

APPROXIMATE DYNAMIC PROGRAMMING

LECTURE 3

LECTURE OUTLINE

- Review of theory and algorithms for discounted DP
- Introduction to approximate DP
- Approximation architectures
- Simulation-based approximate policy iteration
- Approximate policy iteration and Q-factors
- Direct and indirect approximation
- Simulation issues

DISCOUNTED PROBLEMS/BOUNDED COST

- Stationary system with arbitrary state space

$$x_{k+1} = f(x_k, u_k, w_k), \quad k = 0, 1, \dots$$

- Cost of a policy $\pi = \{\mu_0, \mu_1, \dots\}$

$$J_\pi(x_0) = \lim_{N \rightarrow \infty} E_{w_k} \left\{ \sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_k(x_k), w_k) \right\}$$

with $\alpha < 1$, and for some M , we have $|g(x, u, w)| \leq M$ for all (x, u, w)

- **Shorthand notation for DP mappings** (operate on functions of state to produce other functions)

$$(TJ)(x) = \min_{u \in U(x)} E_w \left\{ g(x, u, w) + \alpha J(f(x, u, w)) \right\}, \quad \forall x$$

TJ is the optimal cost function for the one-stage problem with stage cost g and terminal cost αJ

- For any stationary policy μ

$$(T_\mu J)(x) = E_w \left\{ g(x, \mu(x), w) + \alpha J(f(x, \mu(x), w)) \right\}, \quad \forall x$$

MDP - TRANSITION PROBABILITY NOTATION

- We will mostly assume the system is an n -state (controlled) Markov chain
- We will often switch to Markov chain notation
 - States $i = 1, \dots, n$ (instead of x)
 - Transition probabilities $p_{i_k i_{k+1}}(u_k)$ [instead of $x_{k+1} = f(x_k, u_k, w_k)$]
 - Stage cost $g(i_k, u_k, i_{k+1})$ [instead of $g(x_k, u_k, w_k)$]
 - Cost functions $J = (J(1), \dots, J(n))$ (vectors in \mathfrak{R}^n)
- Cost of a policy $\pi = \{\mu_0, \mu_1, \dots\}$

$$J_\pi(i) = \lim_{N \rightarrow \infty} E_{\substack{i_k \\ k=1,2,\dots}} \left\{ \sum_{k=0}^{N-1} \alpha^k g(i_k, \mu_k(i_k), i_{k+1}) \mid i_0 = i \right\}$$

- Shorthand notation for DP mappings

$$(TJ)(i) = \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J(j)), \quad i = 1, \dots, n,$$

$$(T_\mu J)(i) = \sum_{j=1}^n p_{ij}(\mu(i)) (g(i, \mu(i), j) + \alpha J(j)), \quad i = 1, \dots, n$$

“SHORTHAND” THEORY – A SUMMARY

- **Bellman’s equation:** $J^* = TJ^*$, $J_\mu = T_\mu J_\mu$ or

$$J^*(i) = \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J^*(j)), \quad \forall i$$

$$J_\mu(i) = \sum_{j=1}^n p_{ij}(\mu(i)) (g(i, \mu(i), j) + \alpha J_\mu(j)), \quad \forall i$$

- **Optimality condition:**

$$\mu: \text{optimal} \quad \iff \quad T_\mu J^* = TJ^*$$

i.e.,

$$\mu(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J^*(j)), \quad \forall i$$

THE TWO MAIN ALGORITHMS: VI AND PI

- **Value iteration:** For any $J \in \mathbb{R}^n$

$$J^*(i) = \lim_{k \rightarrow \infty} (T^k J)(i), \quad \forall i = 1, \dots, n$$

- **Policy iteration:** Given μ^k
 - **Policy evaluation:** Find J_{μ^k} by solving

$$J_{\mu^k}(i) = \sum_{j=1}^n p_{ij}(\mu^k(i)) (g(i, \mu^k(i), j) + \alpha J_{\mu^k}(j)), \quad i = 1, \dots, n$$

$$\text{or } J_{\mu^k} = T_{\mu^k} J_{\mu^k}$$

- **Policy improvement:** Let μ^{k+1} be such that

$$\mu^{k+1}(i) \in \arg \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J_{\mu^k}(j)), \quad \forall i$$

$$\text{or } T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$$

- **Policy evaluation is equivalent to solving an $n \times n$ linear system of equations**
- **For large n , exact PI is out of the question.** We use instead optimistic PI (policy evaluation with a few VIs)

GENERAL ORIENTATION TO ADP

- ADP (late 80s - present) is a breakthrough methodology that **allows the application of DP to problems with many or infinite number of states.**
- Other names for ADP are:
 - **“reinforcement learning”** (RL).
 - **“neuro-dynamic programming”** (NDP).
 - **“adaptive dynamic programming”** (ADP).
- We will mainly adopt an n -state discounted model (the easiest case - but think of HUGE n).
- Extensions to other DP models (continuous space, continuous-time, not discounted) are possible (but more quirky). We will set aside for later.
- There are many approaches:
 - Problem approximation
 - Simulation-based approaches (we will focus on these)
- Simulation-based methods are of three types:
 - Rollout (we will not discuss further)
 - Approximation in value space
 - Approximation in policy space

WHY DO WE USE SIMULATION?

