1cm240: AI for logistics and its interfaces
Lecture 2

14/09/2021

Dr. Albert Schrotenboer, Assistant Professor

School of Engineering, Operations, Planning, Accounting & Control Group

TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Recap of last week

- Five fundamental pillars of SSDPs
  - State variable
  - Decision variable
  - Exogonous information
  - Transition Function
  - Objective Function
- A state variable describes ALL information necessary to make decisions
- The decision describes all information necessary to make decision
- The exogenous information is the uncertainty you play against
- The transition function shows how decision, state and exogenous information result in a new state

Today's lecture

- How to anticipate upon the future: four classes of policies to do so

BREAK

- Two solution procedures suitable for the case-study
- Some hints and tips on the provided code
- State space aggregation and direct rewards in the case-study

Expected progress on the case study this week

- At the end of this week you must be able to:
  - Make predictions of future bike-sharing demand in real-time
  - Store a direct reward based on state variable representation
  - Save the performance based upon the state variable representation

  - Understand what steps are needed to make the trainer ‘model-free’
  - Understand the steps required to do a simple q-learning on very aggregated state-variable representations

  - Being able to improve the link between environment and the actual data (i.e., the quality of our environment/model w.r.t. actual case)

What is needed for exact solutions?

- A decision x_t in a particular state S_t is the result of some (good / bad / optimal) policy $\pi \in \Pi$. Its ``goodness'' has two parts:
- Direct reward: $C(S_t, x_t, W_{t+1})$
- Future rewards..
- $S^M(S_t, X^\pi(S_t), W_{t+1}) := S_{t+1}$, with $C(S_{t+1}, X^\pi(S_{t+1}), W_{t+2})$
  - For each possible W_{t+1}, and each possible $\pi \in \Pi$, and for all states S_t

The three curses of dimensionality

For each possible W_{t+1}, and each possible $\pi \in \Pi$, and for all states S_t …

- An explosion in the number of states (dimension 1)
- An explosion in the number of decision (dimension 2)
- An explosion in the number of transition (dimension 3)

Exact solutions means to find the optimal policy:

Bellman's optimality equation:

$$V(S_t) = \min_{x_t \in X} \left(C(S_t, x_t) + \gamma\, E(V(S_{t+1}(S_t, x_t, W_{t+1}))|S_t])\right)$$

- This can be solved using backward dynamic programming. For our case this is impossible, but this is possible for some applications using backward dynamic programming

Step 0: Initialize $V_{T+1}(S_{T+1}) = 0$ for all states.

Step 1: Step backward $t = T, T-1, T-2, \ldots$

Step 2: Loop over $S_t = (R_t, D_t, p_t, E_t)$ (four loops)

Step 3: Loop over all decisions x_t (all dimensions)

Step 4: Take the expectation over each random dimension $(\hat{D}_t, \hat{p}_t, \hat{E}_t)$

Compute $Q(S_t, x_t) = C(S_t, x_t) +$

$$\sum_{w_1=0}^{100} \sum_{w_2=0}^{100} \sum_{w_3=0}^{100} V_{t+1}\left(S^M\left(S_t, x_t, W_{t+1} = (w_1, w_2, w_3)\right)\right) P^W(w_1, w_2, w_3)$$

End step 4;

End Step 3;

Find $V_t^*(S_t) = \max_{x_t} Q(S_t, x_t)$

Store $X_t^{\pi^*}(S_t) = \arg\max_{x_t} Q(S_t, x_t)$. (This is our policy)

End Step 2;

End Step 1;

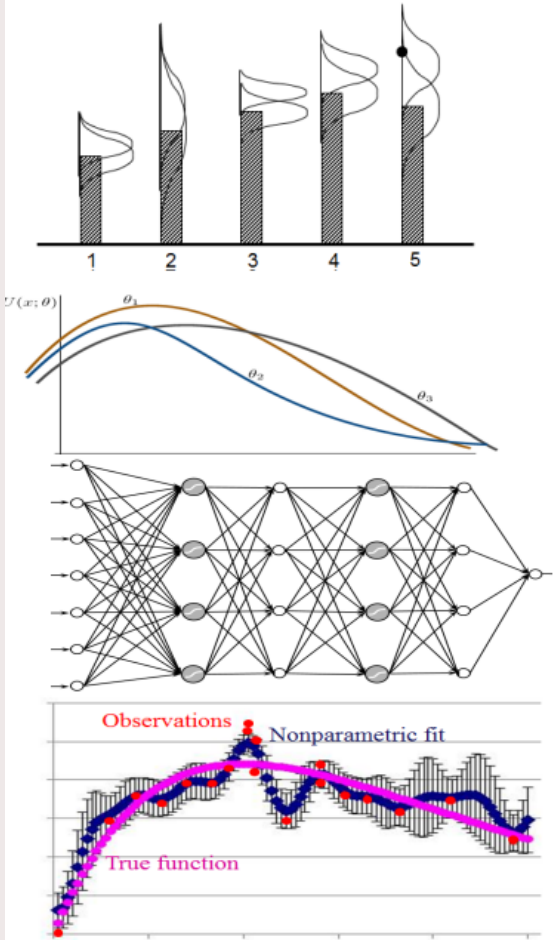
What can we do.. We can LEARN

We consider 5 different classes of learning problems in stochastic optimization:

- Designing a policy $X^{\pi}(S_t)$
- A value function approximation $\bar{V}_t(S_t \mid \theta) \approx V_t(S_t)$
- Designing a cost function approximation
  - The objective function $\bar{C}(S_t, x_t, W_{t+1} \mid \theta)$
  - The constraints of $X^{\pi}(S_t \mid \theta)$
- Approximating the transition function
  - $\bar{S}^M(S_t, x_t, W_{t+1} \mid \theta) = S^M(S_t, x_t, W_{t+1})$
- Approximating the objective function (1-step learning)
  - $\bar{F}(x_t \mid \theta_{(t)}) \approx E[C(S_t, x_t, W_{t+1})]$

Approximation strategies

- Lookup tables
  - Independent beliefs, correlated beliefs/future info
- Linear parametric models
  - OLS/linear models
  - Sparse-linear
  - Tree regression
- Nonparametric models
  - Gaussian process regression, kernel regression, support vector machines, deep neural nets
- Nonlinear parameteric models
  - Logistic regression
  - Neural nets



Chapter 2:

Four classes of policies for SSDPs

To attack the three curses of dimensionality
- States, Decisions, Transitions

Designing policies

We have to start by describing what we mean by a policy.

Definition:

A policy is a mapping from a state to an action.

… any mapping.

