APPROXIMATE DYNAMIC PROGRAMMING

A SERIES OF LECTURES GIVEN AT

TSINGHUA UNIVERSITY

JUNE 2014

DIMITRI P. BERTSEKAS

Based on the books:

(1) "Neuro-Dynamic Programming," by DPB and J. N. Tsitsiklis, Athena Scientific, 1996

(2) "Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming," by DPB, Athena Scientific, 2012

(3) "Abstract Dynamic Programming," by DPB, Athena Scientific, 2013

http://www.athenasc.com

For a fuller set of slides, see

http://web.mit.edu/dimitrib/www/publ.html

*Athena is MIT's UNIX-based computing environment. OCW does not provide access to it.

APPROXIMATE DYNAMIC PROGRAMMING

BRIEF OUTLINE I

- **Our subject:**
  - Large-scale DP based on approximations and in part on simulation.
  - This has been a research area of great interest for the last 25 years known under various names (e.g., reinforcement learning, neuro-dynamic programming)
  - Emerged through an enormously fruitful cross-fertilization of ideas from artificial intelligence and optimization/control theory
  - Deals with control of dynamic systems under uncertainty, but applies more broadly (e.g., discrete deterministic optimization)
  - A vast range of applications in control theory, operations research, artificial intelligence, and beyond ...
  - The subject is broad with rich variety of theory/math, algorithms, and applications. Our focus will be mostly on algorithms ... less on theory and modeling

APPROXIMATE DYNAMIC PROGRAMMING

BRIEF OUTLINE II

- **Our aim:**
  - A state-of-the-art account of some of the major topics at a graduate level
  - Show how to use approximation and simulation to address the dual curses of DP: dimensionality and modeling
- **Our 6-lecture plan:**
  - Two lectures on exact DP with emphasis on infinite horizon problems and issues of large-scale computational methods
  - One lecture on general issues of approximation and simulation for large-scale problems
  - One lecture on approximate policy iteration based on temporal differences (TD)/projected equations/Galerkin approximation
  - One lecture on aggregation methods
  - One lecture on Q-learning, and other methods, such as approximation in policy space

APPROXIMATE DYNAMIC PROGRAMMING

LECTURE 1

LECTURE OUTLINE

- Introduction to DP and approximate DP
- Finite horizon problems
- The DP algorithm for finite horizon problems
- Infinite horizon problems
- Basic theory of discounted infinite horizon problems

DP AS AN OPTIMIZATION METHODOLOGY

- Generic optimization problem:

$$\min_{u \in U} g(u)$$

where u is the optimization/decision variable, $g(u)$ is the cost function, and U is the constraint set

- Categories of problems:
  - Discrete (U is finite) or continuous
  - Linear (g is linear and U is polyhedral) or nonlinear
  - Stochastic or deterministic: In stochastic problems the cost involves a stochastic parameter w, which is averaged, i.e., it has the form

$$g(u) = E_w\big\{G(u, w)\big\}$$

    where w is a random parameter.

- DP deals with multistage stochastic problems
  - Information about w is revealed in stages
  - Decisions are also made in stages and make use of the available information
  - Its methodology is “different”

BASIC STRUCTURE OF STOCHASTIC DP

- Discrete-time system

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

  - k: Discrete time
  - x_k: State; summarizes past information that is relevant for future optimization
  - u_k: Control; decision to be selected at time k from a given set
  - w_k: Random parameter (also called “disturbance” or “noise” depending on the context)
  - N: Horizon or number of times control is applied

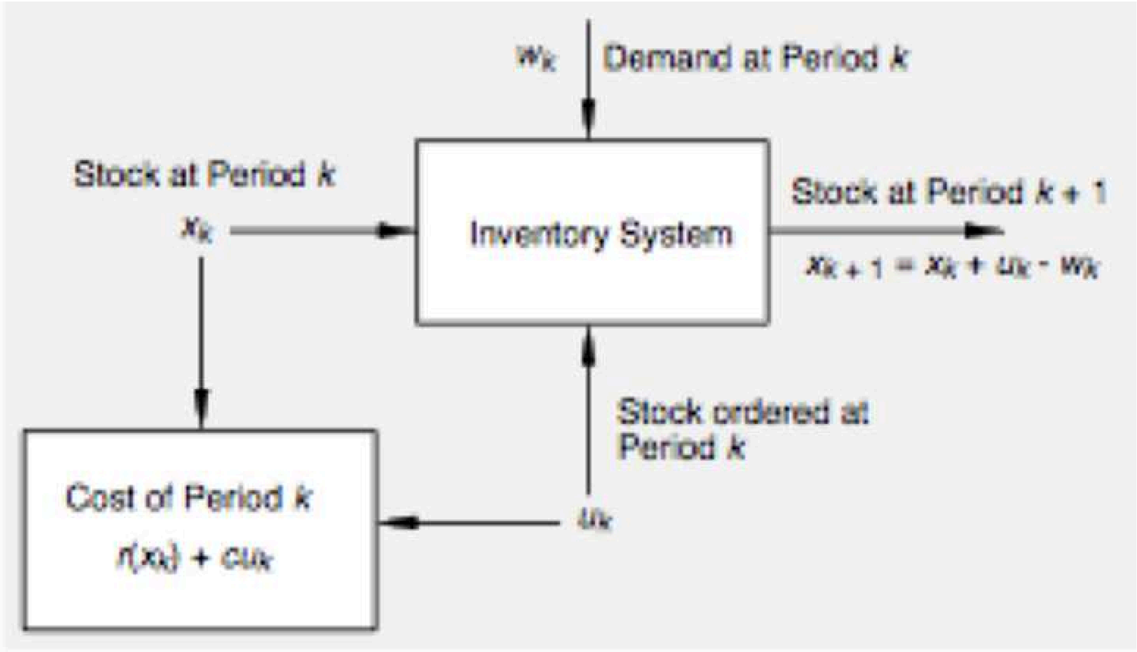
- Cost function that is additive over time

$$E\left\{ g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right\}$$

- Alternative system description: $P(x_{k+1} \mid x_k, u_k)$

$$x_{k+1} = w_k \quad \text{with } P(w_k \mid x_k, u_k) = P(x_{k+1} \mid x_k, u_k)$$

INVENTORY CONTROL EXAMPLE



- Discrete-time system

$$x_{k+1} = f_k(x_k, u_k, w_k) = x_k + u_k - w_k$$

- Cost function that is additive over time

$$E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\right\}$$
$$= E\left\{\sum_{k=0}^{N-1} \big(cu_k + r(x_k + u_k - w_k)\big)\right\}$$

ADDITIONAL ASSUMPTIONS

- Probability distribution of w_k does not depend on past values $w_{k-1}, \ldots, w_0$, but may depend on x_k and u_k
  - Otherwise past values of w, x, or u would be useful for future optimization

- The constraint set from which u_k is chosen at time k depends at most on x_k, not on prior x or u

- Optimization over policies (also called feedback control laws): These are rules/functions

$$u_k = \mu_k(x_k), \qquad k = 0, \ldots, N-1$$

that map state/inventory to control/order (closed-loop optimization, use of feedback)

- MAJOR DISTINCTION: We minimize over sequences of functions (mapping inventory to order)

$$\{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$$

NOT over sequences of controls/orders

$$\{u_0, u_1, \ldots, u_{N-1}\}$$

GENERIC FINITE-HORIZON PROBLEM

- System $x_{k+1} = f_k(x_k, u_k, w_k)$, $k = 0, \ldots, N-1$
- Control contraints $u_k \in U_k(x_k)$
- Probability distribution $P_k(\cdot \mid x_k, u_k)$ of w_k
- Policies $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, where μ_k maps states x_k into controls $u_k = \mu_k(x_k)$ and is such that $\mu_k(x_k) \in U_k(x_k)$ for all x_k
- Expected cost of π starting at x_0 is

$$J_\pi(x_0) = E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\right\}$$

- Optimal cost function

$$J^*(x_0) = \min_\pi J_\pi(x_0)$$

- Optimal policy π^* satisfies

$$J_{\pi^*}(x_0) = J^*(x_0)$$

When produced by DP, π^* is independent of x_0.

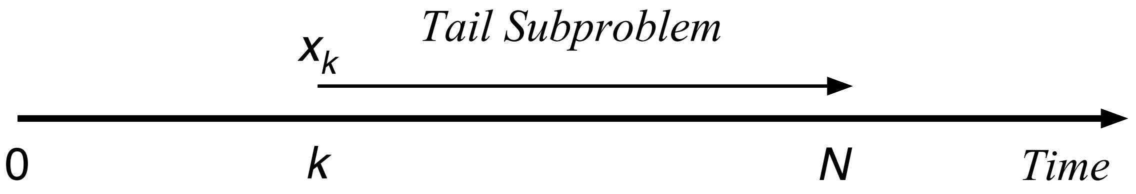
PRINCIPLE OF OPTIMALITY

- Let $\pi^* = \{\mu_0^*, \mu_1^*, \ldots, \mu_{N-1}^*\}$ be optimal policy

- Consider the "tail subproblem" whereby we are at x_k at time k and wish to minimize the "cost-to-go" from time k to time N

$$E\left\{g_N(x_N) + \sum_{\ell=k}^{N-1} g_\ell\big(x_\ell, \mu_\ell(x_\ell), w_\ell\big)\right\}$$

and the "tail policy" $\{\mu_k^*, \mu_{k+1}^*, \ldots, \mu_{N-1}^*\}$



- Principle of optimality: The tail policy is optimal for the tail subproblem (optimization of the future does not depend on what we did in the past)

- DP solves ALL the tail subroblems

- At the generic step, it solves ALL tail subproblems of a given time length, using the solution of the tail subproblems of shorter time length

DP ALGORITHM

- Computes for all k and states x_k:

  $J_k(x_k)$: opt. cost of tail problem starting at x_k

- Initial condition:

$$J_N(x_N) = g_N(x_N)$$

Go backwards, $k = N-1, \ldots, 0$, using

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} \mathop{E}_{w_k} \Big\{ g_k(x_k, u_k, w_k) + J_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\},$$

- To solve tail subproblem at time k minimize

kth-stage cost + Opt. cost of next tail problem starting from next state at time $k+1$

- Then $J_0(x_0)$, generated at the last step, is equal to the optimal cost $J^*(x_0)$. Also, the policy

$$\pi^* = \{\mu_0^*, \ldots, \mu_{N-1}^*\}$$

where $\mu_k^*(x_k)$ minimizes in the right side above for each x_k and k, is optimal

- Proof by induction

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
- The curse of modeling
  - Sometimes a simulator of the system is easier to construct than a model
- 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
- All of the above are motivations for approximation and simulation

A MAJOR IDEA: COST APPROXIMATION

- Use a policy computed from the DP equation where the optimal cost-to-go function J_{k+1} is replaced by an approximation $\tilde{J}_{k+1}$.

- Apply $\overline{\mu}_k(x_k)$, which attains the minimum in

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

- Some approaches:

(a) Problem Approximation: Use $\tilde{J}_k$ derived from a related but simpler problem

(b) Parametric Cost-to-Go Approximation: Use as $\tilde{J}_k$ a function of a suitable parametric form, whose parameters are tuned by some heuristic or systematic scheme (we will mostly focus on this)

– This is a major portion of Reinforcement Learning/Neuro-Dynamic Programming

(c) Rollout Approach: Use as $\tilde{J}_k$ the cost of some suboptimal policy, which is calculated either analytically or by simulation

ROLLOUT ALGORITHMS

- At each k and state x_k, use the control $\overline{\mu}_k(x_k)$ that minimizes in

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

where $\tilde{J}_{k+1}$ is the cost-to-go of some heuristic policy (called the base policy).

- Cost improvement property: The rollout algorithm achieves no worse (and usually much better) cost than the base policy starting from the same state.

- Main difficulty: Calculating $\tilde{J}_{k+1}(x)$ may be computationally intensive if the cost-to-go of the base policy cannot be analytically calculated.
  - May involve Monte Carlo simulation if the problem is stochastic.
  - Things improve in the deterministic case (an important application is discrete optimization).
  - Connection w/ Model Predictive Control (MPC).

INFINITE HORIZON PROBLEMS

- Same as the basic problem, but:
  - The number of stages is infinite.
  - The system is stationary.
- Total cost problems: Minimize

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

  - Discounted problems ($\alpha < 1$, bounded g)
  - Stochastic shortest path problems ($\alpha = 1$, finite-state system with a termination state) - we will discuss sparringly
  - Discounted and undiscounted problems with unbounded cost per stage - we will not cover
- Average cost problems - we will not cover
- Infinite horizon characteristics:
  - Challenging analysis, elegance of solutions and algorithms
  - Stationary policies $\pi = \{\mu, \mu, \ldots\}$ and stationary forms of DP play a special role

DISCOUNTED PROBLEMS/BOUNDED COST

- Stationary system

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

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

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

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

- Optimal cost function: $J^*(x) = \min_\pi J_\pi(x)$

- Boundedness of g guarantees that all costs are well-defined and bounded: $\big|J_\pi(x)\big| \leq \frac{M}{1-\alpha}$

- All spaces are arbitrary - only boundedness of g is important (there are math fine points, e.g. measurability, but they don't matter in practice)

- Important special case: All underlying spaces finite; a (finite spaces) Markovian Decision Problem or MDP

- All algorithms ultimately work with a finite spaces MDP approximating the original problem

SHORTHAND NOTATION FOR DP MAPPINGS

- For any function J of x, denote

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

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

- T operates on bounded functions of x to produce other bounded functions of x

- For any stationary policy μ, denote

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

- The critical structure of the problem is captured in T and T_μ

- The entire theory of discounted problems can be developed in shorthand using T and T_μ

- True for many other DP problems.

- T and T_μ provide a powerful unifying framework for DP. This is the essence of the book “Abstract Dynamic Programming”

FINITE-HORIZON COST EXPRESSIONS

- Consider an N-stage policy $\pi_0^N = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ with a terminal cost J:

$$\begin{aligned} J_{\pi_0^N}(x_0) &= E\left\{ \alpha^N J(x_k) + \sum_{\ell=0}^{N-1} \alpha^\ell g\big(x_\ell, \mu_\ell(x_\ell), w_\ell\big) \right\} \\ &= E\left\{ g\big(x_0, \mu_0(x_0), w_0\big) + \alpha J_{\pi_1^N}(x_1) \right\} \\ &= (T_{\mu_0} J_{\pi_1^N})(x_0) \end{aligned}$$

where $\pi_1^N = \{\mu_1, \mu_2, \ldots, \mu_{N-1}\}$

- By induction we have

$$J_{\pi_0^N}(x) = (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_{N-1}} J)(x), \qquad \forall\ x$$

- For a stationary policy μ the N-stage cost function (with terminal cost J) is

$$J_{\pi_0^N} = T_\mu^N J$$

where T_μ^N is the N-fold composition of T_μ

- Similarly the optimal N-stage cost function (with terminal cost J) is $T^N J$

- $T^N J = T(T^{N-1} J)$ is just the DP algorithm

“SHORTHAND” THEORY – A SUMMARY

- Infinite horizon cost function expressions [with $J_0(x) \equiv 0$]

$$J_\pi(x) = \lim_{N\to\infty} (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_N} J_0)(x), \quad J_\mu(x) = \lim_{N\to\infty} (T_\mu^N J_0)(x)$$

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

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

- Value iteration: For any (bounded) J

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \forall\, x$$

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

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

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

$$T_{\mu^{k+1}} J_{\mu^k} = TJ_{\mu^k}$$

TWO KEY PROPERTIES

- Monotonicity property: For any J and J' such that $J(x) \leq J'(x)$ for all x, and any μ

$$(TJ)(x) \leq (TJ')(x), \qquad \forall\, x,$$

$$(T_\mu J)(x) \leq (T_\mu J')(x), \qquad \forall\, x.$$

- Constant Shift property: For any J, any scalar r, and any μ

$$\big(T(J + re)\big)(x) = (TJ)(x) + \alpha r, \qquad \forall\, x,$$

$$\big(T_\mu(J + re)\big)(x) = (T_\mu J)(x) + \alpha r, \qquad \forall\, x,$$

where e is the unit function $[e(x) \equiv 1]$.

- Monotonicity is present in all DP models (undiscounted, etc)
- Constant shift is special to discounted models
- Discounted problems have another property of major importance: T and T_μ are contraction mappings (we will show this later)

CONVERGENCE OF VALUE ITERATION

- For all bounded J,

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \text{for all } x$$

Proof: For simplicity we give the proof for $J \equiv 0$. For any initial state x_0, and policy $\pi = \{\mu_0, \mu_1, \ldots\}$,

$$\begin{aligned} J_\pi(x_0) &= E\left\{\sum_{\ell=0}^{\infty} \alpha^\ell g\big(x_\ell, \mu_\ell(x_\ell), w_\ell\big)\right\} \\ &= E\left\{\sum_{\ell=0}^{k-1} \alpha^\ell g\big(x_\ell, \mu_\ell(x_\ell), w_\ell\big)\right\} \\ &\qquad + E\left\{\sum_{\ell=k}^{\infty} \alpha^\ell g\big(x_\ell, \mu_\ell(x_\ell), w_\ell\big)\right\} \end{aligned}$$

The tail portion satisfies

$$\left| E\left\{\sum_{\ell=k}^{\infty} \alpha^\ell g\big(x_\ell, \mu_\ell(x_\ell), w_\ell\big)\right\}\right| \le \frac{\alpha^k M}{1-\alpha},$$

where $M \ge |g(x,u,w)|$. Take min over π of both sides, then lim as $k \to \infty$. **Q.E.D.**

BELLMAN'S EQUATION

- The optimal cost function J^* is a solution of Bellman's equation, $J^* = TJ^*$, i.e., for all x,

$$J^*(x) = \min_{u \in U(x)} \mathop{E}_{w} \;\; g(x,u,w) + \alpha J^* \;\; f(x,u,w)$$

Proof: For all x and k,

$$J^*(x) - \frac{\alpha^k M}{1-\alpha} \le (T^k J_0)(x) \le J^*(x) + \frac{\alpha^k M}{1-\alpha},$$

where $J_0(x) \equiv 0$ and $M \ge |g(x,u,w)|$. Applying T to this relation, and using Monotonicity and Constant Shift,

$$\begin{aligned}(TJ^*)(x) - \frac{\alpha^{k+1} M}{1-\alpha} &\le (T^{k+1} J_0)(x) \\ &\le (TJ^*)(x) + \frac{\alpha^{k+1} M}{1-\alpha}\end{aligned}$$

Taking the limit as $k \to \infty$ and using the fact

$$\lim_{k \to \infty} (T^{k+1} J_0)(x) = J^*(x)$$

we obtain $J^* = TJ^*$. **Q.E.D.**

THE CONTRACTION PROPERTY

- Contraction property: For any bounded functions J and J', and any μ,

$$\max_x \big|(TJ)(x) - (TJ')(x)\big| \le \alpha \max_x \big|J(x) - J'(x)\big|,$$

$$\max_x \big|(T_\mu J)(x) - (T_\mu J')(x)\big| \le \alpha \max_x \big|J(x) - J'(x)\big|.$$

Proof: Denote $c = \max_{x \in S} \big|J(x) - J'(x)\big|$. Then

$$J(x) - c \le J'(x) \le J(x) + c, \qquad \forall\ x$$

Apply T to both sides, and use the Monotonicity and Constant Shift properties:

$$(TJ)(x) - \alpha c \le (TJ')(x) \le (TJ)(x) + \alpha c, \qquad \forall\ x$$

Hence

$$\big|(TJ)(x) - (TJ')(x)\big| \le \alpha c, \qquad \forall\ x.$$

Q.E.D.

- Note: This implies that J^* is the unique solution of $J^* = TJ^*$, and J_μ is the unique solution of $J_\mu = T_\mu J_\mu$

NEC. AND SUFFICIENT OPT. CONDITION

- A stationary policy μ is optimal if and only if $\mu(x)$ attains the minimum in Bellman's equation for each x; i.e.,

$$TJ^* = T_\mu J^*,$$

or, equivalently, for all x,

$$\mu(x) \in \arg \min_{u \in U(x)} \mathop{E}_{w} \Big\{ g(x,u,w) + \alpha J^* \big(f(x,u,w) \big) \Big\}$$

Proof: If $TJ^* = T_\mu J^*$, then using Bellman's equation ($J^* = TJ^*$), we have

$$J^* = T_\mu J^*,$$

so by uniqueness of the fixed point of T_μ, we obtain $J^* = J_\mu$; i.e., μ is optimal.

