6.231 DYNAMIC PROGRAMMING

LECTURE 8

LECTURE OUTLINE

- Suboptimal control
- Cost approximation methods: Classification
- Certainty equivalent control: An example
- Limited lookahead policies
- Performance bounds
- Problem approximation approach
- Parametric cost-to-go approximation

PRACTICAL DIFFICULTIES OF DP

- The curse of dimensionality
  - Exponential growth of the computational and storage requirements as the number of state variables and control variables increases
  - Quick explosion of the number of states in combinatorial problems
  - Intractability of imperfect state information problems
- The curse of modeling
  - Mathematical models
  - Computer/simulation models
- There may be real-time solution constraints
  - A family of problems may be addressed. The data of the problem to be solved is given with little advance notice
  - The problem data may change as the system is controlled – need for on-line replanning

COST-TO-GO FUNCTION APPROXIMATION

- Use a policy computed from the DP equation where the optimal cost-to-go function J_{k+1} is replaced by an approximation $\tilde{J}_{k+1}$. (Sometimes $E\big\{g_k\big\}$ is also replaced by an approximation.)
- Apply $\overline{\mu}_k(x_k)$, which attains the minimum in

$$\min_{u_k \in U_k(x_k)} E\Big\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\}$$

- There are several ways to compute $\tilde{J}_{k+1}$:
  - Off-line approximation: The entire function $\tilde{J}_{k+1}$ is computed for every k, before the control process begins.
  - On-line approximation: Only the values $\tilde{J}_{k+1}(x_{k+1})$ at the relevant next states x_{k+1} are computed and used to compute u_k just after the current state x_k becomes known.
  - Simulation-based methods: These are off-line and on-line methods that share the common characteristic that they are based on Monte-Carlo simulation. Some of these methods are suitable for are suitable for very large problems.

CERTAINTY EQUIVALENT CONTROL (CEC)

- Idea: Replace the stochastic problem with a deterministic problem

- At each time k, the future uncertain quantities are fixed at some "typical" values

- On-line implementation for a perfect state info problem. At each time k:

  (1) Fix the w_i, $i \geq k$, at some $\overline{w}_i$. Solve the deterministic problem:

$$\text{minimize} \quad g_N(x_N) + \sum_{i=k}^{N-1} g_i\big(x_i, u_i, \overline{w}_i\big)$$

where x_k is known, and

$$u_i \in U_i, \quad x_{i+1} = f_i\big(x_i, u_i, \overline{w}_i\big).$$

  (2) Use the first control in the optimal control sequence found.

- Equivalently, we apply $\bar{\mu}_k(x_k)$ that minimizes

$$g_k\big(x_k, u_k, \overline{w}_k\big) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, \overline{w}_k)\big)$$

where $\tilde{J}_{k+1}$ is the optimal cost of the corresponding deterministic problem.

