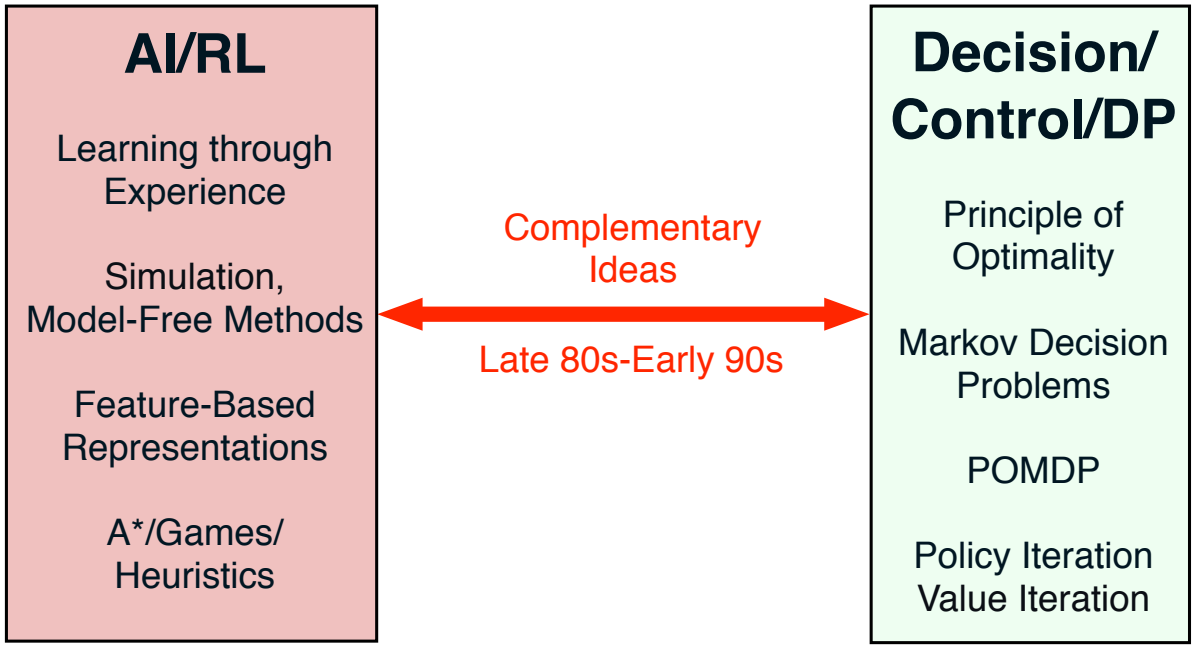
Reinforcement Learning and Optimal Control
A Selective Overview

Dimitri P. Bertsekas

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

March 2019

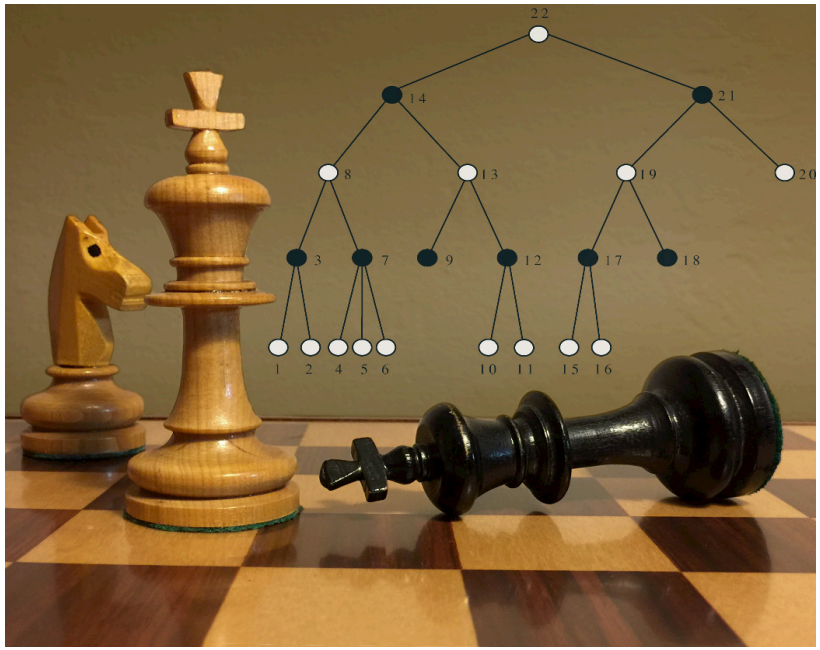
Reinforcement Learning (RL): A Happy Union of AI and Decision/Control Ideas



Historical highlights

- Exact DP, optimal control (Bellman, Shannon, 1950s ...)
- First impressive successes: Backgammon programs (Tesauro, 1992, 1996)
- Algorithmic progress, analysis, applications, first books (mid 90s ...)
- Machine Learning, BIG Data, Robotics, Deep Neural Networks (mid 2000s ...)

AlphaGo (2016) and AlphaZero (2017)



AlphaZero

Plays much better than all chess programs

Plays different!

Learned from scratch ... with 4 hours of training!

Same algorithm learned multiple games (Go, Shogi)

Methodology:

- Simulation-based approximation to a form of the policy iteration method of DP
- Uses self-learning, i.e., self-generated data for policy evaluation, and Monte Carlo tree search for policy improvement

The success of AlphaZero is due to:

- A skillful implementation/integration of known ideas
- Awesome computational power

Approximate DP/RL Methodology is now Ambitious and Universal

Exact DP applies (in principle) to a very broad range of optimization problems

- Deterministic <—-> Stochastic
- Combinatorial optimization <—-> Optimal control w/ infinite state/control spaces
- One decision maker <—-> Two player games
- ... BUT is plagued by the curse of dimensionality and need for a math model

Approximate DP/RL overcomes the difficulties of exact DP by:

- Approximation (use neural nets and other architectures to reduce dimension)
- Simulation (use a computer model in place of a math model)

State of the art:

- Broadly applicable methodology: Can address broad range of challenging problems. Deterministic-stochastic-dynamic, discrete-continuous, games, etc
- There are no methods that are guaranteed to work for all or even most problems
- There are enough methods to try with a reasonable chance of success for most types of optimization problems
- Role of the theory: Guide the art, delineate the sound ideas

Aims and References of this Talk

The purpose of this talk

- To selectively review some of the methods, and bring out some of the AI-DP connections.
- To briefly describe a few recent ideas on aggregation.

References

- Quite a few Exact DP books (1950s-present starting with Bellman). My latest book "Abstract DP" came out a year ago: aims at algorithmic unification through an operator formalism.
- Quite a few DP/Approximate DP/RL/Neural Nets books (1996-Present)
  - Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
  - Sutton and Barto, 1998, Reinforcement Learning (new edition 2018)
  - NEW BOOK: Bertsekas, Reinforcement Learning and Optimal Control, 2019, (to appear). Draft, slides, and videolectures from ASU course at http://web.mit.edu/dimitrib/www/RLbook.html
  - Aims for a coherent synthesis that spans the spectrum of common RL/OC ideas (with some new research here and there)
- Many books and surveys on all aspects of the subject; Tesauro's papers on computer backgammon; Silver, et al., papers on AlphaZero

Terminology in RL/AI and DP/Control

RL uses Max/Value, DP uses Min/Cost

- Reward of a stage = (Opposite of) Cost of a stage.
- State value = (Opposite of) State cost.
- Value (or state-value) function = (Opposite of) Cost function.

