6.231 DYNAMIC PROGRAMMING

LECTURE 22

LECTURE OUTLINE

• Aggregation as an approximation methodology
• Aggregate problem
• Examples of aggregation
• Simulation-based aggregation
• Q-Learning
Another major idea in ADP is to approximate the cost-to-go function of the problem with the cost-to-go function of a simpler problem. The simplification is often ad-hoc/problem dependent.

Aggregation is a systematic approach for problem approximation. Main elements:

- Introduce a few “aggregate” states, viewed as the states of an “aggregate” system
- Define transition probabilities and costs of the aggregate system, by relating original system states with aggregate states
- Solve (exactly or approximately) the “aggregate” problem by any kind of value or policy iteration method (including simulation-based methods)
- Use the optimal cost of the aggregate problem to approximate the optimal cost of the original problem

Hard aggregation example: Aggregate states are subsets of original system states, treated as if they all have the same cost.
AGGREGATION/DISAGGREGATION PROBS

- The aggregate system transition probabilities are defined via two (somewhat arbitrary) choices:

\[
\hat{p}_{xy}(u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u) \phi_{jy},
\]

\[
\hat{g}(x, u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u) g(i, u, j).
\]

- For each original system state \( j \) and aggregate state \( y \), the aggregation probability \( \phi_{jy} \)
  
  - The “degree of membership of \( j \) in the aggregate state \( y \).”
  
  - In hard aggregation, \( \phi_{jy} = 1 \) if state \( j \) belongs to aggregate state/subset \( y \).

- For each aggregate state \( x \) and original system state \( i \), the disaggregation probability \( d_{xi} \)
  
  - The “degree of \( i \) being representative of \( x \).”
  
  - In hard aggregation, one possibility is all states \( i \) that belongs to aggregate state/subset \( x \) have equal \( d_{xi} \).
AGGREGATE PROBLEM

• The transition probability from aggregate state \( x \) to aggregate state \( y \) under control \( u \)

\[
\hat{p}_{xy}(u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u) \phi_{jy}, \quad \text{or } \hat{P}(u) = DP(u)\Phi
\]

where the rows of \( D \) and \( \Phi \) are the disaggr. and aggr. probs.

• The aggregate expected transition cost is

\[
\hat{g}(x, u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u) g(i, u, j), \quad \text{or } \hat{g} = DPg
\]

• The optimal cost function of the aggregate problem, denoted \( \hat{R} \), is

\[
\hat{R}(x) = \min_{u \in U} \left[ \hat{g}(x, u) + \alpha \sum_{y} \hat{p}_{xy}(u) \hat{R}(y) \right], \quad \forall x
\]

or \( \hat{R} = \min_{u} [\hat{g} + \alpha \hat{P} \hat{R}] \) - Bellman’s equation for the aggregate problem.

• The optimal cost \( J^* \) of the original problem is approximated using interpolation, \( J^* \approx \tilde{J} = \Phi \hat{R} \):

\[
\tilde{J}(j) = \sum_{y} \phi_{jy} \hat{R}(y), \quad \forall j
\]
**EXAMPLE I: HARD AGGREGATION**

- Group the original system states into subsets, and view each subset as an aggregate state.

- Aggregation probs: \( \phi_{jy} = 1 \) if \( j \) belongs to aggregate state \( y \).

- Disaggregation probs: There are many possibilities, e.g., all states \( i \) within aggregate state \( x \) have equal prob. \( d_{xi} \).

- If optimal cost vector \( J^* \) is piecewise constant over the aggregate states/subsets, hard aggregation is exact. Suggests grouping states with “roughly equal” cost into aggregates.

- Soft aggregation (provides “soft boundaries” between aggregate states).
EXAMPLE II: FEATURE-BASED AGGREGATION

- If we know good features, it makes sense to group together states that have “similar features”
- Essentially discretize the features and assign a weight to each discretization point

- A general approach for passing from a feature-based state representation to an aggregation-based architecture
- Hard aggregation architecture based on features is more powerful (nonlinear/piecewise constant in the features, rather than linear)
- ... but may require many more aggregate states to reach the same level of performance as the corresponding linear feature-based architecture
EXAMPLE III: REP. STATES/COARSE GRID

• Choose a collection of “representative” original system states, and associate each one of them with an aggregate state. Then “interpolate”

\[ x_j \]

Original State Space

Representative/Aggregate States

• Disaggregation probs. are \( d_{xi} = 1 \) if \( i \) is equal to representative state \( x \).

