APPROXIMATE DYNAMIC PROGRAMMING

LECTURE 3

LECTURE OUTLINE

- Review of theory and algorithms for discounted DP
- MDP and stochastic shortest path problems (briefly)
- Introduction to approximation in policy and value space
- 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

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

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

$$J_\pi(x_0) = \lim_{N\to\infty} \mathop{E}_{\substack{w_k \\ k=0,1,\ldots}} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(x_k, \mu_k(x_k), w_k\big) \right\}$$

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)

$$(TJ)(x) = \min_{u\in U(x)} \mathop{E}_{w} \left\{ g(x,u,w) + \alpha J\big(f(x,u,w)\big) \right\}, \ \forall\, x$$

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

- For any stationary policy μ

$$(T_\mu J)(x) = \mathop{E}_{w} \left\{ g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big) \right\}, \ \forall\, x$$

MDP - TRANSITION PROBABILITY NOTATION

- Assume the system is an n-state (controlled) Markov chain
- Change to Markov chain notation
  - States $i = 1, \ldots, n$ (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 of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$

$$J_\pi(i) = \lim_{N\to\infty} \mathop{E}_{\substack{i_k\\ k=1,2,\ldots}} \left\{ \sum_{k=0}^{N-1} \alpha^k g\big(i_k, \mu_k(i_k), i_{k+1}\big) \mid i_0 = i \right\}$$

- Shorthand notation for DP mappings

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

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

“SHORTHAND” THEORY – A SUMMARY

- Cost function expressions [with $J_0(i) \equiv 0$]

$$J_\pi(i) = \lim_{k\to\infty} (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_k} J_0)(i), \quad J_\mu(i) = \lim_{k\to\infty} (T_\mu^k J_0)(i)$$

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

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

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

- Optimality condition:

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

i.e.,

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

THE TWO MAIN ALGORITHMS: VI AND PI

- Value iteration: For any $J \in \Re^n$

$$J^*(i) = \lim_{k\to\infty} (T^k J)(i), \qquad \forall\ i = 1, \ldots, n$$

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

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

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

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

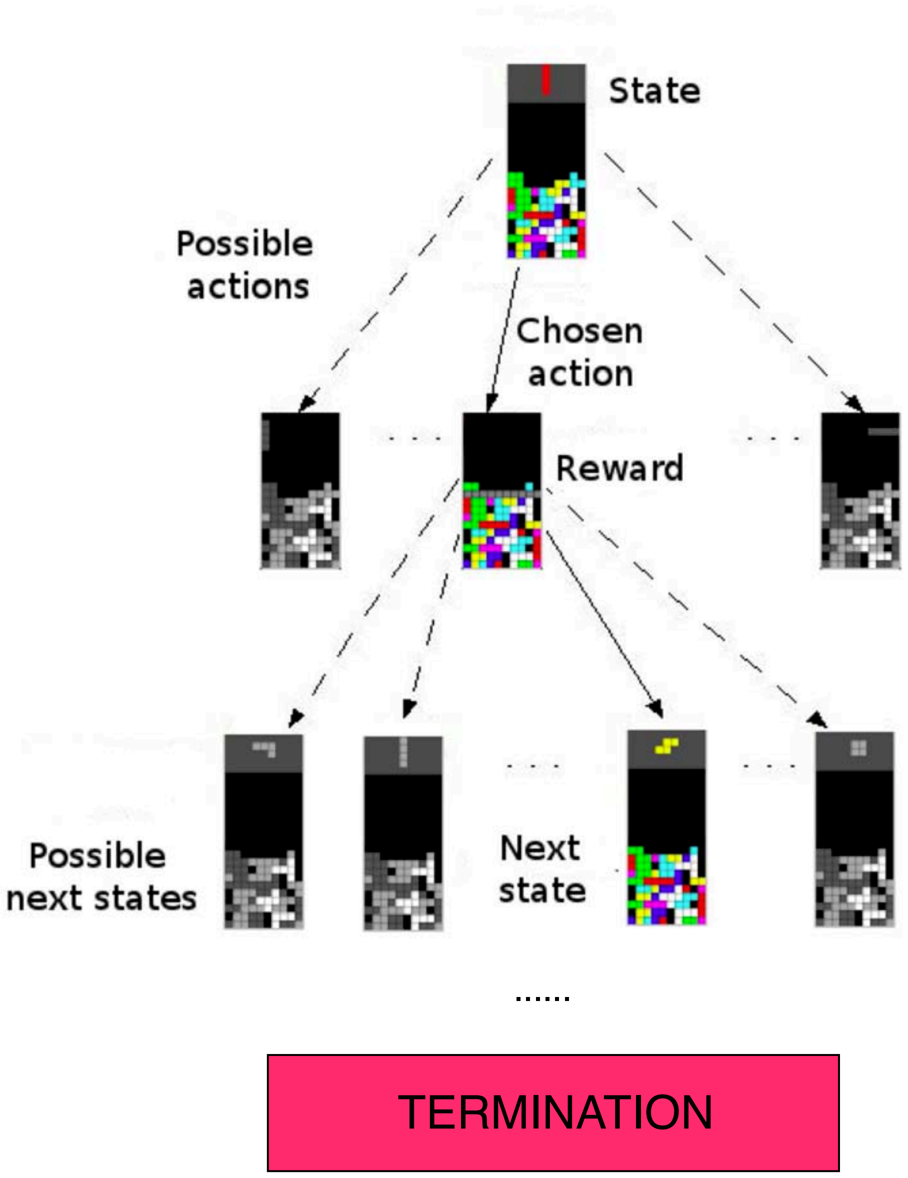
$$\mu^{k+1}(i) \in \arg \min_{u\in U(i)} \sum_{j=1}^{n} p_{ij}(u)\Big(g(i, u, j) + \alpha J_{\mu^k}(j)\Big), \quad \forall\ i$$

