6.231 Dynamic Programming and Stochastic Control
Fall 2008

# 6.231 DYNAMIC PROGRAMMING

## LECTURE 20

## LECTURE OUTLINE

- We begin a 6-lecture series on approximate DP for large/intractable problems (see the detailed outline posted)
- We will mainly follow Chapter 6, Vol. II of the text (with supplemental refs)
- In this lecture we classify/overview the main approaches:
  - Rollout/Simulation-based single policy iteration (we will not discuss this further)
  - Approximation in value space (approximate policy iteration, Q-Learning, Bellman error approach, approximate LP)
  - Approximation in policy space (policy parametrization, gradient methods)
  - Problem approximation (simplification - aggregation - limited lookahead) - we will briefly discuss this today

# APPROXIMATION IN VALUE SPACE

- We will mainly adopt an $n$-state discounted model (the easiest case - but think of HUGE $n$).
- Extensions to SSP and average cost are possible (but more quirky). We will discuss them later.
- Main idea: Approximate $J^*$ or $J_\mu$ with an approximation architecture

$$J^*(i) \approx \tilde{J}(i,r) \qquad \text{or} \qquad J_\mu(i) \approx \tilde{J}(i,r)$$

- Principal example: Subspace approximation

$$\tilde{J}(i,r) = \phi(i)'r = \sum_{k=1}^{s} \phi_k(i) r_k$$

where $\phi_1, \ldots, \phi_s$ are basis functions spanning an $s$-dimensional subspace of $\Re^n$

- Key issue: How to optimize $r$ with low/$s$-dimensional operations only
- Other than manual/trial-and-error approaches (e.g/, as in computer chess), the only other approaches are simulation-based. They are collectively known as "neuro-dynamic programming" or "reinforcement learning"

# APPROX. IN VALUE SPACE - APPROACHES

- Policy evaluation/Policy improvement
  - Uses simulation algorithms to approximate the cost $J_\mu$ of the current policy $\mu$
- Approximation of the optimal cost function $J^*$
  - $Q$-Learning: Use a simulation algorithm to approximate the optimal costs $J^*(i)$ or the $Q$-factors