- Conversely, if the stationary policy μ is optimal, we have $J^* = J_\mu$, so

$$J^* = T_\mu J^*.$$

Combining this with Bellman's Eq. ($J^* = TJ^*$), we obtain $TJ^* = T_\mu J^*$. **Q.E.D.**

APPROXIMATE DYNAMIC PROGRAMMING

LECTURE 2

LECTURE OUTLINE

- Review of discounted problem theory
- Review of shorthand notation
- Algorithms for discounted DP
- Value iteration
- Various forms of policy iteration
- Optimistic policy iteration
- Q-factors and Q-learning
- Other DP models - Continuous space and time
- A more abstract view of DP
- Asynchronous algorithms

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 $|g(x,u,w)| \le M$ for all (x,u,w)

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

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

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

- For any stationary policy μ

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

“SHORTHAND” THEORY – A SUMMARY

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

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

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

- Optimality condition:

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

i.e.,

$$\mu(x) \in \arg\min_{u\in U(x)} \mathop{E}_{w} \left\{ g(x,u,w) + \alpha J^*\big(f(x,u,w)\big)\right\}, \; \forall\, x$$

- Value iteration: For any (bounded) J

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \forall\, x$$

- Policy iteration: Given μ^k,
  - Find J_{μ^k} from $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$ (policy evaluation); then
  - Find μ^{k+1} such that $T_{\mu^{k+1}} J_{\mu^k} = T J_{\mu^k}$ (policy improvement)

MAJOR PROPERTIES

- Monotonicity property: For any functions J and J' on the state space X such that $J(x) \leq J'(x)$ for all $x \in X$, and any μ

$$(TJ)(x) \leq (TJ')(x), \quad (T_\mu J)(x) \leq (T_\mu J')(x), \ \forall\, x \in X$$

- Contraction property: For any bounded functions J and J', and any μ,

$$\max_x \bigl|(TJ)(x) - (TJ')(x)\bigr| \leq \alpha \max_x \bigl|J(x) - J'(x)\bigr|,$$

$$\max_x \bigl|(T_\mu J)(x) - (T_\mu J')(x)\bigr| \leq \alpha \max_x \bigl|J(x) - J'(x)\bigr|$$

- Compact Contraction Notation:

$$\|TJ - TJ'\| \leq \alpha \|J - J'\|, \ \ \|T_\mu J - T_\mu J'\| \leq \alpha \|J - J'\|,$$

where for any bounded function J, we denote by $\|J\|$ the sup-norm

$$\|J\| = \max_x \bigl|J(x)\bigr|$$

THE TWO MAIN ALGORITHMS: VI AND PI

- Value iteration: For any (bounded) J

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \forall\, x$$

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

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

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

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

$$\mu^{k+1}(x) \in \arg \min_{u\in U(x)} \mathop{E}_{w}\left\{g(x,u,w) + \alpha J_{\mu^k}\big(f(x,u,w)\big)\right\},\ \forall\, x$$

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

- For the case of n states, policy evaluation is equivalent to solving an $n \times n$ linear system of equations: $J_\mu = g_\mu + \alpha P_\mu J_\mu$

- For large n, exact PI is out of the question (even though it terminates finitely as we will show)

JUSTIFICATION OF POLICY ITERATION

- We can show that $J_{\mu^k} \geq J_{\mu^{k+1}}$ for all k
- Proof: For given k, we have

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

Using the monotonicity property of DP,

$$J_{\mu^k} \geq T_{\mu^{k+1}} J_{\mu^k} \geq T^2_{\mu^{k+1}} J_{\mu^k} \geq \cdots \geq \lim_{N \to \infty} T^N_{\mu^{k+1}} J_{\mu^k}$$

- Since

$$\lim_{N \to \infty} T^N_{\mu^{k+1}} J_{\mu^k} = J_{\mu^{k+1}}$$

we have $J_{\mu^k} \geq J_{\mu^{k+1}}$.

- If $J_{\mu^k} = J_{\mu^{k+1}}$, all above inequalities hold as equations, so J_{μ^k} solves Bellman's equation. Hence $J_{\mu^k} = J^*$
- Thus at iteration k either the algorithm generates a strictly improved policy or it finds an optimal policy
  - For a finite spaces MDP, the algorithm terminates with an optimal policy
  - For infinite spaces MDP, convergence (in an infinite number of iterations) can be shown

OPTIMISTIC POLICY ITERATION

- Optimistic PI: This is PI, where policy evaluation is done approximately, with a finite number of VI

- So we approximate the policy evaluation

$$J_\mu \approx T_\mu^m J$$

for some number $m \in [1, \infty)$ and initial J

- Shorthand definition: For some integers m_k

$$T_{\mu^k} J_k = T J_k, \qquad J_{k+1} = T_{\mu^k}^{m_k} J_k, \qquad k = 0, 1, \ldots$$

- If $m_k \equiv 1$ it becomes VI

- If $m_k = \infty$ it becomes PI

- Converges for both finite and infinite spaces discounted problems (in an infinite number of iterations)

- Typically works faster than VI and PI (for large problems)

APPROXIMATE PI

- Suppose that the policy evaluation is approximate,

$$\|J_k - J_{\mu^k}\| \le \delta, \qquad k = 0, 1, \ldots$$

and policy improvement is approximate,

$$\|T_{\mu^{k+1}} J_k - T J_k\| \le \epsilon, \qquad k = 0, 1, \ldots$$

where δ and ϵ are some positive scalars.

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

$$\limsup_{k\to\infty} \|J_{\mu^k} - J^*\| \le \frac{\epsilon + 2\alpha\delta}{(1-\alpha)^2}$$

- 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^*.

- Error Bound II: If in addition the sequence $\{\mu^k\}$ "terminates" at $\overline{\mu}$ (i.e., keeps generating $\overline{\mu}$)

$$\|J_{\overline{\mu}} - J^*\| \le \frac{\epsilon + 2\alpha\delta}{1-\alpha}$$

Q-FACTORS I

- Optimal Q-factor of (x,u):

$$Q^*(x,u) = E\left\{g(x,u,w) + \alpha J^*(\overline{x})\right\}$$

with $\overline{x} = f(x,u,w)$. It is the cost of starting at x, applying u is the 1st stage, and an optimal policy after the 1st stage

- We can write Bellman's equation as

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

- We can equivalently write the VI method as

$$J_{k+1}(x) = \min_{u \in U(x)} Q_{k+1}(x,u), \qquad \forall\, x,$$

where Q_{k+1} is generated by

$$Q_{k+1}(x,u) = E\left\{ g(x,u,w) + \alpha \min_{v \in U(\overline{x})} Q_k(\overline{x},v) \right\}$$

with $\overline{x} = f(x,u,w)$

Q-FACTORS II

- Q-factors are costs in an “augmented” problem where states are (x,u)

- They satisfy a Bellman equation $Q^* = FQ^*$ where

$$(FQ)(x,u) = E\left\{g(x,u,w) + \alpha \min_{v\in U(\overline{x})} Q(\overline{x},v)\right\}$$

where $\overline{x} = f(x,u,w)$

- VI and PI for Q-factors are mathematically equivalent to VI and PI for costs

- They require equal amount of computation ... they just need more storage

- Having optimal Q-factors is convenient when implementing an optimal policy on-line by

$$\mu^*(x) = \min_{u\in U(x)} Q^*(x,u)$$

- Once $Q^*(x,u)$ are known, the model [g and $E\{\cdot\}$] is not needed. Model-free operation

- Q-Learning (to be discussed later) is a sampling method that calculates $Q^*(x,u)$ using a simulator of the system (no model needed)

OTHER DP MODELS

- We have looked so far at the (discrete or continuous spaces) discounted models for which the analysis is simplest and results are most powerful

- Other DP models include:
  - Undiscounted problems ($\alpha = 1$): They may include a special termination state (stochastic shortest path problems)
  - Continuous-time finite-state MDP: The time between transitions is random and state-and-control-dependent (typical in queueing systems, called Semi-Markov MDP). These can be viewed as discounted problems with state-and-control-dependent discount factors

- Continuous-time, continuous-space models: Classical automatic control, process control, robotics
  - Substantial differences from discrete-time
  - Mathematically more complex theory (particularly for stochastic problems)
  - Deterministic versions can be analyzed using classical optimal control theory
  - Admit treatment by DP, based on time discretization

CONTINUOUS-TIME MODELS

- System equation: $dx(t)/dt = f\big(x(t), u(t)\big)$
- Cost function: $\int_0^\infty g\big(x(t), u(t)\big)$
- Optimal cost starting from x: $J^*(x)$
- δ-Discretization of time: $x_{k+1} = x_k + \delta \cdot f(x_k, u_k)$
- Bellman equation for the δ-discretized problem:

$$J_\delta^*(x) = \min_u \big\{\delta \cdot g(x,u) + J_\delta^*\big(x + \delta \cdot f(x,u)\big)\big\}$$

- Take $\delta \to 0$, to obtain the Hamilton-Jacobi-Bellman equation [assuming $\lim_{\delta \to 0} J_\delta^*(x) = J^*(x)$]

$$0 = \min_u \big\{g(x,u) + \nabla J^*(x)' f(x,u)\big\}, \qquad \forall\ x$$

- Policy Iteration (informally):
  - Policy evaluation: Given current μ, solve

$$0 = g\big(x, \mu(x)\big) + \nabla J_\mu(x)' f\big(x, \mu(x)\big), \qquad \forall\ x$$

  - Policy improvement: Find

$$\overline{\mu}(x) \in \arg\min_u \big\{g(x,u) + \nabla J_\mu(x)' f(x,u)\big\}, \qquad \forall\ x$$

- Note: Need to learn $\nabla J_\mu(x)$ NOT $J_\mu(x)$

A MORE GENERAL/ABSTRACT VIEW OF DP

- Let Y be a real vector space with a norm $\|\cdot\|$

- A function $F : Y \mapsto Y$ is said to be a contraction mapping if for some $\rho \in (0,1)$, we have

$$\|Fy - Fz\| \leq \rho\|y - z\|, \qquad \text{for all } y, z \in Y.$$

ρ is called the modulus of contraction of F.

- Important example: Let X be a set (e.g., state space in DP), $v : X \mapsto \Re$ be a positive-valued function. Let $B(X)$ be the set of all functions $J : X \mapsto \Re$ such that $J(x)/v(x)$ is bounded over x.

- We define a norm on $B(X)$, called the weighted sup-norm, by

$$\|J\| = \max_{x \in X} \frac{|J(x)|}{v(x)}.$$

- Important special case: The discounted problem mappings T and T_μ [for $v(x) \equiv 1$, $\rho = \alpha$].

CONTRACTION MAPPINGS: AN EXAMPLE

- Consider extension from finite to countable state space, $X = \{1, 2, \ldots\}$, and a weighted sup norm with respect to which the one stage costs are bounded

- Suppose that T_μ has the form

$$(T_\mu J)(i) = b_i + \alpha \sum_{j \in X} a_{ij}\, J(j), \qquad \forall\ i = 1, 2, \ldots$$

where b_i and a_{ij} are some scalars. Then T_μ is a contraction with modulus ρ if and only if

$$\frac{\sum_{j \in X} |a_{ij}|\, v(j)}{v(i)} \le \rho, \qquad \forall\ i = 1, 2, \ldots$$

- Consider T,

$$(TJ)(i) = \min_{\mu} (T_\mu J)(i), \qquad \forall\ i = 1, 2, \ldots$$

where for each $\mu \in M$, T_μ is a contraction mapping with modulus ρ. Then T is a contraction mapping with modulus ρ

- Allows extensions of main DP results from bounded one-stage cost to interesting unbounded one-stage cost cases.

CONTRACTION MAPPING FIXED-POINT TH.

- Contraction Mapping Fixed-Point Theorem: If $F : B(X) \mapsto B(X)$ is a contraction with modulus $\rho \in (0,1)$, then there exists a unique $J^* \in B(X)$ such that

$$J^* = FJ^*.$$

Furthermore, if J is any function in $B(X)$, then $\{F^k J\}$ converges to J^* and we have

$$\|F^k J - J^*\| \le \rho^k \|J - J^*\|, \qquad k = 1, 2, \ldots.$$

- This is a special case of a general result for contraction mappings $F : Y \mapsto Y$ over normed vector spaces Y that are complete: every sequence $\{y_k\}$ that is Cauchy (satisfies $\|y_m - y_n\| \to 0$ as $m, n \to \infty$) converges.

- The space $B(X)$ is complete (see the text for a proof).

ABSTRACT FORMS OF DP

- We consider an abstract form of DP based on monotonicity and contraction
- Abstract Mapping: Denote $R(X)$: set of real-valued functions $J : X \mapsto \Re$, and let $H : X \times U \times R(X) \mapsto \Re$ be a given mapping. We consider the mapping

$$(TJ)(x) = \min_{u \in U(x)} H(x, u, J), \qquad \forall\, x \in X.$$

- We assume that $(TJ)(x) > -\infty$ for all $x \in X$, so T maps $R(X)$ into $R(X)$.
- Abstract Policies: Let $\mathcal{M}$ be the set of "policies", i.e., functions μ such that $\mu(x) \in U(x)$ for all $x \in X$.
- For each $\mu \in \mathcal{M}$, we consider the mapping $T_\mu : R(X) \mapsto R(X)$ defined by

$$(T_\mu J)(x) = H\big(x, \mu(x), J\big), \qquad \forall\, x \in X.$$

- Find a function $J^* \in R(X)$ such that

$$J^*(x) = \min_{u \in U(x)} H(x, u, J^*), \qquad \forall\, x \in X$$

EXAMPLES

- Discounted problems

$$H(x,u,J) = E\big\{g(x,u,w) + \alpha J\big(f(x,u,w)\big)\big\}$$

- Discounted "discrete-state continuous-time" Semi-Markov Problems (e.g., queueing)

$$H(x,u,J) = G(x,u) + \sum_{y=1}^{n} m_{xy}(u)J(y)$$

where m_{xy} are "discounted" transition probabilities, defined by the distribution of transition times

- Minimax Problems/Games

$$H(x,u,J) = \max_{w \in W(x,u)} \big[g(x,u,w)+\alpha J\big(f(x,u,w)\big)\big]$$

- Shortest Path Problems

$$H(x,u,J) = \begin{cases} a_{xu} + J(u) & \text{if } u \neq d, \\ a_{xd} & \text{if } u = d \end{cases}$$

where d is the destination. There are stochastic and minimax versions of this problem

ASSUMPTIONS

- Monotonicity: If $J, J' \in R(X)$ and $J \leq J'$,

$$H(x,u,J) \leq H(x,u,J'), \qquad \forall\, x \in X,\ u \in U(x)$$

- We can show all the standard analytical and computational results of discounted DP if monotonicity and the following assumption holds:

- Contraction:
  - For every $J \in B(X)$, the functions $T_\mu J$ and TJ belong to $B(X)$
  - For some $\alpha \in (0,1)$, and all μ and $J, J' \in B(X)$, we have

$$\|T_\mu J - T_\mu J'\| \leq \alpha \|J - J'\|$$

- With just monotonicity assumption (as in undiscounted problems) we can still show various forms of the basic results under appropriate conditions

- A weaker substitute for contraction assumption is semicontractiveness: (roughly) for some μ, T_μ is a contraction and for others it is not; also the "noncontractive" μ are not optimal

RESULTS USING CONTRACTION

- Proposition 1: The mappings T_μ and T are weighted sup-norm contraction mappings with modulus α over $B(X)$, and have unique fixed points in $B(X)$, denoted J_μ and J^*, respectively (cf. Bellman's equation).

Proof: From the contraction property of H.

- Proposition 2: For any $J \in B(X)$ and $\mu \in \mathcal{M}$,

$$\lim_{k\to\infty} T_\mu^k J = J_\mu, \qquad \lim_{k\to\infty} T^k J = J^*$$

(cf. convergence of value iteration).

Proof: From the contraction property of T_μ and T.

- Proposition 3: We have $T_\mu J^* = TJ^*$ if and only if $J_\mu = J^*$ (cf. optimality condition).

Proof: $T_\mu J^* = TJ^*$, then $T_\mu J^* = J^*$, implying $J^* = J_\mu$. Conversely, if $J_\mu = J^*$, then $T_\mu J^* = T_\mu J_\mu = J_\mu = J^* = TJ^*$.

RESULTS USING MON. AND CONTRACTION

- Optimality of fixed point:

$$J^*(x) = \min_{\mu \in \mathcal{M}} J_\mu(x), \qquad \forall\, x \in X$$

- Existence of a nearly optimal policy: For every $\epsilon > 0$, there exists $\mu_\epsilon \in \mathcal{M}$ such that

$$J^*(x) \le J_{\mu_\epsilon}(x) \le J^*(x) + \epsilon, \qquad \forall\, x \in X$$

- Nonstationary policies: Consider the set Π of all sequences $\pi = \{\mu_0, \mu_1, \ldots\}$ with $\mu_k \in \mathcal{M}$ for all k, and define

$$J_\pi(x) = \liminf_{k \to \infty} (T_{\mu_0} T_{\mu_1} \cdots T_{\mu_k} J)(x), \qquad \forall\, x \in X,$$

with J being any function (the choice of J does not matter)

- We have

$$J^*(x) = \min_{\pi \in \Pi} J_\pi(x), \qquad \forall\, x \in X$$

THE TWO MAIN ALGORITHMS: VI AND PI

- Value iteration: For any (bounded) J

$$J^*(x) = \lim_{k\to\infty} (T^k J)(x), \qquad \forall\ x$$

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

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

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

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

- Optimistic PI: This is PI, where policy evaluation is carried out by a finite number of VI
  - Shorthand definition: For some integers m_k

$$T_{\mu^k} J_k = T J_k, \qquad J_{k+1} = T_{\mu^k}^{m_k} J_k, \qquad k = 0, 1, \ldots$$

  - If $m_k \equiv 1$ it becomes VI
  - If $m_k = \infty$ it becomes PI
  - For intermediate values of m_k, it is generally more efficient than either VI or PI

ASYNCHRONOUS ALGORITHMS

- Motivation for asynchronous algorithms
  - Faster convergence
  - Parallel and distributed computation
  - Simulation-based implementations

- General framework: Partition X into disjoint nonempty subsets $X_1, \dots, X_m$, and use separate processor ℓ updating $J(x)$ for $x \in X_\ell$

- Let J be partitioned as

$$J = (J_1, \dots, J_m),$$

where J_ℓ is the restriction of J on the set X_ℓ.

- Synchronous VI algorithm:

$$J_\ell^{t+1}(x) = T(J_1^t, \dots, J_m^t)(x), \quad x \in X_\ell,\ \ell = 1, \dots, m$$

- Asynchronous VI algorithm: For some subsets of times $\mathcal{R}_\ell$,

$$J_\ell^{t+1}(x) = \begin{cases} T(J_1^{\tau_{\ell 1}(t)}, \dots, J_m^{\tau_{\ell m}(t)})(x) & \text{if } t \in \mathcal{R}_\ell, \\ J_\ell^t(x) & \text{if } t \notin \mathcal{R}_\ell \end{cases}$$

where $t - \tau_{\ell j}(t)$ are communication “delays”

ONE-STATE-AT-A-TIME ITERATIONS

- Important special case: Assume n "states", a separate processor for each state, and no delays

- Generate a sequence of states $\{x^0, x^1, \ldots\}$, generated in some way, possibly by simulation (each state is generated infinitely often)

- Asynchronous VI:

$$J_\ell^{t+1} = \begin{cases} T(J_1^t, \ldots, J_n^t)(\ell) & \text{if } \ell = x^t, \\ J_\ell^t & \text{if } \ell \neq x^t, \end{cases}$$

where $T(J_1^t, \ldots, J_n^t)(\ell)$ denotes the ℓ-th component of the vector

$$T(J_1^t, \ldots, J_n^t) = TJ^t,$$

- The special case where

$$\{x^0, x^1, \ldots\} = \{1, \ldots, n, 1, \ldots, n, 1, \ldots\}$$

is the Gauss-Seidel method

ASYNCHRONOUS CONV. THEOREM I

- KEY FACT: VI and also PI (with some modifications) still work when implemented asynchronously

- Assume that for all $\ell, j = 1, \ldots, m$, $\mathcal{R}_\ell$ is infinite and $\lim_{t\to\infty} \tau_{\ell j}(t) = \infty$

- Proposition: Let T have a unique fixed point J^*, and assume that there is a sequence of nonempty subsets $S(k) \subset R(X)$ with $S(k+1) \subset S(k)$ for all k, and with the following properties:

(1) Synchronous Convergence Condition: Every sequence $\{J^k\}$ with $J^k \in S(k)$ for each k, converges pointwise to J^*. Moreover,

$$TJ \in S(k+1), \qquad \forall\, J \in S(k),\ k = 0, 1, \ldots .$$

(2) Box Condition: For all k, $S(k)$ is a Cartesian product of the form

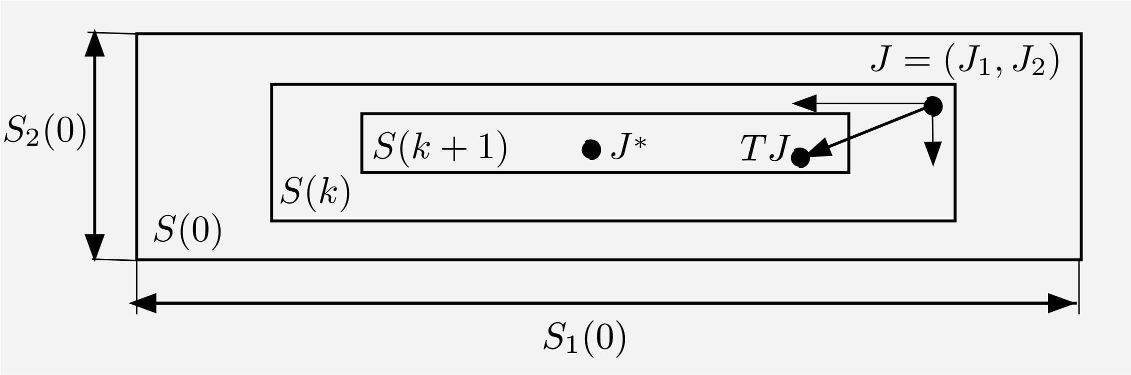
$$S(k) = S_1(k) \times \cdots \times S_m(k),$$

where $S_\ell(k)$ is a set of real-valued functions on X_ℓ, $\ell = 1, \ldots, m$.