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 (even though it terminates finitely)

STOCHASTIC SHORTEST PATH (SSP) PROBLEMS

- Involves states $i = 1, \ldots, n$ plus a special cost-free and absorbing termination state t
- Objective: Minimize the total (undiscounted) cost. Aim: Reach t at minimum expected cost
- An example: Tetris



SSP THEORY

- SSP problems provide a "soft boundary" between the easy finite-state discounted problems and the hard undiscounted problems.
  - They share features of both.
  - Some of the nice theory is recovered because of the termination state.
- Definition: A proper policy is a stationary policy that leads to t with probability 1
- If all stationary policies are proper, T and T_μ are contractions with respect to a common weighted sup-norm
- The entire analytical and algorithmic theory for discounted problems goes through if all stationary policies are proper (we will assume this)
- There is a strong theory even if there are improper policies (but they should be assumed to be nonoptimal - see the textbook)

GENERAL ORIENTATION TO ADP

- 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 set aside for later.
- There are many approaches:
  - Manual/trial-and-error approach
  - Problem approximation
  - Simulation-based approaches (we will focus on these): "neuro-dynamic programming" or "reinforcement learning".
- Simulation is essential for large state spaces because of its (potential) computational complexity advantage in computing sums/expectations involving a very large number of terms.
- Simulation also comes in handy when an analytical model of the system is unavailable, but a simulation/computer model is possible.
- Simulation-based methods are of three types:
  - Rollout (we will not discuss further)
  - Approximation in value space
  - Approximation in policy space

APPROXIMATION IN VALUE SPACE

- Approximate J^* or J_μ from a parametric class $\tilde{J}(i,r)$ where i is the current state and $r = (r_1, \ldots, r_m)$ is a vector of "tunable" scalars 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).