- One reason: **Computational complexity advantage** in computing sums/expectations involving a very large number of terms

- **Any sum**

$$\sum_{i=1}^n a_i$$

can be written as an expected value:

$$\sum_{i=1}^n a_i = \sum_{i=1}^n \xi_i \frac{a_i}{\xi_i} = E_{\xi} \left\{ \frac{a_i}{\xi_i} \right\},$$

where ξ is any prob. distribution over $\{1, \dots, n\}$

- It can be approximated by generating many samples $\{i_1, \dots, i_k\}$ from $\{1, \dots, n\}$, according to distribution ξ , and Monte Carlo averaging:

$$\sum_{i=1}^n a_i = E_{\xi} \left\{ \frac{a_i}{\xi_i} \right\} \approx \frac{1}{k} \sum_{t=1}^k \frac{a_{i_k}}{\xi_{i_k}}$$

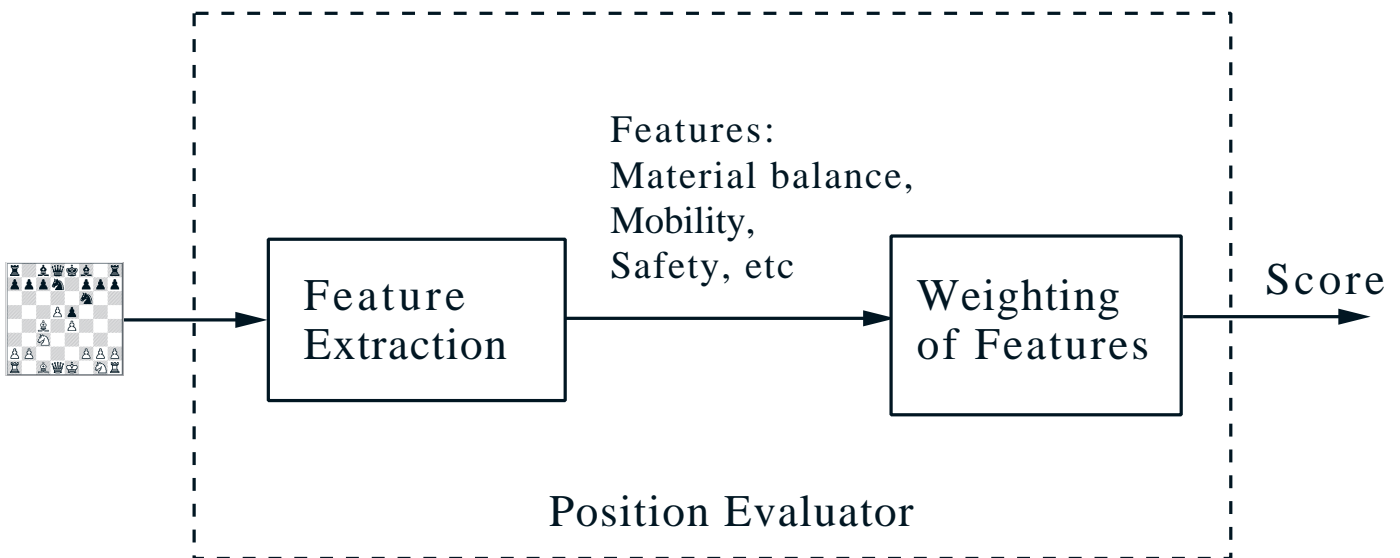
- Simulation is also convenient when **an analytical model of the system is unavailable**, but a simulation/computer model is possible.

APPROXIMATION IN VALUE SPACE

- Approximate J^* or J_μ from a parametric class $\tilde{J}(i; r)$ where i is the current state and $r = (r_1, \dots, r_m)$ is a vector of “tunable” scalar weights
- By adjusting r we can change the “shape” of \tilde{J} so that it is reasonably close to the true optimal J^*
- Two key issues:
 - The choice of parametric class $\tilde{J}(i; r)$ (the approximation architecture)
 - Method for tuning the weights (“training” the architecture)
- Success depends strongly on how these issues are handled, and on insight about the problem
- A simulator may be used, particularly when there is no mathematical model of the system (but there is a computer model)
- We will focus on simulation, but this is not the only possibility
- We may also use parametric approximation for Q -factors or cost function differences

APPROXIMATION ARCHITECTURES

- Divided in **linear and nonlinear** [i.e., linear or nonlinear dependence of $\tilde{J}(i; r)$ on r]
- Linear architectures are easier to train, but nonlinear ones (e.g., neural networks) are richer
- **Computer chess example:**
 - Think of **board position as state** and **move as control**
 - Uses a feature-based position evaluator that assigns a score (or approximate Q -factor) to each position/move



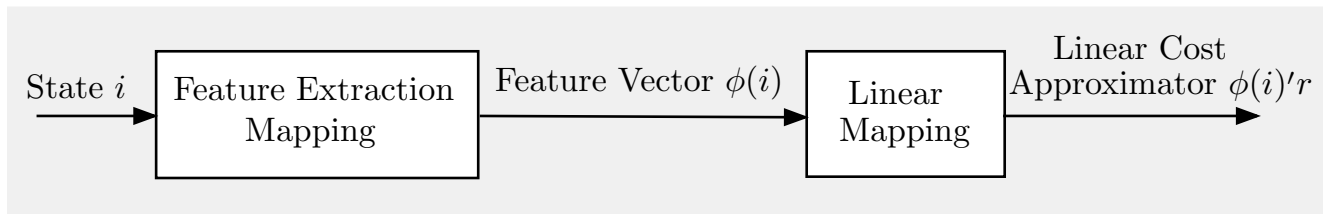
- Relatively few special features and weights, and multistep lookahead

