Today

- All pairs shortest paths
- Dynamic programming failure
- Dynamic programming takeaways
- Planning and Decision Processes

All-pairs shortest paths

- How fast can we compute all shortest paths in a graph?
  - Dijkstra’s gives $O(nm \log_2 n)$. (Requires non-negative weights)
  - Bellman-Ford gives $O(n^2 m)$. (Allows negative weights)
  - (new) Floyd-Warshall gives $O(n^3)$.

Problem. Given $G = (V,E,c)$ with non-negative weights, compute $n \times n$ array $M$ where $M[s,t]$ is the cost of shortest $s \rightarrow t$ path.

- What are good subproblems?

Floyd-Warshall algorithm

- Let $cost(s,t,k)$ be cost of shortest $s \rightarrow t$ path using only vertices $\{1,...,k\}$ as intermediate points.
- Consider $cost(s,t,n)$ for fixed $s,t$.
  - If $n$ not on shortest path, then $cost(s,t,n) = cost(s,t,n-1)$.
  - Otherwise, $cost(s,t,n) = cost(s,n,n-1) + cost(n,t,n-1)$.

\[
\begin{align*}
  cost(s,t,k+1) &= \min \{cost(s,t,k), cost(s,k+1,k) + cost(k+1,t,k)\}
\end{align*}
\]

- Running time. $O(n^3)$.
- Recovering paths requires careful book-keeping.

Interval Scheduling

Problem. Given $n$ shows with start time $s_i$ and finish time $f_i$, watch as many shows as possible, with no overlap.

- Greedy: order by $f_i$ (ascending), take next show if no conflict.
- Dynamic program:
  - Order by finish time $f_1 \leq f_2 \leq \ldots \leq f_n$
  - Compute $p(i) = \max\{j : f_j \leq s_i\}$.
  - $VAL(n) = \max\{VAL(p(n)) + 1, VAL(n-1)\}$.

Another attempt

- Order shows arbitrarily, let $Q(i)$ be the shows that conflict with $i$ (including $i$).
- Consider optimal solution $O$:
  - If $i \notin O$ then $O$ is optimal on $\{1,...,n-1\}$.
  - If $i \in O$ then $O$ is optimal on $\{1,...,n-1\} \setminus Q(i)$.
- Generally, for set of shows $S$, if $i \in S$,
  \[
  VAL(S) = \max\{VAL(S \setminus \{i\}), 1 + VAL(S \setminus Q(i))\}.
  \]

- How many subproblems? $\Omega(2^{n/2})$!
Proof Idea

Suppose shows are 1, . . . , n and show i conflicts with \( n - i + 1 \).

- Process \( \{1, \ldots, n\} \) requires \( \{2, \ldots, n - 1\} \) and \( \{1, \ldots, n - 1\} \).
- \( \{2, \ldots, n - 1\} \) requires \( \{2, \ldots, n - 2\} \) and \( \{3, \ldots, n - 2\} \).
- \( \{1, \ldots, n - 1\} \) requires \( \{1, \ldots, n - 2\} \) and \( \{1, 3, \ldots, n - 2\} \).
- Creates 4 distinct subproblems.

Proof

- Suppose shows are 1, . . . , n and show i conflicts with \( n - i + 1 \).
- Represent subsets as binary strings of length \( n \).
- Only worry about first \( n/2 \) bits (shows 1, . . . , \( n/2 \)).
- Create binary tree, where at level \( i \) process show \( n - i + 1 \). Two subproblems, \( i \) th bit on and \( i \) th bit off.
- Generates all strings on \( n/2 \) bits \( \Rightarrow \Omega(2^{n/2}) \) subproblems.

Dynamic Programming Takeaways

Recipe

- Devise recursive form for solution
- Observe that recursive implementation involves redundant computation. (Often exponential time)
- Design iterative algorithm that solves all subproblems without redundancy.

Concerns

- What are the subproblems? How many are there?
  - Runtime and space complexity.

Decision Processes

- Model of an agent performing a task in an environment.
- Used in AI, robotics, and many other places.

Decision Process

- Set of states \( S = \{1, \ldots, n\} \).
- Set of actions \( A = \{1, \ldots, k\} \).
- Transition model: \( T : S \times A \rightarrow S \).
- Reward function: \( R : S \times A \rightarrow \mathbb{Z} \).
- Timer \( H \).

Trajectories

- Agent starts in \( s_1 \), takes action \( a_1 \), receives reward \( R(s_1, a_1) \) and transitions to \( s_2 \), etc.
- Generates trajectory \( s_1, a_1, r_1, s_2, a_2, r_2, \ldots, s_H, a_H, r_H \), where \( r_h = R(s_h, a_h) \).
- Total reward is, \[ \sum_{h=1}^{H} r_h = \sum_{h=1}^{H} R(s_h, a_h) \]

Goal. Choose actions to maximize total reward.
Decision Process

▶ A policy chooses an action at every state and time,
\[ \pi : (S \times \{1, \ldots, H\}) \rightarrow A \]

Goal. Compute policy to maximize total reward.

Example

If \( H = 1 \):

\[ \pi^*(\cdot, 1) \]

The Planning Problem

Problem. Compute optimal policy in decision process \((S, A, T, R, H)\).

Base case

Consider \( H = 1 \).

▶ The optimal policy is,
\[ \pi^*(s, 1) = \arg\max_{a \in A} R(s, a) \]

▶ The optimal values are,
\[ V^*(s, 1) = \max_{a \in A} R(s, a) \]

▶ \( V^*(s, H) \) is maximum total reward you can achieve starting in state \( s \) with \( H \) actions.

Inductive step

Consider arbitrary \( h \).

▶ If in state \( s \), action \( a \), receive \( R(s, a) \) and transition to \( T(s, a) \) with one less time point.

▶ How much more reward can you receive from \( s' = T(s, a) \) with \( h - 1 \) actions left?

\[ V^*(s, h) = \max_{a \in A} R(s, a) + V^*(T(s, a), h - 1) \]

▶ Policy is,
\[ \pi^*(s, h) = \arg\max_{a \in A} R(s, a) + V^*(T(s, a), h - 1) \]
\[ = \arg\max_{a \in A} Q^*(s, a, h) \]

Example

\[ V^*(\cdot, 1) \]
Value iteration

\[ \text{ValueIteration}(T,R,H) \]

- Initialize \( V^\star(s,0) = 0 \) for all \( s \).
- Initialize \( \pi^\star(s,h) = \text{null} \) for all \( s,h \).
- For \( h=1,\ldots,H \) do
  - For each state \( s \) do
    - \( V^\star(s,h) \leftarrow \max_a R(s,a) + V^\star(T(s,a),h-1) \).
    - \( \pi^\star(s,h) \leftarrow \arg\max_a R(s,a) + V^\star(T(s,a),h-1) \).
  - End for
- End for
- Return \( \pi^\star \).

Extensions

- Works without timer (under some conditions)
- Also works for stochastic (Markov) Decision Processes
- Reinforcement learning: Compute optimal policy when you don’t know \( T,R \)
  - But can sample through experience.