$$Q^*(i,u) = g(i,u) + \alpha \sum_{j=1}^{n} p_{ij}(u) J^*(j)$$

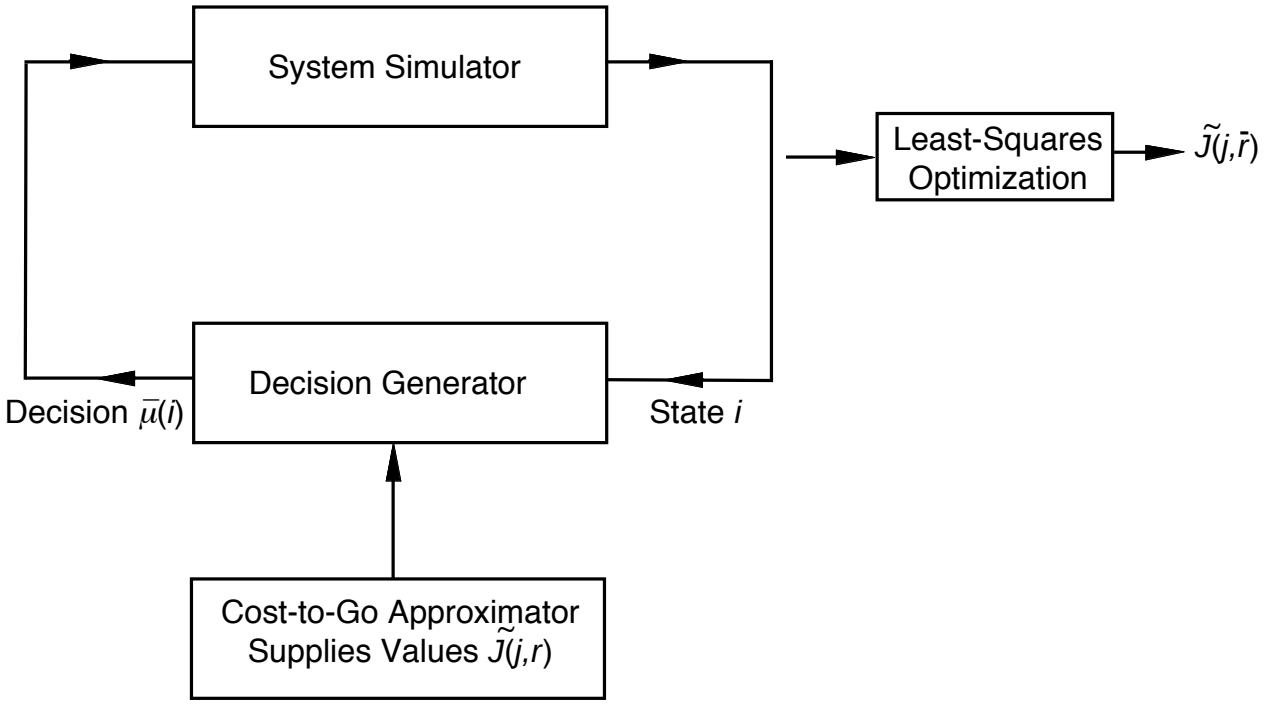
  - Bellman error approach: Find $r$ to

$$\min_r E_i \Big\{ \big( \tilde{J}(i,r) - (T\tilde{J})(i,r) \big)^2 \Big\}$$

    where $E_i\{\cdot\}$ is taken with respect to some distribution
  - Approximate LP (discussed earlier - supplemented with clever schemes to overcome the large number of constraints issue)

# POLICY EVALUATE/POLICY IMPROVE

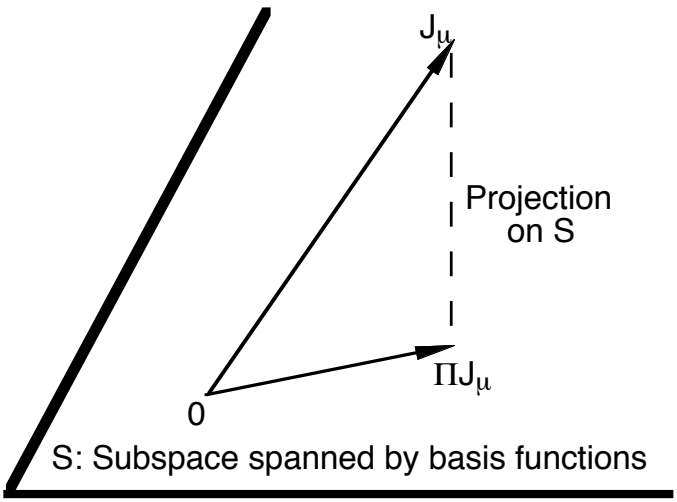
- An example



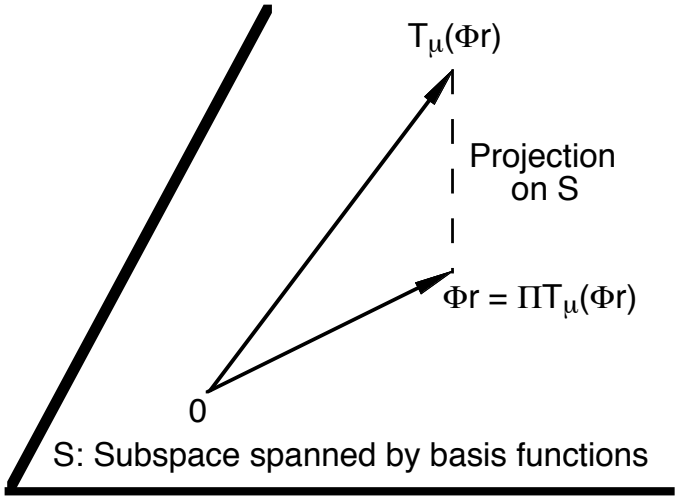

- The "least squares optimization" may be replaced by a different algorithm

# POLICY EVALUATE/POLICY IMPROVE I

- Approximate the cost of the current policy by using a simulation method.
  - Direct policy evaluation - Cost samples generated by simulation, and optimization by least squares
  - Indirect policy evaluation - solving the projected equation $\Phi r = \Pi T_\mu(\Phi r)$ where $\Pi$ is projection w/ respect to a suitable weighted Euclidean norm




