UNIT-III

Dynamic programming: General method, Multistage graphs, All pairs shortest paths, 0/1 Knapsack problem, Reliability design problem, Travelling sales person problem

DYNAMIC PROGRAMING GENERAL METHOD:

1. The idea of dynamic programming is thus quit simple: avoid calculating the same thing twice, usually by keeping a table of known result that fills up a sub instances are solved.
2. Divide and conquer is a top-down method.
3. When a problem is solved by divide and conquer, we immediately attack the complete instance, which we then divide into smaller and smaller sub- instances as the algorithm progresses.
4. Dynamic programming on the other hand is a bottom-up technique.
5. We usually start with the smallest and hence the simplest sub-instances.
6. By combining their solutions, we obtain the answers to sub-instances of increasing size, until finally we arrive at the solution of the original instances.
7. The essential difference between the greedy method and dynamic programming is that the greedy method only one decision sequence is evergenerated.
8. In dynamic programming, many decision sequences may be generated. However, sequences containing sub-optimal sub-sequences can not beoptimal and so will not be generated.

2.5. MULTISTAGE GRAPH:

A multistage graph G = (V,E) is a directed graph in which the vertices areportioned into K > = 2 disjoint sets Vi, 1 <= i<= k.

In addition, if < u,v > is an edge in E, then u < = Vi and V Vi+1 for some i, 1<= i < k.

If there will be only one vertex, then the sets Vi and Vk are such that [Vi]=[Vk] = 1.

Let ‘s’ and ‘t’ be the source and destination respectively.

The cost of a path from source (s) to destination (t) is the sum of the costs ofthe edger on the path.

The MULTISTAGE GRAPH problem is to find a minimum cost path

from ‘s’ to‘t’.

Each set Vi defines a stage in the graph. Every path from ‘s’ to ‘t’ starts instage-1, goes to stage-2 then to stage-3, then to stage-4, and so on, and terminates in stage-k.

This MULISTAGE GRAPH problem can be solved in 2 ways. o

Forward Method.

o Backward Method.

2.8.1. FORWARD METHOD

Assume that there are ‘k’ stages in a graph.

In this FORWARD approach, we will find out the cost of each and every nodestarling from the ‘k’ th stage to the 1st stage.

We will find out the path (i.e.) minimum cost path from source to thedestination (ie) [Stage-1 to Stage-k].

PROCEDURE:

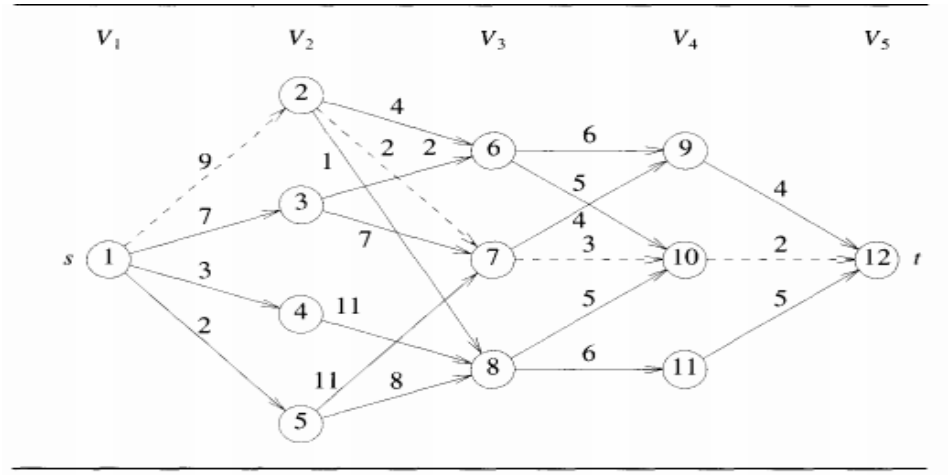


Figure 5.2 Five-stage graph

1. maintain a cost matrix cost (n) which stores the distance from any vertex tothe destination.
2. If a vertex is having more than one path, then we have to choose the minimum distance path and the intermediate vertex, which gives the minimum distance path, will be stored in the distance array ‘D’.
3. In this way we will find out the minimum cost path from each

and everyvertex.

4. Finally cost(1) will give the shortest distance from source to destination.

5. For finding the path, start from vertex-1 then the distance array D(1) will give the minimum cost neighbour vertex which in turn give the next nearest vertex and proceed in this way till we reach the Destination.

