Lab06-Linear Programming

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name: Log Creative Student ID: Email: logcreative-lzl@sjtu.edu.cn

1. *Hirschberg Algorithm.* Recall the **String Similarity** problem in class, in which we calculate the edit distance between two strings in a sequence alignment manner.

   (a) Implement the algorithm combining **dynamic programming** and **divide-and-conquer** strategy in C/C++. Analyze the time complexity of your algorithm. (The template *Code-SequenceAlignment.cpp* is attached on the course webpage).

   Solution. The function to be implemented is shown as follows:

```
1  int editDistanceDP(string &X, string &Y, const int &delta, vector<vector<int>> *dpo =
       NULL) {
2    // dynamic programming ...
3    if (dpo) *dpo = dp;
4    return dp[m][n];
5  }
6
7  int editDistanceDP(string &X, string &Y, int xLow, int yLow, int xHigh, int yHigh,
       const int &delta, bool rev, vector<int> &dist) {
8    string _X, _Y;
9    int m = xHigh - xLow;
10   int n = yHigh - yLow;
11   _X = X.substr(xLow, m);
12   _Y = Y.substr(yLow, n);
13   if (rev) {
14     reverse(_X.begin(), _X.end());
15     reverse(_Y.begin(), _Y.end());
16   }
17
18   vector<vector<int>> dpo(m + 1);
19   for (int i = 0; i <= m; ++i) {
20     dpo[i].resize(n + 1);
21   }
22   editDistanceDP(_X, _Y, delta, &dpo);
23
24   for (int x = 0; x <= xHigh - xLow; ++x)
25     dist.push_back(dpo[x][n]);
26
27   return 0;
28 }
```

   Note that the dynamic programming function is also modified to get the array directly. Analyze the time complexity of the implemented function:

   Initialize and Reverse. The smart `substr()` and `reverse()` only cost $O(m+n)$.

   Dynamic Programming. Calculating the DP array costs $O(mn)$.

   Store dist. Cost $O(m)$.

   Hence the function costs $O(mn)$. According to the lecture, the time complexity of the whole algorithm is $O(mn)$. □

(b) Given $\alpha(x, y) = |ascii(x) - acsii(y)|$, where $ascii(c)$ is the ASCII code of character c, and $\delta = 13$. Find the edit distance between the following two strings.

$$X[1..60] = CMQHZZRIQOQJOCFPRWOUXXCEMYSWUJ$$
$$TAQBKAJIETSJPWUPMZLNLOMOZNLTLQ$$

$$Y[1..50] = SUYLVMUSDROFBXUDCOHAATBKN$$
$$AAENXEVWNLMYUQRPEOCJOCIMZ$$

Solution. The running result is 385.

DP: 385
DP+DC: 385
(0, 0) (1, 0) (2, 0) (3, 1) (4, 1) (5, 2) (6, 3) (7, 3) (8, 4) (9, 5) (10, 6) (11, 7) (12, 7) (13, 8) (14, 9) (15, 9) (16, 10) (17, 11) (18, 11) (19, 12) (20, 13) (21, 14) (22, 15) (23, 16) (24, 17) (25, 18) (26, 18) (27, 19) (28, 19) (29, 19) (30, 20) (31, 20) (32, 21) (33, 22) (34, 23) (35, 24) (36, 25) (37, 26) (38, 27) (39, 28) (40, 29) (41, 30) (42, 31) (43, 32) (44, 33) (45, 34) (46, 35) (47, 36) (48, 37) (49, 38) (50, 39) (51, 40) (52, 41) (53, 42) (54, 43) (54, 44) (55, 45) (56, 46) (57, 47) (58, 48) (59, 49) (60, 50)

□

2. *Travelling Salesman Problem.* Given a list of cities and the distances between each pair of cities ($G = (V, E, W)$), we want to find the shortest possible route that visits each city exactly once and returns to the origin city. Similar to **Maximum Independent Set** and **Dominating Set**, please turn the traveling salesman problem into an ILP form.

   Remark: W is the set of weights corresponds to the edges that connecting adjacent cities.