LINEAR APPROXIMATION ARCHITECTURES

- Often, the features encode much of the nonlinearity inherent in the cost function approximated
- Then the approximation may be quite accurate without a complicated architecture (as an extreme example, the ideal feature is the true cost function)
- With well-chosen features, we can use a **linear architecture**: $\tilde{J}(i; r) = \phi(i)'r$, $i = 1, \dots, n$, or

$$\tilde{J}(r) = \Phi r$$

Φ : the matrix whose rows are $\phi(i)'$, $i = 1, \dots, n$



- This is approximation on the subspace

$$S = \{ \Phi r \mid r \in \mathbb{R}^s \}$$

spanned by the columns of Φ (basis functions)

- **Many examples of feature types**: Polynomial approximation, radial basis functions, kernels of all sorts, interpolation, and special domain specific features

ILLUSTRATIONS: POLYNOMIAL TYPE

- **Polynomial Approximation**, e.g., a quadratic approximating function. Let the state be $i = (i_1, \dots, i_q)$ (i.e., have q “dimensions”) and define

$$\phi_0(i) = 1, \quad \phi_k(i) = i_k, \quad \phi_{km}(i) = i_k i_m, \quad k, m = 1, \dots, q$$

Linear approximation architecture:

$$\tilde{J}(i; r) = r_0 + \sum_{k=1}^q r_k i_k + \sum_{k=1}^q \sum_{m=k}^q r_{km} i_k i_m,$$

where r has components r_0 , r_k , and r_{km} .

- **Interpolation**: A subset I of special/representative states is selected, and the parameter vector r has one component r_i per state $i \in I$. The approximating function is

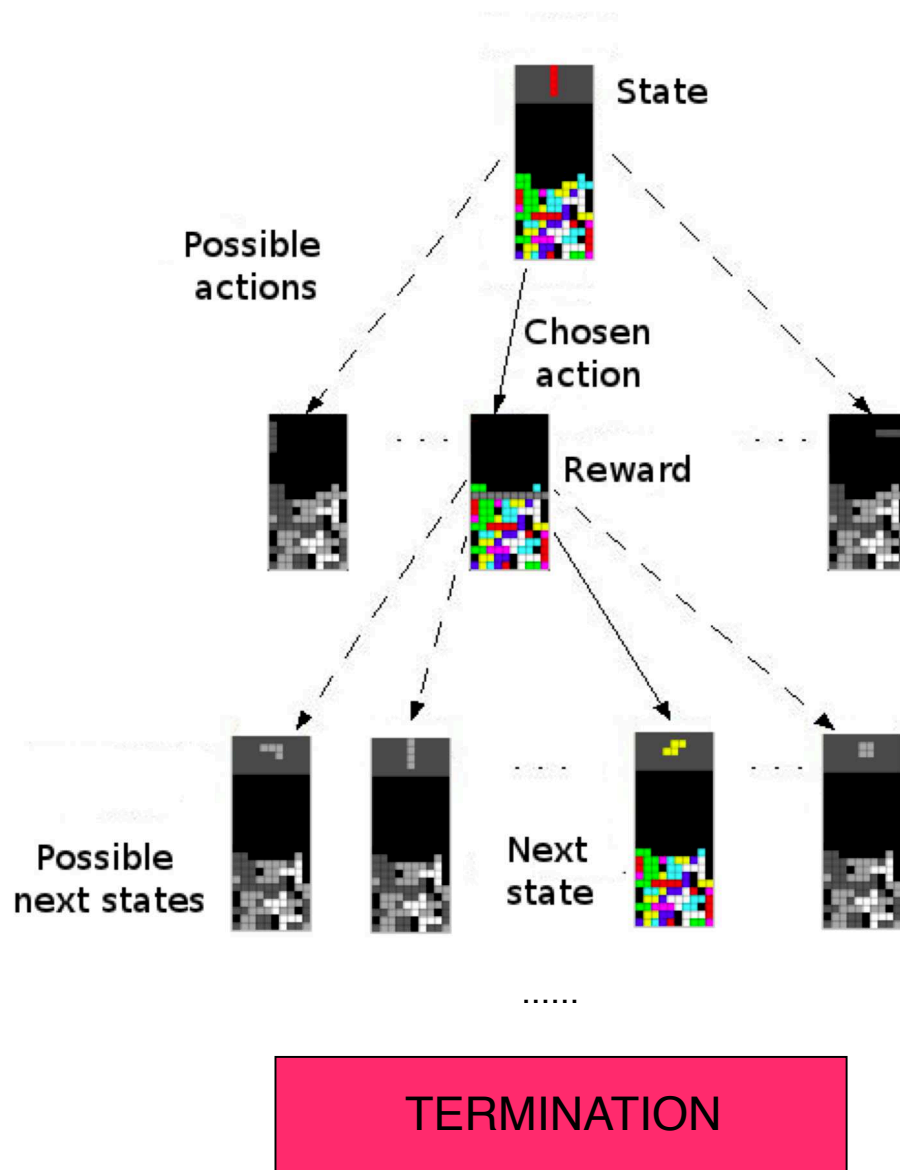
$$\tilde{J}(i; r) = r_i, \quad i \in I,$$

$\tilde{J}(i; r) =$ interpolation using the values at $i \in I$, $i \notin I$

For example, **piecewise constant**, **piecewise linear**, **more general polynomial interpolations**.

