Spare parts optimization

This computer exercise report submitted in fulfillment of the requirements for the course

in

SF2863 Systems Engineering

by

SF2863 2021 HA2 55

Davron Patkhullaev	Magnus Axén
19980830-1916	19980915-7119
davronp@kth.se	maaxen@kth.se

KTH ROYAL INSTITUTE OF TECHNOLOGY

Contents

List of Figures

List of Tables

Problem formulation

In this homework we are asked to solve a "Model 2" problem with both marginal allocation and dynamic programming. The formulation is as follows; We consider air crafts equipped with 9 different LRU's, LRU_j and that there is one basis to which we have random arrival of airplanes with a LRU_j unit that needs repairing. The malfunctioning LRU_j arrive with intensity,

$$\lambda = [50, 40, 45, 50, 25, 48, 60, 35, 15]/1000.$$

When the airplanes with LRU_j malfunctioning it is replaced with a new functioning LRU_j from the local inventory, if there are spar parts available. Else a queue will form and a back order is issued and the airplane remains grounded till it gets the LRU_j. The malfunctioning LRU_j will arrive directly to the basis, and the time to repair the LRU_j is assumed to be on average, in hours,

$$T_1 = 6, T_2 = 8, T_3 = 14, T_4 = 25, T_5 = 12, T_6 = 18, T_7 = 33, T_8 = 8, T_9 = 12$$

And each purchase of the spare part LRU_j will be described by,

$$c_1 = 12, c_2 = 14, c_3 = 21, c_4 = 20, c_5 = 11, c_6 = 45, c_7 = 75, c_8 = 30, c_9 = 22.$$

Moreover at the start of the simulation there are no spares to any LRU.

Marginal Allocation

1

We wish to define f_j and g_j, (where j is an integer $j = 1, \cdots, 9$), as integer-convex functions which have the property,

$$\Delta f(x+1) \geq \Delta f(x),$$

with f_j being strictly decreasing and g_j strictly increasing, this is to be able to utilize the marginal allocation algorithm. Let f_j be the EBO and g_j the cost, more formally we have,

$$f_j = \mathbb{E}[BO(s_j)], \quad g_j = C(s_j).$$

Observe that there is no initial spare parts, which implies g_j must be a linear function starting at 0, $g_j = c_j \cdot s_j$. To make sure that the functions are integer-convex and strictly increasing respectively decreasing, we consider the following proofs. Since we can write f_j as $f_j(s_j+1) = \mathbb{E}[BO(s_j+1)] = \mathbb{E}[BO(s_j)] - R(s_j)$ where $R(s_j) = \sum_{k=s_j+1}^{\infty} p(k)$, with $p(k)$ being the probabilities of our Poisson distribution we can conclude further that,

$$\Delta f_j(s_j+1) \geq \Delta f(s_j),$$

is equivalent to

$$f_j(s_j+2) - f_j(s_j+1) \geq f_j(s_j+1) - f_j(s_j).$$

Thus by the earlier rewriting, we have

$$\mathbb{E}[BO(s_j+2)] - \mathbb{E}[BO(s_j+1)] \geq \mathbb{E}[BO(s_j+1)] - \mathbb{E}[BO(s_j)]$$

$$-R(s_j+1) \geq -R(s_j) \Leftrightarrow R(s_j+1) \leq R(s_j)$$

Which is true by the definition of $R(s_j)$ since all its terms are probabilities and $p(k) \geq 0$ and as the right hand side contains one more term than the left hand side.

By the definition of expected backorders, it is going to decrease with s increasing, seeing as the more spare parts we have the fewer backorders we can expect. So we are done in showing that f is integer-convex and strictly decreasing.

Now what remains to show is that g is integer-convex and strictly increasing,

$$\Delta g_j(s_j+1) \geq \Delta g_j(s_j),$$

which again by the definition is,

$$\Delta g_j(s_j+2) - \Delta g_j(s_j+1) \geq \Delta g_j(s_j+1) - \Delta g_j(s_j).$$

Rewriting $g_j = c_j \cdot s_j$,

$$c_j \cdot (s_j+2) - c_j \cdot (s_j+1) \geq c_j \cdot (s_j+1) - c_j \cdot (s_j)$$

and simplifying,

$$c_j \geq c_j$$

which holds true for all positive costs, thus it must indeed be integer convex. And so finally to show it is strictly increasing we note first that seeing as it is a linear increasing function with $c_j \geq 0$, then we have that it is strictly increasing via,

$$\Delta^2 g_j(s_j) = \Delta c_j = 0 \geq 0.$$

Then we have that both f, g satisfy the criteria for using the theorem for marginal allocation.

