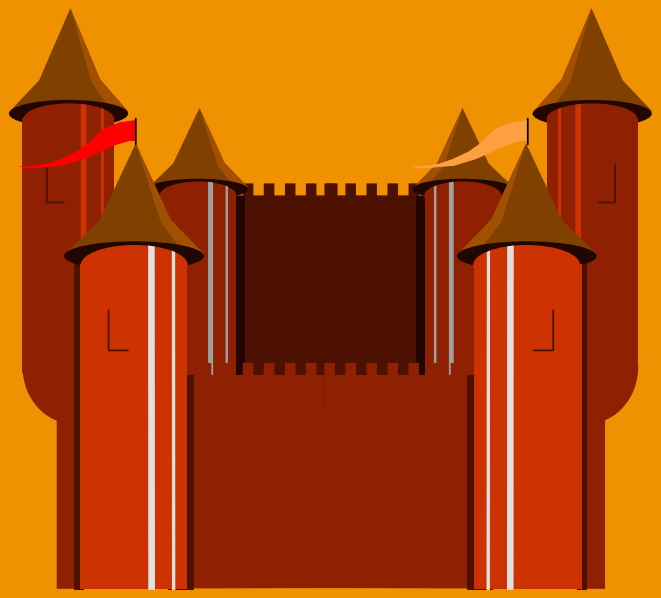
ORF 411

Sequential decision analytics and modeling

Fall, 2018



Warren Powell
Princeton University
http://www.castlelab.princeton.edu

Week 0 - Wednesday

Introduction

Some administrative issues

Administration

- Distribute course questionnaire
- When to hold office hours?
  - » Monday afternoon and evening – how many cannot make one of these two slots.
  - » For the first problem set, we need to set up a session to introduce students who need it to github and the python library. Please sign up on the doodle poll.
- Ask questions in class!!

Illustrations of sequential decision problems

Learning the market in Africa

- How do African consumers respond to energy pricing strategies for recharging cell phones?
  - » Cell phone use is widespread in Africa, but the lack of a reliable power grid complicates recharging cell phone batteries.
  - » A low cost strategy is to encourage an entrepreneurial market to develop which sells energy from small, low cost solar panels.
  - » We do not know the demand curve, and we need to learn it as quickly as possible.

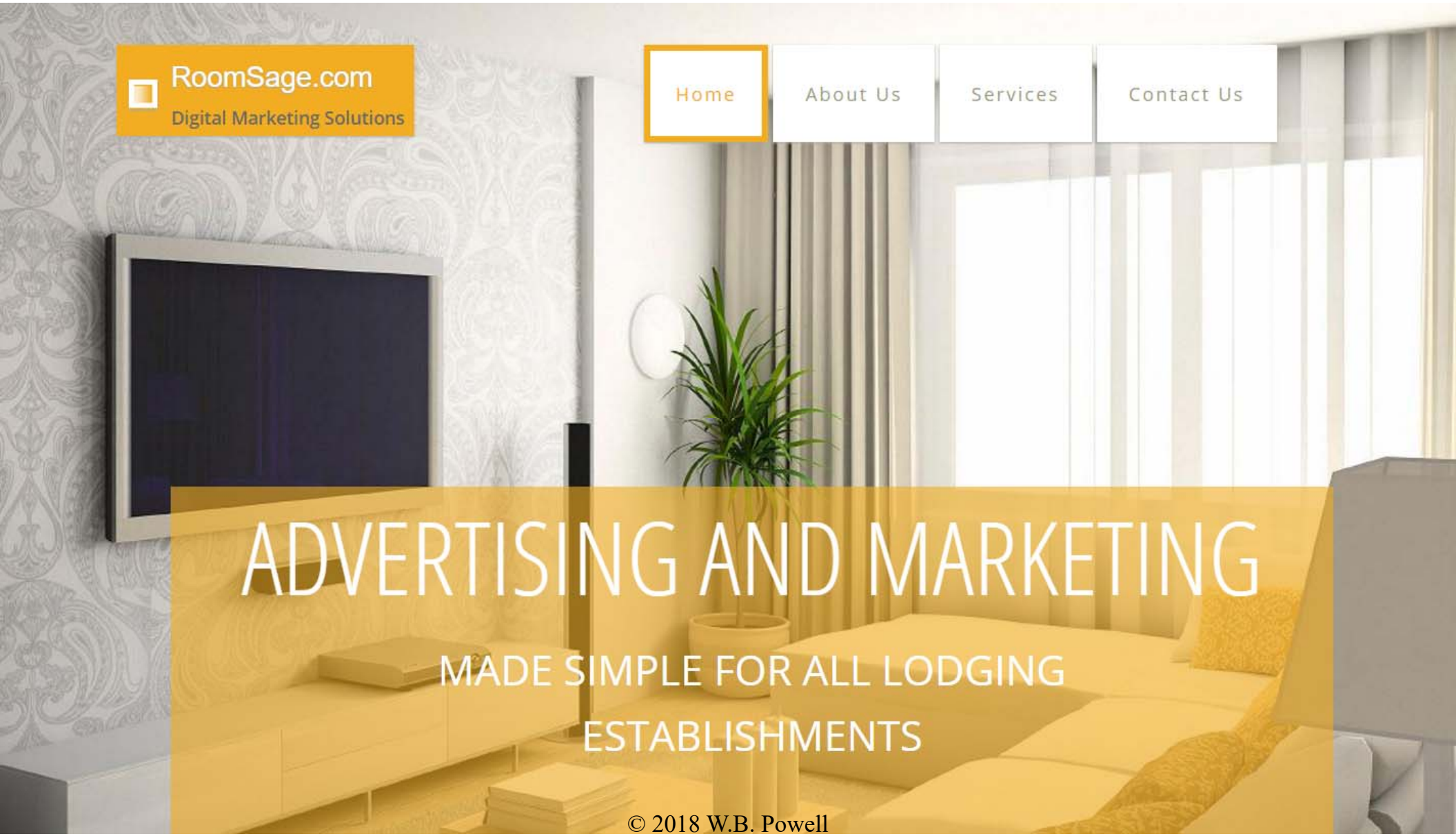
Cell Charging Challenges:
Using Optimal Learning to Determine the Profitability of a Solar Mobile Charging System in Africa

Interim Progress Report

Megan J. Wong
1/31/2011

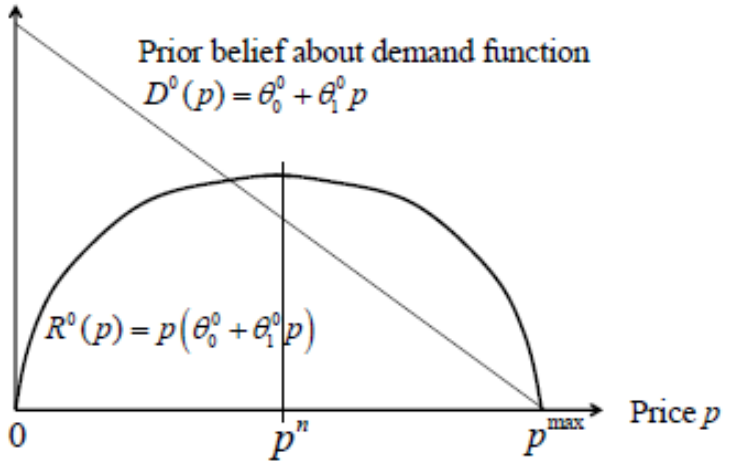
Roomsage.com

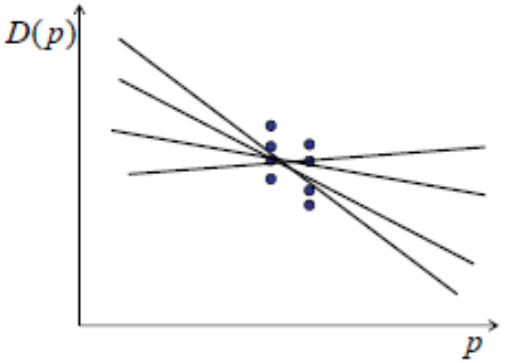
- Robert Wojciechowski ‘98



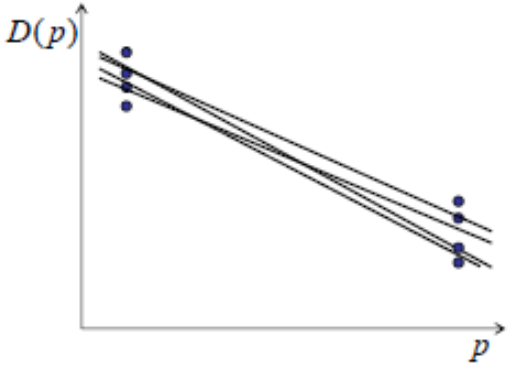
Revenue management

- Earning vs. learning
  - » You want to maximize revenues, but you do not know how demand responds to price.





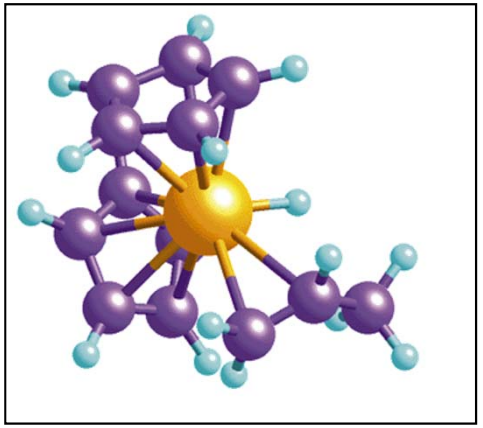
You earn the most with prices near the middle, but you do not learn anything.

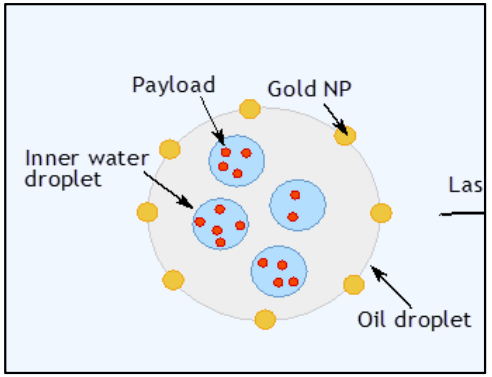


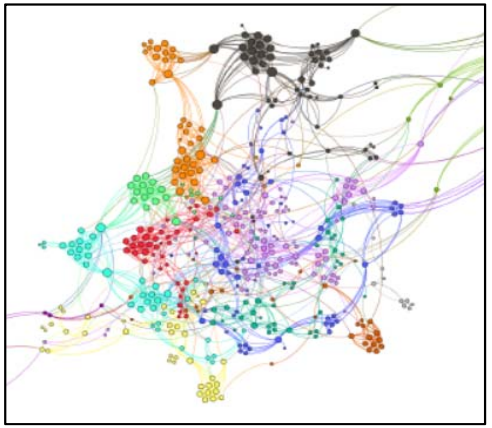
You learn the most by sampling endpoints, but then you do not earn anything.

Health applications

- Health sciences
  - » Sequential design of experiments for drug discovery
  - » Drug delivery – Optimizing the design of protective membranes to control drug release
  - » Medical decision making – Optimal learning for medical treatments.

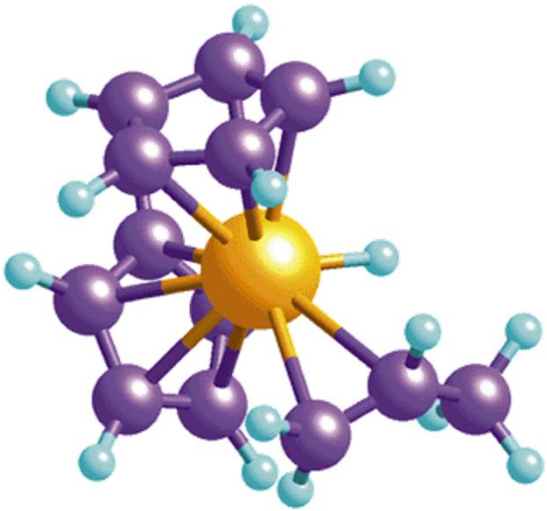


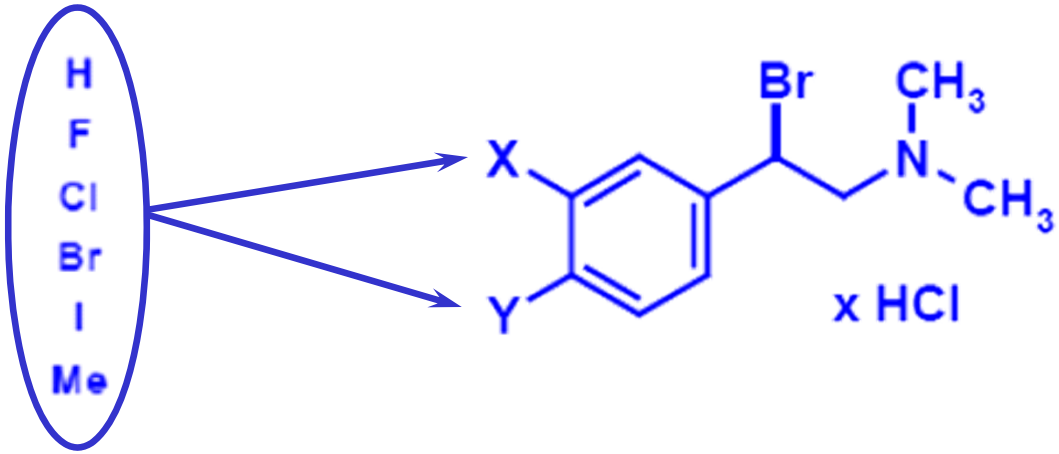




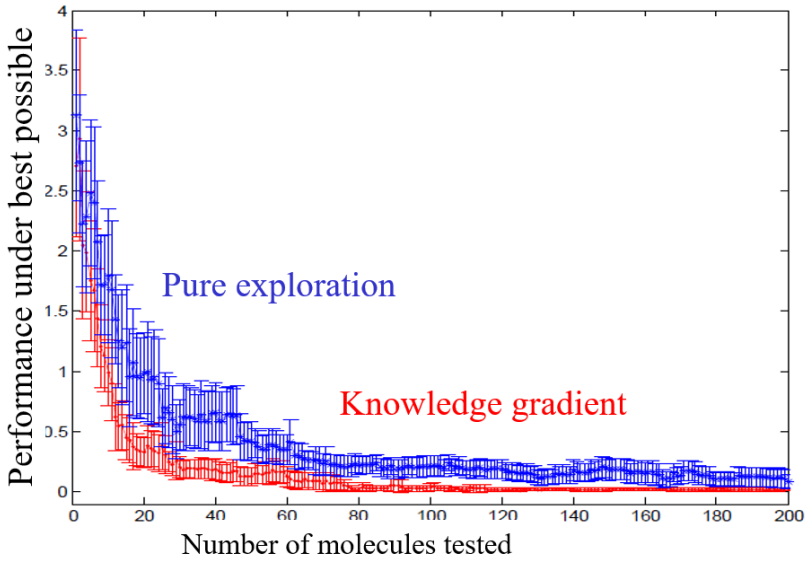
Drug discovery

- Optimizing the configuration of molecules



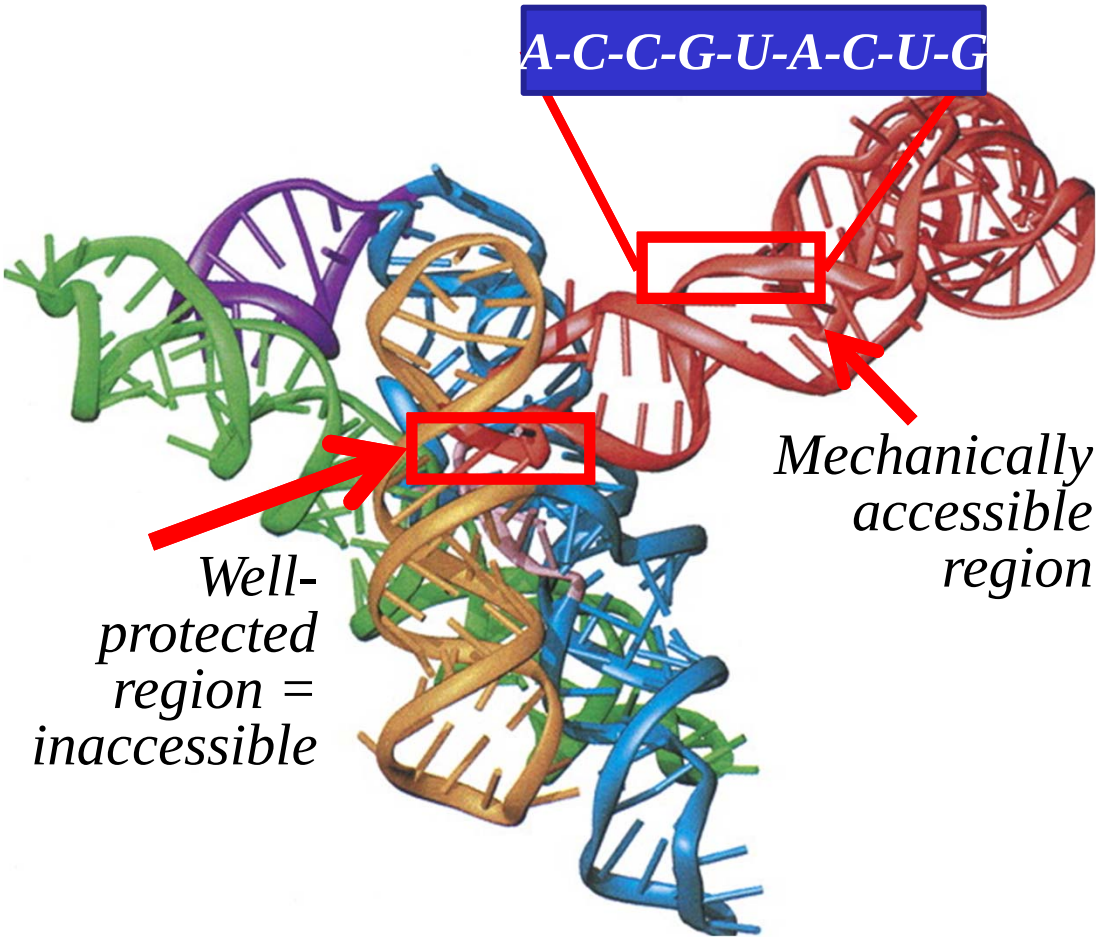


Design of effective policies can accelerate the search process for new drugs.



Accessibility of an RNA molecule

- *Can we efficiently learn the accessible regions of an RNA molecule?*



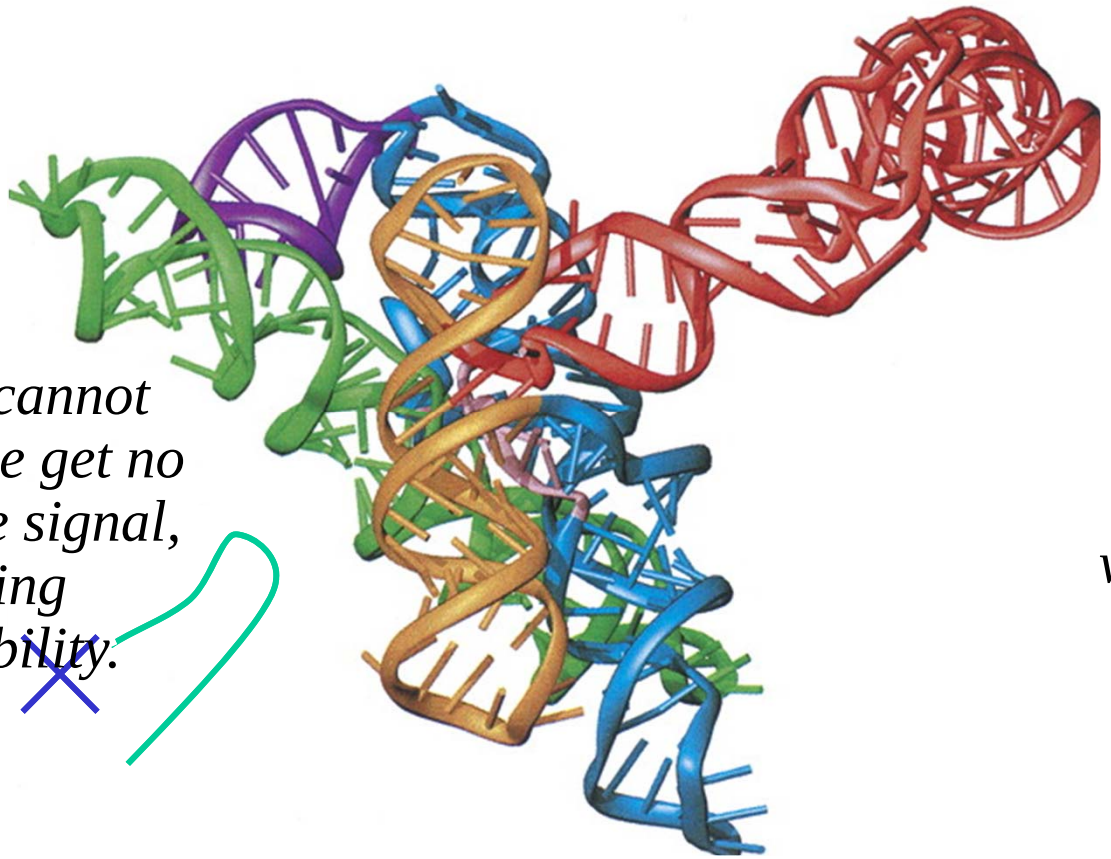
- *The accessibility of a region of an RNA molecule mediates interaction with other molecules.*
- *Depends on how you define interactions, resulting in several methods for accessing accessibility.*
  - *Mechanical*
  - *Energetic*

Image courtesy Adams et al., RNA (2004)

Binding Probe/Reporter Complex

New Methodology: attempt to bind probe/reporter complex with fluorescent marker. If bound, we observe strong fluorescence signal.

Probe/Reporter complex

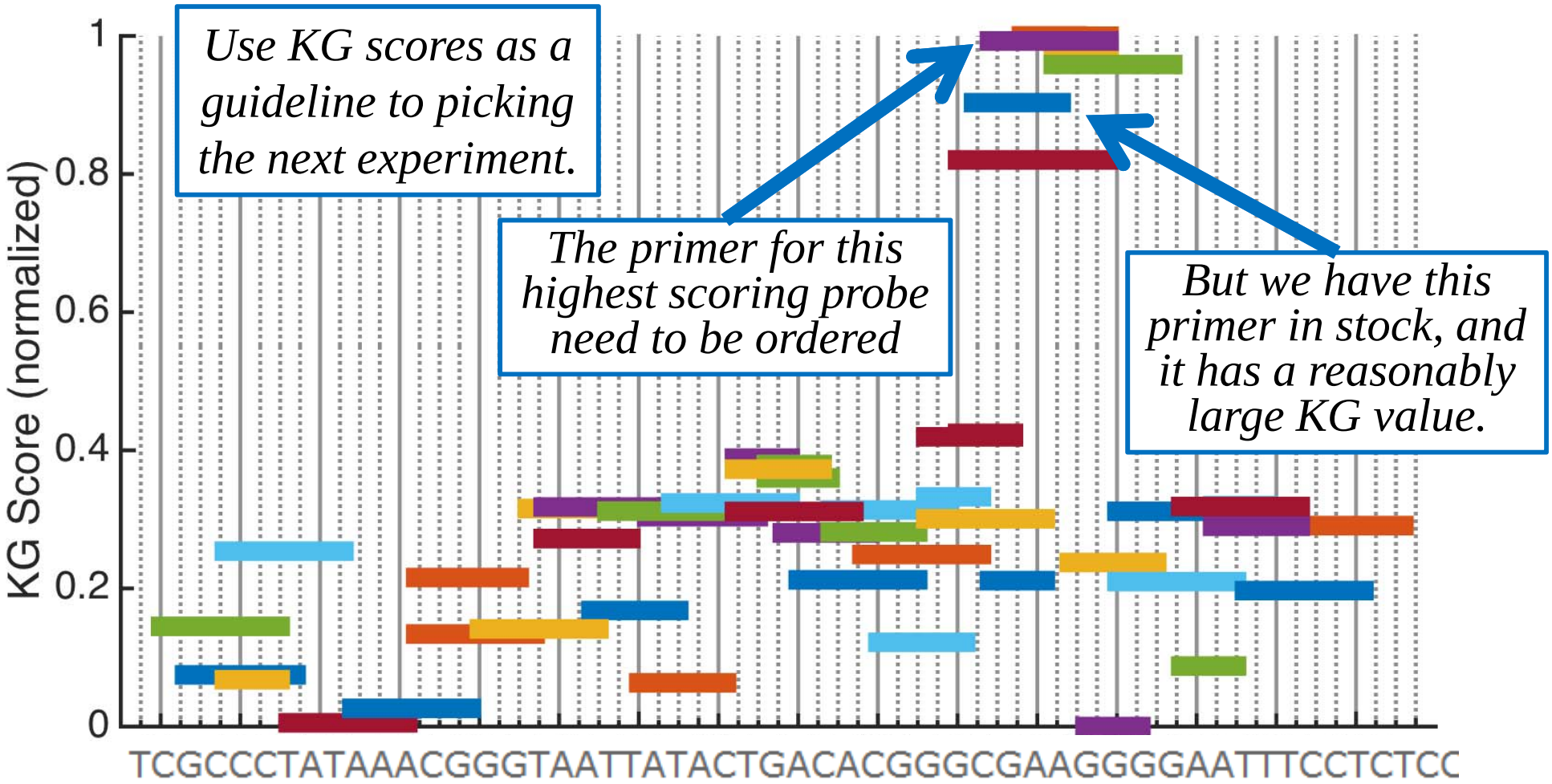


If a probe cannot bind, then we get no fluorescence signal, indicating inaccessibility.

If probe can bind, we get fluorescence signal, indicating accessibility.

Image courtesy Adams et al., RNA (2004)

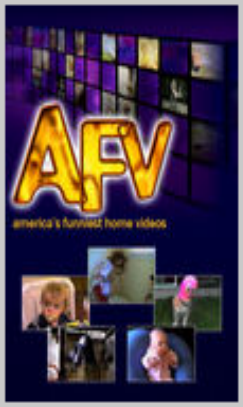
Knowledge Gradient scores



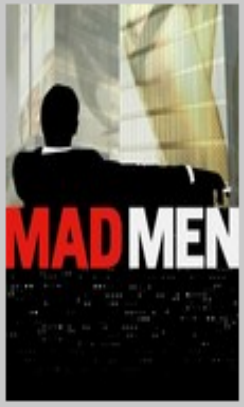
NETFLIX

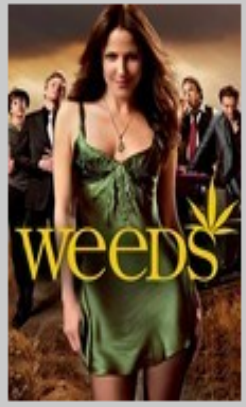
Member Sign In

New Arrivals in TV



TV Drama

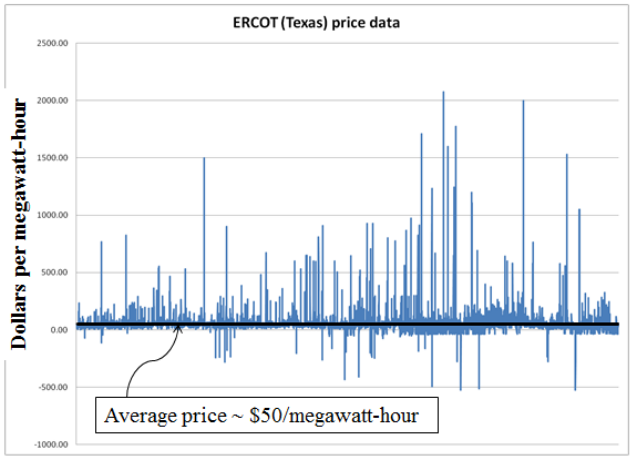




TV Comedy

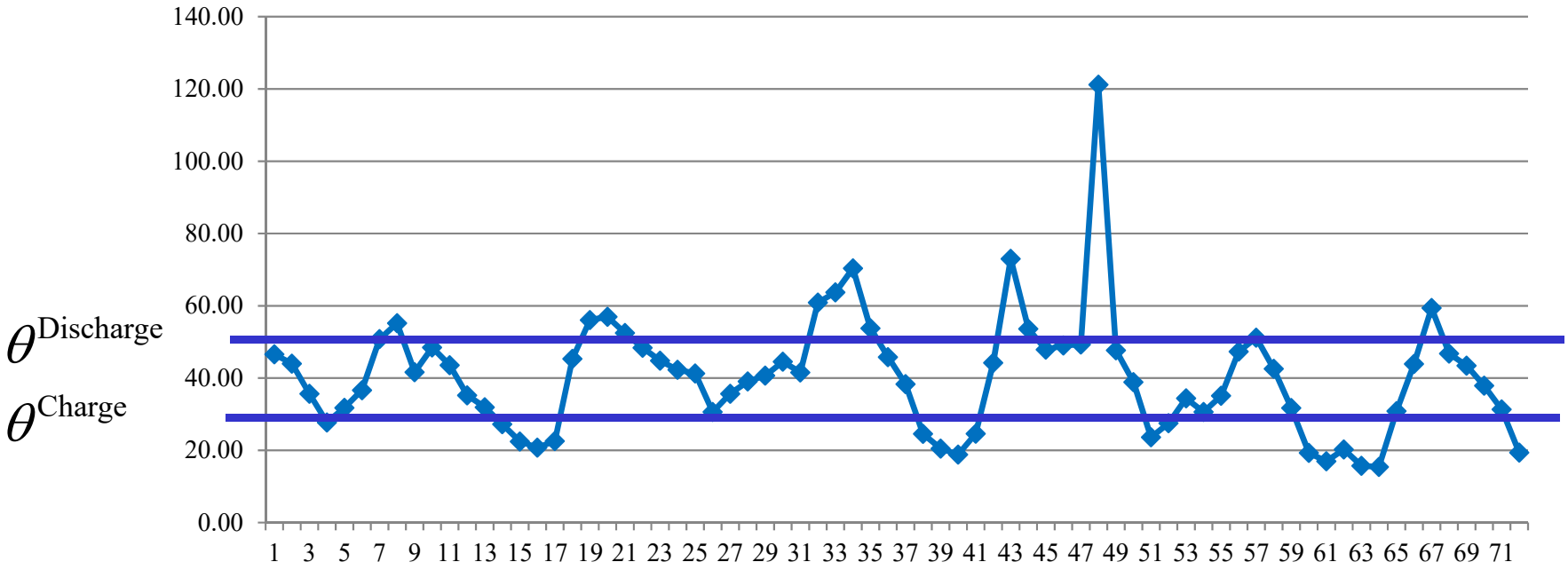
Energy arbitrage

- Battery arbitrage – When to charge, when to discharge, given volatile LMPs



Policy function approximations

- Grid operators require that batteries bid charge and discharge prices, an hour in advance.



- We have to search for the best values for the policy parameters θ^{Charge} and $\theta^{\text{Discharge}}$.

Energy storage

- How much energy to store in a battery to handle the volatility of wind and spot prices to meet demands?

Energy usage management

- Chilled water management at Princeton University
  - » Major issue is how much to store in chilled water tank

Capacity (Calc)	8,584 TonHrs
Flow	3,177 gpm
Storage	21,132 TonHrs
DISCHARGE RATE	2361.3 Tons

[TES Chillers] Total Tons	0.00 ktons
Tank ChargeRate (by chlr flow)	0.00 ktons
Difference going to HX	0.00 ktons

HX Sensors

	FCV	Inlet °F	Outlet °F	ΔT	Flow (gpm)	Tons
HX-2100						
Cold		31.9	51.7	19.8	2	0
Hot		53.4	40.6	12.9	2	0
HX-2200						
Cold		31.0	51.4	20.4	1,064	904
Hot		52.4	40.2	12.2	1,756	889
HX-2300						
Cold		31.4	52.0	20.5	1,065	911
Hot		52.0	40.7	11.4	1,666	790
HX-2400						
Cold		31.3	51.9	20.6	1,074	922
Hot		52.2	41.0	11.2	1,784	830
Totals						
Cold					3,204	2736
Hot					5,208	2509

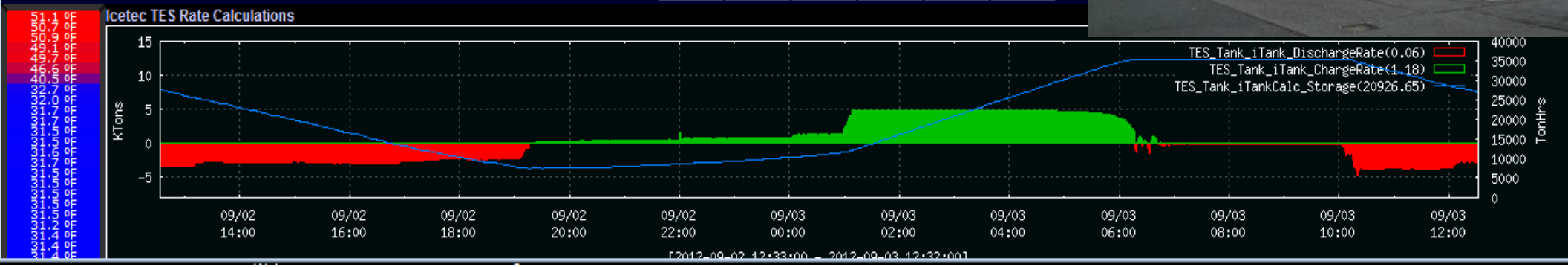
Chiller Sensors

	Inlet °F	Outlet °F
CH-2100		
Evap	52.3	51.9
CW	77.4	77.5
CH-2200		
Evap	51.4	49.5
CW	77.5	77.5

TES Plant Pumps

	2100	2200
CHW	ON	OFF
TES	OFF	ON
Chiller	OFF	OFF

Tank Trends | IcetecOverview | HX Trends: | Flows | Tons | 2100 | 2200 | 2300 | 2400



Energy usage management

- Real time spot prices

Green line is real-time (5 minute) electricity spot prices

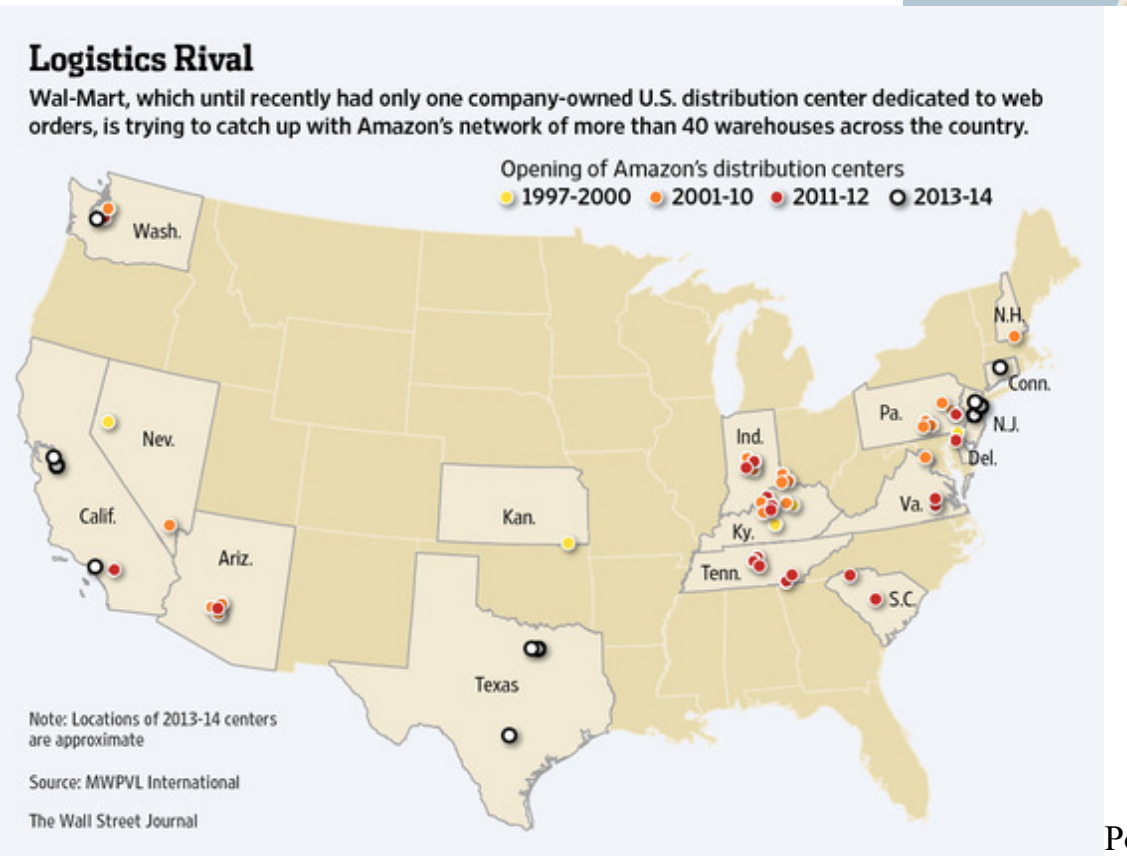
Planning cash reserves

- How much money should we hold in cash given variable market returns and interest rates to meet the needs of a business?

