Greedy Method and Dynamic Programming

Instructor: Dr Om Prakash

IIITDM Kancheepuram

Greedy Method—Formal Setup

- An optimization problem P is a tuple:

$$P = (I, \mathcal{F}, f, opt)$$

Where,

- I is the set of all instances of the problem
- For each instance $x \in I$, $\mathcal{F}(x)$ is the set of all feasible solutions for x .
- $f : \bigcup_{x \in I} \mathcal{F}(x) \to \mathbb{R}$ is the objective function that assigns a value to each feasible solution.
- $opt \in \{min, max\}$ specifies whether the problem is **minimization** or **maximize**

Greedy Method—Formal Setup

- The goal is to find, for a given instance $x \in I$, a feasible solution

$$S^* \in \mathcal{F}(x)$$

Such that,

$$f(S^*) = opt\{f(S) \mid S \in \mathcal{F}(x)\}$$

Greedy Algorithm

- A **greedy algorithm** for an optimization problem P is an algorithms that constructs a solution incrementally as follows:

1. Start with the empty solution $S_0 = \phi$.
2. At each iteration t, given a partial solution S_t, choose an element $c_t \in \mathcal{F}(x)$

That is feasible to add and maximizes or (minimizes) a **local selection criterion** $g(c_t, S_t)$.

3. Extend the solution: $S_{t+1} = S_t \cup \{c_t\}$
4. Continue until a complete feasible solution is obtained.

The algorithm never revises earlier choices (irrevocability)

Greedy Algorithm– Correctness Condition

- A greedy algorithm yields an optimal solution if:

1. Optimal Substructure: An optimal solution to the problem can be composed from optimal solution to its subproblems.

2. Greedy-Choice Property: A global optimum can be obtained by choosing a local optimum at each step.

Coin Change Problem

- A child buys candy valued at less than $1 and gives a $1 bill to the cashier.

- The cashier wishes to return change using the fewest number of coins.

- Assume that an unlimited supply of quarters, dimes, nickels, and pennies is available.

Coin Change Problem

- Example: Suppose the worth of candy was 33 cents.
- The solution to this problem is (2 quarters, 1 dime, 1 nickel and 2 pennies).

- Show that the greedy algorithm for the coin-change problem generates change with the fewest number of coins when the cashier has an unlimited supply of quarters, dimes, nickels, and pennies.

Coin Change Problem

- Canonical system= {quarter, dime, nickel, penny}
- Non-canonical system = {1,3,4} check for 6!

- Prove the greedy choice property using contradiction.
- Let $(x_1, x_2, \ldots, x_n)$ be the greedy solution and $(y_1, y_2, \ldots, y_n)$

Be the optimal solution.

Coin Change Problem--Correctness

- Clearly, if the optimal solution is different from the greedy solution
- Choose the least index i such that $\exists i, x_i \neq y_i$
- Observation: $x_i > y_i$
- Construct a new feasible solution y' from y by incrementing y_i to x_i decrementing the added value by removing coins of least denominations than coin c_i

Coin Change Problem--Correctness

- Thus, we obtained a new feasible solution whose number of coins is less than the optimal solution.
- This is a contradiction and hence our assumption was false and the greedy solution is in-fact the optimal solution.

Container Loading Problem

- A large ship is to be loaded with cargo. The cargo is containerized, and all containers are the same size.
- Different containers may have different weights.

- Let w_i be the weight of the $i\mathrm{th}$ container $1 \leq i \leq n$.
- The cargo capacity of the ship is c .
- We wish to load the ship with the maximum number of containers.

Container Loading Problem

- Let x_i be a variable whose value can be either 0 or 1. If we set x_i to 0, then container i is not to be loaded. If x_i is 1 then the container is to be loaded.

$$\sum_{i=1}^{n} w_i x_i \leq c \qquad x_i \in \{0,1\} \; 1 \leq i \leq n$$

- The optimization function is $\sum_{i=1}^{n} x_i$
- Every set of x_i 's that satisfy the constraints is a feasible solution.
- Every feasible solution that maximizes $\sum_{i=1}^{n} x_i$ is an optimal solution.

Container Loading Problem—Greedy Strategy

- *From the remaining containers, select the one with least weight*.

