6.231 Dynamic Programming and Stochastic Control
Fall 2008

6.231 DYNAMIC PROGRAMMING

LECTURE 10

LECTURE OUTLINE

- Suboptimal control
- Certainty equivalent control
- Limited lookahead policies
- Performance bounds
- Problem approximation approach
- Heuristic cost-to-go approximation

PRACTICAL DIFFICULTIES OF DP

- The curse of modeling
- 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
- 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

CERTAINTY EQUIVALENT CONTROL (CEC)

- Replace the stochastic problem with a deterministic problem
- At each time k, the uncertain quantities are fixed at some "typical" values
- Implementation for an imperfect info problem. At each time k:

(1) Compute a state estimate $\overline{x}_k(I_k)$ given the current information vector I_k.

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

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

subject to $x_k = \overline{x}_k(I_k)$ and for $i \geq k$,

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

(3) Use as control the first element in the optimal control sequence found.

ALTERNATIVE IMPLEMENTATION

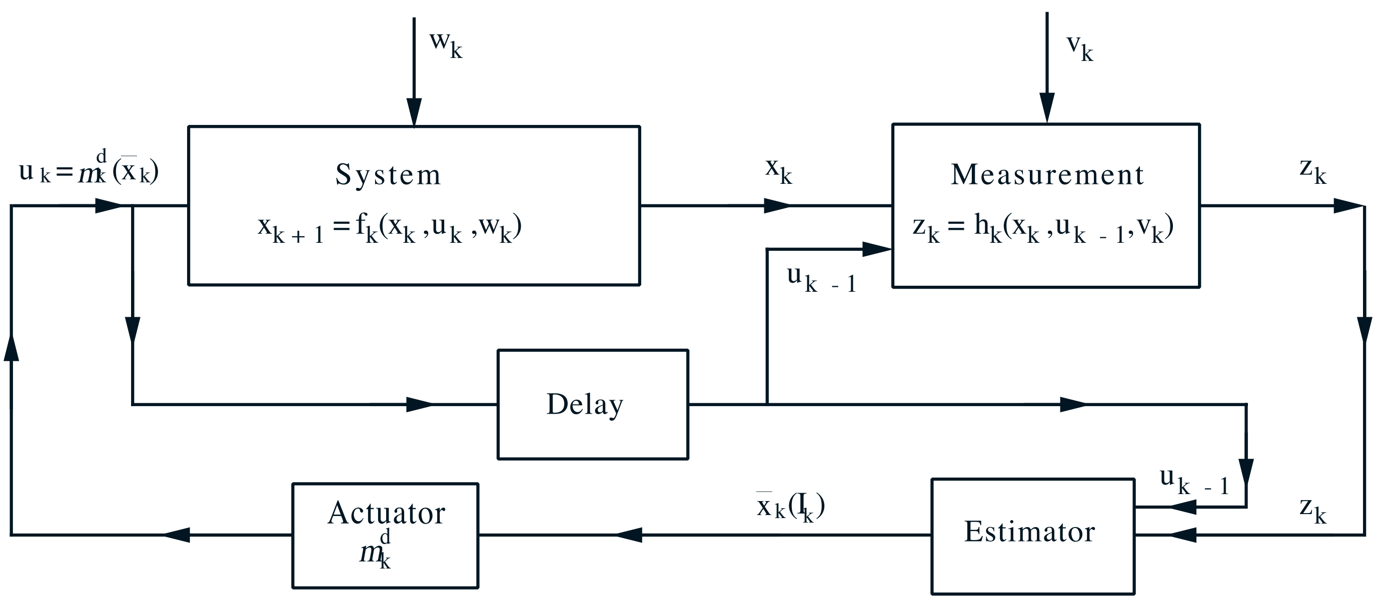
- Let $\{\mu_0^d(x_0), \ldots, \mu_{N-1}^d(x_{N-1})\}$ be an optimal controller obtained from the DP algorithm for the deterministic problem

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

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

The CEC applies at time k the control input

$$\tilde{\mu}_k(I_k) = \mu_k^d\big(\overline{x}_k(I_k)\big)$$



CEC WITH HEURISTICS

- Solve the "deterministic equivalent" problem using a heuristic/suboptimal policy

- Improved version of this idea: At time k minimize the stage k cost and plus the heuristic cost of the remaining stages, i.e., apply at time k a control $\tilde{u}_k$ that minimizes over $u_k \in U_k(x_k)$

$$g_k\big(x_k, u_k, \overline{w}_k(x_k, u_k)\big) + H_{k+1}\Big(f_k\big(x_k, u_k, \overline{w}_k(x_k, u_k)\big)\Big)$$

where H_{k+1} is the cost-to-go function corresponding to the heuristic.