   Solution. Let v_0 be the original city, $v_1, v_2, \cdots, v_n$ be the cities to be visited. Suppose c_e is the 0-1 indicator of edge e, such that

$$c_e = \begin{cases} 0, & e \text{ is selected in the route} \\ 1, & e \text{ is not selected in the route} \end{cases}$$

   Denote w_e to be the weight of the edge. Then the ILP form could be as follows:

$$\begin{aligned} &\min \sum_{e \in E} c_e w_e \\ s.t. &\sum_{e \in E} c_e [v_0 \in e] = 2, \\ &\sum_{e \in E} c_e [v_i \in e] = 1 \qquad \forall i \in \mathbb{N}_+ \end{aligned}$$

   where $[v_i \in e]$ is the Iverson's invention of whether v_i is on the edge e. □

3. *Investment Strategy.* A company intends to invest 0.3 million yuan in 2021, with a proper combination of the following 3 projects:

- **Project 1:** Invest at the beginning of a year, and can receive a 20% profit of the investment in this project at the end of this year. Both the capital and profit can be invested at the beginning of next year;
- **Project 2:** Invest at the beginning of 2021, and can receive a 50% profit of the investment in this project at the end of 2022. The investment in this project cannot exceed 0.15 million dollars;
- **Project 3:** Invest at the beginning of 2022, and can receive a 40% profit of the investment in this project at the end of 2022. The investment in this project cannot exceed 0.1 million dollars.

Assume that the company will invest *all* its money at the beginning of a year. Please design a scheme of investment in 2021 and 2022 which maximizes the overall sum of capital and profit at the end of 2022.

(a) Formulate a linear programming with necessary explanations.

Solution. At the beginning of 2021, the ratios for Project 1 and Project 2 are α and β. At the end of 2021, the money that could be freely invested is

$$0.3\alpha \times 120\%$$

At the beginning of 2022, the investiment on Project 3 is γ. Then goal about the overall sum of captial and profit at the end of 2022 is

$$\max\left[(0.3\alpha \times 120\% - \gamma) \times 120\% + \gamma \times 140\% + 0.3\beta \times 150\%\right]$$

with the constraints of

$$\begin{aligned} s.t. \quad \alpha + \beta &= 1 \\ \gamma &\leq 0.36\alpha \\ 0.3\beta &\leq 0.15 \\ \gamma &\leq 0.1 \\ \alpha, \beta, \gamma &\geq 0 \end{aligned}$$

□

(b) Transform your LP into its standard form and slack form.

Solution. The standard form is

$$\begin{aligned} & \max(0.432\alpha + 0.45\beta + 0.2\gamma) \\ s.t. \quad -\alpha - \beta &\leq 1 \\ \alpha + \beta &\leq 1 \\ \gamma - 0.36\alpha &\leq 0 \\ \beta &\leq 0.5 \\ \gamma &\leq 0.1 \\ \alpha, \beta, \gamma &\geq 0 \end{aligned}$$

The slack form is

$$\begin{aligned} & \max(0.432\alpha + 0.45\beta + 0.2\gamma) \\ s.t. \quad & \alpha + \beta = 1 \\ & \gamma - 0.36\alpha + s_1 = 0 \\ & \beta + s_2 = 0.5 \\ & \gamma + s_3 = 0.1 \\ & \alpha, \beta, \gamma, s_1, s_2, s_3 \geq 0 \end{aligned}$$

□

(c) Transform your LP into its dual form.

Solution. The multipliers for the first 5 constraints in the standard form are $(y_1, y_2, y_3, y_4, y_5)$. Then the dual form is:

$$\begin{aligned} & \max(y_1 + y_2 + 0.5y_4 + 0.1y_5) \\ s.t. \quad & -y_1 + y_2 - 0.36y_3 \leq 0.432 \\ & -y_1 + y_2 + y_4 \leq 0.45 \\ & y_3 + y_5 \leq 0.2 \end{aligned}$$

□

(d) Use the simplex method to solve your LP.

Solution. Eliminate α by $\alpha = 1 - \beta$ in the slack form:

$$\begin{aligned} & \max(0.432 + 0.018\beta + 0.2\gamma) \\ s.t. \quad & \gamma + 0.36\beta + s_1 = 0.36 \\ & \beta + s_2 = 0.5 \\ & \gamma + s_3 = 0.1 \\ & \beta, \gamma, s_1, s_2, s_3 \geq 0 \end{aligned}$$

From the slack form, the basic solution is:

$$\bar{x} = (\bar{\beta}, \bar{\gamma}, \bar{s_1}, \bar{s_2}, \bar{s_3}) = (0, 0, 0.36, 0.5, 0.1)$$

Choose the nonbasic variable γ, $\gamma \uparrow$, $0 \leq s_1, s_3 \downarrow$,

i. $s_1 \leq 0$ when $\gamma \geq 0.36$

ii. $s_3 \leq 0$ when $\gamma \geq 0.1$

Then

$$\gamma = 0.1 - s_3$$

is the tightest constraint for γ. After pivoting,

$$\begin{aligned} & \max(0.432 + 0.018\beta + 0.02 - 0.2s_3) \\ s.t. \quad & 0.36\beta + s_1 - s_3 = 0.26 \\ & \beta + s_2 = 0.5 \\ & 0.1 - s_3 = \gamma \\ & \beta, \gamma, s_1, s_2, s_3 \geq 0 \end{aligned}$$

