APPLICATION OF DYNAMIC PROGRAMMING TO THE ECONOMY

A. Jovliev, J. Tulkinov
National university of Uzbekistan named after Mirzo Ulugbek
jovliyevabdulbosit@gmail.com

It is no exaggeration to say that today programming and computer science have already penetrated into any field and have a strong place. Programming, which has penetrated into any area of the economy, and especially business, is still an important support for its development. While in the past the human brain took days to come up with a solution to this or that economic problem, today the computer is finding the optimal solution to such problems even in seconds.

First of all, let's speak about what dynamic programming is.

Dynamic programming is a process of solving a bigger problem by dividing into several smaller problems (‘from particular to general’ principle). In dynamic programming, we first consider the "backpack problem" algorithm. This algorithm works as follows:

- A thief broke into the store. Her bag may contain m kg, and there are n items in the store. The weight and price of these items are different. **Problem:** We need to find the most expensive combination of items that will fit in the bag.

If we approach the problem using the Greedy algorithm, we can say that the thief must take an expensive and baggy item, but this solution is not optimal. We need to look at the right solution in different combinations.

For example, a thief has a 4-kilogram bag and 3 items that he can take. They are: a laptop weighing 3 kg, and pricing 2000$, a computer weighing 4 kg and pricing 3000 $. Mini Macbook weighing 1kg and pricing 1500 $. We will consider combinations of these items. We put mark 1 to the item that the thief should take and a mark 0 to the item that the thief does not take.

Notebook	**Computer**	**Macbook**	**Total weight**	**Total profit ($)**
0	0	0	0 kg	0
0	0	1	1 kg	1500
0	1	0	4 kg	3000
0	1	1	5 kg	4500
1	0	0	3 kg	2000
1	0	1	4 kg	3500
1	1	0	7 kg	5000
1	1	1	8 kg	6500

Here we get their maximum profit value from rows weighing less than or equal to 4 kg, which means that when we get a Macbook with a laptop, we get the most expensive and optimal solution that fits in a bag. But we now have a relatively fast solution because of the small number of items in this algorithm, but as the number of items we have, the speed of taking that solution slows down. Because now we have to find 2^n combinations. Here n - is the number of items. So far we have seen only 8 cases 〖(2〗^3=8) for 3 different items.

In this case, it is very convenient to use the algorithm "knapsack problem". In this case, we divide the problem into the following problems. Let's say we have one Macbook in the store and now we have 4 bags with a capacity of 1, 2, 3, 4 kg respectively. If a thief steals with a 1kg bag, he takes away an item worth $ 1,500, and if he steals with a 2kg bag, he takes away an item worth $ 1,500 because there is no other item in the store. Even if it is stolen with another similar bag, it will take away the same amount of items, i.e. with a maximum of $ 1,500 worth of items (s).

Now, one more item has been added to the store: a computer. It weighs 4 kg, if a thief enters to the store with a bag of 1 kg to steal, then he has to steal a Macbook from the store, because another item in the bag of 1 kg doesn't fit. As a result, the thief leaves the store with a $ 1,500 item, and if the thief goes with the remaining 2 and 3 kg bags, he will make a profit of $ 1,500. A $ 3,000-worth item will be taken out, where the optimal solution was $ 3,000.

Now let's say a laptop is added to the store. Previously, the maximum profit for 3-kg-knapsack was $ 1,500, now it is $ 2,000, because a new item has been added to the store and it is more profitable for the thief to get it. Now the previous maximum profit for 4 kg was $ 3000, now if the thief takes the laptop and Macbook, he will take the item out of the store for $ 3500, for our case the 2nd conclusion is better than the first, now with 4kg-knapsack, the thief can take an item of maximum of $ 3,500.

In conclusion, we take the maximum values in each step and compare them with the maximum value in the next step, and we get the combination of which value is the largest. And now, we can create a table as follows.

	1-kg-knapsack	2-kg-knapsack	3-kg-knapsack	4-kg-knapsack
Macbook	1500 $	1500 $	1500 $	1500 $
Computer	1500 $	1500 $	1500 $	3000 $
Notebook	1500 $	1500 $	2000 $	3500 $

Nowadays, almost all problems are solved with the help of programming. We use the **Python** programming language to solve this problem, because it is the closest language to human understanding in programming languages and is widely used today for data analysis. We write a function with the following argument.

