Dynamic Programming and Greedy Algorithms

Introduction

In this lecture we learn about the principles of dynamic programming and greedy algorithms via a small series of illustrative examples. These include some simple scheduling problems, shortest path problems and mean cycle problems.

Dynamic programming was formulated in many different ways in different segments of the literature. The principle idea is very similar to the idea of a mathematical recursion. In the context of sequential decision making Bellman (1952,1957) formulated it as the "Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the (current) state resulting from the first decision."

Greedy algorithms build a solution by extending a partial solution, without any backtracking, in such a way that provides the most improvement with respect to the overall objective.

Both types of algorithms can be applied to almost any optimization problem. If designed appropriately, dynamic programming will deliver an optimal solution, though it may require extensive computational resources, in cases when the principle of optimality does not hold. Even in cases when it does, it takes careful analysis to find the most efficient implementation. Greedy algorithms, on the other hand, always can be implemented to run efficiently, though they may not deliver an optimal (or even close to optimal) solution. It will take careful analysis to see under what conditions will it provide us with the right solutions.

We start as an illustration with a simple example.

A scheduling problem

Let us consider a single processor that can process one job at a time, and a set of n jobs, each with a start time s_i, and a processing time p_i, $i = 1, ..., n$. Our task is to chose a subset S of the jobs that can be processed without overlap and without any delays and interruptions, on the single processor such that the number of the chosen jobs is as large as possible. In other words, if we chose job i, then we have to process it in the time interval $[s_i, s_i + p_i)$. Let us denote by J the set of jobs, described as pairs $j_i = (s_i, p_i)$, $i = 1, ..., n$.

Dynamic programming solution

Assume that job i^* is chosen in the optimal solution $S \subseteq J$, and denote by $J(> i^*) = \{j_i \mid s_i \geq s_{i^*} + p_{i^*}\}$.

Claim 1 (Principle of Optimality) *$S \cap J(> i^*)$ must be an optimal solution with respect to the input $J(> i^*)$.*

Proof.If there were a better solution in this subproblem, say $Q \subseteq J(> i^*)$, with $|Q| > |S \cap J(> i^*)|$, then $S' = (S \setminus J(> i^*)) \cup Q$ would also be better for the original problem. □

Let us assume that the jobs are sorted such that $s_1 \leq s_2 \leq \cdots \leq s_n$, define $k(i) = \min\{k \mid s_k \geq s_i + p_i\}$ for all $i = 1, 2, ...$, and let $v(i)$ denote the size of the optimal solution in the subproblem $J(i) = \{j_i, j_{i+1}, \ldots, j_n\}$. Then, by the above principle, we have

$$v(i) \;=\; \max\{1 + v(k(i)),\; v(i+1)\}$$

(see proof in class.)

Algorithm 1 Dynamic programming for the simple scheduling problem

```
1: procedure DYNAMIC PROGRAMMING SCEDULING(J)        ▷ Set of jobs
2:     Sort J by start times.
3:     Compute k(i) for all i = 1, ..., n.
4:     Set v(n + 1) = 1.
5:     for ℓ = n → 1 do
6:         Set v(ℓ) = max{1 + v(k(ℓ)), v(ℓ + 1)}.
7:     end for
8:     return v(1)                                   ▷ The optimum value
9: end procedure
```

Claim 2 *The above procedure compute the optimum value in $O(n \log n)$ time.*

Proof.See in class. □

How to compute an optimal solution?

Greedy solution

Let us assume now that the jobs are sorted by their finish time, that is $s_1 + p_1 \leq s_2 + p_2 \leq \cdots \leq s_n + p_n$, and assume that J is an ordered set of jobs, sorted by the above order.

In the greedy algorithm for this problem we will create a solution $S \subseteq J$ in the following way:

Algorithm 2 Greedy algorithm for the simple scheduling problem

```
1: procedure GREEDY SCHEDULING(J)                    ▷ Set of jobs
2:     Sort J by finish times, set k = 1, J_1 = J, and S = ∅.
3:     while J_k ≠ ∅ do
4:         Let j_k be the first job in J_k.
5:         Set S = S ∪ {j_k}
6:         Set J_{k+1} = J_k \ {j_i ∈ J_k | s_i < s_k + p_k}, and set k = k + 1.
7:     end while
8:     return S                                      ▷ The greedy optimum
9: end procedure
```

Shortest Path Problems

Let us consider a directed graph $G = (V, A)$, and a "length" function $\ell : A \to \mathbb{R}$. For a path $P \subseteq A$ the length of P is

$$\ell(P) = \sum_{(u,v)\in P} \ell(u, v).$$

SHORTEST PATH PROBLEM[1]: Given a directed graph $G = (V, A)$, arc lengths $\ell : A \to \mathbb{R}$, and vertices $s, t \in V$, find the shortest $s \longrightarrow t$ path P.