• Aggregation probs. associate original system states with convex combinations of rep. states

\[ j \sim \sum_{y \in A} \phi_{jy} y \]

• Well-suited for Euclidean space discretization

• Extends nicely to continuous state space, including belief space of POMDP
EXAMPLE IV: REPRESENTATIVE FEATURES

- Choose a collection of “representative” subsets of original system states, and associate each one of them with an aggregate state

![Diagram showing original state space and aggregate states/subsets]

- Common case: $S_x$ is a group of states with “similar features”

- Hard aggregation is special case: $\bigcup_x S_x = \{1, \ldots, n\}$

- Aggregation with representative states is special case: $S_x$ consists of just one state

- With rep. features, aggregation approach is a special case of projected equation approach with “seminorm” projection. So the TD methods and multistage Bellman Eq. methodology apply
APPROXIMATE PI BY AGGREGATION

- Consider approximate PI for the original problem, with evaluation done using the aggregate problem (other possibilities exist - see the text)

- **Evaluation of policy** $\mu$: $\tilde{J} = \Phi R$, where $R = DT_\mu(\Phi R)$ ($R$ is the vector of costs of aggregate states corresponding to $\mu$). May use simulation.

- Similar form to the projected equation $\Phi R = \Pi T_\mu(\Phi R)$ ($\Phi D$ in place of $\Pi$).

- **Advantages**: It has no problem with exploration or with oscillations.

- **Disadvantage**: The rows of $D$ and $\Phi$ must be probability distributions.
Q-LEARNING I

• **Q-learning** has two motivations:
  
  – Dealing with multiple policies simultaneously
  
  – Using a model-free approach [no need to know $p_{ij}(u)$, only be able to simulate them]

• The $Q$-factors are defined by

\[
Q^*(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J^*(j) \right), \quad \forall (i, u)
\]

• Since $J^* = TJ^*$, we have $J^*(i) = \min_{u \in U(i)} Q^*(i, u)$
  
  so the $Q$ factors solve the equation

\[
Q^*(i, u) = \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{u' \in U(j)} Q^*(j, u') \right)
\]

• $Q^*(i, u)$ can be shown to be the unique solution of this equation. Reason: This is Bellman’s equation for a system whose states are the original states 1, \ldots , n, together with all the pairs $(i, u)$.

• **Value iteration**: For all $(i, u)$

\[
Q(i, u) := \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha \min_{u' \in U(j)} Q(j, u') \right)
\]
Q-LEARNING II

- Use some randomization to generate sequence of pairs \((i_k, u_k)\) [all pairs \((i, u)\) are chosen infinitely often]. For each \(k\), select \(j_k\) according to \(p_{i_k,j_k}(u_k)\).

- **Q-learning algorithm**: updates \(Q(i_k, u_k)\) by

\[
Q(i_k, u_k) := (1 - \gamma_k(i_k, u_k)) Q(i_k, u_k)
+ \gamma_k(i_k, u_k) \left( g(i_k, u_k, j_k) + \alpha \min_{u' \in U(j_k)} Q(j_k, u') \right)
\]

- Stepsize \(\gamma_k(i_k, u_k)\) must converge to 0 at proper rate (e.g., like \(1/k\)).

- **Important mathematical point**: In the \(Q\)-factor version of Bellman’s equation the order of expectation and minimization is reversed relative to the ordinary cost version of Bellman’s equation:

\[
J^*(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left( g(i, u, j) + \alpha J^*(j) \right)
\]

- **Q-learning can be shown to converge to true/exact \(Q\)-factors (sophisticated stoch. approximation proof).**

- **Major drawback**: Large number of pairs \((i, u)\) - no function approximation is used.
**Q-FACTOR APPROXIMATIONS**

- Basis function approximation for \( Q \)-factors:
  \[
  \tilde{Q}(i, u, r) = \phi(i, u)'r
  \]

- We can use approximate policy iteration and LSPE/LSTD/TD for policy evaluation (exploration issue is acute).

- Optimistic policy iteration methods are frequently used on a heuristic basis.

- **Example** (very optimistic). At iteration \( k \), given \( r_k \) and state/control \((i_k, u_k)\):
  1. Simulate next transition \((i_k, i_{k+1})\) using the transition probabilities \( p_{i_k,j}(u_k) \).
  2. Generate control \( u_{k+1} \) from
     \[
     u_{k+1} = \arg \min_{u \in U(i_{k+1})} \tilde{Q}(i_{k+1}, u, r_k)
     \]
  3. Update the parameter vector via
     \[
     r_{k+1} = r_k - \text{(LSPE or TD-like correction)}
     \]

- **Unclear validity.** Solid basis for aggregation case, and for case of optimal stopping (see text).