6. For a 'k' stage graph, there will be 'k' vertex in the path.

7. In the above graph V1…V5 represent the stages. This 5 stage graph can besolved by using forward approach as follows,

STEPS: -		DESTINATION, D
Cost (12)=0	→	D (12)=0
Cost (11)=5	→	D (11)=12
Cost (10)=2	→	D (10)=12
Cost (9)=4	→	D (9)=12

1. For forward approach,

$$Cost(i,j) = \min \{C(j,l) + Cost(i+1,l)\} \quad l \in V_{i+1}, (j,l) \in E$$

cost(8) = min {C (8,10) + Cost (10), C (8,11) + Cost (11) }

= min (5 + 2, 6 + 5)

= min (7,11)

= 7

cost(8) =7 =>D(8)=10

cost(7) = min(c (7,9)+ cost(9),c

(7,10)+ cost(10))(4+4,3+2)

= min(8,5)

= 5

cost(7) = 5 =>D(7) = 10

cost(6) = min (c (6,9) + cost(9),c (6,10) +cost(10))

= min(6+4 , 5 +2)

= min(10,7)

= 7

cost(6) = 7 =>D(6) = 10

cost(5) = min (c (5,7) + cost(7),c (5,8) +cost(8))

= min(11+5 , 8 +7)

= min(16,15)

= 15

cost(5) = 15 =>D(5) = 18

cost(4) = min (c (4,8) + cost(8))

= min(11+7)

= 18

cost(4) = 18 =>D(4) = 8

cost(3) = min (c (3,6) + cost(6),c (3,7) +cost(7))

= min(2+7 , 7 +5)

= min(9,12)= 9

cost(3) = 9 =>D(3) = 6

cost(2) = min (c (2,6) + cost(6),c (2,7) +cost(7) ,c (2,8) +cost(8))

= min(4+7 , 2+5 , 1+7)

= min(11,7,8)

= 7

cost(2) = 7 =>D(2) = 7

cost(1) = min (c (1,2)+cost(2) ,c (1,3)+cost(3) ,c (1,4)+cost(4) ,c(1,5)+cost(5))

= min(9+7 , 7 +9 , 3+18 , 2+15)

= min(16,16,21,17)

= 16

cost(1) = 16 =>D(1) = 2

..... ⟶ The path through which you have to find the shortest distance.

(i.e.) ① ⟶ ② ⟶ ⑦ ⟶ ⑩ ⟶ ⑫

Start from

vertex - 2

D(1) = 2

D(2) = 7

D(7) =10

D (10) = 12

So, the minimum –cost path is,

The cost is 9+2+3+2+=16

ALGORITHM:

FORWARD METHOD

Algorithm FGraph (G,k,n,p)

// The I/p is a k-stage graph G=(V,E) with ‘n’ vertex.

// Indexed in order of stages E is a set of edges.

// and c[i,J] is the cost of<i,j>,p[1:k] is a minimum cost path.

{

cost[n]=0.0;

for j=n-1 to 1 step-1 do

{

//compute cost[j],

// let ‘r’ be the vertex such that <j,r> is an edge of ‘G’ &

// c[j,r]+cost[r] is minimum.

cost[j] =

c[j+r] +

cost[r];d[j]

=r;

}

// find a minimum cost path.

P[1]=1;

```
        P[k]=n;
        For
        j=2
        to k-
        1 do
        P[j]=
        d[p[j-
        1]];
}
```

ANALYSIS:

The time complexity of this forward method is O($|V| + |E|$)

2.8.2.BACKWARD METHOD

if there one 'K' stages in a graph using back ward approach. we will findout the cost of each & every vertex starting from 1^{st} stage to the k^{th} stage.

We will find out the minimum cost path from destination to source (ie)[fromstage k to stage 1]

PROCEDURE:

It is similar to forward approach, but differs only in two or three ways.

Maintain a cost matrix to store the cost of every vertices and a distancematrix to store the minimum distance vertex.

Find out the cost of each and every vertex starting from vertex 1 up tovertex k.

To find out the path star from vertex 'k', then the distance array D (k) will give the minimum cost neighbor vertex which in turn gives the next nearest neighbor vertex and proceed till we reach the destination.

STEP:

Cost(1) =

0 =>

D(1)=0

Cost(2) =

9 =>

D(2)=1

Cost(3) =

7 =>

D(3)=1

Cost(4) =

3 =>

D(4)=1

Cost(5) =

2 =>

D(5)=1

Cost(6) =min(c (2,6) + cost(2),c (3,6) + cost(3))

=

min(13,9)

cost(6) =

9

=>D(6)=

3

Cost(7) =min(c (3,7) + cost(3),c (5,7) + cost(5) ,c (2,7) + cost(2))

=m

in(14,13,11)

cost(7) = 11

=>D(7)=2

Cost(8) =min(c (2,8) + cost(2),c (4,8) + cost(4) ,c (5,8) +cost(5))

=m

in(10,14,10)

cost(8) = 10

=>D(8)=2

Cost(9) =min(c (6,9) + cost(6),c (7,9) + cost(7))