Designing policies

“Policies” and the English language

Behavior	Habit	Procedure
Belief	Laws/bylaws	Process
Bias	Manner	Protocols
Commandment	Method	Recipe
Conduct	Mode	Ritual
Convention	Mores	Rule
Culture	Patterns	Style
Customs	Plans	Technique
Dogma	Policies	Tenet
Etiquette	Practice	Tradition
Fashion	Prejudice	Way of life
Formula	Principle	

Designing policies

Two fundamental strategies:

1) Policy search – Search over a class of functions for making decisions to optimize some metric.

$$\max_{\pi=(f\in F,\theta^f\in\Theta^f)} E\left\{\sum_{t=0}^{T} C\left(S_t, X_t^{\pi}(S_t\,|\,\theta)\right)|\, S_0\right\}$$

2) Lookahead approximations – Approximate the impact of a decision now on the future.

$$X_t^*(S_t) = \arg\max_{x_t}\left(C(S_t,x_t) + \mathsf{E}\left\{\max_{\pi\in\Pi}\left\{\mathsf{E}\sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'}))\,|\,S_{t+1}\right\}|\,S_t, x_t\right\}\right)$$

Policy search:

1a. Policy function approximations (PFAs) $x_t = X^{PFA}(S_t \mid \theta)$

- Lookup tables
  - "when in this state, take this action"
- Parametric functions
  - Order-up-to policies: if inventory is less than s, order up to S.
  - Affine policies - $x_t = X^{PFA}(S_t \mid \theta) = \sum_{f \in F} \theta_f \phi_f(S_t)$
  - Neural networks
- Locally/semi/non parametric
  - Requires optimizing over local regions

1b) Cost function approximations (CFAs)

- Optimizing a deterministic model modified to handle uncertainty (buffer stocks, schedule slack)

$$X^{CFA}(S_t \mid \theta) = \arg\max_{x_t \in \bar{X}_t^{\pi}(\theta)} \bar{C}^{\pi}(S_t, x_t \mid \theta)$$

Lookahead approximations

Approximate the impact of a decision now on the future:

- An optimal policy (based on looking ahead):

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathsf{E}\left\{ \max_{\pi \in \Pi} \left\{ \mathsf{E} \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \mid S_{t+1} \right\} \mid S_t, x_t \right\} \right)$$

- Approximating the value of being in a downstream state using machine learning ("value function approximations")

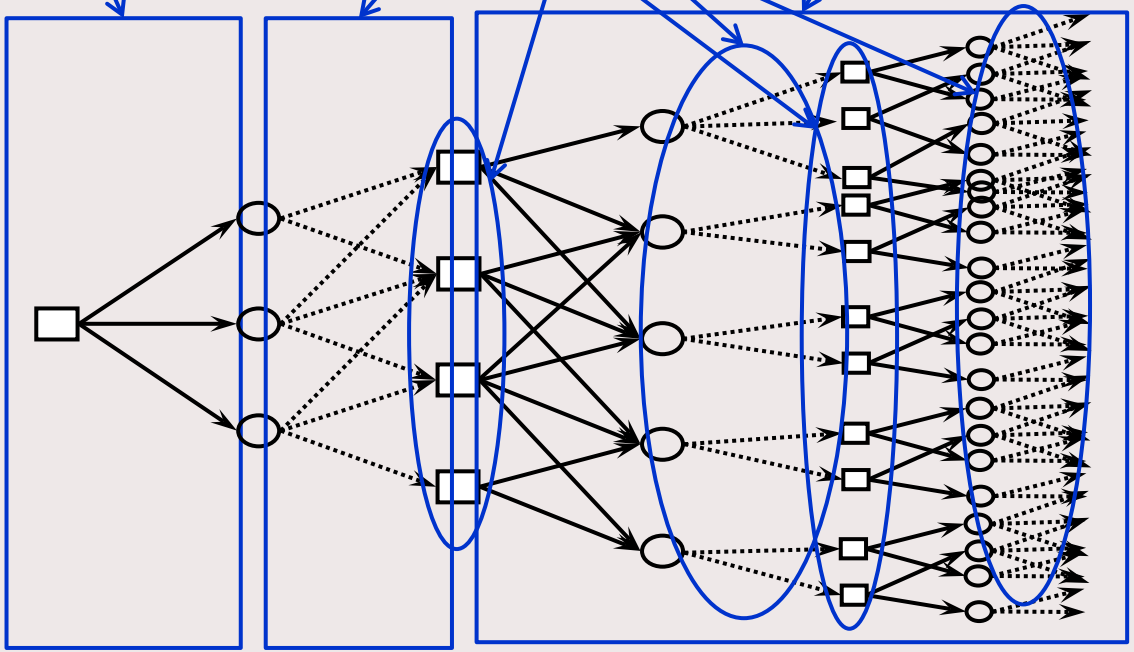
$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathsf{E}\left\{ V_{t+1}(S_{t+1}) \mid S_t, x_t \right\} \right)$$

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathsf{E}\left\{ \bar{V}_{t+1}(S_{t+1}) \mid S_t, x_t \right\} \right)$$

$$= \arg\max_{x_t} \left(C(S_t, x_t) + \bar{V}_t^x(S_t^x) \right)$$

The ultimate lookahead policy is optimal

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathsf{E}\left\{ \max_{\pi \in \Pi} \left\{ \mathsf{E} \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \mid S_{t+1} \right\} \mid S_t, x_t \right\} \right)$$



The ultimate lookahead policy is optimal

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \mathsf{E}\left\{ \max_{\pi \in \Pi} \left\{ \mathsf{E} \sum_{t'=t+1}^{T} C(S_{t'}, X_{t'}^{\pi}(S_{t'})) \mid S_{t+1} \right\} \mid S_t, x_t \right\} \right)$$

2b) Instead, we have to solve an approximation called the *lookahead model:*

$$X_t^*(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{\mathsf{E}}\left\{ \max_{\tilde{\pi} \in \tilde{\Pi}} \left\{ \tilde{\mathsf{E}} \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{X}_{tt'}^{\pi}(\tilde{S}_{tt'})) \mid \tilde{S}_{t,t+1} \right\} \mid \tilde{S}_{tt}, x_t \right\} \right)$$

A lookahead policy works by approximating the *lookahead model*.

Types of lookahead approximations

One-step lookahead – Widely used in pure learning policies:

- Point estimate (easiest)

$$X^{Greedy}(S^n) = \arg\max_x F(x, \mathsf{E}W)$$

- Bayes greedy/naïve Bayes (harder)

$$X^{Bayes}(S^n) = \arg\max_x \mathsf{E}F(x, W)$$

- Thompson sampling

$$X^{TS}(S^n) = \arg\max_x \hat{\mu}_x \quad \text{where } \hat{\mu}_x : \; N(\bar{\mu}_x^n, b_x^n)$$

- Value of information (knowledge gradient)

$$X^{KG}(S^n) = \arg\max_x \hat{\mu}_x \quad \text{where } \hat{\mu}_x : \; N(\bar{\mu}_x^n, b_x^n)$$

Creating a lookahead model

We still need a policy $\tilde{X}_t^{\pi}(\tilde{S}_{tt'})$ in our lookahead model. Assume we use a linear decision rule:

$$\tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\theta_t) = \theta_{t0} + \theta_{t1}\phi_1(\tilde{S}_{tt'}) + \theta_{t2}\phi_2(\tilde{S}_{tt'}),$$

Our lookahead policy might then be

$$X_t^{LA-Stoch}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E}\left\{ \max_{\tilde{\theta}_t} \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\tilde{\theta}_t))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \right)$$

But this means that we have to optimize $\bar{\theta}_t(S_t)$ for each time period and given the state S_t that we are in!

Creating a lookahead model

A more practical idea is to replace

$$X_t^{LA-Stoch}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E}\left\{ \max_{\tilde{\theta}_t} \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\tilde{\theta}_t))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \right)$$

with

$$X_t^{LA-Stoch}(S_t|\theta) = \arg\max_{x_t} \left(C(S_t, x_t) + \tilde{E}\left\{ \tilde{E}\left\{ \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}, \tilde{X}_t^{Lin}(\tilde{S}_{tt'}|\theta))|\tilde{S}_{t,t+1} \right\} |S_t, x_t \right\} \right).$$

Now we have to tune θ as we would any tunable parameter.