A DOMAIN SPECIFIC EXAMPLE

- **Tetris game** (used as testbed in competitions)



- $J^*(i)$: optimal score starting from position i
- **Number of states** $> 2^{200}$ (for 10×20 board)
- Success with just 22 features, readily recognized by tetris players as capturing important aspects of the board position (heights of columns, etc)

APPROXIMATION IN POLICY SPACE

- A brief discussion; we will return to it later.
- Use parametrization $\mu(i; r)$ of policies with a vector $r = (r_1, \dots, r_s)$. Examples:
 - Polynomial, e.g., $\mu(i; r) = r_1 + r_2 \cdot i + r_3 \cdot i^2$
 - Linear feature-based

$$\mu(i; r) = \phi_1(i) \cdot r_1 + \phi_2(i) \cdot r_2$$

- Optimize the cost over r . For example:
 - Each value of r defines a stationary policy, with cost starting at state i denoted by $\tilde{J}(i; r)$.
 - Let (p_1, \dots, p_n) be some probability distribution over the states, and minimize over r

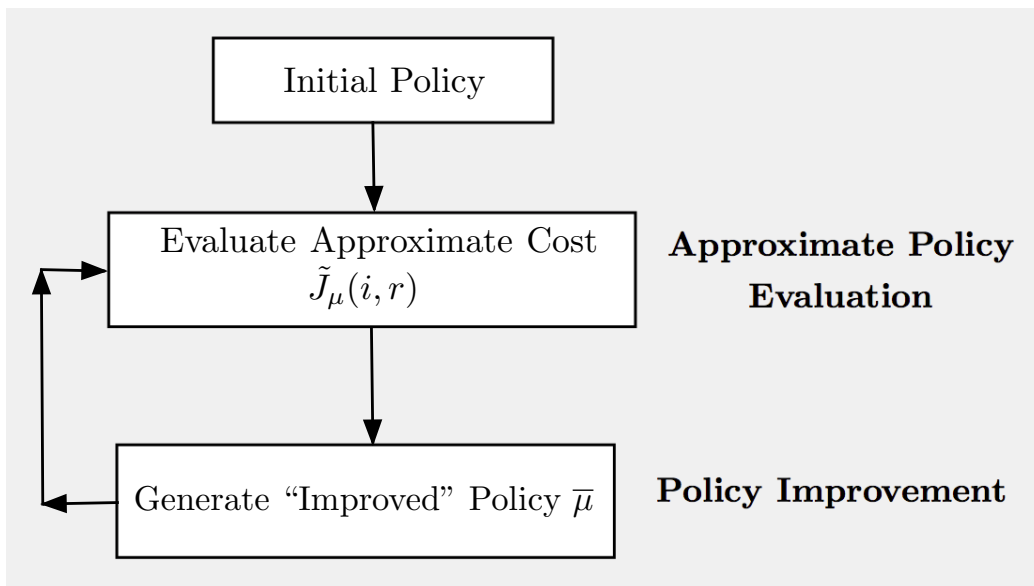
$$\sum_{i=1}^n p_i \tilde{J}(i; r)$$

- Use a random search, gradient, or other method
- A special case: The parameterization of the policies is indirect, through a cost approximation architecture \hat{J} , i.e.,

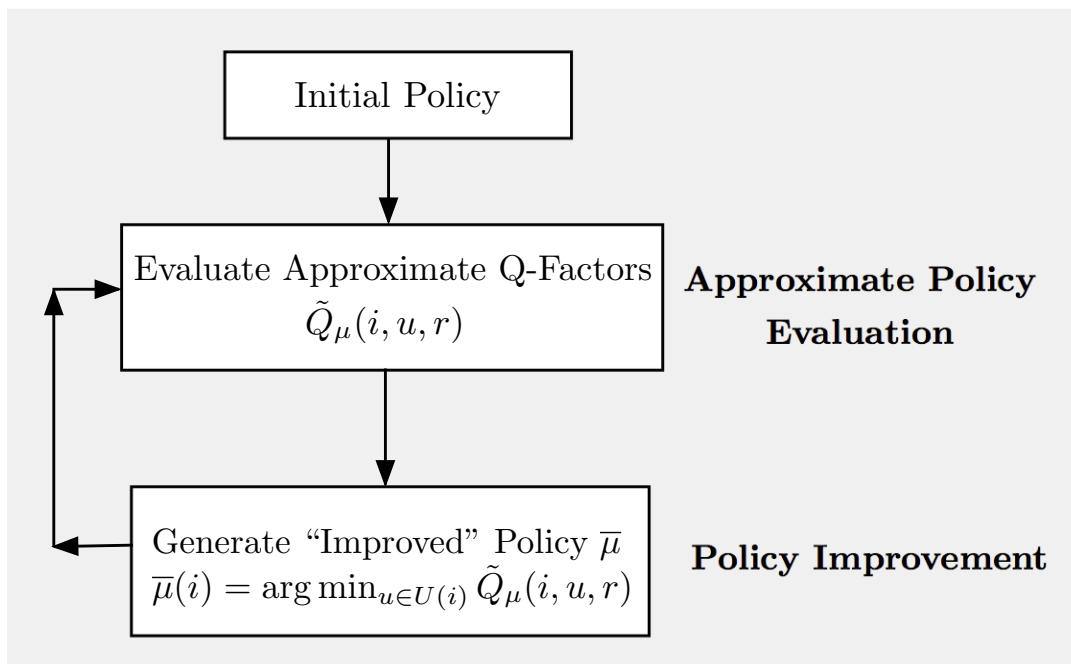
$$\mu(i; r) \in \arg \min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha \hat{J}(j; r))$$

