Dynamic Programming II

The (General) Knapsack Problem

We generalize the Simple Knapsack Problem (SKS) to the Knapsack Problem by assigning a positive integer value v_i to each weight w_i.

Input: $d, w_1, ..., w_d, v_1, ..., v_d, C \in \mathbb{N}$ (where C is the capacity)

Problem: Find $\max_S\{V(S)|K(S) \leq C\}$, where $S \subseteq \{1, ..., d\}$, $K(S) = \sum_{i \in S} w_i$, and $V(S) = \sum_{i \in S} v_i$.

Thus $V(S)$ is the total value of the set S of weights. The goal is to maximize $V(S)$, subject to the contraint that $K(S)$ (the sum of the weights in S) is at most C.

The Simple Knapsack Problem is the special case in which $v_i = w_i, 1 \leq i \leq d$.

To solve the general Knapsack problem, we start by computing the same Boolean array $R(i, j)$ that was used to solve SKS. Thus $R(i, j)$ ignores the values v_i, and only depends on the weights w_i. Next we define another array $V(i, j)$ that depends on the values v_i as follows: For $0 \leq i \leq d$, $0 \leq j \leq C$

$$V(i, j) = \max\{V(S)|S \subseteq \{1, ..., i\} \text{ and } K(S) = j\}$$

Exercise: Give a recurrence for computing the array $V(i, j)$, using the Boolean array $R(i, j)$. (You may assume that the array $R(i, j)$ has already been computed.) Also, give an algorithm for computing $V(i, j)$.

Remark: It turns out that if the above definition of $V(i, j)$ is changed so that we only require $K(S) \leq j$ instead of $K(S) = j$, then the Boolean array $R(i, j)$ is not needed in the recurrence.

The Activity Selection Problem

An *activity* i has a fixed start time s_i, finish time f_i, and profit p_i. Given a set of activities, we want to select a subset of non-overlapping activities with maximum total profit. (An example is a set of lectures with fixed start and finish times that need to be scheduled in a single class room.

Input: A list of activities (s_1, f_1, p_1),..., (s_n, f_n, p_n). Assume $p_i > 0$, $s_i < f_i$, and $s_i, f_i, p_i \in \mathbb{R}$ where $1 \leq i \leq n$.

Problem: Find a set $S \subseteq \{1, ..., n\}$ of selected activities such that no two selected activities overlap, and the profit $P(S) = \sum_{i \in S} p_i$ is as large as possible.

Greedy methods do not work for this problem. We will describe a Dynamic Programming solution.

1.) Define an array (of subproblems):

Sort the activities by their finish time: $f_1 \leq f_2 \leq ... \leq f_n$. It is possible that two or more activities have the same finish time. Partition the activities according to their finish times. (Remark: This partitioning is not strictly necessary, although it may lead to a more efficient algorithm in some cases.) Denote these *distinct* finish times by $u_1 < u_2 < ... < u_k$. For instance, if we have activities finishing at times 1.24, 4, 3.77, 1.24, 5 and 3.77, then we partition them into four groups: activities finishing at 1.24, 3.77, 4 and 5 respectively. In this case $u_1 = 1.24$, $u_2 = 3.77$, $u_3 = 4$, $u_4 = 5$. Let u_0 be $\min_{1 \leq i \leq n} s_i$, i.e., the earliest start time. Thus, $u_0 < u_1 < u_2 < ... < u_k$. Define an array $A(0..k)$ as follows.

$$A(j) = \max_{S \subseteq \{1,...,n\}} \{P(S) | S \text{ is feasible and } f_i \leq u_j \text{ for each } i \in S\}.$$

where S is *feasible* if no two activities in S overlap.

Note that $A(k)$ is the maximum possible profit for all feasible schedules S.

In order to give a recurrence for computing $A(0...k)$ we define an array $H(1..n)$ such that $H(i)$ is the index of the largest distinct finish time no greater than the start time of activity i. Formally, $H(i) = l$ if l is the largest number such that $u_l \leq s_i$. To compute $H(i)$, we need to search the list of distinct finish times. To do it efficiently, for each i, apply the binary search procedure that runs in logarithmic time in the length of the list of distinct finish times (try $l = \lfloor \frac{k}{2} \rfloor$ first). Since the length k of the list of distinct finish times is at most n, and we need to apply binary search for each element of the array $H(1..n)$, the time required to compute all entries of the array is $O(n \log n)$.

2.) Define a recurrence for $A(0..k)$.

Initialization: $A(0) = 0$.