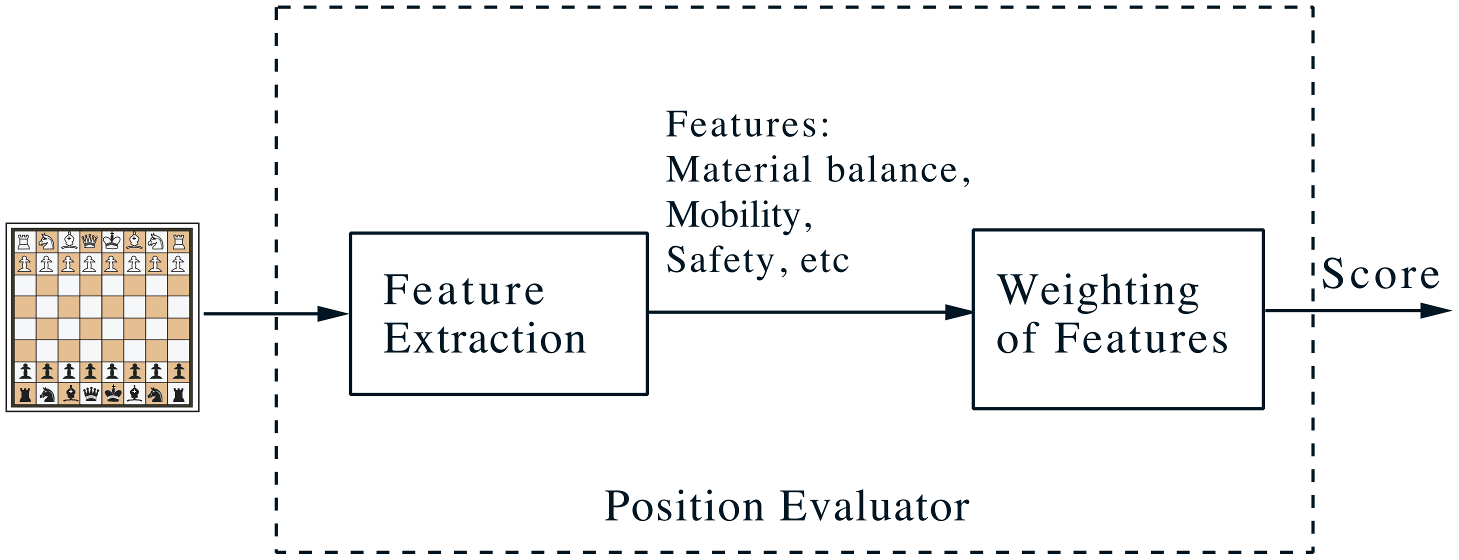
- Successful application strongly depends on how these issues are handled, and on insight about the problem.

- 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 [e.g., $\tilde{J}(i,r)$ may be a lower bound approximation based on relaxation, or other problem approximation]

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 non-linear ones (e.g., neural networks) are richer.
- Computer chess example: Uses 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.
- In chess, most often the training is done by trial and error.

LINEAR APPROXIMATION ARCHITECTURES

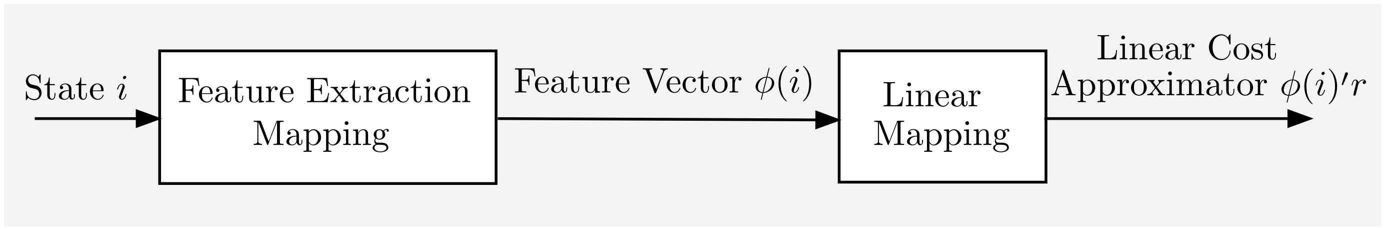
- Ideally, the features encode much of the nonlinearity inherent in the cost-to-go approximated

- Then the approximation may be quite accurate without a complicated architecture.

