August Barot

931104-1074

Yue Jiao

911024-7799

System engineering

Assignment 2

**Task 1: Marginal allocation**

The objective is to determine the optimal storage of reserve parts for aircraft, such that the expected number of grounded aircrafts is as low as possible. The aircrafts have seven different parts, each of which can malfunction and thus needs to be replaced. (It is assumed that the probability of an aircraft having two or more malfunctioning parts at the same time is negligibly low). An aircraft with a malfunctioning unit arrives at the storage with a Poisson probability (due to Palm's theorem) distribution of intensity , and it takes time units to repair such an aircraft, assuming there is a unit of type in storage. If that is not the case, the aircraft stays grounded and a backorder is issued. Each unit of type costs money units.  
The vectors lambda, and which gives all failure intensities, repair times and costs for the seven unit types, are the following:

According to the lecture notes the for unit , , is given by the following recursive formula ( is number of spare parts of unit , and is number of aircrafts with a malfunction of unit ):

where. Since the probability of a malfunctioning aircraft is a Poisson distribution, is given by the following formula:

Furthermore, the total of the system is the sum of all (the total number of backorders is obviously the sum of all types of backorders), allowing us to calculate the for a given vector s containing the number of spare parts of all 7 types.

To calculate , we use the fact that due to being a probability.

This is used to avoid having to calculate the value of an infinite series. Therefore, we can calculate the recursively.

This is what we desire to minimize, with a total budget of money units. In order to be able to utilize the MALLOC algorithm, we need to know that is integer-convex, and that (the cost of all spare parts) is integer-convex.

The will be given by , which is an integer-convex function since it is separable and is a strictly increasing function since all and all .

The function is also integer-convex. This is because is a separable function since , and all are strictly decreasing, since is strictly increasing due to being strictly positive. Therefore, MALLOC can be used for these two functions and . We thus need to find expressions for and Δ.

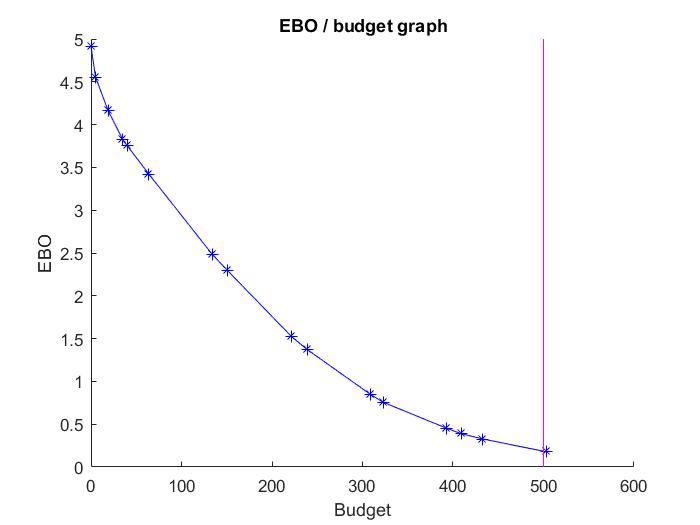
is given by , which is equal to according to the recursive formula for above. is given by , which is equal to which is equal to (since , note the difference between , a function, and , a cost value).Therefore, the quotient used in MALLOC calculations, namely , is given by .

To implement the algorithm, we start with the spare parts vector (no spare parts; = 0 for all ) and calculate the quotients for all . After cancelling the largest quotient, we increase the corresponding by 1 and repeat the process. For each cancellation, we also calculate and , and the algorithm is stopped once . This algorithm gives all points on the efficient curve, all of which are stored in a vector and plotted afterwards. Note that we include the first point on the efficient curve where , this is in order to prove that the point before that is the "ideal" solution to the problem (i.e. the solution which minimizes the EBO under the given budget of ). We also store the corresponding vectors (corresponding to the distribution of spare parts)

The resulting and the corresponding costs are the following:

And point after these is when which is the unreachable since the required budget is .

This returns the following curve:

  
  
We can thus see that the minimum possible is , corresponding to a cost of . The s vector looks as follows: .

The EBO is close to zero, suggesting that the given budget is enough to account for the problems with planes malfunctioning. (Obviously the will go towards zero when all sj go to infinity, but that is not possible with our given budget). We can note that it is actually possible to purchase more parts of type 1-6 with the cost (remaining money = ) but this is not efficient according to the MALLOC calculation (the reduced EBO from additional purchases does not account for the increased cost).

**Task 2: Dynamic Programming**

In this task, we are supposed to solve the exact same problem with dynamic programming. So, we do the following assumption. Let be the number of components we are going to store of the type . Let be the of the component given that the number of components bought of type is . Let be the total budget. Let be the number of types of the components. We want to solve the following optimization problem.

Where . This problem can be rewritten in our expression as following.