Note that if we disregard alternative shortest paths, then shortest paths from a source vertex s to all other vertices form a (spanning) tree (assuming that all vertices can be reached from s.)

SHORTEST PATH PROBLEM[2]: Given a directed graph $G = (V, A)$, arc lengths $\ell : A \to \mathbb{R}$, and a vertex $s \in V$, find the tree of shortest paths from s to all other vertices.

ALL PAIRS SHORTEST PATHS: Given a directed graph $G = (V, A)$, arc lengths $\ell : A \to \mathbb{R}$, find the shortest paths between all pairs of vertices.

Remark 3 *Shortest path problems make sense if all arc lengths are nonnegative. Some applications may still involve both positive and negative arc lengths.*

Remark 4 *Shortest paths exists if there are no negative cycles in the graph. Finding the shortest simple path in the presence of negative cycles is a hard problem!*

Remark 5 *If there are no negative cycles, one can apply a potential transformation so that all arc lengths become nonnegative; Gallai (1958).*

See class for the details of all the above remarks.

Reachability

Let us first consider the simpler problem of finding all vertices that can be reached via a directed path from a given vertex s. For a vertex $u \in V$ let us denote by $N^+(u)$ the set of out-neighbors of u, that is the set of vertices v such that $(u, v) \in A$.

Algorithm 3 Reachability in directed graphs

```
1: procedure LABELING ALGORITHM(G = (V, A) and s ∈ V)
2:     Set L = ∅.
3:     Set T = {s}.
4:     Set π(v) = nil for all v ∈ V \ {s}, and set π(s) = s.
5:     while T ≠ ∅ do
6:         Let u ∈ T and set T = T \ {u}.
7:         for v ∈ N⁺(u) do
8:             if π(v) = nil then
9:                 Set π(v) = u and T = T ∪ {v}.
10:            end if
11:        end for
12:        Set L = L ∪ {u}.
13:    end while
14:    return L                                  ▷ Reachable vertices
15: end procedure
```

Claim 6 *The vertices in L are exactly the ones that are reachable from s by a directed path in G.*

Bellman-Ford Algorithm

The following dynamic programming algorithm computes the shortest paths in a given directed graph from a given vertex to all other vertices; Bellman (1958), Ford (1956).

Claim 7 (Principle of Optimality) *If P is a shortest $s \longrightarrow t$ path and x is a vertex on P, then $P[x,t]$ is also a shortest $x \longrightarrow t$ path.*

Let us denote by $d_{\leq k}(s,v)$ the length of a shortest $s \longrightarrow v$ path that uses at most k arcs. What is $d_{\leq 1}(s,v)$?

Claim 8 *The above algorithm computes the lengths of the shortest paths from s to all other vertices in $O(|V||A|)$ time.*

Algorithm 4 Dynamic programming for shortest paths

procedure BELLMAN-FORD ALGORITHM($G = (V, A)$, $\ell : A \to \mathbb{R}$ and $s \in V$)
  Set $d_{\leq 0}(s, v) = +\infty$ for all $v \neq s$.
  Set $d_{\leq k}(s, s) = 0$ for all $k = 0, 1, ..., n-1$.
  for $k = 1 \to n-1$ **do**
    for $v \in V \setminus \{s\}$ **do**
      $d_{\leq k}(s, v) = \min_{(u,v) \in A} d_{\leq k-1}(s, u) + \ell(u, v)$
    end for
  end for
  return $d_{\leq n-1}(s, v)$ for $v \in V$.
end procedure

Proof.See in class. □

How to find those shortest paths?? What happens if there are negative arc lengths?

Dijkstra's Algorithm

The following "greedy" algorithm to compute shortest path from a vertex to all others was proposed by Dijkstra in 1956 (published in 1959.)

