1 Overview

The lecture discusses one of the classic approaches in computer science, Dynamics Programming. In this lecture, we go over couple of dynamic programming problems and develop an approximation algorithm for each which uses less resources in space and time. The two main dynamic programming problems that will be discussed are knapsack problem and bin-packing problem.

2 Dynamic Programming

Dynamic Programming is a technique of recursively solve overlapping sub-problems to obtain optimal value for the larger problem. For example, Knapsack problem is one of the classic problems which can be solved using dynamic programming.

2.1 Knapsack Problem

The traditional knapsack problem goes by the wordings: "Given a sack of size $B$, and $n$ objects each of size $w_i$ and profit $p_i$, then find the subset of objects which fits in the sack and maximizes the total profit." [Vaz01]

$$\Pi = \max_{(c_1,..,c_n)}(\sum c_i p_i)$$

$$s.t. \sum c_i w_i \leq B$$

The Hicksian knapsack problem is the inverted problem of the given problem. It asks ”what is the minimum $B$ if we need to have atleast $P^*$ profit in total?”

The dynamic program solution involves recursive algorithm filling ($profit$, #goods) matrix.[Vaz01]

$$b(i, p) = \begin{cases} 
\min((w_i + b(i-1, p-p_i)), w(i-1, p)) & \text{for } p_i \leq p \\
 w(i-1, p) & \text{for } p < p_i 
\end{cases}$$

2.1.1 Approximation by rounding

Knapsack is a pseudopolynomial time algorithm, so it can possbile have a PTAS.[Vaz01] Let $k = P\epsilon/n$, where $P = \max(profit)$. We round all profitvaues down to a multiple of $k$, so essentially we are forcing GCD to be $k$. Thus, we have decreased the number of row from old problem = $nP$ to the number of rows in new approximate problem = $n^2/\epsilon$. Running time for this procedure is $\text{poly}(n, 1/\epsilon)$.

What is the approximation factor?
ALG(rounded) = OPT(rounded).
Also ALG(original) ≥ ALG(rounded) and OPT(rounded) ≥ OPT(original) − εP
Thus, \(\text{ALG}(\text{original}) \geq (1 - \varepsilon)\text{OPT}(\text{original})\)

2.2 Bin Packing Problem

**Definition 1.** Given \(n\) items of sizes \(s_i \in [0, 1]\) and bins of size 1, we want to pack all the items in the bins such that we use minimum number of bins.[FdlVL81]

2.2.1 Algorithm 1: First Fit

**Algorithm 1** First Fit

1: function PACK ITEMS
2:     for \(i \leq n\) do
3:         for \(j \leq n_{\text{bins}}\) do
4:             if \(s_i \leq \text{bin}_j\) and \(\text{fit}_i \neq \text{True}\) then
5:                 \(\text{bin}_j \leftarrow \text{bin}_j - s_i\)
6:                 \(\text{fit}_i \leftarrow \text{True}\)
7:             if \(\text{fit}_i \neq \text{True}\) then
8:                 \(n_{\text{bins}} \leftarrow n_{\text{bins}} + 1\)
9:                 \(\text{bin}_{n_{\text{bins}}} \leftarrow 1 - s_i\)
10:                \(\text{fit}_i \leftarrow \text{True}\)
11:           \(i \leftarrow i + 1\)

**Lemma 1.** Every bin except one is at least half full

**Proof.** Assume there were two bins which were less than half full. If we were following First fit strategy we would not have started a new bin rather put the contents of both the bins into one bin. This means there cannot be two bins which were less than half full.

Based on Lemma 1, for First-fit algorithm: \(\text{ALG} \leq 2\text{OPT} + 1\).

2.2.2 Algorithm 2: Approximation Scheme

Algorithm[FdlVL81]:

**Algorithm 2** Approximate Bin Packing

1: function APPROXIMATE PACKING
2: Sort items in increasing order of their sizes
3: Remove items of size less than \(\varepsilon\)
4: Make blocks of \(\varepsilon^2 n\) items \(\implies \text{blocks} = \frac{1}{\varepsilon^2}\)
5: Round all items to max item size in block
6: Solve exactly for the rounded instance
7: Pack remaining small items using first-fit
Effectively, we are reducing the number of item sizes. So what is the advantage of having fewer item sizes?

**Lemma 2.** For a fixed \( \varepsilon \) and \( c \), take a bin-packing instance where \( \varepsilon \leq s_i \leq 1 \) and number of distinct \( s_i \) is equal to \( c \). There exist an exact bin-packing algorithm which runs in \( \text{poly}(n^{1/\varepsilon}) \)

**Proof.** Let \( b_i \) be a bin \( i \)’s vector of size \( c \) such that \( b_i[j] \) is number of elements of size \( s_j \) in \( \text{bin}_i \). As all the items are at least of size \( \varepsilon \), there can be at most \( 1/\varepsilon \) elements in any bin. The number of combinations \( (Q) = \binom{c}{1/\varepsilon} \). Define a vector \( q \) of all possible combination such that \( q_k \) is number of bins filled with combination \( k \). Let \( n_{i,j} \) be equal to number of size \( s_j \) objects in \( \text{pattern}_k \). There are at most \( n^Q \) possibilities of \( q_i \). Thus if we just search through all those possibilities to find optimal configuration, it will run in \( O(n^{1/\varepsilon}) \).

Thus this approximation algorithm run in polynomial time, \( O(n^{(1/\varepsilon)^2(1/\varepsilon)}) \)

**Lemma 3.** \( \text{ALG} \leq (1 + \varepsilon) \text{OPT} \)

**Proof.** \( \text{ALG} \text{(rounded – down)} \leq \text{OPT} \leq \text{ALG} \text{(rounded – up)} \leq \text{OPT} + \varepsilon^2 n \). Now for items of size \( \varepsilon \): \( \text{OPT} \geq \varepsilon n \). For small items: \( \text{ALG} \text{(small)} \leq (1 + \varepsilon) V_{\text{small}} \leq (1 + \varepsilon) \text{OPT}_{\text{small}} \) where \( V_{\text{small}} \) is volume of all small items.

**Theorem 4.** Approximating the Bin Packing problem to a factor better than \( \frac{3}{2} \) is NP-hard.[FdlVL81]

**Proof.** Let \( a_1, a_2, ..., a_n \) be an instance of subset-sum problem. Let us construct a bin-packing problem’s instance such that \( s_i = \frac{2a_i}{\Sigma_i} \). It can be noticed that \( \Sigma s_i = 2 \), thus we atleast need 2 bins to pack all objects. So if the subset-sum problem is a yes instance then \( \exists S \subset \{s_1, ..., s_n\} \) such that \( \Sigma_{s_i \in S} s_i = \Sigma_{s_i \not\in S} s_i \). This means we cannot fit every object in exactly 2 bins. So, if there exists a bin-packing solution such that \# bins < 3, then we can have at most 2 bins. This can only happen when the subset-sum problem is a yes instance. Now subset-sum problem in itself is a NP-Hard problem. As it is equivalent to solve a subset-sum problem for getting an approximation factor better than \( \frac{3}{2} \), it means approximating the Bin Packing problem to a factor better than \( \frac{3}{2} \) is NP-hard.

3 Summary

The lecture introduces the topic of dynamics programming and talks about two classic problems: knapsack problem and bin packing problem. For knapsack problem, both profit-maximizing and budget-minimizing cases are discussed with exact solutions first and then with approximate algorithm by rounding the profits as a multiple of some chosen \( k \). In the case of bin packing, two approximation algorithms discussed are 1) First-fit and 2) Approximation by rounding sizes.

References