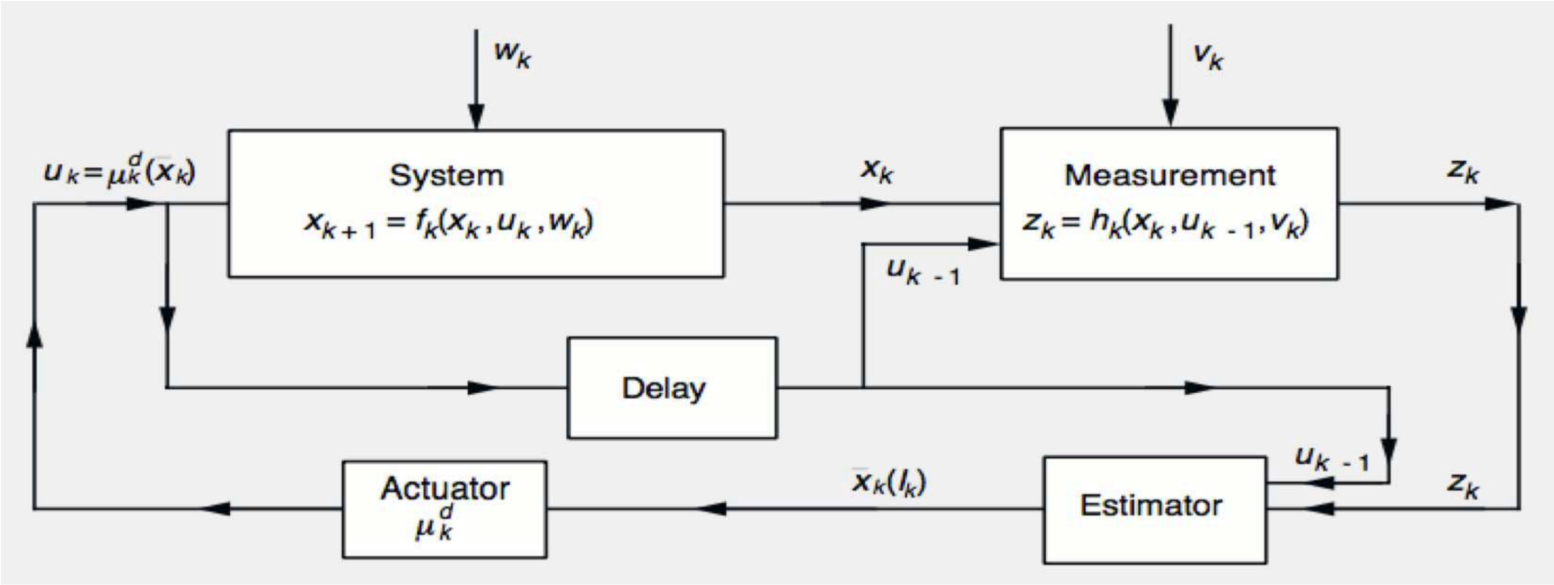
EQUIVALENT OFF-LINE IMPLEMENTATION

- Let $\left\{\mu_0^d(x_0), \ldots, \mu_{N-1}^d(x_{N-1})\right\}$ be an optimal controller obtained from the DP algorithm for the deterministic problem

$$\text{minimize} \quad g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), \overline{w}_k\big)$$

$$\text{subject to} \quad x_{k+1} = f_k\big(x_k, \mu_k(x_k), \overline{w}_k\big), \qquad \mu_k(x_k) \in U_k$$

- The CEC applies at time k the control input $\mu_k^d(x_k)$.
- In an imperfect info version, x_k is replaced by an estimate $\overline{x}_k(I_k)$.



PARTIALLY STOCHASTIC CEC

- Instead of fixing *all* future disturbances to their typical values, fix only some, and treat the rest as stochastic.

- Important special case: Treat an imperfect state information problem as one of perfect state information, using an estimate $\overline{x}_k(I_k)$ of x_k as if it were exact.

- Multiaccess communication example: Consider controlling the slotted Aloha system (Example 5.1.1 in the text) by optimally choosing the probability of transmission of waiting packets. This is a hard problem of imperfect state info, whose perfect state info version is easy.

- Natural partially stochastic CEC:

$$\tilde{\mu}_k(I_k) = \min\left[1, \frac{1}{\overline{x}_k(I_k)}\right],$$

where $\overline{x}_k(I_k)$ is an estimate of the current packet backlog based on the entire past channel history of successes, idles, and collisions (which is I_k).

GENERAL COST-TO-GO APPROXIMATION

- One-step lookahead (1SL) policy: At each k and state x_k, use the control $\overline{\mu}_k(x_k)$ that

$$\min_{u_k \in U_k(x_k)} E\Big\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\},$$

where

  - $\tilde{J}_N = g_N$.
  - $\tilde{J}_{k+1}$: approximation to true cost-to-go J_{k+1}

- Two-step lookahead policy: At each k and x_k, use the control $\tilde{\mu}_k(x_k)$ attaining the minimum above, where the function $\tilde{J}_{k+1}$ is obtained using a 1SL approximation (solve a 2-step DP problem).

- If $\tilde{J}_{k+1}$ is readily available and the minimization above is not too hard, the 1SL policy is implementable on-line.

- Sometimes one also replaces $U_k(x_k)$ above with a subset of "most promising controls" $\overline{U}_k(x_k)$.

- As the length of lookahead increases, the required computation quickly explodes.

PERFORMANCE BOUNDS FOR 1SL

- Let $\overline{J}_k(x_k)$ be the cost-to-go from (x_k, k) of the 1SL policy, based on functions $\tilde{J}_k$.
- Assume that for all (x_k, k), we have

$$\hat{J}_k(x_k) \le \tilde{J}_k(x_k), \qquad (*)$$

where $\hat{J}_N = g_N$ and for all k,

$$\hat{J}_k(x_k) = \min_{u_k \in U_k(x_k)} E\Big\{g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\},$$

[so $\hat{J}_k(x_k)$ is computed along with $\overline{\mu}_k(x_k)$]. Then

$$\overline{J}_k(x_k) \le \hat{J}_k(x_k), \qquad \text{for all } (x_k, k).$$

- Important application: When $\tilde{J}_k$ is the cost-to-go of some heuristic policy (then the 1SL policy is called the rollout policy).
- The bound can be extended to the case where there is a δ_k in the RHS of (*). Then

$$\overline{J}_k(x_k) \le \tilde{J}_k(x_k) + \delta_k + \cdots + \delta_{N-1}$$

COMPUTATIONAL ASPECTS

- Sometimes nonlinear programming can be used to calculate the 1SL or the multistep version [particularly when $U_k(x_k)$ is not a discrete set]. Connection with stochastic programming (2-stage DP) methods (see text).