APPROX. IN VALUE SPACE - APPROX. J_μ OR Q_μ

- Use simulation to **approximate the cost J_μ** of the current policy μ
- Generate “improved” policy $\bar{\mu}$ by minimizing in (approx.) Bellman equation



- Alternatively **approximate the Q -factors of μ**



APPROX. IN VALUE SPACE - APPROX. J^* OR Q^*

- Approximation of the optimal cost function J^*
 - **Q-Learning**: Use a simulation algorithm to approximate the Q -factors

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

and the optimal costs

$$J^*(i) = \min_{u \in U(i)} Q^*(i, u)$$

- **Bellman Error approach**: Find r to

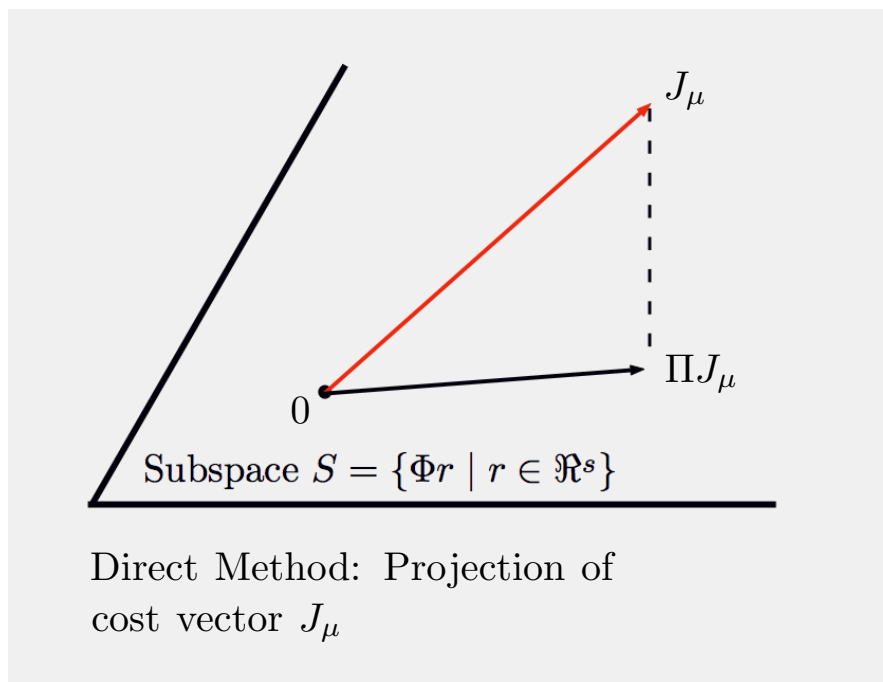
$$\min_r E_i \left\{ \left(\tilde{J}(i; r) - (T \tilde{J})(i; r) \right)^2 \right\}$$

where $E_i\{\cdot\}$ is taken with respect to some distribution over the states

- **Approximate Linear Programming** (we will not discuss here)
- Q-learning can also be used with approximations
- Q-learning and Bellman error approach can also be used for policy evaluation

DIRECT POLICY EVALUATION

- Approximate the cost of the current policy by using least squares and simulation-generated cost samples
- Amounts to projection of J_μ onto the approximation subspace



- Solution by least squares methods
- Regular and optimistic policy iteration
- Nonlinear approximation architectures may also be used

DIRECT EVALUATION BY SIMULATION

- **Projection by Monte Carlo Simulation:** Compute the projection ΠJ of a vector $J \in \mathbb{R}^n$ on subspace $S = \{\Phi r \mid r \in \mathbb{R}^s\}$, with respect to a weighted Euclidean norm $\|\cdot\|_\xi$

- Equivalently, find Φr^* , where

$$r^* = \arg \min_{r \in \mathbb{R}^s} \|\Phi r - J\|_\xi^2 = \arg \min_{r \in \mathbb{R}^s} \sum_{i=1}^n \xi_i (\phi(i)'r - J(i))^2$$

- Setting to 0 the gradient at r^* ,

$$r^* = \left(\sum_{i=1}^n \xi_i \phi(i) \phi(i)'\right)^{-1} \sum_{i=1}^n \xi_i \phi(i) J(i)$$

- **Generate samples $\{(i_1, J(i_1)), \dots, (i_k, J(i_k))\}$**

- Approximate by simulation the two “expected values” with **low-dimensional calculations**

