Scheduling Problem : Uncapacitated Lot Sizing Problem

Hajar Douaik

December 20, 2025

1 Introduction

In this project, I used Python as the programming language and followed the formulations introduced during the course to implement both the integer programming models and the dynamic programming approach. Each question was implemented in a separate file to improve readability, visibility, and portability. For the final questions, additional files were created to test different instances.

2 Reading data

The function **read** opens a text file and reads all its lines, removing any empty lines and unnecessary whitespace. It then parses the content in a fixed order: the first line is converted into an integer, and the following lines are split into space-separated integers and stored as lists. Finally, the function returns these values as a tuple, providing structured numeric data that can be directly used by the rest of the program.

3 Integer Linear programming formulations

These two models are solved using IBM ILOG **CPLEX** through the **PuLP** Python interface.

3.1 First formulation

This formulation models the Problem using a mixed-integer linear programming approach with a **Big-M constraint**. The objective is to determine when to produce and in what quantities in order to satisfy all demands over the planning horizon at minimum total cost.
Inputs: time periods n, demand $d = (d_1, .., d_n)$, fixed costs $f = (f_1, .., f_n)$, production costs $p = (p_1, .., p_n)$, storage costs $h = (h_1, .., h_n)$, inital inventory s_0.
Outputs: Optimal total cost, production quantities x_t, setup decisions y_t, and inventory levels s_t.
Steps of the algorithm:

- Uses LpProblem, LpVariable to define a minimization model, decision variables.
- Builds the objective function using lpSum to aggregate setup, production, and holding costs.
- Computes a Big-M value to link production and decisions, add inventory balance and linking constraints to the model.
- Solves the model with CPLEX_CMD and extracts results using value.

3.2 Second formulation

This second ILP models uses assignment variables: variable zi,k decides which production period i is used to satisfy demand of period k. It follows the same steps as the last formulation with replacing the variables xt and st with zi,k.
Inputs:

- time periods n, demand $d = (d_1, \ldots, d_n)$, fixed costs $f = (f_1, \ldots, f_n)$, production costs $p = (p_1, \ldots, p_n)$, storage costs $h = (h_1, \ldots, h_n)$

Output:

- Optimal total cost, optimal setup decisions y_i, optimal assignment quantities and $z_{i,k}$

steps of the algorithm:

- Computes holding costs through a dedicated Python function **holding_cost** that aggregates period-wise holding cost values.
- Declares decision variables with `LpVariable`, using binary variables for setup decisions and continuous variables stored in a dictionary for assignment quantities.
- Adds linear constraints directly to the model to enforce demand satisfaction and logical consistency between variables.

4 Dynamic programming algorithm

4.1 main function

This program is used to read an instance from a .dat file provided as a command-line argument, and then display the data contained in that instance in order to verify that it has been loaded correctly. The statement if __name__ == "__main__" is a standard Python construct that allows you to control when the main code of a file is executed.

4.2 dynamic progamming function

The function computes an optimal production plan over a finite planning horizon by deciding in which periods production should occur and in what quantities. It ensures that all demands are satisfied while minimizing the total cost, which includes fixed setup costs, variable production costs, and inventory holding costs.
Inputs: time periods n, demand $d = (d_1, .., d_n)$, fixed costs $f = (f_1, .., f_n)$, production costs $p = (p_1, .., p_n)$, storage costs $h = (h_1, .., h_n)$, initial inventory s_0.
Outputs: Optimal total cost, setup decisions y_t, production quantities x_t, and inventory levels s_t.
Steps of the algorithm:

- Preprocesses cost parameters by incorporating cumulative holding costs into production costs.
- Initializes dynamic programming arrays to store the optimal cost and regeneration decisions for each period.
- Uses a recurrence relation to compute the minimum cost up to each period by testing all possible regeneration points.
- Stores the optimal regeneration period decisions and reconstructs the setup policy.
- Computes the final objective value using the original cost parameters.

4.3 Results

The theoretical complexity of the three approaches is summarized in Table 1. While both ILP formulations are NP-hard mixed-integer linear programs, the dynamic programming (DP) approach has a polynomial complexity.

Formulation	Complexity
ILP1	NP-hard (MILP); $\Theta(n)$
ILP2	NP-hard (MILP); $\Theta(n^2)$
DP	$\mathcal{O}(n^2)$ time, $\mathcal{O}(n)$ memory

Table 1: Theoretical complexity of the different formulations.

The two ILP formulations were solved using CPLEX. For small instances, both ILP1 and ILP2 return the same optimal objective value and the same regeneration slots. Table 2 reports the results and execution times obtained with CPLEX.
ILP2 introduces $\Theta(n^2)$ continuous assignment variables, which significantly increases the model size. As a result, for medium and large instances, the model exceeds the limitations of the CPLEX Community Edition. The absence of results for ILP2 on larger instances is therefore due to solver limitations and not to infeasibility of the problem.

Formulation	Instance	Objective value	Regeneration slots	Execution time
ILP1	instance_0005	86.00	1, 4	0.409s
ILP2	instance_0005	86.00	1, 4	0.076s
ILP1	instance_0100	2527.00	–	0.0819s
ILP2	instance_0100	–	–	not solved

Table 2: CPLEX results for ILP formulations.

The dynamic programming approach was tested on several instances of increasing size. The results are reported in Table 3. The DP algorithm always returns the same optimal objective values as the ILP formulations on small instances, while remaining computationally efficient for larger instances.

Instance	n	Objective value	Execution time
instance_0005.dat	5	86.00	0.000000s
instance_0100.dat	100	2527.00	0.008097s
instance_0500.dat	500	13204.00	0.156438s
instance_1000.dat	1000	24126.00	0.621197s
instance_2000.dat	5000	126070.00	17.877289s

Table 3: Dynamic programming results.

Finally, Table 4 compares the total runtime of the three approaches on a small instance. All methods return the same optimal objective value, but the DP approach is significantly faster.

Formulation	Instance	Objective value	Elapsed time
ILP1	instance_0005	86.00	0.11s
ILP2	instance_0005	86.00	0.09s
DP	instance_0005	86.00	0.00s

Table 4: Global runtime comparison for instance_0005.

Overall, these results show that while ILP formulations are suitable for small instances and provide a clear mathematical modeling framework, the dynamic programming approach is more scalable and better suited for large problem sizes, in accordance with its polynomial time complexity.