- This an example of an important suboptimal control idea:

Minimize at each stage k the sum of approximations to the current stage cost and the optimal cost-to-go.

- This is a central idea in several other suboptimal control schemes, such as limited lookahead, and rollout algorithms.

- $H_{k+1}(x_{k+1})$ may be computed off-line or on-line.

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 (discussed in Ch. 5) 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).

LIMITED LOOKAHEAD POLICIES

- *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 methods.
- 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) *Heuristic 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

- Example of *enforced decomposition*: Route m vehicles that move over a graph. Each node has a "value." The first vehicle that passes through the node collects its value. 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

HEURISTIC 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.

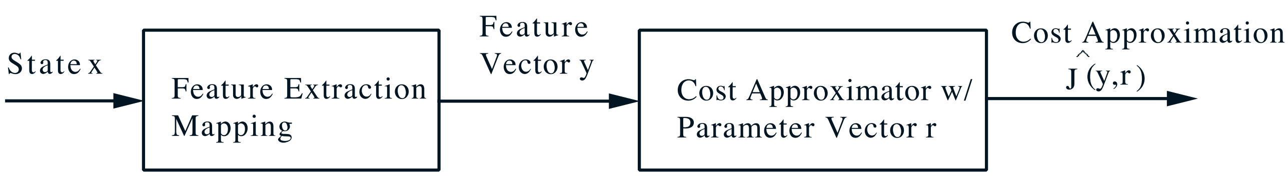
- Sometimes a simulator is used, particularly when there is no mathematical model of the system.

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.

- Architectures based on feature extraction



- 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.

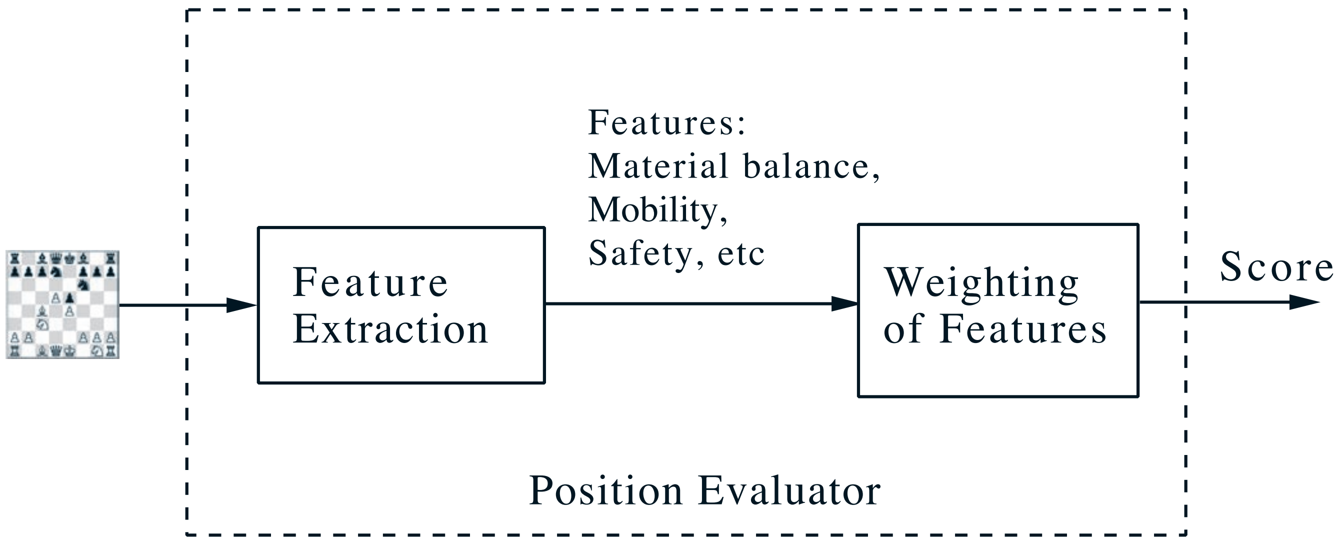
- Sometimes the state space is partitioned, and "local" features are introduced for each subset of the partition (they are 0 outside the subset).

- With a well-chosen feature vector $y(x)$, we can use a linear architecture

$$\tilde{J}(x, r) = \hat{J}\big(y(x), r\big) = \sum_i r_i y_i(x)$$

COMPUTER CHESS

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



- Most often the weighting of features is linear but multistep lookahead is involved.

- Most often the training is done by trial and error.

- Additional features:
  - Depth first search
  - Variable depth search when dynamic positions are involved
  - Alpha-beta pruning