Controlled system terminology

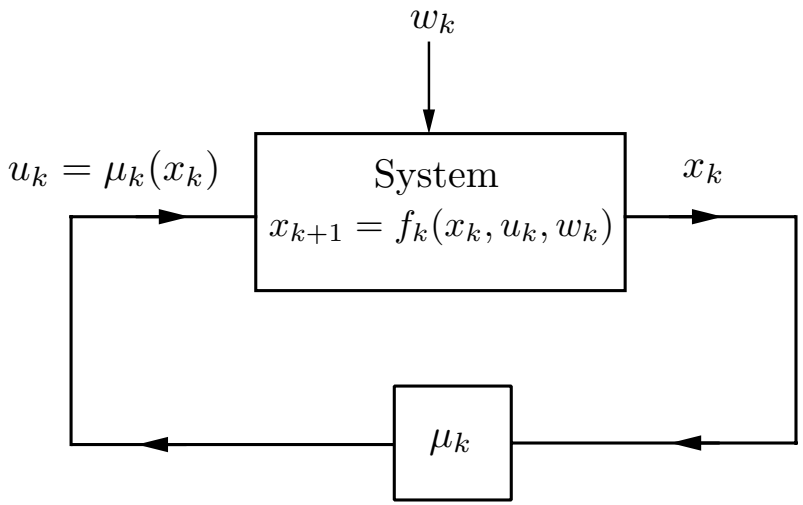
- Agent = Decision maker or controller.
- Action = Control.
- Environment = Dynamic system.

Methods terminology

- Learning = Solving a DP-related problem using simulation.
- Self-learning (or self-play in the context of games) = Solving a DP problem using simulation-based policy iteration.
- Planning vs Learning distinction = Solving a DP problem with model-based vs model-free simulation.

Outline

Finite Horizon Problem - Exact DP



- System

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

  where x_k: State, u_k: Control, w_k: Random disturbance

- Cost function:

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

- Perfect state information: u_k is applied with (exact) knowledge of x_k
- Optimization over feedback policies $\{\mu_0, \ldots, \mu_{N-1}\}$: Rules that specify the control $\mu_k(x_k)$ to apply at each possible state x_k that can occur

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

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

$$J_k(x_k) = \min_{u_k} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\}$$

$J_k(x_k)$: Optimal cost-to-go starting from state x_k

Approximate DP is motivated by the ENORMOUS computational demands of exact DP

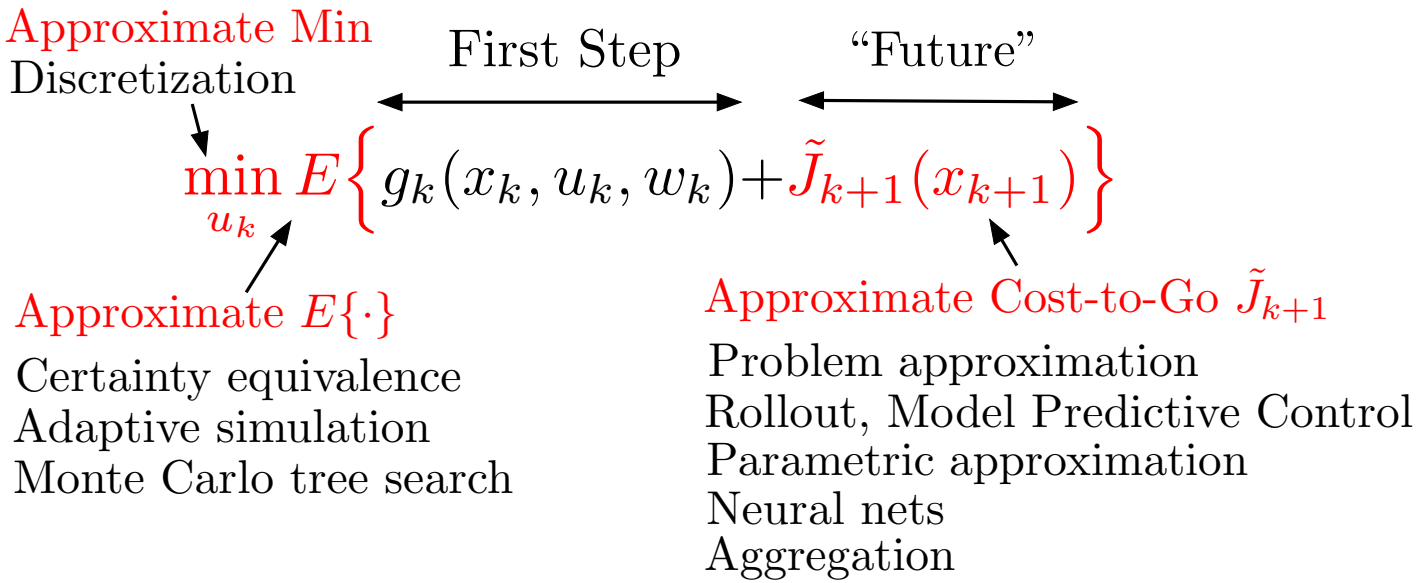
Approximation in value space: Use an approximate cost-to-go function $\tilde{J}_{k+1}$

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big)\Big\}$$

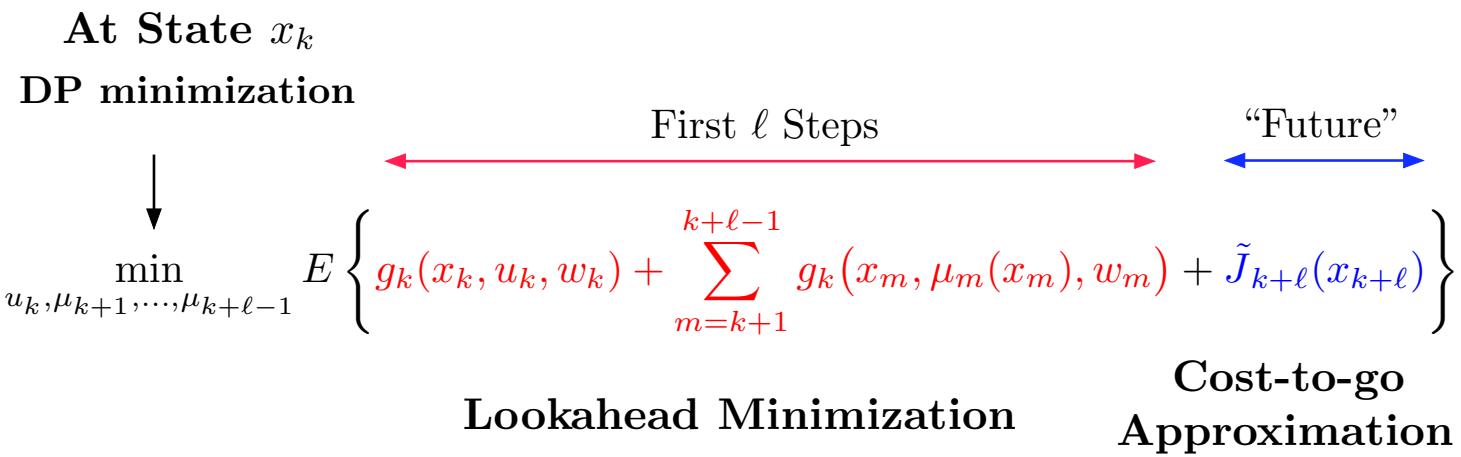
There is also a multistep lookahead version

At state x_k solve an ℓ-step DP problem with terminal cost function approximation $\tilde{J}_{k+\ell}$. Use the first control in the optimal ℓ-step sequence.