Container Loading Problem

- Example: Suppose that

$n = 8, [w_1, \ldots, w_8] = [100, 200, 50, 90, 150, 50, 20, 80]$, and $c = 400$

When the greedy algorithm is used, the containers are considered for loading in the order $7, 3, 6, 8, 4, 1, 5, 2$ and the greedy solution we have $[x_1, \ldots, x_8] = [1, 0, 1, 1, 0, 1, 1, 1]$ and $\sum x_i = 6$

Container Loading—Greedy Algorithm

```
ContainerLoading(c, capacity, numberOfContainers, x)
Sort(c, numberOfContainers)
n := numberOfContainers
for i = 1 to n
        do
            x[i] = 0
i = 1
while (i ≤ n and c[i] · weight ≤ capacity)
        do
            x[c[i], id] = 1
            capacity− = c[i] · weight
            i + +
```

Container Loading—Proof of Correctness

- Theorem 4.1: The greedy algorithm generates optimal loadings.
- Proof: Let $x = [x_1, x_2, \ldots, x_n]$ be the solution produced by the greedy algorithm and let $y = [y_1, y_2, \ldots, y_n]$ be any feasible solution.

We will show that $\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n} y_i$. Without loss of generality we may assume that the containers have been ordered so that $w_i \leq w_{i+1}$, $1 \leq i < n$

- From the way greedy algorithm works $\exists k, 0 \leq k \leq n$ such that

$x_i = 1, i \leq k \text{ and } x_i = 0, i > k$

Container Loading—Proof of Correctness

- The proof is by induction on the number p of positions i such that $x_i \neq y_j$.
- Base Case: $p = 0$ and so x and y are the same. So, $\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n} y_i$
- Induction Hypothesis: Let m be the arbitrary natural number

$$\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n} y_i \text{ whenever } p \leq m$$

Container Loading—Proof of Correctness

- Induction Step: We show that $\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n} y_i \text{ when } p = m + 1$
- Find the least integer j, $1 \leq j \leq n$ such that $x_j \neq y_j$.
- Since, $p \neq 0$ such a j exists.
- Also, $j \leq k$, as otherwise y is not a feasible solution. Since, $x_j \neq y_j$ and $x_j = 1, y_j = 0$. Set y_j to 1.

Container Loading—Proof of Correctness

- If the resulting y is a feasible solution, let z denote the resulting y
- If the resulting y denotes a infeasible solution, there must be ℓ in the range $[k+1, n]$ for which $y_\ell = 1$. Set $y_\ell = 0$.
- Let z denote the resulting y. As $w_j \leq w_l$, z is a feasible solution.
- In either case, $\sum_{i=1}^{n} z_i \geq \sum_{i=1}^{n} y_i$ and z differs from x in at most $p-1=m$ positions. From the Induction Hypothesis it follows that $\sum_{i=1}^{n} x_i \geq \sum_{i=1}^{n} z_i \geq \sum_{i=1}^{n} y_i$

Knapsack Problem

- Given n objects and a knapsack or bag.
- Object i has a weight w_i and the knapsack has a capacity m .
- If a fraction $x_i, 0 \leq x_i \leq 1$ of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned.

- The objective is to obtain a filling of the knapsack that maximizes the total profit earned.

Knapsack Problem

$$\text{maximize} \quad \sum_{1 \leq i \leq n} p_i x_i$$

$$\text{subject to} \quad \sum_{1 \leq i \leq n} w_i x_i \leq m$$

$$\text{and } 0 \leq x_i \leq 1, 1 \leq i \leq n$$

Knapsack Problem

Example: Consider the following instance of the knapsack problem:

$n = 3, m = 20,\ (p_1, p_2, p_3) = (25, 24, 15)$ and

$(w_1, w_2, w_3) = (18, 15, 10)$. Four feasible solutions are:

(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
$(1/2, 1/3, 1/4)$	16.5	24.25
$(1, 2/15, 0)$	20	28.2
$(0, 2/3, 1)$	20	31
$(0, 1, 1/2)$	20	31.5

Knapsack Problem

- **Lemma**: In case the sum of all the weights is $\leq m$, then $x_i = 1, 1 \leq i \leq n$ is an optimal solution.

- **Lemma:** All optimal solutions will fill the knapsack exactly.

