KTH Royal Institute of Technology

SF2863 Systems Engineering

Home Assignment 2

Birgir Steinn Hermannsson (bshe@kth.se)

Daniel Landberg (dlandb@kth.se)

December 15, 2020

Marginal Allocation

(1) There are four different requirements that must be met in order to apply the marginal allocation algorithm:

- The objective functions are separable.
- The objective functions are integer-convex.
- One objective functions is increasing.
- One objective functions is decreasing.

The two objective functions for this problem are defined as:

$$f = \mathbf{c}^\intercal \mathbf{s}$$
$$g = EBO(\mathbf{s})$$

where,

$$\begin{aligned} X_i &= \text{Number of } LRU_i \text{ in workshop} \\ s_i &= \text{Number of spare part units} \\ EBO &= \text{Expected number of back-orders for every LRU} \\ c_i &= \text{Costs for spare units} \\ \mathbf{c}^\intercal \mathbf{s} &= \text{Increasing objective function} \\ EBO(\mathbf{s}) &= \text{Decreasing objective function} \end{aligned}$$

The EBO can be written as the sum of the expected number of back-orders due to shortage of each LRU in the following manner

$$EBO(\mathbf{s}) = \sum_{i=1}^{9} EBO_i(s_i)$$

The objective function g is thus separable. The total cost due to purchase of LRUs, $\mathbf{c}^\intercal \mathbf{s}$, can similarly be written as the sum of the individual costs for each LRU

$$\mathbf{c}^\intercal \mathbf{s} = \sum_{i=1}^{9} c_i s_i$$

So f is separable as well. It is apparent from the definitions above that with each increasing s_i, g decreases and f increases, so all four requirements are met and the marginal allocation method is applicable to solve this problem.

(2) The marginal allocation algorithm was implemented as described in "On spare parts optimization" by Krister Svanberg and was initiated with no spares. The development of the EBO when adding each new spare LRU unit was then calculated.

(3) The marginal allocation algorithm returned the efficient curve for the problem as presented in Figure 1. The cost, EBO, purchase strategy and the marginal decrease in EBO for each new unit of cost spent on the LRUs is shown in Table 1.