Approximation in Value Space

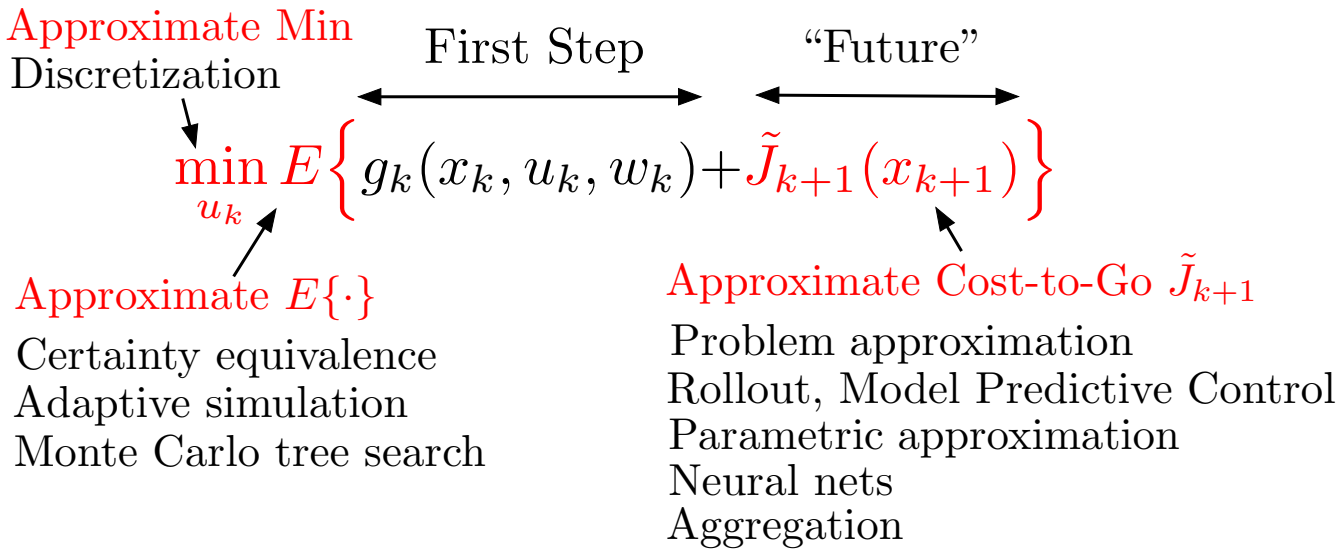


ONE-STEP LOOKAHEAD



MULTISTEP LOOKAHEAD

On-Line and Off-Line Lookahead Implementations



- Off-line methods: All the functions $\tilde{J}_{k+1}$ are computed for every k, before the control process begins.
- Examples of off-line methods: Neural network and other parametric approximations; also aggregation.
- On-line methods: The values $\tilde{J}_{k+1}(x_{k+1})$ are computed only at the relevant next states x_{k+1}, and are used to compute the control to be applied at the N time steps.
- Examples of on-line methods: Rollout and model predictive control.
- On-line methods are well-suited for on-line replanning.
- The minimizing controls $\tilde{\mu}_k(x_k)$ are computed on-line for many-state problems (because of the storage issue, as well as an off-line excessive computation issue).

Model-Based Versus Model-Free Implementation

Our layman's use of the term "model-free": A method is called model-free if it involves calculations of expected values using Monte Carlo simulation.

Model-free implementation is necessary when:

- A mathematical model of the probabilities $p_k(w_k \mid x_k, u_k)$ is not available but a computer model/simulator is. For any (x_k, u_k), it simulates probabilistic transitions to a successor state x_{k+1}, and generates the corresponding transition costs.
- When for reasons of computational efficiency we prefer to compute an expected value by using sampling and Monte Carlo simulation; e.g., approximate an integral or a huge sum of numbers by a Monte Carlo estimate.

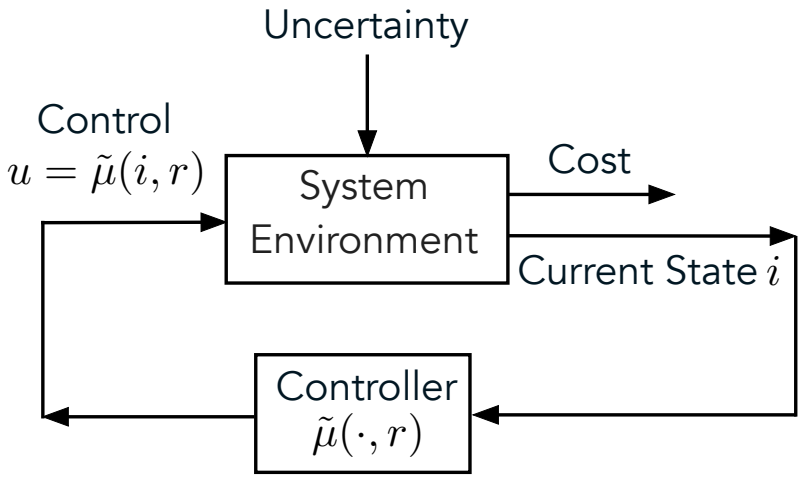
An example of model-free implementation. Assume $\tilde{J}_{k+1}$ has been computed:

Form a training set of (State,Control,Q-factor) triplets: $\big((x_k^s, u_k^s), \beta_k^s\big)$, $s = 1, \ldots, q$, where

$$\beta_k^s = E\Big\{ g_k(x_k^s, u_k^s, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u_k^s, w_k)\big) \Big\}$$

Construct a Q-factor approximation $\tilde{Q}_k(x_k, u_k)$ using a least squares fit - approximation in policy space on top of approximation in value space.