Knapsack Greedy Algorithm

```
GreedyKnapsack(m, n)
    //p[1 : n] and w[1 : n] contain the profits and weights respectively
    // of the n objects ordered such that p[i]/w[i] >= p[i+1]/w[i+1]
    // m is the knapsack size and x[1 : n] is the solution vector...
    for i = 1 to n
            do
                x[i] = 0.0
    U := m;
    for i = 1 to n
            do
                if (w[i] > U)
                    then break;
                        x[i] := 1.0; U := U - w[i];

    if (i <= n)
        then x[i] := U/w[i]
```

Knapsack Greedy Algorithm--Correctness

- **Theorem:** If $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \frac{p_n}{w_n}$ then GreedyKnapsack generates an optimal solution to the given instance of the knapsack problem.

- **Proof of Correctness:** Let $x = (x_1, x_2, \ldots, x_n)$ be the solution generated by GreedyKnapsack. If all the x_i equal one, then clearly the solution is optimal.
- Let j be the least index such that $x_j \neq 1$. From the algorithm it follows that

$x_i = 1$ for $1 \leq i < j$ $\quad x_i = 0$ for $j < i \leq n$ and $0 \leq x_j < 1$

Knapsack Greedy Algorithm--Correctness

- Let $y = (y_1, y_2, \ldots, y_n)$ be an optimal solution. From earlier Lemma, we can assume that $\sum w_i y_i = m$
- Let k be the least index such that $y_k \neq x_k$. Clearly, such a k must exist. It also follows that $y_k < x_k$.

Knapsack Greedy Algorithm--Correctness

- Consider three possiblilities:

  $k < j, k = j, \text{ or } k > j$

- 1. If $k < j$, then $x_k = 1$. But, $y_k \neq x_k$, and so $y_k < x_k$
- 2. If $k = j$, then since $\sum w_i x_i = m$ and $y_i = x_i$ for $1 \leq i < j$

  it follows that either $y_k < x_k$ or $\sum w_i y_i > m$

3. If $k > j$, then $\sum w_i y_i > m$, and this is not possible.

Knapsack Greedy Algorithm--Correctness

- Now suppose we increase y_k to x_k and decrease as many of $(y_{k+1}, \ldots, y_n)$ as necessary so that the total capacity used is still m
- This results in a new solution $z = (z_1, \ldots, z_n)$ with $z_i = x_i, 1 \leq i \leq k$ and $\sum_{k<i\leq n} w_i(y_i - z_i) = w_k(z_k - y_k)$

Knapsack Greedy Algorithm--Correctness

- Then, for z we have,

$$
\begin{aligned}
\sum_{1\le i\le n} p_i z_i &= \sum_{1\le i\le n} p_i y_i + (z_k - y_k) w_k \frac{p_k}{w_k} - \sum_{k<i\le n} (y_i - z_i) w_i \frac{p_i}{w_i} \\
&\ge \sum_{1\le i\le n} p_i y_i + \left[(z_k - y_k) w_k - \sum_{k<i\le n} (y_i - z_i) w_i \right] \frac{p_k}{w_k} \\
&= \sum_{1\le i\le n} p_i y_i
\end{aligned}
$$

Knapsack Greedy Algorithm--Correctness

- If $\sum p_i z_i > \sum p_i y_i$ then y could not have an optimal solution.

If these sums are equal, then either $z = x$ and x is optimal, or $z \neq x$

- In the latter case, repeated use of the above argument will either show that y is not optimal, or transform y into x and thus show that x too is optimal.

Job Scheduling without Deadline

- You are given n tasks and an infinite supply of machines on which those tasks can be performed.

- Each task has a start time s_i and a finish time f_i , $s_i < f_i$, $[s_i, f_i]$ is the processing interval for task i .

- Two tasks i and j overlap iff their processing intervals overlap at a

Point other than the interval start or end.