Stock prices

Bonds

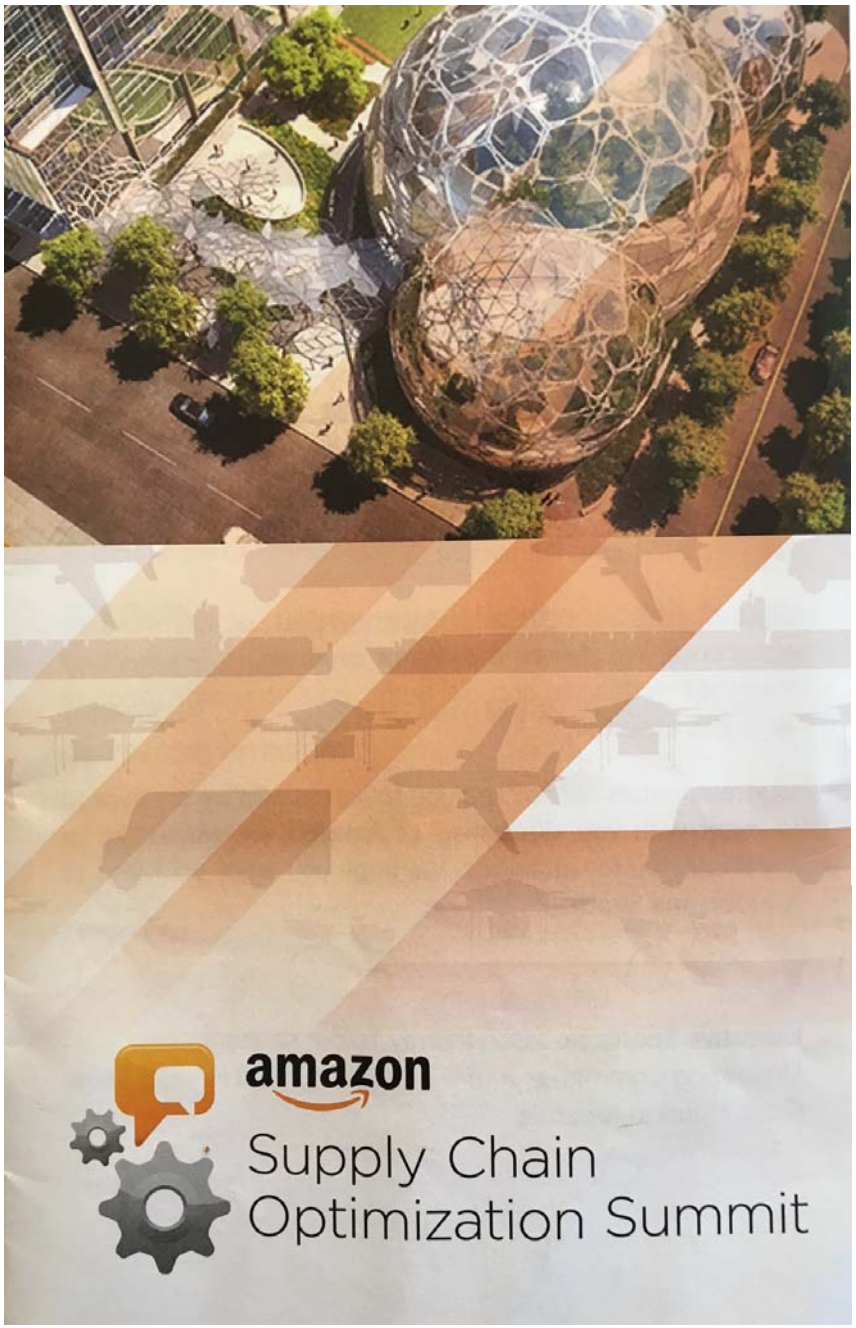
- Amazon distribution network
  - » Small number of DC's
  - » Uses UPS, FedEx, for final distribution



- Walmart distribution network
  - » Extensive retail network
  - » Limited need for distribution centers
  - » Increasing role of internet sales

Amazon

- First supply chain logistics conference in 2015
- Still running off of spreadsheets developed by Jeff Bezos
- Numerous instances where uncertainty is an important element.
- Exclusively using deterministic optimization tools at that time.



Supply chain management

- Amazon-UPS meltdown, Christmas, 2013
  - » Amazon promised 2-day service (based on UPS service guarantees). UPS was overwhelmed.

NEW REPUBLIC

amazon.com

DECEMBER 26, 2013

If the Private Sector Is So Great, Why Did UPS Botch Christmas?

A corrective for market triumphalists

By Alec MacGillis @AlecMacGillis

Photo: Lionel Bonaventure/AFP/Getty

At the heart of the great big pile-on of ridicule for the flawed healthcare.gov rollout the past few months was a large helping of private-sector triumphalism. Just imagine, the chorus went, if tech giants like Amazon or Google had been in charge of the Web site instead of those clueless, fusty bureaucrats – first, the problems would not have

UPS Shipping Delays Show Perils of Stores Overpromising

Mary Schlangenstein, Leslie Patton and Alex Barinka
December 26, 2013 — 5:19 PM EST

(Corrects 22nd paragraph to show UPS did make most of its deliveries.)

Dec. 26 (Bloomberg) -- The failure of United Parcel Service Inc. and FedEx Corp. to deliver packages in time for Christmas has exposed the perils of retailers promising to get last-minute gifts to customers.

Chains from Kohl's Corp. to Amazon.com Inc. to 1-800-Flowers.com Inc. offered gift cards and refunds after angry shoppers took to social media to vent their frustrations at the missed shipments. On its website, UPS said the volume of last-minute air packages exceeded its capacity to process them.

That's Logistics: Without Tim Cook, iPad would cost $5,000

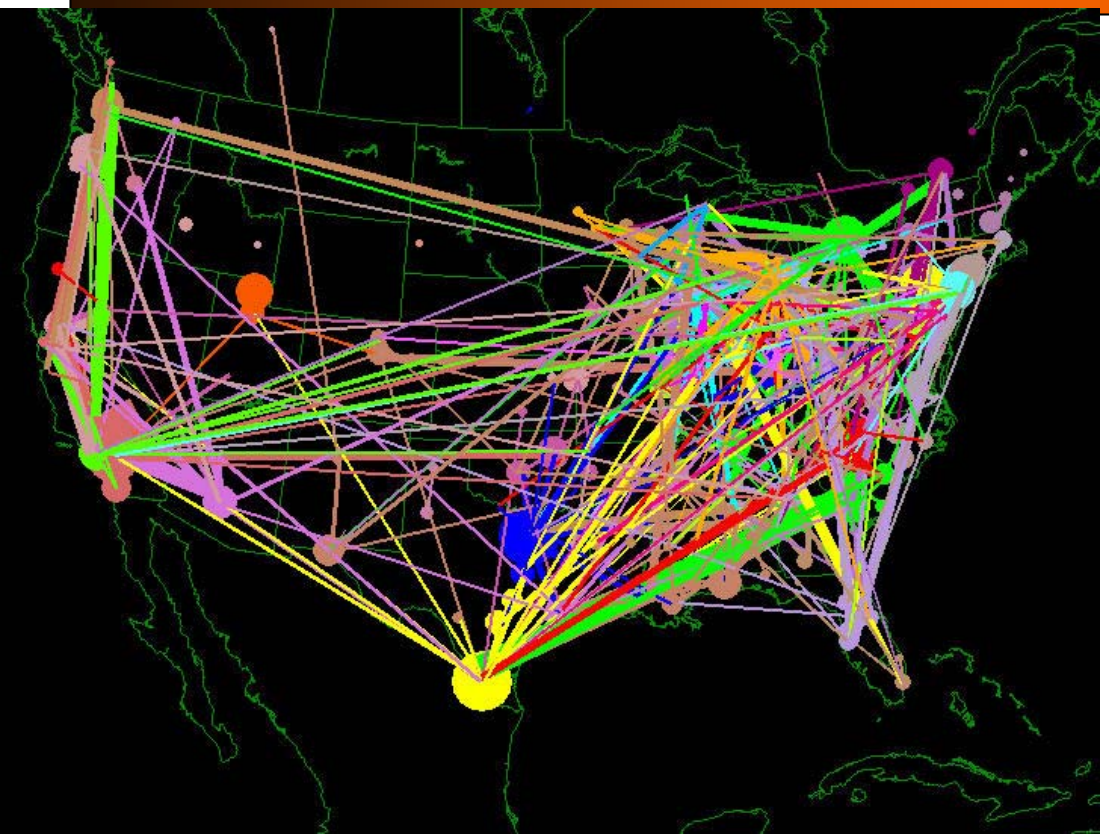

PHOTO: CHRIS HONDROS/GETTY IMAGES

Before becoming the CEO of Apple last year, Tim Cook was known as the "logistics king" of the company. MacTrast.com pointed to an article on Business Insider that may demonstrate just how much: *"If it weren't for Tim Cook, the iPad would cost $5,000"*.

Global logistics

- Apple, Inc.

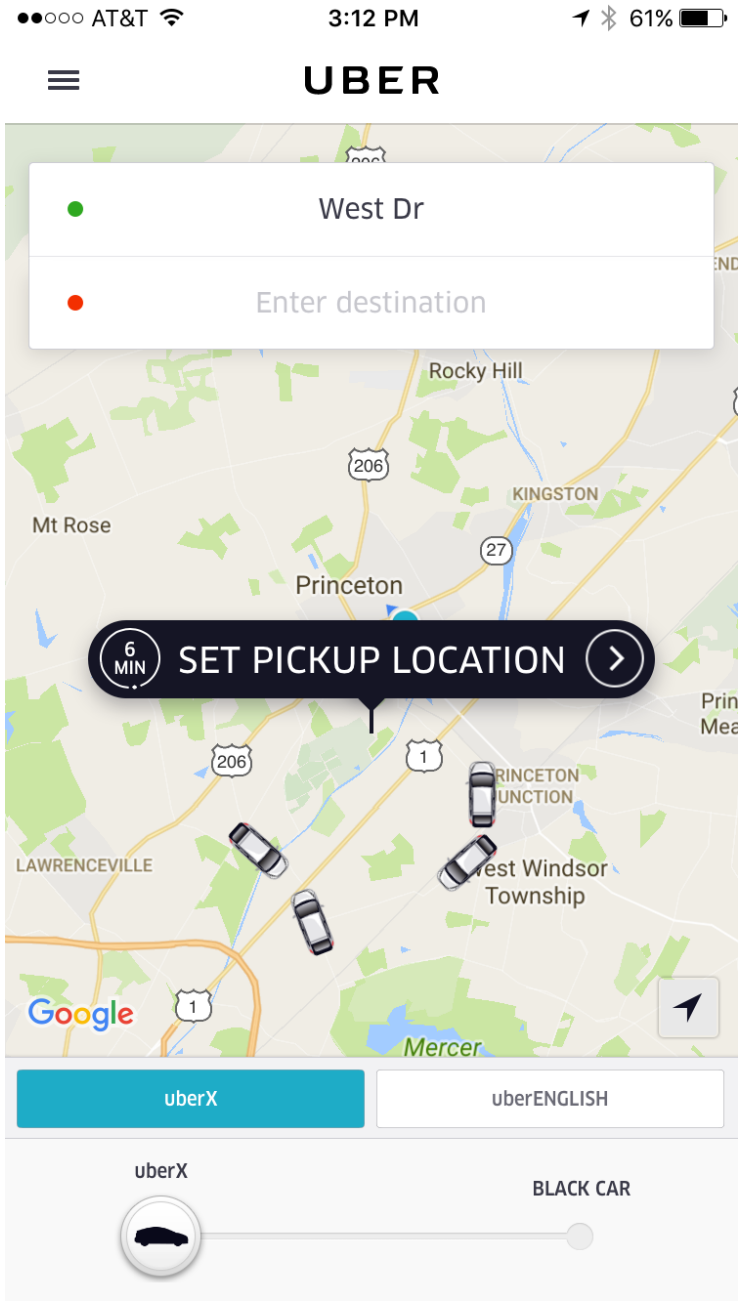
Fleet management



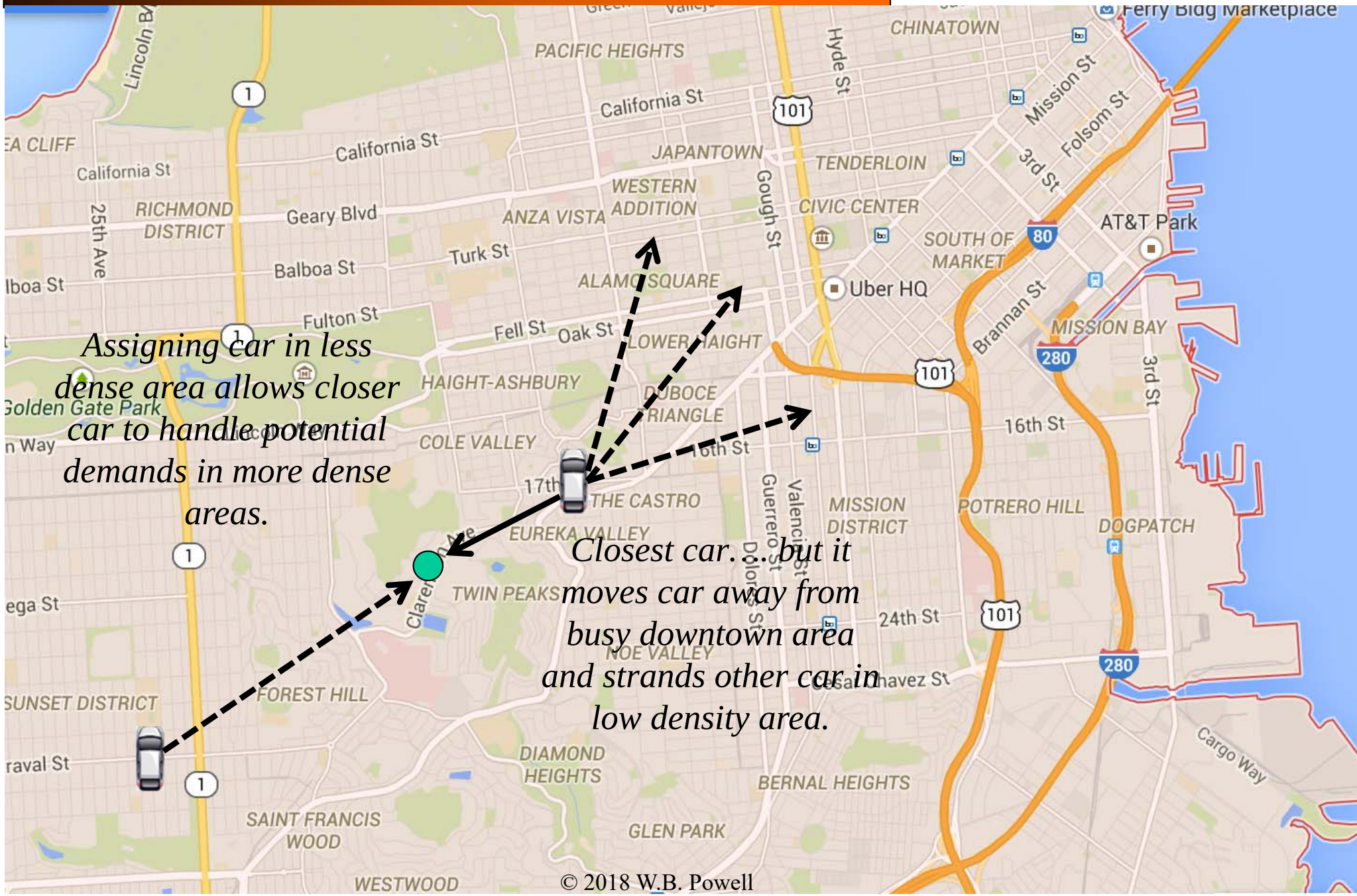
- Fleet management problem
  - » Optimize the assignment of drivers to loads over time.
  - » Tremendous uncertainty in loads being called in

Real-time logistics

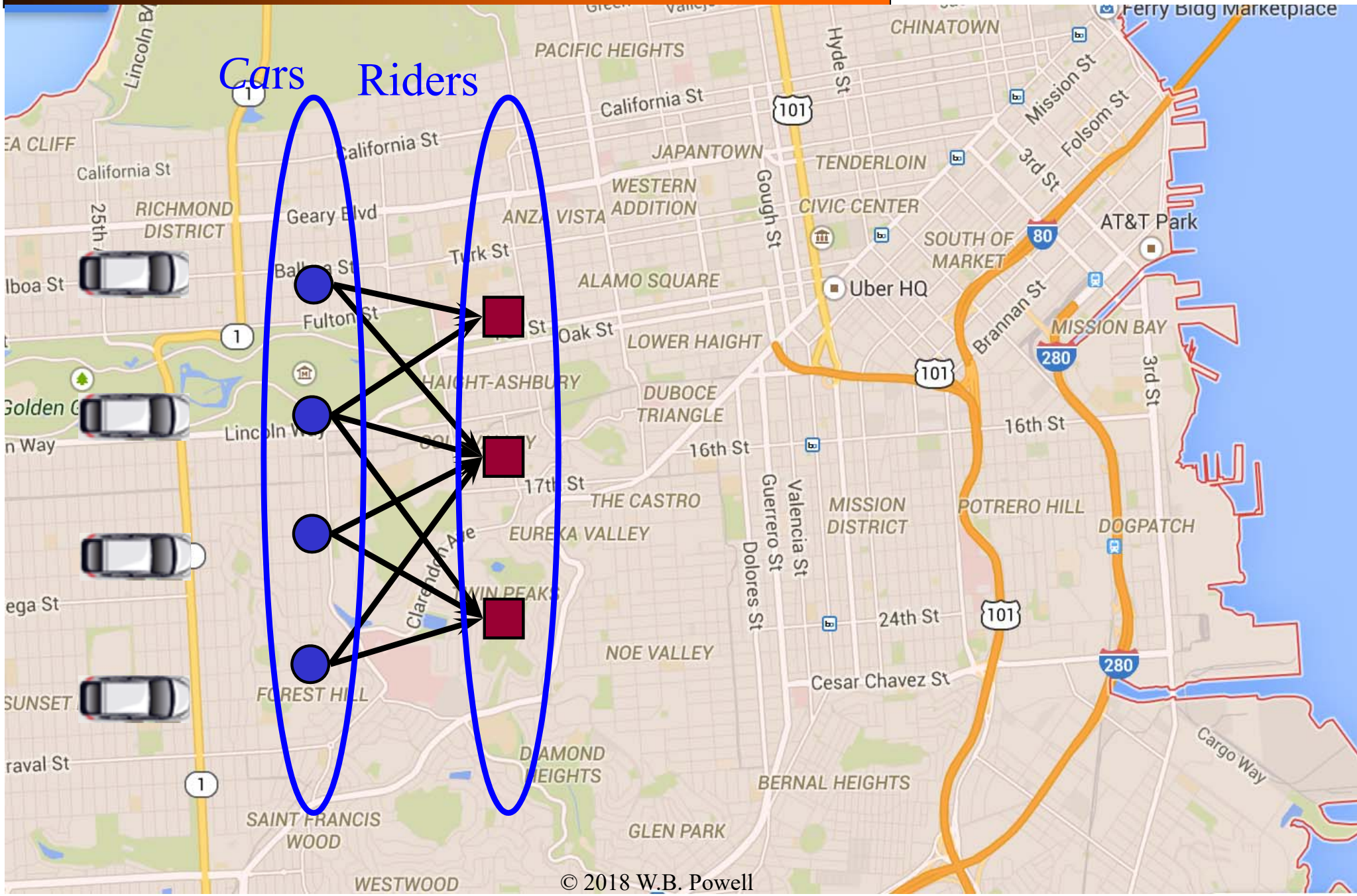
- Uber
  - » Provides real-time, on-demand transportation.
  - » Drivers are encouraged to enter or leave the system using pricing signals and informational guidance.
- Decisions:
  - » How to price to get the right balance of drivers relative to customers.
  - » Assigning and routing drivers to manage Uber-created congestion.
  - » Real-time management of drivers.
  - » Pricing (trips, new services, …)
  - » Policies (rules for managing drivers, customers, …)



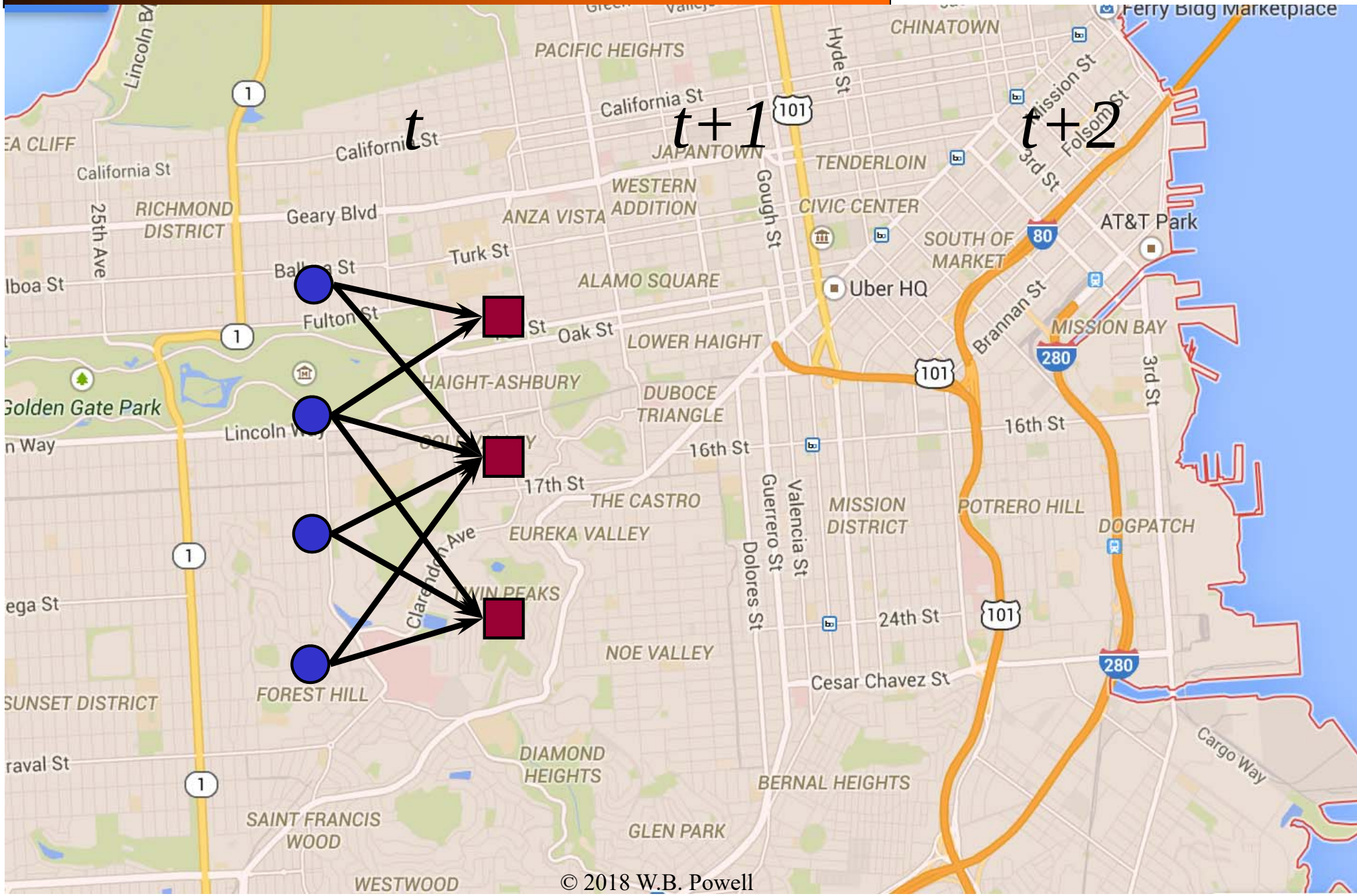
Effect of Current Decision on the Future



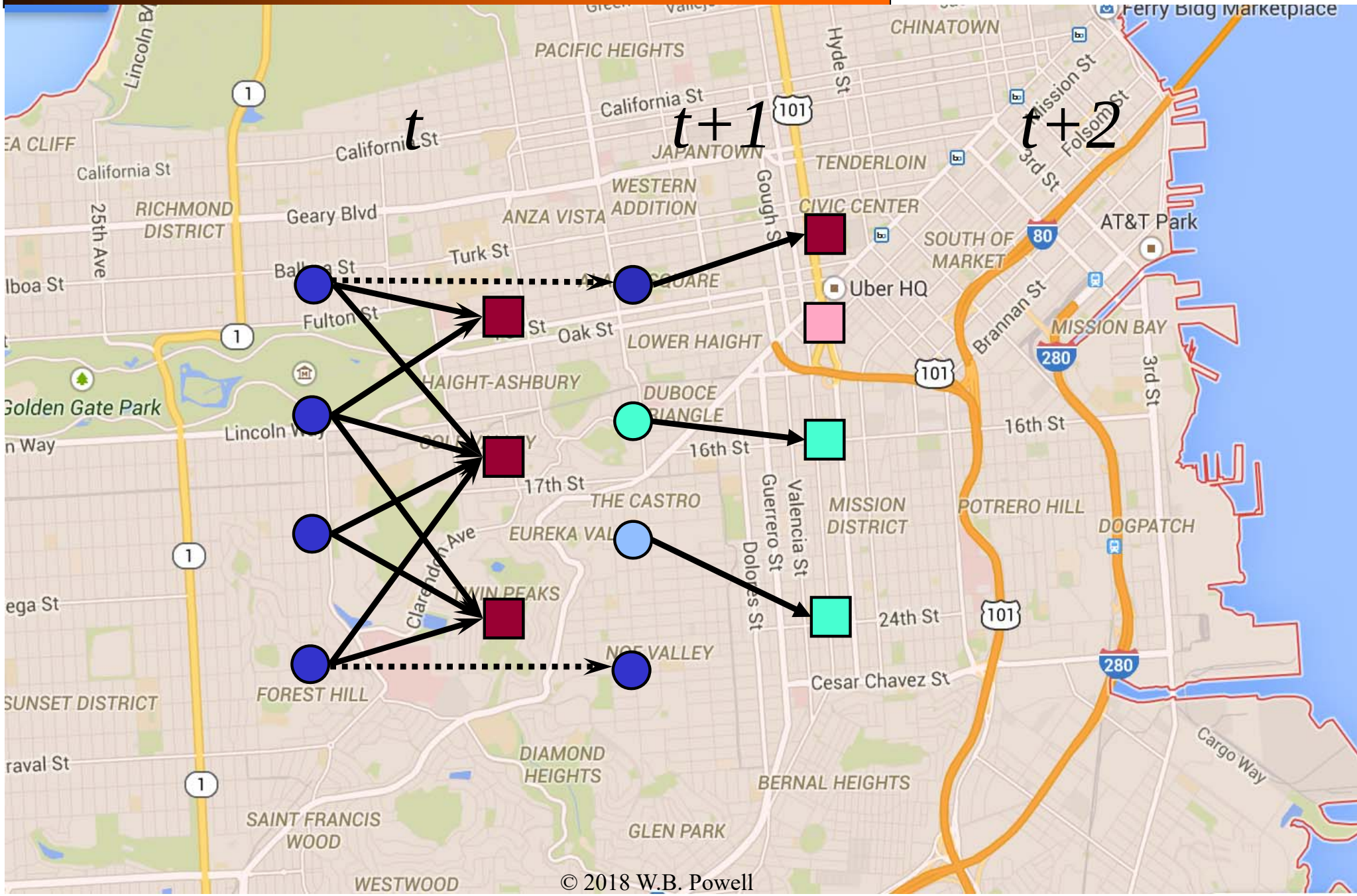
Cost function approximations



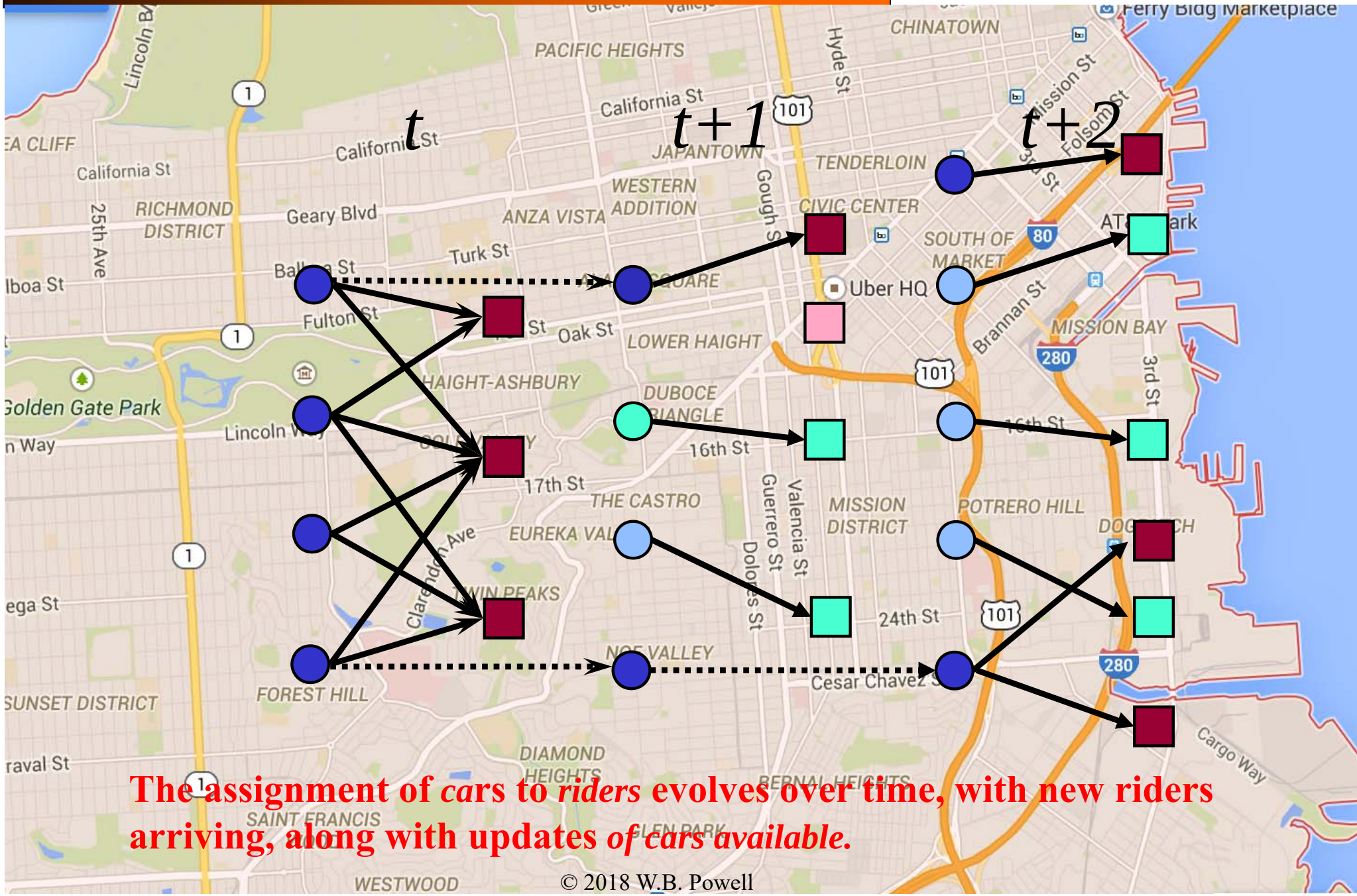
Optimizing over time



Optimizing over time



Optimizing over time



The assignment of *cars* to *riders* evolves over time, with new riders arriving, along with updates *of cars available.*

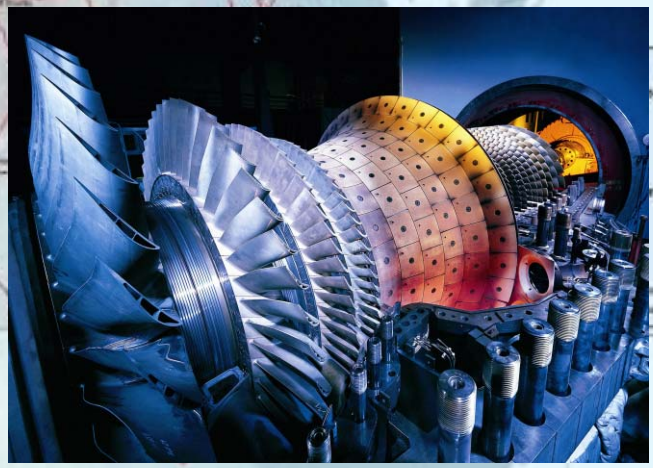
Matching buyers with sellers

- Now we have a logistic curve for each origin-destination pair (i,j)

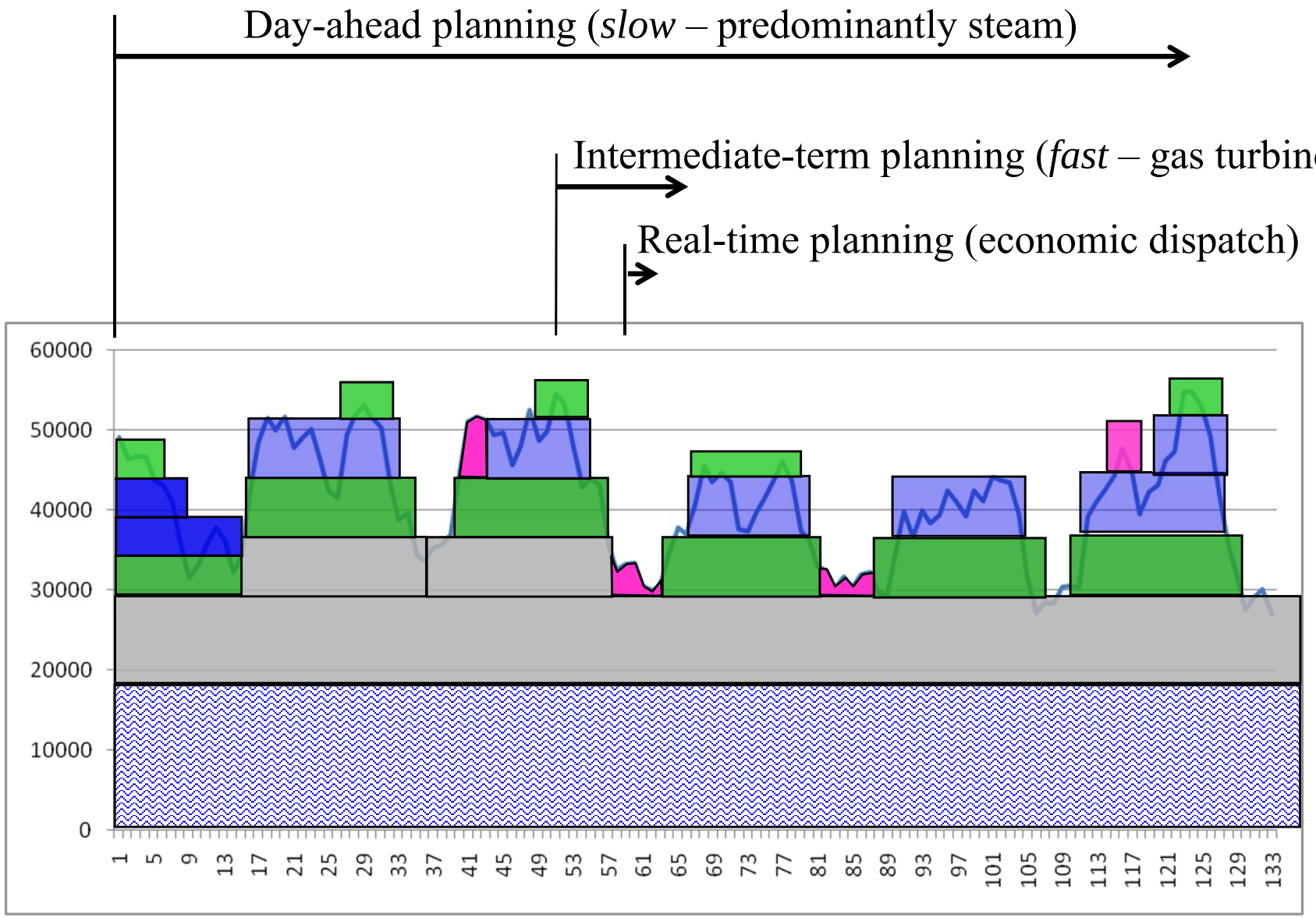
$$P^{Y}(p,a\,|\,\theta)=\frac{e^{\theta_{ij}^{0}+\theta_{ij}p+\theta_{ij}^{a}a}}{1+e^{\theta_{ij}^{0}+\theta_{ij}p+\theta_{ij}^{a}a}}$$

- Number of offers for each (i,j) pair is relatively small.
- Need to generalize the learning across hundreds to thousands of markets.