The current optimal solution is

$$\bar{x} = (\bar{\beta}, \bar{\gamma}, \bar{s_1}, \bar{s_2}, \bar{s_3}) = (0, 0.1, 0.36, 0.5, 0)$$

Then

$$\beta + s_2 = 0.5$$

is the tightest constraint for β. Then the optimal solution is

$$\bar{x} = (\bar{\beta}, \bar{\gamma}, \bar{s_1}, \bar{s_2}, \bar{s_3}) = (0.5, 0.1, 0.08, 0, 0)$$

As a result, the scheme for investment is as follows (M RMB):

	Project 1	**Project 2**	**Project 3**
2021	0.15	0.15	-
2022	0.08	-	0.1

□

4. *Factory Production.* An engineering factory makes seven products (PROD 1 to PROD 7) on the following machines: four grinders, two vertical drills, three horizontal drills, one borer and one planer. Each product yields a certain contribution to profit (in £/unit). These quantities (in £/unit) together with the unit production times (hours) required on each process are given below. A dash indicates that a product does not require a process.

	PROD 1	**PROD 2**	**PROD 3**	**PROD 4**	**PROD 5**	**PROD 6**	**PROD 7**
Contribution to profit	10	6	8	4	11	9	3
Grinding	0.5	0.7	-	-	0.3	0.2	0.5
Vertical drilling	0.1	0.2	-	0.3	-	0.6	-
Horizontal drilling	0.2	-	0.8	-	-	-	0.6
Boring	0.05	0.03	-	0.07	0.1	-	0.08
Planing	-	-	0.01	-	0.05	-	0.05

There are marketing limitations on each product in each month, given in the following table:

	PROD 1	**PROD 2**	**PROD 3**	**PROD 4**	**PROD 5**	**PROD 6**	**PROD 7**
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	300	600	0	0	500	400	100
April	200	300	400	500	200	0	100
May	0	100	500	100	1000	300	0
June	500	500	100	300	1100	500	60

It is possible to store up to 100 of each product at a time at a cost of £0.5 per unit per month (charged at the end of each month according to the amount held at that time). There are no stocks at present, but it is desired to have a stock of exactly 50 of each type of product at the end of June. The factory works six days a week with two shifts of 8h each day. It may be assumed that each month consists of only 24 working days. Each machine must be down for maintenance in one month of the six. No sequencing problems need to be considered.

When and what should the factory make in order to maximize the total net profit?

(a) Use *CPLEX Optimization Studio* to solve this problem. Describe your model in *Optimization Programming Language* (OPL). Remember to use a separate data file (.dat) rather than embedding the data into the model file (.mod).

```
int NbMonths = ...;
range Months = 1..NbMonths;
{string} Prod = ...;
{string} Process = ...;
```

```
5
6  int ProfitProd[Prod] = ...;
7  float ProcessProd[Process][Prod] = ...;
8  int MarketProd[Months][Prod] = ...;
9
10 float CostHold = ...;
11 int StartHold = ...;
12 int EndHold = ...;
13 int MaxHold = ...;
14
15 int HoursMonth = ...;
16 int NumProcess[Process] = ...;
17 int NumDown[Process] = ...;
18
19 // number of machines down in each month
20 dvar int+ MonthDown[Process][Months];
21 // number of products to be
22 // made, sell, hold in each month
23 dvar int+ MonthMake[Prod][Months];
24 dvar int+ MonthSell[Prod][Months];
25 // Sell at the month could not exceed the sum of
26 // the hold in the previous month and
27 // the make in the current month.
28 // So it is a nonnegative integer.
29 dvar int+ MonthHold[Prod][Months];
30
31 maximize
32   sum(p in Prod)
33     sum(m in Months)
34       (MonthSell[p][m] * ProfitProd[p]
35         - MonthHold[p][m] * CostHold);
36 subject to {
37   // The relation of Hold between months.
38   forall (p in Prod)
39     forall (m in Months)
40       ctHold:
41         MonthHold[p][m] ==
42           // No Stock at the beginning.
43           (m==1 ? StartHold : MonthHold[p][m-1])
44           + MonthMake[p][m] - MonthSell[p][m];
45
46   // The maximum hold for every month.
47   forall (p in Prod)
48       forall (m in Months)
49           ctMaxHold:
50             MonthHold[p][m] <= MaxHold;
51
52   // Exactly 50 stocks for every product at the end of June.
53   forall (p in Prod)
54       ctEndHold:
55         MonthHold[p][6] == EndHold;
56
57   // Every Month should be down once.
58   forall (q in Process)
59       ctDown:
60         sum(m in Months)
61             MonthDown[q][m] == NumDown[q];
62
63   // Time limitation
64   forall (q in Process)
```

```
    forall (m in Months)
      ctWork:
      sum (p in Prod)
        MonthMake[p][m] * ProcessProd[q][p] <=
          (NumProcess[q] - MonthDown[q][m]) * HoursMonth;

  // Marketing limitation
  forall (p in Prod)
    forall (m in Months)
      ctMarket:
        MonthSell[p][m] <= MarketProd[m][p];
}

float TotalSellingProfit =
  sum(p in Prod)
    sum(m in Months)
      MonthSell[p][m] * ProfitProd[p];

float TotalHoldingCost =
  sum(p in Prod)
    sum(m in Months)
      MonthHold[p][m] * CostHold;

float TotalNetProfit =
  TotalSellingProfit - TotalHoldingCost;

execute{
  writeln("Total␣Selling␣Profit=", TotalSellingProfit);
  writeln("Total␣Holding␣Cost=", TotalHoldingCost);
  writeln("Total␣Net␣Profit=", TotalNetProfit);
}
```