Approximation in Policy Space: A Major Alternative

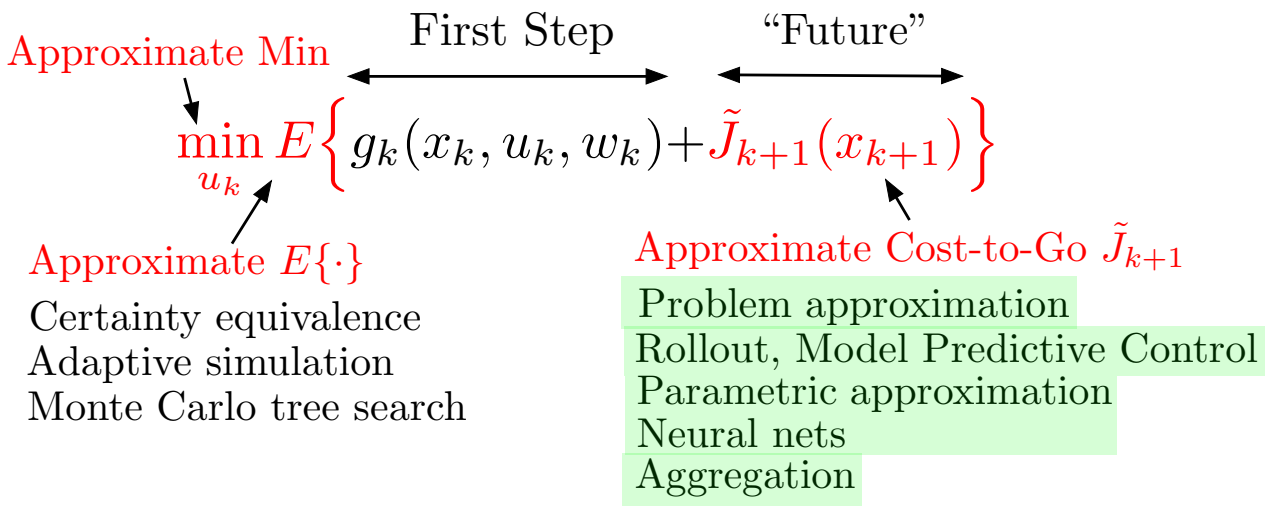


- Parametrize policies with a parameter vectors r. Each r defines a policy.
- The idea is to optimize some measure of performance with respect to the r_k.
- Important advantage: On-line computation of controls is often easier.

Five contexts where approximation in policy space is helpful

- Problems with natural policy parametrizations (like supply chain problems)
- Approximation in policy space on top of approximation in value space (e.g., train in value space, use a neural network to “learn" the lookahead policy).
- Problems with natural value parametrizations, where policy training works well.
- Learning from a software or human expert.
- Unconventional information structures, e.g., multiagent systems with local info.

Problem Approximation: Simplify the Tail Problem and Solve it Exactly

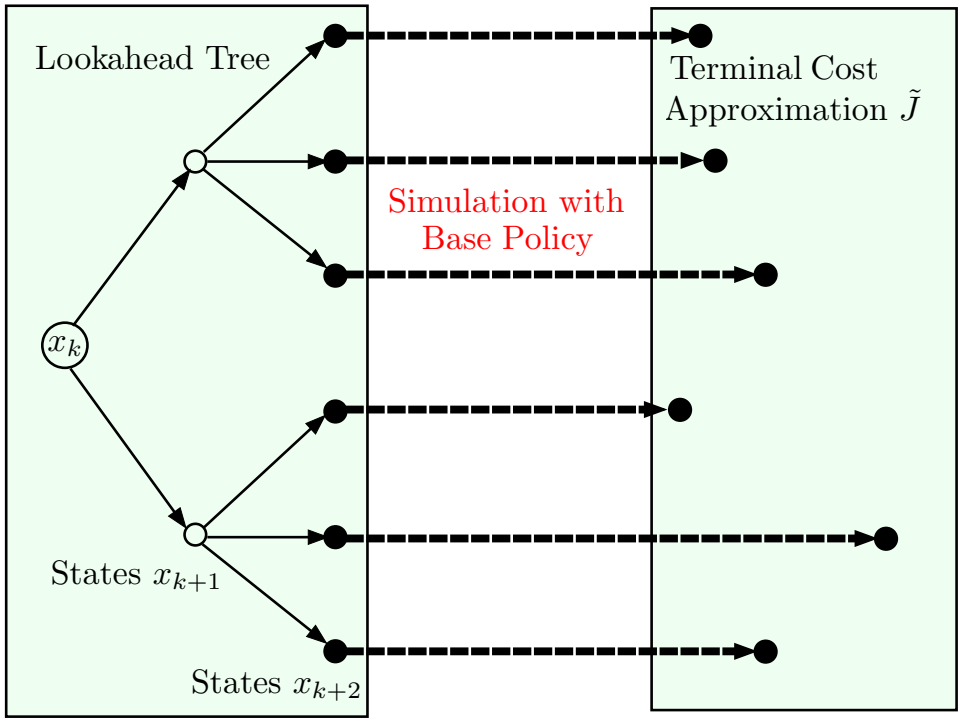


Use as cost-to-go approximation $\tilde{J}_{k+1}$ the exact cost-to-go of a simpler problem

Many problem-dependent possibilities:

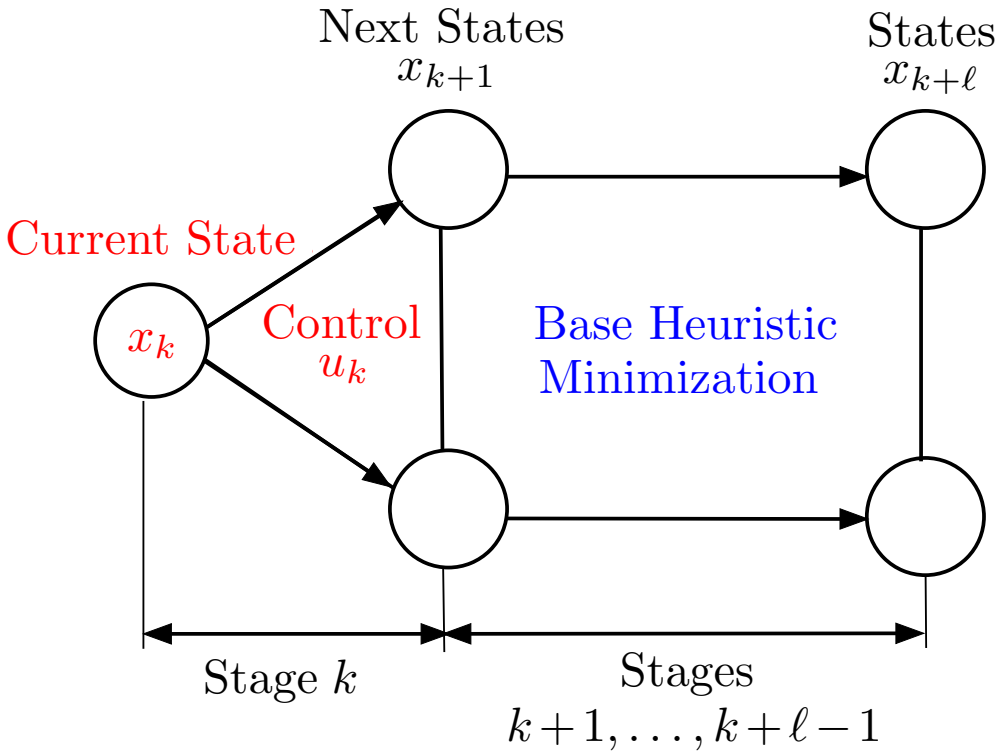
- Probabilistic approximation
  - Certainty equivalence: Replace stochastic quantities by deterministic ones (makes the lookahead minimization deterministic)
  - Approximate expected values by limited simulation
- Enforced decomposition of coupled subsystems
  - One-subsystem-at-a-time optimization
  - Constraint decomposition
  - Lagrangian relaxation
- Aggregation: Group states together and view the groups as aggregate states

Rollout: On-Line Simulation-Based Approximation in Value Space



- Use a base policy: Any suboptimal policy (obtained by another method)
- One-step or multistep lookahead: Exact minimization or a “randomized form of lookahead" that involves “adaptive" simulation and Monte Carlo tree search
- Use (optionally) a terminal cost approximation (obtained by another method)
- Important theoretical fact: With exact lookahead and no terminal cost approximation, the rollout policy improves over the base policy
- Additional error bounds: The rollout policy “improves" approximately on the base policy (depending on assumptions on $\tilde{J}$)