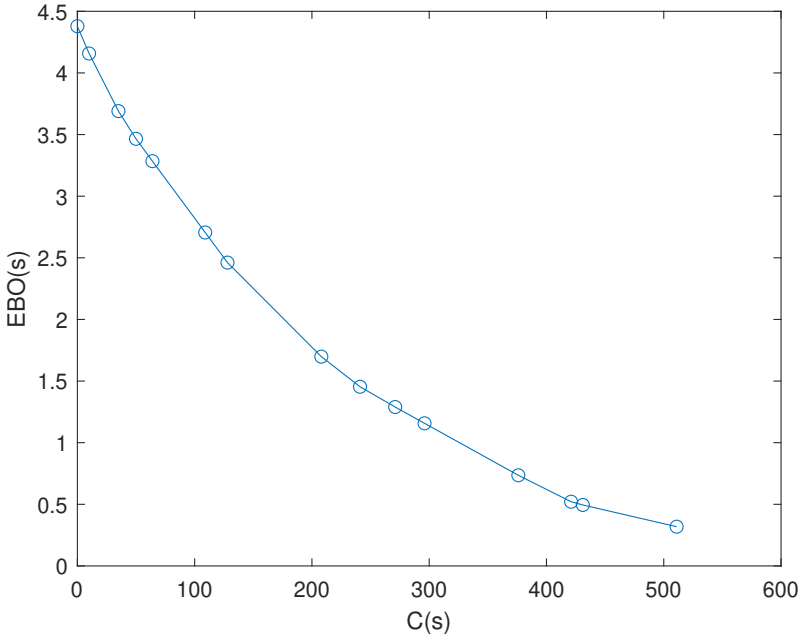

Figure 1: Efficient curve according to marginal allocation

Table 1: Efficient solutions

Cost	EBO	ΔEBO	P1	P2	P3	P4	P5	P6	P7	P8	P9
0	4.3790	0	0	0	0	0	0	0	0	0	0
10	4.1578	0.2212	0	0	0	0	1	0	0	0	0
35	3.6904	0.4674	0	0	1	0	1	0	0	0	0
50	3.4653	0.2251	0	0	1	1	1	0	0	0	0
64	3.2840	0.1813	1	0	1	1	1	0	0	0	0
109	2.7055	0.5785	1	1	1	1	1	0	0	0	0
128	2.4613	0.2442	1	1	1	1	1	1	0	0	0
208	1.6982	0.7631	1	1	1	1	1	1	1	0	0
241	1.4540	0.2442	1	1	1	1	1	1	1	1	0
271	1.2893	0.1647	1	1	1	1	1	1	1	1	1
296	1.1574	0.1319	1	1	2	1	1	1	1	1	1
376	0.7355	0.4219	1	1	2	1	1	1	2	1	1
421	0.5211	0.2144	1	1	2	1	1	2	2	1	1
431	0.4946	0.0265	1	1	2	1	2	2	2	1	1
511	0.3184	0.1762	1	1	2	1	2	2	3	1	1

Dynamic Programming

(4) In this case the problem will be solved using a dynamic programming approach instead. The notation for the problem setup is as follows

- States, $\mathbf{s}_n$: The available inventory of parts at the start of each stage.
- Stages, n: Possible available budgets, $n = 0, 1, \ldots, 500$.
- Decisions, x_n: Which part (P1, P2, ..., P9) should be purchased at each stage n and added to the inventory.
- State-update EQ: $s_{n+1} = s_n + x_n$
- Value functions: $f_n^*(s_n) = \min\{f_n(s_n, x_n)\} = \min\{EBO(s_n, x_n)\}$
- Recursive relation: $f_n^*(s_n) = \min\{f_{n+1}(s_n, x_n) + \sum_{k=1}^{Budget/c(i)} G(s_k, x_k)\}$

The recursive relation are returning the optimal solution using $EBO(k{*}\#parts){+}EBO(current\ budget{-}k * \#parts)$

The fundamental process of the dynamic programming algorithm is as follows

Algorithm 1 Dynamic Programming

```
for Every budget do                                          ▷ Setup before recursion
    for Every part do
        Create matrix with how many parts we can purchase using budget
    end for
end for
for Every row in matrix do
    Calculate EBO for part #9                    ▷ Our start value, needed to start recursion
end for

for Every part in order [8, 7, 6, 5, 4, 3, 2, 1] do                          ▷ Recursion
    matrix = AllVals(Data)
end for

function AllVals(Data)
    for Every row in matrix (Every budget) do          ▷ Find optimal for every sub problem
        OptimalData = FindOptimal(Data)                                  ▷ (every budget)
    end for
    return OptimalDataMatrix
end function

function FindOptimal(Data)          ▷ Find the setup with lowest EBO using previous optimal
    while Money left do
        Array(i) = Part(i) + Matrix(budget-Part(i))        ▷ Matrix(idx) gives previous optimal
    return min(Array)                                    ▷ Returning the array with lowest EBO

```

(5) The resultant efficient curve generated by the dynamic programming method can be seen in Figure 2, and the corresponding EBO values and strategies for some selected budgets are presented in Table 2.

