CS218M: Homework-5

Titas Chakraborty (170070019)
Siddharth Saha (170100025)
Parth Shettiwar (170070021)
Koustav Jana (170070051)

November 2020

1 DP formulation

$$Cost(k, i, e) = \min_{h} \min_{y}(a + b + c + d)$$

where

$$a = (e + h) * S + y * OTPrice$$

$$b = b_1 + b_2$$

$$b_1 = \left\{ \begin{array}{l} -h * Fcost, if\ h < 0 \\ h * Hcost, else \end{array} \right\}$$

$$b_2 = \left\{ \begin{array}{l} -Fcost * (e + h - E), if\ last\ month\ and\ e + h - E > 0 \\ Hcost * (E - e - h), if\ last\ month\ and\ e + h - E \leq 0 \\ 0, else \end{array} \right\}$$

$$c = \left\{ \begin{array}{l} 0, if\ last\ month \\ W * (i + (e + h) * C + y - D[k]), else \end{array} \right\}$$

$$d = Cost(k + 1, i + (e + h) * C + y - D[k], e + h)$$

Here, the range of h and y over which minimum are calculated are:

$$-e \leq h \leq max(int(\frac{TotalDemand}{C}) + 1, E + 1)$$

$$0 \leq y \leq OTC * (e + h)$$

The table size ranges are:

$$0 \leq k \leq M$$

$$0 \leq i < max(TotalDemand, C)$$

$$0 \leq e < max(int(\frac{TotalDemand}{C}) + 1, E + 1)$$

The overall Time complexity of the formulation is:

$$O(\frac{M * (\sum_{0 \leq k < M} D[k])^4 * OTC}{C^3})$$

- In the recurrence, the $(e + h) * S$ term represents the salary paid during the k^{th} month. Since the next number of employees will be $e + h$ and one has to pay one months salary.
- The $y \times OTPrice$ represents the cost of carpets produced in overtime as y is the number of overtime carpets.

- $W * (i + (e + h) * C + y - D[k])$ represents the warehouse cost since $i + (e + h) * C + y - D[k]$ represents the number of carpets to be stored. i carpets are present initially, $(e + h) * C + y$ are produced and $D[k]$ carpets are sold in month k.
- All the terms represent the cost in month k. Now for the remaining months, we have $i + (e + h) * C + y - D[k]$ carpets initially and $e + h$ employees already hired. Hence the minimum cost for the remaining months will be $Cost(k + 1, i + (e + h) * C + y - D[k], e + h)$
- Iterating over h and e gives the minimum cost $Cost(k, i, e)$.
- The range of y is 0 to $OTC * (e + h)$ since $e + h$ employees can produce a maximum of $OTC * (e + h)$ carpets in overtime.
- For the range of h, we notice that if e employees are present, at most e can be fired. Also the maximum number of employees can be feasibly $max(\frac{int(TotalDemand)}{C} + 1, E + 1)$ as these employees can satiate the entire demand in one month. Also, E employees may be present initially.
- If $i + (e + hr)C + y - D[k] < 0$, the case is not considered since that case implies that there are not enough carpets to supply the demand.
- If $e + h >= max(int(totalprod/C) + 1, E + 1)$, the same thing is done since there would be no reasonable case where so many employees would be hired to finish the total demand in one month. Such a configuration cannot be optimal.
- h represents the number of people hired and fired at the beginning of a particular month. On the other hand, e represents the number of employees at the beginning of a particular month.
- This means that the hiring-firing cost is $Hcost \times h$ if h is positive and $-Fcost \times h$ if h is negative. However, for the last month, the final employees have to be E, so to make it equal to that, $e + h - E$ employees have to be hired or fired. So that adds an additional cost on top of the regular hiring firing cost.
- For the table size, the number of months is fixed. For the initial number of carpets, it cannot be more that $max(TotalDemand, C)$ in an optimal sequence. Also the maximum number of employees initially can be feasibly $max(\frac{int(TotalDemand)}{C} + 1, E + 1)$ as these employees can satiate the entire demand in one month.

2 Random test case Generation

The script `generate_random_testcases.py` was written for generating test cases by randomly choosing a value for all the involved parameters from the ranges tabulated below.