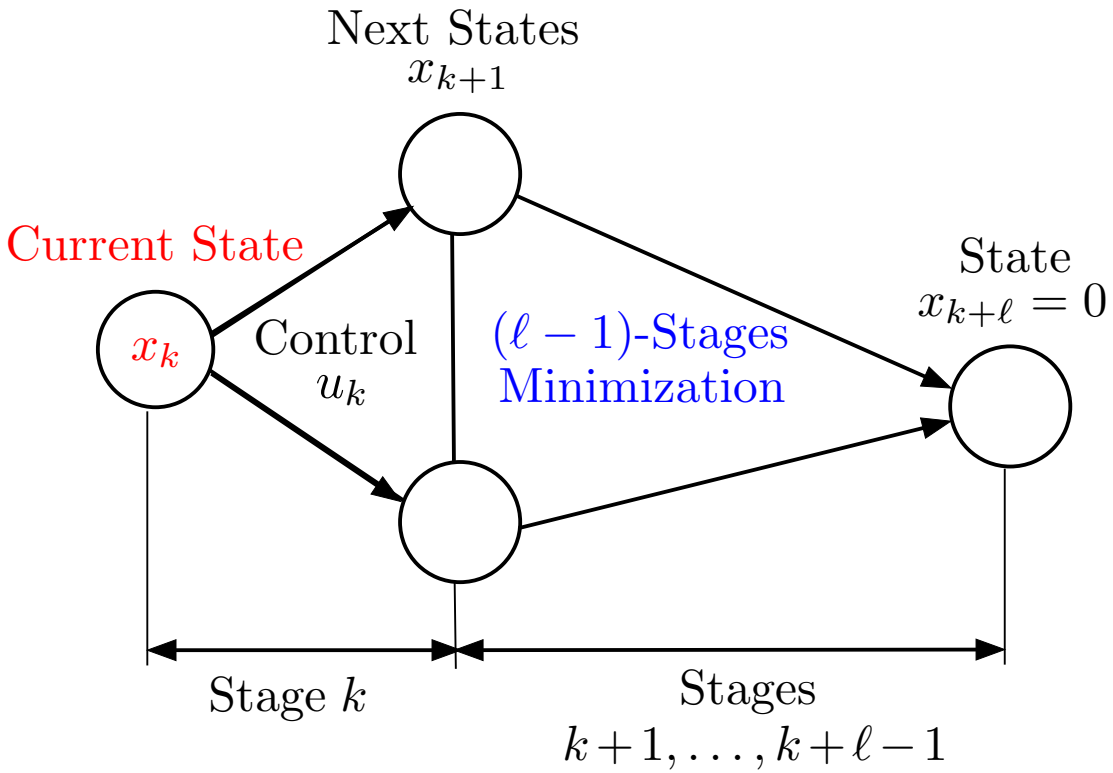
On-Line Rollout for Deterministic Infinite-Spaces Problems



When the control space is infinite rollout needs a different implementation

- One possibility is discretization of $U_k(x_k)$; but then excessive number of Q-factors.
- The major alternative is to use optimization heuristics.
- Seemlessly combine the kth stage minimization and the optimization heuristic into a single ℓ-stage deterministic optimization.
- Can solve it by nonlinear programming/optimal control methods (e.g., quadratic programming, gradient-based).

Model Predictive Control for Deterministic Regulation Problems



- System: $x_{k+1} = f_k(x_k, u_k)$.
- Cost per stage: $g_k(x_k, u_k) \geq 0$, the origin 0 is cost-free and absorbing.
- State and control constraints: $x_k \in X_k$, $u_k \in U_k(x_k)$ for all k.
- At x_k solve an ℓ-step lookahead version of the problem, requiring $x_{k+\ell} = 0$ while satisfying the state and control constraints.
- If $\{\tilde{u}_k, \ldots, \tilde{u}_{k+\ell-1}\}$ is the control sequence so obtained, apply $\tilde{u}_k$.

Parametric Approximation in Value Space

Lookahead Minimization | **Cost-to-go Approximation**

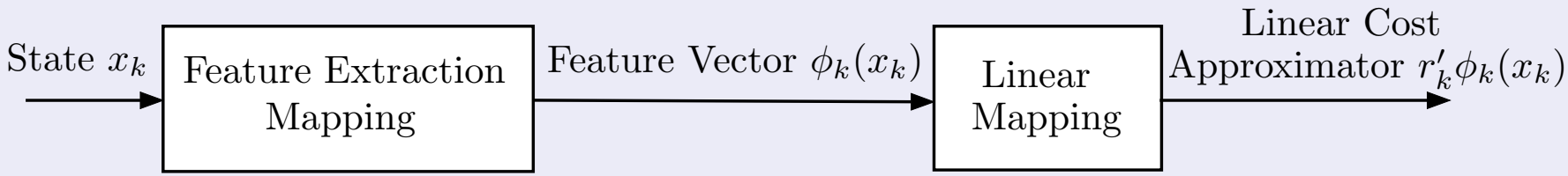
First ℓ Steps | "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}$$

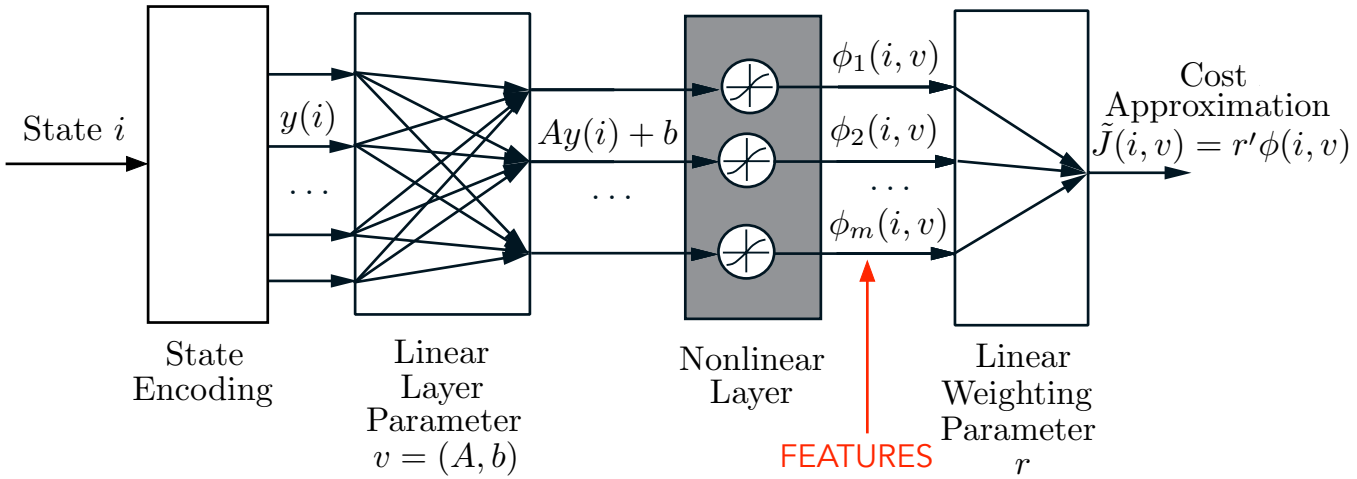
Parametric approximation

- $\tilde{J}_k$ comes from a class of functions $\tilde{J}_k(x_k,r_k)$, where r_k is a tunable parameter
- Proceed as in DP (w/ approximation): Sample (State,Cost) pairs, least squares fit