2

To utilise the marginal allocation algorithm we start by generating a table with the various fractions of $-\frac{R_j(l)}{c_j}$ (where j denotes the column of the table and the corresponding spare part, and l the row index), moreover $R_j(i) = 1 - \sum_{k=0}^{i} \frac{(\lambda_j T_j)^k}{k!} e^{-\lambda_j T_j}$.

Secondly we select the largest uncancelled quotient in our table with some index j and cancel that quotient but add that spare part j, here we have moved to an efficient point.

Lastly, we adjust our $\mathbb{E}[BO(s)]$, which initially is $\mathbb{E}[BO(s)]^{(0)} = \sum_{i=1}^{9} \lambda_i T_i$, but after each iteration is, $\mathbb{E}[BO(s)]^{(k)} = \mathbb{E}[BO(s)]^{(k-1)} - R_l(s_l^{(k-1)})$, here, $R_l(s_l^{(k-1)})$, is selected by the largest fraction left in the column table from before. After each iteration a cost of that respective spare part is added to a total cost variable. This process is repeated till the total costs exceeds 500, then the iteration is stopped and the last purchase is not be added.

3

Table 1: Marginal allocation - $\frac{-\Delta f_j(s_j)}{\Delta g_j(s_j)}$

	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$	$j=8$	$j=9$
$s_j=0$	0.0216	0.0196	0.0223	0.0357	0.0236	0.0129	0.0115	0.0081	0.0075
$s_j=1$	0.0031	0.0030	0.0063	0.0178	0.0034	0.0048	0.0078	0.0011	0.0007
$s_j=2$	0.0003	0.0003	0.0012	0.0066	0.0003	0.0013	0.0042	0.0001	0.0000
$s_j=3$	0.0000	0.0000	0.0002	0.0019	0.0000	0.0003	0.0019	0.0000	0.0000

Moreover, why only four rows are included is since this was the required rows needed to obtain the listed solution below, obviously one could generate rows indefinitely but the fourth row was the last needed to be able to obtain the efficient curve. Listed below is the given plot of the efficient curve and efficient points. Moreover we decided to include the coming purchase after the 500 mark too, for illustrative purposes.

