MAT3007 - Assignment 1

Yohandi [SID: 120040025]

Problem 1. Modeling

a.

$$\begin{aligned}\max_{x_1,x_2}\ & 7.8x_1 + 7.1x_2 \\ \texttt{s.t.}\ & \frac{1}{8}x_1 + \frac{1}{4}x_2 \le 90 \\ & \frac{1}{2}x_1 + \frac{1}{6}x_2 \le 80 \\ & x_1, x_2 \ge 0\end{aligned}$$

b.

$$\begin{aligned}\min_{x_1,x_2}\ & -7.8x_1 - 7.1x_2 \\ \texttt{s.t.}\ & \frac{1}{8}x_1 + \frac{1}{4}x_2 + s_1 = 90 \\ & \frac{1}{2}x_1 + \frac{1}{6}x_2 + s_2 = 80 \\ & x_1, x_2, s_1, s_2 \ge 0\end{aligned}$$

c. By letting x_3 as the number of overtime assembly labor that is scheduled. Then, $x_3 \le 40$ due to the limitation and $-8x_3$ due to the additional cost.

$$\begin{aligned}\min_{x_1,x_2}\ & -7.8x_1 - 7.1x_2 + 8x_3 \\ \texttt{s.t.}\ & \frac{1}{8}x_1 + \frac{1}{4}x_2 - x_3 + s_1 = 90 \\ & \frac{1}{2}x_1 + \frac{1}{6}x_2 + s_2 = 80 \\ & x_3 \le 40 \\ & x_1, x_2, x_3, s_1, s_2 \ge 0\end{aligned}$$

d. Python code is as follows:

Privacy - Terms

```
import cvxpy as cp

if __name__ == "__main__":
    x = [cp.Variable(nonneg = True) for i in range(2)] # 0-based index

    obj = cp.Maximize(7.8 * x[0] + 7.1 * x[1])
    consts = [
        x[0] / 8 + x[1] / 4 <= 90,
        x[0] / 2 + x[1] / 6 <= 80,
    ]

    print("Maximum objective value:", cp.Problem(obj, consts).solve())
    print("Obtainable by having x1 =", x[0].value, "and x2 =", x[1].value)
```

```
Maximum objective value: 2759.9999981806723
Obtainable by having x1 = 47.999999790375576 and x2 = 335.99999997404825
```

The maximum objective value can be obtained is 2760 (note that this differs a bit from the console output due to some precision error). This is obtainable by having $x_1 = 48$ and $x_2 = 336$.

Problem 2. Reformulate NLP as LP

NLP form:

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & 2x_2 + |x_1 - x_3| \\ \texttt{s.t.} \quad & |x_1 + 2| + |x_2| \leq 5 \\ & x_3^2 \leq 1 \end{aligned}$$

Let y_1, y_2, and y_3 as variables that satisfy:

- $y_1 = |x_1 - x_3|$
- $y_2 = |x_1 + 2|$
- $y_3 = |x_2|$

Then,

- $y_1 \geq x_1 - x_3$
- $y_1 \geq -(x_1 - x_3)$
- $y_2 \geq x_1 + 2$
- $y_2 \geq -(x_1 + 2)$
- $y_3 \geq x_2$
- $y_3 \geq -x_2$

Reformulated NLP form (LP form):

$$
\begin{aligned}
\min_{x_1,x_2,x_3} \quad & 2x_2 + y_1 \\
\texttt{s.t.} \quad & y_2 + y_3 \leq 5 \\
& y_1 \geq x_1 - x_3 \\
& y_1 \geq -(x_1 - x_3) \\
& y_2 \geq x_1 + 2 \\
& y_2 \geq -(x_1 + 2) \\
& y_3 \geq x_2 \\
& y_3 \geq -x_2 \\
& x_3 \leq 1 \\
& x_3 \geq -1
\end{aligned}
$$

Problem 3.