To analyse the problem with dynamic programming approach, let the stage means the component of type from to and state be the budget left for the stage . At each stage the decision made is and cost for each component of type is . So, let the optimization problem at stage be the following.

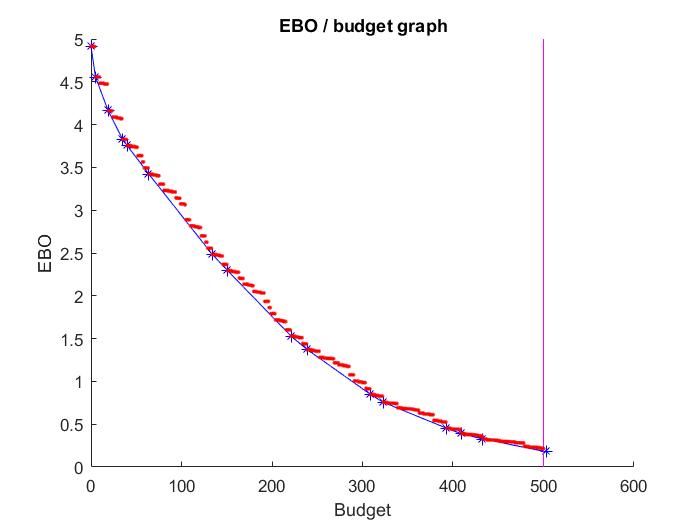
With these formulation, we can express the optimal value of problem by and the minimum of given that at stage n we did the decision shall be . So:

With the definition of the , we know that:

This is then the recursive expression we shall use to calculate the optimal value. By the definition, the total optimized value is given by where is the given budget. Since we only have 7 different components type so when reaches 7 we should put all money in buy the last component, i.e. . Here the function returns the biggest integer that is not bigger than .

To calculate the result, we used Matlab. But Matlab is not good at dealing with recursion so we choose to rewrite the recursion to a loop and rewrite function as a matrix with elements on row and colume. The calculation procedure is however the same. Since we have the boundary condition at the last row so we calculate the matrix backward.

The result is plotted here below.



Here the red dots are the optimal values corresponding to each budget from 0 to 500. We can see that the last red dot is lower than the last possible blue star which is reasonable and at all blue stars, a red dot can be found at the exact same place. So, these two methods give the same result.

From the dynamic programming, we get the minimum reachable is with the budget .

**Matlab codes:**

|  |
| --- |
| %% data  lambda = [55, 43, 36, 70, 29, 45, 111]'/1000;  c = [5, 18, 14, 17, 16, 24, 70]';  T = [8, 4, 14, 3, 14, 9, 25]';  C\_max = 500;  s\_max = ceil(C\_max/(min(c)));  N = size(c)\*[1; 0];  %% Problem 1  p = zeros(N, s\_max);  for j = 1:1:N  for k = 0:1:s\_max  p(j, k+1) = (((lambda(j)\*T(j))^k)/(factorial(k)))\*exp((-lambda(j)\*T(j)));  end  end  R = zeros(N, s\_max);  for sj = 0:1:s\_max  for j = 1:1:N  R(j, sj+1) = 1;  for i = 0:1:sj  R(j, sj+1) = R(j, sj+1)- p(j, i+1);  end  end  end  M = zeros(N, s\_max);  for sj = 0:1:s\_max  for j = 1:1:N  M(j, sj+1) = R(j, sj+1)/c(j);  end  end  M = M';  s = [zeros(1,N)];  C = [0];  pi = 0;  for i=1:1:N  pi = pi + lambda(i)\*T(i);  end  EBO = [pi];  k = 1;  while 1==1  mk = zeros(1,N);  for j = 1:1:N  mk(1, j) = M(s(k,j)+1, j);  end  [~, l] = max(mk);  sl = s(k,:);  k = k+1;  s(k, :) = sl;  s(k, l) = s(k, l)+1;  C(k) = C(k-1) + c(l);  EBO(k) = EBO(k-1) - R(l, s(k-1, l)+1);  if C(k) >= C\_max  break  end  end  %% Problem 2  EBOi = zeros(N,1);  for i=1:1:N  EBOi(i, 1) = lambda(i)\*T(i);  end  for i=1:1:s\_max+1  EBOi = [EBOi EBOi(:, i)-R(:, i)];  end  f = zeros(N+1,C\_max+1);  for k = N:-1:1  for s=0:1:C\_max  sup = floor(s/c(k));  list = zeros(1,sup+1);  for xi = 0:1:sup  list(xi+1) = EBOi(k, xi+1)+f(k+1, s-xi\*c(k)+1);  end  f(k, s+1) = min(list);  end  end  %% Plots  hold on;  xlabel('Budget');  ylabel('EBO');  title('EBO / budget graph');  plot(C(:), EBO(:),'\*-b');  plot([0:1:C\_max], f(1,:), '.r');  plot(C\_max\*[1,1], [floor(min(EBO(:))) ceil(max(EBO(:)))], 'm-'); |