=

min(15,15)

cost(9) =

15

=>D(9)=6

Cost(10)=min(c(6,10)+cost(6),c(7,10)+cost(7)),c(8,10)+cost(8))
=min(14,14,15)

cost(10)= 14 =>D(10)=6

Cost(11) =min(c

(8,11) + cost(8))

cost(11) = 16

=>D(11)=8

cost(12)=min(c(9,12)+cost(9),c(10,12)+cost(10),c(11,12)+cost(11))

=mi

n(19,16,21)

cost(12) = 16

=>D(12)=10

PATH:

Start

from

verte

x-12

D(12)

= 10

D(10) = 6

D(6) = 3

D(3) = 1

So the minimum cost path is,

$1^{7} \rightarrow 3^{2} \rightarrow 6^{5} \rightarrow 10^{2} \rightarrow 12$

The cost is 16.

ALGORITHM : BACKWARD METHOD

```
Algorithm BGraph (G,k,n,p)
//  The I/p is a k-stage graph G=(V,E) with 'n' vertex.
//  Indexed in order of stages E is a set of edges.
// and c[i,J] is the cost of<i,j>,p[1:k] is a minimum cost path.
{
     bcost[
      1]=0.
      0;
      for
      j=2
      to n
      do
  {
      //compute bcost[j],
     //    let 'r' be the vertex such that <r,j> is an edge of 'G' &
    //    bcost[r]+c[r,j] is minimum.
     bcost[j] =
      bcost[r] +
      c[r,j];d[j] =r;
   }
  // find a minimum cost path.

        P[1]=1;
        P[k]=n;
```

```
For j=
k-1 to
2 do
P[j]=d
[p[j+1
]];
}
```

2.6. TRAVELLING SALESMAN PROBLEM

Let G(V,E) be a directed graph with edge cost c_{ij} is defined such that c_{ij} >0 forall i and j and c_{ij} = ,if <i,j> E.

Let V=h and assume n>1.

The traveling salesman problem is to find a tour of minimum

cost. Atour of G is a directed cycle that include every vertex in

V.

The cost of the tour is the sum of cost of the edges on the tour.

The tour is the shortest path that starts and ends at the same vertex (ie) 1.

APPLICATION :

Suppose we have to route a postal van to pick up mail from the mail boxeslocated at 'n' different sites.

An n+1 vertex graph can be used to represent the situation.

One vertex represent the post office from which the postal van starts andreturn.

Edge <i,j> is assigned a cost equal to the distance from site 'i' to site 'j'.

the route taken by the postal van is a tour and we are finding a tour of minimum length.

every tour consists of an edge <1,k> for some k V-{} and a path from vertexk to vertex 1.

the path from vertex k to vertex 1 goes through each vertex in V-{1,k}exactly once.

the function which is used to

find the path is g(1,V-{1}) =

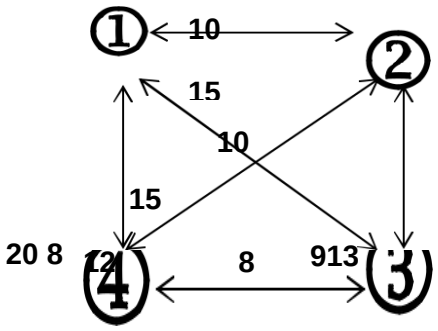
min{ c_{ij} + g(j,s-{j})}

g(i,s) be the length of a shortest path starting at vertex i,

goingthrough all vertices in S,and terminating at vertex 1.

the function g(1,v-{1}) is the length of an optimal tour.

1. Find g(i,) =c_{i1}, 1<=i<n, hence we can use equation(2) to obtain g(i,s) for alls to size 1.
2. That we have to start with s=1,(ie) there will be only one vertex in set 's'.
3. Then s=2, and we have to proceed until |s| <n-1.
4. for example consider the graph.



Cost matrix

$$\begin{vmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{vmatrix}$$

g(i,s) set of nodes ⟶

↓

/vertex have to visited.

starting position

g(i,s) =min{c_{ij} +g(j,s-{j})