1- weight of the item in the bag (type int), 2- weight of the item (type list), 3- price of the item (type list), 4- number of items (type int), function name knapsack (W, wt, val, n).

```
    def knapSack(W, wt, val, n):
  K = [[0 for x in range(W + 1)] for x in range(n + 1)]
  # Build table K[][] in bottom up manner
  selected=[2 for i in range(n+1)]
  for i in range(n + 1):
    for w in range(W + 1):
      if i == 0 or w == 0:
        K[i][w] = 0
      elif wt[i - 1] <= w:
        K[i][w] = max(val[i - 1]
                  + K[i - 1][w - wt[i - 1]],
                  K[i - 1][w])
        selected[i] = 1
      else:
        K[i][w] = K[i - 1][w]
        selected[i] = 0
  tempW = W
  y = 0
  x=n
  for  i in range(x,0,-1):
    if (tempW - wt[i - 1] >= 0) and (K[i][tempW] - K[i - 1][tempW - wt[i - 1]]
== val[i - 1]):
      selected[y] = i-1
      y+=1
      tempW -= wt[i-1]
  for j in range(y-1,-1,-1):
    print("industry index" , (selected[j] ) ,'profit',val[selected[j]])
  return f"total profit: {K[n][W]}"
val = [1500 , 2000 , 3000]
wt = [1 , 3 , 4 ]
W = 4
```

```
n = 3
print(knapSack(W, wt, val, n))
# result
# industry index 0 profit 1500
# industry index 1 profit 2000
# total profit: 3500
```

Using the function above, we determined that the optimal solution was $ 3,500. So how do we use this algorithm in economics?

Suppose we produce m_1, m_2, m_3,……m_n units of products from n types of products with capital and labor resources, and let us make a profit of p_1 , p_2 , p_3 , … …p_n from them, respectively. Now the volume of these products should not exceed W and it is necessary to get the maximum benefit. Here we can use the knapsack algorithm.

So m_i is the weight of the item, p_i is the price of the item, W is the bag capacity, and n is the product type.

val=[〖 p〗_1 , p_2 , p_3 , … …p_n] , wt=[m_1 , m_2 , m_3,……m_n]

Given the above information, we can make a plan of how to carry out production with maximum profit for production.

Economic real-case problem

An investor has 150 million soums. He wants to invest his money in any type of business. Naturally, the investor wants to invest in the industry or sectors where he can make the most profit and invest as much of his money as possible. Because the more money you have under your pillow, the more it loses value. The faster it rotates, the faster it benefits.

So, the investor asked for advice from business consultants. The consultants suggested 12 different enterpreneurship:

1) Opening of a bakery (invests 25 million, income 20 million per month);

2) Opening a sewing workshop (invests 45 million, income 25 million per month);

3) Production of dishes (Invests 32 million, income 24 million per month);

4) Production of towel (Invests 24 million, income14 million per month);

5) Production of stationery (invests 35 million, income 30 million per month);

6) Opening a grocery store (invests 32 million, income 30 million per month);

7) Opening a fast food kitchen (invests 45 million, income 23 million per month);

8) Foam block production workshop (invests 30 million, income 25 million per month);

9) Car service (invests 60 million, income 34 million per month);

10) Purchase and rent of 2 cars (invests 120 million, income 10 million per month);

11) Air conditioning and camera installation services (invests 15 million, income 25 million per month);

12) Opening a recording studio (invests 140 million, income 40 million per month).

Respectively, the consultants calculated how much each of these areas would cost and how much the investor would earn. Using this, we create the following (taking into account that here, the investment is a one-time, the total investment required to start a business. The income is considered stable and is the average monthly income).