The algorithm monotonically decreases an upper bound $\Delta(v)$ on the length of a shortest $s \to v$ path. In each main iteration it accepts $\Delta(v)$ as the true shortest path length for one vertex, the one with the lowest upper bound value.

Algorithm 5 Greedy type algorithm for shortest paths

```
1: procedure Dijkstra's Algorithm(G = (V, A), ℓ : A → ℝ and s ∈ V)
2:     Set Δ(v) = +∞ for all v ≠ s, and Δ(s) = 0.
3:     Set π(s) = s, and π(v) = nil for all v ∈ V \ {s}.
4:     Set S = ∅, L = {s}.
5:     while L ≠ ∅ do
6:         Let u ∈ L be the vertex with the lowest Δ(u) value.
7:         Set L = L \ {u}.
8:         for v ∈ N⁺(u) do
9:             if Δ(v) = +∞ then
10:                Set L = L ∪ {v}.
11:                Set Δ(v) = Δ(u) + ℓ(u, v).
12:                Set π(v) = u.
13:            else
14:                if Δ(v) < Δ(u) + ℓ(u, v) then
15:                    Set Δ(v) = Δ(u) + ℓ(u, v).
16:                    Set π(v) = u.
17:                end if
18:            end if
19:        end for
20:        Set S = S ∪ {u}.
21:    end while
22:    return Δ(v) and π(v) for v ∈ V.
23: end procedure
```

Claim 9 *The above algorithm computes the shortest $s \to v$ paths for all vertices reachable from s (and all others will have $\Delta(v) = \infty$.) Furthermore, $\pi(v)$ encodes the tree of shortest paths. The algorithm runs in $O(|A|)$ time.*

Proof.See in class. □

All Pairs Shortest Paths

Given two matrices $A, B \in \mathbb{R}^{n \times n}$, let us define $C = A \otimes B$ by

$$C[i,j] \;=\; \min_k A[i,k] + B[k,j]$$

for all $1 \leq i, j \leq n$.

Claim 10 *If $A[u,v] = d_{\leq p}(u,v)$ and $B[u,v] = d_{\leq q}(u,v)$ for all $u, v \in V$, then with $C = A \oplus B$ we have that $C[u,v] = d_{\leq p+q}(u,v)$.*

Proof.See in class. □

Let us then define matrix A as the vertex-vertex distance adjacency matrix of G, that is for which

$$A[u,v] = \begin{cases} \ell(u,v) & \text{if } (u,v) \in A, \\ +\infty & \text{otherwise.} \end{cases}$$

Then we have $A[u,v] = d_{\leq 1}(u,v)$ for all $u, v \in V$.

Claim 11 *Let B be the $(n-1)$th power of A with respect to the $\oplus$ operation. Then $B[u,v]$ is the length of the shortest $u \longrightarrow v$ path, for all $u, v \in V$. Matrix B can be computed in $O(|V|^3 \log |V|)$ time.*

Proof.See in class. □

Floyd-Warshall Improvement (1962)

Compute the all pairs shortest paths via the first k vertices, $k = 1, 2, ..., n$ recursively.

Claim 12 *The above algorithm computes the lengths of all pairs shortest paths in $O(|V|^3)$ time.*

Algorithm 6 All pairs shortest paths

procedure FLOYD-WARSHALL ALGORITHM($G = (V, A)$, $\ell : A \to \mathbb{R}$)
    We assume $V = \{1, 2, ..., n\}$.
    Set $B_0[u, v] = \ell(u, v)$ if $(u, v) \in A$, and $= +\infty$ otherwise, for all $u, v \in V$.
    for $w = 1 \to n - 1$ **do**
        for $u, v \in V$ **do**
            $B_w[u, v] = \min\{B_{w-1}[u, v], B_{w-1}[u, w] + B_{w-1}[w, v]\}$
        end for
    end for
    return $B_{n-1}[u, v]$ for $u, v \in V$.
end procedure

Mean Cycle Problems

Cyclic games

Consider a directed graph $G = (V, A)$ with $N^+(v) \neq \emptyset$ for all $v \in V$, and a partition $V = B \cup W$ of the vertices. We also have a real valued *local reward* $\ell : A \longrightarrow \mathbb{R}$ on the arcs. There are two players, MIN(inimizer) who controls vertices in V and MAX(imizer) who controls vertices in W. There is an initial vertex $v_0 \in V$ commonly agreed. Players choose an outgoing arc from each of the vertices they control. This defines a unique infinite walk v_0, v_1, ... We associate to such a walk the infinite sequence of local rewards: $\ell(v_0, v_1)$, $\ell(v_1, v_2)$, ... MIN pays MAX the amount

$$\phi \;=\; \liminf_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \ell(v_{t-1}, v_t),$$

which is also called the *mean payoff*. Naturally, MIN would like to minimize this, while MAX is trying to maximize it.