Then for every $J \in S(0)$, the sequence $\{J^t\}$ generated by the asynchronous algorithm converges pointwise to J^*.

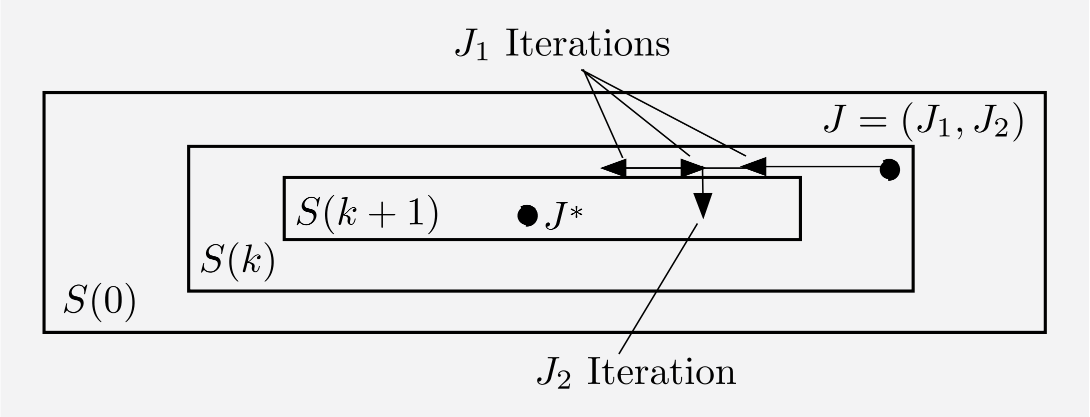
ASYNCHRONOUS CONV. THEOREM II

- Interpretation of assumptions:



A synchronous iteration from any J in $S(k)$ moves into $S(k+1)$ (component-by-component)

- Convergence mechanism:



Key: "Independent" component-wise improvement. An asynchronous component iteration from any J in $S(k)$ moves into the corresponding component portion of $S(k+1)$

APPROXIMATE DYNAMIC PROGRAMMING

LECTURE 3

LECTURE OUTLINE

- Review of discounted DP
- Introduction to approximate DP
- Approximation architectures
- Simulation-based approximate policy iteration
- Approximate policy evaluation
- Some general issues about approximation and simulation

REVIEW

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)

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

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

- For any stationary policy μ

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

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 $i = 1, \ldots, n$ (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 functions $J = \big(J(1), \ldots, J(n)\big)$ (vectors in $\Re^n$)

- 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

- 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.,

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

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. We use instead optimistic PI (policy evaluation with a few VIs)

APPROXIMATE DP

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

$$\sum_{i=1}^{n} \boldsymbol{a}_i$$

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 $\{1,\ldots,n\}$

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

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

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

APPROXIMATION IN VALUE AND 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

- Use $\tilde{J}$ in place of J^* or J_μ in various algorithms and computations

- Role of r: By adjusting r we can change the "shape" of $\tilde{J}$ so that it is "close" to J^* or J_μ

- 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 ... also 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

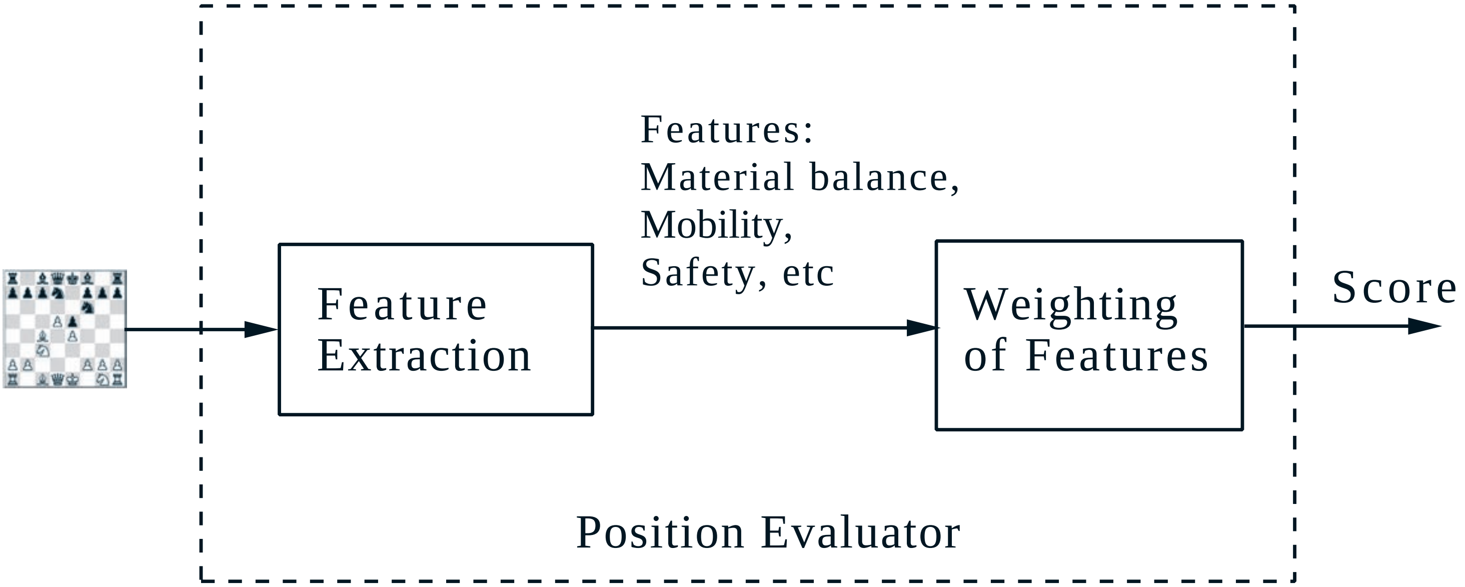
- We may also use parametric approximation for Q-factors or cost function differences

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

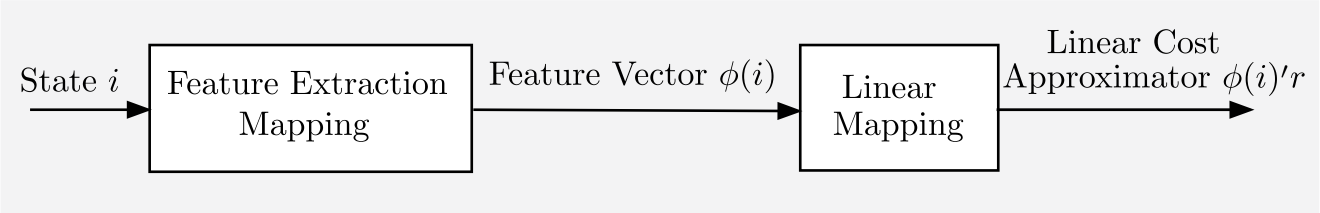
- Often, the features encode much of the nonlinearity inherent in the cost function approximated

- 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: $\tilde{J}(i; r) = \phi(i)'r$, $i = 1, \ldots, n$, or

$$\tilde{J}(r) = \Phi r = \sum_{j=1}^{s} \Phi_j r_j$$

Φ: the matrix whose rows are $\phi(i)'$, $i = 1, \ldots, n$, Φ_j is the jth column of Φ


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, etc

ILLUSTRATIONS: POLYNOMIAL TYPE

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

$$\phi_0(i) = 1, \;\; \phi_k(i) = i_k, \;\; \phi_{km}(i) = i_k i_m, \qquad k, m = 1, \ldots, q$$

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

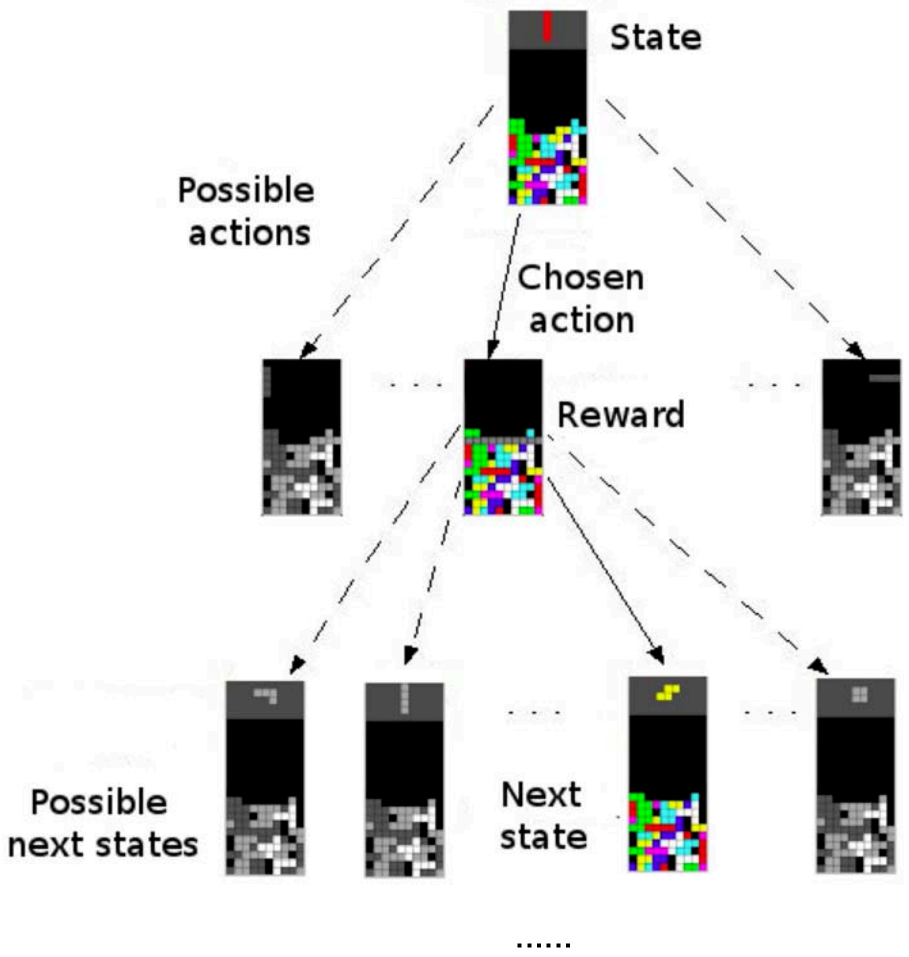
$$\tilde{J}(i; r) = r_i, \qquad i \in I,$$

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

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

A DOMAIN SPECIFIC EXAMPLE

- Tetris game (used as testbed in competitions)



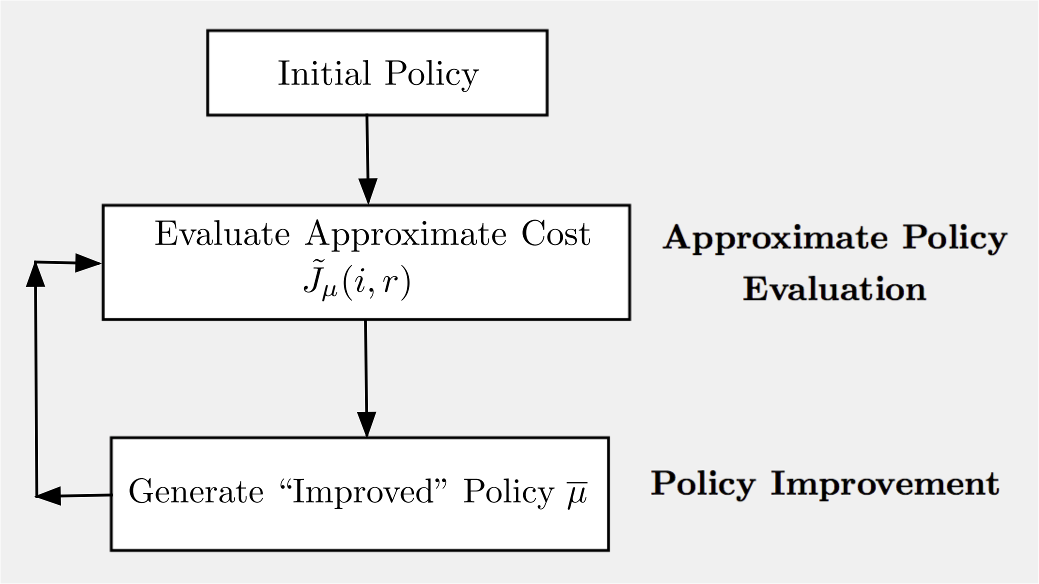
TERMINATION

- $J^*(i)$: 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)

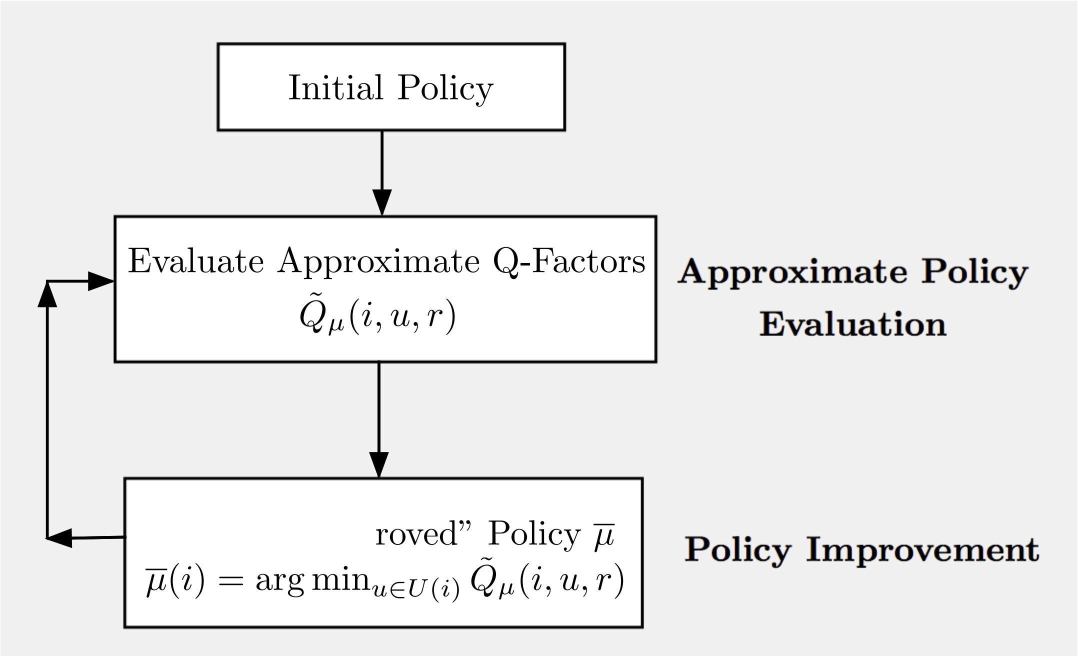
APPROX. PI - OPTION TO APPROX. J_μ OR Q_μ

- Use simulation to approximate the cost J_μ of the current policy μ

- Generate “improved” policy $\overline{\mu}$ by minimizing in (approx.) Bellman equation


Initial Policy
Evaluate Approximate Cost $\tilde{J}_\mu(i,r)$
Approximate Policy Evaluation
Generate “Improved” Policy $\overline{\mu}$
Policy Improvement

- Altenatively approximate the Q-factors of μ


Initial Policy
Evaluate Approximate Q-Factors $\tilde{Q}_\mu(i,u,r)$
Approximate Policy Evaluation
roved” Policy $\overline{\mu}$
$\overline{\mu}(i) = \arg\min_{u \in U(i)} \tilde{Q}_\mu(i,u,r)$
Policy Improvement

APPROXIMATING 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

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

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

APPROXIMATION IN POLICY SPACE

- A brief discussion; we will return to it later.
- Use parametrization $\mu(i;r)$ of policies with a vector $r = (r_1,\ldots,r_s)$. 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 $(p_1,\ldots,p_n)$ 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.,

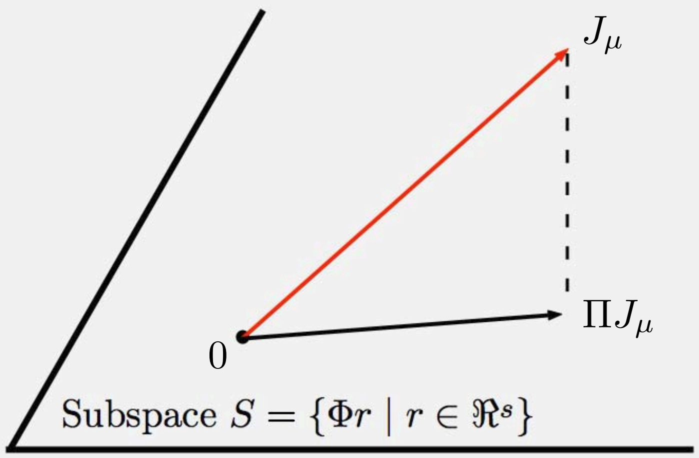
$$\mu(i;r) \in \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \hat{J}(j;r)\big)$$

APPROXIMATE POLICY EVALUATION METHODS

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 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_μ of J_μ on subspace $S = \{\Phi r \mid r \in \Re^s\}$, with respect to a weighted Euclidean norm $\|\cdot\|_\xi$

- Equivalently, find Φr^*, where

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

- Setting to 0 the gradient at r^*,

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

- Generate samples $\big\{(i_1, J_\mu(i_1)), \ldots, (i_k, J_\mu(i_k))\big\}$ using distribution ξ

- Approximate by Monte Carlo the two "expected values" with low-dimensional calculations

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

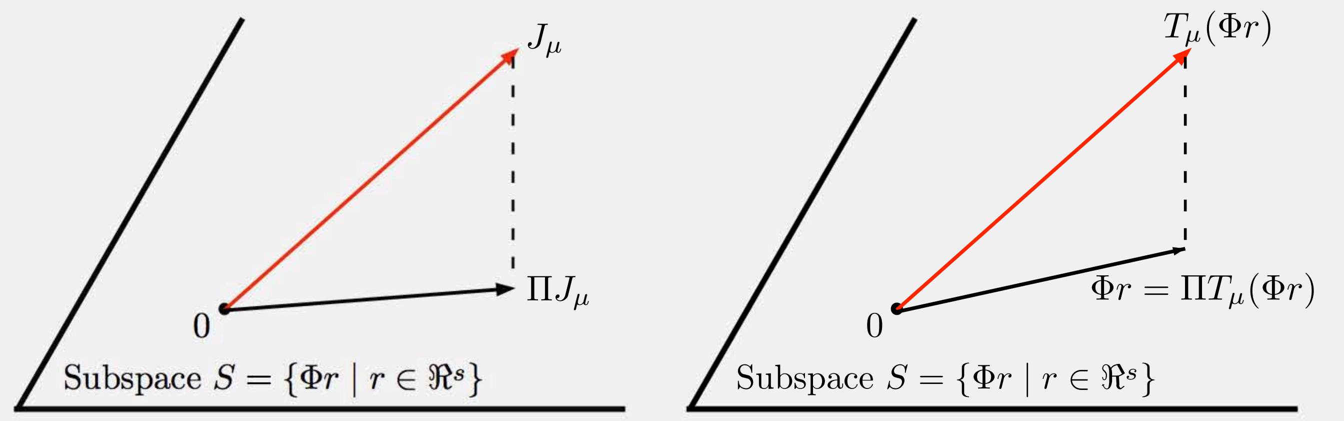
- Equivalent least squares alternative calculation:

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

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


J_μ
ΠJ_μ
0
Subspace $S = \{\Phi r \mid r \in \Re^s\}$
$T_\mu(\Phi r)$
$\Phi r = \Pi T_\mu(\Phi r)$
0
Subspace $S = \{\Phi r \mid r \in \Re^s\}$

Direct Method: Projection of cost vector J_μ

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

- Solution methods that use simulation (to manage the 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 METHODS

- Another example of indirect approximate policy evaluation:

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

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)

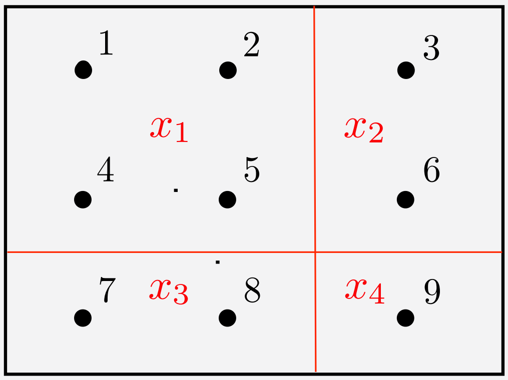
- Several ways to implement projected equation and Bellman error methods by simulation. They involve:
  - Generating many random samples of states i_k using the distribution ξ
  - Generating many samples of transitions (i_k, j_k) using the policy μ
  - Form a simulation-based approximation of the optimality condition for projection problem or problem (*) (use sample averages in place of inner products)
  - Solve the 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

- A first idea: Group similar states together into "aggregate states" $x_1, \ldots, x_s$; assign a common cost value r_i to each group x_i.