- The choice of the approximating functions $\tilde{J}_k$ is critical, and is calculated in a variety of ways.

- Some approaches:

  (a) Problem Approximation: Approximate the optimal cost-to-go with some cost derived from a related but simpler problem

  (b) Parametric Cost-to-Go Approximation: Approximate the optimal cost-to-go with a function of a suitable parametric form, whose parameters are tuned by some heuristic or systematic scheme (Neuro-Dynamic Programming)

  (c) Rollout Approach: Approximate the optimal cost-to-go with the cost of some suboptimal policy, which is calculated either analytically or by simulation

PROBLEM APPROXIMATION

- Many (problem-dependent) possibilities
  - Replace uncertain quantities by nominal values, or simplify the calculation of expected values by limited simulation
  - Simplify difficult constraints or dynamics

- Enforced decomposition example: Route m vehicles that move over a graph. Each node has a "value." First vehicle that passes through the node collects its value. Want to max the total collected value, subject to initial and final time constraints (plus time windows and other constraints).

- Usually the 1-vehicle version of the problem is much simpler. This motivates an approximation obtained by solving single vehicle problems.

- 1SL scheme: At time k and state x_k (position of vehicles and "collected value nodes"), consider all possible kth moves by the vehicles, and at the resulting states we approximate the optimal value-to-go with the value collected by optimizing the vehicle routes one-at-a-time

PARAMETRIC COST-TO-GO APPROXIMATION

- Use a cost-to-go approximation from a parametric class $\tilde{J}(x, r)$ where x is the current state and $r = (r_1, \ldots, r_m)$ is a vector of "tunable" scalars (weights).

- By adjusting the weights, one can change the "shape" of the approximation $\tilde{J}$ so that it is reasonably close to the true optimal cost-to-go function.

- Two key issues:
  - The choice of parametric class $\tilde{J}(x, r)$ (the approximation architecture).
  - Method for tuning the weights ("training" the architecture).

- Successful application strongly depends on how these issues are handled, and on insight about the problem.

- Sometimes a simulation-based algorithm is used, particularly when there is no mathematical model of the system.

- We will look in detail at these issues after a few lectures.

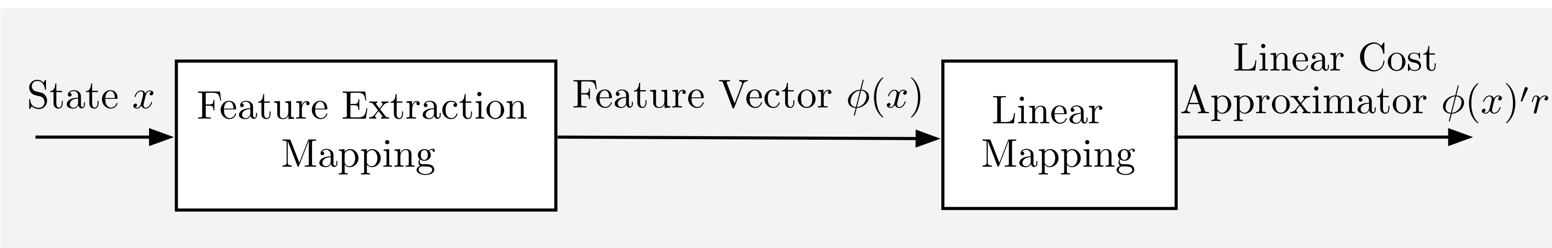
APPROXIMATION ARCHITECTURES

- Divided in linear and nonlinear [i.e., linear or nonlinear dependence of $\tilde{J}(x, r)$ on r]

- Linear architectures are easier to train, but nonlinear ones (e.g., neural networks) are richer

