(7) Dynamic Programming

Knapsack Problem:
Given a set J of n items, knapsack of capacity $B \geq 0$ (B - integer). Each item has a value v_j and a weight w_j ($v_j, w_j \geq 0$, v_j, w_j integers).
The goal is: Choose a set $S \subseteq J$ of items of max. value whose total weight $\leq B$.

Observation:
Suppose S^* is an optimal solution to the above problem and $j \in S^*$. Notice that $S^* \setminus \{j\}$ must be an optimal solution to the subproblem with itemset $J \setminus \{j\}$ and capacity $B - w_j$.

Strategy:
Let $D(j, w)$ denote the subproblem with itemset $\{1, \ldots, j\}$ and capacity w. Our goal is to calculate $D(n, B)$ from the subproblems.

[Sol'n].
We first handle the base cases:

- $D(j, 0) = 0 \quad \forall j = 1, \ldots, n$
- $D(0, w) = 0 \quad \forall w = 1, \ldots, B$.

In the general case:

$$D(j, w) = \begin{cases} \max(\underbrace{v_j + D(j-1, w - w_j)}_{\text{use item } j}, \underbrace{D(j-1, w)}_{\text{Don't use item } j}) & ; w_j \leq w \\ D(j-1, w) & ; \text{otherwise} \end{cases}$$

This is known as the "DP recurrence"

Continued

Runtime Analysis:
Each $D(j, w)$ can be calculated in $O(1)$ time, and in total we have nB $D(j, w)$'s to calculate. This gives us a total runtime of $O(nB)$.

Note: Our DP recurrence gives us the optimal value. To obtain the optimal itemset we can trace backwards from $D(n, B)$ (giving the sequence of steps required to obtain the optimal value)

Theorem 7.1
Knapsack is NP-hard

$O(nB)$ is not polynomial in the size of the input since B requires $\log B$ bits (and this is exponential). We call this pseudopolynomial

$1||\sum w_j U_j$:

Recall $U_j = \begin{cases} 1 & ; \text{if } C_j > d_j \\ 0 & ; \text{otherwise.} \end{cases}$, so $1||\sum w_j U_j$ is the problem of minimizing total weight of late jobs. This problem is NP-hard. Further, we may assume $p_j \le d_j \ \forall j$.

Observation:
An optimal schedule for $1||\sum w_j U_j$ has the form:

| On-time jobs | LATE jobs |

Notice that all LATE jobs can be rearranged arbitrarily, and $\sum w_j U_j$ remains unchanged. Further, we can assume the on-time jobs are scheduled in increasing d_j order (by EDD rule).

We will show 2 solutions to $1||\sum w_j U_j$:
Order all jobs in $\uparrow d_j$ order, so $d_1 \le d_2 \le \dots \le d_n$.

Solution I:
We will define $P(j, w)$ to be the min. processing time of a set $S \subseteq \{1, \dots, j\}$ that can be completed on-time and whose total weight is $\ge w$.
Formally,

$$P(j, w) = \min \left\{ P(S) : \begin{array}{l} (1)\ S \subseteq \{1, \dots, j\} \\ (2)\ C_j \le d_j \quad \forall j \in S \\ (3)\ \sum_{k \in S} w_k \ge w \end{array} \right\}$$

and if $\nexists S$ satisfying (1)-(3), then $P(j, w) = \infty$

↳ con't

Solution I: (con't)

How does this recurrence work?

Base Case:

- $P(0,0) = 0$
- $P(0,w) = \infty \quad \forall w > 0$
- $P(j,0) = 0 \quad \forall j = 0,\dots,n.$

General Case:

1) $j \in$ optimal set S^* for $P(j,w)$:

Then, j must be scheduled last (by EDD rule). So,

$$P(S^*) = P(S^* \setminus \{j\}) + p_j \le d_j$$
$$= P(j-1, w - w_j)$$

So, $P(j,w) = p_j + P(j-1, w-w_j)$.

Note: Here P denotes both the recurrence, and $\sum_{j \in S} p_j$.

2) $j \notin S^*$:

Then, $P(j,w) = P(j-1,w)$

So, our DP recurrence is:

$$P(j,w) = \begin{cases} \min\{p_j + P(j-1, w-w_j),\ P(j-1,w)\} & ; \text{if } P(j-1, w-w_j) + p_j \le d_j \\ P(j-1,w) & ; \text{otherwise} \end{cases}$$

The running time for $P(\cdot,\cdot)$ is then $O(n \sum_{j=1}^{n} w_j)$.

Note:

We can reduce the running time slightly, since if $P(j,w') = \infty$, then $P(j,w^*) = \infty$ for all $w^* \ge w'$. So, we can stop at the smallest w s.t. $P(j,w) = \infty$. This is precisely OPT, and so we get $O(n(\sum_{j=1}^{n} w_j - OPT))$.

Solution 2:

We modify our recurrence slightly.

Define:

$$Q(j,w) = \min \left\{ P(S) : \begin{array}{l} (1)\ S \subseteq \{1,\dots,j\} \\ (2)\ \text{All jobs in } S \text{ can be scheduled on time.} \\ (3)\ \sum_{k \in S} w_k \geq \sum_{k=1}^{j} w_k - w \end{array} \right\}.$$

Notice that (3) is the total weight of jobs from $\{1,\dots,j\} \setminus S$.

$Q(j,w) = P(j, \sum_{k=1}^{j} w_k - w)$, so this is the complement problem.