- Solve an "aggregate" DP problem, involving the aggregate states, to obtain $r = (r_1, \ldots, r_s)$. This is called hard aggregation



$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

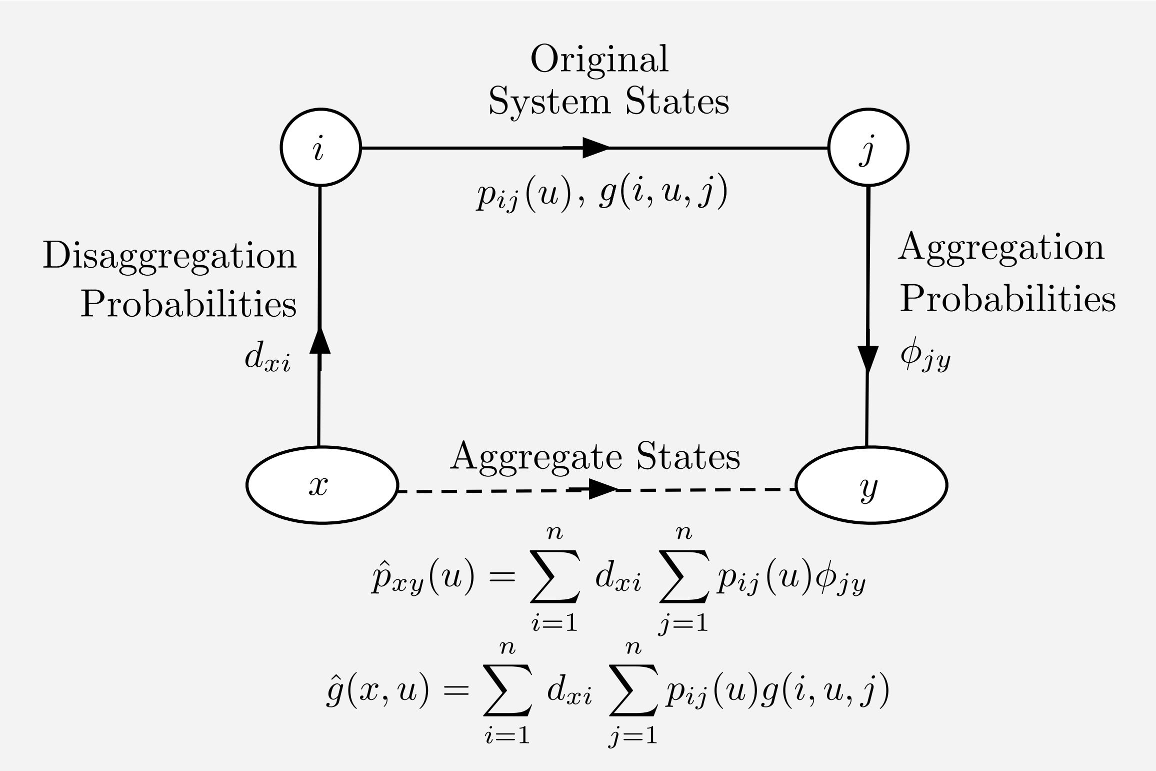
- More general/mathematical view: Solve

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

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

- Compare with projected equation $\Phi r = \Pi T_\mu(\Phi r)$. Note: ΦD is a projection in some interesting cases

AGGREGATION AS PROBLEM APPROXIMATION



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

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

- Aggregation 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

APPROXIMATE POLICY ITERATION

ISSUES

THEORETICAL BASIS OF APPROXIMATE PI

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

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

- If policy improvement is also approximate,

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

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

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

- 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^*.
- Oscillations are quite unpredictable.
  - Some bad examples of oscillations have been constructed.
  - In practice oscillations between policies is probably not the major concern.

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 $\overline{\mu}$

- 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)

- 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

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

and use for policy improvement the minimization

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

- 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

SOME GENERAL ISSUES

STOCHASTIC ALGORITHMS: GENERALITIES

- Consider solution of a linear equation $x = b + Ax$ by using m simulation samples $b + w_k$ and $A + W_k$, $k = 1, \ldots, m$, 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 $k = 1, \ldots, m$

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

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

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

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

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

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

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

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

- Approximating $D_\mu(\overline{x},\overline{x}')$ avoids "noise differencing". This can make a big difference

- Important point: D_μ satisfies a Bellman equation for a system with "state" (x,x')

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

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

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

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

AN EXAMPLE (FROM THE NDP TEXT)

- System and cost per stage:

$$x_{k+1} = x_k + \delta u_k, \qquad g(x,u) = \delta(x^2 + u^2)$$

$\delta > 0$ is very small; think of discretization of continuous-time problem involving $dx(t)/dt = u(t)$

- Consider policy $\mu(x) = -2x$. Its cost function is

$$J_\mu(x) = \frac{5x^2}{4}(1+\delta) + O(\delta^2)$$

and its Q-factor is

$$Q_\mu(x,u) = \frac{5x^2}{4} + \delta\left(\frac{9x^2}{4} + u^2 + \frac{5}{2}xu\right) + O(\delta^2)$$

- The important part for policy improvement is

$$\delta\left(u^2 + \frac{5}{2}xu\right)$$

When $J_\mu(x)$ [or $Q_\mu(x,u)$] is approximated by $\tilde{J}_\mu(x;r)$ [or by $\tilde{Q}_\mu(x,u;r)$], it will be dominated by $\frac{5x^2}{4}$ and will be "lost"

6.231 DYNAMIC PROGRAMMING

LECTURE 4

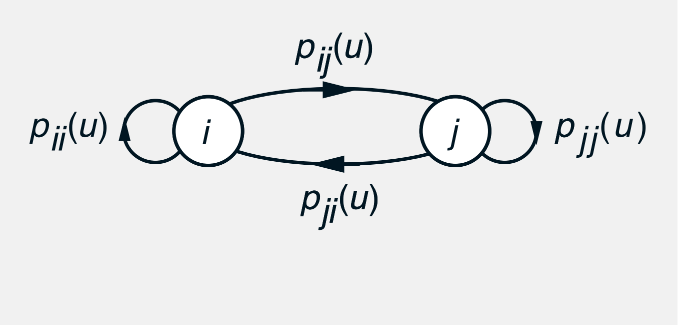
LECTURE OUTLINE

- Review of approximation in value space
- Approximate VI and PI
- Projected Bellman equations
- Matrix form of the projected equation
- Simulation-based implementation
- LSTD and LSPE methods
- Optimistic versions
- Multistep projected Bellman equations
- Bias-variance tradeoff

REVIEW

DISCOUNTED MDP

- System: Controlled Markov chain with states $i = 1, \ldots, n$, and finite control set $U(i)$ at state i
- Transition probabilities: $p_{ij}(u)$



- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$ starting at state i:

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

with $\alpha \in [0, 1)$

- 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

- 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

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

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)

APPROXIMATION IN VALUE SPACE

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

- Think n: HUGE, s: (Relatively) SMALL

- Many types of approximation architectures [i.e., parametric classes $\tilde{J}(i;r)$] to select from

- Any $r \in \Re^s$ defines a (suboptimal) one-step lookahead policy

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

- We want to find a "good" r

- We will focus mostly on linear architectures

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

where Φ is an $n \times s$ matrix whose columns are viewed as basis functions

LINEAR APPROXIMATION ARCHITECTURES

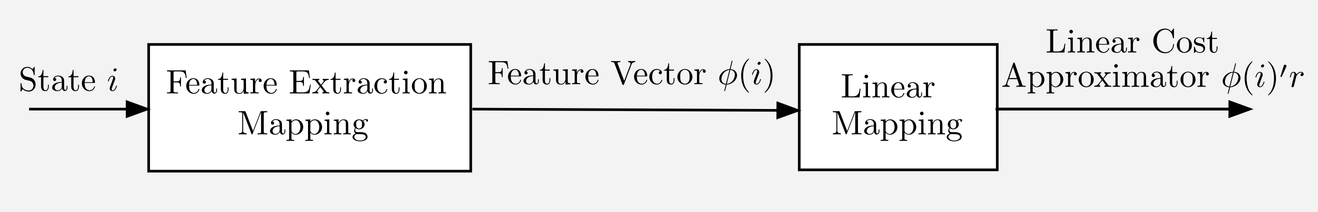
- We have

$$\tilde{J}(i;r) = \phi(i)'r, \qquad i = 1, \ldots, n$$

where $\phi(i)'$, $i = 1, \ldots, n$ is the ith row of Φ, or

$$\tilde{J}(r) = \Phi r = \sum_{j=1}^{s} \Phi_j r_j$$

where Φ_j is the jth column of Φ


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, etc

- Instead of computing J_μ or J^*, which is huge-dimensional, we compute the low-dimensional $r = (r_1, \ldots, r_s)$ using low-dimensional calculations

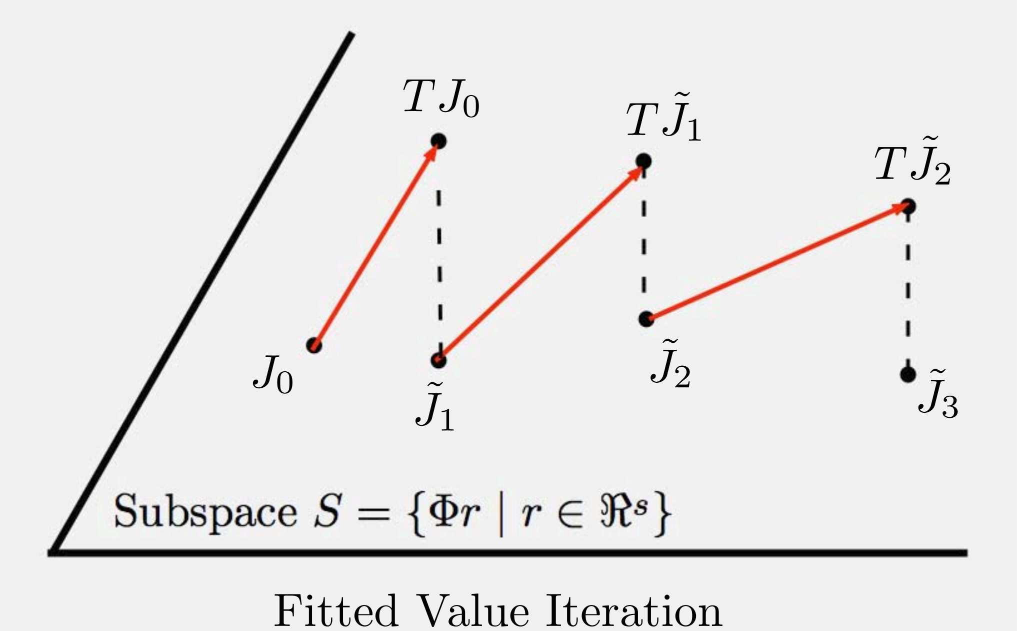
APPROXIMATE VALUE ITERATION

APPROXIMATE (FITTED) VI

- Approximates sequentially $J_k(i) = (T^k J_0)(i)$, $k = 1, 2, \ldots$, with $\tilde J_k(i; r_k)$

- The starting function J_0 is given (e.g., $J_0 \equiv 0$)

- Approximate (Fitted) Value Iteration: A sequential "fit" to produce $\tilde J_{k+1}$ from $\tilde J_k$, i.e., $\tilde J_{k+1} \approx T\tilde J_k$ or (for a single policy μ) $\tilde J_{k+1} \approx T_\mu \tilde J_k$



Fitted Value Iteration

- After a large enough number N of steps, $\tilde J_N(i; r_N)$ is used as approximation $\tilde J(i; r)$ to $J^*(i)$

- Possibly use (approximate) projection Π with respect to some projection norm,

$$\tilde J_{k+1} \approx \Pi T \tilde J_k$$

WEIGHTED EUCLIDEAN PROJECTIONS

- Consider a weighted Euclidean norm

$$\|J\|_\xi = \sqrt{\sum_{i=1}^{n} \xi_i \big(J(i)\big)^2},$$

where $\xi = (\xi_1, \ldots, \xi_n)$ is a positive distribution ($\xi_i > 0$ for all i).

- Let Π denote the projection operation onto

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

with respect to this norm, i.e., for any $J \in \Re^n$,

$$\Pi J = \Phi r^*$$

where

$$r^* = \arg\min_{r \in \Re^s} \|\Phi r - J\|_\xi^2$$

- Recall that weighted Euclidean projection can be implemented by simulation and least squares, i.e., sampling $J(i)$ according to ξ and solving

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

FITTED VI - NAIVE IMPLEMENTATION

- Select/sample a "small" subset I_k of representative states

- For each $i \in I_k$, given $\tilde{J}_k$, compute

$$(T\tilde{J}_k)(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_k(j;r)\big)$$

- "Fit" the function $\tilde{J}_{k+1}(i;r_{k+1})$ to the "small" set of values $(T\tilde{J}_k)(i)$, $i \in I_k$ (for example use some form of approximate projection)

- Simulation can be used for "model-free" implementation

- Error Bound: If the fit is uniformly accurate within $\delta > 0$, i.e.,

$$\max_i |\tilde{J}_{k+1}(i) - T\tilde{J}_k(i)| \leq \delta,$$

then

$$\limsup_{k\to\infty} \max_{i=1,\ldots,n} \big(\tilde{J}_k(i,r_k) - J^*(i)\big) \leq \frac{2\alpha\delta}{(1-\alpha)^2}$$

- But there is a potential problem!

AN EXAMPLE OF FAILURE

- Consider two-state discounted MDP with states 1 and 2, and a single policy.
  - Deterministic transitions: $1 \to 2$ and $2 \to 2$
  - Transition costs $\equiv 0$, so $J^*(1) = J^*(2) = 0$.
- Consider (exact) fitted VI scheme that approximates cost functions within $S = \big\{(r, 2r) \mid r \in \Re\big\}$ with a weighted least squares fit; here $\Phi = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$
- Given $\tilde J_k = (r_k, 2r_k)$, we find $\tilde J_{k+1} = (r_{k+1}, 2r_{k+1})$, where $\tilde J_{k+1} = \Pi_\xi(T\tilde J_k)$, with weights $\xi = (\xi_1, \xi_2)$:

$$r_{k+1} = \arg\min_r \Big[\xi_1\big(r-(T\tilde J_k)(1)\big)^2+\xi_2\big(2r-(T\tilde J_k)(2)\big)^2\Big]$$

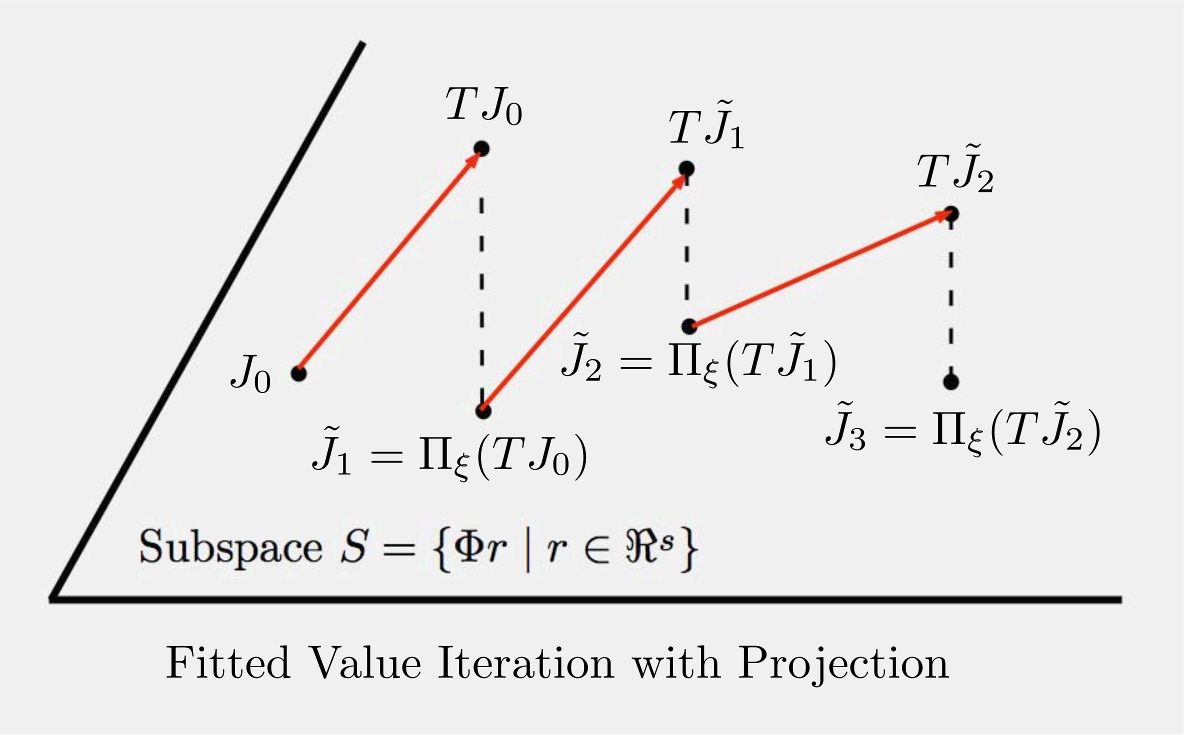
- With straightforward calculation

$$r_{k+1} = \alpha\beta r_k, \qquad \text{where } \beta = 2(\xi_1+2\xi_2)/(\xi_1+4\xi_2) > 1$$

- So if $\alpha > 1/\beta$ (e.g., $\xi_1 = \xi_2 = 1$), the sequence $\{r_k\}$ diverges and so does $\{\tilde J_k\}$.
- Difficulty is that T is a contraction, but $\Pi_\xi T$ (= least squares fit composed with T) is not.

NORM MISMATCH PROBLEM

- For the method to converge, we need $\Pi_\xi T$ to be a contraction; the contraction property of T is not enough


TJ_0
$T\tilde{J}_1$
$T\tilde{J}_2$
J_0
$\tilde{J}_2 = \Pi_\xi(T\tilde{J}_1)$
$\tilde{J}_3 = \Pi_\xi(T\tilde{J}_2)$
$\tilde{J}_1 = \Pi_\xi(TJ_0)$
Subspace $S = \{\Phi r \mid r \in \Re^s\}$

Fitted Value Iteration with Projection

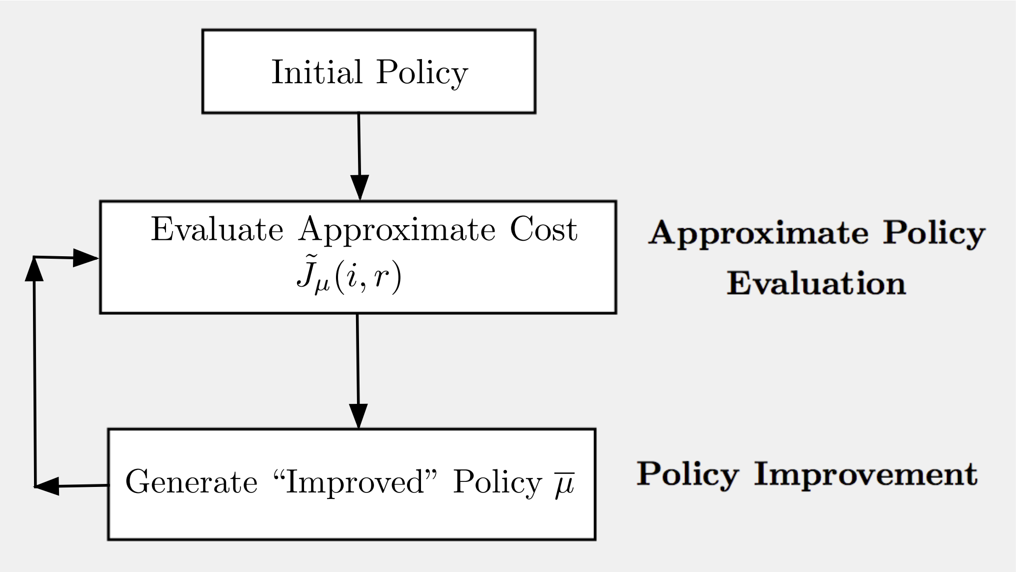
- We need a vector of weights ξ such that T is a contraction with respect to the weighted Euclidean norm $\|\cdot\|_\xi$

- Then we can show that $\Pi_\xi T$ is a contraction with respect to $\|\cdot\|_\xi$

- We will come back to this issue

APPROXIMATE POLICY ITERATION

APPROXIMATE PI


Initial Policy
Evaluate Approximate Cost $\tilde J_\mu(i,r)$
Approximate Policy Evaluation
Generate "Improved" Policy $\overline{\mu}$
Policy Improvement

- Evaluation of typical policy μ: Linear cost function approximation $\tilde J_\mu(r) = \Phi r$, where Φ is full rank $n \times s$ matrix with columns the basis functions, and ith row denoted $\phi(i)'$.

