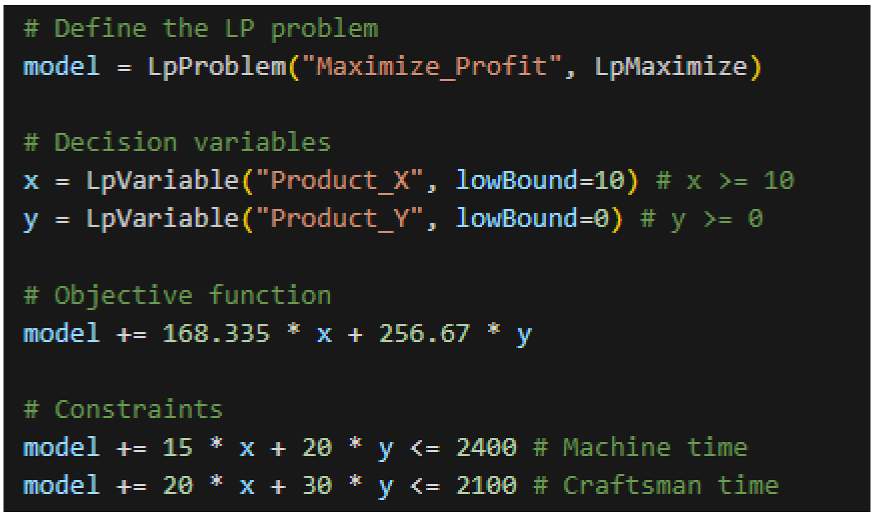
1. Machine time: $15x + 20y \leq 2400$
2. Craftsman time: $20x + 30y \leq 2100$
3. Product X: $x \geq 10$
4. $x, y \geq 0$

Python Solution

As requested in the assignment, a program has been designed to solve the problem. With that objective, the **PuLP** library is used. PuLP is a Python package specifically designed for linear programming. It provides a straightforward interface to define objective functions, constraints, and decision variables.

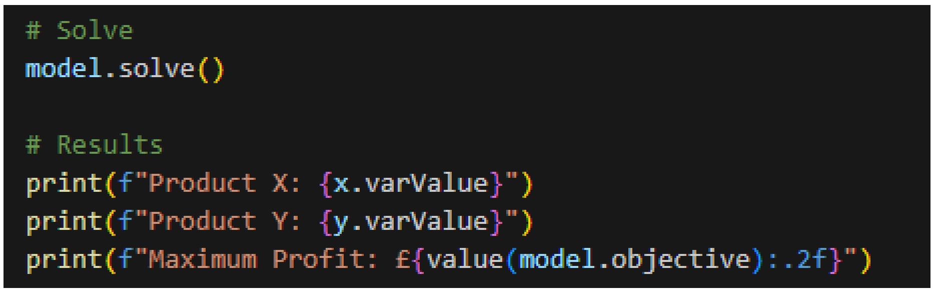
The following components were imported from the PuLP library:

- `LpProblem`: This class allows for the creation of a linear programming model. It serves as the container that holds the objective function and constraints.
- `LpMaximize`: This class is used to indicate that the objective of the problem is to **maximize** a function, in this case, the total weekly profit.
- `LpVariable`: This class is used to define the **decision variables**, representing the number of products X and Y to be produced. Using the `lowBound` parameter, the minimum value that each variable can take is defined.
- `LpInteger`: This constant is used to **define decision variables as integers**, which is necessary in cases where fractional values are not realistic, in this case, the number of products must be a whole number. By setting `cat=LpInteger` in the variable declaration, the solver is forced to search for integer-only solutions.



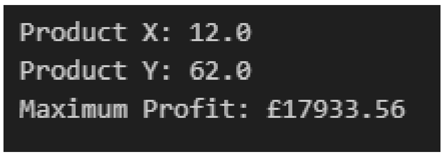
```python
# Define the LP problem
model = LpProblem("Maximize_Profit", LpMaximize)

# Decision variables
x = LpVariable("Product_X", lowBound=10) # x >= 10
y = LpVariable("Product_Y", lowBound=0) # y >= 0

# Objective function
model += 168.335 * x + 256.67 * y

# Constraints
model += 15 * x + 20 * y <= 2400 # Machine time
model += 20 * x + 30 * y <= 2100 # Craftsman time
```

- `value`: This function is used to extract the numerical solution (value) from the optimized objective function once the model has been solved.



```python
# Solve
model.solve()

# Results
print(f"Product X: {x.varValue}")
print(f"Product Y: {y.varValue}")
print(f"Maximum Profit: £{value(model.objective):.2f}")
```

After running the program, the following result is obtained:



```
Product X: 12.0
Product Y: 62.0
Maximum Profit: £17933.56
```