Types of lookahead approximations

Multi-step lookahead

- Deterministic lookahead, also known as model predictive control, rolling horizon procedure

- Stochastic lookahead:
  - Two-stage (widely used in stochastic linear programming)
  - Multistage
    - Monte carlo tree search (MCTS) for discrete action spaces
    - Multistage scenario trees (stochastic linear programming) – typically not tractable.

Four (meta)classes of policies

Policy search

1) **Policy function approximations (PFAs)**

Lookup tables, rules, parametric/nonparametric functions

2) **Cost function approximation (CFAs)**

$$X^{CFA}(S_t \mid \theta) = \arg\max_{x_t \in \bar{X}_t^{\pi}(\theta)} \bar{C}^{\pi}(S_t, x_t \mid \theta)$$

Lookahead approximations

3) **Policies based on value function approximations (VFAs)**

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \bar{V}_t^{x} \left(S_t^{x}(S_t, x_t) \right) \right)$$

4) **Direct lookahead policies (DLAs)**

Deterministic lookahead/rolling horizon proc./model predictive control

$$X_t^{LA-D}(S_t) = \arg\max_{\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'})$$

Chance constrained programming

$$P[A_t x_t \le f(W)] \le 1 - \delta$$

Stochastic lookahead /stochastic prog/Monte Carlo tree search

$$X_t^{LA-S}(S_t) = \arg\max_{\tilde{x}_{tt}, \tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+T}} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{\tilde{\omega} \in \tilde{\Omega}_t} p(\tilde{\omega}) \sum_{t'=t+1}^{t+T} C(\tilde{S}_{tt'}(\tilde{\omega}), \tilde{x}_{tt'}(\tilde{\omega}))$$

"Robust optimization"

$$X_t^{LA-RO}(S_t) = \arg\max_{\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}} \min_{w \in W_t(\theta)} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}(w), \tilde{x}_{tt'}(w))$$

Four (meta)classes of policies

Function approx.

1) **Policy function approximations (PFAs)**

Lookup tables, rules, parametric/nonparametric functions

2) **Cost function approximation (CFAs)**

$$X^{CFA}(S_t \mid \theta) = \arg\max_{x_t \in \bar{X}_t^{\pi}(\theta)} \bar{C}^{\pi}(S_t, x_t \mid \theta)$$

3) **Policies based on value function approximations (VFAs)**

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \bar{V}_t^{x} \left(S_t^{x}(S_t, x_t) \right) \right)$$

4) **Direct lookahead policies (DLAs)**

Deterministic lookahead/rolling horizon proc./model predictive control

$$X_t^{LA-D}(S_t) = \arg\max_{\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{t} C(\tilde{S}_{tt'}, \tilde{x}_{tt'})$$

Chance constrained programming

$$P[A_t x_t \le f(W)] \le 1 - \delta$$

Stochastic lookahead /stochastic prog/Monte Carlo tree search

$$X_t^{LA-S}(S_t) = \arg\max_{\tilde{x}_{tt}, \tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+T}} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{\tilde{\omega} \in \tilde{\Omega}_t} p(\tilde{\omega}) \sum_{t'=t+1}^{t} C(\tilde{S}_{tt'}(\tilde{\omega}), \tilde{x}_{tt'}(\tilde{\omega}))$$

"Robust optimization"

$$X_t^{LA-RO}(S_t) = \arg\max_{\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}} \min_{w \in W_t(\theta)} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}(w), \tilde{x}_{tt'}(w))$$

TU/e

Four (meta)classes of policies

1) **Policy function approximations (PFAs)**

Lookup tables, rules, parametric/nonparametric functions

2) **Cost function approximation (CFAs)**

$$X^{CFA}(S_t \mid \theta) = \arg\max_{x_t \in \bar{X}_t^{\pi}(\theta)} \bar{C}^{\pi}(S_t, x_t \mid \theta)$$

3) **Policies based on value function approximations (VFAs)**

$$X_t^{VFA}(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + \bar{V}_t^{x} \left(S_t^{x}(S_t, x_t) \right) \right)$$

4) **Direct lookahead policies (DLAs)**

Deterministic lookahead/rolling horizon proc./model predictive control

$$X_t^{LA-D}(S_t) = \arg\max_{\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{t} C(\tilde{S}_{tt'}, \tilde{x}_{tt'})$$

Chance constrained programming

$$P[A_t x_t \le f(W)] \le 1 - \delta$$

Stochastic lookahead /stochastic prog/Monte Carlo tree search

$$X_t^{LA-S}(S_t) = \arg\max_{\tilde{x}_{tt}, \tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+T}} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{\tilde{\omega} \in \tilde{\Omega}_t} p(\tilde{\omega}) \sum_{t'=t+1}^{t} C(\tilde{S}_{tt'}(\tilde{\omega}), \tilde{x}_{tt'}(\tilde{\omega}))$$

"Robust optimization"

$$X_t^{LA-RO}(S_t) = \arg\max_{\tilde{x}_{tt}, \ldots, \tilde{x}_{t,t+H}} \min_{w \in W_t(\theta)} C(\tilde{S}_{tt}, \tilde{x}_{tt}) + \sum_{t'=t+1}^{T} C(\tilde{S}_{tt'}(w), \tilde{x}_{tt'}(w))$$

Imbedded optimization

Finding the best policy

We have to first articulate our classes of policies

$$f \in F = \{PFAs, CFAs, VFAs, DLAs\}$$

$$\theta \in \Theta^f = \text{Parameters that characterize each family.}$$

So minimizing over $\pi \in \Pi$ means:

$$\Pi = \left\{ f \in F, \theta \in \Theta^f \right\}$$

We then have to pick an objective such as

$$\max_\pi \mathsf{E} C\left(S_T, X_T^\pi\right) \quad \text{or} \quad \mathsf{E} F(X^{\pi,N}, W)$$

or

$$\max_\pi \mathsf{E} \sum_{t=0}^{T} C\left(S_t, X^\pi(S_t \mid \theta)\right) \quad \text{or} \quad \mathsf{E} \sum_{n=0}^{N-1} F\left(X^\pi(S^n \mid \theta), W^{n+1}\right)$$

Evaluating a policy

We simulate a policy N times and take an average:

$$\bar{F}^{\pi} = \frac{1}{N}\sum_{n=1}^{N} F^{\pi}(\omega^n)$$

If we simulate policies π_1 and π_2, we would like to conclude that π_1 is better than π_2 if

$$\bar{F}^{\pi_1} > \bar{F}^{\pi_2}$$

There are some technical issues we deal with later:

- How large should N be?
- How do we deal with the fact that this is at best a statistically noisy measurement?
  - Need to compute confidence intervals
- How do we search over different policies?
  - Stochastic search (or “optimal learning”)

There are two ways to evaluate a policy:

Computer simulation

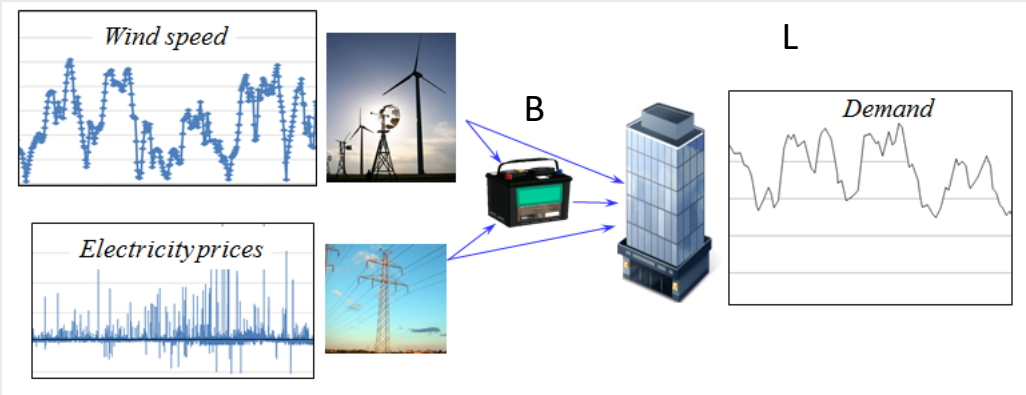
- We use a computer model to simulate a policy.
- Provides controlled testing environment.
- Requires living with model assumptions.
- Can quickly compare different policies.