- Policy "improvement" to generate $\overline{\mu}$:

$$\overline{\mu}(i) = \arg \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha\phi(j)'r\big)$$

- Error Bound (same as approximate VI): If

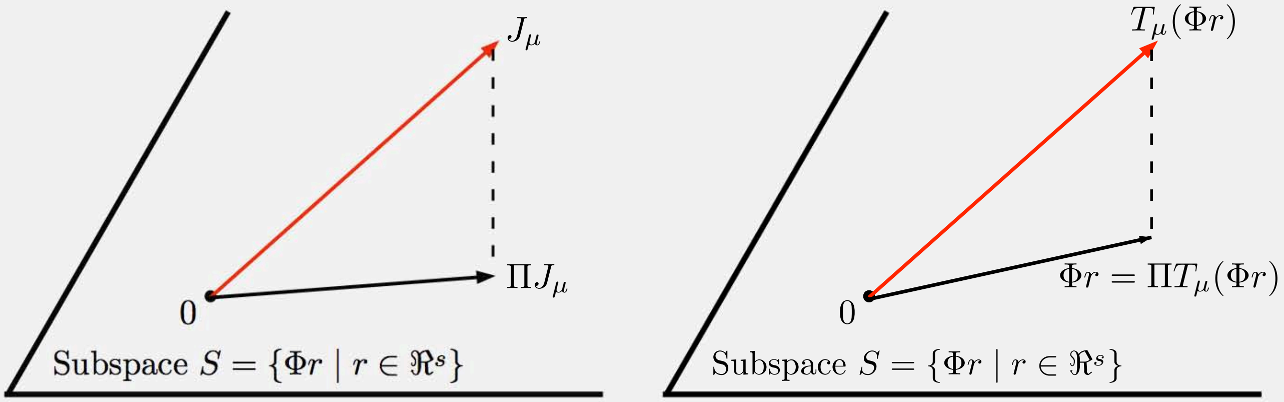
$$\max_i |\tilde J_{\mu^k}(i, r_k) - J_{\mu^k}(i)| \le \delta, \qquad k = 0, 1, \ldots$$

the sequence $\{\mu^k\}$ satisfies

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

POLICY EVALUATION

- Let's consider approximate evaluation of the cost of the current policy by using simulation.
  - 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 Π is projection w/ respect to a suitable weighted Euclidean norm

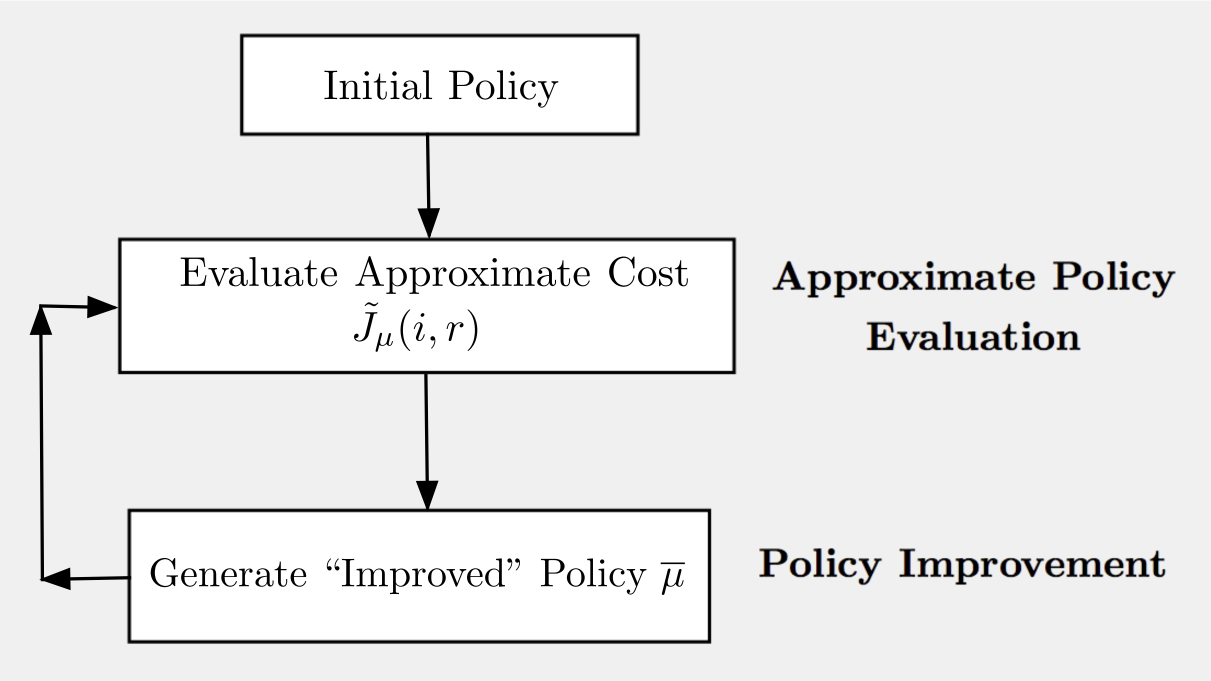

J_μ
ΠJ_μ
0
Subspace $S = \{\Phi r \mid r \in \Re^s\}$
$T_\mu(\Phi r)$
$\Phi r = \Pi T_\mu(\Phi r)$
0
Subspace $S = \{\Phi r \mid r \in \Re^s\}$

Direct Method: Projection of cost vector J_μ

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

- Recall that projection can be implemented by simulation and least squares

PI WITH INDIRECT POLICY EVALUATION


Initial Policy
Evaluate Approximate Cost $\tilde{J}_\mu(i,r)$
Approximate Policy Evaluation
Generate "Improved" Policy $\overline{\mu}$
Policy Improvement

- Given the current policy μ:
  - We solve the projected Bellman's equation

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

  - We approximate the solution J_μ of Bellman's equation

    $$J = T_\mu J$$

    with the projected equation solution $\tilde{J}_\mu(r)$

KEY QUESTIONS AND RESULTS

- Does the projected equation have a solution?

- Under what conditions is the mapping ΠT_μ a contraction, so ΠT_μ has unique fixed point?

- Assumption: The Markov chain corresponding to μ has a single recurrent class and no transient states, i.e., it has steady-state probabilities that are positive

$$\xi_j = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} P(i_k = j \mid i_0 = i) > 0$$

Note that ξ_j is the long-term frequency of state j.

- Proposition: (Norm Matching Property) Assume that the projection Π is with respect to $\|\cdot\|_\xi$, where $\xi = (\xi_1, \ldots, \xi_n)$ is the steady-state probability vector. Then:

  (a) ΠT_μ is contraction of modulus α with respect to $\|\cdot\|_\xi$.

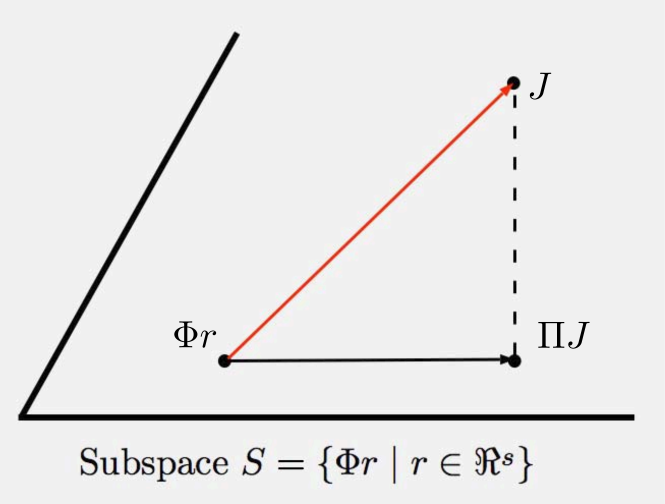
  (b) The unique fixed point Φr^* of ΠT_μ satisfies

$$\|J_\mu - \Phi r^*\|_\xi \le \frac{1}{\sqrt{1-\alpha^2}} \|J_\mu - \Pi J_\mu\|_\xi$$

PRELIMINARIES: PROJECTION PROPERTIES

- Important property of the projection Π on S with weighted Euclidean norm $\|\cdot\|_\xi$. For all $J \in \Re^n$, $\Phi r \in S$, the Pythagorean Theorem holds:

$$\|J - \Phi r\|_\xi^2 = \|J - \Pi J\|_\xi^2 + \|\Pi J - \Phi r\|_\xi^2$$



- The Pythagorean Theorem implies that the projection is nonexpansive, i.e.,

$$\|\Pi J - \Pi \bar{J}\|_\xi \leq \|J - \bar{J}\|_\xi, \qquad \text{for all } J, \bar{J} \in \Re^n.$$

To see this, note that

$$\begin{aligned}\left\|\Pi(J - \overline{J})\right\|_\xi^2 &\leq \left\|\Pi(J - \overline{J})\right\|_\xi^2 + \left\|(I - \Pi)(J - \overline{J})\right\|_\xi^2 \\ &= \|J - \overline{J}\|_\xi^2\end{aligned}$$

PROOF OF CONTRACTION PROPERTY

- Lemma: If P is the transition matrix of μ,

$$\|Pz\|_\xi \leq \|z\|_\xi, \qquad z \in \Re^n$$

Proof: Let p_{ij} be the components of P. For all $z \in \Re^n$, we have

$$\|Pz\|_\xi^2 = \sum_{i=1}^n \xi_i \left(\sum_{j=1}^n p_{ij} z_j \right)^2 \leq \sum_{i=1}^n \xi_i \sum_{j=1}^n p_{ij} z_j^2$$

$$= \sum_{j=1}^n \sum_{i=1}^n \xi_i p_{ij} z_j^2 = \sum_{j=1}^n \xi_j z_j^2 = \|z\|_\xi^2,$$

where the inequality follows from the convexity of the quadratic function, and the next to last equality follows from the defining property $\sum_{i=1}^n \xi_i p_{ij} = \xi_j$ of the steady-state probabilities.

- Using the lemma, the nonexpansiveness of Π, and the definition $T_\mu J = g + \alpha P J$, we have

$$\|\Pi T_\mu J - \Pi T_\mu \bar{J}\|_\xi \leq \|T_\mu J - T_\mu \bar{J}\|_\xi = \alpha \|P(J - \bar{J})\|_\xi \leq \alpha \|J - \bar{J}\|_\xi$$

for all $J, \bar{J} \in \Re^n$. Hence ΠT_μ is a contraction of modulus α.

PROOF OF ERROR BOUND

- Let Φr^* be the fixed point of ΠT. We have

$$\|J_\mu - \Phi r^*\|_\xi \leq \frac{1}{\sqrt{1-\alpha^2}} \|J_\mu - \Pi J_\mu\|_\xi.$$

Proof: We have

$$\begin{aligned}
\|J_\mu - \Phi r^*\|_\xi^2 &= \|J_\mu - \Pi J_\mu\|_\xi^2 + \left\|\Pi J_\mu - \Phi r^*\right\|_\xi^2 \\
&= \|J_\mu - \Pi J_\mu\|_\xi^2 + \left\|\Pi T J_\mu - \Pi T(\Phi r^*)\right\|_\xi^2 \\
&\leq \|J_\mu - \Pi J_\mu\|_\xi^2 + \alpha^2 \|J_\mu - \Phi r^*\|_\xi^2,
\end{aligned}$$

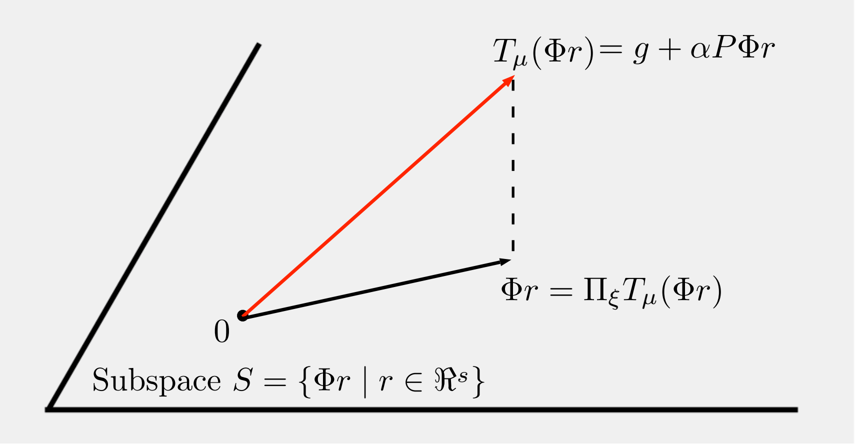
where

- The first equality uses the Pythagorean Theorem
- The second equality holds because J_μ is the fixed point of T and Φr^* is the fixed point of ΠT
- The inequality uses the contraction property of ΠT.

Q.E.D.

SIMULATION-BASED SOLUTION OF PROJECTED EQUATION

MATRIX FORM OF PROJECTED EQUATION


$T_\mu(\Phi r) = g + \alpha P \Phi r$
$\Phi r = \Pi_\xi T_\mu(\Phi r)$
0
Subspace $S = \{\Phi r \mid r \in \Re^s\}$

- The solution Φr^* satisfies the orthogonality condition: The error

$$\Phi r^* - (g + \alpha P \Phi r^*)$$

is "orthogonal" to the subspace spanned by the columns of Φ.

- This is written as

$$\Phi' \Xi \big(\Phi r^* - (g + \alpha P \Phi r^*)\big) = 0,$$

where Ξ is the diagonal matrix with the steady-state probabilities $\xi_1, \ldots, \xi_n$ along the diagonal.

- Equivalently, $Cr^* = d$, where

$$C = \Phi' \Xi (I - \alpha P) \Phi, \qquad d = \Phi' \Xi g$$

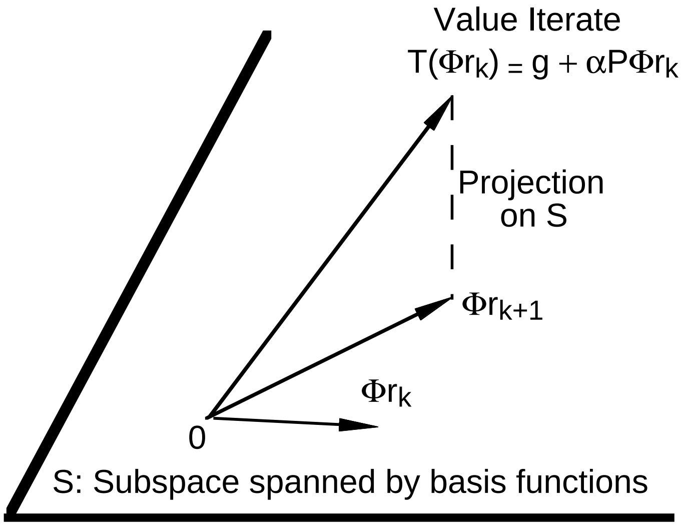
but computing C and d is HARD (high-dimensional inner products).

SOLUTION OF PROJECTED EQUATION

- Solve $Cr^* = d$ by matrix inversion: $r^* = C^{-1}d$
- Projected Value Iteration (PVI) method:

$$\Phi r_{k+1} = \Pi T(\Phi r_k) = \Pi(g + \alpha P \Phi r_k)$$

Converges to r^* because ΠT is a contraction.



- PVI can be written as:

$$r_{k+1} = \arg\min_{r \in \Re^s} \left\| \Phi r - (g + \alpha P \Phi r_k) \right\|_\xi^2$$

By setting to 0 the gradient with respect to r,

$$\Phi' \Xi \big(\Phi r_{k+1} - (g + \alpha P \Phi r_k) \big) = 0,$$

which yields

$$r_{k+1} = r_k - (\Phi' \Xi \Phi)^{-1} (C r_k - d)$$

SIMULATION-BASED IMPLEMENTATIONS

- Key idea: Calculate simulation-based approximations based on k samples

$$C_k \approx C, \qquad d_k \approx d$$

- Matrix inversion $r^* = C^{-1}d$ is approximated by

$$\hat{r}_k = C_k^{-1} d_k$$

This is the LSTD (Least Squares Temporal Differences) Method.

- PVI method $r_{k+1} = r_k - (\Phi'\Xi\Phi)^{-1}(Cr_k - d)$ is approximated by

$$r_{k+1} = r_k - G_k(C_k r_k - d_k)$$

where

$$G_k \approx (\Phi'\Xi\Phi)^{-1}$$

This is the LSPE (Least Squares Policy Evaluation) Method.

- Key fact: C_k, d_k, and G_k can be computed with low-dimensional linear algebra (of order s; the number of basis functions).

SIMULATION MECHANICS

- We generate an infinitely long trajectory $(i_0, i_1, \ldots)$ of the Markov chain, so states i and transitions (i,j) appear with long-term frequencies ξ_i and p_{ij}.

- After generating each transition (i_t, i_{t+1}), we compute the row $\phi(i_t)'$ of Φ and the cost component $g(i_t, i_{t+1})$.

- We form

$$d_k = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) g(i_t, i_{t+1}) \approx \sum_{i,j} \xi_i p_{ij} \phi(i) g(i,j) = \Phi' \Xi g = d$$

$$C_k = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) \big(\phi(i_t) - \alpha \phi(i_{t+1})\big)' \approx \Phi' \Xi (I - \alpha P) \Phi = C$$

Also in the case of LSPE

$$G_k = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) \phi(i_t)' \approx \Phi' \Xi \Phi$$

- Convergence based on law of large numbers.

- C_k, d_k, and G_k can be formed incrementally. Also can be written using the formalism of temporal differences (this is just a matter of style)

OPTIMISTIC VERSIONS

- Instead of calculating nearly exact approximations $C_k \approx C$ and $d_k \approx d$, we do a less accurate approximation, based on few simulation samples

- Evaluate (coarsely) current policy μ, then do a policy improvement

- This often leads to faster computation (as optimistic methods often do)

- Very complex behavior (see the subsequent discussion on oscillations)

- The matrix inversion/LSTD method has serious problems due to large simulation noise (because of limited sampling) - particularly if the C matrix is ill-conditioned

- LSPE tends to cope better because of its iterative nature (this is true of other iterative methods as well)

- A stepsize $\gamma \in (0, 1]$ in LSPE may be useful to damp the effect of simulation noise

$$r_{k+1} = r_k - \gamma G_k (C_k r_k - d_k)$$

MULTISTEP PROJECTED EQUATIONS

MULTISTEP METHODS

- Introduce a multistep version of Bellman's equation $J = T^{(\lambda)}J$, where for $\lambda \in [0,1)$,

$$T^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty} \lambda^\ell T^{\ell+1}$$

Geometrically weighted sum of powers of T.

- Note that T^ℓ is a contraction with modulus α^ℓ, with respect to the weighted Euclidean norm $\|\cdot\|_\xi$, where ξ is the steady-state probability vector of the Markov chain.

- Hence $T^{(\lambda)}$ is a contraction with modulus

$$\alpha_\lambda = (1-\lambda)\sum_{\ell=0}^{\infty} \alpha^{\ell+1}\lambda^\ell = \frac{\alpha(1-\lambda)}{1-\alpha\lambda}$$

Note that $\alpha_\lambda \to 0$ as $\lambda \to 1$

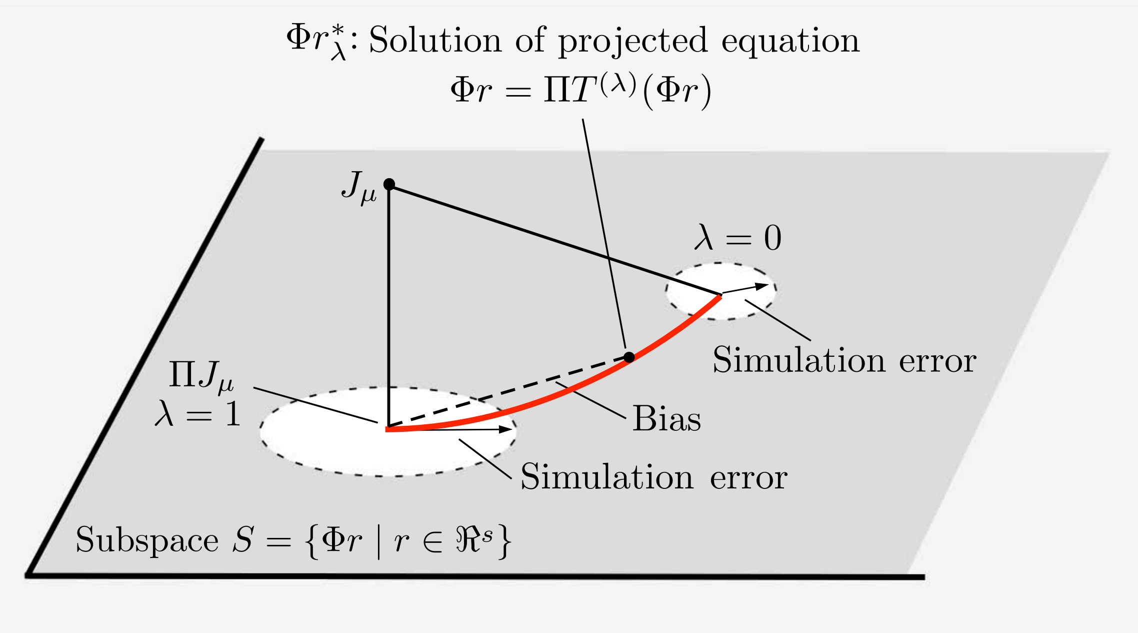
- T^ℓ and $T^{(\lambda)}$ have the same fixed point J_μ and

$$\|J_\mu - \Phi r_\lambda^*\|_\xi \leq \frac{1}{\sqrt{1-\alpha_\lambda^2}}\|J_\mu - \Pi J_\mu\|_\xi$$

where Φr_λ^* is the fixed point of $\Pi T^{(\lambda)}$.