An energy generation portfolio

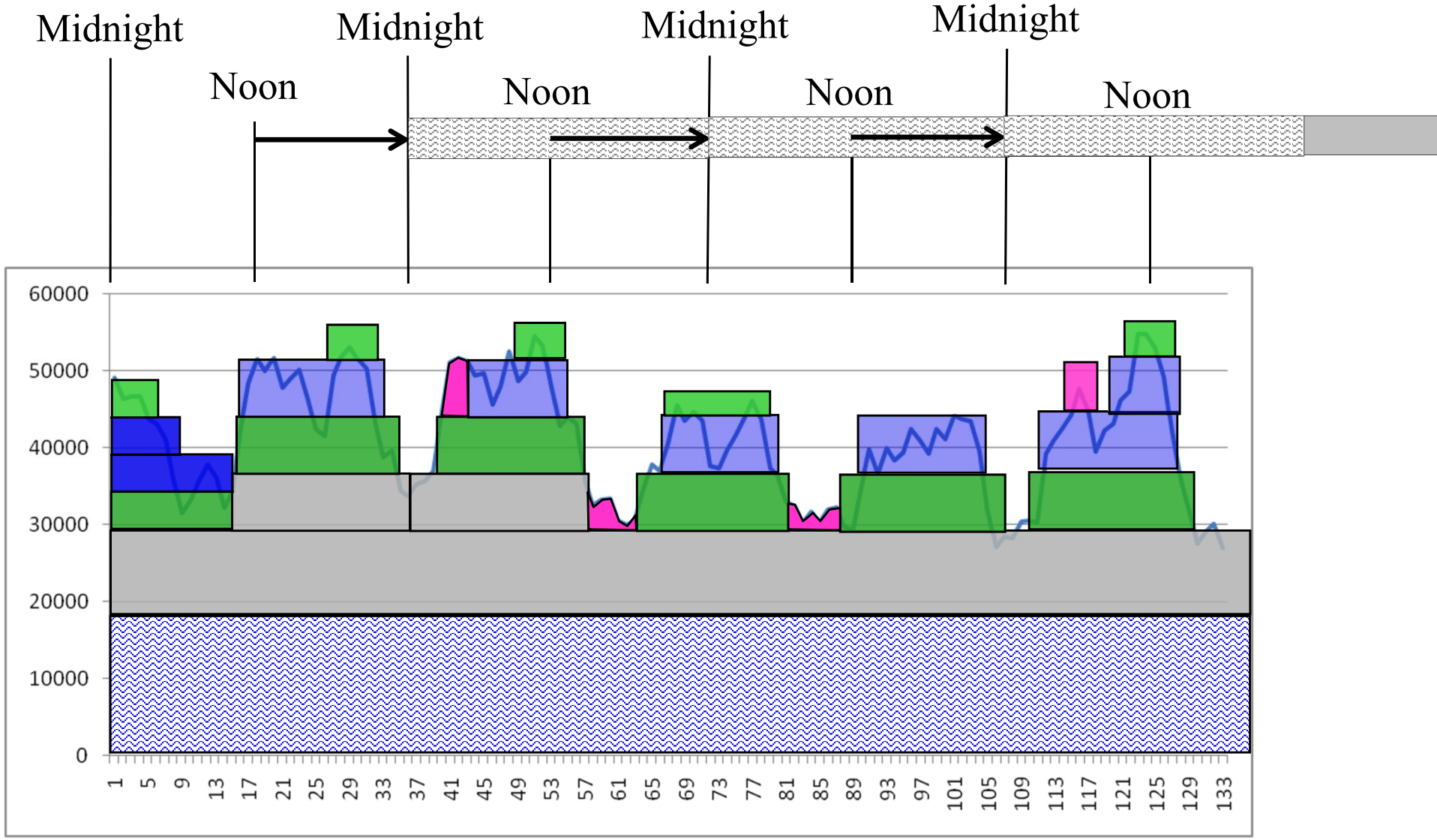


The PJM planning process



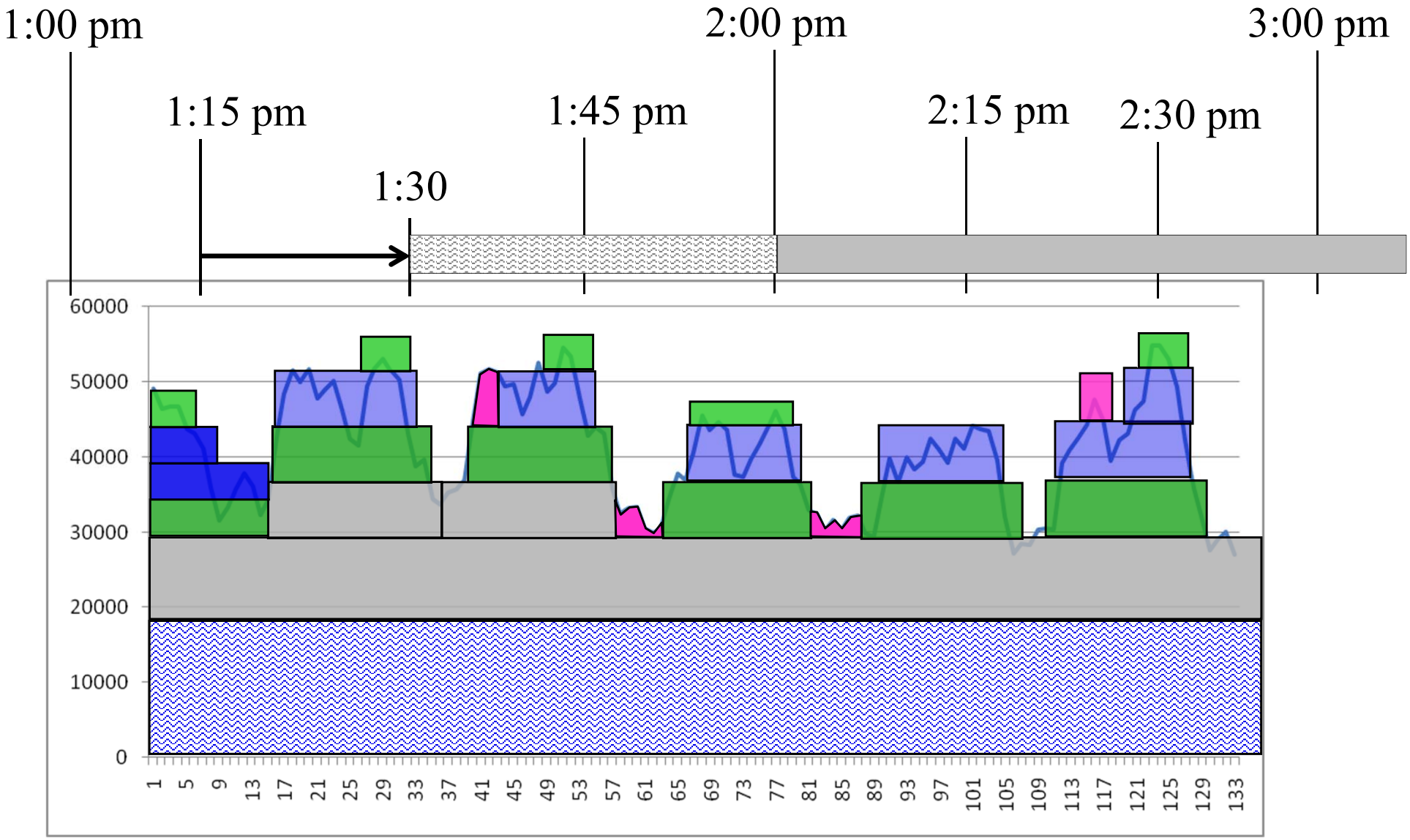
The PJM planning process

- The day-ahead unit commitment problem



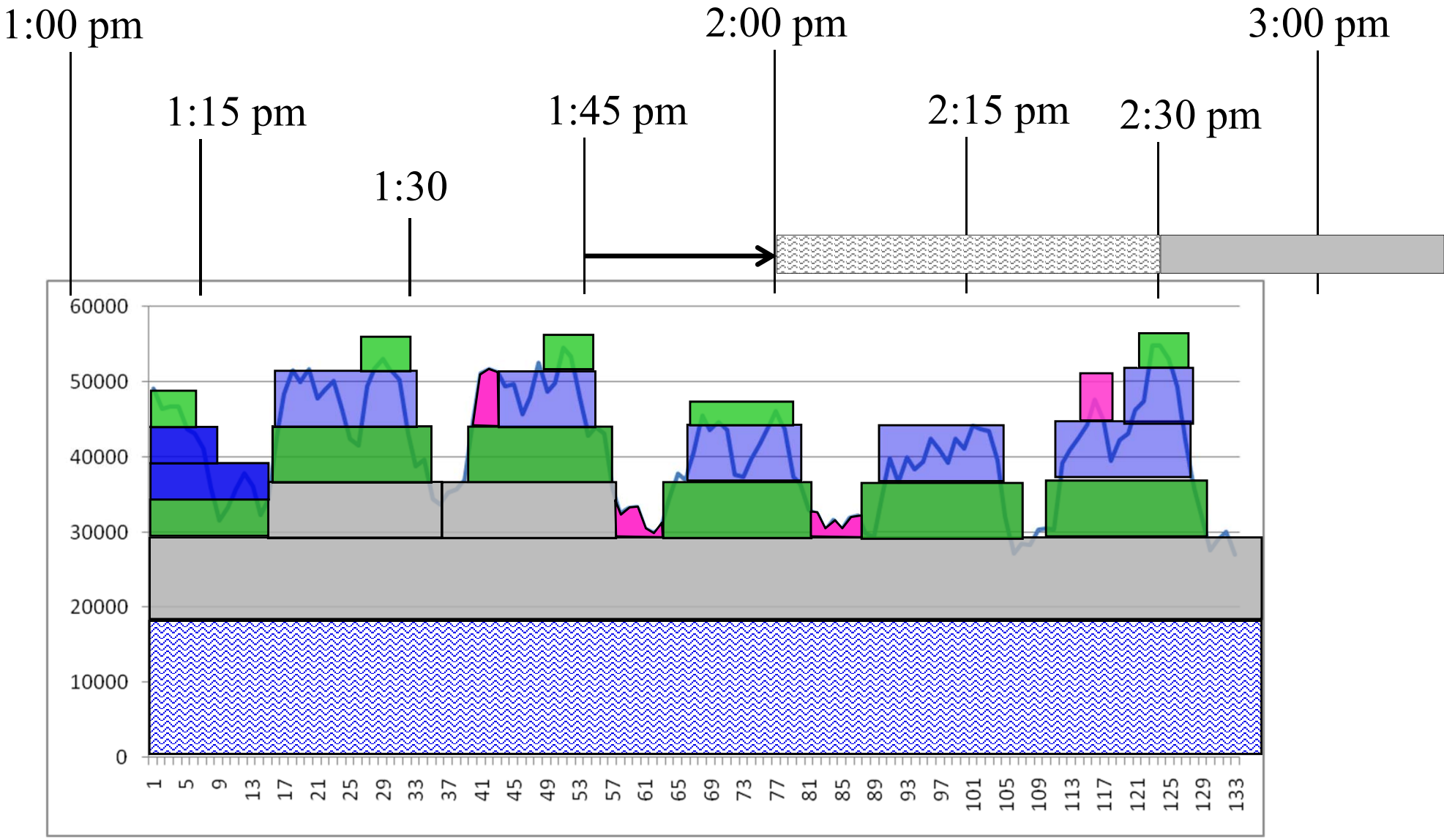
The PJM planning process

- Intermediate-term unit commitment problem



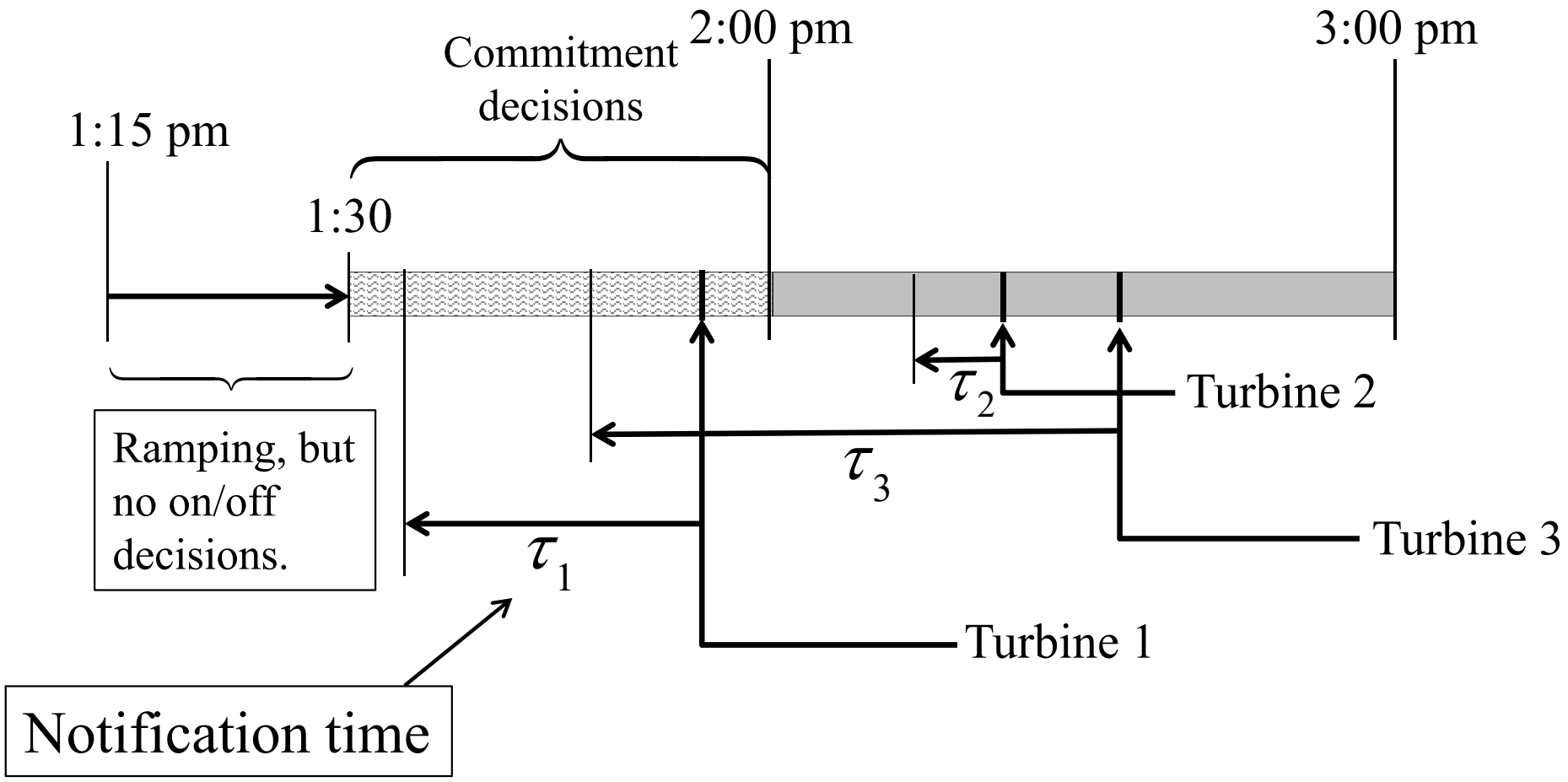
The PJM planning process

- Intermediate-term unit commitment problem



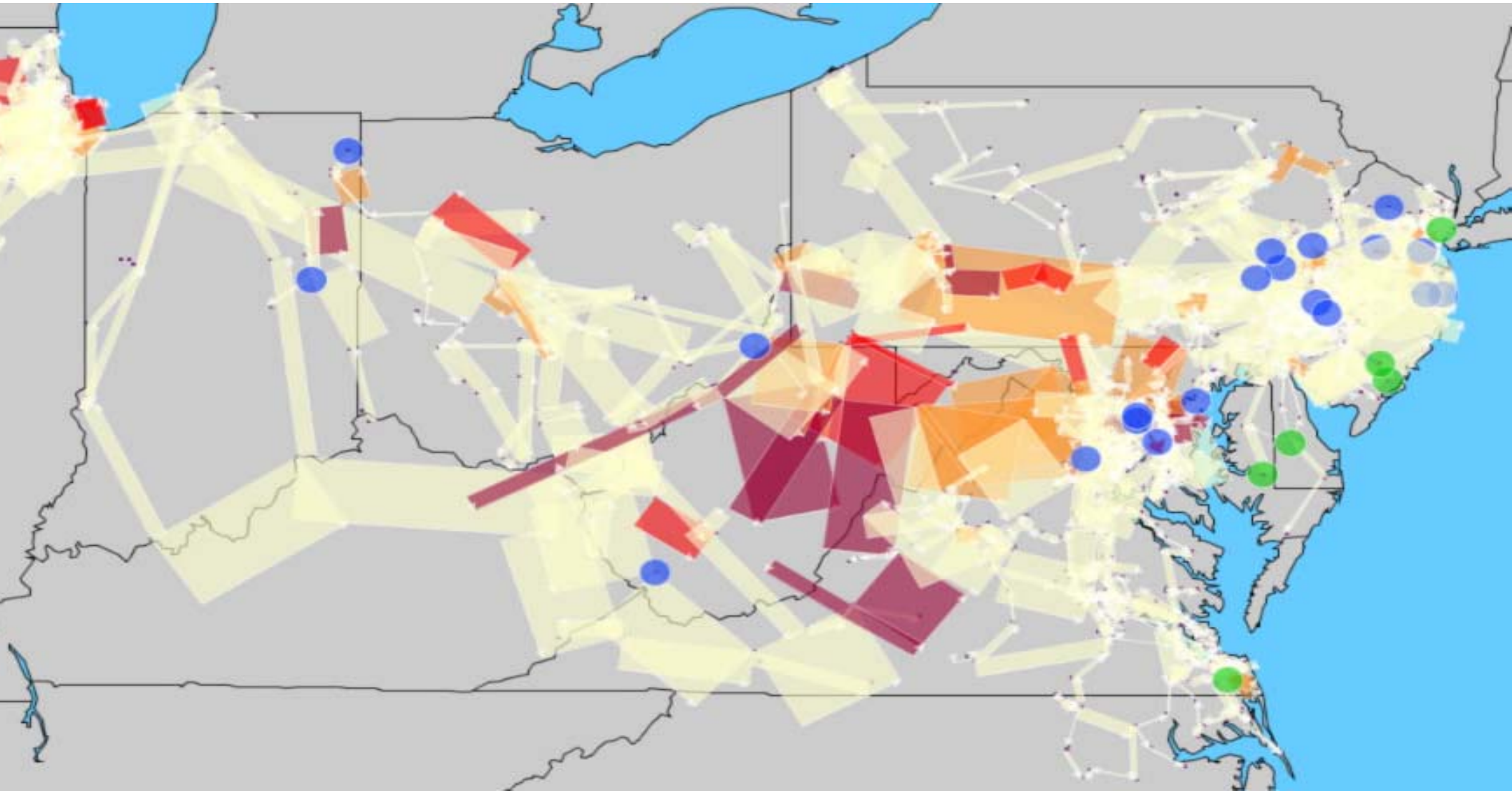
The PJM planning process

- Intermediate-term unit commitment problem



Electricity

- The PJM power grid



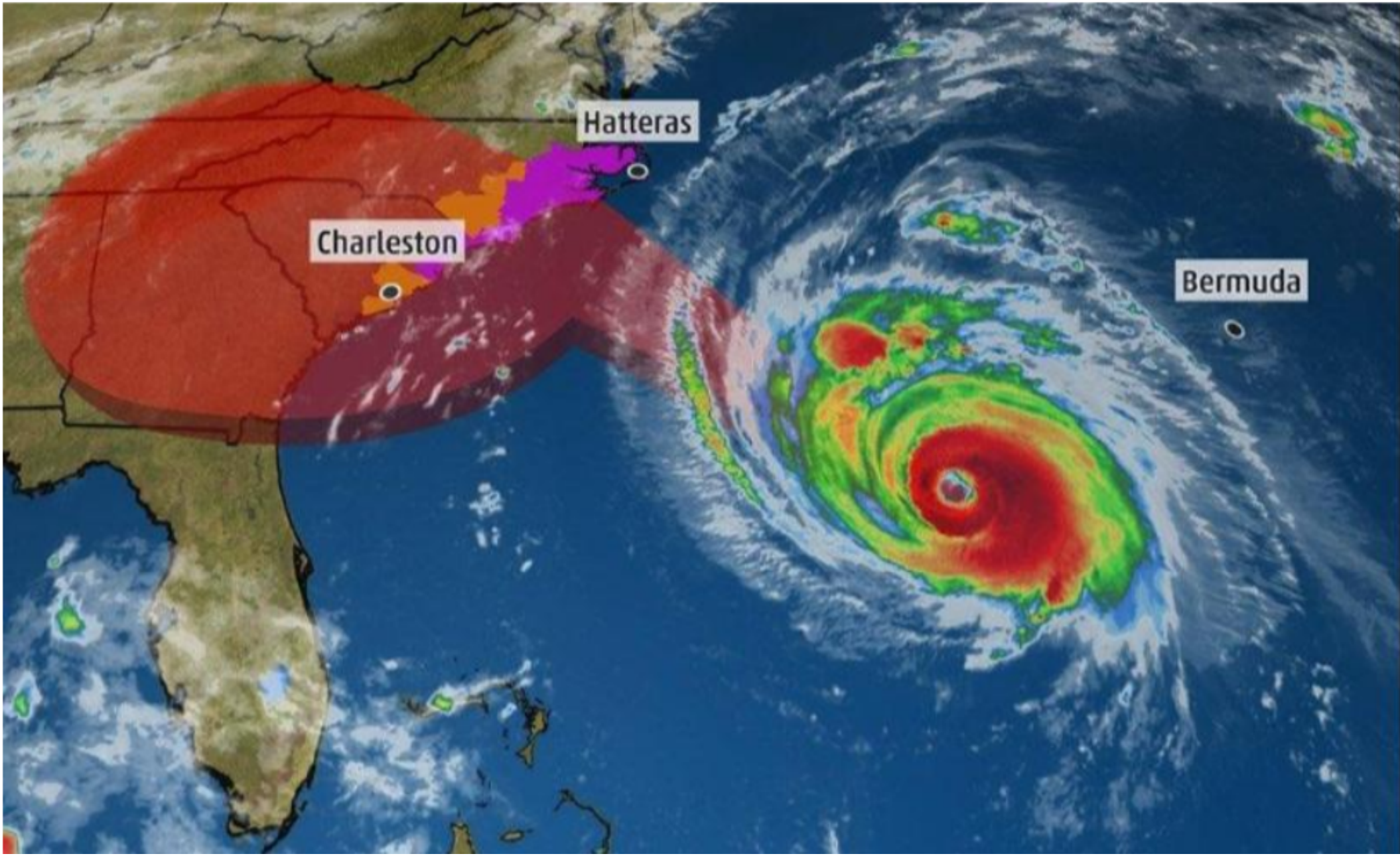
Emergency storm response

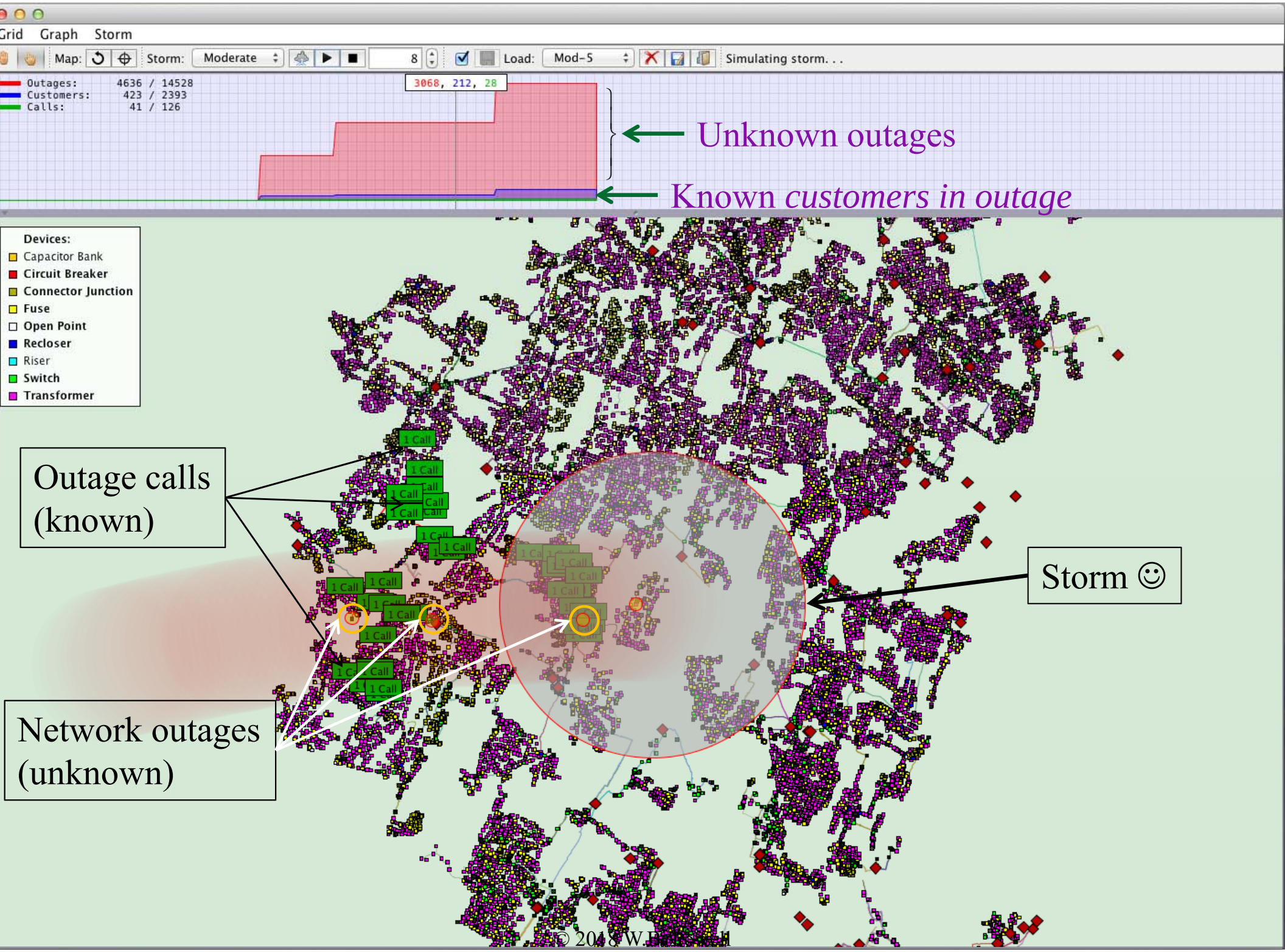
- Hurricane Sandy
  - » Once in 100 years?
  - » Rare convergence of events
  - » But, meteorologists did an amazing job of forecasting the storm.

- The power grid
  - » Loss of power creates cascading failures (lack of fuel, inability to pump water)
  - » How to plan?
  - » How to react?

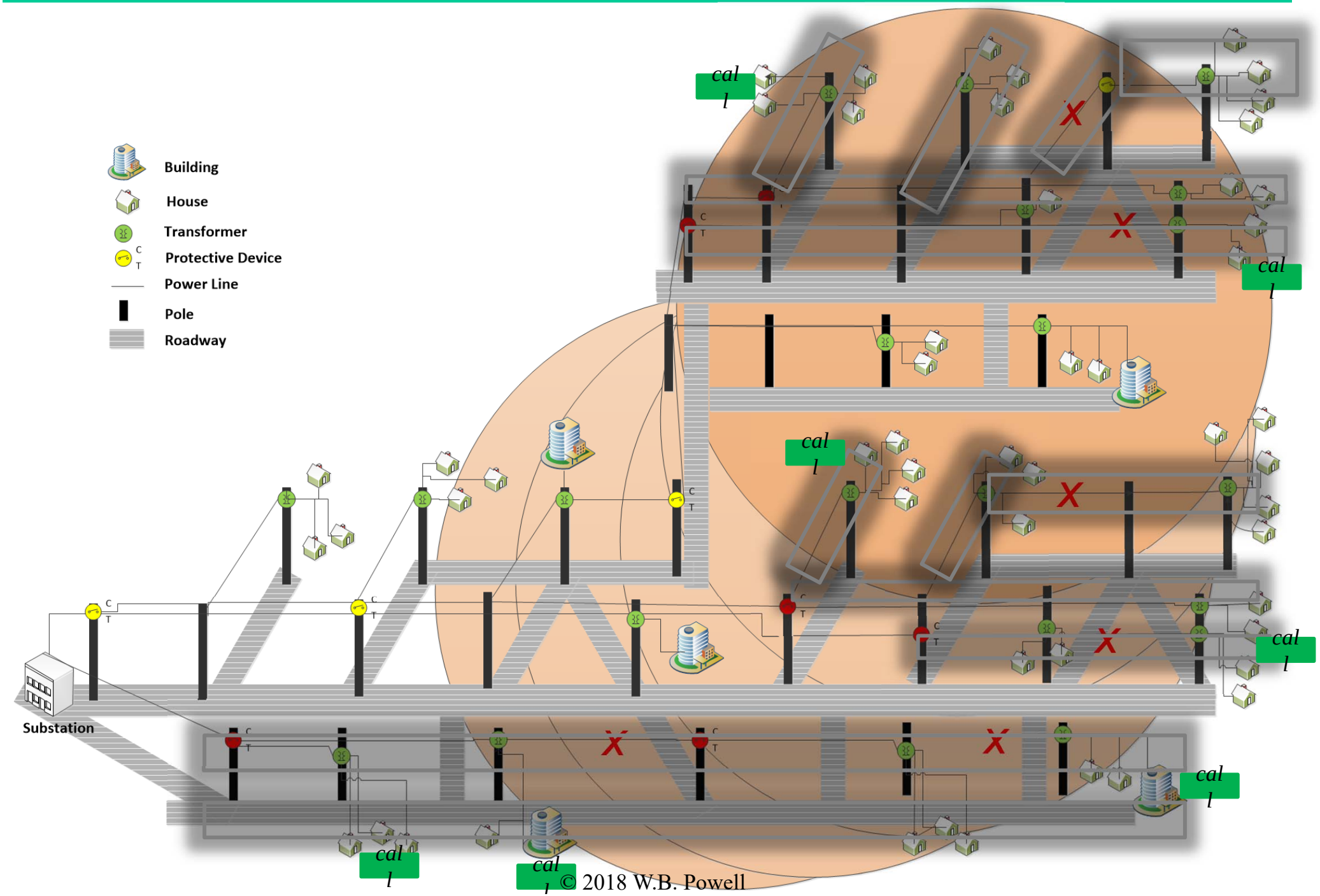
Hurricane Florence Targets Carolinas, Appalachians With Potentially Catastrophic Flooding, Destructive Winds; Hurricane Warning Issued

By weather.com meteorologists · 10 hours ago · weather.com

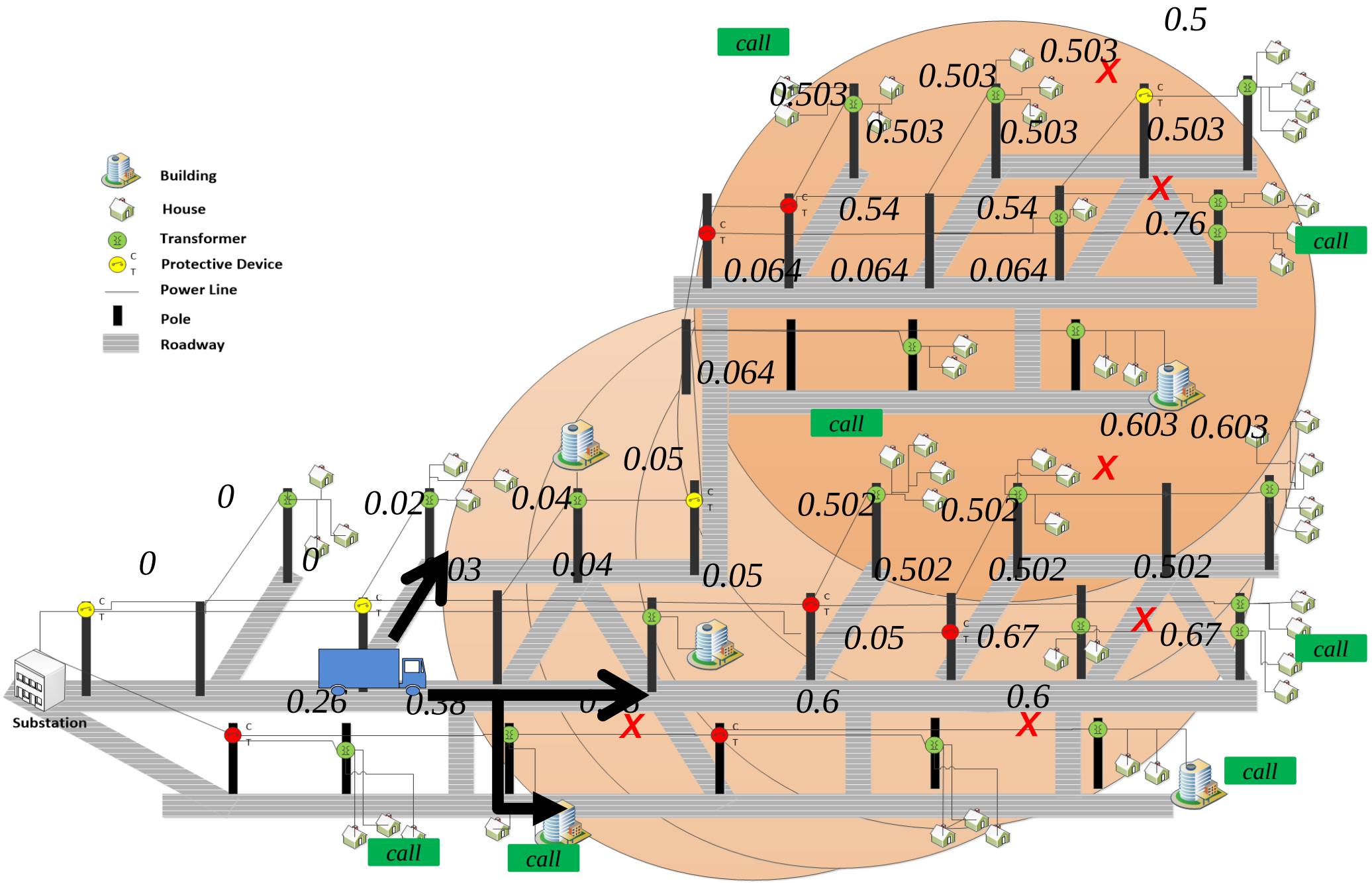



Grid
Graph
Storm
Map:
Storm:
Moderate
8
Load:
Mod-5
Simulating storm. . .
Outages: 4636 / 14528
Customers: 423 / 2393
Calls: 41 / 126
3068, 212, 28
Unknown outages
Known customers in outage
Devices:
Capacitor Bank
Circuit Breaker
Connector Junction
Fuse
Open Point
Recloser
Riser
Switch
Transformer
Outage calls (known)
Network outages (unknown)
Storm ☺
1 Call