Real world

- We observe the performance of a policy in the field.
- Have to live with what the real world offers.
- No control over test environment.
- Policy evaluations are quite slow.

An energy storage problem

Designing policies



We can illustrate this problem with each of the four classes of policies

An energy storage problem

Policy function approximation

11.8.1 Policy function approximation

Our policy function approximation is given by

$$X_t^{PFA}(S_t|\theta) = \begin{cases} x_t^{EL} = \min\{L_t, E_t\}, \\ x_t^{BL} = \begin{cases} h_t & \text{If } p_t > \theta^U \\ 0 & \text{If } p_t < \theta^U \end{cases} \\ x_t^{GL} = L_t - x_t^{EL} - x_t^{BL}, \\ x_t^{EB} = \min\{E_t - x_t^{EL}, \rho^{chrg}\}, \\ x_t^{GB} = \begin{cases} \rho^{chrg} - x_t^{EB} & \text{If } p_t < \theta^L \\ 0 & \text{If } p_t > \theta^L \end{cases} \end{cases}$$

where $h_t = \min\{L_t - x_t^{EL}, \min\{R_t, \rho^{chrg}\}\}$. This policy is parameterized by (θ^L, θ^U) which determine the price points at which we charge or discharge.

An energy storage problem

Cost function approximation

11.8.2 Cost function approximation

The cost function approximation minimizes a one-period cost plus a tunable error correction term:

$$X^{CFA-EC}(S_t|\theta) = \arg\min_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \theta(x_t^{GB} + x_t^{EB} + x_t^{BL})\right), \quad (11.25)$$

where $\mathcal{X}_t$ is defined by (9.21)-(9.25). We use a linear correction term for simplicity which is parameterized by the scalar θ.

Notes:

- This is a very simple “cost function correction term” with a scalar parameter θ

An energy storage problem

Value function approximation

11.8.3 Value function approximation

Our VFA policy uses an approximate value function approximation, which we write as

$$X^{VFA}(S_t) = \arg\min_{x_t \in \mathcal{X}_t} \left(C(S_t, x_t) + \overline{V}_t^x(R_t^x)\right), \quad (11.26)$$

where $\overline{V}_t^x(R_t^x)$ is a piecewise linear function approximating the marginal value of the post-decision resource state. We use methods described in chapter 19 to compute the value function approximation which exploits the natural convexity of the problem. For now, we simply note that the approximation is quite good.

We will describe methods for estimating $\bar{V}_t^x(R_t^x)$ later in the course.

An energy storage problem

Direct lookahead

11.8.4 Deterministic lookahead

The next policy is a deterministic lookahead over a horizon H which has access to a forecast of wind energy.

$$X_t^{LA-DET}(S_t|H) = \underset{(x_t,\tilde{x}_{t+1,t},\ldots,\tilde{x}_{t,t+H})}{\arg\min} \left(C(S_t, x_t) + \sum_{t'=t+1}^{t+H} C(\tilde{S}_{tt'}, \tilde{x}_{tt'}) \right) \quad (11.27)$$

subject to, for $t' = t, \ldots, T$:

$$\tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{EB} \leq f_{tt'}^{E}, \quad (11.28)$$

$$f_{tt'}^{\eta}(\tilde{x}_{tt'}^{GL} + \tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{BL}) = f_{tt'}^{L}, \quad (11.29)$$

$$\tilde{x}_{tt'}^{BL} \leq \tilde{R}_{tt'}, \quad (11.30)$$

$$\tilde{R}_{t,t'+1} - (\tilde{R}_{tt'} + f_{t,t'+1}^{\eta}(\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{EB}) - \tilde{x}_{tt'}^{BL}) = f_{t,t'+1}^{R}, \quad (11.31)$$

$$\tilde{x}_{tt'} \geq 0. \quad (11.32)$$

This is a classic rolling horizon procedure.

TU/e

Recap on the four fundamental classes

The four fundamental policies

- Policy Function Approximations (PFAs)
- Cost Function Approximations (CFAs)

- Value Function Approximations (VFAs)
- Direct lookahead approximations (DLAs)

- Recal that at time t, we observe state S_t, face action space $X(S_t)$, and need a decision rule π or policy to take decision $x_t \in X(S_t)$

Policy Function Approximations (PFAs)

These are analytical functions that map a state (which includes all the information available to us) to a decision.

- Lookup tables
- Parametric functions (OLS, regression)
- Nonparametric functions (deep neural nets)

PFAs in the Hydrogen Plant

- In reality, electricity needs to be bid on the day-ahead market
- Neural net that uses information of today (weather, prices, weather forecasts) for tomorrows prices so that we can strategically bid
- Lookup Table: <If price is above a threshold sell, otherwise buy>

Cost Function Approximations (CFAs)

We rely on solving a parameterized optimization problems for each state. This optimization problem gives us a decision. The parameters steer which decision is taken

Example: Google Maps – You enter a destination, it says 1h20m. You leave 1h30m because you buffer against uncertainty. This is a very simple cost function approximation, you just add a certain amount of time to the problem.

CFAs in the Hydrogen Plant

- We need to fulfill PPA obligations – scale the time until deadline with some parameter
- Past variability in prices, parameterize and see how we can act upon that (direct sell, later sell/buy)

Value Function Approximations (VFAs)

- We define the value of being in a state as $V_t(S_t)$, and we want to solve:

$$V_t(S_t) = \max_{x_t \in X(S_t)} \left(C(S_t, x_t) + E_{W_{t+1}}\{V_{t+1}(S_{t+1})|S_t\} \right)$$

- We can do this to optimality by dynamic programming and Markov decision process theory.
  - We will discuss the details of this later in this course.

- A value function approximation provides an approximation of $V_{t+1}(S_{t+1})$. In its most simplest form (basically a simulation) it is called Q-learning. All more efficient forms are typically some form of Deep Reinforcement Learning or use techniques from Approximate Dynamic Programming.

VFA for the Hydrogen market

- Only look two periods ahead;
- Simulate future events to update value functions
- Stochastic Dual Dynamic Integer Programming (google it, it is AMAZING)
- Solve to optimality

Chapter 3:

TD Learning of a policy

Temporal difference learning

The "temporal difference" is given by

$$\begin{aligned}\delta_t(S_t^n, x_t^n) &= \bar{V}_t^{n-1}(S_t^n) - \left(C(S_t^n, x_t^n) + \bar{V}^{n-1}(S^M(S_t^n, x_t^n, W_{t+1})\right) \\ &= \bar{V}_t^{n-1}(S_t^n) - v_t^n\end{aligned}$$

In other words, this is "old estimate minus new estimate". We can update our value function approximation using

$$\begin{aligned}\bar{V}_t^n(S_t^n) &= \bar{V}_t^{n-1}(S_t^n) - \alpha_{n-1}\delta(S_t^n, x_t^n) \\ &= (1 - \alpha_{n-1})\bar{V}_t^{n-1}(S_t^n) + \alpha_{n-1}v_t^n\end{aligned}$$

This is a basic form of "temporal difference learning" known as TD(0). The "temporal difference" reflects learning from one decision to the next (which occurs over time).