- With well-chosen features, we can use a linear architecture: $\tilde{J}(i, r) = \phi(i)'r$, $i = 1, \ldots, n$, or more compactly

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

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


State i
Feature Extraction Mapping
Feature Vector $\phi(i)$
Linear Mapping
Linear Cost Approximator $\phi(i)'r$

- This is approximation on the subspace

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

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 problem-specific (as in chess and tetris)

APPROXIMATION IN POLICY SPACE

- A brief discussion; we will return to it at the end.
- We parameterize the set of policies by a vector $r = (r_1, \ldots, r_s)$ and we optimize the cost over r
- Discounted problem example:
  - Each value of r defines a stationary policy, with cost starting at state i denoted by $\tilde{J}(i; r)$.
  - Use a random search, gradient, or other method to minimize over r

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

    where $(p_1, \ldots, p_n)$ is some probability distribution over the states.

- 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.

APPROX. IN VALUE SPACE - APPROACHES

- Approximate PI (Policy evaluation/Policy improvement)
  - Uses simulation algorithms to approximate the cost J_μ of the current policy μ
  - Projected equation and aggregation approaches
- Approximation of the optimal cost function J^*
  - Q-Learning: Use a simulation algorithm to approximate the optimal costs $J^*(i)$ or the Q-factors

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

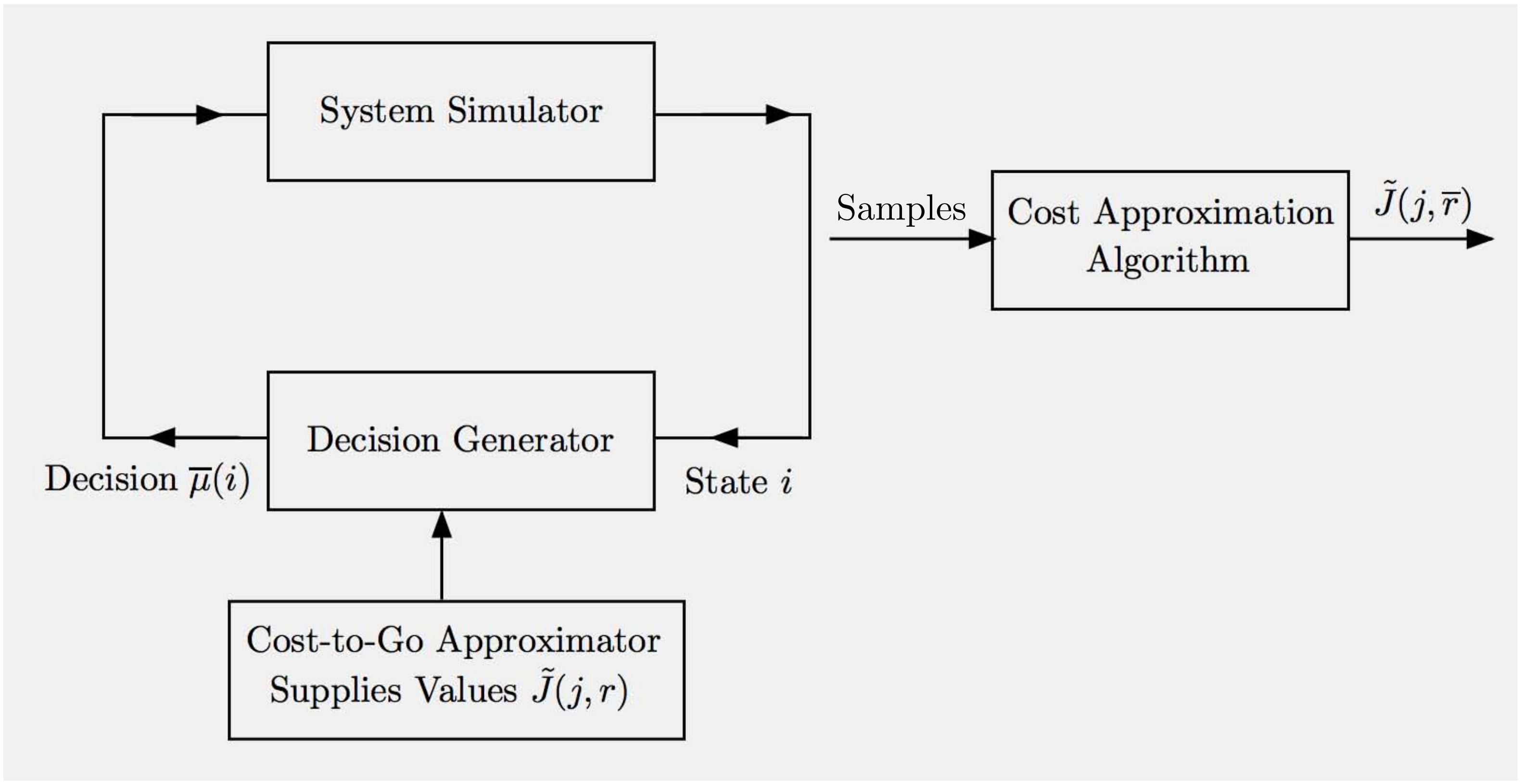
  - Bellman error approach: Find r to

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

    where $E_i\{\cdot\}$ is taken with respect to some distribution
  - Approximate LP (we will not discuss here)

APPROXIMATE POLICY ITERATION

- General structure


System Simulator
Samples
Cost Approximation Algorithm
$\tilde J(j,\overline{r})$
Decision Generator
Decision $\overline{\mu}(i)$
State i
Cost-to-Go Approximator Supplies Values $\tilde J(j,r)$

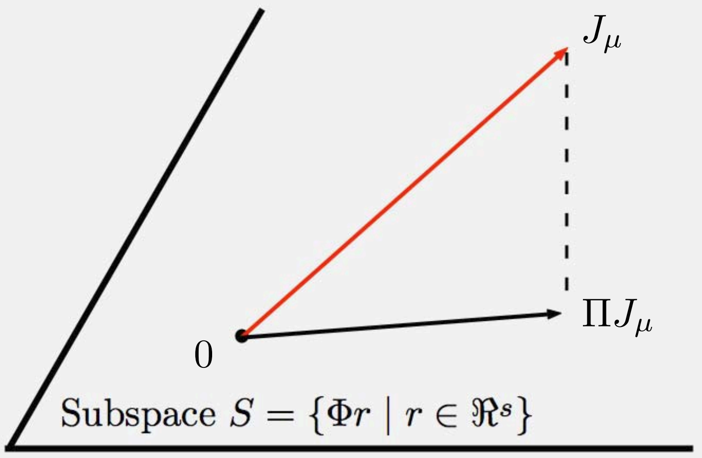
- $\tilde J(j,r)$ is the cost approximation for the preceding policy, used by the decision generator to compute the current policy $\overline{\mu}$ [whose cost is approximated by $\tilde J(j,\overline{r})$ using simulation]

- There are several cost approximation/policy evaluation algorithms

- There are several important issues relating to the design of each block (to be discussed in the future).