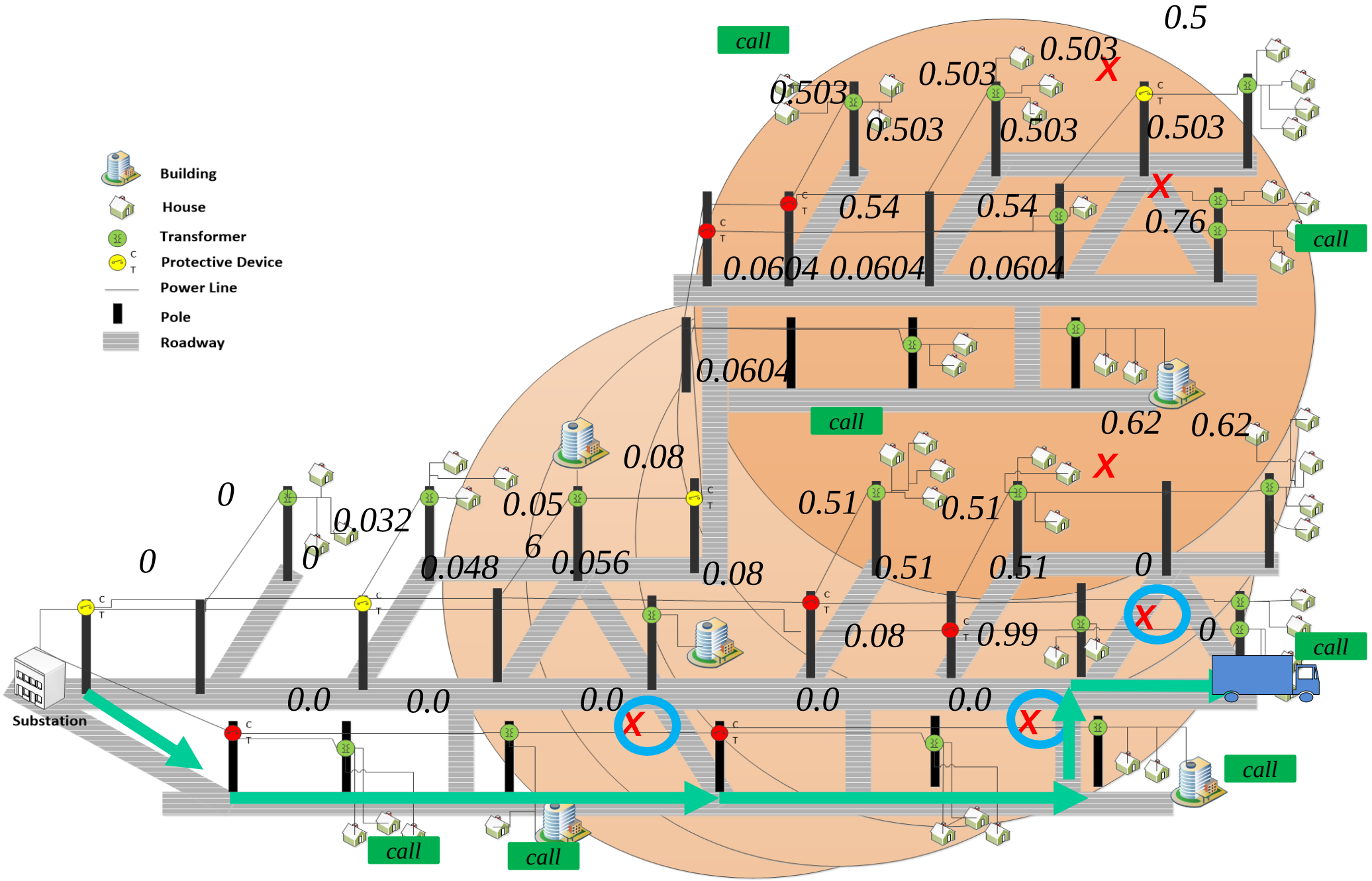
Problem Description - Emergency Storm Response



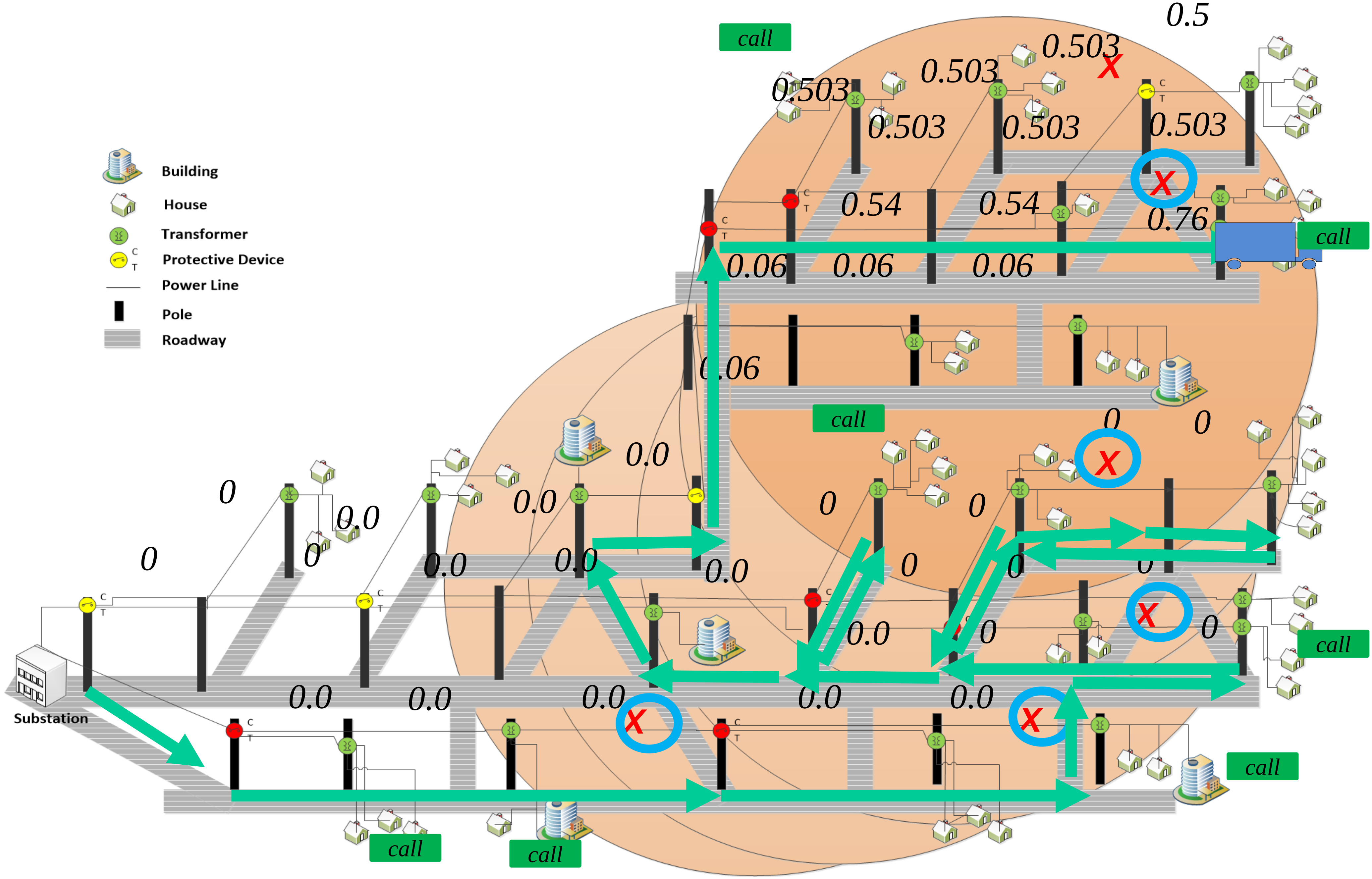
Emergency storm response



Emergency storm response



Emergency storm response



Modeling

- “To do good modeling, all you have to do is listen.”

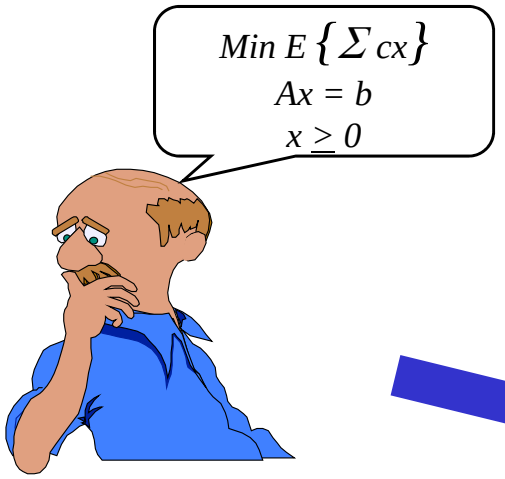
Click here for decisions.

Carrie Hillebrand	X	when to book a flight pay the least	Another rich problem - how to do learning, how is the optimal solution affected by exogenous information....	18	1	1	1	1
Khyati Agrawal		Determine if the intern programs at top paying firms (Google, Microsoft, Facebook) are a good return on investment, and how much money should they spend on their interns	Determining if the intern program is a good return is a metric, not a decision. Is the decision how much to spend? Are there any decisions affecting how the decision is run?	15	1	1	1	1
Hunter Johnson	X	How to invest effectively to profit off of Trump's twitter posts	Investing from twitter feeds is a nice problem. I supervised a senior thesis on this which might be a good starting point.	13	1	1	1	1
Sarah Cheng	x	In a restaurant / hotel with a reservation system, how long to wait before giving table / room to a current customer if the customer with the reservation doesn't show up	I might tweak this to more of an overbooking policy, similar to what airlines use. Or, simply how many tables to allocate to reservations vs. walk-ins.	9	1	1	1	1
Millian Gehrer	X	which medical treatment to give a patient	This is a great learning problem.	9	1	1	1	1
Gregory Kernisan	x	Which keywords you should pick in order to boost traffic to a website (Search engine optimization, or SEO)	Great problem - real challenge here is coming up with the belief model.	9	1	1	1	1
Daniel Girshik	X	If owning a home, when is the optimal point to refinance	problem of pricing generators. To find the price, you have to solve the problem of when to sell.	7	1	1	1	
Selina Pi	X	How to incentivize customer service reps to optimize customer satisfaction and ultimately revenue (i.e. comparing bonuses tied to customer satisfaction, bonuses tied to quality of interactions monitored by a manager, or no bonus)	Great context! These are nice learning problems.	7	1	1	1	
		In Venture Capital, which companies and in which industry	Good problem - hard part is modeling the belief about the company. Some VCs will even imbed people within a company to learn more about the management team (this would also be a					

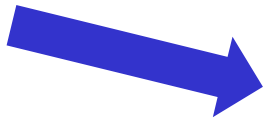
Modeling frameworks

Modeling stochastic, dynamic problems

- Before we can *solve* complex problems, we have to know how to *think* about them.



Mathematician



Software

- The biggest challenge when making decisions under uncertainty is *modeling*.


Stochastic programming
Robust optimization
Decision analysis
Approximate dynamic programming
Simulation optimization
Optimal learning
Model predictive control
Dynamic Programming and control
Stochastic search
Bandit problems
Optimal control
Online computation
Reinforcement learning
Markov decision processes
Stochastic control
Simulation optimization


John R. Birge
François Louveaux
Introduction to Stochastic Programming
Second Edition
Michael C. Fu Editor
Handbook of Simulation Optimization
Princeton Series in Applied Mathematics
Robust Optimization
Introduction to Decision Analysis
A Practitioner's Guide to Improving Decision Quality
SECOND EDITION
Approximate Dynamic Programming
Solving the Curses of Dimensionality
Warren B. Powell
VOLUME 2 • 4th EDITION
Dynamic Programming and Optimal Control
APPROXIMATE DYNAMIC PROGRAMMING
Optimal Learning
SECOND EDITION
Model Predictive Control
OPTIMAL CONTROL
Dimitri P. Bertsekas
Athena Scientific
WILEY
INTRODUCTION TO STOCHASTIC SEARCH AND OPTIMIZATION
Estimation, Simulation, and Control
JAMES C. SPALL
MULTI-ARMED BANDIT ALLOCATION INDICES
SECOND EDITION
John Gittins, Kevin Glazebrook and Richard Weber
Reinforcement Learning
Introduction
Richard S. Sutton and Andrew G. Barto
Markov Decision Processes
Discrete Stochastic Dynamic Programming
MARTIN L. PUTERMAN
Vol. 1
Online Computation and Competitive Analysis
Allan Borodin
Ran El-Yaniv
STOCHASTIC SIMULATION OPTIMIZATION
An Optimal Computing Budget Allocation
Chun-Hung Chen • Loo Hay Lee
43
Jiongmin Yong
Xun Yu Zhou
Stochastic Controls
Hamiltonian Systems and HJB Equations
Modelling and Applied Probability

- Canonical modeling frameworks
  - » Statistics
    - Nonlinear model: $y = f(x|\theta)$
    - Linear model: $y = \theta_0 + \theta_1\phi_1(x) + \theta_2\phi_2(x) + \cdots$
  - » Linear programming
    - $\min_x cx$

    Subject to $Ax = b, x \geq 0$.
  - » Markov chains
    - $P(S^{n+1} = s'|S^n = s)$
  - » Sequential decision problems lack a standard canonical framework.

Modeling dynamic systems

- Almost all sequential decision problems can be modeled using five core components:
  - » State variables
    - What is in the state variable? What do we need to know at time t?
  - » Decision variables
    - What are our decisions?
  - » Exogenous information
    - What do we learn for the first time between t and t+1?
  - » Transition function
    - How does the problem evolve from t to t+1.
  - » Objective function
    - What are we minimizing and how?

- Board setup:
  - » List all five elements across upper two boards
  - » Then work through each element providing notation and description.
  - » Erase and do uncertainty modeling on left, then design policies on the right.
  - » For asset selling, start over with all five elements, and move organically through the problem.

Modeling dynamic problems

- The state variable:

Controls community

x_t = "Information state"

Operations research/MDP/Computer science

$S_t = (R_t, I_t, B_t)$ = System state, where:

R_t = Resource state (physical state)

Location/status of truck/train/plane

Energy in storage

I_t = Information state

Prices

Weather

B_t = Belief state ("state of knowledge")

Belief about traffic delays

Belief about the status of equipment

Modeling

There are five basic elements to an sequential decision problem. We encourage readers to flip to next chapter which illustrates this framework in the context of a simple asset selling problem in section 2 for a simple motivating application.

State variables - The state S_t of the system at time t is a function of history which, combined with a policy and exogenous information, contains all the information that is necessary and sufficient to model our system from time t onward. This means it has to capture the information needed to compute costs/contributions, constraints on decisions, and the evolution of this information over time in the transition function (these are the equations that describe how the state variables evolve), all defined below. In fact, it is often best to write out the objective function (with costs/contributions), the constraints for the decisions, and the transition function, and then return to define the state variables.

We distinguish between the initial state S_0 and the dynamic state S_t for $t > 0$. The initial state contains all deterministic parameters, initial values of dynamic parameters, and, in some settings, initial probabilistic beliefs about unknown parameters. The dynamic state S_t contains all information that is evolving over time. Note that the initial state S_0 may include some deterministic parameters that do not change over time, which means these would be excluded from the dynamic state variable.

State variables (cont'd)

There are three types of information in S_t:

- The physical state, R_t, which in most (but not all) applications is the state variable that is being controlled. R_t may be a scalar, or a vector with element R_{ti} where i could be a type of resource (e.g. a blood type) or the amount of inventory at location i.
- Other information, I_t, which is any information that is known deterministically not included in R_t. The information state often evolves exogenously, but may be controlled or at least influenced by decisions (e.g. selling a large number of shares may depress prices).
- The belief state B_t, which contains information about a probability distribution describing some unknown parameter. Thus, B_t could capture the mean and variance of a normal distribution. Alternatively, it could be a vector of probabilities that evolve over time.

We note that the distinction of "physical state" R_t versus "other information" I_t is somewhat imprecise, but there are many problems that involve the management of physical (and financial) resources where the state of the physical resources (inventories, location of a vehicle, location and speed of a robot) is a natural "state variable" (in fact, a common error is to assume that the state of physical resources

State variables (cont'd)

is the only state variables). By contrast, the "other information" I_t is literally any other information that does not seem to belong in R_t. For example, R_t might be the amount of money in a cash account, while I_t might be the current state of the stock and bond markets.

The state S_t is sometimes referred to as the *pre-decision state* because it is the state just before we make a decision. We sometimes find it useful to define a *post-decision state* S_t^x which is the state immediately after we make a decision, before any new information has arrived, which means that S_t^x is a deterministic function of S_t and x_t. For example, in a basic inventory problem where $R_{t+1} = \max\{0, R_t + x_t - \hat{D}_{t+1}\}$, the post-decision state would be $S_t^x = R_t^x = R_t + x_t$, which is the state after we have implemented our decision but before any new information (which would be the demands $\hat{D}_{t+1}$) arrive. Post-decision states are often simpler, because there may be information in S_t that is only needed to make the decision x_t (we will see this in chapter 5). However, there are other situations where we place an order $x_{tt'}$ at time t to be implemented at time $t' > t$. In the case of these lagged decisions, we would have to keep these decisions in the state variable at time $t + 1$.

Modeling dynamic problems

- Decisions:

Markov decision processes/Computer science

a_t = Discrete action

Control theory

u_t = Low-dimensional continuous vector

Operations research

x_t = Usually a discrete or continuous but high-dimensional vector of decisions.

At this point, we do not specify *how* to make a decision. Instead, we define the function $X^\pi(s)$ (or $A^\pi(s)$ or $U^\pi(s)$), where π specifies the type of policy. "π" carries information about the type of function f, and any tunable parameters $\theta \in \Theta^f$.

Decision variables - Decisions are typically represented as a_t for discrete actions, u_t for continuous (typically vector-valued) controls, and x_t for general continuous or discrete vectors. We use x_t as our default, but there are situations where a_t is useful when actions are discrete.

Decisions may be binary (e.g. for a stopping problem), discrete (e.g. an element of a finite set), continuous (scalar or vector), integer vectors, and categorical (e.g. the attributes of a patient). We note that entire fields of research are sometimes distinguished by the nature of the decision variable.

We assume that decisions are made with a policy, which we might denote $X^\pi(S_t)$ (if we use x_t as our decision), $A^\pi(S_t)$ (if we use a_t), or $U^\pi(S_t)$ (if we use u_t). We assume that a decision $x_t = X^\pi(S_t)$ is feasible at time t. We let "π" carry the information about the type of function $f \in \mathcal{F}$ (for example, a linear model with specific explanatory variables, or a particular nonlinear model), and any tunable parameters $\theta \in \Theta^f$.

The decision variables

- Styles of decisions
  - » Binary
    $$x \in X = \{0,1\}$$
  - » Finite
    $$x \in X = \{1,2,...,M\}$$
  - » Continuous scalar
    $$x \in X = [a,b]$$
  - » Continuous vector
    $$x = (x_1,...,x_K), \quad x_k \in \mathbb{R}$$
  - » Discrete vector
    $$x = (x_1,...,x_K), \quad x_k \in \mathbb{Z}$$
  - » Categorical
    $$x = (a_1,...,a_I), \quad a_i \text{ is a category (e.g. red/green/blue)}$$

Modeling dynamic problems

- Exogenous information:

W_t = New information that first became known at time t

$= \left(\hat{R}_t, \hat{D}_t, \hat{p}_t, \hat{E}_t\right)$

$\hat{R}_t$ = Equipment failures, delays, new arrivals
New drivers being hired to the network

$\hat{D}_t$ = New customer demands

$\hat{p}_t$ = Changes in prices

$\hat{E}_t$ = Information about the environment (temperature, ...)

Note: Any variable indexed by t is known at time t. This convention, which is not standard in control theory, dramatically simplifies the modeling of information.

Below, we let ω represent a sequence of actual observations $W_1, W_2, \ldots$.

$W_t(\omega)$ refers to a sample realization of the random variable W_t.

Exogenous information - We let W_t be any new information that first becomes known at time t (that is, between $t-1$ and t). When modeling specific variables, we use "hats" to indicate exogenous information. Thus, $\hat{D}_t$ could be the demand that arose between $t-1$ and t, or we could let $\hat{p}_t$ be the change in the price between $t-1$ and t.

The exogenous information process may be stationary or nonstationary, purely exogenous or state (and possibly action) dependent. We let ω represent a sample path $W_1, \ldots, W_T$, where $\omega \in \Omega$ represents one sample path of the exogenous information. Often, we will create a set Ω with a set of discrete samples that we might represent using $(\omega_1, \omega_2, \ldots, \Omega_N)$.

Modeling dynamic problems

- The transition function

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

$$R_{t+1} = R_t + x_t + \hat{R}_{t+1} \quad \text{Inventories}$$

$$p_{t+1} = p_t + \hat{p}_{t+1} \quad \text{Spot prices}$$

$$D_{t+1} = D_t + \hat{D}_{t+1} \quad \text{Market demands}$$

Also known as the:

"System model"	"Transfer function"
"State transition model"	"Transformation function"
"Plant model"	"Law of motion"
"Plant equation"	"Model"
"State equation"	"transition function"

For many applications, these equations are unknown. This is known as "model-free" dynamic programming

Transition function - We denote the transition function by

$$S_{t+1} = S^M(S_t, x_t, W_{t+1}), \tag{1.1}$$

where $S^M(\cdot)$ is also known by names such as system model, plant model, plant equation and transfer function. Equation (1.1) is the classical form of a transition function which gives the equations from the pre-decision state S_t to pre-decision state S_{t+1}. We can also break down these equations into two steps: pre-decision to post-decision S_t^x, and then the post-decision S_t^x to the next pre-decision S_{t+1}. The transition function may be a known set of equations, or unknown, such as when we describe human behavior or the evolution of CO2 in the atmosphere. When the equations are unknown the problem is often described as "model free" or "data driven."

Transition functions may be linear, continuous nonlinear or step functions. When the state S_t includes a belief state B_t, then the transition function has to include the frequentist or Bayesian updating equations.

Given a policy $X^\pi(S_t)$, an exogenous process W_t and a transition function, we can write our sequence of states, decisions, and information as

$$(S_0, x_0, S_0^x, W_1, S_1, x_1, S_1^x, W_2, \ldots, x_{T-1}, S_{T-1}^x, W_T, S_T).$$

Below we use time t when modeling in the context of a process that is evolving over time. We also use t as our default index when there is no context. However, we use iteration n when there is some other process such as running experiments or performing simulations. If we use n, then we will write states, decisions and information as S^n, x^n and W^{n+1}.

Modeling stochastic, dynamic problems

- Objective functions
  - » Performance metrics:
    - Rewards, profits, revenues, costs, contributions, rewards (business)
    - Gains, losses (engineering)
    - Strength, conductivity, diffusivity (materials science)
    - Tolerance, toxicity, effectiveness (health)
    - Stability, reliability (engineering)
    - Risk, volatility (finance)
    - Utility (economics)
    - Errors (machine learning)
    - Time (to complete a task)

Modeling stochastic, dynamic problems

- Objective functions
  - » Uncertainty metrics:
    - Expectations
    - Risk measures
      - Variance
      - Quantiles $\max_x Q_\alpha F(x,W)$
      - Expected semi-deviations
      - Var, CVaR, …
    - Robust (worst-case) $\max_x \min_w Q_\alpha F(x,w)$

Modeling stochastic, dynamic problems

- Objective functions
  - » Cumulative reward ("online learning")

$$\max_{\pi} \mathbb{E}\left\{\sum_{t=0}^{T} C_t\left(S_t, X_t^{\pi}(S_t), W_{t+1}\right) \mid S_0\right\}$$

    - Policies have to work well *over time.*
  - » Final reward ("offline learning")

$$\max_{\pi} \mathbb{E}\left\{F(x^{\pi,N}, \hat{W}) \mid S_0\right\}$$

    - We only care about how well the final decision $x^{\pi,N}$ works.
  - » Risk

$$\max_{\pi} \rho\left\{C(S_0, X_0^{\pi}(S_0)), C(S_1, X_1^{\pi}(S_1)), \ldots, C(S_T, X_T^{\pi}(S_T)) \mid S_0\right\}$$

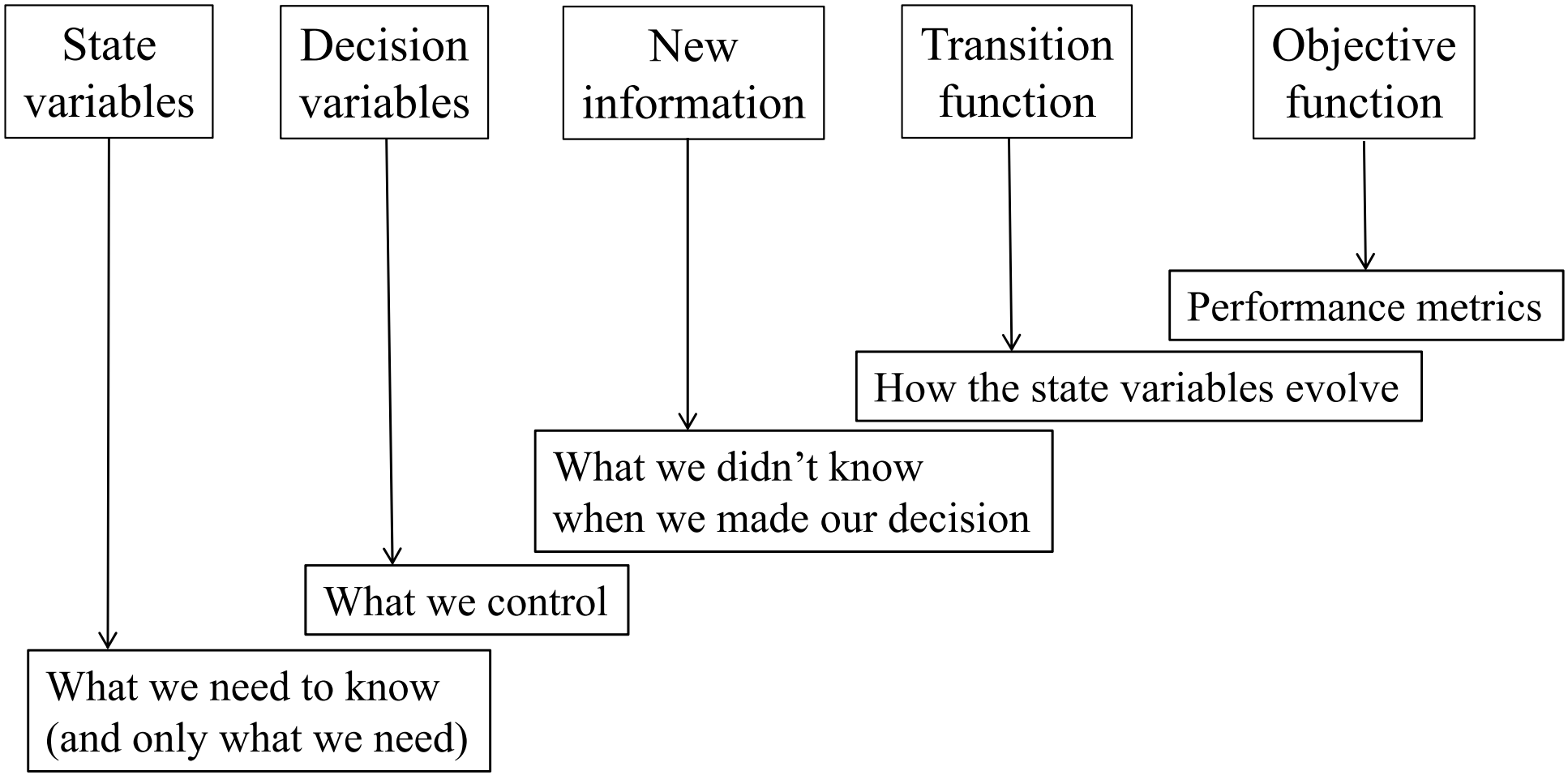
- The modeling process:
  - » The narrative
    - This is a plain English description of the problem. This should be fairly short, but should give the audience a sense of the problem.
  - » The model
    - This consists of the five elements:

States | Decisions | Exogenous information | Transition function | Objective function

  - » The modeling process
    - Filling in the elements of the model is not linear. State variables are built up as part of the process.

The modeling process

- Modeling real applications
  - » I conduct a conversation with a domain expert to fill in the elements of a problem:



Modeling

- Deterministic
  - » Objective function

$$\min_{x_0,\ldots,x_T} \sum_{t=0}^{T} c_t x_t$$

  - » Decision variables:

$$(x_0,\ldots,x_T)$$

  - » Constraints:
    - at time t

$$\left.\begin{array}{l} A_t x_t = R_t \\ x_t \geq 0 \end{array}\right\} \mathcal{X}_t$$

    - Transition function

$$R_{t+1} = b_{t+1} + B_t x_t$$

- Stochastic
  - » Objective function

$$\max_{\pi} E^{\pi}\left\{\sum_{t=0}^{T} C_t\left(S_t, X_t^{\pi}(S_t), W_{t+1}\right) \mid S_0\right\}$$

  - » Policy

$$X^{\pi} : S \mapsto \mathcal{X}$$

  - » Constraints at time t

$$x_t = X_t^{\pi}(S_t) \in \mathcal{X}_t$$

  - » Transition function

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

  - » Exogenous information

$$(S_0, W_1, W_2, \ldots, W_T)$$

An asset selling problem

The narrative

We are holding a block of shares of stock, looking for an opportune time to sell. Start by assuming that we are a small player which means it does not matter how many shares we sell, so we are going to assume that we have just one share. If we sell at time t, we receive a price that varies according to some random process over time, although we do not believe prices are trending up or down. Once we sell the stock, the process stops.

State variables

State variables

Our process has two state variables: the "physical state," which captures whether or not we are still holding the asset, and an "informational state" which for this problem is the price of the stock.

This means that our "physical state" is given by

$$R_t = \begin{cases} 1 & \text{If we are holding the stock at time } t, \\ 0 & \text{If we are no longer holding the stock at time } t. \end{cases}$$

If we sell the stock, we receive the price per share of p_t. This means our state variable is

$$S_t = (R_t, p_t).$$

» Highlight:

- Physical state R_t
- Information state p_t

Decision variables

The decision variable is whether to hold or sell the stock. We write this using

$$x_t = \begin{cases} 1 & \text{If we sell the stock at time } t, \\ 0 & \text{If do not sell the stock at time } t. \end{cases}$$

Of course, we can only sell the stock if we are holding the stock, so we might write a constraint as

$$x_t \leq R_t.$$

We are going to define our policy $X^\pi(S_t)$ which is going to define how we make decisions. For example, a simple policy might be to sell if the price drops below some limit point that we think suggests that it is starting a big decline. Thus, we could write this policy as

$$X^{sell-low}(S_t|\theta^{low}) = \begin{cases} 1 & \text{If } p_t < \theta^{low} \text{ and } R_t = 1 \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \tag{2.1}$$

We would of course need to tune θ^{low} to get the best performance within this class of policy. A separate question is whether this is a good structure of a policy, since it is easy to design others.

Exogenous information

The only random process in our basic model is the change in price. There are two ways to write this. One is to assume that the exogenous information is the change in price. We can write this as

$$\hat{p}_{t+1} = p_{t+1} - p_t.$$

This means that our price process is evolving according to

$$p_{t+1} = p_t + \hat{p}_{t+1}.$$

We would then write our exogenous information W_{t+1} as

$$W_{t+1} = (\hat{p}_{t+1}).$$

» We could drill into the probability model, but we are going to put "uncertainty modeling" after the basic model.

Transition function

The transition function is the equations that describe how the state evolves. The transition equation for R_t is given by

$$R_{t+1} = R_t - x_t, \tag{2.2}$$

where we have the constraint that $x_t \leq R_t$ to ensure that we do not sell the asset when we no longer own it.

Next we have to write how the price process evolves over time. If we use the $\hat{p}_t$ notation, the transition function for the price p_t would be given by

$$p_{t+1} = p_t + \hat{p}_{t+1}. \tag{2.3}$$

Equations (2.2) and (2.3) make up what we call our *transition function* that we write as

$$S_{t+1} = S^M(S_t, X^\pi(S_t), W_{t+1}). \tag{2.4}$$

If we use our policy $X^\pi(S_t)$ to make decisions, and if we choose sample path ω that determines the sequence $W_1, W_2, \ldots, W_T$, then we can write a simulation of our process as

$$(S_0, x_0 = X^\pi(S_0), W_1(\omega), S_1, x_1 = X^\pi(S_1), W_2(\omega), \ldots, x_{T-1}, W_T(\omega), S_T).$$

Note that as we write the sequence, we index variables by their information content. For example, S_0 is an initial state, and x_0 depends only on S_0. By contrast, any variable indexed by t is allowed to "see" any of the outcomes of our exogenous process $W_1, \ldots, W_t$, but are not allowed to see W_{t+1}.

Objective function

We close with the problem of determining the best policy. To start, we have to have some performance metric, which for this problem would be how much we earn from selling our stock. We can define a generic contribution function that we write as $C(S_t, x_t)$, which would be given by

$$C(S_t, x_t) = p_t x_t.$$

In our problem, $x_t = 0$ until we sell our asset, at which point we would have $x_t = 1$, which is going to happen just once over our horizon. We write the dependence of $C(S_t, x_t)$ to capture the dependence on the state, which is because of the presence of the price p_t.

We now want to formulate our optimization problem. If the prices were given to us in advance, we would write

$$\max_{x_0,\ldots,x_{T-1}} \sum_{t=0}^{T-1} p_t x_t, \tag{2.5}$$

where we would impose the constraints

$$\sum_{t=0}^{T-1} x_t = 1, \tag{2.6}$$

$$x_t \leq 1, \tag{2.7}$$

$$x_t \geq 0. \tag{2.8}$$

This is fine when the problem is deterministic, but how do we model the problem to handle the uncertainty in the prices? What we do is to imagine that we are simulating a policy following a sample path ω of prices $p_1(\omega), p_2(\omega), \ldots$. Using a policy π, we would then generate a series of states using

$$S_{t+1}(\omega) = S^M(S_t(\omega), X^\pi(S_t(\omega)), W_{t+1}(\omega)).$$

We write $S_t(\omega)$ to express the dependence on the sample path. We could also have written $S_t^\pi(\omega)$ to express the dependence on the policy π, but we tend to suppress this for simplicity.