Parameter	Range
D[month]	[1,20]
E	[1,10]
Hcost,Fcost	[20,50]
S	[100,400]
OTC	[1,4]
OTPrice	[30,200]
W	[2,10]

Table 1: Ranges chosen

The two remaining parameters M(No of months) and C(No of carpets per month by an employee) have been varied to observe the impact on the relative performances of LP(Python) and DP(C++) based method.

3 Performance Comparison and Observations

For the sake of performance comparison we consider batches of 20 test cases with pre-decided ranges for the parameters M and C from which they are randomly chosen. We choose M and C as these 2 parameters are the ones which have the maximum effect on the table size in DP. The D[month] values are chosen at random and they also affect the table size, but choosing 20 test cases for every run keeps things fair. We report the effect of the chosen range for the parameters M and C on the execution times. In order to run heavier tests, we have multi-threaded the evaluation on the randomly generated test cases (`n = 40 threads`) in the `evaluate_testcases.py` script. We have ensured that all `linear_programming.py` threads are completed before running the `dynamic_programming.cpp` script and that the times are recorded independently.
To begin with we fix the range of C to be [1,5] and see the variation of execution times of both DP and LP, for the ranges [1,5],[6,10],[11,15],[16,20],[21,25] and [25,30] for M. Given below is the obtained plot for the same. The upper limit of a range has been used to represent it on the x-axis and this holds for all the subsequent plots as well.

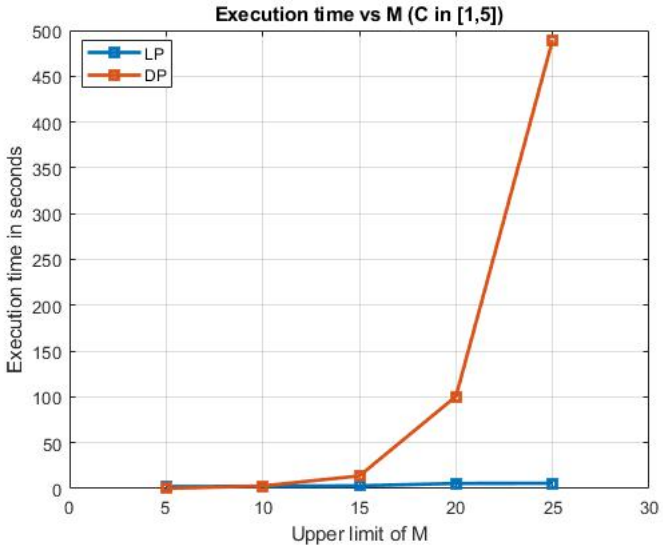

Figure 1: C-[1,5]

We repeat the above for C in [5,10] and get the following plot.

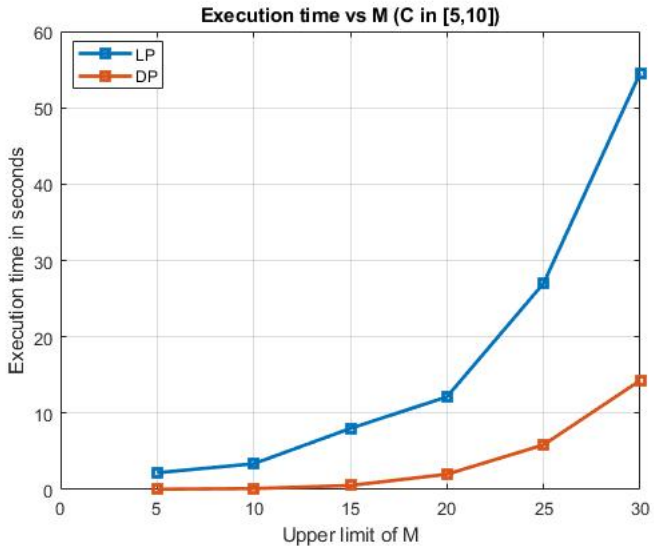

Figure 2: C-[5,10]

Next we fix the range of M to be [1,5] and see the variation of execution times of both DP and LP, for the ranges [1,2],[3,4],[5,6],[7,8] and [9,10] for C. We get the following for the same

