Assignment 2: Production Scheduling Problem
Transportation and Scheduling (FEM21043)

Wessel Boosman (589273)

January 21, 2022

Question 1

(a) *Try to solve each instance using the ILP formulation with a time limit of 5 minutes and allowing the solver to use at most 1 thread.*

The ILP formulation of the problem formulated in slide 5 of the second lecture of the course is implemented in Java. The problem is solved with Cplex version 20.1.0.0. Table 1 summarizes the performance of the ILP solver for all of the five instances. The ILP solver was able solve 4 of the 5 instance within the time limit of 5 minutes to optimality.

Table 1: Performance of Cplex on solving the production scheduling problem (ILP) for multiple instances.

ILP Performance / Instance	Optimality	Run Time [*sec*]	Lower Bound	Upper Bound
1	Yes	0.13	1811	1811
2	Yes	0.19	2728	2728
3	Yes	1.97	47,081	47,081
4	Yes	295.33	94,379	94,379
5	No	300.00	132,182.7	133,506

(b) *Develop a heuristic for the PSP. Describe the heuristic and report the solution values and the computation times for all instances.*

Aiming to keep the fixed costs corresponding to the production as low as possible, I built my heuristic as described in Algorithm 1. The idea is to produce as much as possible in a certain period if the inventory is not sufficient to cover the demand. Except for the case when the production capacity is so high that exceeds the cumulative demand over the remaining periods. In that case you produce just enough such that the inventory at the end of the final period is zero. This algorithm is extremely fast for all instances as can be seen in Table

2. Furthermore, Table 2 presents the objective value for each instance corresponding to the heuristic.

Algorithm 1 Algorithm describing the heuristic for the production scheduling problem

```
inventory ← 0
for i ← 1, All periods do
    if inventory - demand in period i > 0 then
        production in period i ← 0
        inventory ← inventory - demand in period i
    else
        max ← Maximum of the production capacity and the cumulative demand over the
                remaining periods minus the current inventory.
        production in period i ← max
        inventory ← inventory - demand in period i + max
    end if
end for
```

Table 2: Table summarizing the performance of the heuristic explained in algorithm 1 for all instances

Heuristic Performance / Instance	Objective	Run Time [*sec*]
1	2011	< 0.01
2	2933	< 0.01
3	52,876	< 0.01
4	102,905	< 0.01
5	144,248	< 0.01

(c) *Implement the dynamic programming algorithm (with a quadratic running time). For each instance, select a factor F such that the computation time is below 5 seconds. Report the lower bound, upper bound, and computation time that are obtained for this factor. Iteratively halve the factor F and run the dynamic program, as long as the computation time is below 5 minutes.*

I implemented the dynamic programming algorithm as presented in slides 7 to 9 from the lecture slides of the second lecture of this course. As can be seen from this formulation DP variables of the next period only depend on the DP variables of the previous period and in order to get the objective value you only need the DP variables of the last period. In other words, most of the DP variables are worthless and are not worth it to keep track of. Realizing this, I stored the DP variables in a 2-dimensional array with only two rows. In the initialization step I store all the DP variables corresponding to the first period in the first row. The for the second period, I use the DP variables from the first row and store them in the second row. The DP variables corresponding to the third period I store in the first row again, so replacing the DP variables from the first period, etc. This saves memory and speeds up my

algorithm by approximately 15%. This is a memory efficient way to get the objective value for the production scheduling problem using the quadratic DP algorithm. On the other hand, in order to get a feasible production plan corresponding to this objective value we cant discard all information from previous periods. We need a full backtracking scheme of how much is spent in which period. I made the backtracking scheme in the following way. For each period a DP variable is created for a certain budget. The value of this DP variable corresponds to a certain distribution of the budget, i.e. a certain amount of the budget is spent in the current period and the remainder is spent in the periods previous to the this one. This is exactly the information I keep track of; for each period, for each DP variable, I store the corresponding distribution of the budget. Or to be specific, I actually only store the amount of money spent in previous periods with respect to the current period (So just to be clear, this corresponds to the variable notated with an 'a' in slide 8 of the lecture slides).

So in the end I have an objective value (total expenditure for the PSP), and a backtracking scheme. Now we can check how much is spent in each period. First, we look for the budget for which the corresponding DP variable in the last period is larger or equal to zero. From the backtracking scheme we now how this budget has to be distributed. Obviously, we now know exactly how much is spent in the last period and how much is spent in the previous periods combined. We go one step backwards to second last period. We know the combined expenditure for all periods up to this period and can again look in the backtracking scheme how to distribute the money the best, etc. We continue this procedure until the first period, after which we exactly know how much money is spent in each period.

Now we can construct a feasible production scheme using the information of how much is spent each period. We start form the last period and fix the inventory at zero (zero-inventory property). Then we check whether the total amount of money spent in the last period is enough to produce the demand (and cover the inventory costs, but these are obviously zero in the last period) in that period. If this is the case, we produce exactly the demand and fix the inventory of the previous period to zero. If this is not the case, we can not produce enough to cover the demand. In this case we produce as much as the budget allows us to (but no more than capacity obviously). Then I subtract this production quantity from the demand and use the result to fix the inventory at the previous period. Now we take a step back to the second to last period and we know what the inventory must be as well as how much money is spent in this period. From this we can again calculate the production quantity and fix the inventory for the previous period, etc. In the end this procedure results in a feasible production scheme. For this production scheme we can easily calculate the actual costs, and use this as an upper bound to the solution. This is exactly why Table 3 to Table 7 show the objective value (the budget for which the corresponding DP variable in the last period is larger or equal to zero) as well as an upper bound (the actual costs corresponding to a feasible solution based on the distribution of the budget).

Table 3 to Table 7 summarize the main findings of this algorithm for each of the instances. The

fifth instance shows some strange run time behaviour which is far from quadratic, especially for large 'F'. I have no idea why this is the case and why this is only happening at the last instance. The other instances clearly show longer running times for smaller values of 'F', however not always in a quadratic behaviour as expected. This can be seen, for example, in Table 5 where the running time between $F = 32$ and $F = 16$ only slightly differs. On the other hand, Table 6 shows running times which become roughly four times large when F is halved.

Table 3: Instance 1

F	1
Run Time [sec]	4.05
Objective	1811
Actual Cost (*Upper Bound*)	1811
Lower Bound	1801

Table 4: Instance 2

F	2	1
Run Time [sec]	4.81	10.90
Objective	2736	2728
Actual Cost (*Upper Bound*)	2728	2728
Lower Bound	2696	2708

Table 5: Instance 3

F	256	128	64	32	16
Run Time [sec]	3.06	8.03	27.39	94.82	128.86
Objective	51,712	49,408	48,256	47,712	47,392
Actual Cost (*Upper Bound*)	48,355	47,174	47,174	47,174	47,081
Lower Bound	38,912	43,008	45,056	46,112	46,592

Table 6: Instance 4

F	512	256	128	64	32
Run Time [sec]	4.55	10.10	28.79	91.74	388.12
Objective	108,544	103,936	98,560	96,786	95,584
Actual Cost (*Upper Bound*)	96,510	94,587	94,587	94,554	94,477
Lower Bound	57,344	78,336	85,760	90,368	92,384

Table 7: Instance 5

F	131,072	...	1024	512	256	128
Run Time [sec]	7.04	...	27.27	52.29	32.27	334.45
Objective	10,485,760	...	212,992	156,672	149,248	141,568
Actual Cost (*Upper Bound*)	180,264	...	148,093	137,272	136,586	134,008
Lower Bound	0	...	59,392	79,872	110,848	122,368