- The fixed point Φr_λ^* depends on λ.

BIAS-VARIANCE TRADEOFF


Φr_λ^*: Solution of projected equation
$\Phi r = \Pi T^{(\lambda)}(\Phi r)$
J_μ
$\lambda = 0$
Simulation error
ΠJ_μ
$\lambda = 1$
Bias
Simulation error
Subspace $S = \{\Phi r \mid r \in \Re^s\}$

- Error bound $\|J_\mu - \Phi r_\lambda^*\|_\xi \leq \frac{1}{\sqrt{1-\alpha_\lambda^2}} \|J_\mu - \Pi J_\mu\|_\xi$

- As $\lambda \uparrow 1$, we have $\alpha_\lambda \downarrow 0$, so error bound (and the quality of approximation) improves as $\lambda \uparrow 1$. In fact

$$\lim_{\lambda \uparrow 1} \Phi r_\lambda^* = \Pi J_\mu$$

- But the simulation noise in approximating

$$T^{(\lambda)} = (1 - \lambda) \sum_{\ell=0}^{\infty} \lambda^\ell T^{\ell+1}$$

increases

- Choice of λ is usually based on trial and error

MULTISTEP PROJECTED EQ. METHODS

- The projected Bellman equation is

$$\Phi r = \Pi T^{(\lambda)}(\Phi r)$$

- In matrix form: $C^{(\lambda)} r = d^{(\lambda)}$, where

$$C^{(\lambda)} = \Phi' \Xi \big(I - \alpha P^{(\lambda)}\big)\Phi, \qquad d^{(\lambda)} = \Phi' \Xi g^{(\lambda)},$$

with

$$P^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty} \alpha^\ell \lambda^\ell P^{\ell+1}, \quad g^{(\lambda)} = \sum_{\ell=0}^{\infty} \alpha^\ell \lambda^\ell P^\ell g$$

- The LSTD(λ) method is

$$\big(C_k^{(\lambda)}\big)^{-1} d_k^{(\lambda)},$$

where $C_k^{(\lambda)}$ and $d_k^{(\lambda)}$ are simulation-based approximations of $C^{(\lambda)}$ and $d^{(\lambda)}$.

- The LSPE(λ) method is

$$r_{k+1} = r_k - \gamma G_k \big(C_k^{(\lambda)} r_k - d_k^{(\lambda)}\big)$$

where G_k is a simulation-based approx. to $(\Phi' \Xi \Phi)^{-1}$

- TD(λ): An important simpler/slower iteration [similar to LSPE(λ) with $G_k = I$ - see the text].

MORE ON MULTISTEP METHODS

- The simulation process to obtain $C_k^{(\lambda)}$ and $d_k^{(\lambda)}$ is similar to the case $\lambda = 0$ (single simulation trajectory $i_0, i_1, \ldots$, more complex formulas)

$$C_k^{(\lambda)} = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) \sum_{m=t}^{k} \alpha^{m-t} \lambda^{m-t} \big(\phi(i_m) - \alpha\phi(i_{m+1})\big)'$$

$$d_k^{(\lambda)} = \frac{1}{k+1} \sum_{t=0}^{k} \phi(i_t) \sum_{m=t}^{k} \alpha^{m-t} \lambda^{m-t} g_{i_m}$$

- In the context of approximate policy iteration, we can use optimistic versions (few samples between policy updates).
- Many different versions (see the text).
- Note the λ-tradeoffs:
  - As $\lambda \uparrow 1$, $C_k^{(\lambda)}$ and $d_k^{(\lambda)}$ contain more "simulation noise", so more samples are needed for a close approximation of r_λ (the solution of the projected equation)
  - The error bound $\|J_\mu - \Phi r_\lambda\|_\xi$ becomes smaller
  - As $\lambda \uparrow 1$, $\Pi T^{(\lambda)}$ becomes a contraction for arbitrary projection norm

6.231 DYNAMIC PROGRAMMING

LECTURE 5

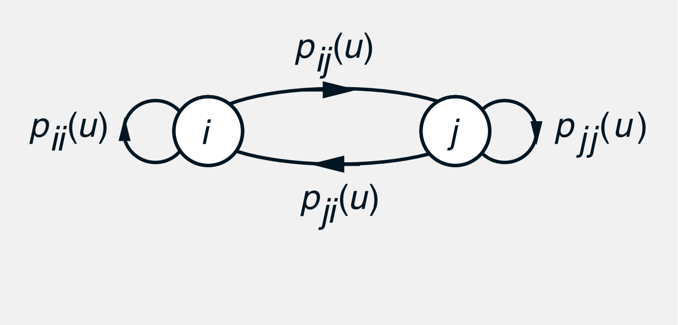
LECTURE OUTLINE

- Review of approximate PI based on projected Bellman equations
- Issues of policy improvement
  - Exploration enhancement in policy evaluation
  - Oscillations in approximate PI
- Aggregation – An alternative to the projected equation/Galerkin approach
- Examples of aggregation
- Simulation-based aggregation
- Relation between aggregation and projected equations

REVIEW

DISCOUNTED MDP

- System: Controlled Markov chain with states $i = 1, \ldots, n$ and finite set of controls $u \in U(i)$
- Transition probabilities: $p_{ij}(u)$



- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$ starting at state i:

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

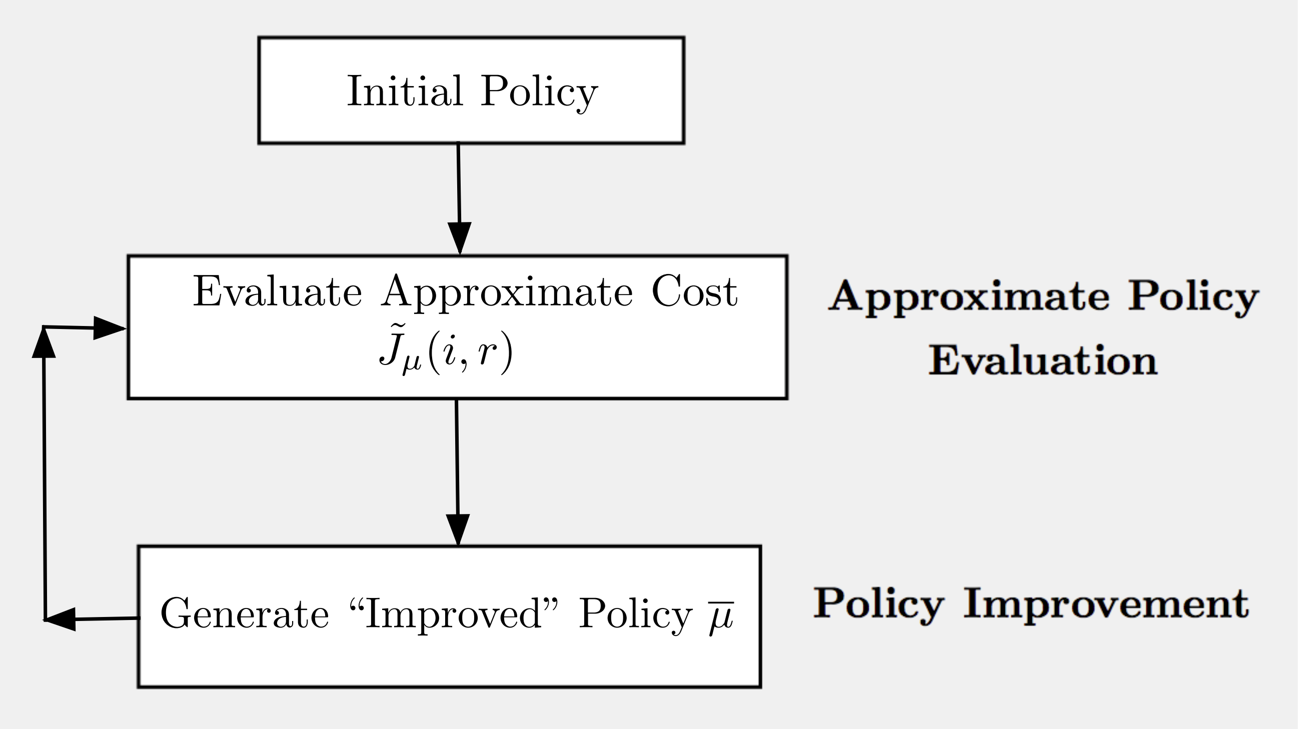
with $\alpha \in [0,1)$

- Shorthand notation for DP mappings

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

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

APPROXIMATE PI


Initial Policy
Evaluate Approximate Cost $\tilde{J}_\mu(i,r)$
Approximate Policy Evaluation
Generate "Improved" Policy $\overline{\mu}$
Policy Improvement

- Evaluation of typical policy μ: Linear cost function approximation

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

where Φ is full rank $n \times s$ matrix with columns the basis functions, and ith row denoted $\phi(i)'$.

- Policy "improvement" to generate $\overline{\mu}$:

$$\overline{\mu}(i) = \arg\min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha\phi(j)'r\big)$$

EVALUATION BY PROJECTED EQUATIONS

- Approximate policy evaluation by solving

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

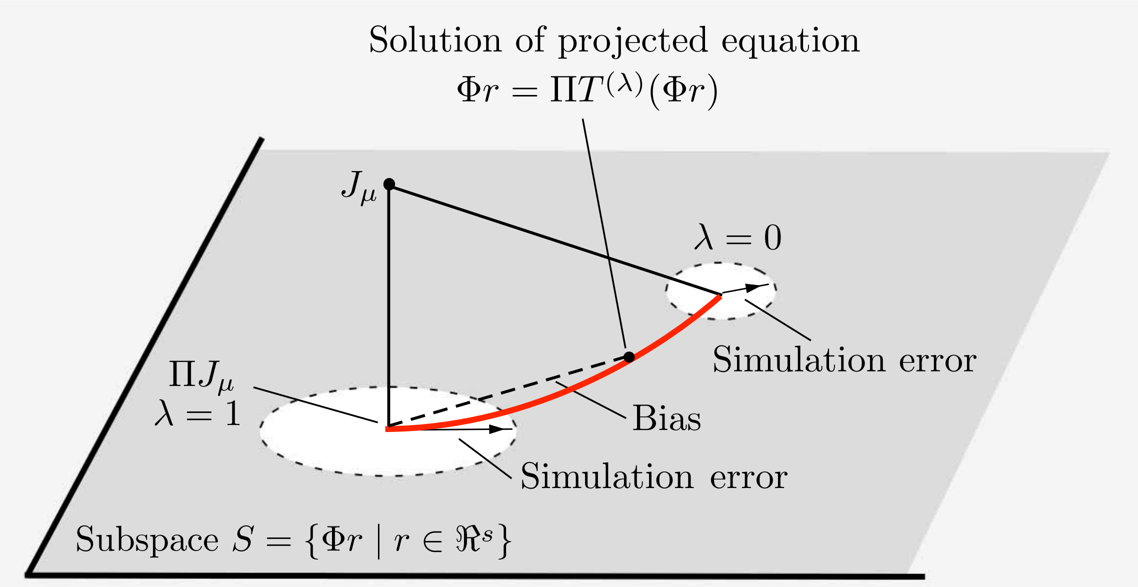
Π: weighted Euclidean projection; special nature of the steady-state distribution weighting.

- Implementation by simulation (single long trajectory using current policy - important to make ΠT_μ a contraction). LSTD, LSPE methods.

- Multistep option: Solve $\Phi r = \Pi T_\mu^{(\lambda)}(\Phi r)$ with

$$T_\mu^{(\lambda)} = (1-\lambda)\sum_{\ell=0}^{\infty} \lambda^\ell T_\mu^{\ell+1}, \qquad 0 \le \lambda < 1$$

  - As $\lambda \uparrow 1$, $\Pi T_\mu^{(\lambda)}$ becomes a contraction for any projection norm (allows changes in Π)
  - Bias-variance tradeoff


Solution of projected equation
$\Phi r = \Pi T^{(\lambda)}(\Phi r)$
J_μ
$\lambda = 0$
Simulation error
ΠJ_μ
$\lambda = 1$
Bias
Simulation error
Subspace $S = \{\Phi r \mid r \in \Re^s\}$

ISSUES OF POLICY IMPROVEMENT

EXPLORATION

- **1st major issue:** exploration. To evaluate μ, we need to generate cost samples using μ

- 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, and seriously impact 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).

- To deal with this we must change the sampling mechanism and modify the simulation formulas.

- Solve

$$\Phi r = \overline{\Pi} T_\mu(\Phi r)$$

where $\overline{\Pi}$ is projection with respect to an exploration-enhanced norm [uses a weight distribution $\zeta = (\zeta_1, \ldots, \zeta_n)$].

- ζ is more "balanced" than ξ the steady-state distribution of the Markov chain of μ.

- This also addresses any lack of ergodicity of μ.

EXPLORATION MECHANISMS

- One possibility: Use multiple short simulation trajectories instead of single long trajectory starting from a rich mixture of states. This is known as geometric sampling, or free-form sampling.
  - By properly choosing the starting states, we enhance exploration
  - The simulation formulas for LSTD(λ) and LSPE(λ) have to be modified to yield the solution of $\Phi r = \overline{\Pi} T_\mu^{(\lambda)}(\Phi r)$ (see the DP text)
- Another possibility: Use a modified policy to generate a single long trajectory. This is called an off-policy approach.
  - Modify the transition probabilities of μ to enhance exploration
  - Again the simulation formulas for LSTD(λ) and LSPE(λ) have to be modified to yield the solution of $\Phi r = \overline{\Pi} T_\mu^{(\lambda)}(\Phi r)$ (use of importance sampling; see the DP text)
- With larger values of $\lambda > 0$ the contraction property of $\overline{\Pi} T_\mu^{(\lambda)}$ is maintained.
- LSTD may be used without $\overline{\Pi} T_\mu^{(\lambda)}$ being a contraction ... LSPE and TD require a contraction.

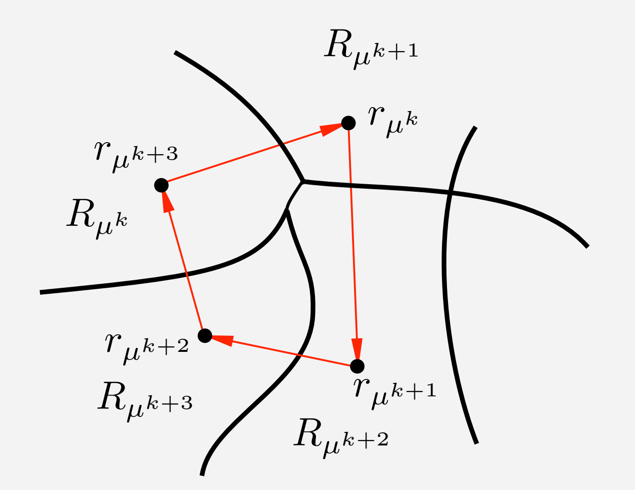
POLICY ITERATION ISSUES: OSCILLATIONS

- **2nd major issue:** oscillation of policies

- Analysis using the greedy partition of the space of weights r: R_μ is the set of parameter vectors r for which μ is greedy with respect to $\tilde{J}(\cdot; r) = \Phi r$

$$R_\mu = \big\{ r \mid T_\mu(\Phi r) = T(\Phi r) \big\} \qquad \forall\ \mu$$

If we use r in R_μ the next "improved" policy is μ



- If policy evaluation is exact, there is a finite number of possible vectors r_μ, (one per μ)

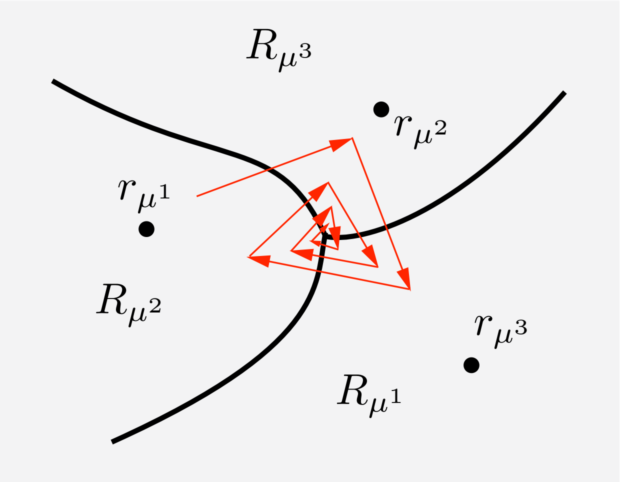
- The algorithm ends up repeating some cycle of policies $\mu^k, \mu^{k+1}, \ldots, \mu^{k+m}$ with

$$r_{\mu^k} \in R_{\mu^{k+1}}, \; r_{\mu^{k+1}} \in R_{\mu^{k+2}}, \ldots, r_{\mu^{k+m}} \in R_{\mu^k}$$

- Many different cycles are possible

MORE ON OSCILLATIONS/CHATTERING

- In the case of optimistic policy iteration a different picture holds (policy evaluation does not produce exactly r_μ)



- Oscillations of weight vector r are less violent, but the "limit" point is meaningless!

- Fundamentally, oscillations are due to the lack of monotonicity of the projection operator, i.e., $J \leq J'$ does not imply $\Pi J \leq \Pi J'$.

- If approximate PI uses an evaluation of the form

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

with W: monotone and $W T_\mu$: contraction, the policies converge (to a possibly nonoptimal limit).

- These conditions hold when aggregation is used

AGGREGATION

PROBLEM APPROXIMATION - AGGREGATION

- Another major idea in ADP is to approximate J^* or J_μ with the cost-to-go functions of a simpler problem.

- Aggregation is a 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 relating original system states with aggregate states
  - Solve (exactly or approximately) the "aggregate" problem by any kind of VI or PI method (including simulation-based methods)

- If $\hat{R}(y)$ is the optimal cost of aggregate state y, we use the approximation

$$J^*(j) \approx \sum_y \phi_{jy} \hat{R}(y), \qquad \forall\ j$$

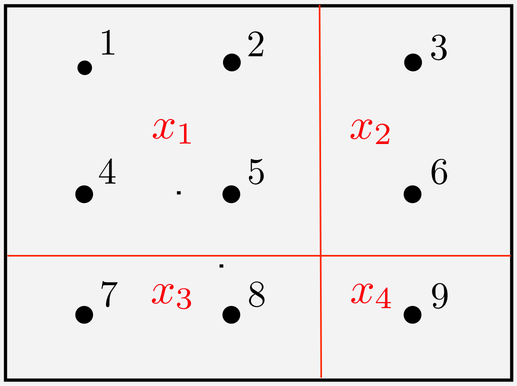
where ϕ_{jy} are the aggregation probabilities, encoding the "degree of membership of j in the aggregate state y"

- This is a linear architecture: ϕ_{jy} are the features of state j

HARD AGGREGATION EXAMPLE

- Group the original system states into subsets, and view each subset as an aggregate state

- Aggregation probs.: $\phi_{jy} = 1$ if j belongs to aggregate state y (piecewise constant approx).



$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

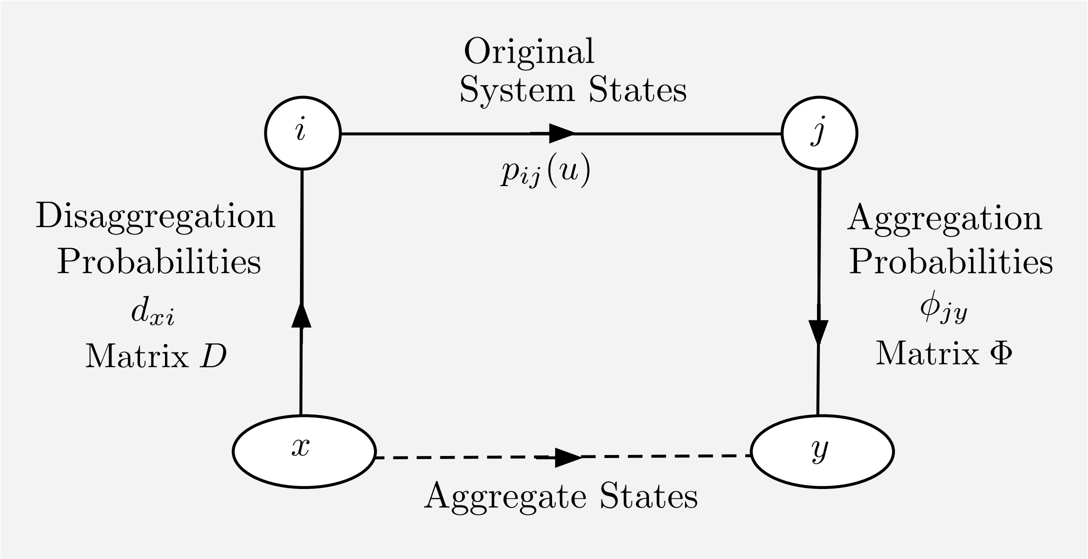
- What should be the “aggregate” transition probs. out of x?