POLICY EVALUATION APPROACHES I

- 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

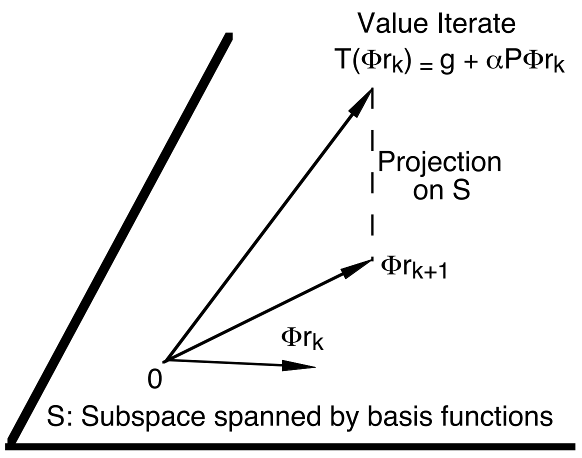


Direct Method: Projection of cost vector J_μ

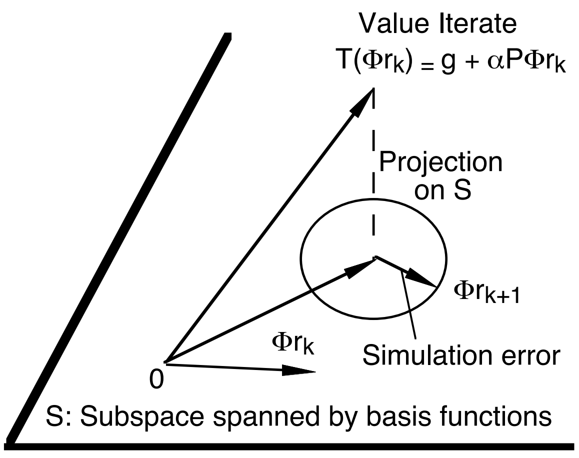
- Solution of the least squares problem by batch and incremental methods
- Regular and optimistic policy iteration
- Nonlinear approximation architectures may also be used

POLICY EVALUATION APPROACHES II

- Indirect policy evaluation



Projected Value Iteration (PVI)



Least Squares Policy Evaluation (LSPE)

- An example of indirect approach: 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
  - TD(λ): Stochastic iterative algorithm for solving $\Phi r = \Pi T_\mu(\Phi r)$
  - LSPE(λ): A simulation-based form of projected value iteration

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

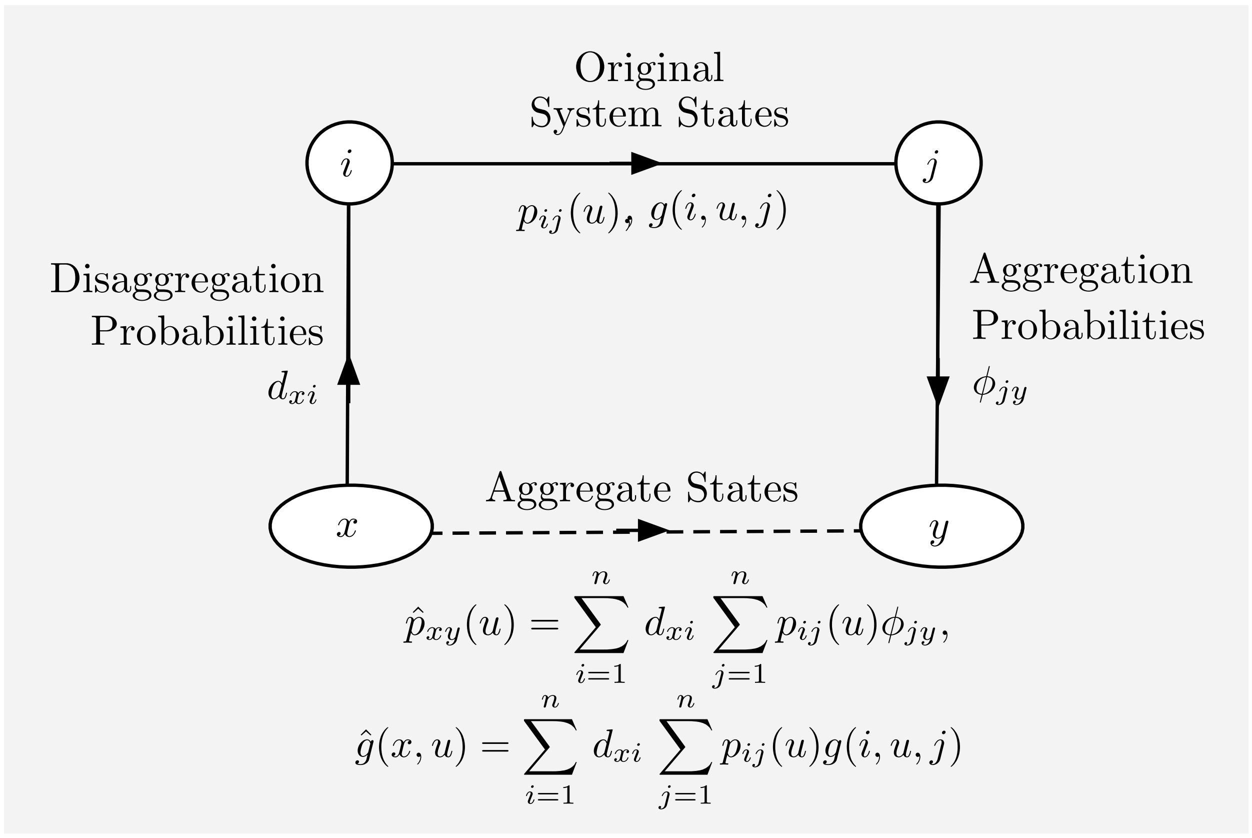
  - LSTD(λ): Solves a simulation-based approximation w/ a standard solver (Matlab)