Direct Mehod: Projection of cost vector $J_\mu$



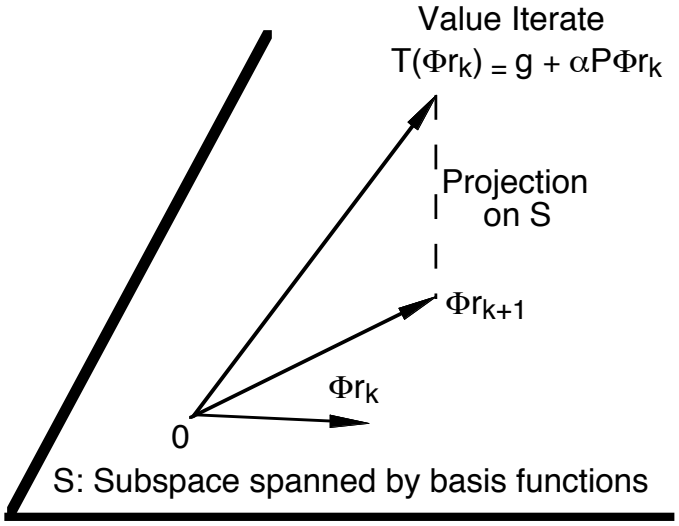

Indirect method: Solving a projected form of Bellman's equation

- Batch and incremental methods
- Regular and optimistic policy iteration

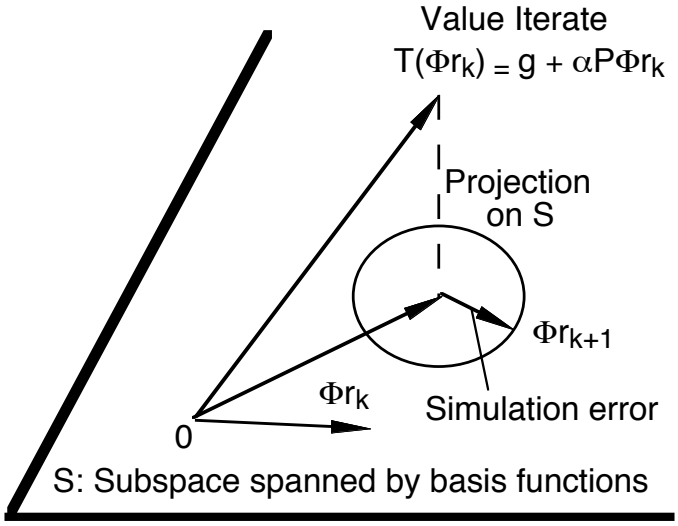
# POLICY EVALUATE/POLICY IMPROVE II

- Projected equation methods are preferred and have rich theory
- TD($\lambda$): Stochastic iterative algorithm for solving $\Phi r = \Pi T_\mu(\Phi r)$
- LSPE($\lambda$): A simulation-based form of *projected value iteration*

$$\Phi r_{k+1} = \Pi T_\mu(\Phi r_k) + \text{ simulation noise}$$




Projected Value Iteration (PVI)




Least Squares Policy Evaluation (LSPE)

- LSTD($\lambda$): Solves a simulation-based approximation $\Phi r = \hat{\Pi}\hat{T}_\mu(\Phi r)$ of the projected equation, using a linear system solver (e.g., Gaussian elimination/Matlab)

# APPROXIMATION IN POLICY SPACE

- We parameterize the set of policies by a vector $r = (r_1, \ldots, r_s)$ and we optimize the cost over $r$.
- In a special case of this approach, the parameterization of the policies is indirect, through an approximate cost function.
  - A cost approximation architecture parameterized by $r$, defines a policy dependent on $r$ via the minimization in Bellman's equation.
- Discounted problem example:
  - Denote by $g_i(r)$, $i = 1, \ldots, n$, the one-stage expected cost starting at state $i$, and by $p_{ij}(r)$ the transition probabilities.
  - Each value of $r$ defines a stationary policy, with cost starting at state $i$ denoted by $J_i(r)$.
  - Use a gradient (or other) method to minimize over $r$

$$\bar{J}(r) = \sum_{i=1}^{n} q(i) J_i(r),$$

where $\big(q(1), \ldots, q(n)\big)$ is some probability distribution over the states.

# PROBLEM APPROXIMATION - AGGREGATION

- 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 (semi-)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 associating multiple states of the original system with each aggregate state
  - Solve (exactly or approximately) the "aggregate" problem by any kind of value or policy iteration method (including simulation-based methods, such as $Q$-learning)
  - Use the optimal cost of the aggregate problem to approximate the optimal cost of the original problem

- Example (Hard Aggregation): We are given a partition of the state space into subsets of states, and each subset is viewed as an aggregate state (each state belongs to one and only one subset).