STEP 1:

g(1,{2,3,4})=min{c12+g(2{3,4}),c13+g(3,{2,4}),c14+g(4,{2,3})}

min{10+25,15+25,20+23}

min{35,35,43}

=35

STEP 2:

g(2,{3,4}) = min{c23+g(3{4}),c24+g(4,{3})}

min{9+20,10+15}min{29,25}=25

g(3,{2,4}) =min{c32+g(2{4}),c34+g(4,{2})}

min{13

+18,12+13}

min{31,25}

=25

g(4,{2,3}) = min{c42+g(2{3}),c43+g(3,{2})}

STEP3:min{8+15,9+8}min{23,27}=23

1. g(3,{4}) = min{c34 +g{4, }} 12+8=20
2. 2. g(4,{3}) = min{c43 +g{3, }} 9+6

=15

3. g(2,{4}) = min{c24 +g{4, }} 10+8

=18

4. g(4,{2}) = min{c42 +g{2, }} 8+5

=13

5. g(2,{3}) = min{c23 +g{3, }}

9+6=15

6. g(3,{2}) = min{c32 +g{2, }}

13+5=18

STEP 4:

g{4, } =c41 = 8 g{3, }

=c31 = 6 g{2, } =c21

= 5

$|s| = 0.$

i =1 to n.

g(1,) = c11 => 0

g(2,) =c21 => 5

g(3,) = c31 => 6

g(4,) = c41 => 8

$|s| = 1$

i=

2 to 4

g(2,{3}) =

c23 + g(3,

)

= 9+6 =15

g(2,{4}) = c24 + g(4,)

= 10+8 =18

g(3,{2}) = c32 + g(2,)

= 13+5 =18

g(3,{4}) = c34 + g(4,)

= 12+8 =20

g(4,{2}) = c42 + g(2,)

= 8+5 =13

g(4,{3}) = c43 + g(3,)

= 9+6 =15

$|s| = 2$

i 1, 1 s and i s.

g(2,{3,4}) = min{c23+g(3{4}),c24+g(4,{3})}

min{9+20,10+15}

min{29,25}=25

g(3,{2,4}) =min{c32+g(2{4}),c34+g(4,{2})}

min{13+18,12+13}min{31,25}=25

g(4,{2,3}) = min{c42+g(2{3}),c43+g(3,{2})}

min{8+15,9+18}min{23,27}=23

$|s| = 3$

g(1,{2,3,4})=min{c12+g(2{3,4}),c13+g(3,{2,4}),c14+g(4,{2,3})}

min{10+25,15+25,20+23}

min{35,35,43}=35

Optimal cost is 35

the shortest path is,

g(1,{2,3,4}) = c12 + g(2,{3,4}) => 1->2

g(2,{3,4}) = c24 + g(4,{3}) => 1->2->4

g(4,{3}) = c43 + g(3{ }) => 1->2->4->3->1

so the optimal tour is 1 2 4 3 1

All pairs shortest paths:

In the all pairs shortest path problem, we are to find a shortest path between every pair of vertices in a directed graph G. That is, for every pair of vertices (i, j), we are to find a shortest path from i to j as well as one from j to i. These two paths are the

same when G is undirected.
When no edge has a negative length, the all-pairs shortest path problem may be solved by using Dijkstra's greedy single source algorithm n times, once with each of the n vertices as the source vertex.
The all pairs shortest path problem is to determine a matrix A such that A (i, j) is the length of a shortest path from i to j. The matrix A can be obtained by solving n single-source problems using the algorithm shortest Paths. Since each application of this procedure requires O (n2) time, the matrix A can be obtained in O (n3) time.
The dynamic programming solution, called Floyd's algorithm, runs in O (n3) time. Floyd's algorithm works even when the graph has negative length edges (provided there are no negative length cycles).
The shortest i to j path in G, $i \neq j$ originates at vertex i and goes through some intermediate vertices (possibly none) and terminates at vertex j. If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j, respectively. Otherwise, the i to j path is not of minimum length. So, the principle of optimality holds. Let A k (i, j) represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain:

$A^k(i, j) = \{\min \{\min \{A^{k-1}(i, k) + A^{k-1}(k, j)\}, c(i, j)\}$
$1<k<n$

Algorithm All Paths (Cost, A, n)

```
// cost [1:n, 1:n] is the cost adjacency matrix of a graph which
// n vertices; A [I, j] is the cost of a shortest path from vertex
// i to vertex j. cost [i, i] = 0.0, for 1 < i < n.
{
for i := 1 to n do
for j:= 1 to n do
A [i, j] := cost [i, j];
// copy cost into A.
for k := 1 to n do
for i := 1 to n do
for j := 1 to n do
A [i, j] := min (A [i, j], A [i, k] + A [k, j]);
}
```

Complexity Analysis: A Dynamic programming algorithm based on this recurrence involves in calculating n+1 matrices, each of size n x n. Therefore, the algorithm has a complexity of O (n3).