POLICY EVALUATION APPROACHES III

- Aggregation approximation: Solve

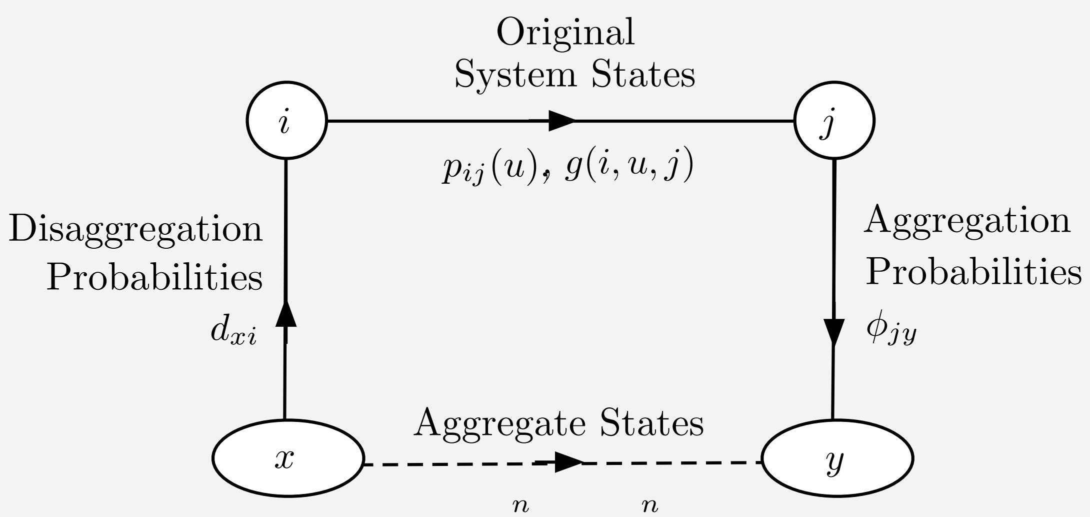
$$\Phi r = \Phi D T_\mu(\Phi r)$$

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


Original System States
i
j
$p_{ij}(u), g(i,u,j)$
Disaggregation Probabilities
d_{xi}
Aggregation Probabilities
ϕ_{jy}
Aggregate States
x
y
$\hat p_{xy}(u) = \sum_{i=1}^n d_{xi} \sum_{j=1}^n p_{ij}(u)\phi_{jy},$
$\hat g(x,u) = \sum_{i=1}^n d_{xi} \sum_{j=1}^n p_{ij}(u) g(i,u,j)$

- Several different choices of D and Φ.

POLICY EVALUATION APPROACHES IV



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

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

- Aggregation is 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

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

- If policy improvement is also approximate,

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

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

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

- 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 USE OF SIMULATION - AN EXAMPLE

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

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

- Setting to 0 the gradient at r^*,

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

- Approximate by simulation the two "expected values"

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

- Equivalent least squares alternative:

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

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.

- As a result, the cost-to-go estimates of these underrepresented states may be highly inaccurate.

- This seriously impacts the improved policy $\overline{\mu}$.

- 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).

- One possibility for adequate exploration: Frequently restart the simulation and ensure that the initial states employed form a rich and representative subset.

- Another possibility: Occasionally generate transitions that use a randomly selected control rather than the one dictated by the policy μ.

- Other methods, to be discussed later, 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

- The approach described so far for policy evaluation requires calculating expected values [and knowledge of $p_{ij}(u)$] for all controls $u \in U(i)$.
- Model-free alternative: Approximate Q-factors

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

and use for policy improvement the minimization

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

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