- Select $i \in x$ and use the transition probs. of i. But which i should I use?

- The simplest possibility is to assume that all states i in x are equally likely.

- A generalization is to randomize, i.e., use “disaggregation probabilities” d_{xi}: Roughly, the “degree to which i is representative of x.”

AGGREGATION/DISAGGREGATION PROBS



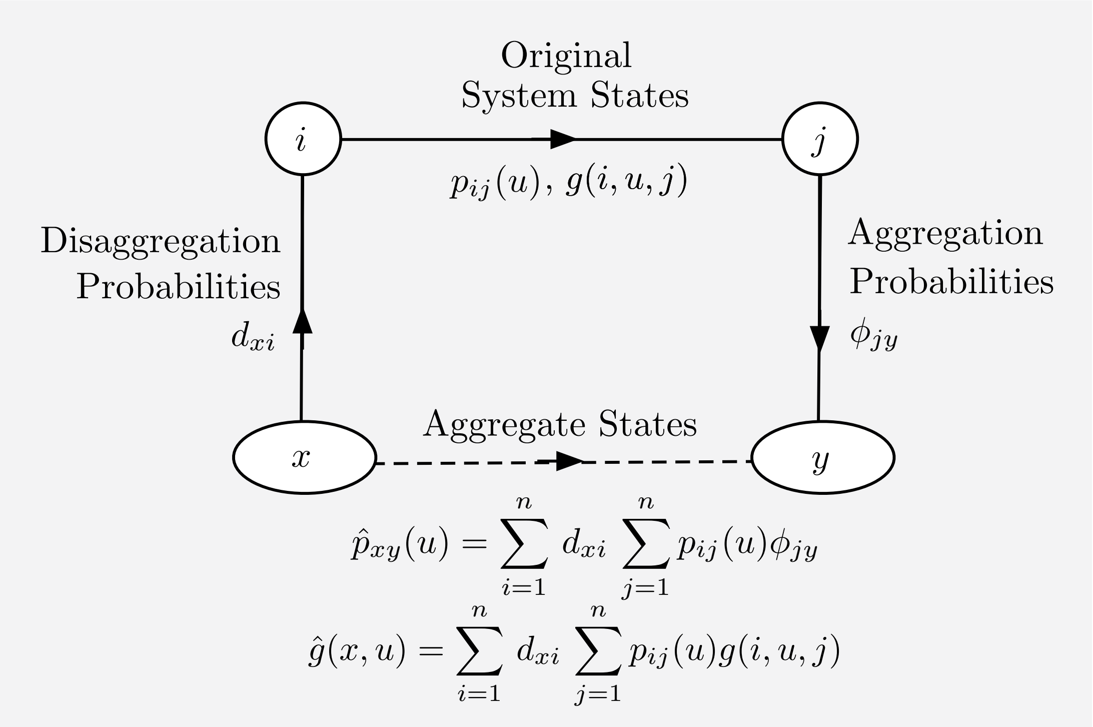
- Define the aggregate system transition probabilities via two (somewhat arbitrary) choices.

- For each original system state j and aggregate state y, the aggregation probability ϕ_{jy}
  - Roughly, the "degree of membership of j in the aggregate state y."
  - In hard aggregation, $\phi_{jy} = 1$ if state j belongs to aggregate state/subset y.

- For each aggregate state x and original system state i, the disaggregation probability d_{xi}
  - Roughly, the "degree to which i is representative of x."

- Aggregation scheme is defined by the two matrices D and Φ. The rows of D and Φ must be probability distributions.

AGGREGATE SYSTEM DESCRIPTION


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

- The transition probability from aggregate state x to aggregate state y under control u

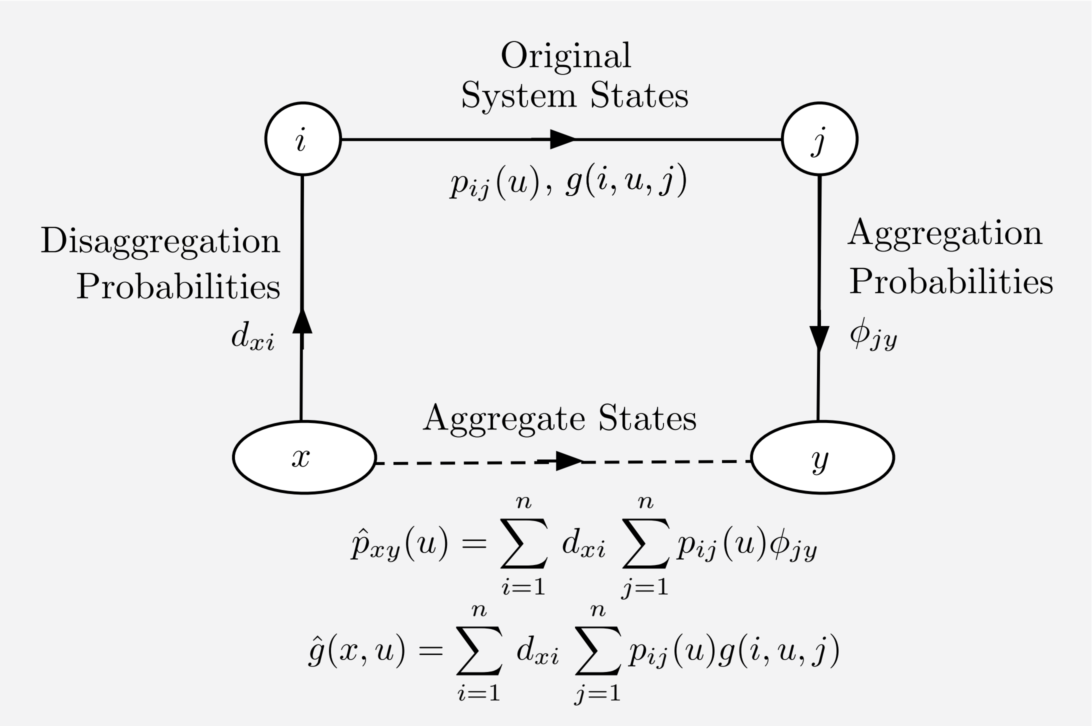
$$\hat{p}_{xy}(u) = \sum_{i=1}^{n} d_{xi} \sum_{j=1}^{n} p_{ij}(u)\phi_{jy}, \quad \text{or } \hat{P}(u) = DP(u)\Phi$$

where the rows of D and Φ are the disaggregation and aggregation probs.

- The expected transition cost is

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

AGGREGATE BELLMAN'S EQUATION


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

- The optimal cost function of the aggregate problem, denoted $\hat{R}$, is

$$\hat{R}(x) = \min_{u \in U} \left[\hat{g}(x,u) + \alpha \sum_{y} \hat{p}_{xy}(u)\hat{R}(y) \right], \qquad \forall\, x$$

Bellman's equation for the aggregate problem.

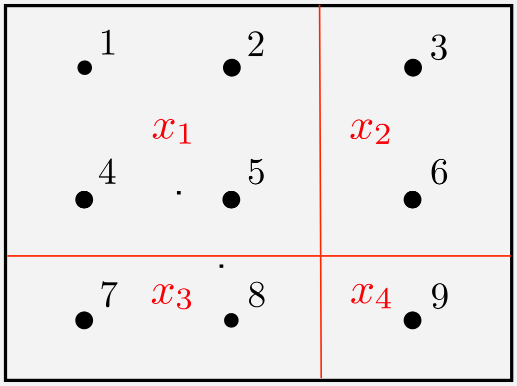
- The optimal cost function J^* of the original problem is approximated by $\tilde{J}$ given by

$$\tilde{J}(j) = \sum_{y} \phi_{jy}\hat{R}(y), \qquad \forall\ j$$

EXAMPLE I: HARD AGGREGATION

- Group the original system states into subsets, and view each subset as an aggregate state

- Aggregation probs.: $\phi_{jy} = 1$ if j belongs to aggregate state y.



$$\Phi = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Disaggregation probs.: There are many possibilities, e.g., all states i within aggregate state x have equal prob. d_{xi}.

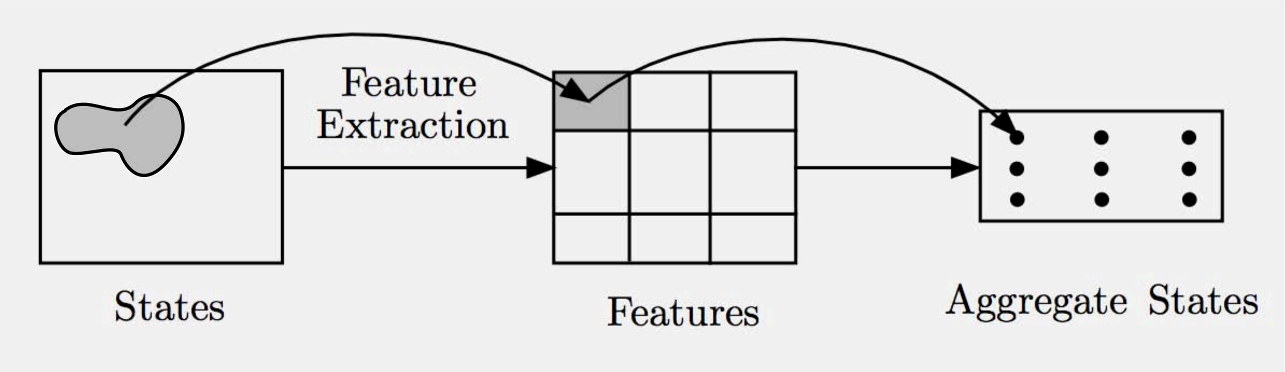
- If optimal cost vector J^* is piecewise constant over the aggregate states/subsets, hard aggregation is exact. Suggests grouping states with “roughly equal” cost into aggregates.

- A variant: Soft aggregation (provides “soft boundaries” between aggregate states).

EXAMPLE II: FEATURE-BASED AGGREGATION

- Important question: How do we group states together?

- If we know good features, it makes sense to group together states that have "similar features"



- A general approach for passing from a feature-based state representation to a hard aggregation-based architecture

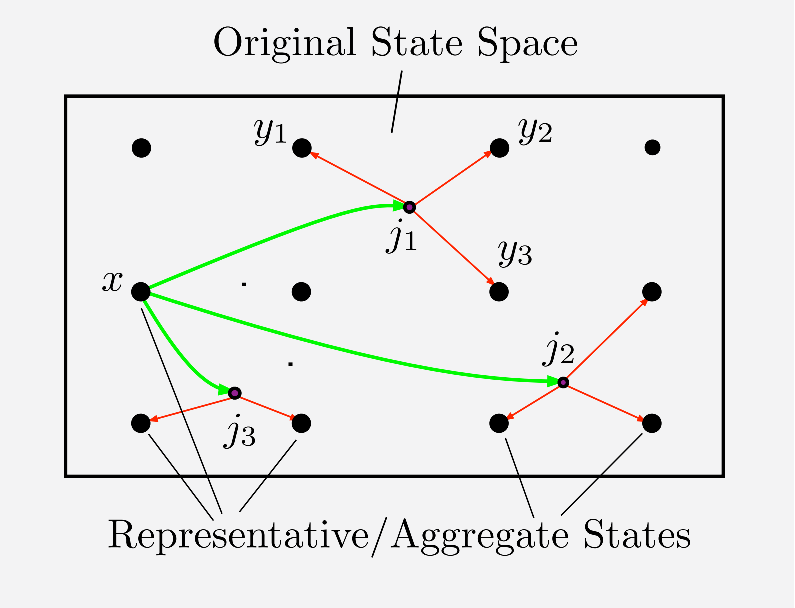
- Essentially discretize the features and generate a corresponding piecewise constant approximation to the optimal cost function

- Aggregation-based architecture is more powerful (it is nonlinear in the features)

- ... but may require many more aggregate states to reach the same level of performance as the corresponding linear feature-based architecture

EXAMPLE III: REP. STATES/COARSE GRID

- Choose a collection of "representative" original system states, and associate each one of them with an aggregate state



- Disaggregation probabilities are $d_{xi} = 1$ if i is equal to representative state x.

- Aggregation probabilities associate original system states with convex combinations of representative states

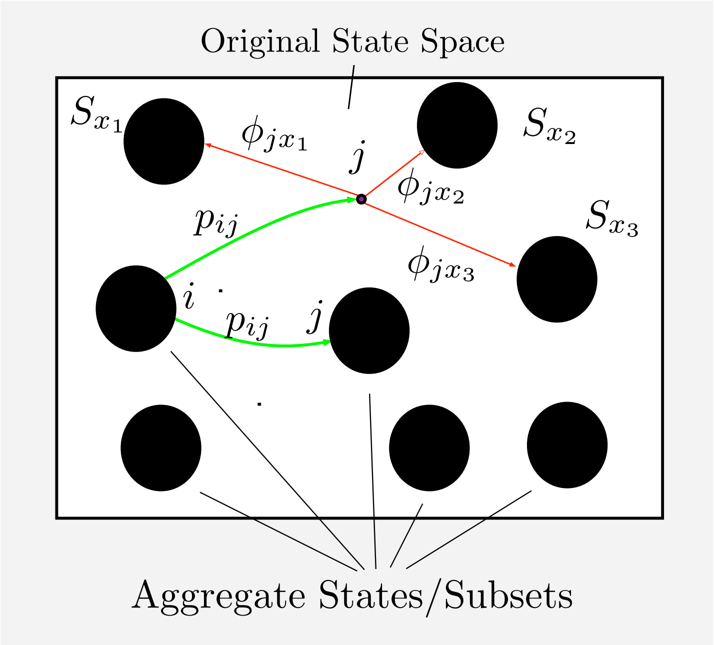
$$j \sim \sum_{y \in \mathcal{A}} \phi_{jy} y$$

- Well-suited for Euclidean space discretization

- Extends nicely to continuous state space, including belief space of POMDP

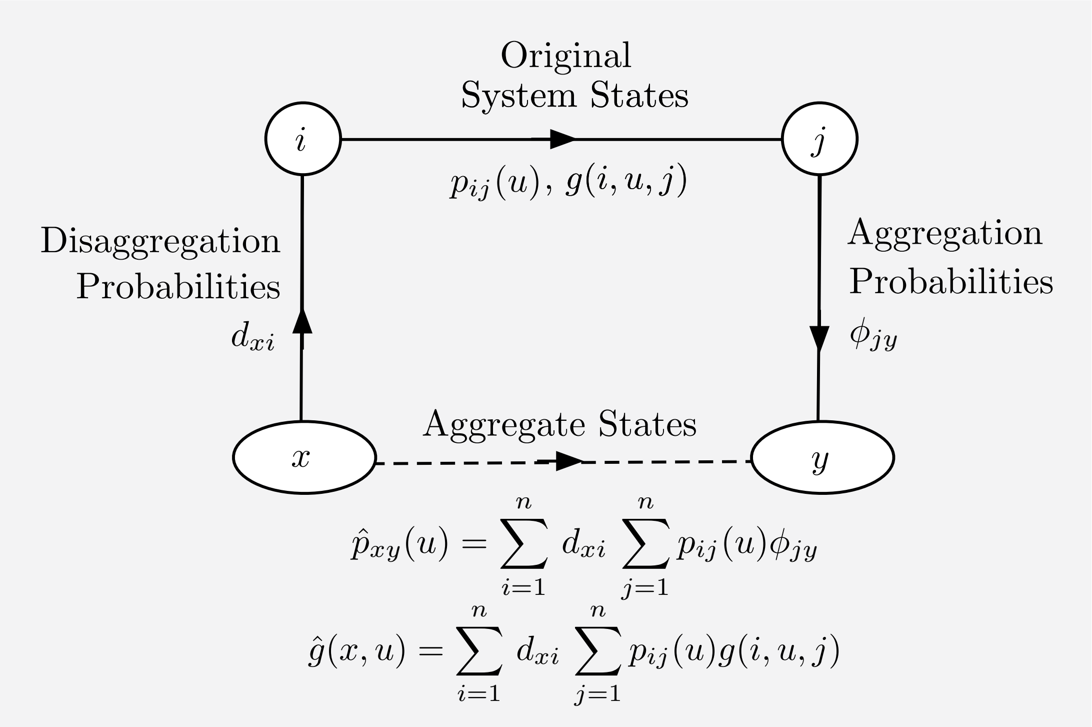
EXAMPLE IV: REPRESENTATIVE FEATURES

- Here the aggregate states are nonempty subsets of original system states. Common case: Each S_x is a group of states with “similar features”



- Restrictions:
  - The aggregate states/subsets are disjoint.
  - The disaggregation probabilities satisfy $d_{xi} > 0$ if and only if $i \in x$.
  - The aggregation probabilities satisfy $\phi_{jy} = 1$ for all $j \in y$.
- Hard aggregation is a special case: $\cup_x S_x = \{1, \ldots, n\}$
- Aggregation with representative states is a special case: S_x consists of just one state

APPROXIMATE PI BY AGGREGATION



Original System States

i j

$p_{ij}(u),\ g(i,u,j)$

Disaggregation Probabilities d_{xi}

Aggregation Probabilities ϕ_{jy}

Aggregate States

x y

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

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

- Consider approximate PI for the original problem, with policy evaluation done by aggregation.

- Evaluation of policy μ: $\tilde{J} = \Phi R$, where $R = DT_\mu(\Phi R)$ (R is the vector of costs of aggregate states for μ). Can be done by simulation.

- Looks like projected equation $\Phi R = \Pi T_\mu(\Phi R)$ (but with ΦD in place of Π).

- Advantage: It has no problem with oscillations.

- Disadvantage: The rows of D and Φ must be probability distributions.

ADDITIONAL ISSUES OF AGGREGATION

ALTERNATIVE POLICY ITERATION

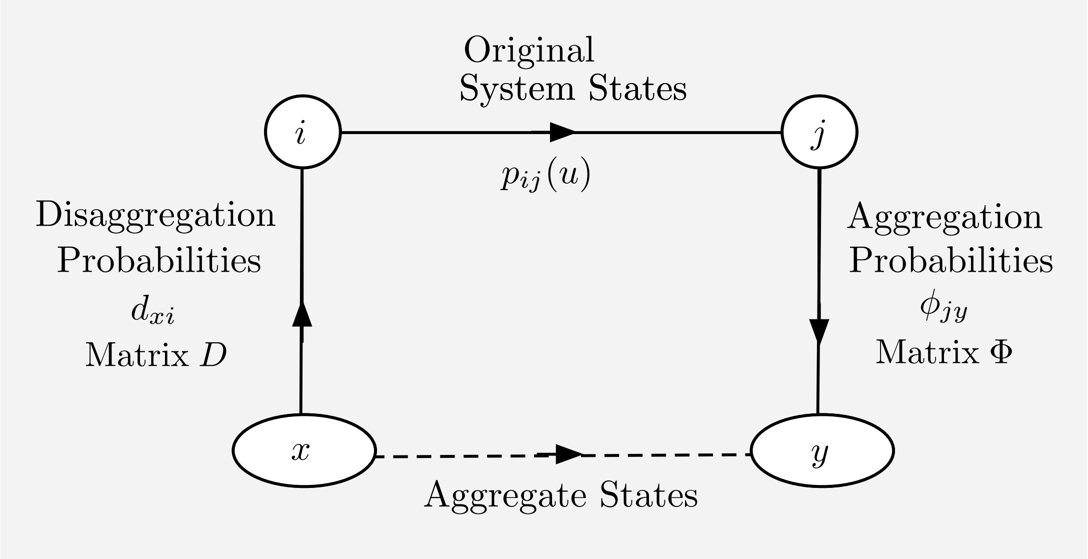
- The preceding PI method uses policies that assign a control to each aggregate state.

- An alternative is to use PI for the combined system, involving the Bellman equations:

$$R^*(x) = \sum_{i=1}^{n} d_{xi} \tilde{J}_0(i), \qquad \forall\, x,$$

$$\tilde{J}_0(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_1(j)\big), \quad i = 1, \ldots, n,$$

$$\tilde{J}_1(j) = \sum_{y \in \mathcal{A}} \phi_{jy} R^*(y), \qquad j = 1, \ldots, n.$$



- Simulation-based PI and VI are still possible.

RELATION OF AGGREGATION/PROJECTION

- Compare aggregation and projected equations

$$\Phi R = \Phi D T(\Phi R), \qquad \Phi r = \Pi T(\Phi r)$$

- If ΦD is a projection (with respect to some weighted Euclidean norm), then the methodology of projected equations applies to aggregation

- Hard aggregation case: ΦD can be verified to be projection with respect to weights ξ_i proportional to the disaggregation probabilities d_{xi}

- Aggregation with representative features case: ΦD can be verified to be a semi-norm projection with respect to weights ξ_i proportional to d_{xi}

- A (weighted) Euclidean semi-norm is defined by $\|J\|_\xi = \sqrt{\sum_{i=1}^n \xi_i \big(J(i)\big)^2}$, where $\xi = (\xi_1, \ldots, \xi_n)$, with $\xi_i \geq 0$.