An equilibrium is when none of the players can change (while the other is not changing) so that he/she improves.

It is easy to see that every situation in this game in fact is a "lasso", that is a sequence of arcs ending in a cycle, in which then it cycles infinitely. Thus, ϕ will simply be the average local reward along this cycle.

In the following subsection we show that finding a minimum mean cycle (or maximum mean cycle is possible in polynomial time, using a shortest path

based dynamic programming algorithm, see Karp (1973). The consequence is that if one of the players fixes his(her) strategy, then we can compute in polynomial time the best response of the other player. Thus, the decision problem "Is the equilibrium value above Z?" belongs to both NP and co-NP. The true complexity of this decision problem is not known, yet.

Minimum mean cycle problem

Let us consider a fixed $s = v_0$ vertex, and denote by $d_{=k}(s, v)$ the length of the shortest $s \to v$ path consisting of **exactly** k arcs.

For a cycle $C \subseteq A$ let

$$\mu(C) \;=\; \frac{\sum_{e \in C} \ell(e)}{|C|}$$

denote the mean cycle length, and let

$$\mu^* \;=\; \min_{C \text{ is a cycle in } G} \mu(C)$$

be the minimum mean cycle length in the graph G.

Claim 13 (Karp (1973))

$$\mu^* \;=\; \min_{v \in V} \max_{0 \le k \le n-1} \frac{d_{=n}(s, v) - d_{=k}(s, v)}{n - k}.$$

Proof.Let us assume first that $\mu^* = 0$, that is that there exists a cycle C such that $\mu(C) = 0$ and that there are not negative cycles in G. Since there are no negative cycles in G, every shortest path is a simple path and hence has length at most $n - 1$.

It is immediate that

$$\max_{0 \le k \le n-1} d_{=n}(s, v) - d_{=k}(s, v) \;=\; d_{=n}(s, v) - d_{\le n-1}(s, v).$$

To prove our claim for the case of $\mu^* = 0$ it is enough to show that this latter quantity is equal to zero.

To this end let us consider the zero cycle C, and let u be a vertex on this cycle and consider a shortest path P from s to u. Since P has at most $n - 1$ arcs, extend P by an appropriate number of copies (≥ 1) of C to a longer

path P' (not simple!) such that the number of arcs in P' is at least n. P was a shortest path to u from s, and C has length zero, and thus the length of P' is the same as that of P. Consequently, P' is also a shortest $s \to u$ path.

Let us then consider vertex v on P' which is the $n+1$st vertex along P', starting at s, and let $P'' = P'[s, v]$. Then P'' is also a shortest path to v, consisting of exactly n arcs, and thus we have

$$d_{=n}(s, v) = d_{\leq n-1}(s, v)$$

as claimed.

To see the claim for $\mu^* \neq 0$, let us update the arc lengths $\widehat{\ell}(u, v) = \ell(u, v) - \mu^*$ for all $(u, v) \in A$. Then for every path P we have

$$\widehat{\ell}(P) \;=\; \ell(P) - \mu^* |P|$$

and for every cycle C we have

$$\widehat{\mu}(C) \;=\; \mu(C) - \mu^*.$$

Consequently, for $(G, \widehat{\ell})$ we have the minimum mean cycle length equal to zero, thus by our previous argument we have

$$\begin{aligned}
0 &= \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{\widehat{d}_{=n}(s, v) - \widehat{d}_{=k}(s, v)}{n - k} \\
&= \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{(d_{=n}(s, v) - n\mu^*) - (d_{=k}(s, v) - k\mu^*)}{n - k} \\
&= \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{d_{=n}(s, v) - d_{=k}(s, v) - (n - k)\mu^*}{n - k} \\
&= \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{d_{=n}(s, v) - d_{=k}(s, v)}{n - k} \;-\; \mu^*
\end{aligned}$$

from which

$$\mu^* = \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{d_{=n}(s, v) - d_{=k}(s, v)}{n - k}$$

follows. □