TD(λ)

A more general form of TD-learning uses discounted costs over the entire trajectory.

$$\overline{V}_t^n(S_t) = \overline{V}_t^{n-1}(S_t) + \alpha_{n-1}\sum_{\tau=t}^{T}\lambda^{\tau-t}\delta_\tau. \quad (17.7)$$

We derived this formula without a time discount factor. We leave as an exercise to the reader to show that if we have a time discount factor γ, then the temporal-difference update becomes

$$\overline{V}_t^n(S_t) = \overline{V}_t^{n-1}(S_t) + \alpha_{n-1}\sum_{\tau=t}^{T}(\gamma\lambda)^{\tau-t}\delta_\tau. \quad (17.8)$$

Think of λ as an "algorithmic discount factor" that helps to give credit for downstream rewards to earlier decisions. This has to be carefully tuned.

Approximating the value function

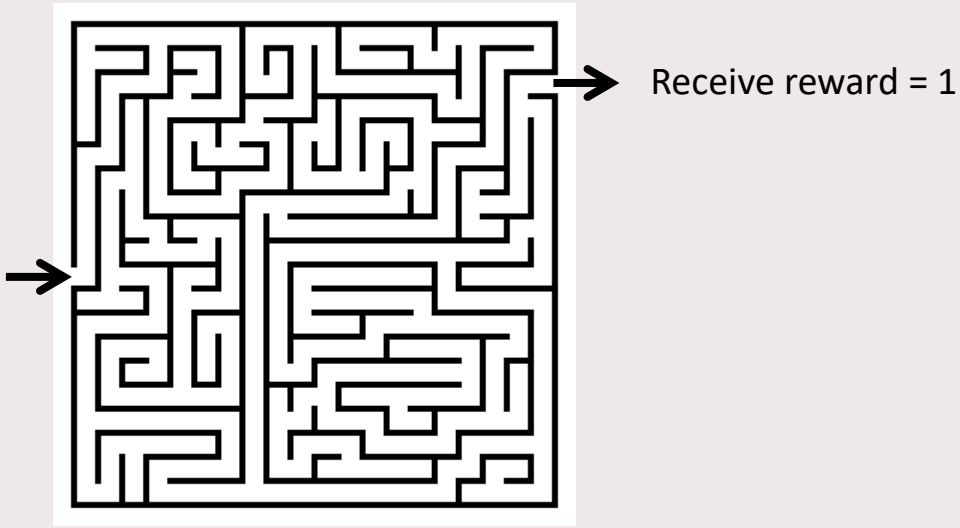
Temporal difference updates can be used in any recursive estimation algorithm:

- Lookup tables
  - Independent beliefs
  - Correlated beliefs
- Parametric models
  - Linear
  - Nonlinear
  - Shallow neural networks
- Nonparametric
  - Kernel regression
  - Locally linear
  - Deep neural networks

Chapter 4:

Q-learning (only the basic ideas)

Mouse in a maze problem

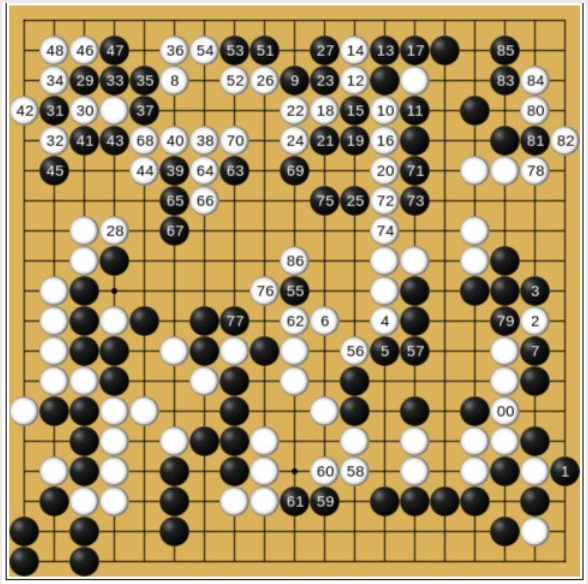


AlphaGo

Much more complex state space.

Uses hybrid of policies:

- PFA
- VFA
- Lookahead (DLA)



Basic Q-learning algorithm

Basic update:

where

$$\hat{q}^n(s^n,a^n) = C(s^n,a^n) + \gamma \max_{a'} \bar{Q}^{n-1}(s',a')$$

$$\bar{Q}^n(s^n,a^n) = (1-\alpha_{n-1})\bar{Q}^{n-1}(s^n,a^n) + \alpha_{n-1}\hat{q}^n(s^n,a^n)$$

$$s' = S^M\left(s^n,a^n,W^{n+1}\right)$$

Given a state s^n and action a^n, we simulate our way to state s'.

Need to determine:

- State sampling process/policy
- Action sampling policy

Some terms from reinforcement learning:

“Behavior policy” is the policy used to choose actions

- E.g. these are actions observed by a real system

“Target policy” is the policy that we are trying to learn, which is to say the policy we want to implement.

When the target policy is different from the behavior policy, then this is termed “off policy learning”

In this course

The “learning policy” is the policy (often called an algorithm) that learns the value functions (or Q-factors)

The “implementation policy” is the policy determined by the value functions (or Q-factors).

Learning policy

This is the policy that determines what action to choose as a part of learning the Q-factors.

"Exploitation":

$$a^n = \arg\max_{a'} \bar{Q}^n(s^n, a')$$

Other policies that involve exploration:

- Epsilon-greedy – Choose greedy policy with probability ϵ, and explore with probability $1 - \epsilon$.
- Policies based on upper confidence bounding, Thompson sampling, knowledge gradient, …

State sampling policies

Trajectory following

$$s^{n+1} = S^M(s^n, a^n, W^{n+1})$$

- Helps to avoid sampling states that never happen
- Problem is that a suboptimal policy may mean that you are not sampling important states.

Exploration

- Pick a state at random

Hybrid

- Use trajectory following with randomization, e.g. $s^{n+1} = S^M(s^n, a^n, W^{n+1}) + \epsilon^{n+1}$

On vs off-policy learning:

On-policy learning – Learning the value of a fixed policy:

From a state s^n, choose action

$$a^n = \arg\max_{a'} \bar{Q}^n(s^n, a')$$

Now go to state s^{n+1}:

$$s^{n+1} = S^M(s^n, a^n, W^{n+1})$$

Where W^{n+1} is observed or sampled from some distribution.

Off-policy learning:

- Sample actions according to a *learning policy* (called "*behavior policy*" in the RL literature). This is the policy used for learning the *implementation policy* (called the "target policy" in the RL literature).
- Needs to be combined with a state sampling policy.

Model-free vs. model-based

Model-based means we have a mathematical statement of how the problem evolves that can be simulated in the computer.

Model-free refers to a physical process that can be observed, but where we do not have equations describing the evolution over time.