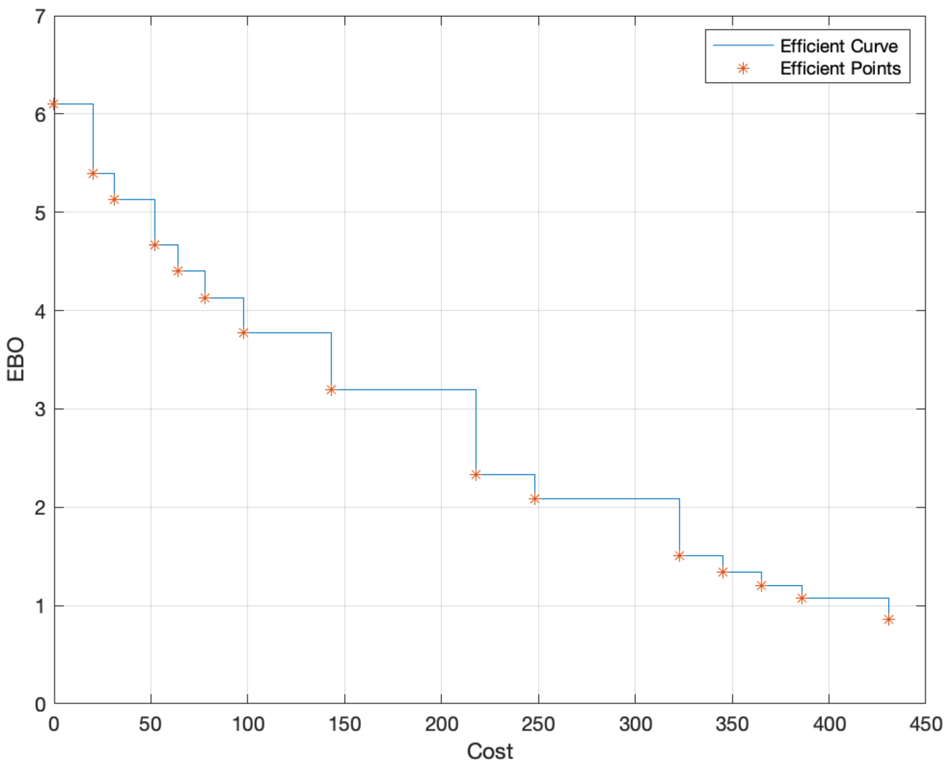

Figure 1: Efficient Curve

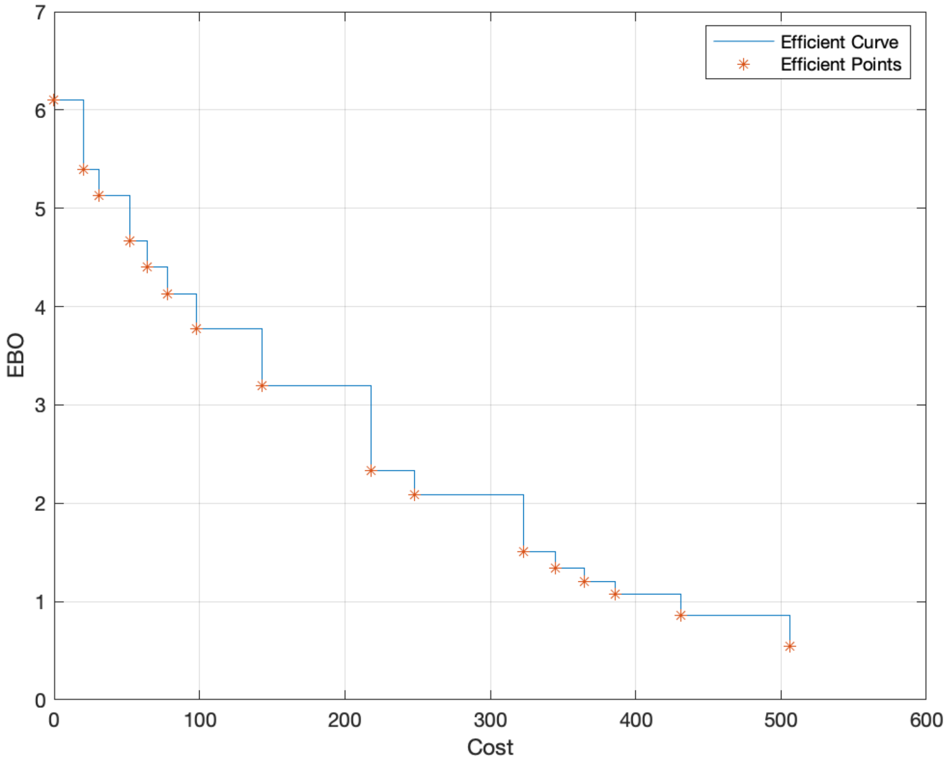

Figure 2: Efficient Curve

Table 2: The efficient solutions, the corresponding objective function values and the marginal decrease in EBO

n	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	Cost	EBO	ΔEBO
0	0	0	0	0	0	0	0	0	0	0	6.10400	-0.71350
1	0	0	0	1	0	0	0	0	0	20	5.39050	-0.25918
2	0	0	0	1	1	0	0	0	0	31	5.13132	-0.46741
3	0	0	1	1	1	0	0	0	0	52	4.66391	-0.25918
4	1	0	1	1	1	0	0	0	0	64	4.40473	-0.27385
5	1	1	1	1	1	0	0	0	0	78	4.13088	-0.35536
6	1	1	1	2	1	0	0	0	0	98	3.77552	-0.57853
7	1	1	1	2	1	1	0	0	0	143	3.19699	-0.86193
8	1	1	1	2	1	1	1	0	0	218	2.33506	-0.24422
9	1	1	1	2	1	1	1	1	0	248	2.09084	-0.58855
10	1	1	1	2	1	1	2	1	0	323	1.50229	-0.16473
11	1	1	1	2	1	1	2	1	1	345	1.33756	-0.13153
12	1	1	1	3	1	1	2	1	1	365	1.20603	-0.13188
13	1	1	2	3	1	1	2	1	1	386	1.07415	-0.21437
14	1	1	2	3	1	2	2	1	1	431	0.85978	-0.31791
15	1	1	2	3	1	2	3	1	1	506	0.54187	0.00000

Dynamic programming

4

In order to implement dynamic programming following terms should be defined for the problem.

- states:= LRU_j denoted as s_j for $j = 1, 2, 3...9$.
- stages:= $\{1, 2, 3....C_{budget}\}$, denoted as c_{stage}.
- decisions:= To buy or not to buy a certain spare part, s_j. $x_j = \begin{cases} 1 & j = l \\ 0 & j \neq l \end{cases}$ where l is an integer such that, $EBO\,(s_l) = \min\{EBO(s_j)\}$.
- state-update equation : $\mathbf{s_n} = \mathbf{s_{n-c(1)+1}} + \mathbf{x_n}$ where $\mathbf{s_n}$ is the vector including all purchased components at stage n, similarly $\mathbf{x_n}$ including all nine decisions at stage n. So $\mathbf{x_n}$ will contain zeros except for the component, l, that will be purchased.
- value functions: $min\{EBO(\mathbf{s})\}$
- recursive relation of the value function: $EBO_j(s_j + 1) = EBO_j(s_j) - R_j(s_j)$