Want $A(j)$ given $A(0), ..., A(j-1)$. Consider $u_0 < u_1 < u_2 < ... < u_{j-1} < u_j$. Can we beat profit $A(j-1)$ by scheduling some activity that finishes at time u_j? Try all activities that finish at this time and compute maximum profit in each case. We obtain the following recurrence:

$$A(j) = \max\{A(j-1), \max_{1 \leq i \leq n} \{p_i + A(H(i)) \mid f_i = u_j\}\}$$

where $H(i)$ is the last finish time before the start of activity i.

Example:

Activity i:	1	2	3	4
Start s_i:	0	2	3	2
Finish f_i:	3	6	6	10
Profit p_i:	20	30	20	30
$H(i)$:	0	0	1	0

j:	0	1	2	3
u_j:	0	3	6	10
$A(j)$:	0	20	40	40

$$A(2) = \max\{20, 30 + A(0), 20 + A(1)\} = 40$$
$$A(3) = \max\{40, 30 + A(0)\} = 40$$

The answer is $\max P = A(k) = 40$.

Exercise: Given that A has been computed, how do you find a set of activities S such that $P(S) = A(k)$?

Big hint: If $A(k) = A(k-1)$, then we know that no selected activity finishes at time u_k, so we go on to consider $A(k-1)$. If $A(k) > A(k-1)$, then some selected activity finishes at time u_k. How do we find this activity? We must find an i_0 such that $p_{i_0} + A(H(i_0)) = A(k)$ and $f_{i_0} = u_k$, output i_0, and go on to consider $A(H(i_0))$.

3.) Exercise: write an algorithm.

Job Scheduling with Profits

A job differs from an activity in that a job can be scheduled any time as long as it finishes by its deadline, but an activity has a fixed start time and finish time. Because of the flexibility in scheduling jobs, it is harder to find an optimal schedule for jobs than to select an optimal subset for activities.

Before we considered job scheduling problems for the case in which each job takes unit time. We now generalize this to the case in which each of n jobs i has duration d_i, deadline t_i, and profit p_i. We assume that d_i and t_i are positive integers, but the profit p_i can be a positive real number.

The goal is to find a feasible schedule $C(1...n)$ for the n jobs for which the profit $P(C)$ is maximized.

Our convention is that job i is scheduled to begin at time $C(i) \geq 0$, except that $C(i) = -1$ if job i is not scheduled by C.

The profit is:

$$P(C) = \sum_{C(i)\geq 0} p_i$$

We say that the schedule $C(1...n)$ is *feasible* if the following two conditions hold:

1. If $C(i) \geq 0$, then $C(i) + d_i \leq t_i$ (each scheduled job finishes by its deadline) and,
2. If $i \neq j$ and $C(i) \geq 0$ and $C(j) \geq 0$ then either $C(i) + d_i \leq C(j)$ or $C(j) + d_j \leq C(i)$ (no two scheduled jobs overlap).

Now we can define the problem Job Scheduling with Profits:

Input: A list of jobs (d_1, t_1, p_1),..., (d_n, t_n, p_n)

Problem: A feasible schedule $C(1), ..., C(n)$ such that the profit $P(C)$ is the maximum possible among feasible schedules.

Aside: Notice that this problem is at least as hard as SKS. In fact an SKS instance $w_1, ..., w_n, C$ can be viewed as a job scheduling problem in which each duration $d_i = w_i$, each deadline $t_i = C$, and each profit $p_i = w_i$. Then the maximum profit of any schedule is the same as the maximum weight that can be put into the knapsack. This is our first example of a *reduction*; we will study them in detail in the next part of the course.

To give a Dynamic Programming solution to the job scheduling problem, we start by sorting the jobs according to deadline. Thus we assume that

$$t_1 \leq t_2 \leq ... \leq t_n$$

It turns out that to define a suitable array A for solving the problem, we must consider all possible integer times t, $0 \leq t \leq t_n$ as a deadline for the first i jobs. It is not enough to only consider the specified deadline t_i given in the problem input.

Thus define the array $A(i, t)$ for $0 \leq i \leq n, 0 \leq t \leq t_n$ by:

$$A(i,t) = \max\{P(C) | C \text{is a feasible schedule in which only jobs in } \{1, ..., i\} \text{ are scheduled, and all scheduled jobs finish by time } t\}$$

Exercise: Find a recurrence for $A(i, t)$.

Hint: Consider the two cases that either job i occurs or does not occur in the optimal schedule (and note that job i will not occur in the optimal schedule if $d_i > \min\{t_i, t\}$). If job i does not occur, we already know the optimal profit. If job i occurs in an optimal schedule, then we may as well assume that it is the last job (among jobs $\{1, \ldots, i\}$) to be scheduled, because it has the lastest deadline. Hence we assume that job i is scheduled as late as possible, so that it finishes either at time t, or at time t_i, which ever is smaller, i.e. it finishes at time $t_{\min} = \min\{t_i, t\}$.