If we follow policy π along this sample path, we can compute the performance using

$$\hat{F}^{\pi}(\omega) = \sum_{t=0}^{T-1} p_t(\omega) X^{\pi}(S_t(\omega)).$$

This is for one sample path. We can simulate over a sample of N samples $\omega^1, \ldots, \omega^n, \ldots, \omega^N$ and take an average using

$$\overline{F}^{\pi} = \frac{1}{N} \sum_{n=1}^{N} \hat{F}^{\pi}(\omega^n). \tag{2.9}$$

Finally, we write out optimization problem in terms of finding the best policy, which we can write

$$\max_{\pi \in \Pi} \overline{F}^{\pi}. \tag{2.10}$$

In practice we are typically using averages such as in equation (2.9) for our optimization problem. However, this is just an approximation of taking an actual expectation, which we will write as

$$F^{\pi} = \mathbb{E} \sum_{n=1}^{N} \hat{F}^{\pi}. \tag{2.11}$$

By convention, when we write the expectation, we drop the indexing on ω and instead view $\hat{F}^{\pi}$ as a random variable, whereas $\hat{F}^{\pi}(\omega)$ is treated as a sample realization (this is standard

convention, so just get used to it). Using our expectation operator, we would write our objective function as

$$\max_{\pi \in \Pi} \mathbb{E} \sum_{n=1}^{N} F^{\pi}. \tag{2.12}$$

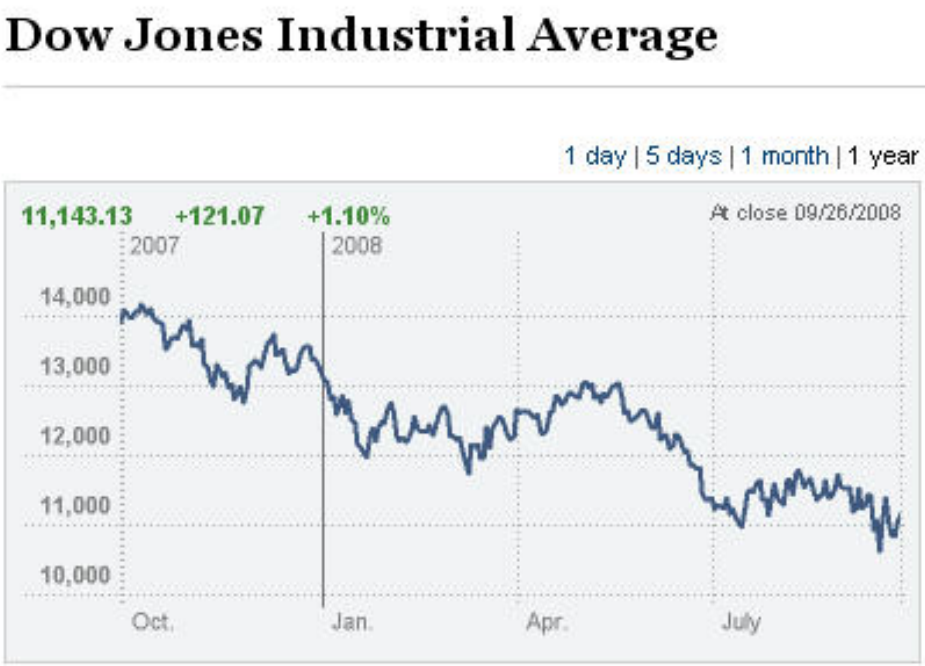
The form in equation (2.12) (or (2.11)) is nice and compact. You just have to remember that it is rarely the case that we can actually compute the expectation, so we generally depend on simulation.

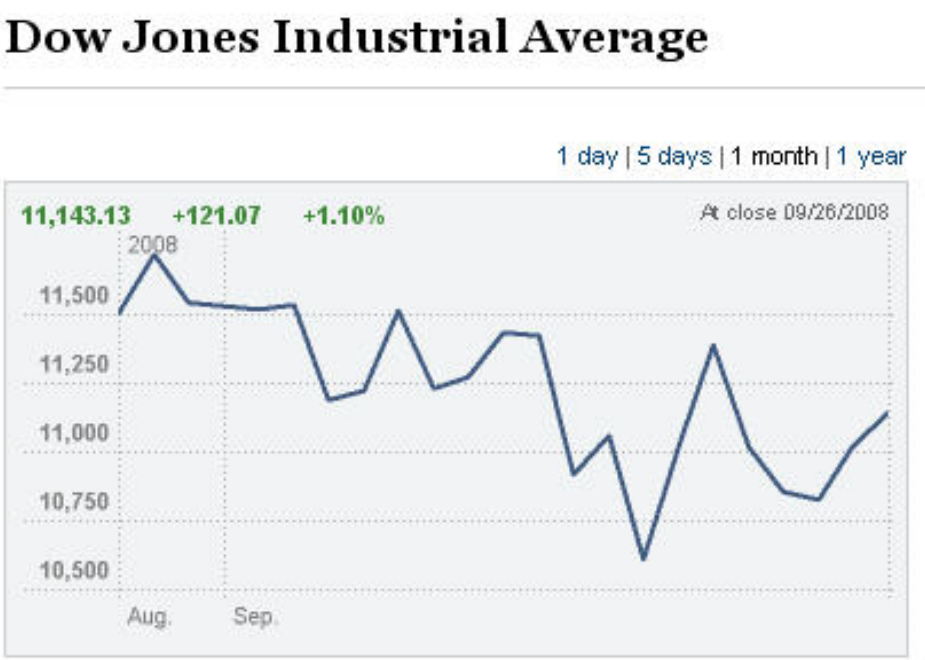
Now we are left with the problem of searching over policies. We are going to return to this below, but for the moment, we can think of this search in two ways. First, we have to think of different types of functions for making decisions. One example of a function is given by the policy in 2.1, but it is not hard to think of others (for example, selling if the price drops by too much, or goes over some upper limit). Then, we have to recognize that functions such as 2.1 may depend on parameters, such as θ^{sell}. To evaluate the policy in (2.1) would require that we first find the best value of θ^{sell}.

Modeling uncertainty

Exogenous information

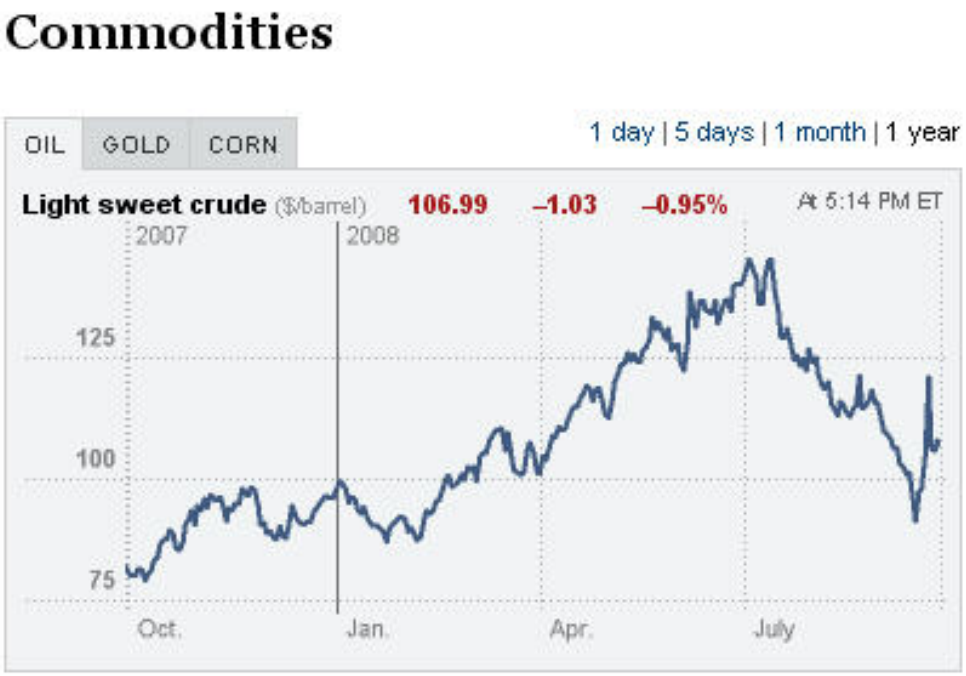
- Guessing what is happening in the real world
  - » Is the stock market going (further) down?
  - » What do you think?

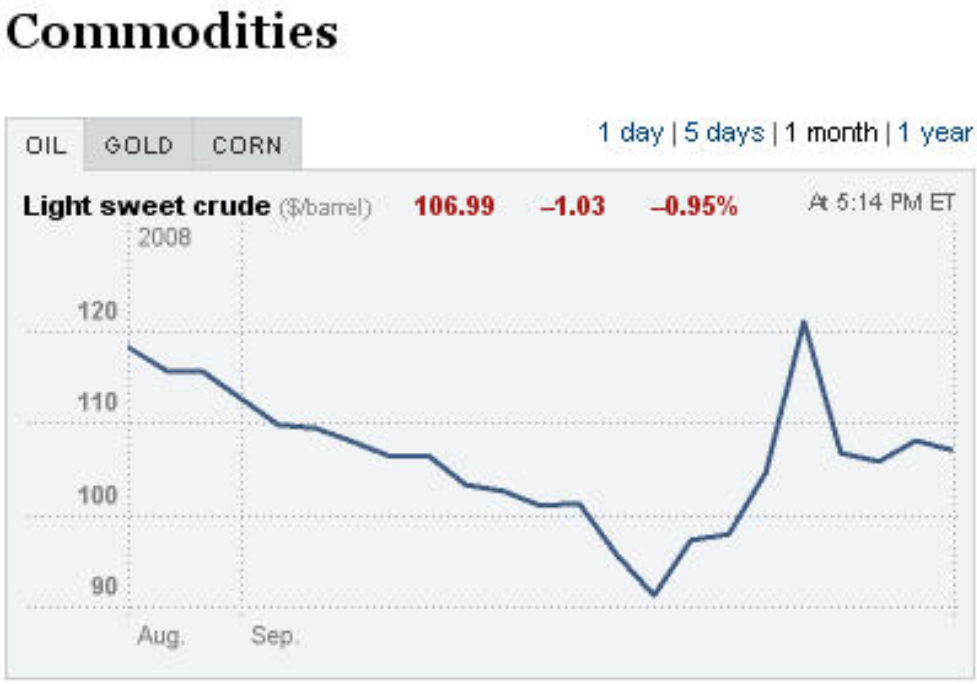




Exogenous information

- Guessing what is happening in the real world
  - » What about the price of the barrel of crude oil?





Exogenous information

The only random process in our basic model is the change in price. There are two ways to write this. One is to assume that the exogenous information is the change in price. We can write this as

$$\hat{p}_{t+1} = p_{t+1} - p_t.$$

This means that our price process is evolving according to

$$p_{t+1} = p_t + \hat{p}_{t+1}.$$

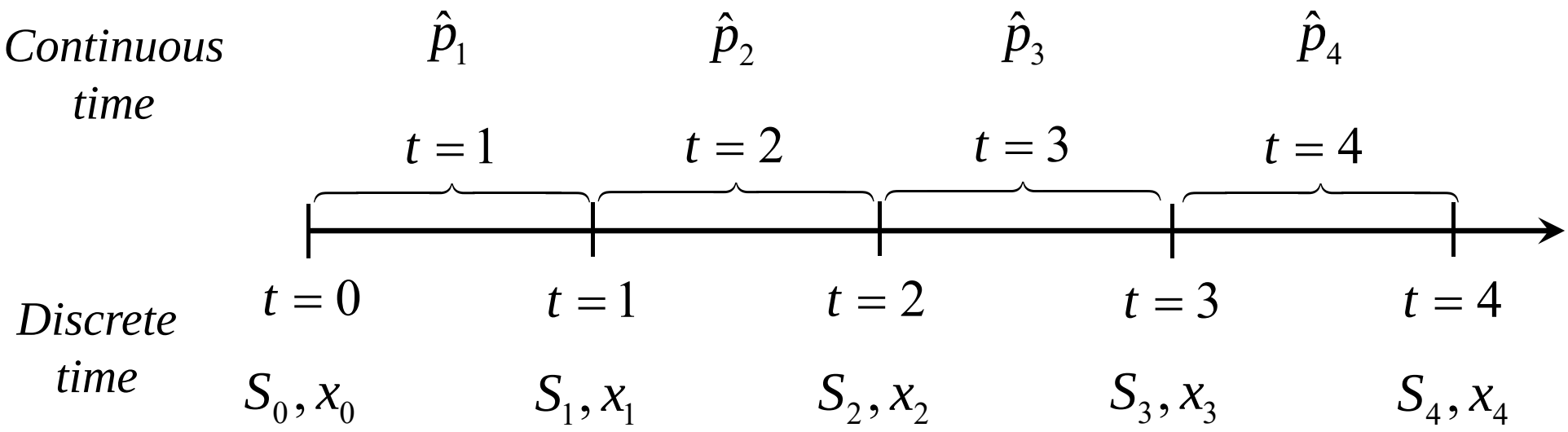
We would then write our exogenous information W_{t+1} as

$$W_{t+1} = (\hat{p}_{t+1}).$$

» We could drill into the probability model, but we are going to put "uncertainty modeling" after the basic model.

Exogenous information

- We need a system for indexing time. In particular, it is important to know the mapping between discrete and continuous time.



It is useful to think of information as arriving continuously over time. Functions (states, decisions) are measured at a point in time.

At time t, anything t' <= t is known, anything t' > t is unknown.

Exogenous information

All the information arriving would then be:

$$\omega = (\omega_1, \omega_2, ..., \omega_t, ...)$$

ω_t is not a random variable (although some people treat it as one). We sometimes need a random variable which is a function providing the information that *might* arrive during time period *t*. If we do not have a specific variable such as a price, quantity or demand, we can use the generic notation:

W_t = The information arriving in time *t*.

We use W_t because it "looks like" ω_t. We can also write:

$$\omega_t = W_t(\omega)$$

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$
ω^n	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
ω^1	42.67	45.53	47.07	47.56	47.80	48.43	46.93	46.57
ω^2	46.35	43.15	42.51	40.51	41.50	41.00	39.16	41.11
ω^3	43.17	45.16	45.37	44.30	45.35	47.23	47.35	46.30
ω^4	45.24	45.67	46.18	46.22	45.69	44.24	43.77	43.57
ω^5	47.68	46.32	46.14	41.53	44.84	45.17	44.92	46.09
ω^6	47.83	44.70	43.05	43.77	42.61	44.32	44.16	45.29
ω^7	45.11	43.67	43.14	44.78	43.12	42.36	41.60	40.83
ω^8	46.78	44.98	44.53	45.42	46.43	47.67	47.68	49.03
ω^9	43.16	44.57	45.99	47.38	45.51	46.27	46.02	45.09
ω^{10}	46.57	45.01	46.73	46.08	47.40	49.14	49.03	48.74

Table 2.2 Illustration of a set of price paths.

The real power of functions such as `Norm.inv`(p, μ, σ) is that it allows us to generate samples of our random variable W. To do this, we use the presence of a function (again, available in all computer packages) to generate a random variable that we call U that is uniformally distributed between 0 and 1. In Excel, this function is called `Rand()`. We can then generate random observations of our normally distributed random variable W using

$$W = Norm.inv(U, \mu, \sigma).$$

Table 2.1 illustrates ten observations of random variables U that are uniformly distributed between 0 and 1, and the corresponding samples of normally distributed random variables with mean 0 and variance 1.

U	W
0.8287	0.9491
0.6257	0.3206
0.9343	1.5086
0.4879	-0.0303
0.3736	-0.3223
0.8145	0.8947
0.0385	-1.7685
0.0089	-2.3698
0.9430	1.5808
0.3693	-0.3336

An alternative way that some will find more natural is to simply let W_t be the new price, in which we would write

$$W_{t+1} = (p_{t+1}).$$

Note that with this style, p_{t+1} is the exogenous information, but we are not writing it with a hat. The value of the hat notation is that it tells us that a variable is determined exogenously, rather than being a variable that depends on other random variables. We could let $\hat{p}_{t+1}$ be the price that arrives exogenously at time $t+1$, but this then requires us to either write $p_{t+1} = \hat{p}_{t+1}$, or just use $\hat{p}_{t+1}$ as the price (rather than the more compact p_{t+1}). Both styles are correct and just depend on the preference of the modeler.

2.3 MODELING UNCERTAINTY

We are going to need some way to sample observations of W_t. One way is to draw samples from history. Imagine that we are interested in running our simulation over a period of a year. We could think of getting 10 years of past data, and scaling it so the processes all start at the same point. This is a popular and widely used method for creating samples, since it avoids making any simplifying modeling assumptions. The drawback is that this only gives us 10 sample paths, which may represent patterns from history that are no longer relevant.

The second strategy, which we will often use, is to estimate a statistical model. For our basic model, we might assume

$$p_{t+1} = p_t + \varepsilon_{t+1}, \tag{2.13}$$

where ε_{t+1} is normally distributed with mean 0 and some variance σ^2_ε. In addition to assuming that ε_{t+1} is normally distributed, we are also going to assume that the sequence $\varepsilon_1, \ldots, \varepsilon_t, \ldots$ are also independent.

Equation (2.13) is a pretty basic price model, but it will help illustrate our modeling framework. Below, we are going to introduce some extensions which include a richer model.

The probability model

The only random process in our basic model is the change in price. There are two ways to write this. One is to assume that the exogenous information is the change in price. We can write this as

$$\hat{p}_{t+1} = p_{t+1} - p_t.$$

This means that our price process is evolving according to

$$p_{t+1} = p_t + \hat{p}_{t+1}.$$

We would then write our exogenous information W_{t+1} as

$$W_{t+1} = (\hat{p}_{t+1}).$$

To simulate our process, we need to assume a probabilistic model for $\hat{p}_{t+1}$. A simple model would be to assume that $\hat{p}_{t+1}$ is normally distributed with mean 0 and variance σ^2. Later we are going to have to simulate observations of W_{t+1} (or $\hat{p}_{t+1}$). There are a number of ways to do this, but most computer languages have provided functions for doing this. For example, Excel provides the function $\texttt{Norm.inv}(p, \mu, \sigma)$, which returns the value w of a random variable W with mean μ and standard deviation σ where $P[W \leq w] = p$. If we set $\mu = 0$ and $\sigma = 1$, then if $p = .5$ we would obtain $Norm.inv(.5, 0, 1) = 0$. Similarly, if $p = 0.975$, we would obtain $\texttt{Norm.inv}(.975, 0, 1) = 1.96$.

- Discuss:
  - How to simulate a normally distributed random variable
  - Simulating other distributions

Evaluating policies

- Modeling simulations
  - » Introduce ω notation
  - » Concept of a sample path
    - Purely exogenous information

$$W_1(\omega), W_2(\omega), \dots, W_T(\omega)$$

    - Full process with states, actions and new information.

$$(S_0, x_0, W_1, S_1, x_1, W_2, S_2, \dots, S_t, x_t, W_{t+1}, \dots)$$

where $x_t = X^\pi(S_t)$

We can also write with iteration n:

$$(S^0, x^0, W^1, W^1, x^1, W^2, S^2, \dots, S^n, x^n, W^{n+1}, \dots)$$

2.4.1 Policy evaluation

We indicated above that we can evaluate a policy by simulating it using

$$\hat{F}^{\pi}(\omega) = \sum_{t=0}^{T-1} p_t(\omega) X^{\pi}(S_t(\omega)).$$

where ω is used to represent a sample path of realizations of whatever exogenous random variables are used in the model. Table 2.2 illustrates a series of sample paths of prices. For example, imagine that we are using the "sell-low" policy with $\theta^{sell-low} = \$42$. Now consider testing it on sample path ω^5. The result would be

$$\hat{F}^{sell-low}(\omega^5) = \$41.53,$$

since $\$41.53$ is the first price that falls below $\$42$. If none of the prices fall below our sell point, then all of our policies are designed to sell at the end.

We can then evaluate each policy (both the class of policy, and the parameters for that class) by doing repeated simulations and taking an average. We write this as

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

Sometimes we need to express a confidence interval, since $\overline{F}^{\pi}$ is nothing more than a statistical estimate. We would first compute an estimate of the variance of our random variable $\hat{F}^{\pi}$, which we do using

$$(\hat{\sigma}^{\pi})^2 = \frac{1}{N-1}\sum_{n=1}^{N}(\hat{F}^{\pi}(\omega^n) - \overline{F}^{\pi})^2.$$

We then get our estimate of the variance of our average $\overline{F}^{\pi}$ using

$$(\bar{\sigma}^{\pi})^2 = \frac{1}{N}(\hat{\sigma}^{\pi})^2.$$

From this, we can construct a confidence interval to compare two policies which we might call π^A and π^B. Let $\mu^p i$ be the true performance of policy π, where $\overline{F}^{\pi}$ is our statistical estimate of μ^{π}. We would like to get a confidence interval for the difference $\mu^{\pi^A} - \mu^{\pi^B}$. Our best estimate of this difference is $(\overline{F}^{\pi^A} - \overline{F}^{\pi^B})$. The variance of this difference is

$$Var(\overline{F}^{\pi^A} - \overline{F}^{\pi^B}) = (\bar{\sigma}^{\pi^A})^2 + (\bar{\sigma}^{\pi^B})^2),$$

where we assume that the estimates $\overline{F}^{\pi^A}$ and $\overline{F}^{\pi^B}$ are independent, which means that each policy is being tested on a different random sample of prices. When this is the case, we would compute our confidence interval using

$$\mu^{\pi^A} - \mu^{\pi^B} \in \left(\overline{F}^{\pi^A} - \overline{F}^{\pi^A} + z_\alpha\sqrt{(\bar{\sigma}^{\pi^A})^2 + (\bar{\sigma}^{\pi^B})^2}\right).$$

A better approach is to use the same samples to evaluate each policy. For example, we could test each policy on the same sample path ω chosen from table 2.2). Testing our policies this way, we would get $\hat{F}^{\pi^A}(\omega)$ and $\hat{F}^{\pi^B}(\omega)$ (using the same set of prices $p_t(\omega)$), and then compute the difference

$$\delta\hat{F}^{A-B}(\omega) = \hat{F}^{\pi^A}(\omega) - \hat{F}^{\pi^B}(\omega).$$

Now we compute the average difference

$$\delta\overline{F}^{A-B} = \frac{1}{N}\sum_{n=1}^{N}\hat{F}^{A-B}(\omega^n),$$

and the variance

$$(\delta\bar{\sigma}^{A-B})^2 = \frac{1}{N}\left(\frac{1}{N-1}\sum_{n=1}^{N}(\delta\hat{F}^{A-B}(\omega^n) - \delta\overline{F}^{A-B})^2\right).$$

Note that the variance $\delta\bar{\sigma}^{A-B,2}$ will be smaller than the variance when independent samples are used. The confidence interval for the difference would then be

$$\delta\mu^{A-B} \in (\delta\overline{F}^{A-B} + z_\alpha\sqrt{\delta\bar{\sigma}^{A-B,2}}).$$

Computing confidence intervals may be useful when comparing different classes of policies. We would not use confidence intervals to decide the best parameter θ that governs a policy.

Designing policies

2.4 DESIGNING POLICIES

We have already illustrated one policy in equation (2.1) which we called our "sell-low" policy, which is parameterized by a selling point that we called $\theta^{sell-low}$. Clearly we can think of other policies. One might be a "high-low" selling policy, where we want to sell if the price jumps too high or too low. Let $\theta^{high-low} = (\theta^{low}, \theta^{high}$. This might be written

$$X^{high-low}(S_t|\theta^{high-low}) = \begin{cases} 1 & \text{If } p_t < \theta^{low} \text{ or } p_t > \theta^{high} \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \tag{2.14}$$

A possible objection to this policy could be that it prematurely sells a rising stock. Perhaps we just want to sell when the stock rises above a tracking signal. First create a smoothed estimate of the price using

$$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha\hat{p}_t. \tag{2.15}$$

Now consider a tracking policy that we might write as

$$X^{track}(S_t|\theta^{track}) = \begin{cases} 1 & \text{If } p_t \geq \bar{p}_t + \theta^{track} \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \tag{2.16}$$

For this policy, we are going to need to tweak our model, because we now need $\bar{p}_t$ in order to make a decision. This means we would now write our state as

$$S_t = (R_t, p_t, \bar{p}_t).$$

We can write our classes of policies as the set $\mathcal{F} = \{\text{"}sell{-}low\text{"}, \text{"}high{-}low\text{"}, \text{"}track\text{"}\}$. For each of these classes, we have a set of parameters that we can write as θ^f for $f \in \mathcal{F}$. For the "sell-low" and "track" there is a single parameter, while $\theta^{high-low}$ has two parameters.

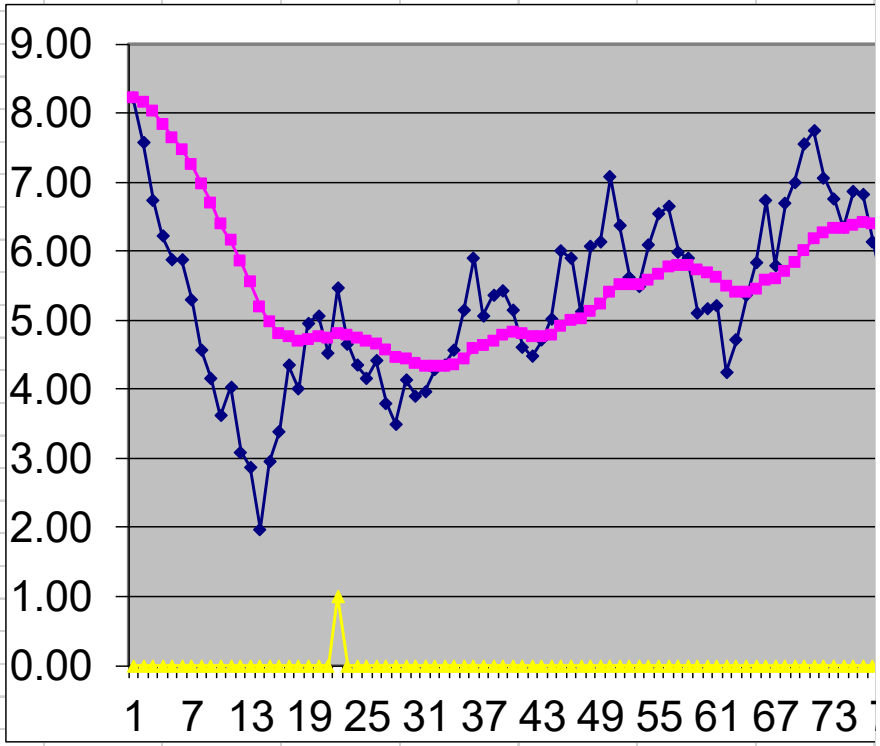
Now we can write our search over policies $\pi \in \Pi$ in a more practical way as searching over function classes $f \in \mathcal{F}$, and then searching over parameters $\theta^f \in \Theta^f$, where Θ^f tells us the range of possible values (capturing at the same time the dimensionality of θ^f).

- Discuss:
  - » Tuning parameters of a policy is actually its own sequential decision problem. ☺
  - » For now, let's just do an ad-hoc search. As the course unfolds, we will be able to do a better job of this.

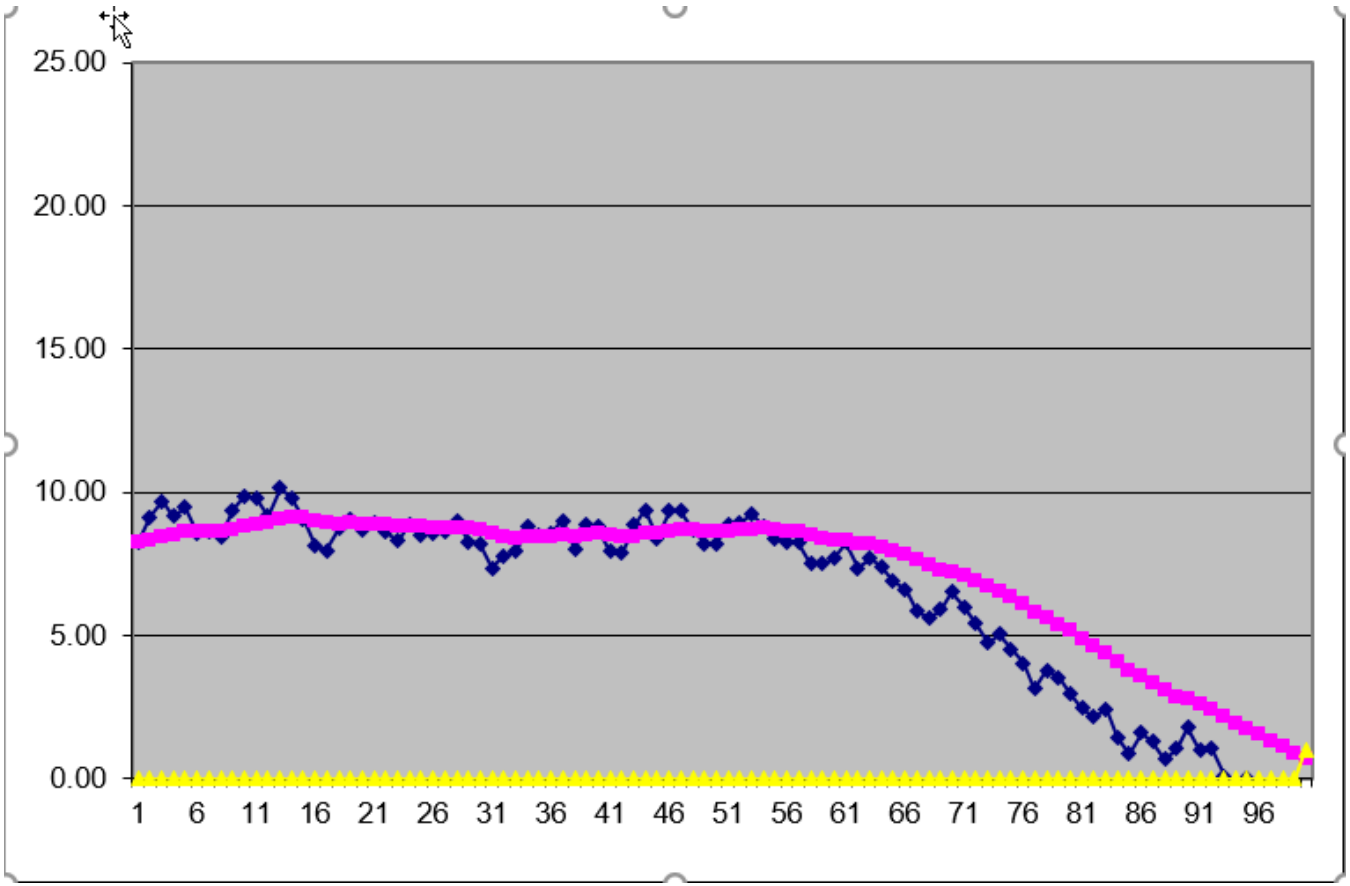
The objective function

- Search for the best value of beta (click here for spreadsheet)

Alpha	Beta						sum=	5.462638
0.1	0.60		R	X	C			
	Price	Smoothed						
1	8.23	8.23	1	0	0			
2	7.57	8.163733	1	0	0			
3	6.74	8.021005	1	0	0			
4	6.22	7.840923	1	0	0			
5	5.88	7.645048	1	0	0			
6	5.87	7.467439	1	0	0			
7	5.29	7.249638	1	0	0			
8	4.57	6.98159	1	0	0			
9	4.15	6.698608	1	0	0			
10	3.62	6.390541	1	0	0			
11	4.04	6.155371	1	0	0			
12	3.09	5.848898	1	0	0			
13	2.88	5.551598	1	0	0			
14	1.97	5.19386	1	0	0			
15	2.95	4.969746	1	0	0			
16	3.39	4.812055	1	0	0			
17	4.35	4.765712	1	0	0			
18	4.00	4.6895	1	0	0			
19	4.96	4.716665	1	0	0			
20	5.06	4.751376	1	0	0			
21	4.52	4.72774	1	0	0			
22	5.46	4.801229	1	1	5.462638			
23	4.66	4.787369	0	0	0			
24	4.34	4.74304	0	0	0			
25	4.16	4.684932	0	0	0			
26	4.41	4.65746	0	0	0			
27	3.79	4.571039	0	0	0			
28	3.50	4.463759	0	0	0			
29	4.13	4.430225	0	0	0			
30	3.91	4.378293	0	0	0			

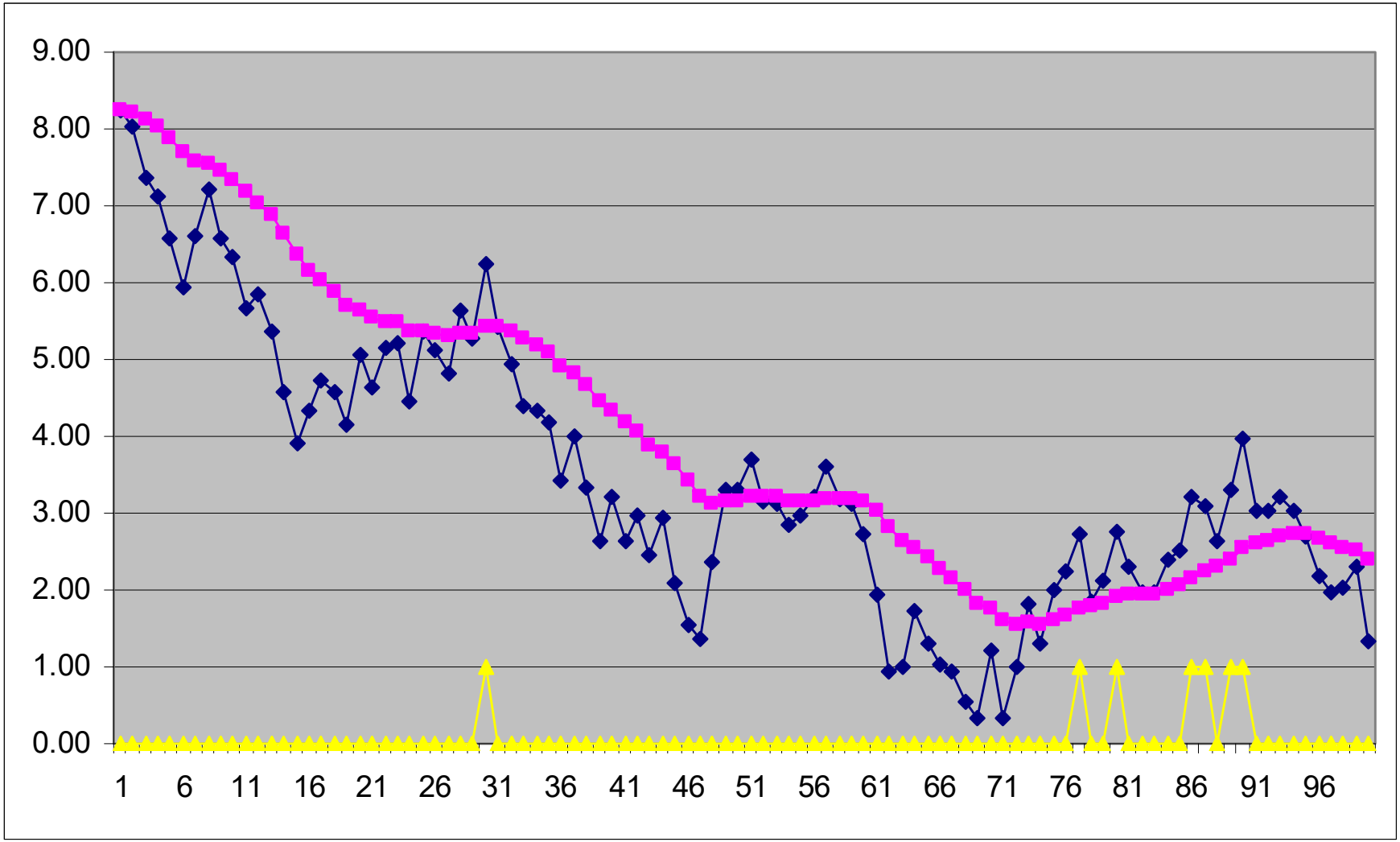


- Asset selling exercise
  - » Click on image for spreadsheet



The objective function

- Simulated price process



Extensions

2.5.1 Time series price processes

Imagine that we want a somewhat more realistic price process that captures autocorrelation over time. We might propose that

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon_{t+1}, \tag{2.17}$$

where we still assume that the random noise ε_t is independent (and identically distributed) over time. We are also going to assume for the moment that we know the coefficients $\theta = (\theta_0, \theta_1, \theta_2)$.

This change requires a subtle change in our model, specifically the state variable. We are going to replace our old transition equation for prices, (2.3), with our new time series model given in (2.17). To compute p_{t+1}, it is no longer enough to know p_t, we now also need to know p_{t-1} and p_{t-2}. Our state variable would now be given by

$$S_t = (R_t, p_t, p_{t-1}, p_{t-2}).$$

For the policies we have considered above, this does not complicate our model very much. Later, we are going to introduce policies where the additional variables represent a major complication.

