Algorithmic Operation Research

Homework 6

Nikolaos Galanis - sdi1700019
Pantelis Papageorgiou - sdi1700115
Maria-Despoina Siampou - sdi1600151

January 24, 2020

Knapsack problem

Defintion

Given a set of items, each with a weight and a value, determine which items we have to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

The 0/1 notation means that we must either take, or leave back one item. Thus, the solution to consider taking a portion of each item is not accepted.

Knapsack problem

Naive solution

An easy Approach

A simple solution is to consider all subsets of items and calculate the total weight and value of all subsets. Consider the only subsets whose total weight is smaller than W. From all such subsets, pick the maximum value subset.

Knapsack problem

Using Dynamic Programming

To solve the problem usind DP, we must follow the following steps:

1. Find an optimal substructure
2. Recursively define the value of the optimal solution
3. Compute the optimal solution

Knapsack problem using DP

Definitions

We shall define the following before solving the problem:

- We are given a set $X = \{x_1, x_2, \ldots x_n\}$ of items.
- Each item has a weight w_i, and a profit p_i.
- The maximum weigth that we can carry is W.
- We want to chose a subset Y of X, s.t. $\sum_{x_i \in Y} p_i = max$ and $\sum_{x_i \in Y} w_i \leq W$

Knapsack problem using DP

Step 1: Optimal substructure

After considering the problem, we can easily conclude that the optimal substructure for the 0/1 problem is the following:

$$P_k(y) = \{max \sum_{j=1}^{k} p_j x_j, \sum_{j=1}^{k} w_j x_j \leq y, x_j = 0|1\}$$

where $0 \leq y \leq W$ and $1 \leq k \leq n$

Knapsack problem using DP

Step 2: Recursive definition

Considering $f_k(y)$ as the value of the optimal solution for the subproblem $P_k(y)$, then, we have :

$$f_{k+1}(y) = \begin{cases} f_k(y) & \text{if } w_{k+1} > y \\ max\{f_k(y), f_k(y - w_{k+1} + p_{k+1})\} & \text{else} \end{cases}$$

The optimal solution for the problem $P_n(W)$, is $f_n(W)$.

Knapsack problem using DP

Step 3: Optimal solution

Algorithm 1 Calculate the optimal solution

1: $n = length(P)$
2: **for** $y = 0$ **to** W **do**
3: **if** $w_1 > y$ **then**
4: $f_1(y) = 0$
5: **else**
6: $f_1(y) = p_1$
7: **end if**
8: **end for**

Knapsack problem using DP

Step 3: Optimal solution

1: **for** $k = 1$ **to** $n - 1$ **do**
2: **for** $y = 0$ **to** W **do**
3: **if** $w_{k+1} > y$ **then**
4: $f_{k+1y} = f_k(y)$
5: $x_{k+1}^{y} = 0$
6: **else if** $f_k(y) > f_k(y - w_{k+1} + p_{k+1}$ **then**
7: $f_{k+1}(y) = f_k(y)$
8: $x_{k+1} = 0$
9: **else**
10: $f_{k+1}(y) = f_k(y - w_{k+1} + p_{k+1}$
11: $x_{k+1} = 1$
12: **end if**
13: **end for**
14: **end for**

Knapsack problem using DP

Step 3: Optimal solution

1: **return** f, x

Now the vector x is filed with $0/1$ values, and we have:

$$x(n) = \begin{cases} 0 & \text{take n} \\ 1 & \text{leave n behind} \end{cases}$$

Knapsack Problem

Real world applications

Knapsack problem could be seen by someone as an another one algorithmic problem, interested enough for computer science enthusiasts but without a real world application. Fortunately, this is not the case! Some really cool real world applications are presented below:

- Internet Download Managers
- Resource allocation
- Traveling
- Exam preparation

Knapsack Problem - Real world applications

Internet Download Managers

First and foremost, the data is broken into chunks. The goal of an Internet Download Manager is to utilize the full size limit. As per the maximum size of data that can be retrieved in one go, the server uses this algorithm and packs the chunks so as to achieve its goal.

Knapsack Problem - Real world applications

Resource allocation

Let an operating system has n resources and there are m tasks with different requirement of the resource in a given moment. Then, Knapsack problem algorithm can help the system allocate that resources which achieve maximum efficiency.

Knapsack Problem - Real world applications

Traveling

Many of us, (especially women) have faced, at least once in their lifetime, difficulty in packing items and clothes when traveling to a place by plane. All airline companies define a max capacity of the bag you can take with you. Given a list of items to take which could all not fit into that single bag, the knapsack algorithm can be used to maximize the value of items according to your needs.

Knapsack Problem - Real world applications

Exam preparation

Probably, you are doing this from the beginning of your student years. We are home just a night before the exams. You know you haven't read enough in the whole semester. Now you are thinking one and only thing. Get more marks by less study in the remaining amount of hours. So, here we are trying to optimize our efforts for getting more marks.

Knapsack Problem - Real world applications

Exam preparation - cont'

Now, we analyze our previous question papers and decide that which chapters have more value and which chapters have less. We also have a sense that how much time it will take to complete the chapter by checking the number of pages of the chapter in the book.

Knapsack Problem - Real world applications

Exam preparation - cont'

Here we are trying to maximize the marks by selecting the chapters whose questions have a high probability of asking in the exams. And also we have to consider the hours required to complete these chapters. Now we check all the combinations of chapters and their values to find the list of chapters that needs to be studied and also, add the hours required to study these chapters so the added hours should not exceed the hours we have for the exam.

Subset The Problem

The Problem

- **Definition:** Given a set of non-negative integers, and a value sum, determine if there is a subset of the given set with sum equal to given sum.
- **Example:** Input: set[] = 3, 34, 4, 12, 5, 2, sum = 9 Output: True, there is a subset (4, 5) with sum 9.
- **Solution using DP:** Create a boolean 2D table named subset and fill it in bottom up manner. The value of subset[i][j] will be true if there is a subset of set[0..j-1] with sum equal to i., otherwise false. Return subset[sum][n]

Subset Sum Problem

DP Solution

Function Implementation

```
def isSubsetSum(set,n,sum):
    subset=([[False for i in range(sum+1)]
                    for i in range(n+1)])
    for i in range(n+1):
        subset[i][0] = True
        for i in range(1,sum+1):
            subset[0][i]=False
```

Subset Sum Problem

DP Solution

Function Implementation cont'd

```
for i in range(1,n+1):
    for j in range(1,sum+1):
        if j<set[i-1]:
            subset[i][j] = subset[i-1][j]
        if j>=set[i-1]:
            subset[i][j] = (subset[i-1][j]
            or
            subset[i - 1][j-set[i-1]])
return subset[n][sum]
```

Knapsack vs Subset Sum Problem

The Problem

Knapsack and Subset Sum are two closely related, well-known NP-complete problems.

There is a direct reduction from Subset Sum to Knapsack, and as we saw above, the dynamic programming solutionis for both problems were almost the same.

Generally, Subset Sum is considered as a special case of Knapsack, where $v_i = w_i$ for all i, that is, the values equal the weights.