Example 1:
Given a weighted digraph G = (V, E) with weight. Determine the length of the shortest path between all pairs of vertices in G. Here we assume that there are no cycles with zero or negative cost.

cost adjacency matrix (A_0) = $\begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$

General formula: $\min \{A^{k-1}(i, k) + A^{k-1}(k, j)\}, c(i, j)\}$
$1<k<n$
Solve the problem for different values of k = 1, 2 and 3
Step 1: Solving the equation for, k = 1;

$A^1(1, 1) = \min \{(A^0(1, 1) + A^0(1, 1)), c(1, 1)\} = \min \{0 + 0, 0\} = 0$

$A^1(1,2) = \min\{(A^o(1,1) + A^o(1,2)), c(1,2)\} = \min\{(0+4), 4\} = 4$
$A^1(1,3) = \min\{(A^o(1,1) + A^o(1,3)), c(1,3)\} = \min\{(0+11), 11\} = 11$
$A^1(2,1) = \min\{(A^o(2,1) + A^o(1,1)), c(2,1)\} = \min\{(6+0), 6\} = 6$
$A^1(2,2) = \min\{(A^o(2,1) + A^o(1,2)), c(2,2)\} = \min\{(6+4), 0)\} = 0$
$A^1(2,3) = \min\{(A^o(2,1) + A^o(1,3)), c(2,3)\} = \min\{(6+11), 2\} = 2$
$A^1(3,1) = \min\{(A^o(3,1) + A^o(1,1)), c(3,1)\} = \min\{(3+0), 3\} = 3$
$A^1(3,2) = \min\{(A^o(3,1) + A^o(1,2)), c(3,2)\} = \min\{(3+4), \square\} = 7$
$A^1(3,3) = \min\{(A^o(3,1) + A^o(1,3)), c(3,3)\} = \min\{(3+11), 0\} = 0$

$$A^{(1)} = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

Step 2: Solving the equation for, K = 2;
$A^2(1,1) = \min\{(A^1(1,2) + A^1(2,1), c(1,1)\} = \min\{(4+6), 0\} = 0$
$A^2(1,2) = \min\{(A^1(1,2) + A^1(2,2), c(1,2)\} = \min\{(4+0), 4\} = 4$
$A^2(1,3) = \min\{(A^1(1,2) + A^1(2,3), c(1,3)\} = \min\{(4+2), 11\} = 6$
$A^2(2,1) = \min\{(A^1(2,2) + A^1(2,1), c(2,1)\} = \min\{(0+6), 6\} = 6$
$A^2(2,2) = \min\{(A^1(2,2) + A^1(2,2), c(2,2)\} = \min\{(0+0), 0\} = 0$
$A^2(2,3) = \min\{(A^1(2,2) + A^1(2,3), c(2,3)\} = \min\{(0+2), 2\} = 2$
$A^2(3,1) = \min\{(A^1(3,2) + A^1(2,1), c(3,1)\} = \min\{(7+6), 3\} = 3$
$A^2(3,2) = \min\{(A(3,2) + A(2,2), c(3,2)\} = \min\{(7+0), 7\} = 7$
$A^2(3,3) = \min\{A^1(3,2) + A^1(2,3), c(3,3)\} = \min\{(7+2), 0\} = 0$

$$A^{(2)} = \begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

Step 3: Solving the equation for, k = 3;
$A^3(1,1) = \min\{A_2(1,3) + A_2(3,1), c(1,1)\} = \min\{(6+3), 0\} = 0$
$A^3(1,2) = \min\{A_2(1,3) + A_2(3,2), c(1,2)\} = \min\{(6+7), 4\} = 4$
$A_3(1,3) = \min\{A_2(1,3) + A_2(3,3), c(1,3)\} = \min\{(6+0), 6\} = 6$
$A_3(2,1) = \min\{A_2(2,3) + A_2(3,1), c(2,1)\} = \min\{(2+3), 6\} = 5$
$A_3(2,2) = \min\{A_2(2,3) + A_2(3,2), c(2,2)\} = \min\{(2+7), 0\} = 0$
$A_3(2,3) = \min\{A_2(2,3) + A_2(3,3), c(2,3)\} = \min\{(2+0), 2\} = 2$
$A_3(3,1) = \min\{A_2(3,3) + A_2(3,1), c(3,1)\} = \min\{(0+3), 3\} = 3$
$A_3(3,2) = \min\{A_2(3,3) + A_2(3,2), c(3,2)\} = \min\{(0+7), 7\} = 7$
$A_3(3,3) = \min\{A_2(3,3) + A_2(3,3), c(3,3)\} = \min\{(0+0), 0\} = 0$

$$A(3) = \begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

This is resultant cost path matrix
⌋

0/1 – KNAPSACK:

We are given n objects and a knapsack. Each object i has a positive weight w_i and a positive value V_i. The knapsack can carry a weight not exceeding W. Fill the knapsack so that the value of objects in the knapsack is optimized.
A solution to the knapsack problem can be obtained by making a sequence of decisions on the variables $x_1, x_2, \ldots, x_n$. A decision on variable x_i involves determining which of the values 0 or 1 is to be assigned to it. Let us assume that decisions on the x_i are made in the order x_n, x_{n-1}, x_1. Following a decision on x_n,

we may be in one of two possible states: the capacity remaining in m – w_n and a profit of p_n has accrued. It is clear that the remaining decisions x_{n-1}, , x_1 must be optimal with respect to the problem state resulting from the decision on x_n. Otherwise, x_n, , x_1 will not be optimal. Hence, the principal of optimality holds.

