EE365: Approximate Dynamic Programming
Vehicle routing problem

- fleet of $m$ vehicles: $1, \ldots, m$

- transportation network modeled as graph with vertex set $\mathcal{V}$

- vehicle $k$ starts at a given $a_k \in \mathcal{V}$ at time $0$,
  must end at a given $b_k \in \mathcal{V}$ at time $T$

- reward $r_i \geq 0$ for visiting vertex $i$

  - each reward can only be earned once
    - no repeat reward if a vehicle visits $i$ multiple times
    - no repeat reward if multiple vehicles visit $i$

  - after each time period, reward at $i$ disappears with probability $p_i$
Dynamic programming for vehicle routing problem

- state: \( x_t = (z_t^{(1)}, \ldots, z_t^{(m)}, S_t) \)
  - \( z_t^{(k)} \in \mathcal{V} \): location of vehicle \( k \) at time \( t \)
  - \( S_t \subseteq \mathcal{V} \): nodes whose rewards have been removed (because visited or randomly removed)
  - initial state: \((a_1, \ldots, a_m, \emptyset)\)
- disturbance: \( w_t \), locations of randomly-removed rewards
- dynamics:
  - \( z_{t+1} = u_t \)
  - \( S_{t+1} = S_t \cup \{z_t^{(1)}, \ldots, z_t^{(m)}\} \cup w_t \)
- stage cost: \( g(z, S) = \sum \{r_i \mid i \in \{z^{(1)}, \ldots, z^{(m)}\}, i \notin S, i \in \mathcal{V}\} \)
- terminal cost: \( g_T(z, S) = \begin{cases} g(z, S) & z^{(1)} = b_1, \ldots, z^{(m)} = b_m \\ -\infty & \text{otherwise} \end{cases} \)
Complexity of dynamic programming

\[ v_T^*(x) = g_T(x) \]

\[ \text{for } t = T - 1, \ldots, 0; \ x \in \mathcal{X}, \]

\[ v_t^*(x) = \min_{u \in \mathcal{U}} \sum_{w \in \mathcal{W}} \left( g_t(x, u, w) + v_{t+1}^*(f_t(x, u, w)) \right) \text{Prob}(w_t = w) \]

\[ O(T|\mathcal{X}||\mathcal{U}||\mathcal{W}|) \] operations

\[ \text{may be intractable if any of } \mathcal{X}, \mathcal{U} \text{ and } \mathcal{W} \text{ is very large} \]

\[ \text{often intractable due to curse of dimensionality} \]
Intractability of dynamic programming for vehicle routing

\[ \mathcal{X} = \mathcal{V}^m \times 2^\mathcal{V} \]

\[ |\mathcal{X}| = |\mathcal{V}|^m 2^{|\mathcal{V}|} \]

for \( |\mathcal{V}| = 25^2 \), \( m = 4 \), have \( |\mathcal{X}| \approx 10^{200} \)
\((\approx 10^{80} \text{ atoms in the observable universe})\)

cannot even store value function

\[ \mathcal{U}(z^{(1)}, \ldots, z^{(m)}) = \mathcal{N}(z^{(1)}) \times \cdots \times \mathcal{N}(z^{(m)}) \]

\[ |\mathcal{U}| = \prod_{k=1}^{m} |\mathcal{N}(z^{(k)})| \sim d_{\text{max}}^m \]

for \( m = 4 \), \( d_{\text{max}} = 4 \), have \( |\mathcal{U}| = 256 \)

not intractable for this problem

\[ \mathcal{W} = 2^\mathcal{V} \]

\[ |\mathcal{W}| = 2^{|\mathcal{V}|} \]

for \( |\mathcal{V}| = 25^2 \), have \( |\mathcal{W}| \approx 10^{188} \)

cannot compute expectation using summation
Approximate dynamic programming

- in state $x$ at time $t$, choose action

$$u_t(x) \in \arg\min_{u \in \tilde{U}_t(x)} \left( \frac{1}{N} \sum_{k=1}^{N} (g_t(x, u, w_t^{(k)}) + \tilde{v}_{t+1}(f_t(x, u, w_t^{(k)}))) \right)$$

- computation performed on-line
  - look one step into the future
  - will consider multi-step lookahead policies later in the class
- $w^{(k)}$ are independent realizations of $w_t$
- three approximations
  - approximate value function $\tilde{v}_{t+1}$
  - subset of actions $\tilde{U}_t(x)$
  - Monte Carlo approximation of expectation
- choosing $\tilde{v}_{t+1}$ and $\tilde{U}_t(x)$ is an art rather than a science
- may not need all three approximations for some problems
Approximate value functions

- used when it is impossible to store/compute the optimal value function
- policy may no longer be optimal
  - lower bound on optimal cost used to estimate suboptimality
- the achieved cost is a continuous function of the value function
  - if approximate value function is close to optimal value function, then achieved cost is close to optimal cost
- can also approximate $Q$-function instead of value function
- a good approximate value function allows us to approximate future costs
  - accounting for future costs is key to dynamic programming
  - additive constants do not affect the policy
Methods for designing approximate value functions

- heuristic formulas
- optimization
  - solve relaxations of the HJB equation
  - not the focus of this class
- an algorithm to compute the approximate value of a state
  - often DP for a simpler problem
Approximate action sets

- heuristic for identifying a few actions that are likely to produce good results
- do not need to determine the entire approximate action set in advance
  - can keep trying actions until you find an acceptable one
  - can use approximate values to determine which actions to try next (e.g., try something similar to an action you know is good)
Approximating expectation

- simplest method is a Monte Carlo sum
- can do better than Monte Carlo sum
  - each particle in the simulation tells us about the value of many states
  - principle behind reinforcement learning
  - more on this later in the class
Approximate value function for vehicle routing: heuristic formula

- if $d(z^{(k)}, b_k) > T - (t + 1)$ for some $k$, then $\tilde{v}_{t+1}(z^{(1)}, \ldots, z^{(m)}, S) = -\infty$
  - impossible to get to destination by time horizon
- otherwise,

$$
\tilde{v}_{t+1}(z^{(1)}, \ldots, z^{(m)}, S) = \frac{1}{|V| - |S|} \sum_{i \notin S} r_i (1 - p_i)^{1 + \min_k d(z^{(k)}, i)}
$$

- $r_i (1 - p_i)^{1 + \min_k d(z^{(k)}, i)}$ is expected reward if send nearest vehicle to $i$
  - $\{z^{(1)}, \ldots, z^{(m)}\}$ defines a Voronoi decomposition of the plane
  - imagine sending a vehicle to every reward in the Voronoi cell
  - ignores fact that each vehicle can only be sent to one reward
  - factor of $\frac{1}{|V| - |S|}$ is a partial correction
Alternative approximate value function for vehicle routing: algorithm

- problem easy for single vehicle if rewards can be collected multiple times
- fix an order for the vehicles: $k_1, \ldots, k_m$
- for $l = 1, \ldots, m$
  - solve the easy problem for vehicle $k_l$
  - remove the rewards collected by vehicle $k_l$
- $\tilde{v}_{t+1}$ is total reward collected by all vehicles
A greedy algorithm

we will compare the ADP algorithm to a simple greedy algorithm

- if $d(z_t^{(k)}, b_k) \geq T - t$, head straight to destination
- otherwise, head to nearest reward
- simple heuristic with many flaws
  - may send multiple vehicle to same reward, wasting effort
  - ignore distant, valuable reward to collect close, cheap rewards
Comparison of ADP and greedy algorithms

- mean reward of ADP algorithm: 329.27
- mean reward of greedy algorithm: 108.95