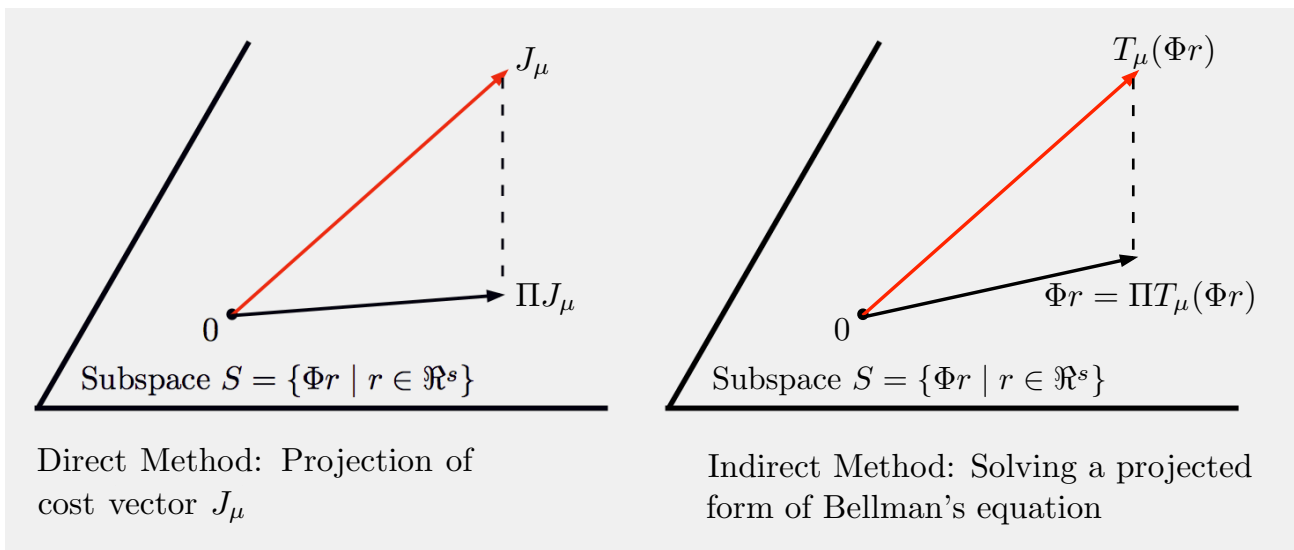
$$\hat{r}_k = \left(\sum_{t=1}^k \phi(i_t) \phi(i_t)'\right)^{-1} \sum_{t=1}^k \phi(i_t) J(i_t)$$

- Equivalent least squares alternative:

$$\hat{r}_k = \arg \min_{r \in \mathbb{R}^s} \sum_{t=1}^k (\phi(i_t)'r - J(i_t))^2$$

INDIRECT POLICY EVALUATION

- An example: **Galerkin approximation**
- Solve the **projected equation** $\Phi r = \Pi T_\mu(\Phi r)$ where Π is projection w/ respect to a suitable weighted Euclidean norm



- Solution methods that use simulation (to manage calculation of Π)
 - TD(λ): Stochastic iterative algorithm for solving $\Phi r = \Pi T_\mu(\Phi r)$
 - LSTD(λ): Solves a simulation-based approximation w/ a standard solver
 - LSPE(λ): A simulation-based form of **projected value iteration**; essentially
$$\Phi r_{k+1} = \Pi T_\mu(\Phi r_k) + \text{simulation noise}$$

BELLMAN EQUATION ERROR METHOD

- Another example of indirect approximate policy evaluation:

$$\min_r \|\Phi r - T_\mu(\Phi r)\|_\xi^2 \quad (*)$$

where $\|\cdot\|_\xi$ is Euclidean norm, weighted with respect to some distribution ξ

- It is closely related to the projected equation/Galerkin approach (with a special choice of projection norm)

- **Several ways to implement projected equation and Bellman error methods by simulation.** They involve:

- Generating many random samples of states i_k using the distribution ξ
- Generating many samples of transitions (i_k, j_k) using the policy μ
- Form a simulation-based approximation of the optimality condition for projection problem or problem (*) (use sample averages in place of inner products)

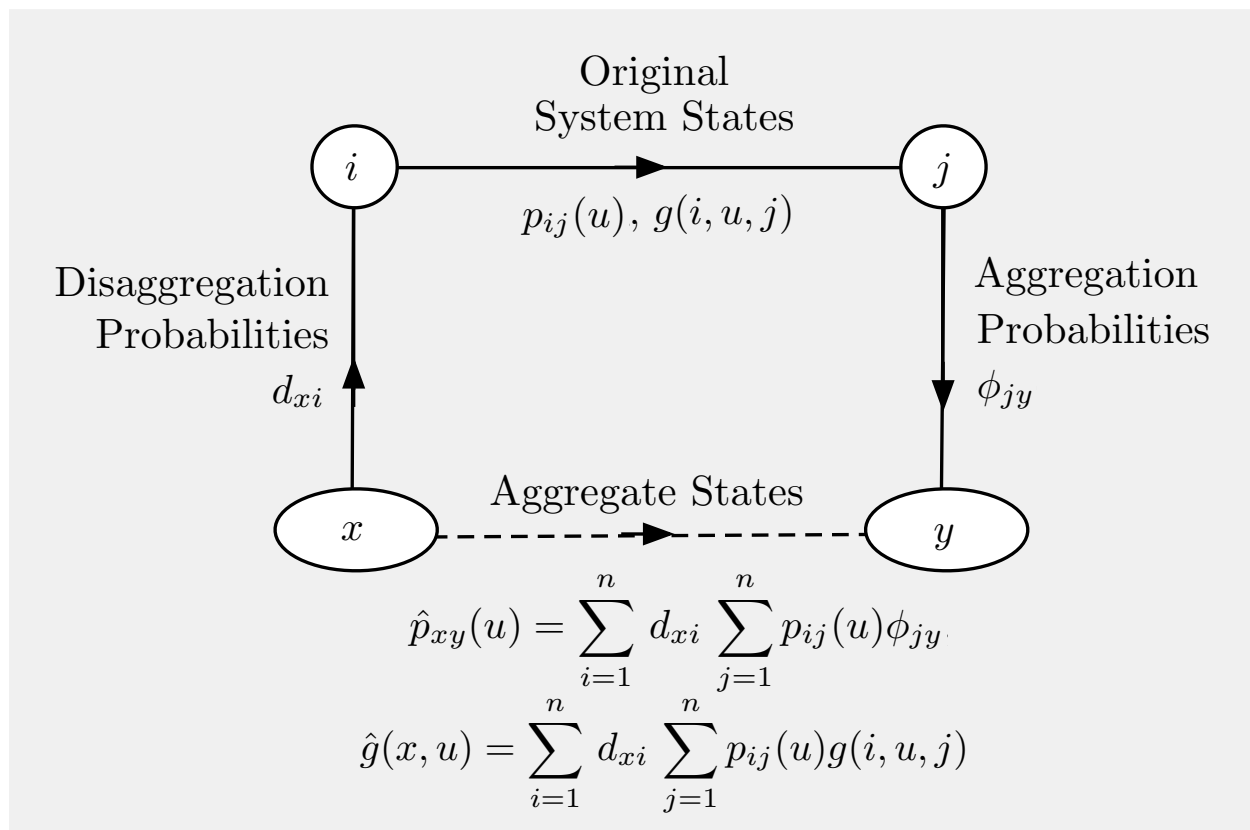
- Solve a Monte-Carlo approximation of the optimality condition
- Issues for indirect methods: **How to generate the samples? How to calculate r efficiently?**