$F_n(m) = \max \{f_{n-1}(m), f_{n-1}(m - w_n) + p_n\}$

--
1

For arbitrary $f_i(y)$, i > 0, this equation generalizes to:

$F_i(y) = \max \{f_{i-1}(y), f_{i-1}(y - w_i) + p_i\}$

--
2

Equation-2 can be solved for $f_n(m)$ by beginning with the knowledge $f_o(y) = 0$ for all y and $f_i(y) = -\infty$, y < 0. Then f_1, f_2, . . . f_n can be successively computed using equation–2.

When the w_i's are integer, we need to compute $f_i(y)$ for integer y, 0 < y < m. Since $f_i(y) = -\infty$ for y < 0, these function values need not be computed explicitly. Since each f_i can be computed from f_i - 1 in **Θ** (m) time, it takes **Θ** (m n) time to compute f_n. When the wi's are real numbers, $f_i(y)$ is needed for real numbers y such that 0 < y < m. So, f_i cannot be explicitly computed for all y in this range. Even when the w_i's are integer, the explicit **Θ** (m n) computation of f_n may not be the most efficient computation. So, we explore **an alternative method for both cases.**

The $f_i(y)$ is an ascending step function; i.e., there are a finite number of y's, $0 = y_1 < y_2 < \ldots < y_k$, such that $f_i(y_1) < f_i(y_2) < \ldots < f_i(y_k)$; $f_i(y) = -\infty$, $y < y_1$; $f_i(y) = f(y_k)$, $y > y_k$; and $f_i(y) = f_i(y_j)$, $y_j < y < y_{j+1}$. So, we need to compute only $f_i(y_j)$, 1 < j < k. We use the ordered set $S_i = \{(f(y_j), y_j) \mid 1 < j < k\}$ to represent $f_i(y)$. Each number of S_i is a pair (P, W), where $P = f_i(y_j)$ and $W = y_j$. Notice that $S_0 = \{(0, 0)\}$. We can compute S_{i+1} from S_i by first computing:

$S_i = \{(P, W) \mid (P - p, W - w) \in S_i\}$

Now, S_{i+1} can be computed by merging the pairs in S_i and S_i together. Note that if S_{i+1} contains two pairs (P_j, W_j) and (P_k, W_k) with the property that $P_j < P_k$ and $W_j > W_k$, then the pair (P_j, W_j) can be discarded because of equation-2. Discarding or purging rules such as this one are also known as dominance rules. Dominated tuples get purged. In the above, (P_k, W_k) dominates (P_j, W_j).

Example 1:

Consider the knapsack instance n = 3, $(w_1, w_2, w_3) = (2, 3, 4)$, $(P_1, P_2, P_3) = (1,2, 5)$ and M = 6.74

1
1
1
1

Solution:

Initially, $f_o(x) = 0$, for all x and $f_i(x) = -\infty$ if x < 0.

$F_n(M) = \max \{f_{n-1}(M), f_{n-1}(M - w_n) + p_n\}$

$F_3(6) = \max (f_2(6), f_2(6 – 4) + 5\} = \max \{f_2(6), f_2(2) + 5\}$

$F_2(6) = \max (f_1(6), f_1(6 – 3) + 2\} = \max \{f_1(6), f_1(3) + 2\}$

$F_1(6) = \max (f_0(6), f_0(6 – 2) + 1\} = \max \{0, 0 + 1\} = 1$

$F_1(3) = \max (f_0(3), f_0(3 – 2) + 1\} = \max \{0, 0 + 1\} = 1$

Therefore, $F_2(6) = \max (1, 1 + 2\} = 3$

$F_2(2) = \max (f_1(2), f_1(2 – 3) + 2\} = \max \{f_1(2), -\infty + 2\}$

$F_1(2) = \max (f_0(2), f_0(2 – 2) + 1\} = \max \{0, 0 + 1\} = 1$

$F_2(2) = \max \{1, -\infty + 2\} = 1$

Finally, $f_3(6) = \max \{3, 1 + 5\} = 6$

Other Solution:

For the given data we have:

$S_0 = \{(0, 0)\}$; $S_0 = \{(1, 2)\}$

$S_1 = (S_0 \cup S_0) = \{(0, 0), (1, 2)\}$

X - 2 = 0 => x = 2.