Feature-based architectures: The linear case



Neural Networks for Constructing Cost-to-Go Approximations $\tilde{J}_k$



Major fact about neural networks

They automatically construct features to be used in a linear architecture

- Neural nets are approximation architectures of the form

$$\tilde{J}(x,v,r) = \sum_{i=1}^{m} r_i \phi_i(x,v) = r'\phi(x,v)$$

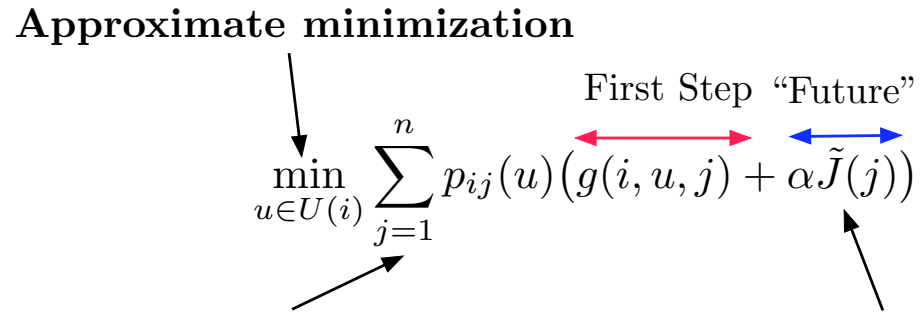
  involving two parameter vectors r and v with different roles

- View $\phi(x,v)$ as a feature vector
- View r as a vector of linear weights for $\phi(x,v)$
- By training v jointly with r, we obtain automatically generated features!

A List of Some Important Topics We Will Not Cover in this Talk

- Infinite horizon extensions: Approximate value and policy iteration methods, error bounds, model-based and model-free methods
- Approximate policy iteration - Self-learning - optimistic and multistep variants: Policy evaluation/Policy improvement
- Temporal difference methods: A class of methods for policy evaluation in infinite horizon problems with a rich theory, issues of variance-bias tradeoff
- Sampling for exploration, in the context of policy evaluation
- Monte Carlo tree search, and related methods
- Q-learning, with and without approximations
- Approximation in policy space, actor-critic methods, policy gradient and cross-entropy methods
- Special aspects of imperfect state information problems, connections with traditional control schemes
- Special aspects of deterministic problems: Shortest paths and their use in approximate DP
- A broad view of using simulation for large-scale computations: Methods for large systems of equations and linear programs, connection to proximal algorithms

Aggregation within the Approximation in Value Space Framework

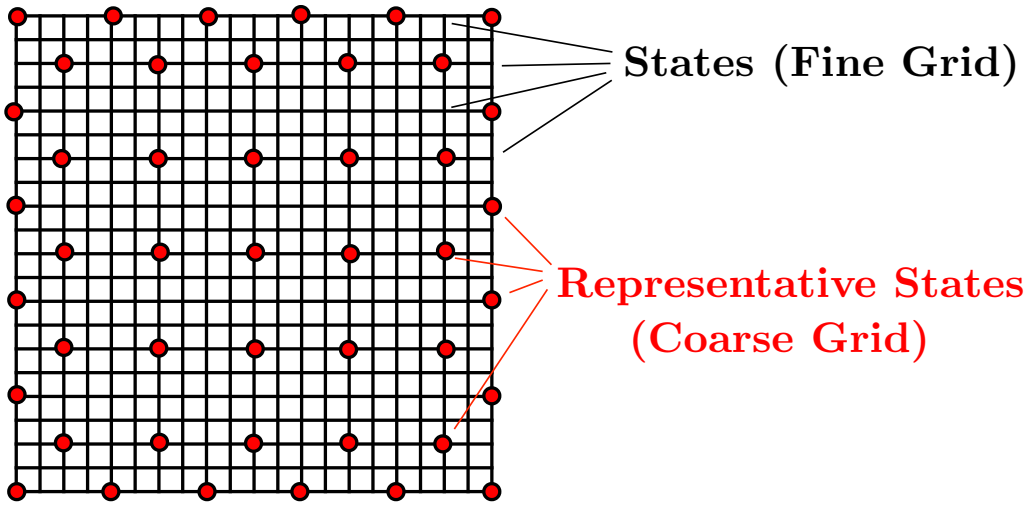


DISCOUNTED INFINITE HORIZON and ONE-STEP LOOKAHEAD
MULTISTEP LOOKAHEAD IS SIMILAR - WE WILL DISCUSS LATER

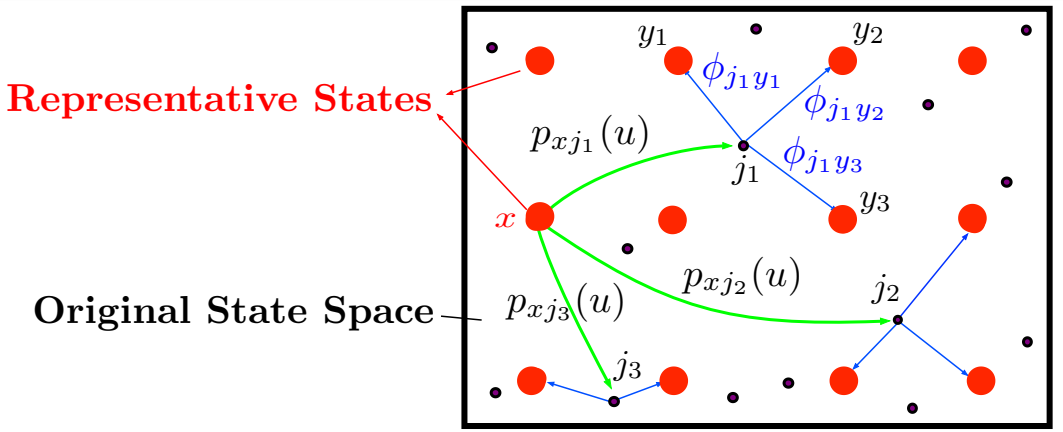
Some important differences from alternative schemes:

- In aggregation, $\tilde{J}$ aims to approximate J^*, not the cost function J_μ of a policy μ, like rollout or approximate PI.
- $\tilde{J}$ converges to J^* as the aggregation becomes "finer".
- Key factor for good performance: Choose properly the aggregation structure so that the "aggregate problem" size needed for good performance is not excessive.