- The behavior of a human or animal
- The behavior of the climate
- The behavior of a complex system such as a chemical plant

Q-learning is often described as “model free” because it can be learned while observing a system. The resulting policy does not require a model:

- $A^{\pi}(s) = argmax_a \bar{Q}^n(s,a)$

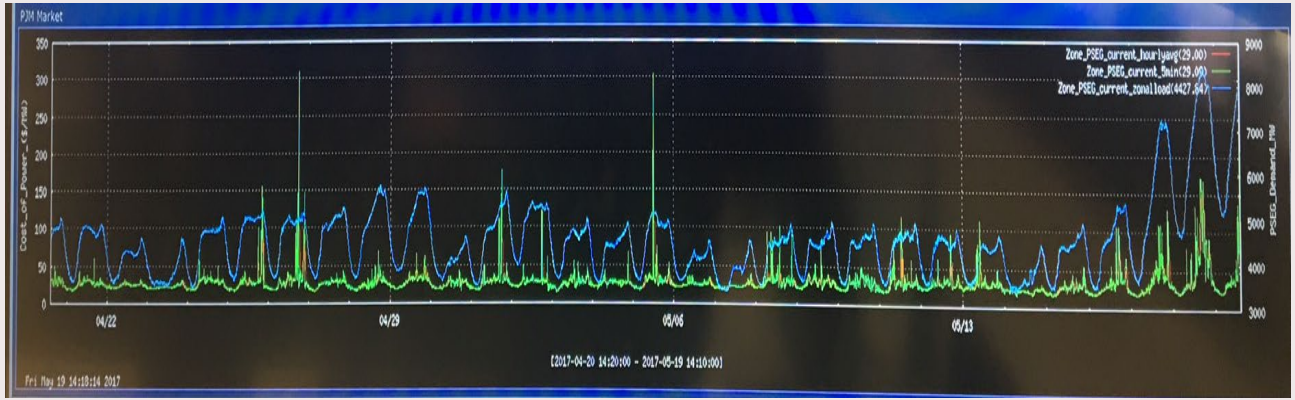
Q-learning

Max operator bias:

Second issue arises when there is randomness in the reward.

Imagine that we are purchasing energy at a price p_t which evolves randomly from one time period to the next.

Imagine buying and selling energy using real time prices:



Q-learning

Max operator bias (cont'd)

This introduces noise in $\hat{q}^n(s^n, a^n)$:

$$\hat{q}^n(s^n, a^n) = C(s^n, a^n) + \gamma \max_{a'} \bar{Q}^{n-1}(s', a')$$
$$\bar{Q}^n(s^n, a^n) = (1 - \alpha_{n-1})\bar{Q}^{n-1}(s^n, a^n) + \alpha_{n-1}\hat{q}^n(s^n, a^n)$$

Finding the max over a set of noisy estimates $\hat{q}^n(s^n, a^n)$ introduces bias in the estimates $\bar{Q}^{n-1}(s', a')$. This bias can be quite large.