Let A_i and B_i denote the number of current and needed cars in region i, respectively. Also, let $C_{i,j}$ denotes the cost of moving one car from region i to region j.
Denote $X_{i,j}$ as the number of cars that move from region i to region j. Then,

- Objective: $\min_X \sum_{i=1}^5 \sum_{j=1}^5 X_{i,j} C_{i,j}$

This is obtained due to the cost of moving such number of cars from region i to region j.

- Constraint: $B_i - A_i \leq \sum_{j=1}^5 X_{j,i} - X_{i,j}$

This is obtained as the number of additional cars, which is defined as the number of cars that come and go, must satisfies the $\texttt{need} - \texttt{current}$. Alternatively, we may replace $\leq$ with $=$ and still have the same result (because the optimal solution will not exceed the $\texttt{need}$ and waste additional costs). However, it might gets trickier if $B_i - A_i$ is negative.
Then, our optimization problem is formulated as follows:

$$
\begin{aligned}
\min_{X} \quad & \sum_{j=1}^{5} \sum_{j=1}^{5} C_{i,j} X_{i,j} \\
\texttt{s.t.} \quad & \sum_{j=1}^{5} X_{j,1} - X_{1,j} \geq 150 - 110 \\
& \sum_{j=1}^{5} X_{j,2} - X_{2,j} \geq 200 - 335 \\
& \sum_{j=1}^{5} X_{j,3} - X_{3,j} \geq 600 - 400 \\
& \sum_{j=1}^{5} X_{j,4} - X_{4,j} \geq 420 - 200 \\
& \sum_{j=1}^{5} X_{j,5} - X_{5,j} \geq 610 - 390 \\
& X_{i,j} \geq 0, \forall i, j
\end{aligned}
$$

Python code is as follows.

```
import cvxpy as cp

if __name__ == "__main__":
    C = [[0, 20, 13, 11, 28],
         [20, 0, 18, 8 , 46],
         [13, 18, 0, 9, 27],
         [11, 8, 9, 0, 20],
         [28, 46, 27, 20, 0]
    ]

    A = [110, 335, 400, 420, 610]
    B = [150, 200, 600, 200, 390]

    X = cp.Variable((5, 5), nonneg = True) # 0-based index

    obj = cp.Minimize(cp.sum(cp.multiply(C, X)))
    consts = [
        cp.sum(X[:, i]) - cp.sum(X[i, :]) >= B[i] - A[i] for i in range(5)
    ]

    print("Minimum objective value:", cp.Problem(obj, consts).solve())
    print("Obtainable by having X =\n", X.value)
```

```
Minimum objective value: 2400.000000082946
Obtainable by having X =
 [[3.36932606e+02 0.00000000e+00 8.38948986e-10 0.00000000e+00
   0.00000000e+00]
 [4.79422167e-08 3.36932606e+02 5.36449206e-08 1.99999999e+01
   0.00000000e+00]
 [2.61796034e-09 0.00000000e+00 3.36932606e+02 0.00000000e+00
   0.00000000e+00]
 [3.99999999e+01 3.14262769e-10 2.00000000e+02 3.36932606e+02
   0.00000000e+00]
 [3.17357437e-09 0.00000000e+00 1.79862029e-09 1.11526483e-09
   3.36932606e+02]]
```

The minimum objective value can be obtained is 2400 (again, this differs a bit from the console output due to some precision error). This is obtainable by having

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 40 & 0 & 200 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The main diagonal (from $(1,1)$ to $(5,5)$) can be ignored as it won't change the optimal value (also with a cost of 0). From the solution, we should:

- move 20 cars from region 2 to 4
- move 40 cars from region 4 to 1
- move 200 cars from region 4 to 3

According to the solution, we will have the cost of $20 \times 8 + 40 \times 11 + 200 \times 9 = 2400$.

Problem 4.

Let E as a set of edges that represent the displayed graph and W as a $n \times n$ matrix that represent the length from a path to another path. Then,

- Objective: $\min_{(u,v) \in E} w_{u,v} x_{u,v}$

, where $x_{u,v}$ is a decision variable that shows the path information: 1 if we go through from u to v, 0 if we don't.