Aggregation with Representative States: A Classical Form of Discretization

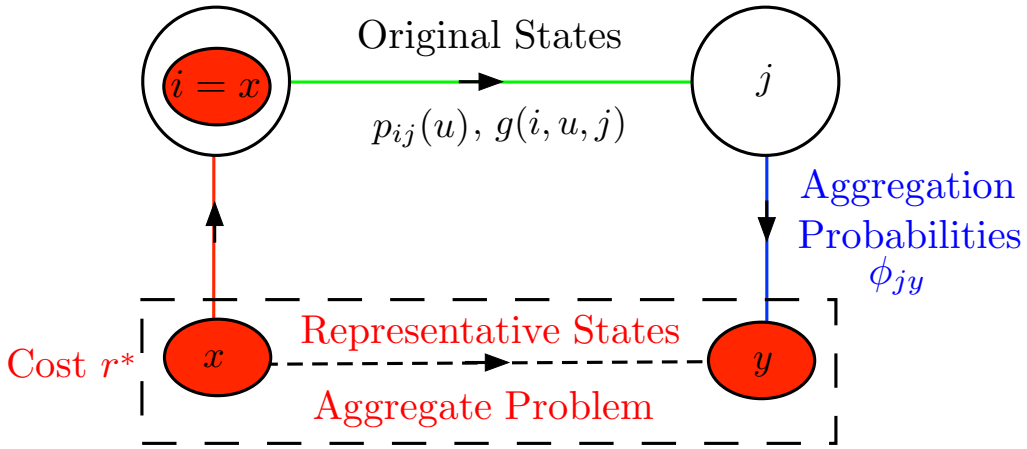


- Relate the original states with the representative states with interpolation coefficients called aggregation probabilities.
- Solve the coarse grid problem, and interpolate.



Aggregation Probabilities
ϕ_{jy}
Relate
Original States to
Representative States

Representative States - The Aggregate Problem



Original cost approximation by interpolation

$$\hat{p}_{xy}(u) = \sum_{j=1}^{n} p_{xj}(u)\phi_{jy}, \quad \hat{g}(x,u) = \sum_{j=1}^{n} p_{xj}(u)g(x,u,j), \quad \tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*$$

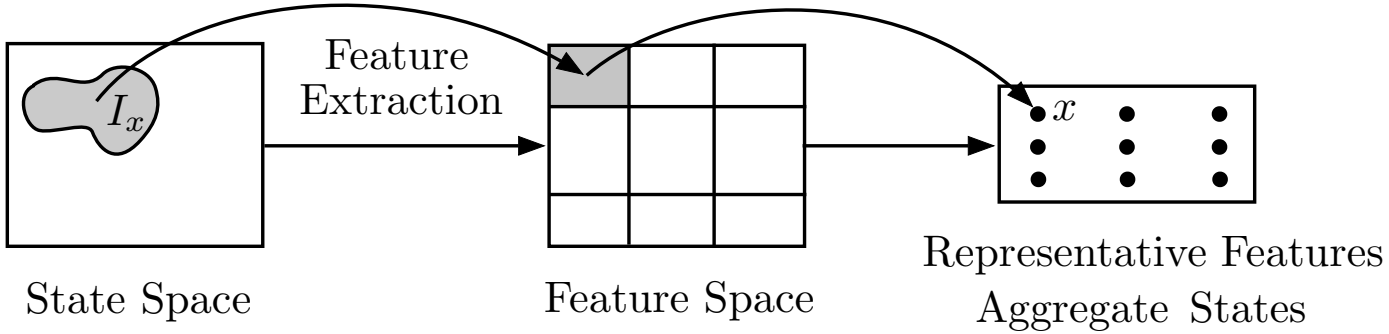
Exact methods

Once the aggregate model is computed (i.e., its transition probs. and cost per stage), any exact DP method can be used: VI, PI, optimistic PI, or linear programming.

Model-free (simulation-based) methods

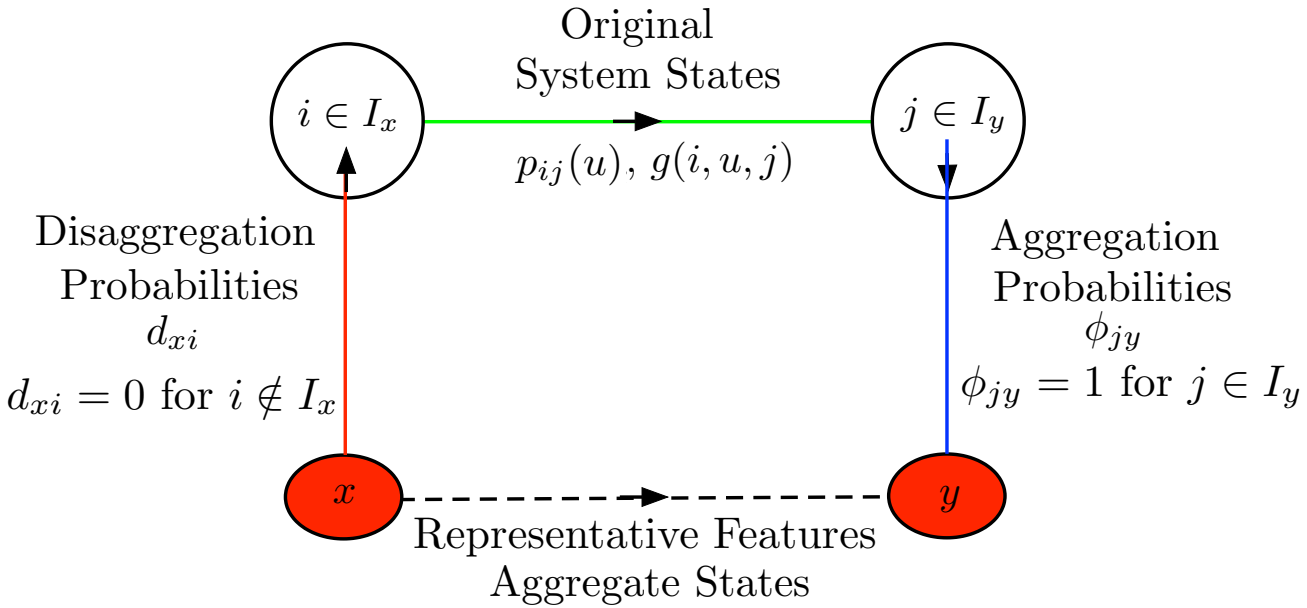
Given a simulator for the original problem, we can obtain a simulator for the aggregate problem. Then use an (exact) model-free method to solve the aggregate problem.

Feature-Based Aggregation - Discretize the Feature Space

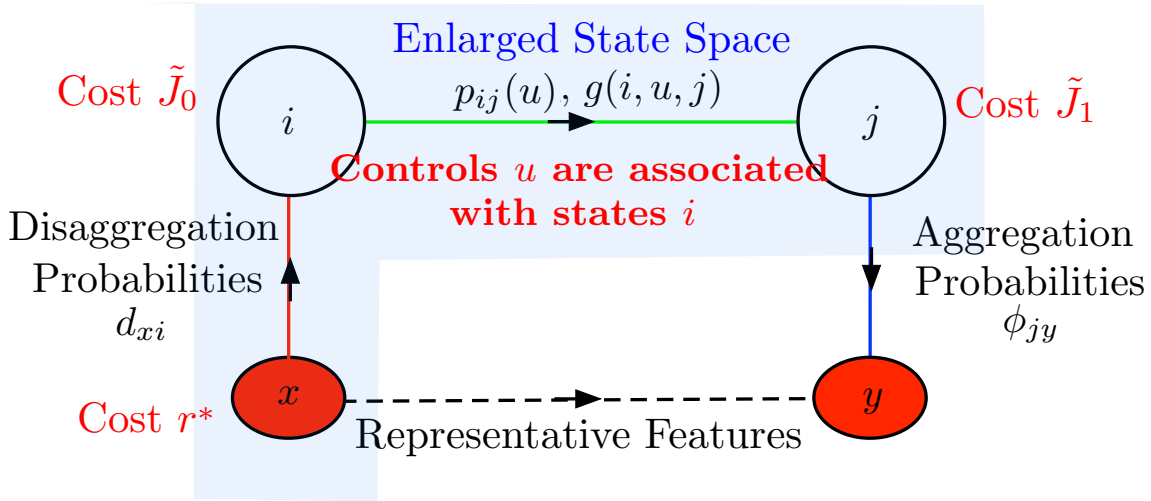


Guiding ideas for representative features formation:

- Feature map F: States i with similar $F(i)$ should have similar $J^*(i)$.
- Footprint I_x of feature x: States i in I_x should have feature $F(i) \approx x$.