- Basket of assets
  - » Imagine that we sell based on an index that reflects a basket of assets, each of which are evolving on their own.
  - » Now the price p_t becomes a vector $p_t = (p_{ti}), i = 1, \ldots, N$

Week 1 - Monday

Adaptive market planning

Narrative

Narrative

3.1 NARRATIVE

There is a broad class of problems that involve allocating some resource to meet an uncertain (and sometimes unobservable) demand. Examples include

- Stocking a perishable inventory (e.g. fresh fish) to meet a demand where leftover inventory cannot be held for the future.
- Stocking parts for high-technology manufacturing (e.g. jet engines) where we need to order parts to meet known demand, but where the parts may not meet required specifications and have to be discarded. So, we may need to order eight parts to meet the demand of five because several of the parts may not meed required engineering specifications.
- We have to allocate time to complete a task (such as driving to work, or allocating time to complete a project).
- We have to allocate annual budgets for activities such as marketing. Left-over funds are returned to the company.

The simplest problem involves making these decisions to meet an uncertain demand with a known distribution, but the most common applications involve distributions that are unknown and need to be learned. There may be other information such as the availability

Narrative

The newsvendor problem is typically formulated in the form

$$\max_x \mathbb{E}F(x,W) = \mathbb{E}\{p\min\{x,W\} - cx\}, \tag{3.1}$$

where x is our decision variable that determines the amount of resource to meet the task, and where W is the uncertain demand for the resource. We assume that we "purchase" our resource at a unit cost of c, and we sell the smaller of x and W at a price p (which we assume is greater than c). The objective function given in equation (3.1) is called the *asymptotic* form of the newsvendor problem.

If W was deterministic (and if $p > c$), then the solution is easily verified to be $x = W$. Now imagine that W is a random variable with probability distribution $f^W(w)$ (W may be discrete or continuous). Let $F^W(w) = Prob[W \leq w]$ be the cumulative distribution function. If W is continuous, and if we could compute $F(x) = \mathbb{E}F(x,W)$, then the optimal solution x^* would satisfy

$$\left.\frac{dF(x)}{dx}\right|_{x=x^*} = 0.$$

Now consider the stochastic gradient, where we take the derivative of $F(x,W)$ assuming we know W, which is given by

$$\frac{dF(x,W)}{dx} = \begin{cases} p-c & x \leq W \\ -c & x > W \end{cases} \tag{3.2}$$

Narrative

Taking expectations of both sides give

$$\begin{aligned}
\mathbb{E}\frac{dF(x,W)}{dx} &= (p-c)Prob[x \leq W] - cProb[x > W] \\
&= (p-c)(1-F^W(x)) - cF^W(x) \\
&= (p-c) - pF^W \\
&= 0 \text{ For } x = x^*.
\end{aligned}$$

We can now solve for $F^W(x^*)$ giving

$$F^W(x^*) = \frac{p-c}{p}.$$

Thus, as c decreases to 0, we want to order an amount x^* that will satisfy demand with probability 1. As c approaches p, then the optimal order quantity will satisfy demand with a probability approaching 0.

This means that we compute $(p-c)/p$, which is a number between 0 and 1, and then find the quantity x^* that corresponds to the order quantity where the probability that the random demand is less than x^* is equal to $(p-c)/p$.

We have just seen two situations where we can find the order quantity exactly: when we know W in advance (we might call this the perfect forecast) or when we know the distribution of W. This result has been known since the 1950's, sparking a number of

Narrative

- Notes:
  - » With enough handwaving, we can boil *any* stochastic optimization problem down to

$$\mathbb{E}F(x,W)$$

  - » The newsvendor problem with the distribution of W is known is an instance of a stochastic optimization problem that we can solve exactly when we can compute the expectation.
  - » In this problem, we are going to assume the distribution of W is unknown, but we can adaptively set x^n and then observe W^{n+1}

Stochastic gradient algorithms

Adaptive market planning

- Gradient-based stochastic search
  - » Powerful algorithm, but it an important class of sequential decision problem

$$x^{n+1} = x^n + \alpha_n \nabla_x F(x^n, W^{n+1}), \tag{3.3}$$

where α_n is known as a *stepsize*. This algorithm has been widely studied for its asymptotic behavior, where it has been shown that we obtain asymptotic optimality

$$\lim_{n\to\infty} x^n = x^*,$$

if the stepsize α_n satisfies

$$\alpha_n > 0, \tag{3.4}$$

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \tag{3.5}$$

$$\sum_{n=1}^{\infty} (\alpha_n)^2 < \infty. \tag{3.6}$$

Equation (3.5) ensures that the stepsizes do not shrink so quickly that we stall out on the way to the optimum, while (3.6) has the effect of insuring that the variance of our estimate of x^* does shrink to zero.

Adaptive market planning

- Stepsize rules
  - » Harmonic stepsize formula (deterministic

$$\alpha_n(\theta^{step}) = \frac{\theta^{step}}{\theta^{step} + n - 1}, \tag{3.8}$$

  - » Kesten's rule (stochastic)

$$\alpha_n(\theta^{step}) = \frac{\theta^{step}}{\theta^{step} + K^n - 1}, \tag{3.9}$$

$$K^n = \begin{cases} K^n + 1 & \text{If } (\nabla^x F(x^n, W^{n+1}))^T \nabla^x F(x^{n-1}, W^n) < 0, \\ K^n & \text{Otherwise} \end{cases} \tag{3.10}$$

  - » Discuss:
    - Scaling
    - Handling bias (learning) vs. noise (smoothing)

Adaptive market planning

- Harmonic stepsize rules

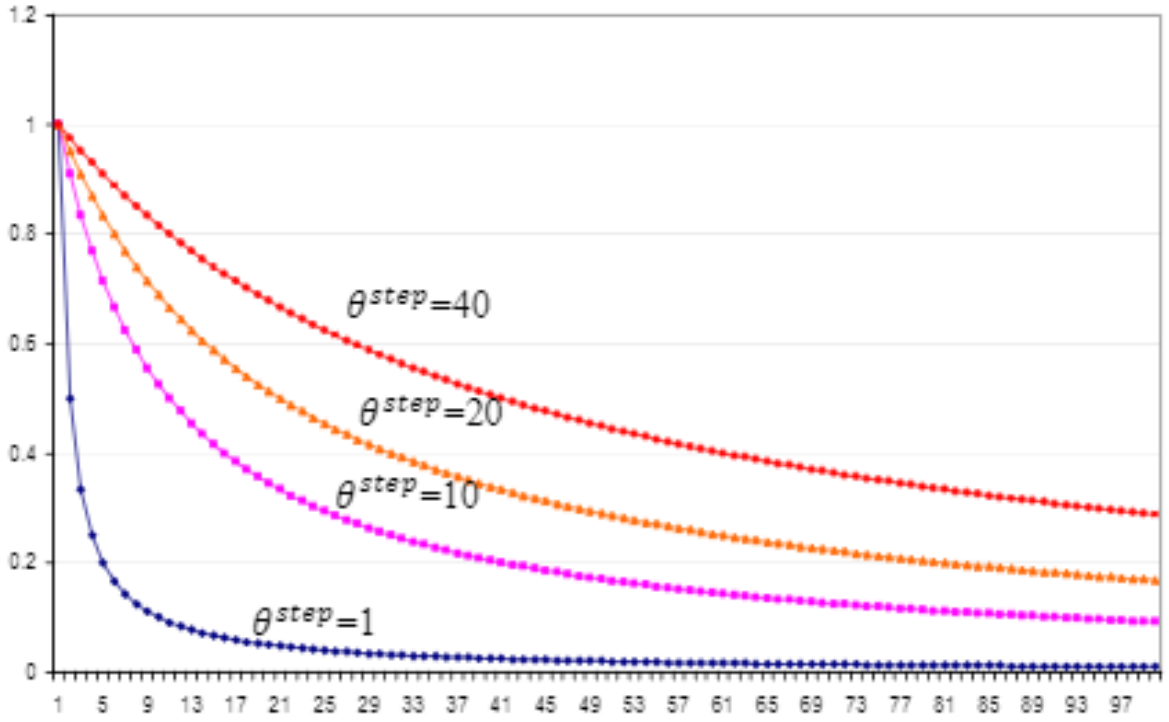


Figure 3.1 Harmonic stepsizes for different values of θ^{step}.

Basic model

Kesten's stepsize rule

Basic model

State variables

The state variable captures the information we have at time n that we need, along with the policy and the exogenous information, to compute the state at time $n+1$. For our stochastic gradient algorithm, our state variable is given by

$$S^n = (x^n).$$

Decision variables

The trick with this problem is recognizing the decision variable. It is tempting to think that x^n is the decision, but in the context of this algorithm, the real decision is the stepsize α_n (note: while we generally index functions and variables using counter n in the superscript, we make an exception with stepsizes, since α^n is easily confused with raising α to the power of n). As with all of our sequential decision problems, the decision (that is, the stepsize) is determined by what is typically referred to as a stepsize rule, but is sometimes called a stepsize policy that we denote by $\alpha^\pi(S^n)$.

Normally we introduce policies later, but to help with the understanding the model, we are going to start with a basic stepsize policy called a *harmonic stepsize rule* given by

$$\alpha^{harmonic}(S^n|\theta^{step}) = \frac{\theta^{step}}{\theta^{step}+n-1}.$$

This is a simple deterministic stepsize rule, which means that we know in advance the stepsize α_n once we know n. Below we introduce a more interesting stochastic stepsize policy that requires a richer state variable.

We are also going to let $X^\pi(S^n)$ be the value of x^n determined by stepsize policy $\alpha^\pi(S^n)$.

Basic model

Exogenous information

The exogenous information is the random demand W^{n+1} for the resource (product, time or money) that we are trying to meet with our supply of product x^n. We may assume that we observe W^{n+1} directly, or we may just observe whether $x^n \leq W^{n+1}$, or $x^n > W^{n+1}$.

Transition function

The transition equation, for the setting where x is unconstrained, is given by

$$x^{n+1} = x^n + \alpha_n \nabla_x F(x^n, W^{n+1})..\tag{3.8}$$

If we require that x^n be in a set $\mathcal{X}$, we write the transition equation as

$$x^{n+1} = \Pi_{\mathcal{X}}[x^n + \alpha_n \nabla_x F(x^n, W^{n+1})].\tag{3.9}$$

where $\Pi_{\mathcal{X}}[x]$ is the projection operator that maps x into the point in $\mathcal{X}$ that is closest to x.

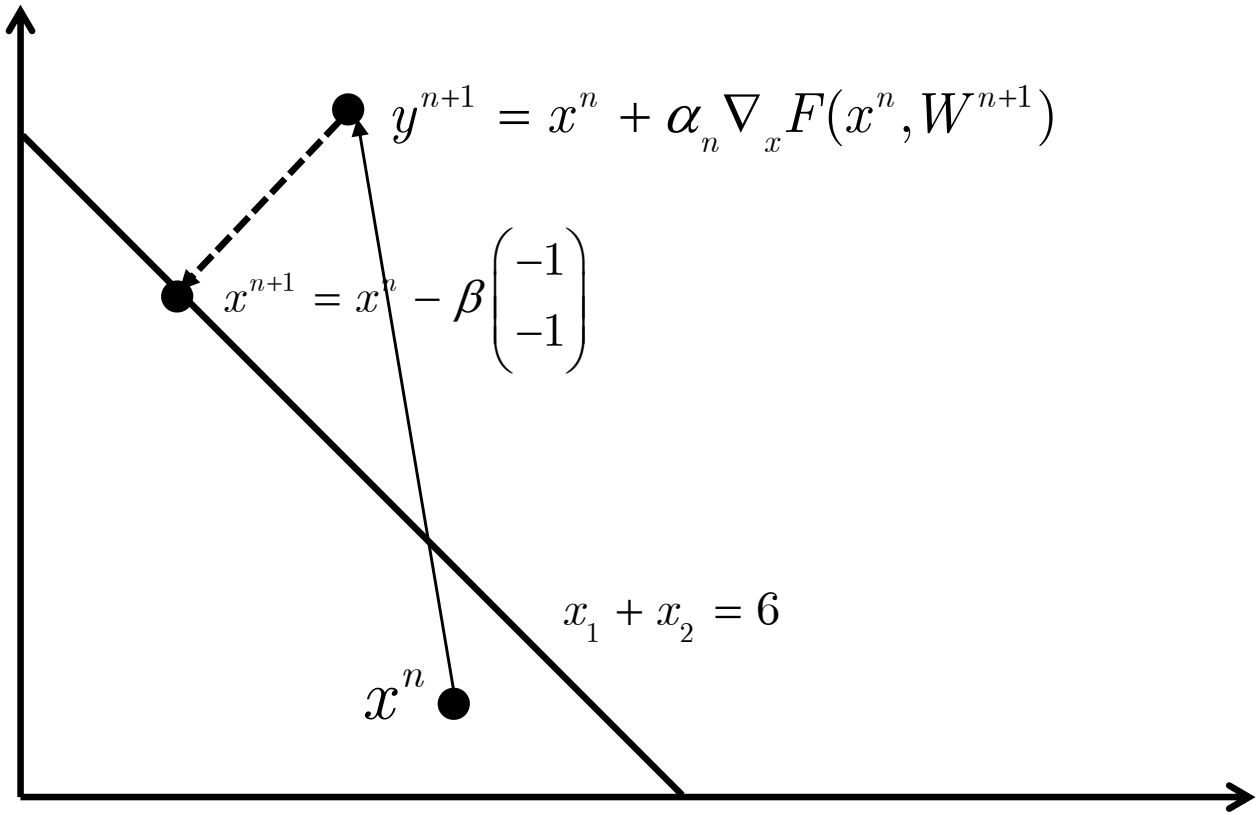
Figure 3.1 illustrates what happens when we step from points x_1 and x_2 to points y_1 and y_2 (respectively) to points outside of a set of box constraints of the form $x \leq u$. The point y_1 just violates one upper bound constraint, so we map back back to that constraint. The point y_2 violates the upper bound constraint in both dimensions. We fix this by mapping each dimension of y in order, giving us the corner point.

Figure 3.2 illustrates the process of mapping back onto a budget constraint of the form $x_1 + x_2 \leq B$. When we first step to a point y^{n+1} where $y_1^{n+1} + y_2^{n+1} > B$, we perform the projection by subtracting a constraint β from each dimension. We then find β that brings us back to the budget. This is very easy to do. If we have two dimensions, we first compute the gap $y_1^{n+1} + y_2^{n+1} - B$, then divide by the number of dimensions (two in this example). We then subtract this number from each dimension of y to get a vector whose sum equals B.

The process of subtracting β from each dimension might result in one or more dimensions of the updated y becoming negative. If this happens, just set each of the elements of the updated y to zero, which then gives us another update that is now larger than B. Lock the dimensions of the updated y that are now zero so they are no longer updated, and repeat the process.

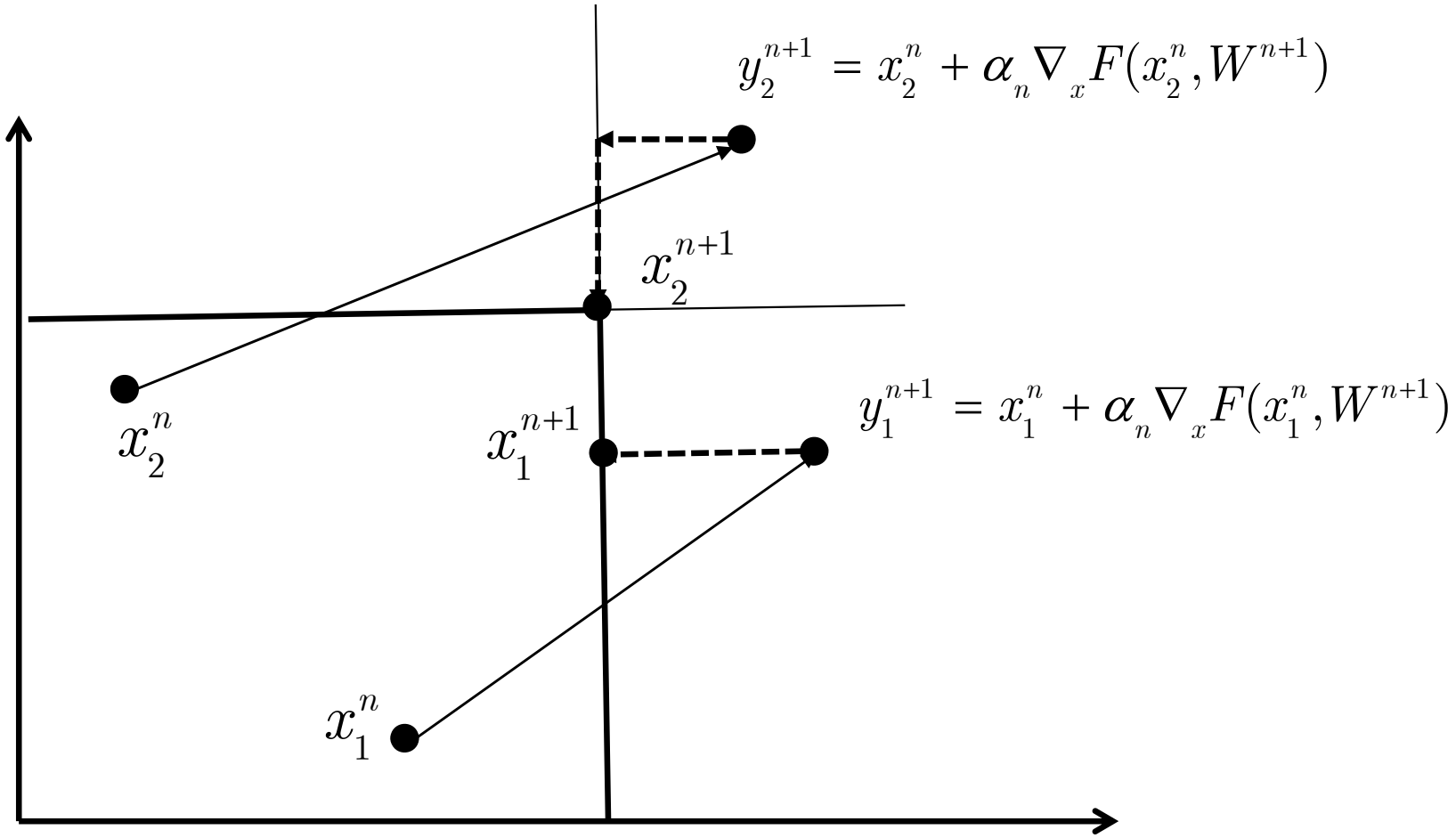
Derivative-based stochastic search

- Mapping to $x_1 + x_2 \leq 6$



Derivative-based stochastic search

- Mapping to box constraints



Basic model

- Objective function

In our market setting, at each iteration we receive a net benefit given by

$$F(x^n, W^{n+1}) \quad = \quad p \min\{x^n, W^{n+1}\} - cx^n.$$

Since we are experiencing our decisions in a market setting, we want to maximize the total reward over some horizon. This means we need to find the best policy (which in this setting means the best stepsize rule) by solving

$$\max_{\pi} \mathbb{E}\left\{\sum_{n=0}^{N-1} F(X^{\pi}(S^n), W^{n+1}) | S^0\right\}. \tag{3.10}$$

where $S^{n+1} = S^M(S^n, X^{\pi}(S^n), W^{n+1})$ describes the evolution of the algorithm. Here, π refers to the type of stepsize rule (e.g harmonic vs. Kesten), and any tunable parameters (such as θ^{step}).

Basic model

Equation (3.14) starts with an expectation over the initial state S_0 only because we might have a distribution of belief about something. For example, we might assume that the demand W is given by

$$W_t = \mu + \varepsilon_t$$

where ε_t is a random variable with some distribution. But what if we do not know μ? We might assume that μ is some fixed number that is somewhere between 10 and 20. We might assume that μ takes on one of a set of values in $(\mu_1, \ldots, \mu_K)$, each with initial probability $p_0 = (p_{0k})_{k=1}^K$. In this case, we would set $S_0 = p_0$, and the expectation $\mathbb{E}_{S_0}$ and would be computed

$$\begin{aligned}\mathbb{E}\{F(x,W)|S_0\} &= \mathbb{E}_{S_0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|\mu_0}F(x^{\pi,N},\widehat{W}) \\ &= \sum_{k=1}^{K} p_{0k}\mathbb{E}_{W^1,\ldots,W^N|\mu=\mu_k}\mathbb{E}_{\widehat{W}|\mu=\mu_k}F(x^{\pi,N},\widehat{W})\end{aligned}$$

Figure 3.2 illustrates different rates of convergence for different stepsize rules, showing $F(x^n, W^{n+1})$ as a function of the number of iterations. If we were optimizing the final reward in (3.14), we could just pick the line that is highest, which depends on the budget N. If we are optimizing the cumulative reward in equation (3.13), then we have to focus on the area under the curve, which favors rapid initial convergence.

Uncertainty modeling

Modeling uncertainty

- Observations of W_t (or W^n) might come from

  » Probability distributions.
  - This requires that we develop a mathematical model of the distribution of W. We will return to this topic later.

  » Field observations
  - We may have a dataset of observations from some exogenous source.
  - We can run our algorithms on these observations without ever developing a mathematical model.
  - This process is known as *data driven*.

3.3 UNCERTAINTY MODELING

Let $f^W(w)$ be the distribution of W (this might be discrete or continuous), with cumulative distribution function $F^W(w) = Prob[W \leq w]$. We might assume that the distribution is known with an unknown parameter. For example, imagine that W follows a Poisson distribution with mean μ given by

$$f^W(w) = \frac{\mu^w e^{-\mu}}{w!}, \quad w = 0, 1, 2, \ldots.$$

We may assume that we know μ, in which case we could solve this problem using the analytical solution given at the beginning of the chapter. Assume, instead, that μ is unknown, but with a known distribution

$$p_k^\mu = Prob[\mu = \mu_k],$$

Note that this distribution $p^\mu = (p_k^\mu)_{k=1}^K$ would be modeled in our initial state S^0.

Designing policies

Policies

- Stepsize policies
  - » Harmonic (deterministic)

A wide range of stepsize policies (often called stepsize rules) have been suggested in the literature. One of the simplest and most popular is the harmonic stepsize policy given by

$$\alpha^{harmonic}(S^n|\theta^{step}) = \frac{\theta^{step}}{\theta^{step}+n-1}.$$

Figure 3.2 illustrates the behavior of the harmonic stepsize rule for different values of θ^{step}.

Policies

- Stepsize policies
  - » Kesten (stochastic

The harmonic stepsize policy is also known as a deterministic policy, because we know its value for a given n. The challenge with deterministic policies is that they are not allowed to adapt to the data. For this reason, it is often useful to use a stochastic rule. One of the earliest and simplest examples is Kesten's rule

$$\alpha^{kesten}(S^n|\theta^{step}) = \frac{\theta^{step}}{\theta^{step} + K^n - 1},$$

where K^n is a counter that counts how many times the gradient has switched direction. We determine this by asking if the product (or inner product, if x is a vector) $(\nabla^x F(x^n, W^{n+1}))^T \nabla^x F(x^{n-1}, W^n) < 0$. If the gradient is switching directions, then it means that we are in the vicinity of the optimum and are stepping past it, so we need to reduce the stepsize. This formula is written

$$K^n = \begin{cases} K^n + 1 & \text{If } (\nabla^x F(x^n, W^{n+1}))^T \nabla^x F(x^{n-1}, W^n) < 0, \\ K^n & \text{Otherwise} \end{cases} \quad (3.12)$$

Now we have a stepsize that depends on a random variable K^n, which is why we call it a stochastic stepsize rule.

Policies

- Stepsize policies
  - » AdaGrad (stochastic)

Another stepsize rule, known as AdaGrad, is particularly well suited when x is a vector with element x_i, $i = 1, \ldots, I$. To simplify the notation a bit, let the stochastic gradient with respect to element x_i be given by

$$g_i^n = \nabla_{x_i} F(x^{n-1}, W^n).$$

Now create a $I \times I$ diagonal matrix G^n where the $(i,i)th$ element G_{ii}^n is given by

$$G_{ii}^n = \sum_{m=1}^{n} (g_i^n)^2.$$

We then set a stepsize for the ith dimension using

$$\alpha_{ni} = \frac{\eta}{(G_{ii}^n)^2 + \epsilon}, \tag{3.13}$$

where ϵ is a small number (e.g. 10^{-8} to avoid the possibility of dividing by zero).

  - » Discuss variable scaling

Policy evaluation

Policies

- Simulating a policy
  - » Walk through steps of a simulation
  - » Represent a sample path of realizations as ω

3.5 POLICY EVALUATION

We illustrate policy iteration for both forms of the objective function given above: cumulative reward, where we add up rewards incurred as we are trying each answer x^n, and final reward, where we only care about the performance of the final design, $x^{\pi,N}$.

3.5.1 Cumulative reward

For a given stepsize policy $\alpha^\pi(S^n)$, we can simulate the value of the policy. If we assume that the true mean μ is known, we just simulate a sample path $\omega = (W^1(\omega), W^2(\omega), \ldots, W^N(\omega))$. If we assume that μ is unknown but where $Prob[\mu = \mu^k] = p_k^\mu$, then we would randomly sample a truth $\mu(\omega)$ from the distribution p^μ. So, when we refer to a sample path ω, we mean a sample of anything random, whether it is μ or the realizations $W^n(\omega)$.

The performance of the policy from a single simulation is given by

$$F^\pi(\omega) = \sum_{n=1}^{N} F(X^\pi(S^n(\omega)), W^{n+1}).$$

We then will run K of these simulations, where ω^k is the kth sample path, and take an average, giving us

$$\overline{F}^\pi = \frac{1}{K} F^\pi(\omega^k).$$

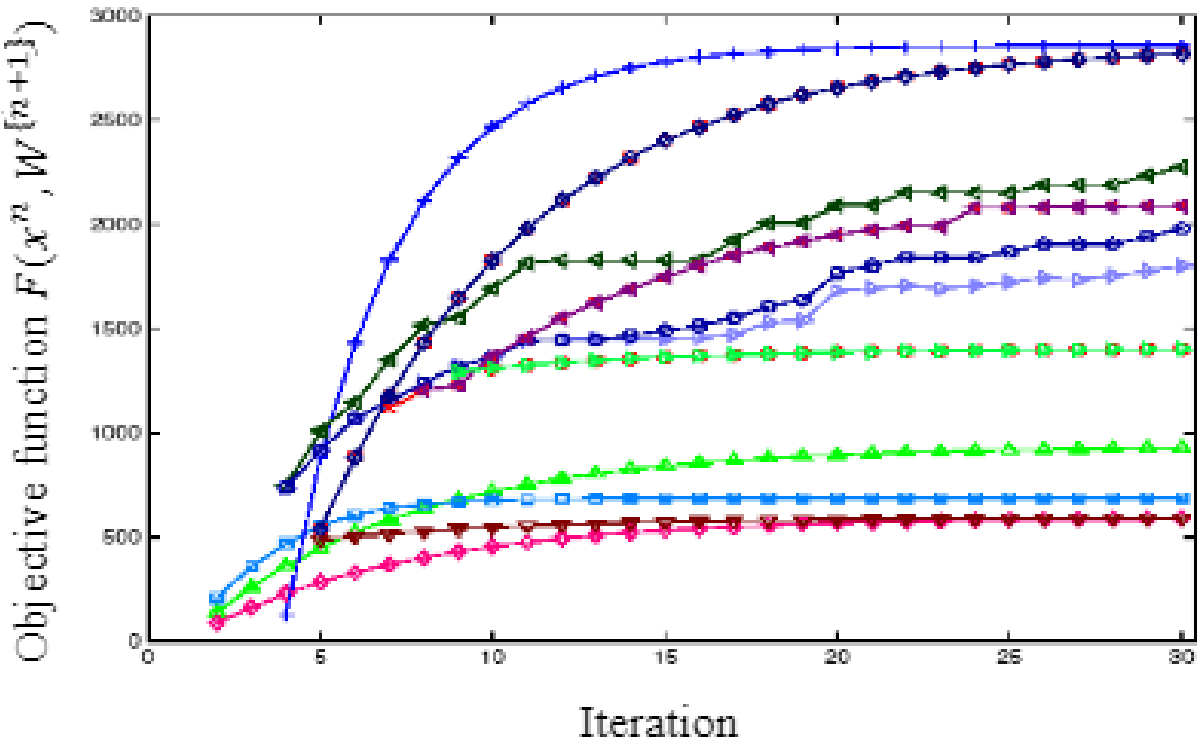


Figure 3.2 Plot of $F(x^n, W^{n+1})$ for different stepsize rules, illustrating different rates of convergence.

3.5.2 Final reward

We might pose the problem that we are able to simulate this problem in the computer using observations of W drawn from some distribution which we would have to assume is known (in order to draw sample realizations). Even in this setting, we cannot run an infinite number of iterations. Assume we have a budget of N experiments, and let $x^{\pi,N}$ be the solution that our algorithm produces using stepsize rule π. The solution $x^{\pi,N}$ is a random variable, because it depends on $W^1, \ldots, W^N$. Once we obtain $x^{\pi,N}$, we need to test it, and we let $\widehat{W}$ be the random variable we use for testing. The problem of finding the best policy (the best stepsize rule) would then be stated

$$\max_{\pi} \mathbb{E}\{F(x, W)|S_0\} = \mathbb{E}_{S_0}\mathbb{E}_{W^1,\ldots,W^N|S_0}\mathbb{E}_{\widehat{W}|S_0}F(x^{\pi,N}, \widehat{W}). \tag{3.13}$$

Equation (3.13) starts with an expectation over the initial state S_0 only because we might have a distribution of belief about something. For example, we might assume that the demand W is given by

$$W_t = \mu + \varepsilon_t$$

where ε_t is a random variable with some distribution. But what if we do not know μ? We might assume that μ is some fixed number that is somewhere between 10 and 20. We might assume that μ takes on one of a set of values in $(\mu_1, \ldots, \mu_K)$, each with probability $p^\mu = (p^\mu_k)_{k=1}^K$.

$$\begin{aligned}
\mathbb{E}\{F(x, W)|S_0\} &= \mathbb{E}_{S_0}\mathbb{E}_{W^1,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N}, \widehat{W}) \\
\mathbb{E}\{F(x, W)|S_0\} &= \mathbb{E}_{\mu}\mathbb{E}_{W^1,\ldots,W^N|\mu}\mathbb{E}_{\widehat{W}|\mu}F(x^{\pi,N}, \widehat{W}) \\
&= \sum_{k=1}^{K} p^\mu_k \mathbb{E}_{W^1,\ldots,W^N|\mu=\mu_k}\mathbb{E}_{\widehat{W}|\mu=\mu_k}F(x^{\pi,N}, \widehat{W})
\end{aligned}$$

Extensions

Extensions

3.6 EXTENSIONS

1) Imagine that we do not know μ, but let's assume that μ can take on one of the values $(\mu_1, \mu_2, \ldots, \mu_K)$. Let H^n be the history of observations up through the nth experiment, and let H^0 be the initial empty history. We assume we start with an initial prior probability on μ that we write as

$$p_k^0 = Prob[\mu = \mu_k | H^0].$$

After we have observed $W^1, \ldots, W^n$, we would write our updated distribution as

$$p_k^n = Prob[\mu = \mu_k | H^n].$$

We can update $p^n = (p_k^n)_{k=1}^K$ using Bayes theorem

$$\begin{aligned} p_k^{n+1} &= Prob\mu = \mu_k | W^{n+1} = w, H^n && (3.14) \\ &= \frac{Prob[W^{n+1} = w | \mu = \mu_k, H^n] Prob[\mu = \mu_k | H^n]}{Prob[W^{n+1} = w | H^n]} && (3.15) \\ &= \frac{Prob[W^{n+1} = w | \mu = \mu_k] p_k^n}{Prob[W^{n+1} = w | H^n]} && (3.16) \end{aligned}$$

where

$$Prob[W^{n+1} = w | H^n] = \sum_{k=1}^{K} Prob[W^{n+1} = w | \mu = \mu_k] p_k^n.$$

With this extension to the basic model, we have two probability distributions: the belief on the true mean μ, and the random demand W given μ. To include this extension, we would have to insert p^n into our state variable, so we would write

$$S^n = (x^n, p^n).$$

Extensions

2) Imagine that our problem is to purchase a commodity (such as oil or natural gas) in month n to be used during month $n+1$. We purchase the commodity at a unit cost c, and sell it up to an unknown demand D^{n+1} at an unknown price p^{n+1}. We would write our objective for this problem as

$$F^n(x^n, W^{n+1}) = p^{n+1} \min(x^n, D^{n+1}) - cx^n. \tag{3.17}$$

For now, assume that p^{n+1} is independent of p^n, and that D^{n+1} is independent of D^n.

a) For the model in equation (3.17), give the state variable S^n and the exogenous information variable W^n.

b) Give the stochastic gradient algorithm for this problem, and show that it is basically the same as the one when price was constant.

Extensions

3) Now assume that our objective is to optimize

$$F^n(x^n, W^{n+1}) = p^n \min(x^n, D^{n+1}) - cx^n. \quad (3.18)$$

The only difference between equations (3.18) and (3.17) is that now we get to see the price p^n *before* we choose our decision x^n. We know this by how the price is indexed.

a) For the model in equation (3.18), give the state variable S^n and the exogenous information variable W^n.

b) Give the stochastic gradient algorithm for this problem. Unlike the previous problem, this gradient will be a function of p^n.

The situation where the gradient depends on the price p^n is a fairly significant complication. What is happening here is that instead of trying to find an optimal solution x^* (or more precisely, $x^{\pi,N}$), we are trying to find a function $x^{\pi,N}(p)$.

The trick here is to pick a functional form for $x^{\pi,N}(p)$. We suggest two alternatives:

Lookup table - Even if p is continuous, we can discretize it into a series of discrete prices $p_1, \ldots, p_K$, where we pick the value p_k closest to a price p^n. Call this price p_k^n. Now think of a stochastic gradient algorithm indexed by whatever p_k is nearest to p^n. We then use the stochastic gradient to update $x^n(p_k^n)$ using

$$x^{n+1}(p_k^n) = x^n(p_k^n) + \alpha_n \nabla_x F^n(x^n, W^{n+1}).$$

Of course, we do not want to discretize p too finely. If we discretize prices into, say, 100 ranges, then this means we are trying to find 100 order quantities $x^{\pi,N}(p)$, which would be quite slow.