ANOTHER INDIRECT METHOD: AGGREGATION

- **Aggregation approximation:** Solve

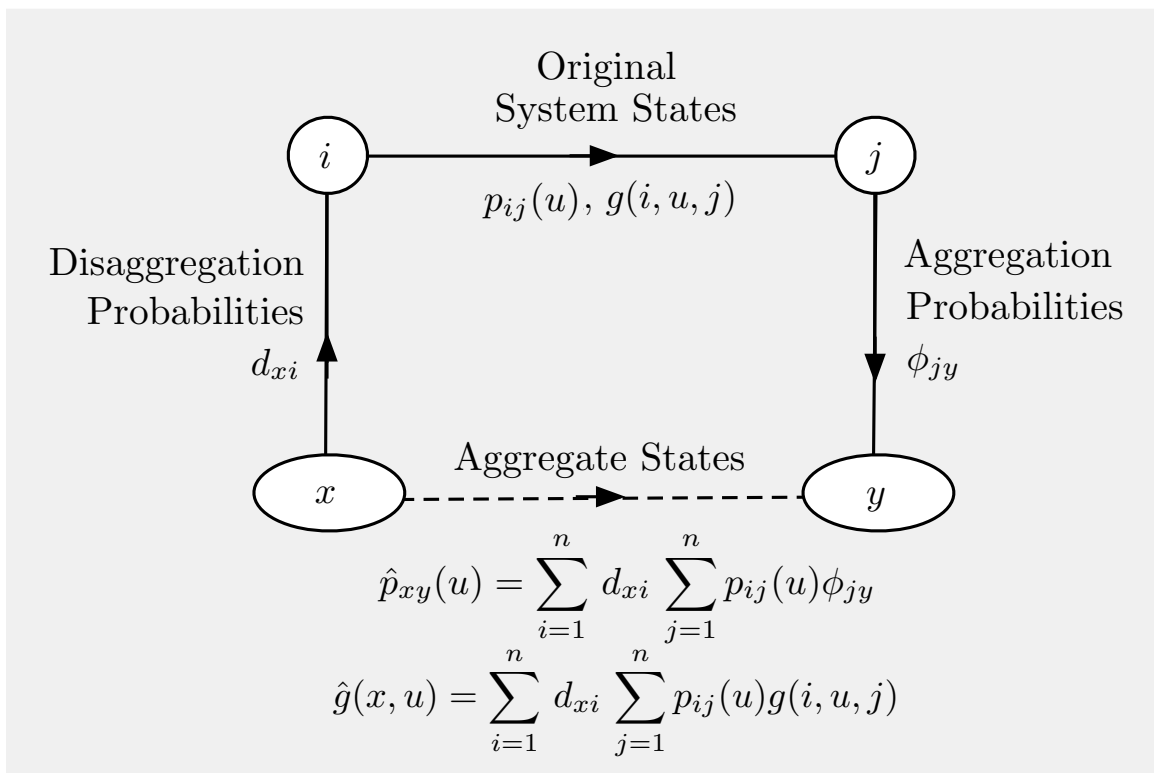
$$\Phi r = \Phi D T_{\mu}(\Phi r)$$

where the rows of D and Φ are prob. distributions (e.g., D and Φ “aggregate” rows and columns of the linear system $J = T_{\mu}J$)



- Several different choices of D and Φ
- Compare with projected equation $\Phi r = \Pi T_{\mu}(\Phi r)$
- ΦD is a projection in some interesting cases