The Aggregate Problem: Control Applied on a State-by-State Basis



Bellman equations for the enlarged problem

$$r_x^* = \sum_{i=1}^{n} d_{xi} \tilde{J}_0(i), \qquad x \in \mathcal{A},$$

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

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

r^* solves uniquely the composite Bellman equation $r^* = Hr^*$:

$$r_x^* = (Hr^*)(x) = \sum_{i=1}^{n} d_{xi} \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \sum_{y \in \mathcal{A}} \phi_{jy}\, r_y^* \right)$$

Error Bound [Tsitsiklis and Van Roy (1995)]

Consider the footprint sets $S_y = \{j \mid \phi_{jy} = 1\}$. The $(J^* - \tilde{J})$ error is small if J^* varies little within each S_y. In particular,

$$\left|J^*(j) - r_y^*\right| \leq \frac{\epsilon}{1-\alpha}, \qquad j \in S_y,$$

where $\epsilon = \max_{y \in \mathcal{A}} \max_{i,j \in S_y} \left|J^*(i) - J^*(j)\right|$ is the max variation of J^* within S_y.

Implication

Choose representative features x so that J^* varies little over the footprint of x.

Value and policy iteration [Tsitsiklis and Van Roy (1995)]

Simulation-based versions of value and policy iteration and convergence analysis.

Variations

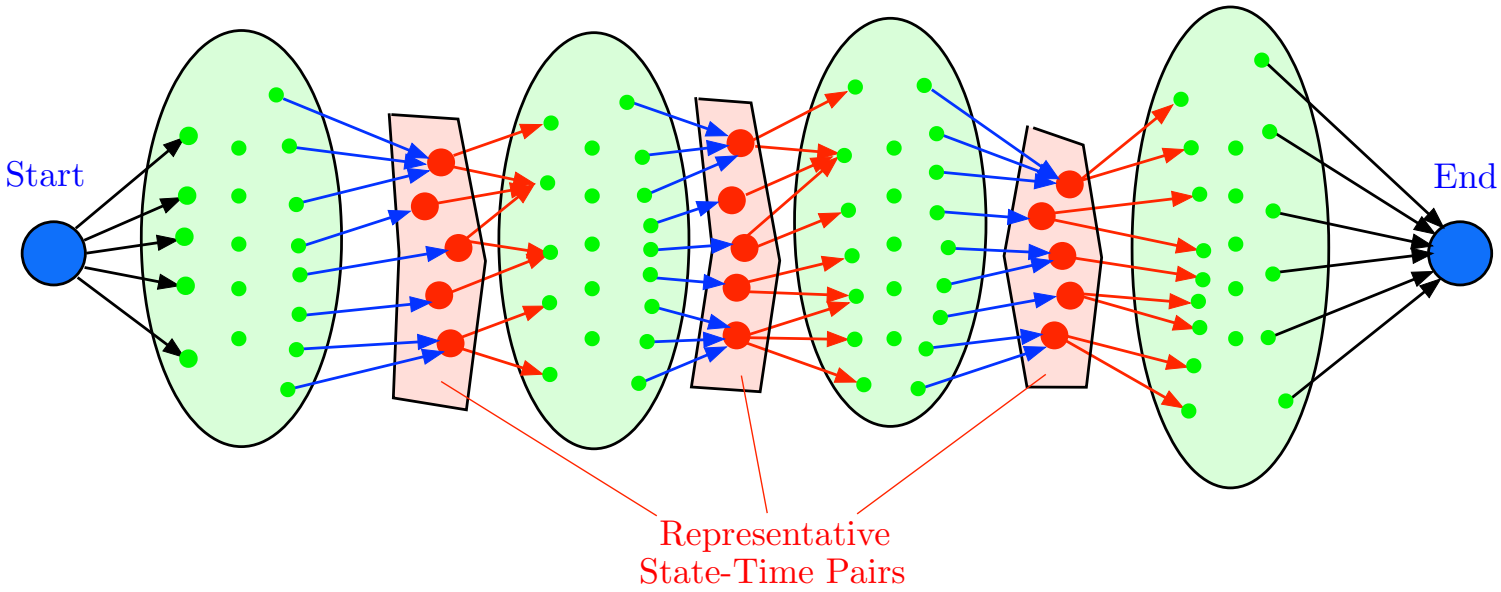
- Use of (deep) neural nets to construct the representative features and their footprints.
- Biased aggregation: Make local corrections to an existing $\tilde{J} \approx J^*$.

Spatio-Temporal Aggregation - Compressing Space and Time

Plan 5-day auto travel from Boston to San Francisco - How would you do it?

- Select major stops/cities (New York, Chicago, Salt Lake City, Phoenix, etc).
- Select major stopping times (times to stop for sleep, rest, etc).
- Decide on space and time schedules at a coarse level. Optimize the details later.
- We may view this as an example of reduction of a very large-scale shortest path problem to a manageable problem by spacio-temporal aggregation.

Spatio-Temporal Decomposition



Formalization - Deterministic shortest path problem

- Consider the space-time tube of a deterministic shortest path problem.
- Introduce space-time barriers, i.e., subsets of representative state-time pairs that "separate past from future" (think of the Boston-San Francisco travel).
- "Compress" the portions of the space-time tube between two successive barriers into shortest path problems between each state-time pair of the left barrier to each state-time pair of the right barrier.
- Form a "master" shortest path problem of low dimension that involves only the representative state-time pairs. Use it to approximate the solution of the original.
- Stochastic extensions; partial certainty equivalence.

Concluding Remarks

Some words of caution

- There are challenging implementation issues in all approaches, and no fool-proof methods
- Problem approximation and feature selection require domain-specific knowledge
- Training algorithms are not as reliable as you might think by reading the literature
- Approximate policy iteration involves exploration and oscillation challenges
- Recognizing success or failure can be difficult!
- The RL successes in game contexts are spectacular, but they have benefited from perfectly known and stable models and small number of controls (per state)
- Problems with partial state observation remain a big challenge

On machine learning (Steven Strogatz, NY Times Article, Dec. 2018)

"What is frustrating about machine learning is that the algorithms can't articulate what they're thinking. We don't know why they work, so we don't know if they can be trusted ... As human beings, we want more than answers. We want insight. This is going to be a source of tension in our interactions with computers from now on."

On the Positive Side

- Massive computational power together with distributed computation are a source of hope
- Silver lining: We can begin to address practical optimization problems of unimaginable difficulty!
- There is an exciting journey ahead!

Thank you!