Extensions

Parametric model - Now imagine that we think we can represent the order quantity $x^{\pi,N}(p)$ as a parametric function

$$x^{\pi,N}(p|\theta) = \theta_0 + \theta_1 p + \theta_2 p^{\theta_3}. \quad (3.19)$$

When we use a parametric function such as this, we are no longer trying to find the order quantity $x^{\pi,N}$; instead, we are trying to find θ that determines the function (in this case, (3.19)). Our stochastic gradient algorithm now becomes

$$\begin{aligned}\theta^{n+1} &= \theta^n + \alpha_n \frac{dF^n(x^{\pi,N}(p^n|\theta^n), W^{n+1})}{d\theta}, \\ &= \theta^n + \alpha_n \frac{dF^n(x^{\pi,N}(p^n|\theta^n), W^{n+1})}{dx} \frac{dx^{\pi,N}(p^n|\theta^n)}{d\theta},\end{aligned}$$

Remember that θ^n is a four-element column vector, while x^n is a scalar. The first derivative is our original stochastic gradient

$$\frac{dF^n(x^{\pi,N}(p^n|\theta^n), W^{n+1})}{dx} = \begin{cases} p - c & x \le W \\ -c & x > W \end{cases}$$

The second derivative is computed directly from the policy (3.19), which is given by

$$\begin{aligned}\frac{dx^{\pi,N}(p^n|\theta^n)}{d\theta} &= \begin{pmatrix} \frac{dx^{\pi,N}(p^n|\theta^n)}{d\theta_0} \\ \frac{dx^{\pi,N}(p^n|\theta^n)}{d\theta_1} \\ \frac{dx^{\pi,N}(p^n|\theta^n)}{d\theta_2} \\ \frac{dx^{\pi,N}(p^n|\theta^n)}{d\theta_3} \end{pmatrix} \\ &= \begin{pmatrix} 1 \\ p^n \\ (p^n)^{\theta_3} \\ \theta_2 (p^n)^{\theta_3} \ln p^n \end{pmatrix}.\end{aligned}$$

Week 1 - Wednesday

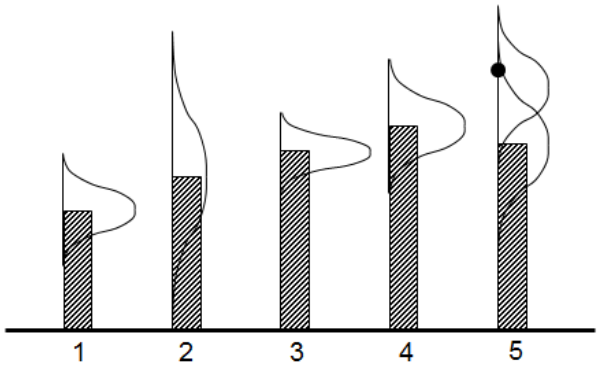
Statistical modeling

- Three core approximation architectures
  - » Lookup tables
  - » Parametric models
    - Linear in the parameters
    - Nonlinear (in the parameters)
  - » Nonparametric

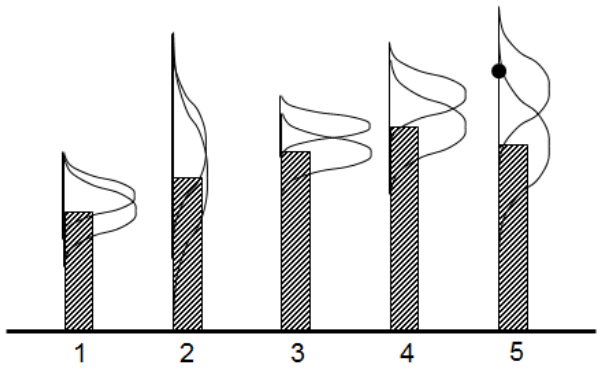
Approximating strategies

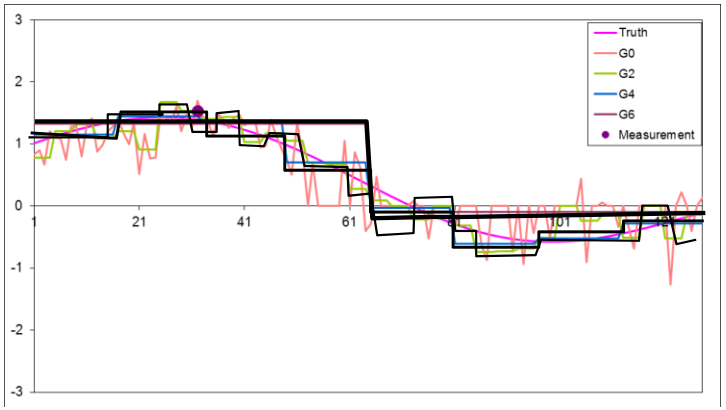
- Lookup tables
  - » Independent beliefs

$$\mu_x^n \approx \mathbb{E}F(x,W) \quad x \in \{x_1,...,x_M\}$$



  - » Correlated beliefs
    - A few dozen observations can teach us about thousands of points.
  - » Hierarchical models
    - Create beliefs at different levels of aggregation and then use weighted combinations



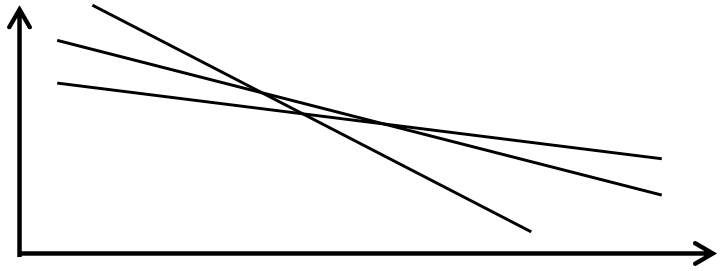


Approximating strategies

- Parametric models
  - » Linear models

$$F(x|\theta) = \sum_{f \in F} \theta_f \phi_f(x)$$

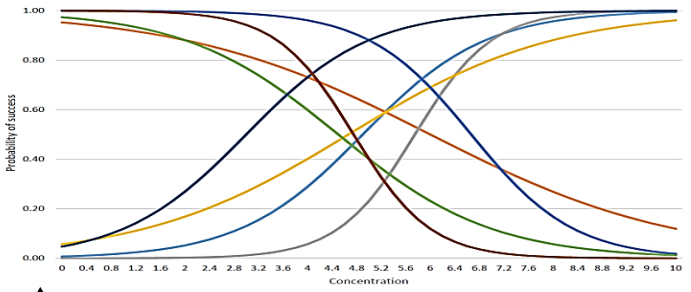
    - Might include sparse-additive, where many parameters are zero.

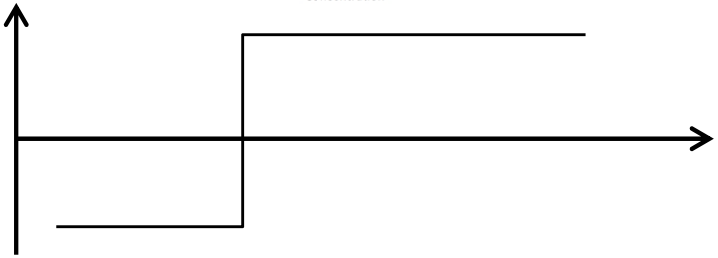


  - » Nonlinear models

$$F(x|\theta) = \frac{e^{\theta_0 + \theta_1 \phi_1(x) + \ldots}}{1 + e^{\theta_0 + \theta_1 \phi_1(x) + \ldots}}$$

$$X^{\pi}(S_t|\theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

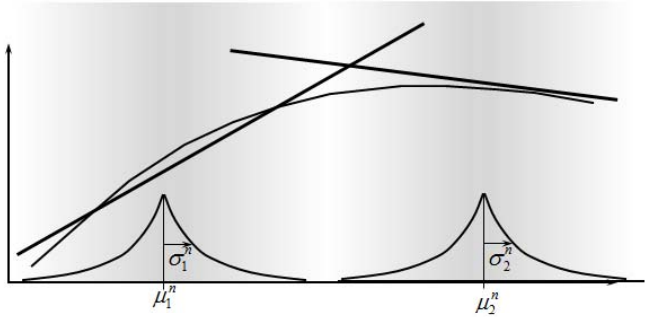


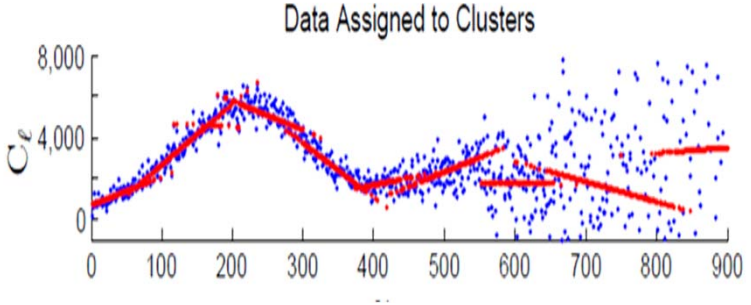


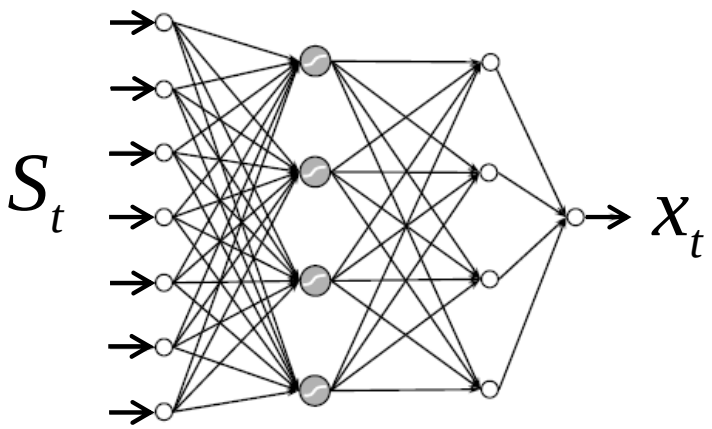
    - (Shallow) Neural networks, …

Approximating strategies

- Nonparametric models
  - » Kernel regression
    - Weighted average of neighboring points
    - Limited to low dimensional problems
  - » Locally linear methods
    - Dirichlet process mixtures
    - Radial basis functions
  - » Splines
  - » Support vector machines
  - » Deep neural networks







Lookup tables

Frequentist

Bayesian – Independent

Bayesian - correlated

Lookup tables

- Frequentist updating - Batch

variance of W. We know from elementary statistics that we can write μ^n and $\hat{\sigma}^{2,n}$ using

$$\mu^n = \frac{1}{n}\sum_{m=1}^{n} W^m \tag{3.4}$$

$$\hat{\sigma}^{2,n} = \frac{1}{n-1}\sum_{m=1}^{n}(W^m - \mu^n)^2. \tag{3.5}$$

The estimate μ^n is a random variable (in the frequentist view) because it is computed from other random variables, namely $W^1, W^2, \ldots, W^n$. Imagine if we had 100 people each choose a sample of n observations of W. We would obtain 100 different estimates of μ^n, reflecting the variation in our observations of W. The best estimate of the variance of the estimator μ^n is easily found to be

$$\bar{\sigma}^{2,n} = \frac{1}{n}\hat{\sigma}^{2,n}.$$

The estimate μ^n is a random variable (in the frequentist view) because it is computed from other random variables, namely $W^1, W^2, \ldots, W^n$. Imagine if we had 100 people each choose a sample of n observations of W. We would obtain 100 different estimates of μ^n, reflecting the variation in our observations of W. The best estimate of the variance of the estimator μ^n is easily found to be

$$\bar{\sigma}^{2,n} = \frac{1}{n}\hat{\sigma}^{2,n}.$$

Note that as $n \to \infty$, $\bar{\sigma}^{2,n} \to 0$, but $\hat{\sigma}^{2,n} \to \sigma^2$ where σ^2 is the true variance of W. If σ^2 is known, there would be no need to compute $\hat{\sigma}^{2,n}$ and $\bar{\sigma}^{2,n}$ would be given as above with $\hat{\sigma}^{2,n} = \sigma^2$

Lookup tables

Frequentist updating - Recursive

The estimate μ^n is a random variable (in the frequentist view) because it is computed from other random variables, namely $W^1, W^2, \ldots, W^n$. Imagine if we had 100 people each choose a sample of n observations of W. We would obtain 100 different estimates of μ^n, reflecting the variation in our observations of W. The best estimate of the variance of the estimator μ^n is easily found to be

$$\bar{\sigma}^{2,n} = \frac{1}{n}\hat{\sigma}^{2,n}.$$

Note that as $n \to \infty$, $\bar{\sigma}^{2,n} \to 0$, but $\hat{\sigma}^{2,n} \to \sigma^2$ where σ^2 is the true variance of W. If σ^2 is known, there would be no need to compute $\hat{\sigma}^{2,n}$ and $\bar{\sigma}^{2,n}$ would be given as above with $\hat{\sigma}^{2,n} = \sigma^2$.

We can write these expressions recursively using

$$\mu^n = \left(1 - \frac{1}{n}\right)\mu^{n-1} + \frac{1}{n}W^n, \tag{3.6}$$

$$\hat{\sigma}^{2,n} = \begin{cases} \frac{1}{n}(W^n - \mu^{n-1})^2 & n = 2, \\ \frac{n-2}{n-1}\hat{\sigma}^{2,n-1} + \frac{1}{n}(W^n - \mu^{n-1})^2 & n > 2. \end{cases} \tag{3.7}$$

We will often speak of our "state of knowledge" which captures what we know about the parameters we are trying to estimate. Given our observations, we would write our state of knowledge as

$$K^n = \left(\mu^n, \hat{\sigma}^{2,n}, n\right).$$

Equations (3.6) and (3.7) describe how our state of knowledge evolves over time.

Lookup tables

- Bayesian – independent beliefs
  - » Discuss Bayesian setting

$$\beta^W = \frac{1}{\sigma_W^2}.$$

After observing W^{n+1}, the updated mean and precision of our estimate of μ is given by

$$\mu^{n+1} = \frac{\beta^n \mu^n + \beta^W W^{n+1}}{\beta^n + \beta^W}, \tag{3.8}$$

$$\beta^{n+1} = \beta^n + \beta^W. \tag{3.9}$$

Equation (3.8) can be written more compactly as

$$\mu^{n+1} = (\beta^{n+1})^{-1} \left(\beta^n \mu^n + \beta^W W^{n+1}\right). \tag{3.10}$$

There is another way of expressing the updating which provides insight into the structure of the flow of information. First define

$\left(\mu^n, \beta^n\right)$ is called a "sufficient statistic" - same as a state variable.

$$\tilde{\sigma}^{2,n} = Var^n[\mu^{n+1}] \tag{3.11}$$

$$= Var^n[\mu^{n+1} - \mu^n] \tag{3.12}$$

where $Var^n[\cdot] = Var[\cdot \mid W^1, ..., W^n]$ denotes the variance of the argument given the information we have through n observations. For example,

$$Var^n[\mu^n] = 0$$

Lookup tables

- Bayesian – independent beliefs

The parameter $\tilde{\sigma}^{2,n}$ can be described as the variance of μ^{n+1} given the information we have collected through iteration n, which means the only random variable is W^{n+1}. Equivalently, $\tilde{\sigma}^{2,n}$ can be thought of as the *change* in the variance of μ^n as a result of the observation of W^{n+1}. Equation (3.12) is an equivalent statement since, given the information collected up through iteration n, μ^n is deterministic and is therefore a constant. We use equation (3.12) to offer the interpretation that $\tilde{\sigma}^{2,n}$ is the *change* in the variance of our estimate of the mean of μ.

It is possible to write $\tilde{\sigma}^{2,n}$ in different ways. For example, we can show that

$$\tilde{\sigma}^{2,n} = \sigma^{2,n} - \sigma^{2,n+1} \tag{3.13}$$

$$= \frac{(\sigma^{2,n})}{1+\sigma_W^2/\sigma^{2,n}} \tag{3.14}$$

$$= (\beta^n)^{-1} - (\beta^n + \beta^W)^{-1}. \tag{3.15}$$

Equations (3.14) and (3.15) come directly from (3.13) and (3.9), using either variances or precisions.

Just as we let $Var^n[\cdot]$ be the variance given what we know after n measurements, let $\mathbb{E}^n$ be the expectation given what we know after n measurements. That is, if $W^1, \ldots, W^n$ are the first n measurements, we can write

$$\mathbb{E}^n \mu^{n+1} \equiv \mathbb{E}(\mu^{n+1}|W^1, \ldots, W^n) = \mu^n.$$

Lookup tables

- Bayesian updating – correlated beliefs

■ **EXAMPLE 3.1**

We are interested in finding the price of a product that maximizes total revenue. We believe that the function $R(p)$ that relates revenue to price is continuous. Assume that we set a price p^n and observe revenue R^{n+1} that is higher than we had expected. If we raise our estimate of the function $R(p)$ at the price p^n, our beliefs about the revenue at nearby prices should be higher.

■ **EXAMPLE 3.2**

We choose five people for the starting lineup of our basketball team and observe total scoring for one period. We are trying to decide if this group of five people is better than another lineup that includes three from the same group with two different people. If the scoring of these five people is higher than we had expected, we would probably raise our belief about the other group, since there are three people in common.

■ **EXAMPLE 3.3**

A physician is trying to treat diabetes using a treatment of three drugs, where she observes the drop in blood sugar from a course of a particular treatment. If one treatment produces a better-than-expected response, this would also increase our belief of the response from other treatments that have one or two drugs in common.

Lookup tables

- Bayesian updating – correlated beliefs

Let μ_x^n be our belief about alternative x after n measurements. Now let

$$Cov^n(\mu_x, \mu_y) \quad = \quad \text{the covariance in our belief about } \mu_x \text{ and } \mu_y.$$

We let Σ^n be the covariance matrix, with element $\Sigma_{xy}^n = Cov^n(\mu_x, \mu_y)$. Just as we defined the precision β_x^n to be the reciprocal of the variance, we are going to define the precision matrix B^n to be

$$B^n = (\Sigma^n)^{-1}.$$

Let e_x be a column vector of zeroes with a 1 for element x, and as before we let W^{n+1} be the (scalar) observation when we decide to measure alternative x. We could label W^{n+1} as W_x^{n+1} to make the dependence on the alternative more explicit. For this discussion, we are going to use the notation that we choose to measure x^n and the resulting observation is W^{n+1}. If we choose to measure x^n, we can also interpret the observation as a column vector given by $W^{n+1}e_{x^n}$. Keeping in mind that μ^n is a column vector of our beliefs about the expectation of μ, the Bayesian equation for updating this vector in the presence of correlated beliefs is given by

$$\mu^{n+1} \quad = \quad (B^{n+1})^{-1}\left(B^n\mu^n + \beta^W W^{n+1}e_{x^n}\right), \tag{3.17}$$

where B^{n+1} is given by

$$B^{n+1} \quad = \quad (B^n + \beta^W e_{x^n}(e_{x^n})^T). \tag{3.18}$$

Lookup tables

- Bayesian updating – correlated beliefs

Let $\lambda^W = \sigma_W^2 = 1/\beta^W$ be the variance of our measurement W^{n+1}. We are going to simplify our notation by assuming that our measurement variance is the same across all alternatives x, but if this is not the case, we can replace λ^W with λ_x^W throughout. Using the Sherman-Morrison formula, and letting $x = x^n$, we can rewrite the updating equations as

$$\mu^{n+1}(x) = \mu^n + \frac{W^{n+1} - \mu_x^n}{\lambda^W + \Sigma_{xx}^n}\Sigma^n e_x, \tag{3.20}$$

$$\Sigma^{n+1}(x) = \Sigma^n - \frac{\Sigma^n e_x (e_x)^T \Sigma^n}{\lambda^W + \Sigma_{xx}^n}. \tag{3.21}$$

where we express the dependence of $\mu^{n+1}(x)$ and $\Sigma^{n+1}(x)$ on the alternative x which we have chosen to measure.

To illustrate, assume that we have three alternatives with mean vector

$$\mu^n = \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix}.$$

Assume that $\lambda^W = 9$ and that our covariance matrix Σ^n is given by

$$\Sigma^n = \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix}.$$

Lookup tables

- Bayesian updating – correlated beliefs

Assume that we choose to measure $x = 3$ and observe $W^{n+1} = W_3^{n+1} = 19$. Applying equation (3.20), we update the means of our beliefs using

$$\begin{aligned}
\mu^{n+1}(3) &= \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix} + \frac{19-22}{9+15} \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 20 \\ 16 \\ 22 \end{bmatrix} + \frac{-3}{24} \begin{bmatrix} 3 \\ 4 \\ 15 \end{bmatrix} \\
&= \begin{bmatrix} 19.625 \\ 15.500 \\ 20.125 \end{bmatrix}.
\end{aligned}$$

Lookup tables

- Bayesian updating – correlated beliefs

The update of the covariance matrix is computed using

$$\Sigma^{n+1}(3) = \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{\begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} [0\ 0\ 1] \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix}}{9+15}$$

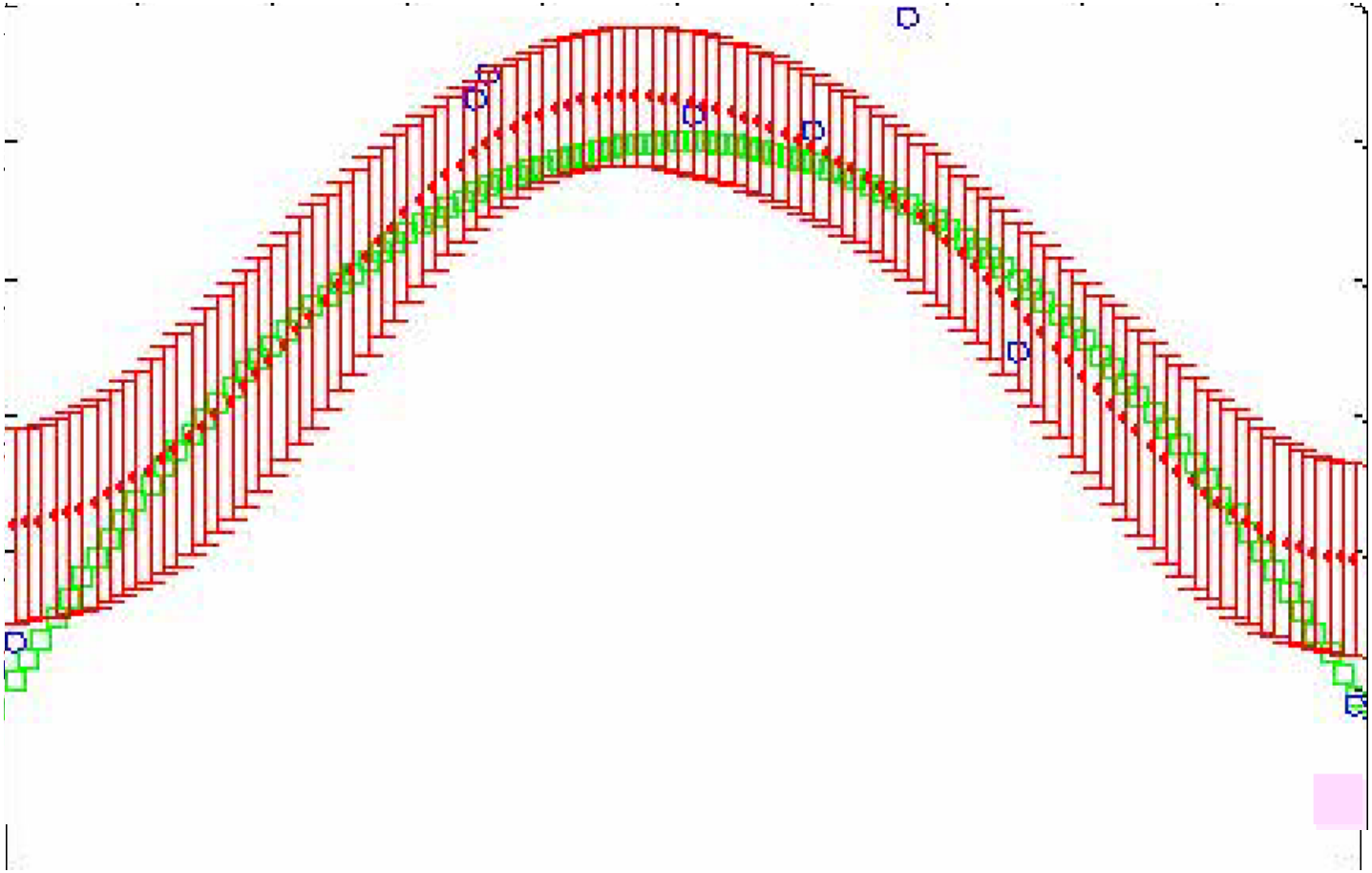
$$= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 3 \\ 4 \\ 15 \end{bmatrix} [3\ 4\ 15]$$

$$= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \frac{1}{24} \begin{bmatrix} 9 & 12 & 45 \\ 12 & 16 & 60 \\ 45 & 60 & 225 \end{bmatrix}$$

$$= \begin{bmatrix} 12 & 6 & 3 \\ 6 & 7 & 4 \\ 3 & 4 & 15 \end{bmatrix} - \begin{bmatrix} 0.375 & 0.500 & 1.875 \\ 0.500 & 0.667 & 2.500 \\ 1.875 & 2.500 & 9.375 \end{bmatrix}$$

$$= \begin{bmatrix} 11.625 & 5.500 & 1.125 \\ 5.500 & 6.333 & 1.500 \\ 1.125 & 1.500 & 5.625 \end{bmatrix}.$$

Correlated beliefs

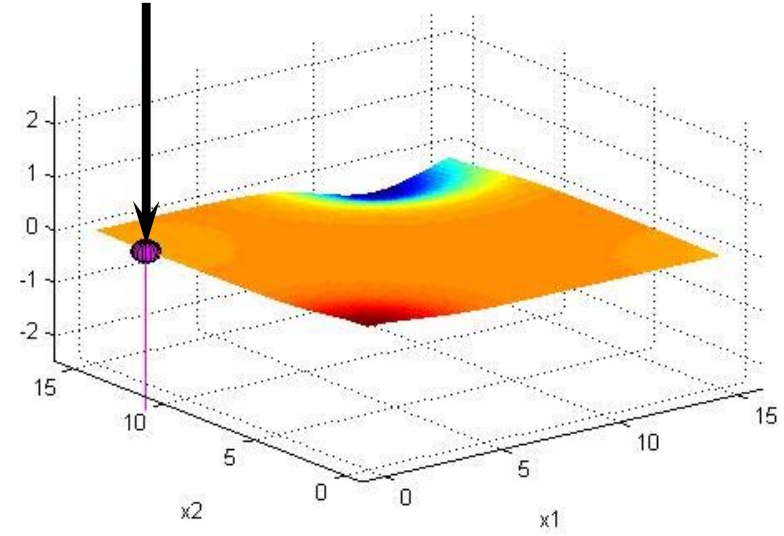


Correlated beliefs

- *After four measurements:*

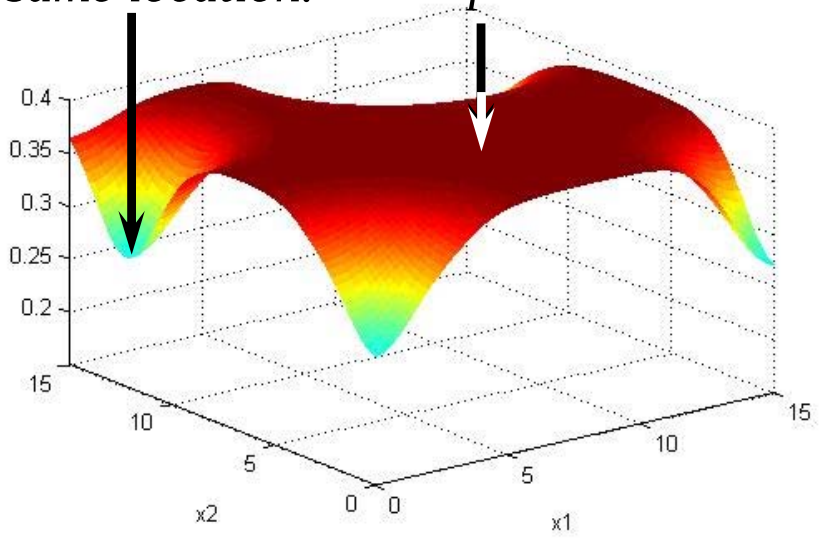
Estimated concentration

Measurement



Knowledge gradient

Value of another measurement at same location. New optimum

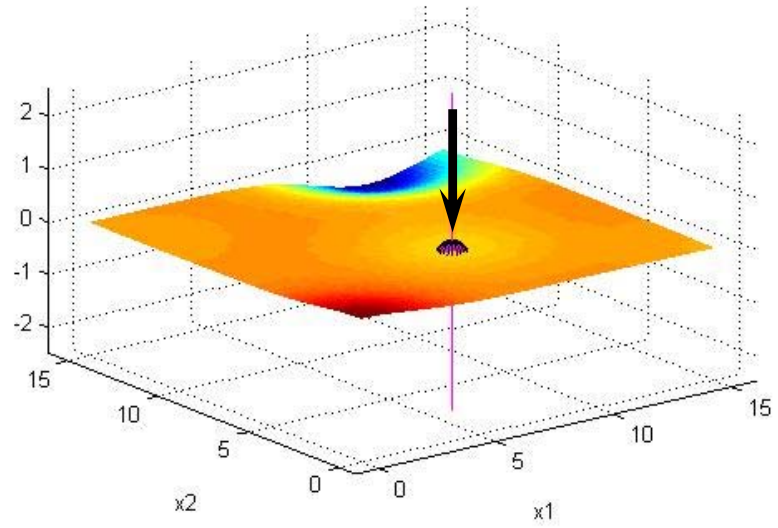


» *Whenever we measure at a point, the value of another measurement at the same point goes down. The knowledge gradient guides us to measuring areas of high uncertainty.*

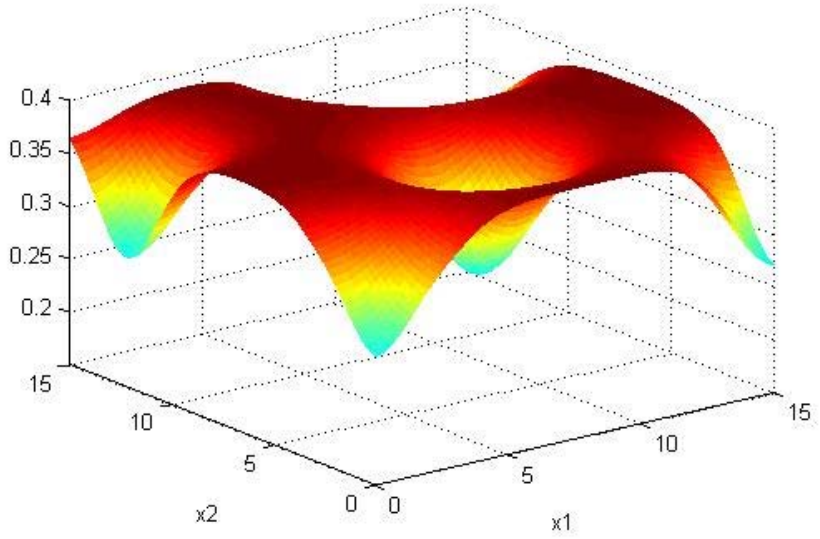
Correlated beliefs

- *After five measurements:*

Estimated concentration



Knowledge gradient



Linear models

Linear models

- Examples:

$$y = \theta_0 + \sum_{i=1}^{I} \theta_i x_i + \varepsilon. \tag{3.38}$$

The variables x_i might be called independent variables, explanatory variables, or covariates, depending on the community. In dynamic programming where we want to estimate a value function $V^\pi(S_t)$, we might write

$$\overline{V}(S|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S),$$

In fact, if we write our policy of the form

$$X^\pi(S_t|\theta) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S_t),$$

we would refer to $X^\pi(S_t|\theta)$ as an *affine policy* ("affine" is just a fancy name for linear, by which we mean linear in θ).

» Independent variables, covariates, basis functions

Linear models

3.4.1 Linear regression review

Let y^n be the n^{th} observation of our dependent variable (what we are trying to predict) based on the observation $(x_1^n, x_2^n, \ldots, x_I^n)$ of our independent (or explanatory) variables (the x_i are equivalent to the basis functions we used earlier). Our goal is to estimate a parameter vector θ that solves

$$\min_{\theta} \sum_{m=1}^{n} \left(y^m - (\theta_0 + \sum_{i=1}^{I} \theta_i x_i^m) \right)^2 . \tag{3.39}$$

This is the standard linear regression problem. Let μ^n be the optimal solution for this problem. Throughout this section, we assume that the underlying process from which the observations y^n are drawn is stationary (an assumption that is almost never satisfied in approximate dynamic programming).

If we define $x_0 = 1$, we let

$$x^n = \begin{pmatrix} x_0^n \\ x_1^n \\ \vdots \\ x_I^n \end{pmatrix}$$

Linear models

Letting θ be the column vector of parameters, we can write our model as

$$y = \theta^T x + \varepsilon.$$

We assume that the errors $(\varepsilon^1, \ldots, \varepsilon^n)$ are independent and identically distributed. We do not know the parameter vector θ, so we replace it with an estimate $\bar{\theta}$ which gives us the predictive formula

$$\bar{y}^n = (\bar{\theta})^T x^n,$$

where $\bar{y}^n$ is our predictor of y^{n+1}. Our prediction error is

$$\hat{\varepsilon}^n = y^n - (\bar{\theta})^T x^n.$$

Our goal is to choose θ to minimize the mean squared error

$$\min_{\theta} \sum_{m=1}^{n} (y^m - \theta^T x^m)^2. \tag{3.40}$$

It is well known that this can be solved very simply. Let X^n be the n by $I+1$ matrix

$$X^n = \begin{pmatrix} x_0^1 & x_1^1 & & x_I^1 \\ x_0^2 & x_1^2 & & x_I^2 \\ \vdots & \vdots & \cdots & \vdots \\ x_0^n & x_1^n & & x_I^n \end{pmatrix}.$$

Next, denote the vector of observations of the dependent variable as

$$Y^n = \begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^n \end{pmatrix}.$$

The optimal parameter vector $\bar{\theta}$ (after n observations) is given by

$$\bar{\theta} = [(X^n)^T X^n]^{-1} (X^n)^T Y^n. \tag{3.41}$$

Linear models

- Recursive least squares-Stationary data

3.5.1 Recursive least squares for stationary data

In the setting of adaptive algorithms in stochastic optimization, estimating the coefficient vector θ using batch methods such as equation (3.46) would be very expensive. Fortunately, it is possible to compute these formulas recursively. The updating equation for θ is

$$\theta^n = \theta^{n-1} - H^n \phi^n \hat{\varepsilon}^n, \tag{3.47}$$

where H^n is a matrix computed using

$$H^n = \frac{1}{\gamma^n} B^{n-1}. \tag{3.48}$$

The error $\hat{\varepsilon}^n$ is computed using

$$\hat{\varepsilon}^n = \overline{V}_s(\theta^{n-1}) - \hat{v}^n. \tag{3.49}$$

Note that it is common in statistics to compute the error in a regression using "actual minus predicted" while we are using "predicted minus actual" (see also equation (17.21) above). Our sign convention is motivated by the derivation from first principles of optimization. B^{n-1} is an $|\mathcal{F}|$ by $|\mathcal{F}|$ matrix which is updated recursively using

$$B^n = B^{n-1} - \frac{1}{\gamma^n}(B^{n-1}\phi^n(\phi^n)^T B^{n-1}). \tag{3.50}$$

γ^n is a scalar computed using

$$\gamma^n = 1 + (\phi^n)^T B^{n-1} \phi^n. \tag{3.51}$$

Nonlinear models

Sampled nonlinear models

- Nonlinear models
  - » This means “nonlinear in the parameters”
  - » Examples:
    - Logistic regression

$$R(x \mid \theta_i) = \theta_{i0} \frac{e^{-\theta_{i1}(x-\theta_{i2})}}{1+e^{-\theta_1(x-\theta_{i2})}}$$

    - Parametric rules (energy storage)

$$X^{\pi}(S_t \mid \theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

- Fitting a nonlinear model.
  - » Solving the fitting problem of a linear model is easy

$$\min_{\theta} \sum_{m=1}^{n} \left(y^m - (\theta_0 + \sum_{i=1}^{I} \theta_i x_i^m) \right)^2 . \tag{3.39}$$

  - » Solving nonlinear problems is *much* harder (see next slide).

Maximum likelihood estimation for nonlinear problems

The most general method for estimating nonlinear models is known as maximum likelihood estimation. Let $f(x|\theta)$ the function given θ, and assume that we observe

$$y = f(x|\theta) + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$ is the error with density

$$f^{\epsilon}(w) = \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{w^2}{2\sigma^2}.$$

Now imagine that we have a set of observations $(y^n, x^n)_{n=1}^N$. The likelihood of observing $(y^n)_{n=1}^N$ is given by

$$L(y|x, \theta) = \Pi_{n=1}^N \exp \frac{(y^n - f(x^n|\theta))^2}{2\sigma^2}.$$

It is common to use the log likelihood $\mathcal{L}(y|x, \theta) = \log L(y|x, \theta)$, which gives us

$$\mathcal{L}(y|x, \theta) = \sum_{n=1}^{N} \frac{1}{\sqrt{2\pi}\sigma}(y^n - f(x^n|\theta))^2, \tag{3.68}$$

where we can, of course, drop the leading constant $\frac{1}{\sqrt{2\pi}\sigma}$ when maximizing $\mathcal{L}(y|x, \theta)$.

- Fitting a nonlinear model using sampled beliefs:
  - » Assume our nonlinear model is a logistic function:

$$f^W(x|\theta) = \frac{e^{-\theta_1(x-\theta_2)}}{1+e^{-\theta_1(x-\theta_2)}} = \text{Prob[Accept price } x\text{]}$$

  - » We charge a price y^n and then observe the customer response of $W^{n+1} = 0/1$ (decline or accept the price).
  - » For this course, we are going to handle nonlinear problems using the technique of a *sampled belief model*.
    - Let $f(x|\theta)$ be our nonlinear function, and assume that $\theta \in \{\theta_1, \theta_2, \ldots, \theta_K\}$.
    - Let $p_k^n = Prob[\theta = \theta_k\}$ after n iterations (or, given the history H^n).
    - Assume we observe the customer accepting our price, so $W^{n+1} = 1$.