## AGGREGATION/DISAGGREGATION PROBS

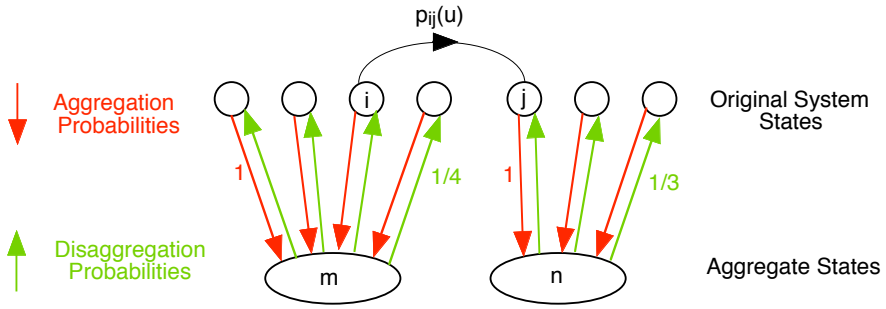
- The aggregate system transition probabilities are defined via two (somewhat arbitrary) choices:

- For each original system state $i$ and aggregate state $m$, the aggregation probability $a_{im}$
  - This may be roughly interpreted as the "degree of membership of $i$ in the aggregate state $m$."
  - In the hard aggregation example, $a_{im} = 1$ if state $i$ belongs to aggregate state/subset $m$.

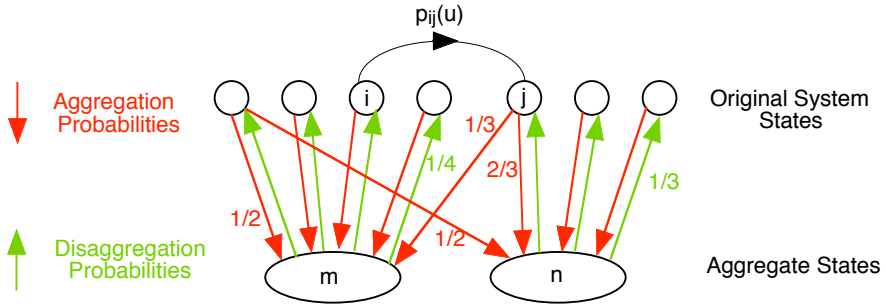
- For each aggregate state $m$ and original system state $i$, the disaggregation probability $d_{mi}$
  - This may be roughly interpreted as the "degree to which $i$ is representative of $m$."
  - In the hard aggregation example (assuming all states that belong to aggregate state/subset $m$ are "equally representative") $d_{mi} = 1/|m|$ for each state $i$ that belongs to aggregate state/subset $m$, where $|m|$ is the cardinality (number of states) of $m$.

# AGGREGATION EXAMPLES

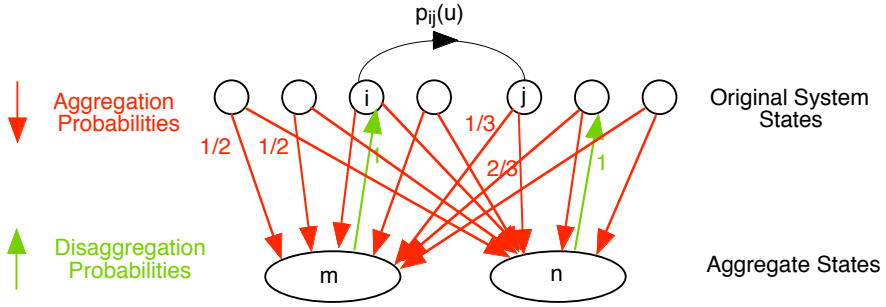
- Hard aggregation (each original system state is associated with one aggregate state):




- Soft aggregation (each original system state is associated with multiple aggregate states):




- Coarse grid (each aggregate state is an original system state):



# AGGREGATE TRANSITION PROBABILITIES

- Let the aggregation and disaggregation probabilities, $a_{im}$ and $d_{mi}$, and the original transition probabilities $p_{ij}(u)$ be given
- The transition probability from aggregate state $m$ to aggregate state $n$ under $u$ is

$$q_{mn}(u) = \sum_i \sum_j d_{mi} p_{ij}(u) a_{jn}$$

and the transition cost is similarly defined.

- This corresponds to a probabilistic process that can be simulated as follows:
  - From aggregate state $m$, generate original state $i$ according to $d_{mi}$.
  - Generate a transition from $i$ to $j$ according to $p_{ij}(u)$, with cost $g(i, u, j)$.
  - From original state $j$, generate aggregate state $n$ according to $a_{jn}$.
- After solving for the optimal costs $\hat{J}(m)$ of the aggregate problem, the costs of the original problem are approximated by

$$\tilde{J}(i) = \sum_m a_{im} \hat{J}(m)$$