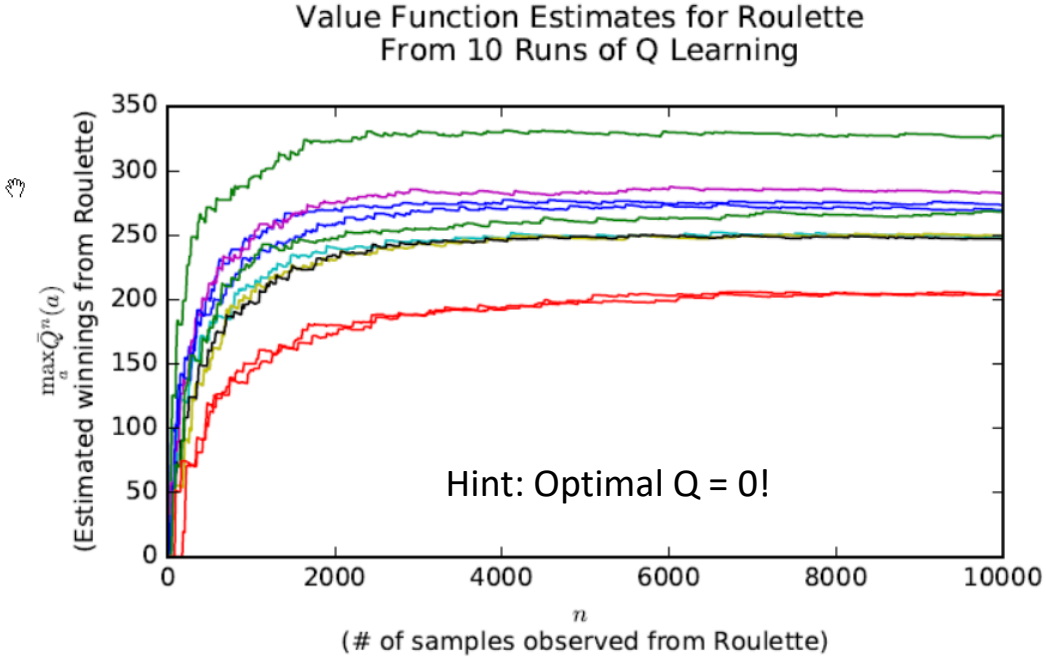
Testing on roulette

Q-learning

Roulette

Optimal solution is not to play – optimal value of game is zero

Q-learning over 10,000 iterations

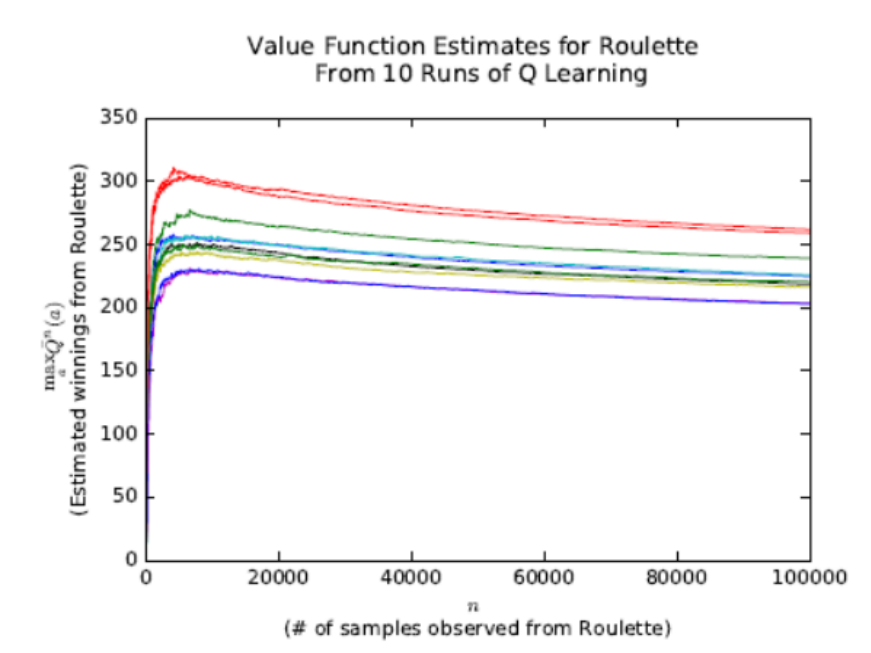


Q-learning

Roulette

Optimal solution is not to play – optimal value of game is zero

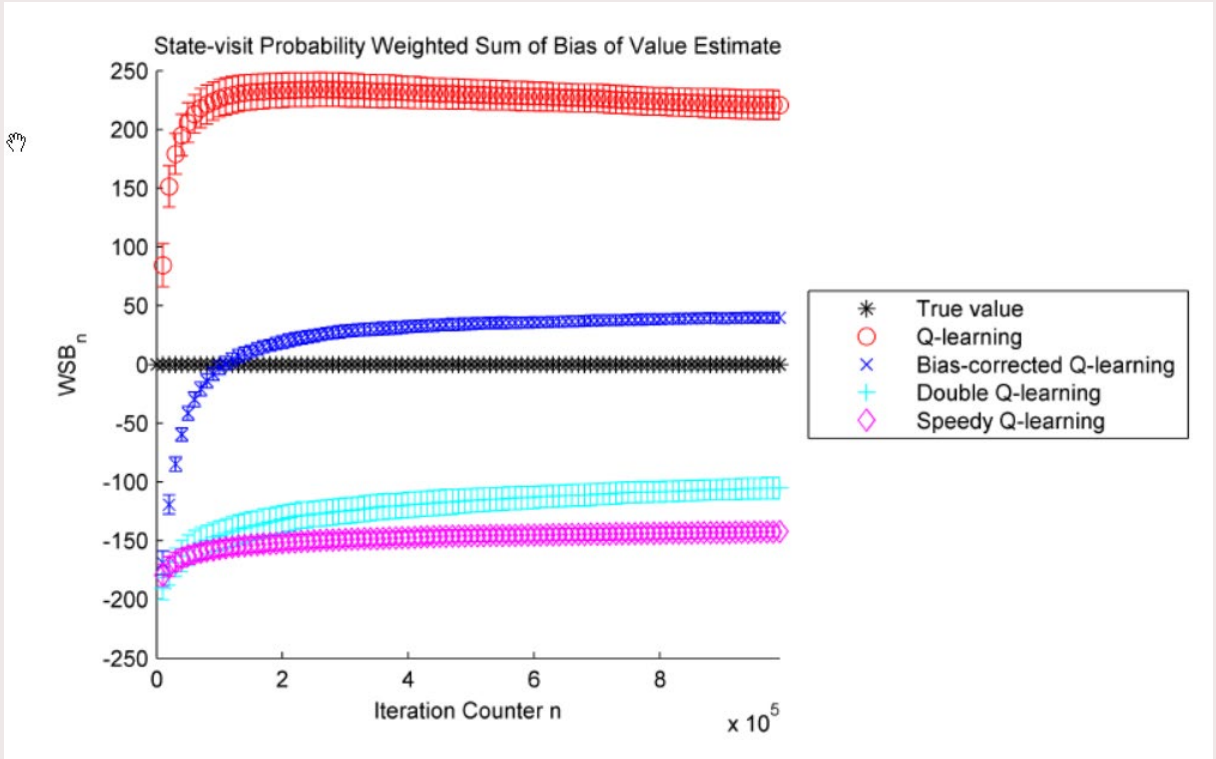
Q-learning over 10,000 iterations



Q-learning

Roulette

Optimal solution is not to play – optimal value of game is zero



Algorithms

Approximate value iteration

Single-pass

- Need to define policies for choosing states and actions

Double-pass with “discount” λ

Approximate policy iteration

Relationship to TD-learning

Approximate value iteration uses TD(0) updates.

Approximate policy iteration uses TD(1) updates.

Chapter 5:

Approximate Value and Policy Iteration (- for selfstudy)

Approximate value iteration

Step 1: Start with a pre-decision state S_t^n

Step 2: Solve the deterministic optimization using an approximate value function:

to obtain x^n. $\hat{v}_t^n = \min_x \left(C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right)$

Deterministic optimization

Step 3: Update the value function approximation

Step 4: Obtain Monte Carlo sample of $W_t(\omega^m)$ and compute the next pre-decision state:

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1-\alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n$$

Recursive statistics

Step 5: Return to step 1.

Simulation

$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

TU/e

Approximate value iteration

Step 1: Start with a pre-decision state S_t^n

Step 2: Solve the deterministic optimization using an approximate value function:

to obtain x^n.

$$\hat{v}^m = \min_x \left(C(S^m, x) + \sum_f \theta_f^{n-1} \phi_f(S^M(S^m, x)) \right)$$

Deterministic optimization

Linear model for post-decision state

Step 3: Update the value function approximation

$$\bar{V}_{t-1}^n(S_{t-1}^{x,n}) = (1-\alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n$$

Recursive statistics

Step 4: Obtain Monte Carlo sample of $W_t(\omega^m)$ and compute the next pre-decision state:

$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Simulation

"Trajectory following"

Step 5: Return to step 1.

Approximate policy iteration

Step 1: Start with a pre-decision state S_t^n

Step 2: Inner loop: Do for m=1,…,M:

Step 2a: Solve the deterministic optimization using an approximate value function:

to obtain x^m.

$$\hat{v}^m = \min_x \left(C(S^m, x) + \bar{V}^{n-1}(S^{M,x}(S^m, x)) \right)$$

Step 2b: Update the value function approximation

$$\bar{V}^{n-1,m}(S^{x,m}) = (1-\alpha_{m-1})\bar{V}^{n-1,m-1}(S^{x,m}) + \alpha_{m-1}\hat{v}^m$$

Step 2c: Obtain Monte Carlo sample of $W(\omega^m)$ and compute the next pre-decision state: $S^{m+1} = S^M(S^m, x^m, W(\omega^m))$

Step 3: Update $\bar{V}^n(S)$ using $\bar{V}^{n-1,M}(S)$ and return to step 1.

Chapter 6:

Scenario based planning for dynamic vehicle routing problems

Partially dynamic vehicle routing problem

- We consider N customer regions, each having N' customer service requests. We consider a single depot 0 where m identical vehicles are available with capacity Q. Travel costs are denoted by c_{ij} and are assumed to satisfy the triangle inequality.
- Each customer service requires has a service time s_i and a demand q_i
- Each customer has a time window specified by $[e_i, \ell_i]$

- A solution is a routing plan where each route corresponds to a vehicle, so that the routes respect vehicle capacity and time windows

- Customers are only partially known at T = 0 !! They arrive dynamically

The basis: A greedy approach

- Suppose we have available some heuristic or method to insert a newly arriving customer request in an existing routing plan:
- Suppose we have available some sophisticated local search method to generate a start solution (with the customer that are known at the start)

- When a request arrives, the heuristic greedy approach inserts whether there is a feasible insertion, and if not, disregards the customer.
- One can reoptimize using the sophisticated local search.
- This typically does not work well because very restrictive solutions are found

The Multiple Plan Approach

- Instead of relying on a single plan, keep multiple plans into memory.
- One of the plans is the distinguished plan (i.e., the one we currently execute)

- Any MPA approach consists of four events
  - Customer requests (require plan updates)
  - Vehicle departures (make plans invalid)
  - Plan generations (may change the distinguished plan)
  - Timeouts (plans become invalid)

The Four Events further detailed.

Let S_t be the set of plans, with σ_t^* the distinguished plan. The question is how the transition function is defined for each of the four events. For that, we need the **Consensus functions** f_t that map each plan at time t to a real value, and so provides an ordering to the plan and selects the distinguished plan.

It is your role as a policy maker to define what happens when the four events take place.

Consensus Function

- It might be intuitive to select a consensus function based on the cost of a plan, however, this is often not a good choice

- It is better to find a plan that is most similar to other plans
  - For instance by comparing customer visits in the plans

The Multiple Scenario Approach

- We have a greedy algorithm, that provides the basic solution to any dynamic vehicle routing problem
- We have a multiple plan approach, that improves dramatically upon the greedy algorithm

- Now we introduce the multiple scenario approach, improving drastically upon the multiple plan approach.