y – 3 = 0 => y = 3

X - 2 = 1 => x = 3.

y – 3 = 2 => y = 5

$S_{11} = \{(2, 3), (3, 5)\}$

$S^2 = (S^1 \cup S^1{}_1) = \{(0, 0), (1, 2), (2, 3), (3, 5)\}$
$X - 5 = 0 \Rightarrow x = 5.$
$y - 4 = 0 \Rightarrow y = 4$
$X - 5 = 1 \Rightarrow x = 6.$
$y - 4 = 2 \Rightarrow y = 6$
$X - 5 = 2 \Rightarrow x = 7.$
$y - 4 = 3 \Rightarrow y = 7$
$X - 5 = 3 \Rightarrow x = 8.$
$y - 4 = 5 \Rightarrow y = 9$
$S^{21} = \{(5, 4), (6, 6), (7, 7), (8, 9)\}$
$S^3 = (S^2 \cup S^2) = \{(0, 0), (1, 2), (2, 3), (3, 5), (5, 4), (6, 6), (7, 7), (8, 9)\}$
By applying Dominance rule,
$S^3 = (S^2 \cup S^2) = \{(0, 0), (1, 2), (2, 3), (5, 4), (6, 6)\}$
From (6, 6) we can infer that the maximum Profit $\sum p_i x_i = 6$ and weight $\sum x_i w_i = 6$

Reliability Design:

The problem is to design a system that is composed of several devices connected in series. Let ri be the reliability of device Di (that is ri is the probability that device i will function properly) then the reliability of the entire system is □ ri. Even if the individual devices are very reliable (the ri's are very close to one), the reliability of the system may not be very good. For example, if n = 10 and ri = 0.99, i < i < 10, then □ ri = .904. Hence, it is desirable to duplicate devices. Multiply copies of the same device type are connected in parallel.

If stage i contains mi copies of device Di. Then the probability that all mi have a malfunction is (1 - r)

m_i . Hence the reliability of stage i becomes $1 - (1 - r)m^i$

The reliability of stage 'i' is given by a function $\phi_i(m_i)$.

Our problem is to use device duplication. This maximization is to be carried out under a cost constraint. Let c_i be the cost of each unit of device i and let c be the maximum allowable cost of the system being designed.

We wish to solve:

Maximize

$$\prod_{1<i<n} \phi_i(m_i)$$

Subject to $\sum_{1<i<n} Cm_i < C$

$m_i > 1$ and interger, $1 < i < n$

$$\sum$$

Assume each $C_i > 0$, each m_i must be in the range $1 < m_i < u_i$, where

$u_i = | \ | C + C_i - \sum C_J \ | C_i$ The upper bound u_i follows from the observation that $m_j > 1$

An optimal solution $m_1, m_2 \ldots\ldots\ldots\ldots m_n$ is the result of a sequence of decisions, one decision for each m_i.

Let $f_i(x)$ represent the maximum value of

Subject to the constrains:

$$\pi_{1<j<I} \phi(m_J)$$

$$\sum C_J m_J < x$$

$1 < j <$ i and $1 < cim_j < u_J$, $1 < j < i$

Example :

Design a three stage system with device types D_1, D_2 and D_3. The costs are $30, $15 and $20 respectively. The Cost of the system is to be no more than $105. The reliability of each device is 0.9, 0.8 and 0.5 respectively.

Solution:

We assume that if if stage I has mi devices of type i in parallel, then $\phi_i(m_i) = 1 - (1 - r_i)^{m_i}$

Since, we can assume each $c_i > 0$, each m_i must be in the range $1 \le m_i \le u_i$. Where:

| (

n

$u_i = \lfloor C + C_i - C_J \rfloor C_i \rfloor$

Using the above equation compute u_1, u_2 and u_3.

$105 + 30 - (30 + 15 + 20)$

$u_1 = = 70/30 = 2$

$105 + 15 - (30 + 15 + 20) = 55/15 = 3$

$u_3 = 105 + 20 - (30 + 15 + 20)/20 = 60/20 = 3$

We use S^i_j

$\varphi \rightarrow i$: *stage number and J*: *no. of devices in stage i* $= m_i$

$S_o = \{f_o(x), x\}$ *initially* $f_o(x) = 1$ *and* $x = 0$, *so*, $S_o = \{1,0\}$

Compute S_1, S_2 and S_3 as follows:

S_1 = depends on u_1 value, as $u_1 = 2$, so

$S_1 = \{S^1_1, S^1_2\}$

S_2 = depends on u_2 value, as $u_2 = 3$, so

$\{S^2_1, S^2_2, S^2_3\}$

S_3 = depends on u_3 value, as $u_3 = 3$, so

$S_3 = \{S_{3\,1}, S_{3\,2}, S_{3\,3}\}$

Now find, $S^1_1 = \{(f(x), x)\}$

$f_1(x) = \{\phi_1(1) f_o(\,), \phi_1(2) f_0()\}$ With devices $m_1 = 1$ and $m_2 = 2$

Compute $\phi_1(1)$ and $\phi_1(2)$ using the formula: $\phi_i(mi)) = 1 - (1 - r_i)^{mi}$

$\phi_1(1) = 1 - (1 - r_1)^{m_1} = 1 - (1 - 0.9)_1 = 0.9$

$\phi_1(2) = 1 - (1 - 0.9)^2 = 0.99$

$S_1 = \{f_1(x), x\} = = (0.9, 30)$

$S^1_1 = \{0.99, 30 + 30\} = (0.99, 60)$

Therefore, $S_1 = \{(0.9, 30), (0.99, 60)\}$

Next find $S^2_1 = \{(f(x), x)\}$

$f_2(x) = \{\phi_2(1) * f_1(\,), \phi_2(2) * f_1(\,), \phi_2(3) * f_1(\,)\}$

$\phi_2(1) = 1 - (1 - r_I)$

$= 1 - (1 - 0.8)^1 = 1 - 0.2 = 0.8$

$\phi_2(2) = 1 - (1 - 0.8)^2 = 0.96$

$\phi_2(3) = 1 - (1 - 0.8)^3 = 0.992$

$S^2 = \{(0.8(0.9), 30 + 15), (0.8(0.99), 60 + 15)\} = \{(0.72, 45), (0.792, 75)\}$

$S^2 = \{(0.96(0.9), 30 + 15 + 15), (0.96(0.99), 60 + 15 + 15)\}$

$= \{(0.864, 60), (0.9504, 90)\}$

$S^2 = \{(0.992(0.9), 30 + 15 + 15 + 15), (0.992(0.99), 60 + 15 + 15 + 15)\}$

$= \{(0.8928, 75), (0.98208, 105)\}$

$S^2 = \{S_{2\,1}, S_{2\,2}, S_{2\,3}\}$

By applying Dominance rule to S_2 :

Therefore, $S_2 = \{(0.72, 45), (0.864, 60), (0.8928, 75)\}$

Dominance Rule:
If S_i contains two pairs (f_1, x_1) and (f_2, x_2) with the property that $f_1 \geq f_2$ and $x_1 \leq x_2$, then (f_1, x_1) dominates (f_2, x_2), hence by dominance rule (f_2, x_2) can be discarded. Discarding or pruning rules such as the one above is known as dominance rule. Dominating tuples will be present in S_i and Dominated tuples has to be discarded from S_i.
Case 1: if $f_1 \leq f_2$ and $x_1 > x_2$ then discard (f_1, x_1)
Case 2: if $f_1 > f_2$ and $x_1 < x_2$ the discard (f_2, x_2)
Case 3: otherwise simply write (f_1, x_1)
$S_2 = \{(0.72, 45), (0.864, 60), (0.8928, 75)\}$

$\phi_3(1) = 1 - (1 - r_1)^{m_i} = 1 - (1 - 0.5)_1 = 1 - 0.5 = 0.5$

$\phi_3(2) = 1 - (1 - 0.5)^2$

$\phi_3(3) = 1 - (1 - 0.5)^3$

$= 0.75$

$= 0.875$

$S^3_1 = \{(0.5\ (0.72), 45 + 20), (0.5\ (0.864), 60 + 20), (0.5\ (0.8928), 75 + 20)\}$

$S^3_2 = \{(0.36, 65), (0.437, 80), (0.4464, 95)\}$

$S^3_3 = \{(0.75\ (0.72), 45 + 20 + 20), (0.75\ (0.864), 60 + 20 + 20),$

$(0.75\ (0.8928), 75 + 20 + 20)\}$

$S^3 = \{(0.54, 85), (0.648, 100), (0.6696, 115)\}$

$S^3 = \{(0.875\ (0.72), 45 + 20 + 20 + 20), (0.875\ (0.864), 60 + 20 + 20 + 20),$

$(0.875\ (0.8928), 75 + 20 + 20 + 20)\}$

$S^3 = \{(0.63, 105), (1.756, 120), (0.7812, 135)\}$

If cost exceeds 105, remove that tuples

$S^3 = \{(0.36, 65), (0.437, 80), (0.54, 85), (0.648, 100)\}$

The best design has a reliability of 0.648 and a cost of 100. Tracing back forthe solution through S_i's we can determine that $m_3 = 2$, $m_2 = 2$ and $m_1 = 1$