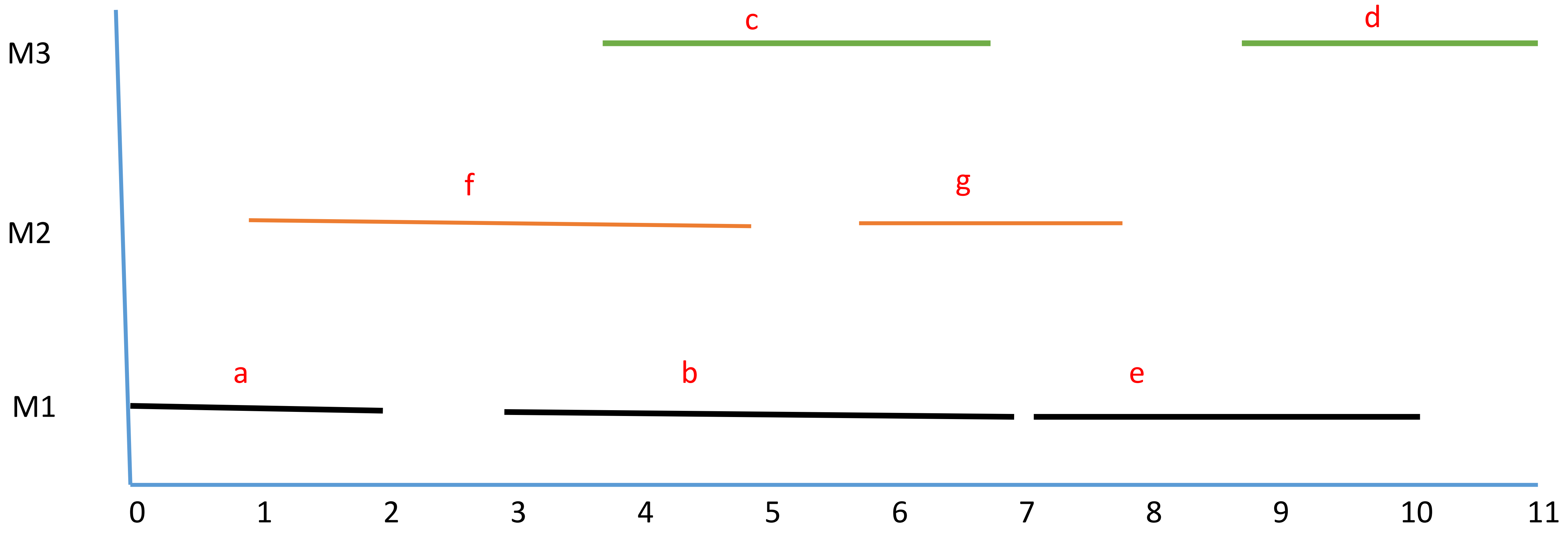
Job Scheduling without Deadline

- A feasible task-to-machine assignment is an assignment in which no machine is assigned two overlapping tasks.

- An optimal assignment is a feasible assignment that utilizes the fewest number of machines.

Job Scheduling without Deadline

task	a	b	c	d	e	f	g
start	0	3	4	9	7	1	6
finish	2	7	7	11	10	5	8



Job Scheduling without Deadline

- Greedy Choice: *If an old machine becomes available by the start time of the task to be assigned, assign the task to this machine; if not, assign it to a new machine.*

Job Scheduling without Deadline

GreedyScheduling(m)

$//p[1:m]$ sorted in non-decreasing order of start time such that $s[j] \leq s[j+1], 1 \leq j \leq m$
//Assume an infinite supply of machines $M[1...\infty]$
Schedule $P[1]$ on $M[1]$
$M[1].availaible = f[1]$
$i = 2, j = 1$
while $P[m]$ is not scheduled
    do
        if $\exists 1 \leq k \leq j \ M[k].availaible \leq s[i]$
            then Schedule $P[i]$ on $M[k]$;
                $i = i + 1$
            else
                j=j+1
                Schedule $P[i]$ on $M[j]$
                $M[j].available = f[i]$
                $i = i + 1$

Job Scheduling without Deadline--Correctness

- Proof by Induction on number of machines
- Base Case: Prove that the largest number of tasks that can be scheduled on the machine is given by the following greedy algorithm:

From the Remaining tasks, select the one that has the least finish time and does not overlap with any of the already selected tasks.

Job Scheduling with Deadline

- We are given a set of n jobs. Associated with job i is an integer deadline $d_i \geq 0$ and a profit $p_i > 0$.

- For any job i the profit p_i is earned iff the job is completed by its deadline.

- To complete a job, one has to process the job on a machine for one unit of time.
- Only one machine is available for processing jobs.

