

# The Greedy Technique

A technique used in solving **optimization** problems.

Typically, we are given a set of  $n$  inputs and the goal is to find a subset (or some output) that satisfies some constraints.

Any subset (or output) that satisfies these constraints is called a *feasible* solution. In an optimization problem, we need to find a feasible solution that *maximizes* or *minimizes* a given objective function. A feasible solution that does this is called an *optimal solution*.

The greedy technique works in stages, considering one input at a time (typically in some clever order). At each stage, a decision is made depending on whether it is best at this stage. For example, a simple criterion can be whether adding the current input will lead to an infeasible solution or not.

Thus, a **locally optimal** choice is made in the hope that it will lead to a **globally optimal** solution.

# Knapsack Problem

We are given  $n$  objects and a knapsack.

Object  $i$  has a weight  $w_i$  and the knapsack has 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 maximize the total profit earned subject to the knapsack constraint.

Observations:

1. If the sum of all the weights is  $\leq m$ , then  $x_i = 1$ ,  $1 \leq i \leq n$  is an optimal solution.
2. All optimal solutions will fill the knapsack exactly.

# Greedy Strategies

Example:  $n = 3$ ,  $m = 20$ ,  $(p_1, p_2, p_3) = (25, 24, 15)$  and  $(w_1, w_2, w_3) = (18, 15, 10)$ .

Four feasible solutions are:

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

Many simple greedy strategies are possible:

1. Fill the knapsack by including next the object with largest profit. If an object under consideration doesn't fit, then a fraction of it is included to fill the knapsack. (solution 2)

2. Fill the knapsack by including objects in nondecreasing order of weights. (solution 3)

3. Consider objects in nonincreasing order of  $p_i/w_i$ . (solution 4) This is optimal.

Can be implemented in  $O(n \log n)$  time.

# Proof of Optimality

**Theorem 1.** *If objects are included in the nonincreasing order of  $p_i/w_i$  then this gives an optimal solution to the knapsack problem.*

We use the following technique which is typically useful in proving optimality of greedy algorithms.

Compare the greedy solution with the optimal. If the two solutions differ, then find the first  $x_i$  at they differ. Then show how to make  $x_i$  in the optimal solution equal to that of the greedy solution without loss of the total value. Repeated use of this transformation shows that the greedy solution is optimal.

**Proof.** Let  $x = (x_1, \dots, x_n)$  be the solution generated by the greedy algorithm.

If  $x_i = 1$ , for all  $i$ , then clearly the solution is optimal.

Let  $j$  be the first index such that  $x_j \neq 1$ .

Then,  $x_i = 1$  for  $1 \leq i < j$ ,  $x_i = 0$  for  $j < i \leq n$ , and  $0 \leq x_j < 1$ .

Let  $(y_1, \dots, y_n)$  be an optimal solution. Then we can assume that  $\sum w_i y_i = m$ . Let  $k$  be the least index such that  $y_k \neq x_k$ . Then  $y_k < x_k$ . To see this consider three possibilities:

- If  $k < j$ , then  $x_k = 1$ . Since  $y_k \neq x_k$ , and so  $y_k < x_k$ .
- If  $k = j$ , then since  $w_i x_i = m$  and  $y_i = x_i$  for all  $1 \leq i < j$ , either  $y_k < x_k$  or  $\sum w_i y_i > m$ .
- If  $k > j$ , then  $\sum w_i y_i > m$ .

Suppose we increase  $y_k$  to  $x_k$  and decrease as many of  $(y_{k+1}, \dots, y_n)$  as necessary.

This results in a new solution  $(z_1, \dots, z_n)$  with  $z_i = x_i$ , for  $1 \leq i \leq k$  and

$$\sum_{k < i \leq n} w_i (y_i - z_i) = w_k (z_k - y_k).$$

We calculate the total profit for  $z$ :

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

$$= \sum_{1 \leq i \leq n} p_i y_i$$

If  $\sum_{1 \leq i \leq n} p_i z_i > \sum_{1 \leq i \leq n} p_i y_i$ , then  $y$  cannot be optimal. Otherwise,  $z$  is optimal and either  $z = x$  or  $z \neq x$ . Repeat the argument with  $z$ .  $\square$

# Caching or Demand Paging

Assume that a processor has a fast memory device — **cache** that can store  $k$  items (pages), and a slower, much larger, secondary memory.

If a processor requests an item that is in the cache (a **hit**) it has no cost.

If the processor requests an item that is not in the cache (a **miss**) it costs one unit of time to **fetch** it to the cache from the secondary memory.

When an item is fetched to the cache, another item is **evicted**.

An item is fetched only when it is requested — Demand Paging.

Given a sequence of item requests, the goal is to **minimize** the number of page misses.

## Possible eviction strategies

- Least Recently Used (LRU)
- First-in First-out (FIFO).
- Least Frequently Used (LFU).

What is the best eviction policy?

Suppose we know the complete sequence of page requests in advance. Then the following greedy strategy is the best:

“If the requested page is not in the cache then evict the page whose next request is the farthest.”

**Longest-Forward-Distance (LFD)** strategy.

## Example

Consider the request sequence  $a, b, c, d, a, d, e, a, d, b, c$  and a cache of size  $k = 3$ .

LFD strategy will evict  $c$  on the 4th request and  $b$  on the 7th request.

However, evicting  $b$  on the 4th step and  $c$  on the 7th step is will also give the same misses.

# Proof of Optimality of LFD

**Theorem 2.** *LFD is an optimal eviction strategy for caching.*

**Proof.** We will show how any optimal algorithm can be modified to behave like LFD without degrading its performance. The proof depends on the following claim.

**Claim 1.** *Let  $ALG$  be an optimal caching algorithm. Let  $\sigma$  be any request sequence.*

*For any  $i, i = 1, 2, \dots, |\sigma|$ , it is possible to construct an algorithm  $ALG_i$  that satisfies the following 3 properties:*

- 1.  $ALG_i$  processes the first  $i - 1$  requests exactly as  $ALG$  does.*
- 2. If the  $i$ th request results in a page fault,  $ALG_i$  uses the LFD strategy to evict a page.*

3.  $ALG_i(\sigma) = ALG(\sigma)$ , i.e.,  $ALG_i$  has the same number of page misses as  $ALG$  on the entire sequence  $\sigma$ .

We can prove the Theorem by applying the Claim  $n = |\sigma|$  times.  $\square$