AGGREGATION AS PROBLEM APPROXIMATION



- Aggregation can be viewed as a systematic approach for problem approximation. Main elements:
 - Solve (exactly or approximately) the “aggregate” problem by any kind of VI or PI method (including simulation-based methods)
 - Use the optimal cost of the aggregate problem to approximate the optimal cost of the original problem
- Because an exact PI algorithm is used to solve the approximate/aggregate problem the method behaves more regularly than the projected equation approach

THEORETICAL BASIS OF APPROXIMATE PI

- If policies are approximately evaluated using an approximation architecture such that

$$\max_i |\tilde{J}(i, r_k) - J_{\mu^k}(i)| \leq \delta, \quad k = 0, 1, \dots$$

- If policy improvement is also approximate,

$$\max_i |(T_{\mu^{k+1}} \tilde{J})(i, r_k) - (T \tilde{J})(i, r_k)| \leq \epsilon, \quad k = 0, 1, \dots$$

- **Error bound:** The sequence $\{\mu^k\}$ generated by approximate policy iteration satisfies

$$\limsup_{k \rightarrow \infty} \max_i (J_{\mu^k}(i) - J^*(i)) \leq \frac{\epsilon + 2\alpha\delta}{(1 - \alpha)^2}$$

- **Typical practical behavior:** The method makes steady progress up to a point and then the iterates J_{μ^k} oscillate within a neighborhood of J^* .

THE ISSUE OF EXPLORATION

- To evaluate a policy μ , we need to generate cost samples using that policy - this biases the simulation by underrepresenting states that are unlikely to occur under μ
- Cost-to-go estimates of underrepresented states may be highly inaccurate
- This seriously impacts the improved policy $\bar{\mu}$
- This is known as **inadequate exploration** - a particularly acute difficulty when the randomness embodied in the transition probabilities is “relatively small” (e.g., a deterministic system)
- Some remedies:
 - **Frequently restart the simulation** and ensure that the initial states employed form a rich and representative subset
 - Occasionally generate transitions that **use a randomly selected control** rather than the one dictated by the policy μ
 - Other methods: **Use two Markov chains** (one is the chain of the policy and is used to generate the transition sequence, the other is used to generate the state sequence).

APPROXIMATING Q-FACTORS

- Given $\tilde{J}(i; r)$, policy improvement requires a **model** [knowledge of $p_{ij}(u)$ for all controls $u \in U(i)$]
- **Model-free alternative:** Approximate Q -factors

$$\tilde{Q}(i, u; r) \approx \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J_{\mu}(j))$$

and use for policy improvement the minimization

$$\bar{\mu}(i) \in \arg \min_{u \in U(i)} \tilde{Q}(i, u; r)$$

- r is an adjustable parameter vector and $\tilde{Q}(i, u; r)$ is a parametric architecture, such as

$$\tilde{Q}(i, u; r) = \sum_{m=1}^s r_m \phi_m(i, u)$$

- **We can adapt any of the cost approximation approaches**, e.g., projected equations, aggregation
- Use the Markov chain with states (i, u) , so $p_{ij}(\mu(i))$ is the transition prob. to $(j, \mu(i))$, 0 to other (j, u')
- **Major concern:** Acutely diminished exploration

STOCHASTIC ALGORITHMS: GENERALITIES

- Consider solution of a linear equation $x = b + Ax$ by using m simulation samples $b + w_k$ and $A + W_k$, $k = 1, \dots, m$, where w_k, W_k are random, e.g., “simulation noise”

- Think of $x = b + Ax$ as approximate policy evaluation (projected or aggregation equations)

- **Stoch. approx. (SA) approach:** For $k = 1, \dots, m$

$$x_{k+1} = (1 - \gamma_k)x_k + \gamma_k((b + w_k) + (A + W_k)x_k)$$

- **Monte Carlo estimation (MCE) approach:** Form Monte Carlo estimates of b and A

$$b_m = \frac{1}{m} \sum_{k=1}^m (b + w_k), \quad A_m = \frac{1}{m} \sum_{k=1}^m (A + W_k)$$

Then solve $x = b_m + A_m x$ by matrix inversion

$$x_m = (1 - A_m)^{-1} b_m$$

or iteratively

- **TD(λ) and Q-learning are SA methods**

- **LSTD(λ) and LSPE(λ) are MCE methods**

COSTS OR COST DIFFERENCES?

- Consider the exact policy improvement process. To compare two controls u and u' at x , we need

$$E\{g(x, u, w) - g(x, u', w) + \alpha(J_\mu(\bar{x}) - J_\mu(\bar{x}'))\}$$

where $\bar{x} = f(x, u, w)$ and $\bar{x}' = f(x, u', w)$

- Approximate $J_\mu(\bar{x})$ or

$$D_\mu(\bar{x}, \bar{x}') = J_\mu(\bar{x}) - J_\mu(\bar{x}')?$$

- **Approximating $D_\mu(\bar{x}, \bar{x}')$ avoids “noise differencing”**. This can make a big difference
- **Important point:** D_μ satisfies a Bellman equation for a system with “state” (x, x')

$$D_\mu(x, x') = E\{G_\mu(x, x', w) + \alpha D_\mu(\bar{x}, \bar{x}')\}$$

where $\bar{x} = f(x, \mu(x), w)$, $\bar{x}' = f(x, \mu(x'), w)$ and

$$G_\mu(x, x', w) = g(x, \mu(x), w) - g(x, \mu(x'), w)$$

- D_μ can be learned by the standard methods (TD, LSTD, LSPE, Bellman error, aggregation, etc). This is known as **differential training**.