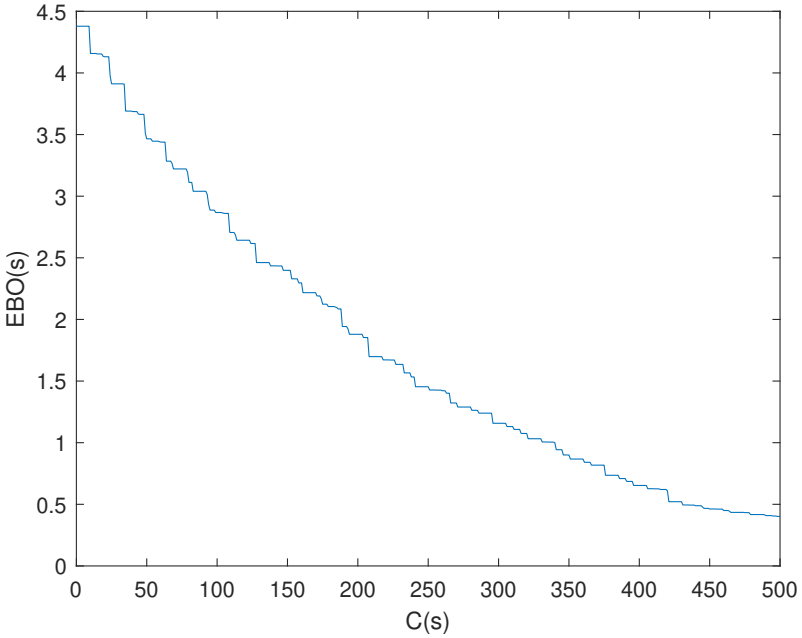

Figure 2: Lowest EBO for every budget

Table 2: Selected results from dynamic programming

Budget	EBO	P1	P2	P3	P4	P5	P6	P7	P8	P9
0	4.379	0	0	0	0	0	0	0	0	0
100	2.867	0	1	1	0	1	1	0	0	0
150	2.398	0	1	1	1	1	1	0	1	0
350	0.900	1	1	2	1	1	1	2	1	0
500	0.402	1	2	2	2	2	2	2	2	1

(6) Comparing the efficient curves obtained from the previous two methods, as seen in Figure 3, there is not substantial disparity. The results are somewhat different but it is clear that the few data-points the marginal allocation algorithm found are all common data-points with the dynamic programming algorithm. Note that at the far-right end of the graph the marginal allocation algorithm exceeds the budget. The reason is because the marginal allocation algorithm is breaking as soon as the budget is greater than the cost but not the other way around. The dynamic programming solution is breaking as soon as it hits the budget.

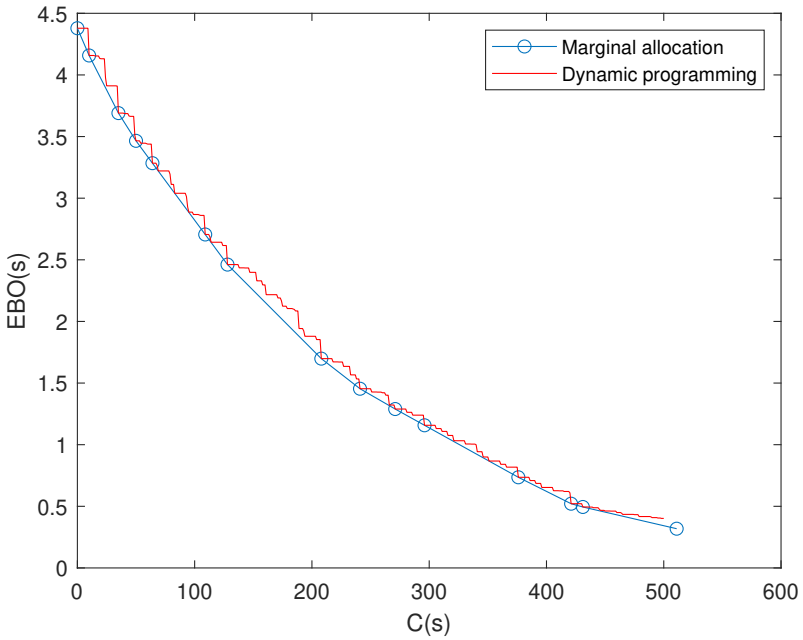

Figure 3: Dynamic programming and Marginal allocation solutions

Using the given budget, the marginal allocation algorithm's best solution which didn't exceed the budget was using 431 cost units. With this solution the EBO was 0.4946. If we would allow a solution a little over the given budget, the best solution would have been 511 with an EBO of 0.318.

Using the dynamic programming approach and 500 money, the best solution had an EBO of 0.402. This solution is under budget and better than the marginal allocation algorithms solution (not over budget). If we would have a little extra money we could have added 11 money to the budget and reduced the EBO with almost 25%.