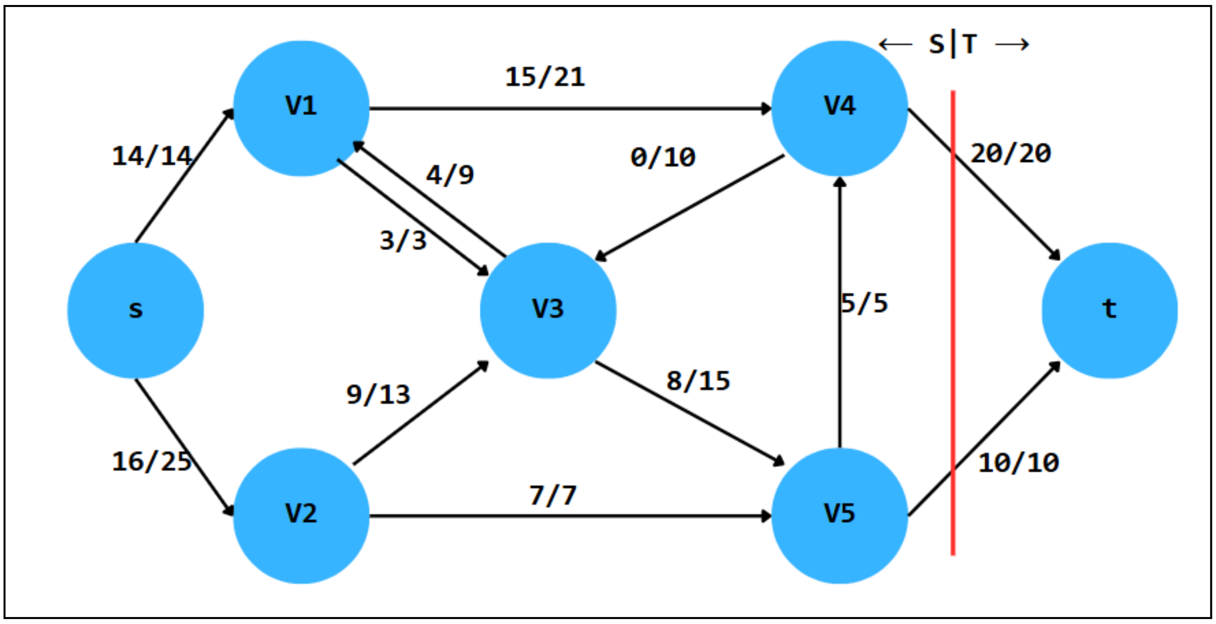
The optimal solution recommends producing **12 units of Product X** and **62 units of Product Y**, resulting in a **maximum profit of £17,933.56**. This solution efficiently uses available resources and confirms that focusing on Product Y maximizes profitability while meeting all constraints.

Task 2 Problem Maximum Flow:

a) Frame the problem of identifying the bottleneck network cut that separates the source from the sink as an optimization problem, and subsequently address and resolve this problem programmatically using the optimization package of your choice.

b) Frame the problem of finding the maximum flow at the edges of the network as an optimization problem and solve it with the optimization package of your choice.

After applying the **Ford-Fulkerson algorithm** as in the previous assignment:



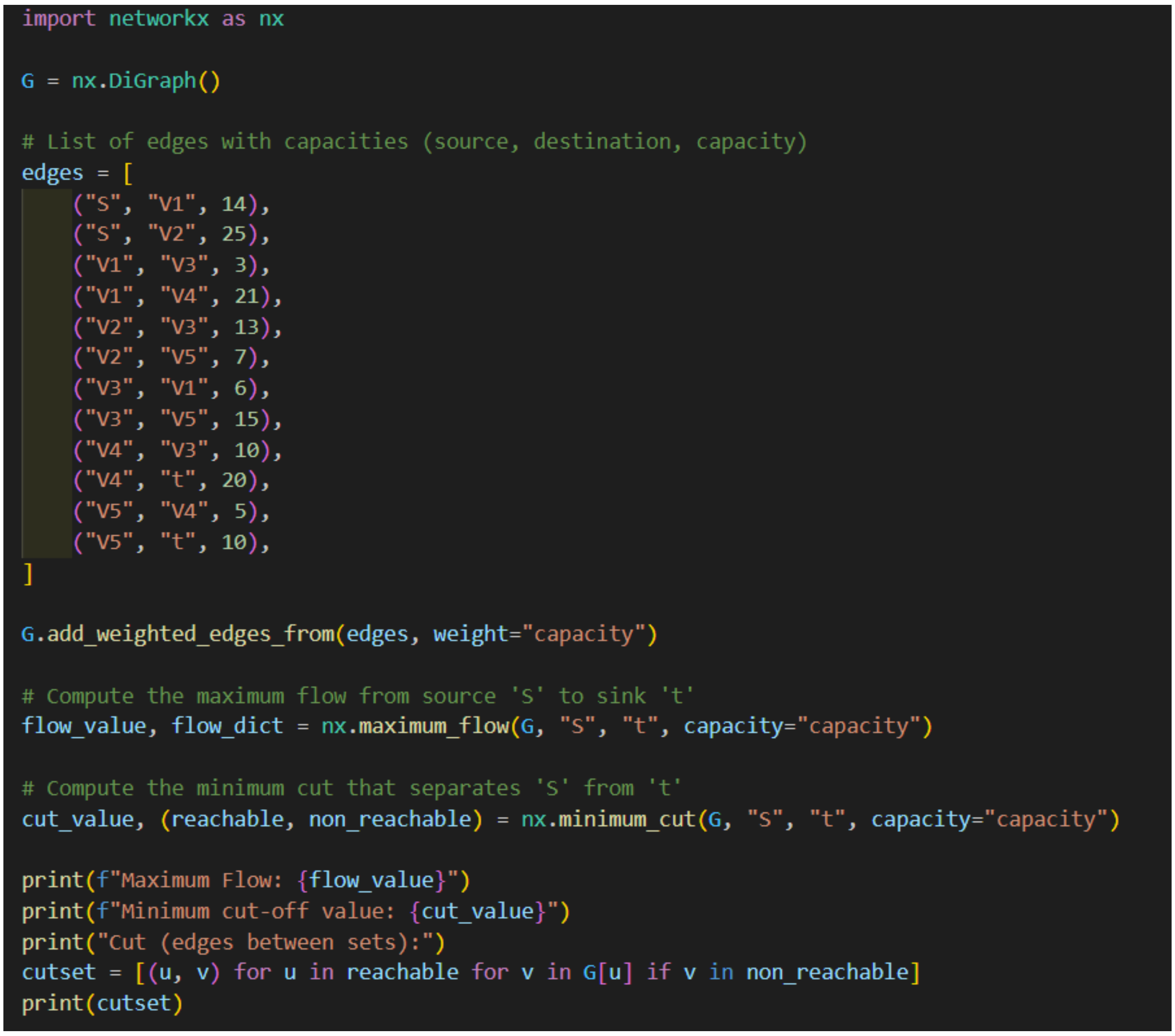
Maximum flow = **30**

S = **{s, V1, V2, V3, V4, V5}**
T = **{t}**

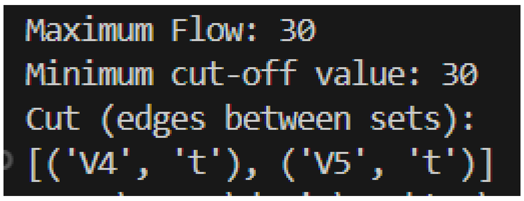
f (S, T) = 20 +10 = **30**
c (S, T) = 20 + 10 = **30**

Python solution

We used the package `networkx,` a Python library designed for the creation, manipulation, and analysis of complex networks and graphs.



```python
import networkx as nx

G = nx.DiGraph()

# List of edges with capacities (source, destination, capacity)
edges = [
    ("S", "V1", 14),
    ("S", "V2", 25),
    ("V1", "V3", 3),
    ("V1", "V4", 21),
    ("V2", "V3", 13),
    ("V2", "V5", 7),
    ("V3", "V1", 6),
    ("V3", "V5", 15),
    ("V4", "V3", 10),
    ("V4", "t", 20),
    ("V5", "V4", 5),
    ("V5", "t", 10),
]

G.add_weighted_edges_from(edges, weight="capacity")

# Compute the maximum flow from source 'S' to sink 't'
flow_value, flow_dict = nx.maximum_flow(G, "S", "t", capacity="capacity")

# Compute the minimum cut that separates 'S' from 't'
cut_value, (reachable, non_reachable) = nx.minimum_cut(G, "S", "t", capacity="capacity")

print(f"Maximum Flow: {flow_value}")
print(f"Minimum cut-off value: {cut_value}")
print("Cut (edges between sets):")
cutset = [(u, v) for u in reachable for v in G[u] if v in non_reachable]
print(cutset)
```



```
Maximum Flow: 30
Minimum cut-off value: 30
Cut (edges between sets):
[('V4', 't'), ('V5', 't')]
```

We can verify that the result of the calculation of the maximum flow and the minimum cut coincides with those obtained by hand.

Github files

https://github.com/AntonMG4/DAT600_AlgorithmTheory/tree/main/Assignment4