(b) Solve your model and give the following results.

i. For each machine:

A. the month for maintenance.

	Grinding	**Vertical drilling**	**Horizontal drilling**	**Boring**	**Planning**
January	0	0	1	0	0
February	0	1	0	0	0
March	0	0	0	0	0
April	4	1	2	1	1
May	0	0	0	0	0
June	0	0	0	0	0

ii. For each product:

A. The amount to make in each month.

	PROD 1	**PROD 2**	**PROD 3**	**PROD 4**	**PROD 5**	**PROD 6**	**PROD 7**
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	400	700	100	100	600	400	200
April	0	0	0	0	0	0	0
May	0	100	500	100	1000	300	0
June	550	550	150	350	1150	550	110

B. The amount to sell in each month.

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
January	500	1000	300	300	800	200	100
February	600	500	200	0	400	300	150
March	300	600	0	0	500	400	100
April	100	100	100	100	100	0	100
May	0	100	500	100	1000	300	0
June	500	500	100	300	1100	500	60

C. The amount to hold at the end of each month.

	PROD 1	PROD 2	PROD 3	PROD 4	PROD 5	PROD 6	PROD 7
January	0	0	0	0	0	0	0
February	0	0	0	0	0	0	0
March	100	100	100	100	100	0	100
April	0	0	0	0	0	0	0
May	0	0	0	0	0	0	0
June	50	50	50	50	50	50	50

iii. The total selling profit.

Total Selling Profit=109330

iv. The total holding cost.

Total Holding Cost=475

v. The total net profit (selling profit minus holding cost).

Total Net Profit=108855

Remark: You can choose to use the attached .dat file or write it yourself.

Appendix

A. FactoryPlanning.dat

```
NbMonths = 6;

Prod = {Prod1, Prod2, Prod3, Prod4, Prod5, Prod6, Prod7};
Process = {Grind, VDrill, HDrill, Bore, Plane};

// profitProd[j] is profit per unit for product j
ProfitProd = [10 6 8 4 11 9 3];

// processProd[i][j] gives hours of process i required by product j
ProcessProd = [[0.5 0.7 0.0 0.0 0.3 0.2 0.5 ]
[0.1 0.2 0.0 0.3 0.0 0.6 0.0 ]
[0.2 0.0 0.8 0.0 0.0 0.0 0.6 ]
[0.05 0.03 0.0 0.07 0.1 0.0 0.08]
[0.0 0.0 0.01 0.0 0.05 0.0 0.05]];

// marketProd[i][j] gives marketing limitation on product j for month i
MarketProd = [[500 1000 300 300 800 200 100]
[600 500 200 0 400 300 150]
[300 600 0 0 500 400 100]
[200 300 400 500 200 0 100]
[0 100 500 100 1000 300 0 ]
[500 500 100 300 1100 500 60 ]];

CostHold = 0.5;
StartHold = 0;
EndHold = 50;
MaxHold = 100;

// process capacity
HoursMonth = 384; // 2 eight hour shifts per day, 24 working days per month;

// number of each type of machine
NumProcess = [4 2 3 1 1];

// how many machines must be down over 6 month period
NumDown = [4 2 3 1 1];
```

Remark: You need to include your .cpp, .mod, .dat, .pdf and .tex files in your uploaded .zip file.