1- The amount of 150 million soums in the hands of the investor corresponds to the size of the bag in the example of the thief, which we first considered:

2- Let's add the income values to a single list type variable:

val = [20, 25, 24 , 14 , 30 ,30 , 23 , 25 , 34 , 10 ,25 ,40]

we can think of it as the price of a stolen item

3- Let's add the ivestment values to a single list of variables:

wt = [25, 45, 32 , 24 , 35 , 32 ,45 , 30 ,60 , 120 ,15 , 140]

we can compare this value with the weights of stolen items

We consider 12 types of business as the number of items, and we use the above function as follows:

knapSack(W, wt, val, n)

```
def knapSack(W, wt, val, n):
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]
    # Build table K[][] in bottom up manner
    selected=[2 for i in range(n+1)]
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i - 1] <= w:
                K[i][w] = max(val[i - 1]
                              + K[i - 1][w - wt[i - 1]],
```

```
                    K[i - 1][w])
            selected[i] = 1
        else:
            K[i][w] = K[i - 1][w]
            selected[i] = 0
    tempW = W
    y = 0
    x=n
    for  i in range(x,0,-1):
        if (tempW - wt[i - 1] >= 0) and (K[i][tempW] -     K[i - 1][tempW - wt[i -
1]] == val[i - 1]):
          selected[y] = i-1
           y+=1
           tempW -= wt[i-1]
    for j in range(y-1,-1,-1):
       print("industry index" ,(selected[j] ), 'profit',

         val[selected[j]])
    return f"total profit: {K[n][W]}"

val = [20, 25, 24 , 14 , 30 ,30 , 23 , 25 , 34 , 10 ,25 ,40]
wt = [25, 45, 32 , 24 , 35 , 32 ,45 , 30 ,60 , 120 ,15 , 140 ]
W = 150
n = 12
print(knapSack(W, wt, val, n))
```

The program gave us the following result:

industry index 2 profit 24
industry index 4 profit 30
industry index 5 profit 30
industry index 7 profit 25
industry index 10 profit 25
total profit: 134

This means that investing in areas 3, 5, 6, 8, 8, 11 (indexing in programming is added to the indexes starting from 0) brings the greatest profit, and its maximum profit is 134 million.

References

1. S. Bollapragada and M. Garbiras. Scheduling commercials on broadcast
television. Operations Research, 52(3):337–345, 2004.

2. K. M. Bretthauer and B. Shetty. The nonlinear knapsack problem–algorithms
and applications. European Journal of Operational Research, 138(3):459–472, 2002.

3. F. Della Croce, F. Salassa, and R. Scatamacchia. A new exact approach for the 0–1 collapsing knapsack problem. European Journal of Operational Research, 260(1):56–69, 2017.

4. F. D´ıaz-N´u˜nez, N. Halman, and O. C. V´ ´ asquez. The TV advertisements scheduling problem. Optimization Letters, 2018. doi:10.1007/s11590-018-1251-0.

5. C. D'Ambrosio, F. Furini, M. Monaci, and E. Traversi. On the product knapsack problem. Optimization Letters, 2018. doi:10.1007/s11590-017-1227-5.

6. A. Giudici, P. Halffmann, S. Ruzika, and C. Thielen. Approximation schemes for the parametric knapsack problem. Information Processing Letters, 120:11–15,

7. H. Kellerer, U. Pferschy, and D. Pisinger. Knapsack problems, 2004.

8. R. Lindsney and E. Verhoef. Traffic congestion and congestion pricing. In Handbook of transport systems and traffic control, pages 77–105. Emerald Group Publishing Limited, 2001.

10. D. Pisinger. The quadratic knapsack problem—a survey. Discrete applied mathematics, 155(5):623–648, 2007.

11. S. Pradhan. Retailing management: Text and cases. Tata McGraw-Hill Education, 2009.

12. G. Scheithauer. Knapsack problems. In Introduction to Cutting and Packing Optimization, pages 19–45. Springer, 2018.

13. J. P. Sousa and L. A. Wolsey. A time indexed formulation of non-preemptive single machine scheduling problems. Mathematical programming, 54(1): 353–367, 1992.

RESEARCH AND DEVELOPMENT OF A CONTROL SYSTEM FOR THE SYNCHRONIZED MOVEMENT OF THE HEAD OF AN ANTHROPOMORPHIC ROBOT

O.B. Hojiyev, Sh.B. Madaliev

Teacher of Joint Belarusian-Uzbek Intersectoral Institute of Applied Technical Qualifications in Tashkent

When considering the development trends of modern industrial robotics, special attention is drawn to a relatively new direction - collaborative assistant robots and telepresence systems that work in a common environment with a