Then, our recurrence is:

$$Q(j,w) = \begin{cases} \min\{Q(j-1,w)+P_j,\ Q(j-1,w-w_j)\} & ; Q(j-1,w)+P_j \leq d_j \\ Q(j-1,w-w_j) & ; \text{otherwise} \end{cases}$$

The running time of $Q(\cdot,\cdot)$ is $O(n \cdot OPT)$, since we can stop once $Q(n,w) < \infty$. So, $Q(\cdot,\cdot)$ is more useful if OPT is small ~~[illegible]~~, and $P(\cdot,\cdot)$ is more useful if OPT is large.

Approximation Schemes

Hilroy

Approximation Schemes:

Dynamic algorithms are useful, but aren't efficient since they are pseudopolynomial. We want to come up with efficient approximation algorithms using the exact ans.

Def'n 7.1 (Polytime approximation scheme - PTAS)

Let X be a minimization problem. An algorithm A for X is called a polytime approximation scheme for X if for every instance I and every fixed $\varepsilon_0 > 0$, A returns a solution of:

$$\text{obj. value} \leq (1+\varepsilon_0) \cdot OPT(I)$$

in time: $f(size(I), \varepsilon_0) = poly(size(I))$

Note:

$f(\cdot, \varepsilon)$ can have an arbitrary dependence on ε. So, the following are valid:

(a) $n^{1/\varepsilon}$, (b) $\left(\frac{1}{\varepsilon}\right)^{1/\varepsilon^2} n^{2 \cdot 1/\varepsilon}$

(c) $2^{1/\varepsilon} n$ (d) $\left(\frac{1}{\varepsilon}\right)^2 n^2 \log n$

The important point is that ε is fixed.

Def'n 7.2 (Fully polytime approx. scheme - FPTAS)

A fully polytime approx. scheme is an algorithm A, where

- A is a PTAS
- f satisfies:

$$f(size(I), \varepsilon) = poly(size(I), 1/\varepsilon)$$

Scanned with CamScanner

$1||\sum w_j U_j$:

We want to come up with an FPTAS for $1||\sum w_j U_j$. We will need the following building blocks:

(1) An <u>exact</u> algorithm with running time $O(n \cdot OPT)$, or, more generally, $O(\text{poly}(n) \times \text{poly}(OPT))$. (We can get this from $Q(\cdot,\cdot)$)

(2) A lower bound on OPT s.t.

$$LB \leq OPT \leq \text{poly}(n) \cdot LB.$$

(This needs to be efficiently calculated).

We have (1), so let's work on (2). How can we obtain LB?

Notice that for any schedule S:

$$\max_j w_j U_j^S \leq \sum_j w_j U_j^S \leq n \cdot \max_j w_j U_j^S$$

So,

$$\min_{\text{schedule } S} \max_j w_j U_j^S \leq OPT \leq n \cdot \min_S \max_j w_j U_j^S$$

And we can solve $1||\max_j w_j U_j$ efficiently using the LCL rule.

Hence $LB = OPT_{1||\max_j w_j U_j}$

Algorithm:

- Choose suitable $\Delta > 0$
- Set $w'_j = \lfloor w_j / \Delta \rfloor \ \forall j$ (then $\frac{w_j}{\Delta} - 1 \leq w'_j \leq \frac{w_j}{\Delta}$)
- Run $Q(\cdot,\cdot)$ DP on the $\{w'_j\}$-instance to get a schedule, and a set A' of late jobs, where $w'(A') = \sum_{j \in A'} w'_j = OPT'$

$\hookrightarrow$ Con't

Hilroy

Claim 7.2:
The previous algorithm is an FPTAS for $1||\sum w_j U_j$.

[Proof]

To show this, we must show:

(1) $w(A') \leq (1+\varepsilon)\cdot OPT$, and

(2) $O(n \cdot OPT')$ is in $poly(n, 1/\varepsilon)$

Let A^*: Set of late jobs in optimal schedule with $\{w_j\}$-weights

Note that:

$$OPT' \leq w'(A^*) \leq \frac{w(A^*)}{\Delta} \leq \frac{OPT}{\Delta}$$

(Since each $w'_j \leq \frac{w_j}{\Delta}$)

which also shows that:

$$O(n \cdot OPT') = O(n \cdot \frac{OPT}{\Delta})$$

We can bound $w(A')$ in terms of OPT and Δ to help us show (1).

$$w(A') = \sum_{j\in A'} w_j \leq \sum_{j\in A'} (w'_j + 1)\Delta \quad \left(\text{Since } \frac{w_j}{\Delta} - 1 \leq w'_j\right)$$

$\leq w'(A') + n\Delta$ (Since there are at most n jobs)

$\leq w'(A^*) + n\Delta$ (Since A' is optimal for $\{w'_j\}$-weights)

$\leq OPT + n\Delta$ (By above)

Con't

[Proof] (con't)

Now, we choose, $\Delta = \frac{\varepsilon \cdot LB}{n}$,
and using that:

$$LB \leq OPT \leq n \cdot LB$$

We have:

(1) $w(A') \leq OPT + n\Delta \leq OPT + \varepsilon \cdot LB \leq (1+\varepsilon) OPT$

(2)

$$O\left(n \cdot \frac{OPT}{\Delta}\right) = O\left(n \cdot \frac{OPT}{\frac{\varepsilon LB}{n}}\right) = O\left(\frac{n^2}{\varepsilon} \cdot \frac{OPT}{LB}\right)$$

$$= O\left(\frac{n^3}{\varepsilon}\right)$$

So, we have an FPTAS. $\square$.