- Linear feature-based architecture: $\phi = (\phi_1, \ldots, \phi_m)$

$$\tilde{J}(x, r) = \phi(x)'r = \sum_{j=1}^{m} \phi_j(x) r_j$$

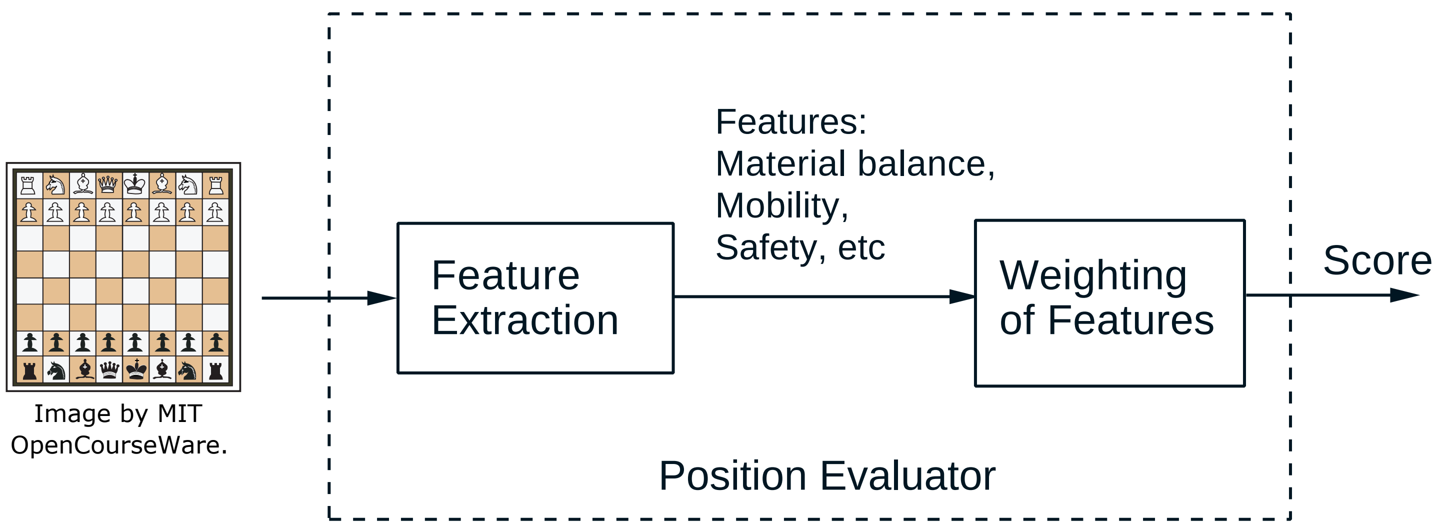


- Ideally, the features will encode much of the nonlinearity that is inherent in the cost-to-go approximated, and the approximation may be quite accurate without a complicated architecture

- Anything sensible can be used as features. Sometimes the state space is partitioned, and “local” features are introduced for each subset of the partition (they are 0 outside the subset)

AN EXAMPLE - COMPUTER CHESS

- Chess programs use a feature-based position evaluator that assigns a score to each move/position



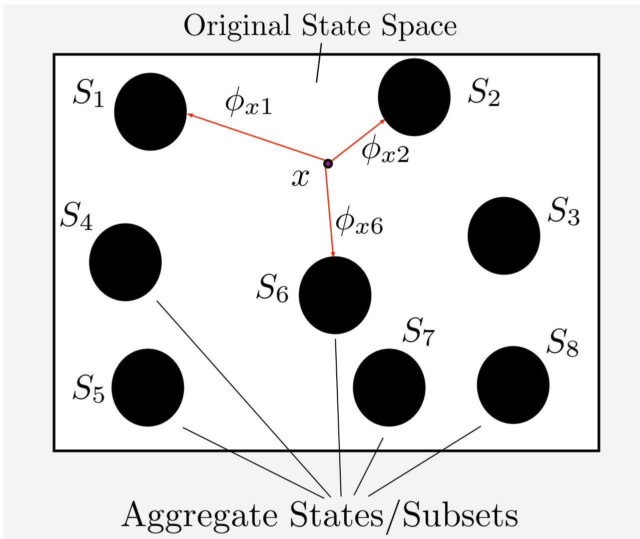
- Many context-dependent special features.
- Most often the weighting of features is linear but multistep lookahead is involved.
- Most often the training is done "manually," by trial and error.

ANOTHER EXAMPLE - AGGREGATION

- Main elements (in a finite-state context):
  - Introduce "aggregate" states $S_1, \ldots, S_m$, 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 (using so called "aggregation and disaggregation probabilities")
  - Solve (exactly or approximately) the "aggregate" problem by any kind of method (including simulation-based) ... more on this later.
  - Use the optimal cost of the aggregate problem to approximate the optimal cost of each original problem state as a linear combination of the optimal aggregate state costs
- This is a linear feature-based architecture (the optimal aggregate state costs are the features)
- Hard aggregation example: Aggregate states S_j are a partition of original system states (each original state belongs to one and only one S_j).

AN EXAMPLE: REPRESENTATIVE SUBSETS

- The aggregate states S_j are disjoint "representative" subsets of original system states



- Common case: Each S_j is a group of states with "similar characteristics"

- Compute a "cost" r_j for each aggregate state S_j (using some method)

- Approximate the optimal cost of each original system state x with $\sum_{j=1}^{m} \phi_{xj} r_j$

- For each x, the ϕ_{xj}, $j = 1, \ldots, m$, are the "aggregation probabilities" ... roughly the degrees of membership of state x in the aggregate states S_j

- Each ϕ_{xj} is prespecified and can be viewed as the jth feature of state x