With the initial EBO being, $EBO^{(0)}(s) = \sum_{i=1}^{9} \lambda_i T_i$

- Algorithm

Iterate through stages from 1 to C_{budget} then:

1. If c_{stage} is larger than $min\{c\}$
   - 1.1 For every j generate new EBO_j if we afford to purchase s_j
   - 1.2 Take the minimum of EBO_j which corresponds to EBO_l
   - 1.3 If the EBO_l is lower than EBO at previous stage then:
     - 1.3.1 Update the states
     - 1.3.2 Update EBO
2. Add the latest EBO as last component to EBO_{vek} (Contains previous EBO's)
3. Add the latest $\mathbf{s}$ as last component to s_{mtrx} (Contains previous purchases)

5

From the above described algorithm we obtain the following statistics.

Table 3: Optimal allocation, number of spare parts

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9
$C_{budget} = 0$	0	0	0	0	0	0	0	0	0
$C_{budget} = 100$	1	1	1	2	1	0	0	0	0
$C_{budget} = 150$	1	1	1	2	1	1	0	0	0
$C_{budget} = 350$	1	1	1	2	1	1	2	1	1
$C_{budget} = 500$	1	2	2	2	1	2	3	1	1

And lastly some other information about the dynamic solutions.

Table 4: Optimal allocation, cost and EBO

	Total cost	*EBO*
$C_{budget} = 0$	0	6.1040
$C_{budget} = 100$	98	3.7755
$C_{budget} = 150$	143	3.1970
$C_{budget} = 350$	345	1.3376
$C_{budget} = 500$	500	0.6319

6

Yes, as seen in the plot below we can tell that the dynamic solution does indeed intersect each and every efficient point. This is indicative of it selecting the most efficient solution for those point. We see it using more money in between the efficient points, which is reasonable as uses more of the budget and finds more optimal points between the efficient points.

Note moreover that we did decide to include the point passing the 500 mark, this could easily be removed but we decided to include it for illustrative purposes.

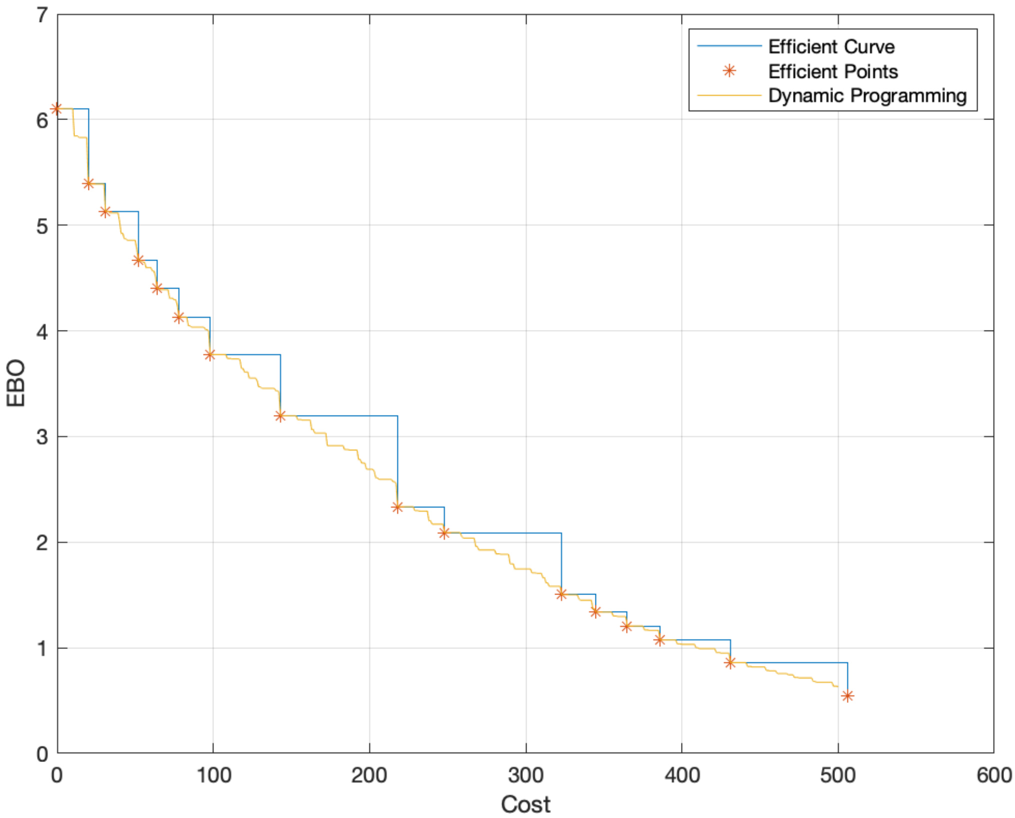

Figure 3: Dynamic programming solution