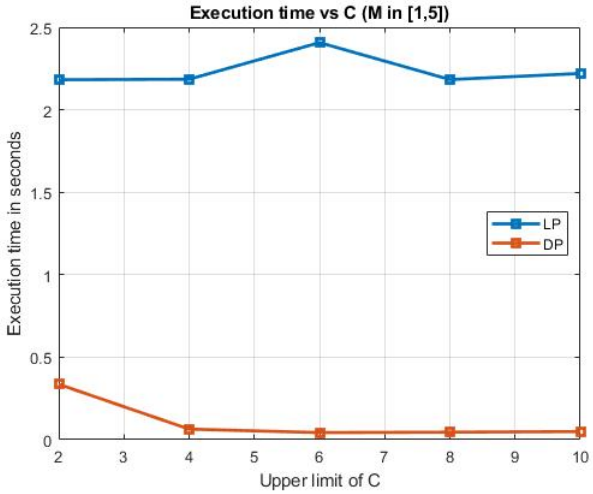

Figure 3: M-[1,5]

Repeating the above for M in [11,15] and [21,25] we get the plots below.

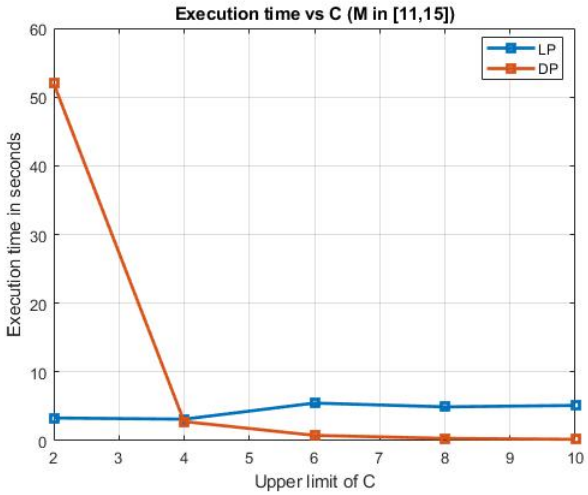

Figure 4: M-[11,15]

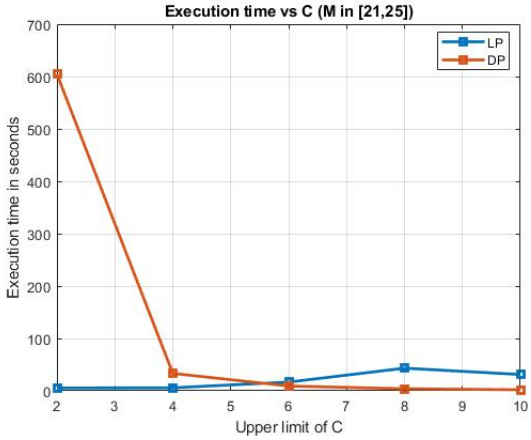

Figure 5: M-[21,25]

Some of the observations we make are as follows:

- Figure 1 and 2 suggest that increasing M increases the execution time as well. Also when C is small, LP outperforms DP by a huge margin when M is large whereas for larger C values its the DP based program which is faster for all values of M.
- DP performs better than LP for small M values irrespective of the value of C as suggested by figure 3
- However when M is large like in figure 4 and 5, for smaller C values it is LP which performs better, on the other hand as C is increased DP becomes faster and dominates

4 Conclusions

- Thus putting all the above observations together we can conclude that DP is favourable over LP for smaller M values and larger C values.
- The table size in DP is proportional to $\frac{M}{C^2}$ and the overall time complexity is proportional to $\frac{M}{C^3}$, and the DP performance being highly dependent on table size, it is expected that DP performs better for small M and large C but performs very poorly when M is large and C is small.
- Also LP performance is not quite affected by changing C but increasing M does slow it down because of the demand array size increasing, thus increasing the solution space complexity. But it does not slow down as much as in the DP case.
- Hence in a nutshell for the production planning problem, LP should be preferred for larger problem sizes whereas DP is the clear winner for smaller problem sizes, where the effect of C++ being faster than Python takes over