



# *DYNAMIC PROGRAMMING ALGORITHMS ASSIGNMENT*

GROUP MEMBERS:

- Pérez Piqueras, Víctor
- Martínez Riveiro, José Ramón

"As the previous experience was successful the same cargo company has been contracted to carry the remainder goods in the warehouse to the other location under the same payment conditions (10€/item). The CEO of the Cargo Co. wants, as usual, to maximize the gain (incomes-costs) therefore since the incomes are already known (#(items) x10) only the costs are in the scope to reach this goal. In this scenario, each truck driver has decided to minimize the non-used weight capacity in each truck with the aim of minimizing in the end the total amount of non-used weight capacity in all the trucks and so minimizing also the number of trucks required to do the whole transportation service. Although the drivers are wrong in their thesis, they are very determined to do in this way, so you are asked to provide a dynamic programming algorithm to choose goods to be carried in each truck according to driver's intentions for their own truck."

## ROUGH DESCRIPTION

Dynamic programming is an algorithmic technique which is based on dividing the main problem into subsets that are overlapping problems and solving them to get the final goal. The consecutive sub-solutions are constructed from the previously found ones. In this problem we divide the solution into subsets composed by the remaining weight of the truck depending on the goods that we carry. We start by computing the remaining weight considering the first good, and having this, we keep adding goods to our set and calculating the best remaining weights until we get to the last good. Then the problem is solved.

## FORMAL DESCRIPTION

- M[ i , j ]= Minimum number of non-used weight of the truck with capacity **"j"**, from a set of goods **"i"**.

- M[ i , j ]=

| | | |
|---|---|---|
| j | ⟶ | *(i* == 0)* |
| M[i-1 , j] | ⟶ | *s(i*)>j* |
| Min{ M[ i-1 , j ], M[ i-1 , j-s(i*) ] } | | *o.w.* |

- Order of candidates to conform the subsets:
  In this problem, there is no need to order the candidates, as our scope is to minimize the non-used capacity.

- Brief description of how does Bellman Optimality Principle apply:
  To reach the correct solution of the problem, each row of the matrix M[i , j] assumes the previous one as correct. The inclusion of a new candidate of the present subset is taken in order to optimize the function. Every subsolution of the problem is an optimal solution for its own subproblem. We calculate the non-used capacity for the first good, and then we keep checking if adding a new one makes it a better choice in order to get a lower non-used capacity or not. Every row is the optimum one for its own set of goods, and the solution is in the last row which includes all the goods given in the problem.

## PSEUDO-CODE

All data is introduced in the **main()** code, **dynamic()** function is used to calculate the matrix of non-used weight, and the **returning**() function shows the chosen packets.

```
main()
    int i,j,n,numPackets, truckTotal=400,sum=0;
    printf "Introduce the number of goods";
    read numPackets;
    numPackets++;
    int goods[numPackets];
    goods[0]=0;//empty

    for(j=1;j<numPackets;j++)
        print "Introduce the weight of the packet"+j;
        read i;
        goods[j]=i;
    end_for

    for(j=1;j<numPackets;j++)
        print "packet"+"j+"="goods[j]";
    end_for
```

```
    int M[numPackets][truckTotal+1];

    dynamic(goods,numPackets,truckTotal,M);
    returning(truckTotal,numPackets,goods,M);
end_main



dynamic(int goods[],int numPackets, int truckTotal, int M[][truckTotal+1]){
    int i=0,j;
    for(j=0;j<truckTotal+1;j++)
        M[i][j]=j;
    end_for
    for(i=1;i<numPackets;i++)
        for(j=0;j<truckTotal+1;j++)
            if(goods[i]>j)
                M[i][j]=M[i-1][j];
            end_if
            else
                if( M[i-1][j-goods[i]] < M[i-1][j])
                    M[i][j] = M[i-1][j-goods[i]];
                end_if
                else
                    M[i][j]=M[i-1][j];
                end_else
            end_else
        end_for
    end_for
end_dynamic

returning(int truckTotal, int numPackets, int goods[], int M[][truckTotal+1])
    int i,j,sum=0;
    j=truckTotal;
    print "Chosen goods for weight"+truckTotal;
    for(i=numPackets;i>=1;i--)
        if(M[i][j]<M[i-1][j])
            j=j-goods[i];
            if(goods[i]>0)
                print goods[i]);
                sum+=goods[i];
            end_if
        end_if
    end_for
    print "Weight left:" truckTotal-sum;
end_returning
```

## COMPUTATIONAL COST

- for j=1 to numPackets ←O(n)
- for j=0 to truckTotal ←O(W)
- for i=1 to numPackets ←O(n·W)
- for j=0 to truckTotal+1 ←O(W)
- for i=numPackets to 1 ←O(n)

Total: $O(n \cdot W) \mid \Omega(n \cdot W) \rightarrow \theta(n \cdot W)$