- If $\Phi' \Xi \Phi$ is invertible, the entire theory and algorithms of projected equations generalizes to semi-norm projected equations [including multi-step methods such as LSTD/LSPE/TD(λ)].

- Reference: Yu and Bertsekas, “Weighted Bellman Equations and their Applications in Approximate Dynamic Programming,” MIT Report, 2012.

DISTRIBUTED AGGREGATION I

- We consider decomposition/distributed solution of large-scale discounted DP problems by hard aggregation.
- Partition the original system states into subsets $S_1, \ldots, S_m$.
- Distributed VI Scheme: Each subset S_ℓ
  - Maintains detailed/exact local costs

$$J(i) \quad \text{for every original system state } i \in S_\ell$$

    using aggregate costs of other subsets
  - Maintains an aggregate cost $R(\ell) = \sum_{i \in S_\ell} d_{\ell i} J(i)$
  - Sends $R(\ell)$ to other aggregate states
- $J(i)$ and $R(\ell)$ are updated by VI according to

$$J_{k+1}(i) = \min_{u \in U(i)} H_\ell(i, u, J_k, R_k), \qquad \forall\ i \in S_\ell$$

with R_k being the vector of $R(\ell)$ at time k, and

$$H_\ell(i, u, J, R) = \sum_{j=1}^{n} p_{ij}(u) g(i, u, j) + \alpha \sum_{j \in S_\ell} p_{ij}(u) J(j)$$
$$+ \alpha \sum_{j \in S_{\ell'},\, \ell' \neq \ell} p_{ij}(u) R(\ell')$$

DISTRIBUTED AGGREGATION II

- Can show that this iteration involves a sup-norm contraction mapping of modulus α, so it converges to the unique solution of the system of equations in (J, R)

$$J(i) = \min_{u \in U(i)} H_\ell(i, u, J, R), \quad R(\ell) = \sum_{i \in S_\ell} d_{\ell i} J(i),$$
$$\forall\, i \in S_\ell,\ \ell = 1, \ldots, m.$$

- This follows from the fact that $\{d_{\ell i} \mid i = 1, \ldots, n\}$ is a probability distribution.

- View these equations as a set of Bellman equations for an "aggregate" DP problem. The difference is that the mapping H involves $J(j)$ rather than $R\big(x(j)\big)$ for $j \in S_\ell$.

- In an asynchronous version of the method, the aggregate costs $R(\ell)$ may be outdated to account for communication "delays" between aggregate states.

- Convergence can be shown using the general theory of asynchronous distributed computation, briefly described in the 2nd lecture (see the text).

6.231 DYNAMIC PROGRAMMING

LECTURE 6

LECTURE OUTLINE

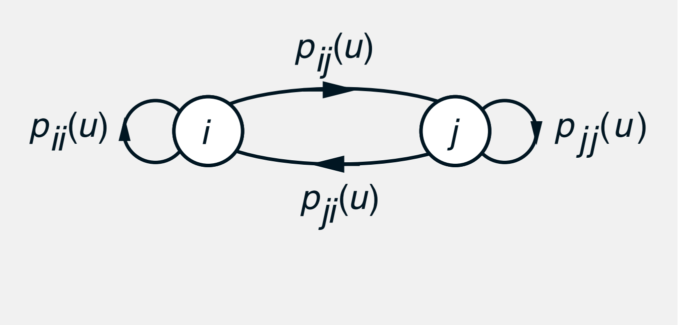
- Review of Q-factors and Bellman equations for Q-factors
- VI and PI for Q-factors
- Q-learning - Combination of VI and sampling
- Q-learning and cost function approximation
- Adaptive dynamic programming
- Approximation in policy space
- Additional topics

REVIEW

DISCOUNTED MDP

- System: Controlled Markov chain with states $i = 1, \ldots, n$ and finite set of controls $u \in U(i)$

- Transition probabilities: $p_{ij}(u)$



- Cost of a policy $\pi = \{\mu_0, \mu_1, \ldots\}$ starting at state i:

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

with $\alpha \in [0, 1)$

- Shorthand notation for DP mappings

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

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

BELLMAN EQUATIONS FOR Q-FACTORS

- The optimal Q-factors are defined by

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

- Since $J^* = TJ^*$, we have $J^*(i) = \min_{u \in U(i)} Q^*(i,u)$ so the optimal Q-factors solve the equation

$$Q^*(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{u' \in U(j)} Q^*(j,u')\right)$$

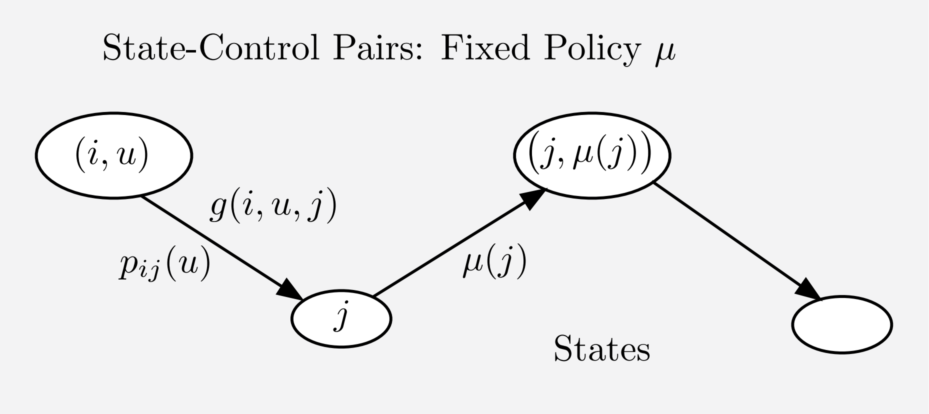
- Equivalently $Q^* = FQ^*$, where

$$(FQ)(i,u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i,u,j) + \alpha \min_{u' \in U(j)} Q(j,u')\right)$$

- This is Bellman's Eq. for a system whose states are the pairs (i,u)

- Similar mapping F_μ and Bellman equation for a policy μ: $Q_\mu = F_\mu Q_\mu$

BELLMAN EQ FOR Q-FACTORS OF A POLICY



- Q-factors of a policy μ: For all (i,u)

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

Equivalently $Q_\mu = F_\mu Q_\mu$, where

$$(F_\mu Q)(i,u)=\sum_{j=1}^n p_{ij}(u)\Big(g(i,u,j)+\alpha Q\big(j,\mu(j)\big)\Big)$$

- This is a linear equation. It can be used for policy evaluation.

- Generally VI and PI can be carried out in terms of Q-factors.

- When done exactly they produce results that are mathematically equivalent to cost-based VI and PI.

WHAT IS GOOD AND BAD ABOUT Q-FACTORS

- All the exact theory and algorithms for costs applies to Q-factors
  - Bellman's equations, contractions, optimality conditions, convergence of VI and PI
- All the approximate theory and algorithms for costs applies to Q-factors
  - Projected equations, sampling and exploration issues, oscillations, aggregation
- A MODEL-FREE (on-line) controller implementation
  - Once we calculate $Q^*(i,u)$ for all (i,u),

$$\mu^*(i) = \arg \min_{u \in U(i)} Q^*(i,u), \qquad \forall\ i$$

  - Similarly, once we calculate a parametric approximation $\tilde{Q}(i,u;r)$ for all (i,u),

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

- The main bad thing: Greater dimension and more storage! (It can be used for large-scale problems only through aggregation, or other approximation.)

Q-LEARNING

Q-LEARNING

- In addition to the approximate PI methods adapted for Q-factors, there is an important additional algorithm:
  - Q-learning, a sampled form of VI (a stochastic iterative algorithm).
- Q-learning algorithm (in its classical form):
  - Sampling: Select sequence of pairs (i_k, u_k) [use any probabilistic mechanism for this, but all (i, u) are chosen infinitely often].
  - Iteration: For each k, select j_k according to $p_{i_k j}(u_k)$. Update just $Q(i_k, u_k)$:

$$Q_{k+1}(i_k,u_k) = (1 - \gamma_k)Q_k(i_k, u_k) + \gamma_k\left(g(i_k, u_k, j_k) + \alpha \min_{u' \in U(j_k)} Q_k(j_k, u')\right)$$

    Leave unchanged all other Q-factors.
  - Stepsize conditions: $\gamma_k \downarrow 0$
- We move $Q(i, u)$ in the direction of a sample of

$$(FQ)(i, u) = \sum_{j=1}^{n} p_{ij}(u)\left(g(i, u, j) + \alpha \min_{u' \in U(j)} Q(j, u')\right)$$

NOTES AND QUESTIONS ABOUT Q-LEARNING

$$
\begin{aligned}
Q_{k+1}(i_k,u_k) = {} & (1-\gamma_k)Q_k(i_k,u_k) \\
& + \gamma_k\left(g(i_k,u_k,j_k) + \alpha \min_{u' \in U(j_k)} Q_k(j_k,u')\right)
\end{aligned}
$$

- Model free implementation. We just need a simulator that given (i,u) produces next state j and cost $g(i,u,j)$

- Operates on only one state-control pair at a time. Convenient for simulation, no restrictions on sampling method. (Connection with asynchronous algorithms.)

- Aims to find the (exactly) optimal Q-factors.

- Why does it converge to Q^*?

- Why can't I use a similar algorithm for optimal costs (a sampled version of VI)?

- Important mathematical (fine) point: In the Q-factor version of Bellman's equation the order of expectation and minimization is reversed relative to the cost version of Bellman's equation:

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

CONVERGENCE ASPECTS OF Q-LEARNING

- Q-learning can be shown to converge to true/exact Q-factors (under mild assumptions).
- The proof is sophisticated, based on theories of stochastic approximation and asynchronous algorithms.
- Uses the fact that the Q-learning map F:

$$(FQ)(i,u) = E_j\Big\{g(i,u,j) + \alpha \min_{u'} Q(j,u')\Big\}$$

is a sup-norm contraction.

- Generic stochastic approximation algorithm:
  - Consider generic fixed point problem involving an expectation:

$$x = E_w\big\{f(x,w)\big\}$$

  - Assume $E_w\big\{f(x,w)\big\}$ is a contraction with respect to some norm, so the iteration

$$x_{k+1} = E_w\big\{f(x_k,w)\big\}$$

converges to the unique fixed point

  - Approximate $E_w\big\{f(x,w)\big\}$ by sampling

STOCH. APPROX. CONVERGENCE IDEAS

- Generate a sequence of samples $\{w_1, w_2, \ldots\}$, and approximate the convergent fixed point iteration $x_{k+1} = E_w\big\{f(x_k, w)\big\}$

- At each iteration k use the approximation

$$x_{k+1} = \frac{1}{k}\sum_{t=1}^{k} f(x_k, w_t) \approx E_w\big\{f(x_k, w)\big\}$$

- A major flaw: it requires, for each k, the computation of $f(x_k, w_t)$ for all values w_t, $t = 1, \ldots, k$.

- This motivates the more convenient iteration

$$x_{k+1} = \frac{1}{k}\sum_{t=1}^{k} f(x_t, w_t), \qquad k = 1, 2, \ldots,$$

that is similar, but requires much less computation; it needs only one value of f per sample w_t.

- By denoting $\gamma_k = 1/k$, it can also be written as

$$x_{k+1} = (1 - \gamma_k)x_k + \gamma_k f(x_k, w_k), \quad k = 1, 2, \ldots$$

- Compare with Q-learning, where the fixed point problem is $Q = FQ$

$$(FQ)(i, u) = E_j\Big\{g(i, u, j) + \alpha \min_{u'} Q(j, u')\Big\}$$

Q-LEARNING COMBINED WITH OPTIMISTIC PI

- Each Q-learning iteration requires minimization over all controls $u' \in U(j_k)$:

$$Q_{k+1}(i_k,u_k) = (1-\gamma_k)Q_k(i_k,u_k) + \gamma_k\left(g(i_k,u_k,j_k) + \alpha \min_{u'\in U(j_k)} Q_k(j_k,u')\right)$$

- To reduce this overhead we may consider replacing the minimization by a simpler operation using just the "current policy" μ_k

- This suggests an asynchronous sampled version of the optimistic PI algorithm which policy evaluates by

$$Q_{k+1} = F_{\mu^k}^{m_k} Q_k,$$

and policy improves by $\mu^{k+1}(i) \in \arg\min_{u\in U(i)} Q_{k+1}(i,u)$

- This turns out not to work (counterexamples by Williams and Baird, which date to 1993), but a simple modification of the algorithm is valid

- See a series of papers starting with D. Bertsekas and H. Yu, "Q-Learning and Enhanced Policy Iteration in Discounted Dynamic Programming," Math. of OR, Vol. 37, 2012, pp. 66-94

Q-FACTOR APPROXIMATIONS

- We introduce basis function approximation:

$$\tilde{Q}(i,u;r) = \phi(i,u)'r$$

- We can use approximate policy iteration and LSTD/LSPE for policy evaluation

- Optimistic policy iteration methods are frequently used on a heuristic basis

- An extreme example: Generate trajectory $\{(i_k,u_k) \mid k = 0,1,\ldots\}$ as follows.

- At iteration k, given r_k and state/control (i_k,u_k):

  (1) Simulate next transition (i_k,i_{k+1}) using the transition probabilities $p_{i_kj}(u_k)$.

  (2) Generate control u_{k+1} from

$$u_{k+1} = \arg \min_{u\in U(i_{k+1})} \tilde{Q}(i_{k+1},u,r_k)$$

  (3) Update the parameter vector via

$$r_{k+1} = r_k - (\text{LSPE or TD-like correction})$$

- Complex behavior, unclear validity (oscillations, etc). There is solid basis for an important special case: optimal stopping (see text)

BELLMAN EQUATION ERROR APPROACH

- Another model-free approach for approximate evaluation of policy μ: Approximate $Q_\mu(i,u)$ with $\tilde{Q}_\mu(i,u;r_\mu) = \phi(i,u)'r_\mu$, obtained from

$$r_\mu \in \arg\min_r \big\|\Phi r - F_\mu(\Phi r)\big\|_\xi^2$$

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

- Implementation for deterministic problems:

(1) Generate a large set of sample pairs (i_k,u_k), and corresponding deterministic costs $g(i_k,u_k)$ and transitions $\big(j_k,\mu(j_k)\big)$ (a simulator may be used for this).

(2) Solve the linear least squares problem:

$$\min_r \sum_{(i_k,u_k)} \Big|\phi(i_k,u_k)'r - \big(g(i_k,u_k) + \alpha\phi\big(j_k,\mu(j_k)\big)'r\big)\Big|^2$$

- For stochastic problems a similar (more complex) least squares approach works. It is closely related to LSTD (but less attractive; see the text).

- Because this approach is model-free, it is often used as the basis for adaptive control of systems with unknown dynamics.

ADAPTIVE CONTROL BASED ON ADP

LINEAR-QUADRATIC PROBLEM

- System: $x_{k+1} = Ax_k + Bu_k, \quad x_k \in \Re^n, u_k \in \Re^m$
- Cost: $\sum_{k=0}^{\infty}(x_k' Q x_k + u_k' R u_k)$, $Q \geq 0$, $R > 0$
- Optimal policy is linear: $\mu^*(x) = Lx$
- The Q-factor of each linear policy μ is quadratic:

$$Q_\mu(x,u) = (\, x' \quad u' \,)\, K_\mu \begin{pmatrix} x \\ u \end{pmatrix} \qquad (*)$$

- We will consider A and B unknown
- We represent Q-factors using as basis functions all the quadratic functions involving state and control components

$$x^i x^j, \qquad u^i u^j, \qquad x^i u^j, \qquad \forall\ i, j$$

These are the "rows" $\phi(x,u)'$ of Φ

- The Q-factor Q_μ of a linear policy μ can be exactly represented within the approximation subspace:

$$Q_\mu(x,u) = \phi(x,u)' r_\mu$$

where r_μ consists of the components of K_μ in (*)

PI FOR LINEAR-QUADRATIC PROBLEM

- Policy evaluation: r_μ is found by the Bellman error approach

$$\min_r \sum_{(x_k,u_k)} \Big|\phi(x_k,u_k)'r - \big(x_k'Qx_k + u_k'Ru_k + \phi\big(x_{k+1},\mu(x_{k+1})\big)'r\big)\Big|^2$$

where (x_k, u_k, x_{k+1}) are many samples generated by the system or a simulator of the system.

- Policy improvement:

$$\overline{\mu}(x) \in \arg\min_u \big(\phi(x,u)'r_\mu\big)$$

- Knowledge of A and B is not required
- If the policy evaluation is done exactly, this becomes exact PI, and convergence to an optimal policy can be shown
- The basic idea of this example has been generalized and forms the starting point of the field of adaptive dynamic programming
- This field deals with adaptive control of continuous-space, (possibly nonlinear) dynamic systems, in both discrete and continuous time

APPROXIMATION IN POLICY SPACE

APPROXIMATION IN POLICY SPACE

- We parametrize policies by a vector $r = (r_1, \ldots, r_s)$ (an approximation architecture for policies).

- Each policy $\tilde{\mu}(r) = \big\{\tilde{\mu}(i;r) \mid i = 1, \ldots, n\big\}$ defines a cost vector $J_{\tilde{\mu}(r)}$ (a function of r).

- We optimize some measure of $J_{\tilde{\mu}(r)}$ over r.

- For example, use a random search, gradient, or other method to minimize over r

$$\sum_{i=1}^{n} \xi_i J_{\tilde{\mu}(r)}(i),$$

where $\xi_1, \ldots, \xi_n$ are some state-dependent weights.

- An important special case: Introduce cost approximation architecture $V(i;r)$ that defines indirectly the parametrization of the policies

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

- This introduces state features into approximation in policy space.

- A policy approximator is called an actor, while a cost approximator is also called a critic. An actor and a critic may coexist.

APPROXIMATION IN POLICY SPACE METHODS

- Random search methods are straightforward and have scored some impressive successes with challenging problems (e.g., tetris).
  - At a given point/r they generate a random collection of neighboring r. They search within the neighborhood for better points.
  - Many variations (the cross entropy method is one).
  - They are very broadly applicable (to discrete and continuous search spaces).
  - They are idiosyncratic.
- Gradient-type methods (known as policy gradient methods) also have been used extensively.
  - They move along the gradient with respect to r of

$$\sum_{i=1}^{n} \xi_i J_{\tilde{\mu}(r)}(i)$$

  - There are explicit gradient formulas which can be approximated by simulation.
  - Policy gradient methods generally suffer by slow convergence, local minima, and excessive simulation noise.

COMBINATION WITH APPROXIMATE PI

- Another possibility is to try to implement PI within the class of parametrized policies.

- Given a policy/actor $\mu(i;r_k)$, we evaluate it (perhaps approximately) with a critic that produces $\tilde{J}_\mu$, using some policy evaluation method.

- We then consider the policy improvement phase

$$\overline{\mu}(i) \in \arg\min_u \sum_{j=1}^n p_{ij}(u)\big(g(i,u,j) + \alpha \tilde{J}_\mu(j)\big), \quad \forall\, i$$

and do it approximately via parametric optimization

$$\min_r \sum_{i=1}^n \xi_i \sum_{j=1}^n p_{ij}\big(\overline{\mu}(i;r)\big)\Big(g\big(i,\overline{\mu}(i;r),j\big)+\alpha \tilde{J}_\mu(j)\Big)$$

where ξ_i are some weights.

- This can be attempted by a gradient-type method in the space of the parameter vector r.

- Schemes like this have been extensively applied to continuous-space deterministic problems.

- Many unresolved theoretical issues, particularly for stochastic problems.

FINAL WORDS

TOPICS THAT WE HAVE NOT COVERED

- Extensions to discounted semi-Markov, stochastic shortest path problems, average cost problems, sequential games ...
- Extensions to continuous-space problems
- Extensions to continuous-time problems
- Adaptive DP - Continuous-time deterministic optimal control. Approximation of cost function derivatives or cost function differences
- Random search methods for approximate policy evaluation or approximation in policy space
- Basis function adaptation (automatic generation of basis functions, optimal selection of basis functions within a parametric class)
- Simulation-based methods for general linear problems, i.e., solution of linear equations, linear least squares, etc - Monte-Carlo linear algebra

CONCLUDING REMARKS

- There is no clear winner among ADP methods

- There is interesting theory in all types of methods (which, however, does not provide ironclad performance guarantees)

- There are major flaws in all methods:
  - Oscillations and exploration issues in approximate PI with projected equations
  - Restrictions on the approximation architecture in approximate PI with aggregation
  - Flakiness of optimization in policy space approximation

- Yet these methods have impressive successes to show with enormously complex problems, for which there is often no alternative methodology

- There are also other competing ADP methods (rollout is simple, often successful, and generally reliable; approximate LP is worth considering)

- Theoretical understanding is important and nontrivial

- Practice is an art and a challenge to our creativity!

THANK YOU