Job Scheduling with Deadline

- Example: Let $n = 4$, $(p_1, p_2, p_3, p_4) = (100, 10, 15, 27)$ and $(d_1, d_2, d_3, d_4) = (2, 1, 2, 1)$. The feasible solutions and their values are:

	Feasible Solution	Processing Sequence	Value
1	(1, 2)	2,1	110
2	(1, 3)	1,3 or 3,1	115
3	(1, 4)	4, 1	127
4	(2, 3)	2, 3	25
5	(3, 4)	4, 3	42
6	(1)	1	100
7	(2)	2	10
8	(3)	3	15
9	(4)	4	27

Job Scheduling with Deadline

- Greedy Choice Property: Choose $\sum_{i \in J} p_i$ as our optimization measure.

Using this measure, the next job to include is the one that increases $\sum_{i \in J} p_i$ the most, subject to the constraint that the resulting J is a feasible solution.

- Thus consider jobs in non-increasing order of the $p_i s$

Job Scheduling with Deadline—Checking Feasibility

- Theorem: Let J be a set of k jobs and $\sigma = i_1, i_2, \ldots, i_k$ a permutation of jobs in J such that $d_{i1} \leq d_{i2} \leq \ldots d_{ik}$. Then J is a feasible solution iff the jobs in J can be processed in the order σ without violating any deadline.

Job Scheduling with Deadline—Checking Feasibility

- Clearly, if the jobs in J can be processed in the order σ without violating any deadline, then J is a feasible solution.

- Enough to show that if J is feasible then σ represents a possible order in which the jobs can be processed.

- Idea is to transform any other feasible processing order to σ.

Job Scheduling with Deadline—Checking Feasibility

- If J is feasible, then there exists $\sigma' = r_1, r_2, \ldots, r_k$ such that $d_{r_q} \geq q, 1 \leq q \leq k$

- Assume $\sigma' \neq \sigma$. Then let a be the least index such that $r_a \neq i_a$.

- Let $r_b = i_a$. Clearly, $b > a$.
- In σ' we can interchange r_a and r_b. Since $d_{r_a} \geq d_{r_b}$

Job Scheduling with Deadline—Checking Feasibility

- The resulting permutation $\sigma'' = s_1, s_2, \ldots, s_k$ represents an order in which the jobs can be processed without violating a deadline.

Job Scheduling with Deadline

- Theorem: The greedy method described before always obtains an optimal solution to the job sequencing problem.

Job Scheduling with Deadline—Greedy Choice

- Let I be the set of jobs in the greedy solution and J be the set of Jobs in the optimal solution

- Observation: $I \neq J$ and $I \not\subset J$ and $J \not\subset I$

- Consider jobs $a \in I \setminus J$ and $b \in J \setminus I$

Job Scheduling with Deadline—Greedy Choice

- Consider a particular schedule S_I for I and S_J for J
- Show that the common jobs in I and J can be scheduled at the same time in both the schedules while preserving feasibility.

- Now, choose job ‘a’ as before but with highest profit. Show that $p_a \geq p_b$

- Replace b from the transformed schedule S'_j with a.

Job Scheduling with Deadline—Greedy Choice

- Repeat the above steps to transform J to I.

Job Scheduling with Deadline--Algorithm

```
GreedyJob(d, J, n)
    // J  is a set of jobs that can be completed by their deadline
    J := {1};
    for i := 2 to n
            do
                if  (all jobs in J ∪ {i} can be completed by their deadlines)
                    then
                            J := J ∪ {i};
```

```
JS(d, j, n)
    // d[i] ≥ 1, 1 ≤ i ≤ n  are the deadlines, n ≥ 1. The jobs
    // are ordered such that p[1] ≥ p[2] ≥ ··· ≥ p[n].J[i]
    // is the ith job in the optimal solution, 1 ≤ i ≤ k
    //Also, at termination d[J[i]] ≤ d[J[i + 1]], 1 ≤ i < k
    d[0] := J[0] := 0//Initialize
    J[1] := 1//Include Job 1
    k := 1;
    for i := 2 to n // Consider jobs in nonincreasing order of p[i].
    //Find position for i and check feasibility of insertion

            do
                r := k
                while ((d[J[r]] > d[i]) and (d[J[r]] ≠ r))
                        do r = r − 1
                if ((d[J[r]] ≤ d[i]) and (d[i] > r))
                    then
                        for q = k to r + 1 Step -1
                            do J[q + 1] = J[1]
    return k;
```

Dynamic Programming -- Introduction