- Constraints:
  - $\sum_{v \neq 1} x_{1,v} = 1$ and $\sum_{u \neq n} x_{u,n} = 1$

This makes the flow out from S and the flow in to T become 1. In a sense, if the flow out is not 1, then the current minimized objective value is not optimal because at least one of the flow out can be removed and still hold the shortest path (same goes for flow in).

- $\sum_{u \neq k} x_{u,k} - \sum_{v \neq k} x_{k,v} = 0$

This makes the flow in equal to the flow out for any intermediate nodes. The reason is simple: we want to maintain the previous idea of constraint to any intermediate node, which is to make the flow in and out equal to 1. With this, only one simple path is constructed (no unnecessary branches).

- $x_{u,v} \leq 1, \forall u, v$

As previously defined, $x_{u,v}$ is a decision variable; hence, the value must be $\{0, 1\}$.

```
import cvxpy as cp

if __name__ == "__main__":
    INF = 999999

    n = 8
    W = [
        [INF, 5, 4, INF, INF, INF, INF, INF],
        [5, INF, INF, 3, INF, 7, INF, INF],
        [4, INF, INF, INF, 1, 2, INF, INF],
        [INF, 3, INF, INF, 2, INF, INF, INF],
        [INF, INF, 1, 2, INF, INF, 2, 5],
        [INF, 7, 2, INF, INF, INF, INF, 3],
        [INF, INF, INF, INF, 2, INF, INF, 1],
        [INF, INF, INF, INF, 5, 3, 1, INF]
    ]

    X = cp.Variable((n, n), nonneg = True)

    obj = cp.Minimize(cp.sum(cp.multiply(W, X)))
    consts = [
        cp.sum(X[0, :]) == 1,
        cp.sum(X[:, -1]) == 1
    ] + [
        cp.sum(X[k, :]) - cp.sum(X[:, k]) == 0 for k in range(1, n - 1)
    ] + [
        X[u, v] <= 1 for u in range(n) for v in range(n)
    ]

    print("Minimum objective value:", cp.Problem(obj, consts).solve())
    print("Obtainable by having X =\n", X.value)

    [print(f"Take edge that connects {u + 1} and {v + 1}.") for u in range(n)
     for v in range(n) if X[u, v].value > 0.5]
```

```
Minimum objective value: 8.000002406936602
Obtainable by having X =
 [[0.00000000e+00 8.73868123e-11 1.00000000e+00 0.00000000e+00
  0.00000000e+00 0.00000000e+00 4.28169070e-14 9.65205990e-14]
 [4.87287356e-11 0.00000000e+00 0.00000000e+00 5.22044440e-11
  0.00000000e+00 2.27546316e-11 1.40434004e-13 1.93701797e-13]
 [5.88667707e-11 1.20795408e-13 0.00000000e+00 1.26320076e-13
  1.00000000e+00 2.01944977e-10 2.72162370e-13 3.25429804e-13]
 [0.00000000e+00 2.39355658e-11 0.00000000e+00 0.00000000e+00
  6.78182005e-11 0.00000000e+00 1.34922244e-13 1.88173349e-13]
 [9.23374037e-14 7.64852914e-14 5.12295265e-11 3.96928470e-11
  0.00000000e+00 0.00000000e+00 1.00000000e+00 4.85537527e-11]
 [9.73577746e-14 1.24662033e-11 2.99377576e-11 8.70060439e-14
  0.00000000e+00 0.00000000e+00 2.32848230e-13 2.06740396e-10]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  2.44622434e-11 0.00000000e+00 0.00000000e+00 1.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
  1.59361509e-11 2.55447338e-11 6.60719199e-11 1.77844574e-13]]
Take edge that connects 1 and 3.
Take edge that connects 3 and 5.
Take edge that connects 5 and 7.
Take edge that connects 7 and 8.
```

The result shows that we should take the $1-3-5-7-8$ path to obtain the minimum objective value, which is 8. The illustration is shown below.