The Multiple Scenario Approach

- The funny part is, this is quite simple, but extremely efficient.
- We select a sample of future scenarios (by generating new customers according to known probability distribution functions)
- For each scenario we obtain a routing

Chapter 7:

Dynamic Look-ahead policies for bike-sharing rebalancing

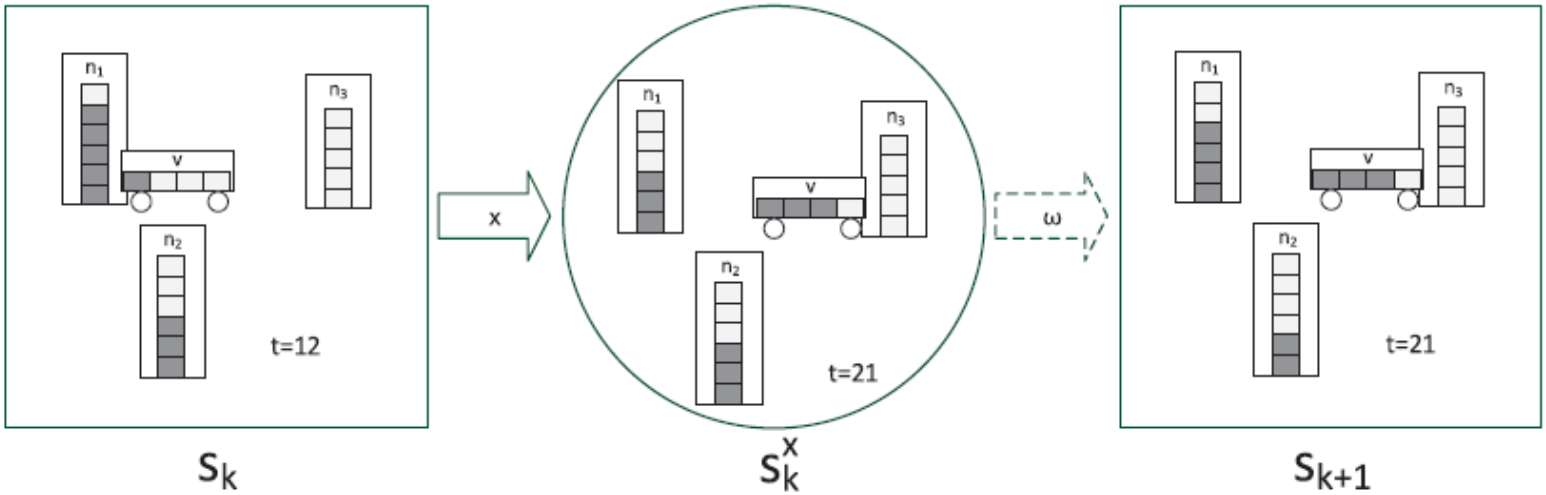
Stochastic-Dynamic inventory routing problem for bike sharing systems

- Station-based bike sharing system (BSS) just as in our case.
- Set of stations N, time horizon T of current capacity c

- A single vehicle transports bikes between stations.

- We will not in detail model the SSDP elements (neither did the authors, see Brinkmann et al. (2019)

Pre and post decision states – relevant for our Case!



Dynamic Look-ahead

- Consists of two parts

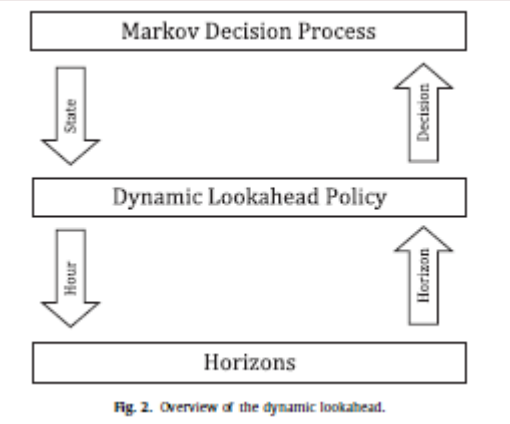


Fig. 2. Overview of the dynamic lookahead.

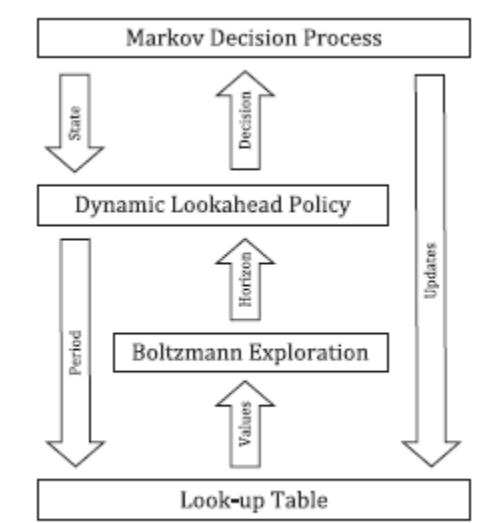


Fig. 3. Approximation run of the value function approximation.

The lookahead policy itself

- Determines the amount of bike to pickup at current station and where to go next

- Consider three possible inventory decisions – fill/deplete up to 25,50,75 %
- Simulate future demand using 32 simulations for each fill rate, and inventory is chosen as the one leading to the least amount of unsatisfied demand at this station.

- The amount of time that is looked into the future is determined exogenously (see next part)

- Subsequently, a routing decision is made.
- The vehicle is sent to the location that has the most failed rentals based on simulation over the exogenously predetermined horizon.

The dynamic part of the lookahead policy

- This part determines the exogenous horizon that is used in the simulations at state.
- Each hour of the day can be set to a particular time horizon (in hours)
  - Total number of parameterizations equals 8.2x10e18
- To search through the parameter space smartly, we use a Value Function Approximation

Value function approximation

- In the SSDP language, this can be seen as a nonparametric policy search
- We create a lookup table storing hour, parameter pairs:

$$\nu(\rho_i, \delta_j) = \underbrace{\mathbb{E}\left[\sum_{k=\bar{k}^{\rho_i}}^{\underline{k}^{\rho_i}} p\left(s_k, \pi^{\delta_j}(s_k)\right) \middle| s_{k_{\rho_i}}\right]}_{\text{current hour}} + \underbrace{\begin{cases} \min_{\delta \in \Delta} \nu(\rho_{i+1}, \delta) & \text{, if } \underline{k}^{\rho_i} \neq k_{\max} \\ 0 & \text{, else} \end{cases}}_{\text{future hours}}$$

$$\tilde{\nu}(\rho, \delta^\rho) := \underbrace{\frac{\alpha(\rho, \delta^\rho) - 1}{\alpha(\rho, \delta^\rho)} \cdot \tilde{\nu}(\rho, \delta^\rho)}_{\text{old approximation}} + \underbrace{\frac{1}{\alpha(\rho, \delta^\rho)} \cdot \hat{\gamma}(\rho)}_{\text{new observation}}.$$

Chapter -1:

Some final notes on coding and the case.

General coding structure for SSDPs

- We consider two main objects or entities
- Environment: That hosts the model of the real world and takes actions
- Trainer: That decides which action is taken

The trainer *can* work completely model free (i.e., it can be put in any)

- An action is simply some ‘number’, and every ‘number’ is feasible and represents an actual action in the environment
- Meaning that the environment should be able to translate/hash this ‘number’

Learn by real-world or learn by model

- This moment, the case-study (and the code provided) works on a learn by model basis
- Distributions are fitted to the input data.
- We strongly urge you to first work in this setting, and only if some moderate success has been achieved improve how well the real world is presented in the model.

The structure of * ANY * SSDP

While (**not** environment.game_over)

- State = environment.getState()
- Action = trainer.getAction(State)
- (Reward) = environment.Step(Action)
- Trainer.update(State, Action, Reward)

In the case-study, we will “cheat” a little bit:

- We assume the trainer does know about the model
- Why wouldn’t we make use of it ?

Go to python code and show its main elements