- Fitting a nonlinear model using sampled beliefs:
  - » Update probabilities using Bayes theorem:

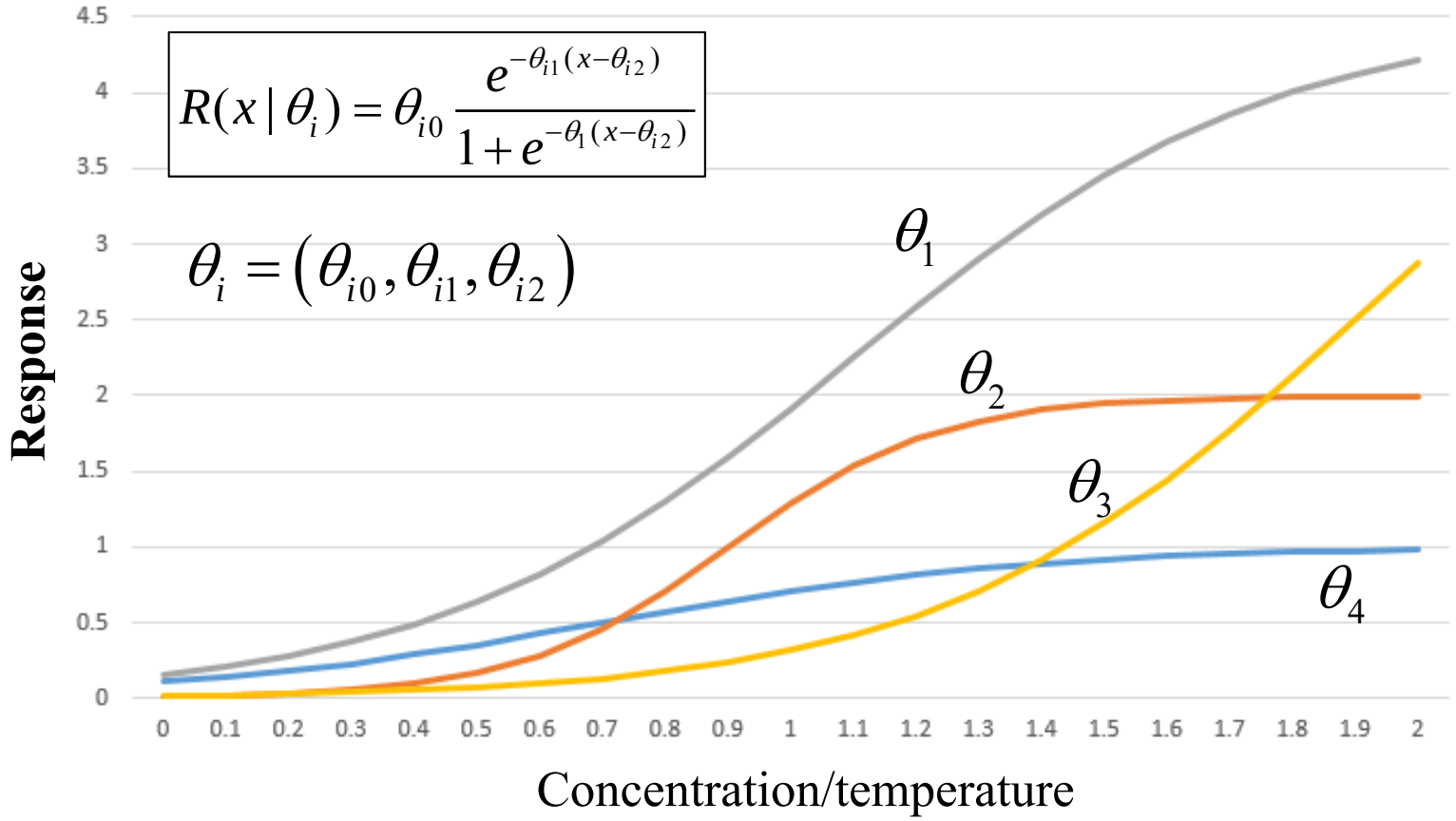
$$
\begin{aligned}
p_k^{n+1} &= \text{Prob}[\theta=\theta_k \,|\, W^{n+1}=1, H^n] \\
&= \frac{\text{Prob}[W^{n+1}=1 \,|\, \theta=\theta_k, H^n]\text{Prob}[\theta=\theta_k \,|\, H^n]}{\text{Prob}[W^{n+1}=1 \,|\, H^n]} \\
&= \frac{\text{Prob}[W^{n+1}=1 \,|\, \theta=\theta_k]p_k^n}{\text{Prob}[W^{n+1}=1 \,|\, H^n]}
\end{aligned}
$$

$$
\text{Prob}[W^{n+1}1 \,|\, H^n] = \sum_{k=1}^{K} \text{Prob}[W^{n+1}=1 \,|\, \theta=\theta_k]p_k^n
$$

  - » It is often the case that the denominator of Bayes theorem can be hard to compute. By using our sampled belief model, it is relatively easy.

Learning with a sampled belief model

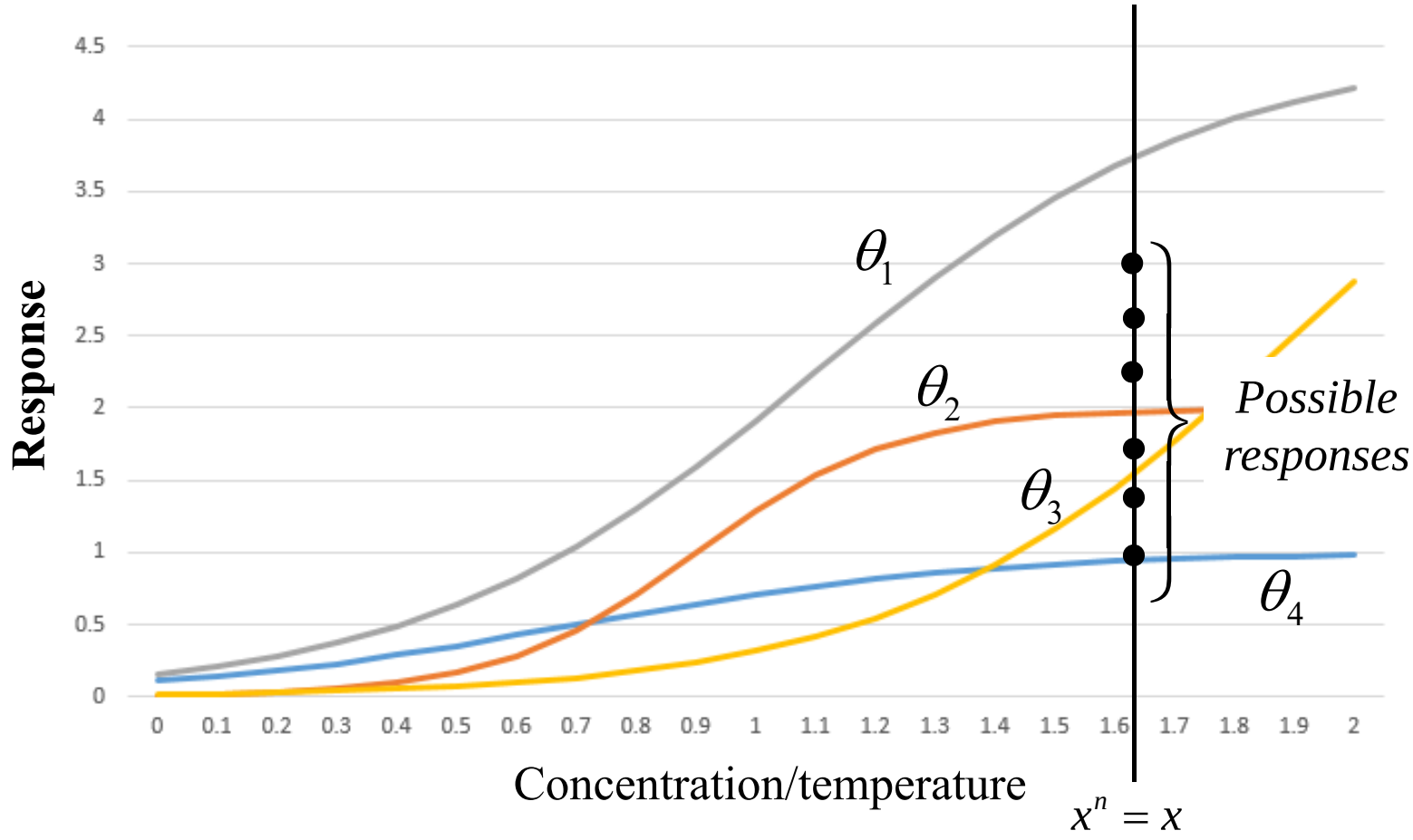
- A sampled belief model



Thickness proportional to p_k^n

Learning with a sampled belief model

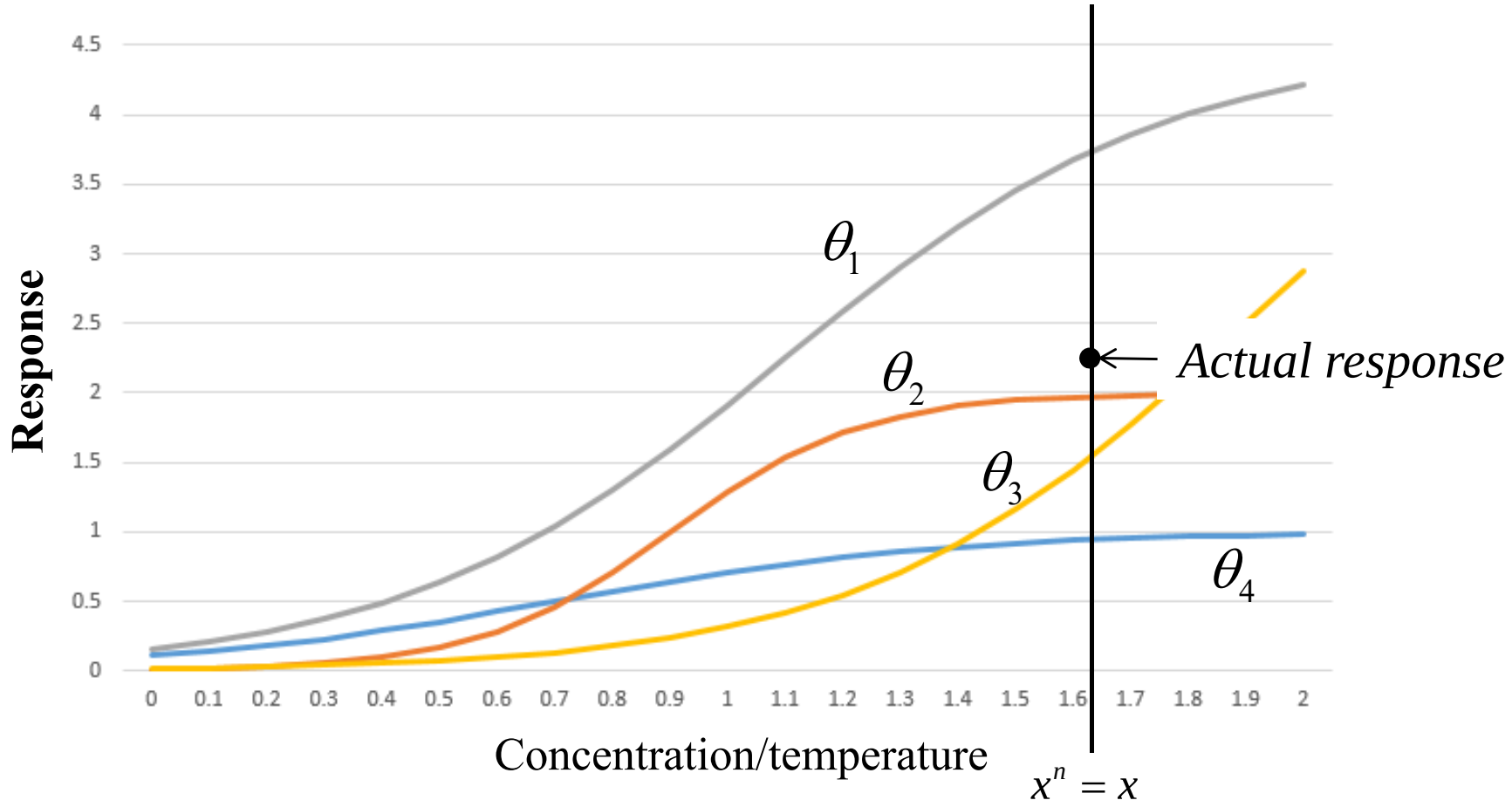
- Possible experimental outcomes



Thickness proportional to p_k^n

Learning with a sampled belief model

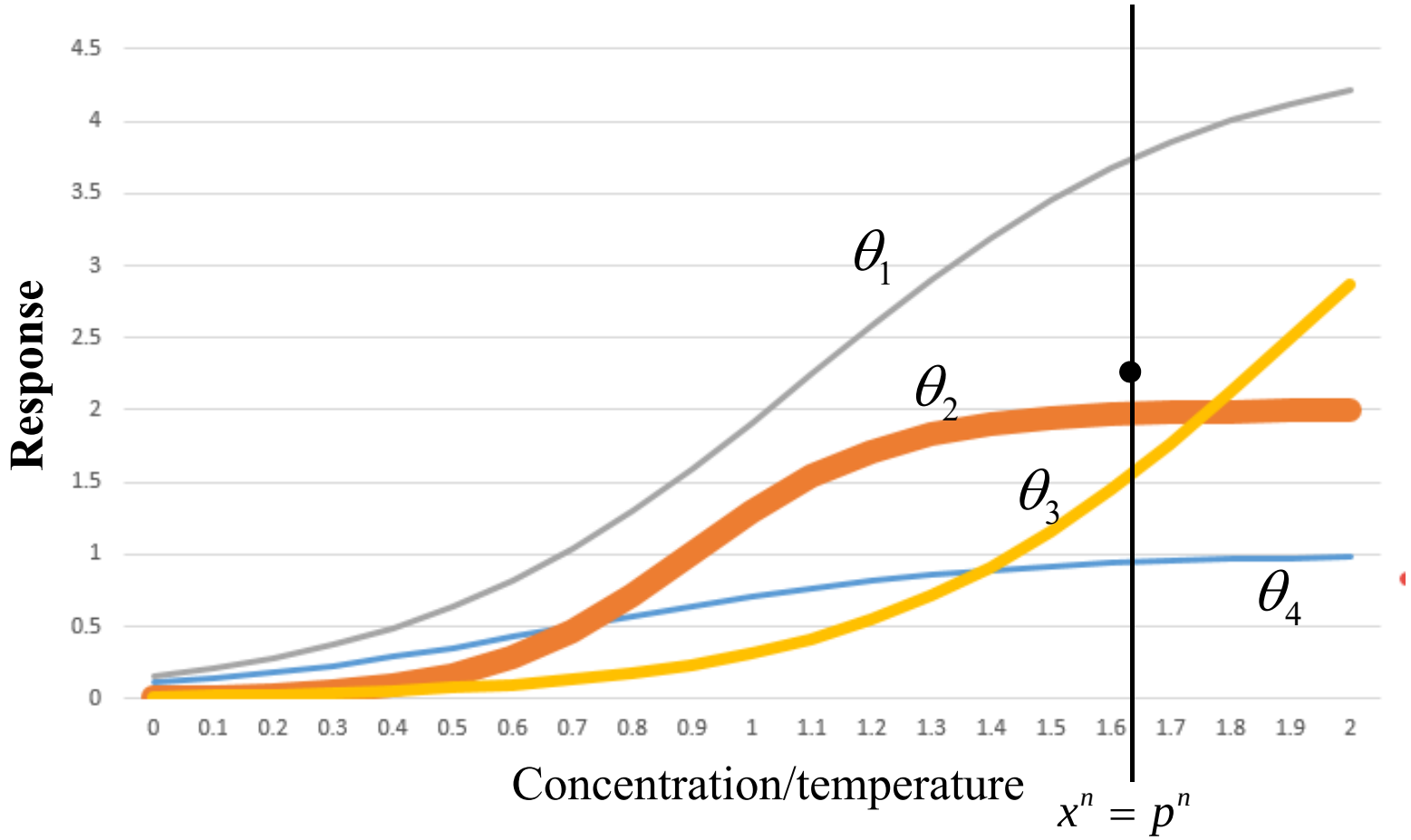
- Running an experiment – here we see the actual r



Thickness proportional to p_k^n

Learning with a sampled belief model

- Updated posterior reflects proximity to each curve



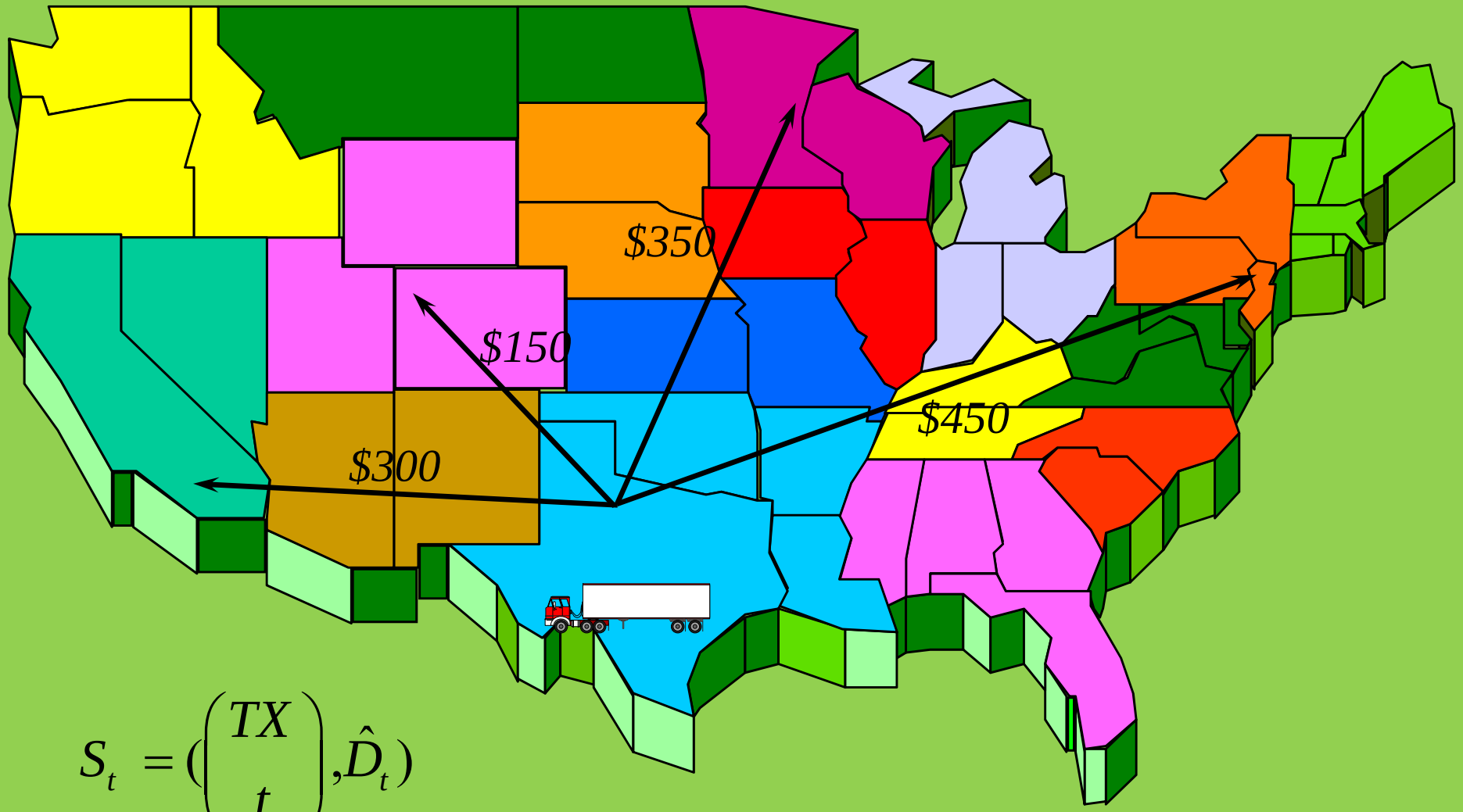
Thickness proportional to p_k^n

Hierarchical aggregation

Defer to later

Approximate dynamic programming

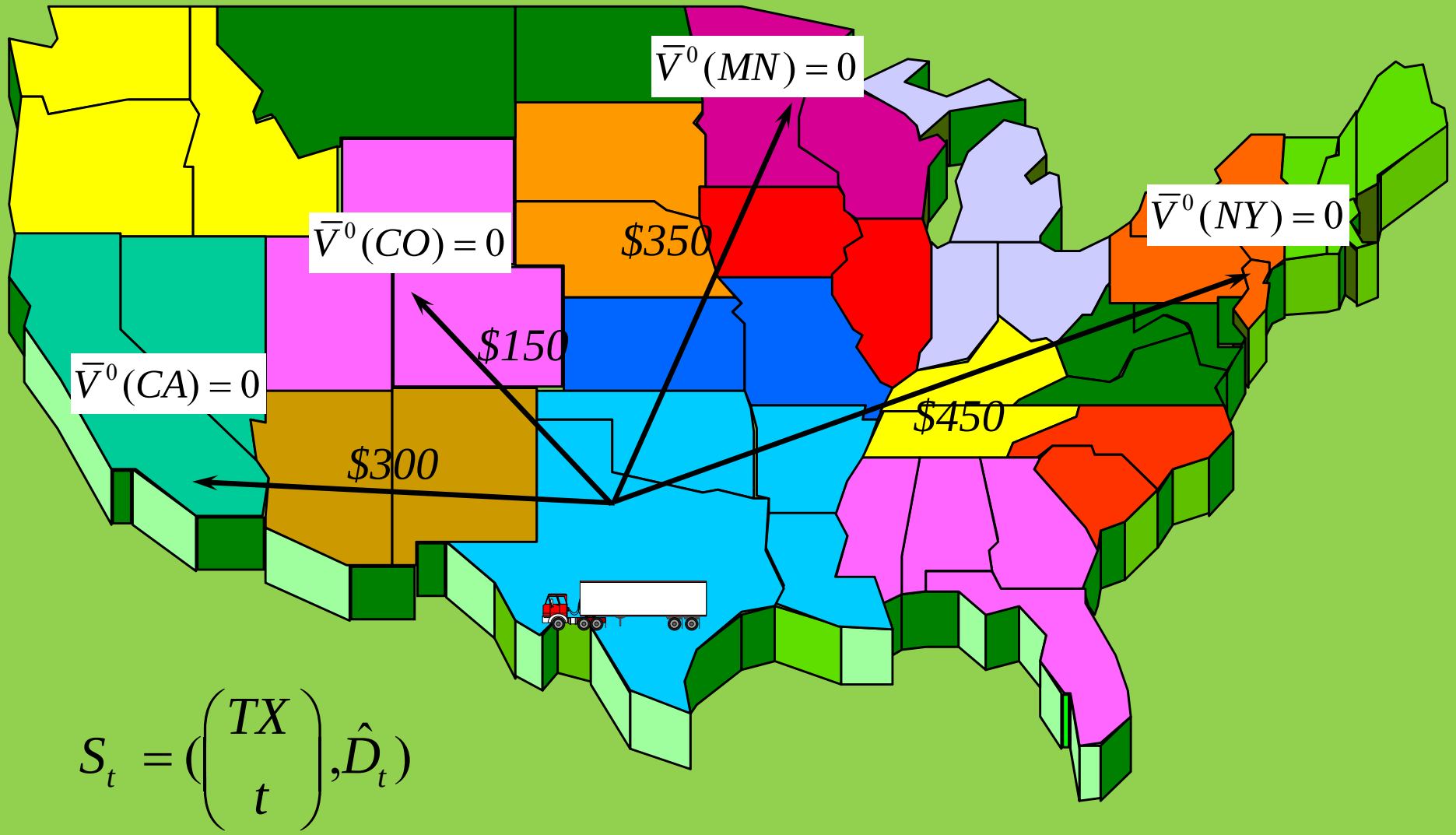
- Pre-decision state: we see the demands



$$S_t = \left(\begin{pmatrix} TX \\ t \end{pmatrix}, \hat{D}_t\right)$$

Approximate dynamic programming

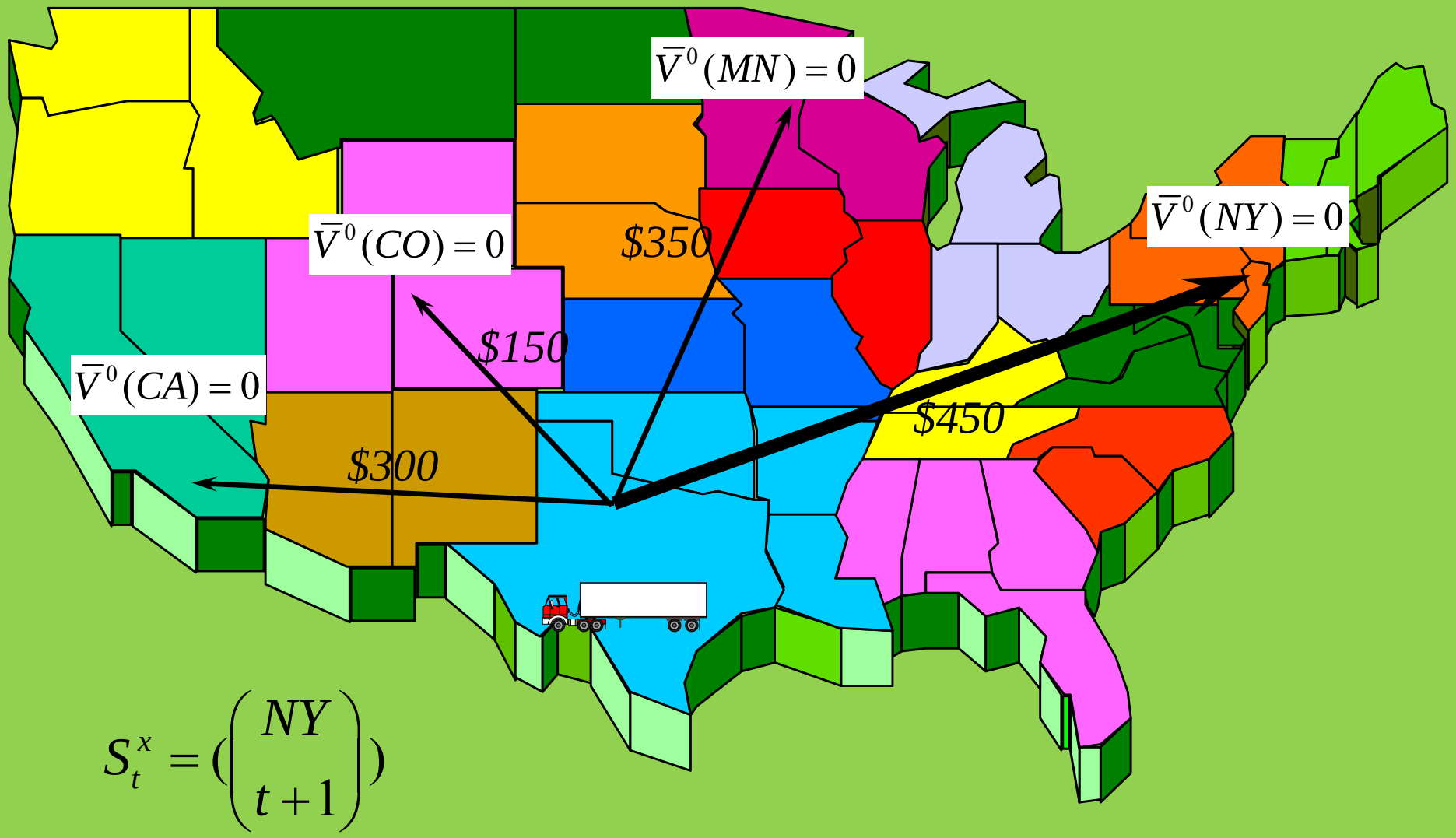
- We use initial value function approximations…



$$S_t = \left(\begin{pmatrix} TX \\ t \end{pmatrix}, \hat{D}_t\right)$$

Approximate dynamic programming

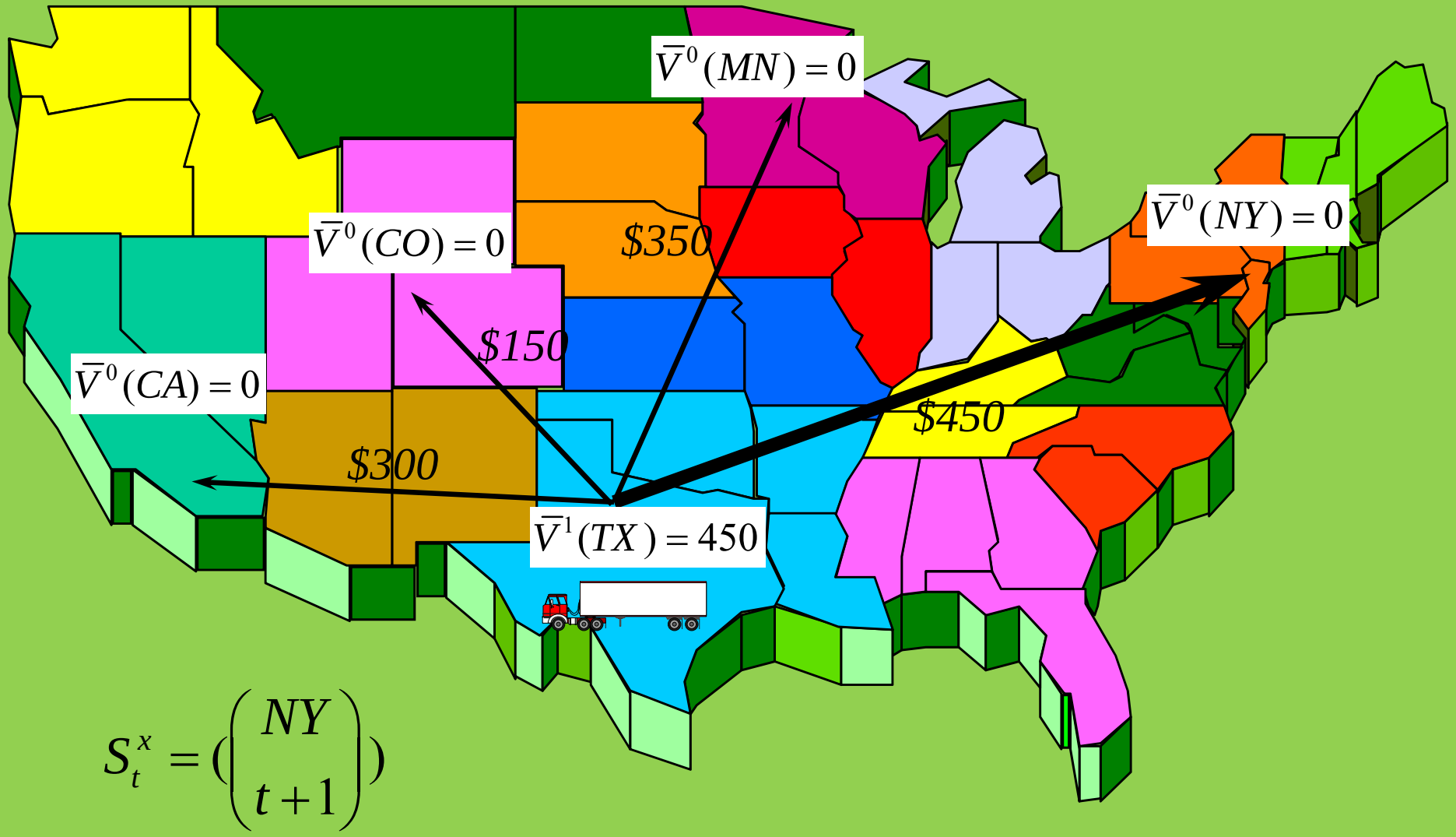
- … and make our first choice: x^1



$$S_t^x = \begin{pmatrix} NY \\ t+1 \end{pmatrix}$$

Approximate dynamic programming

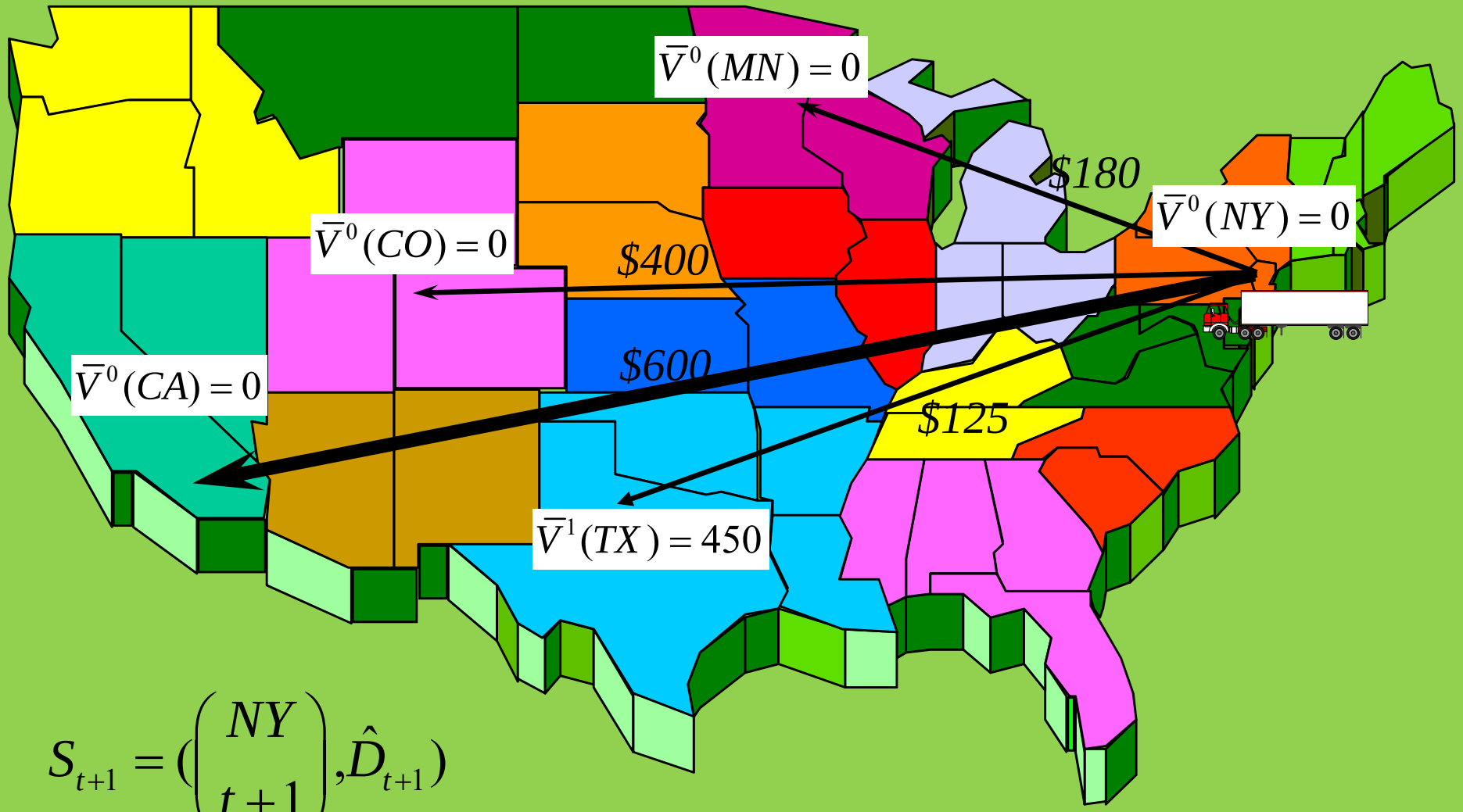
- Update the value of being in Texas.



$$S_t^x = \begin{pmatrix} NY \\ t+1 \end{pmatrix}$$

Approximate dynamic programming

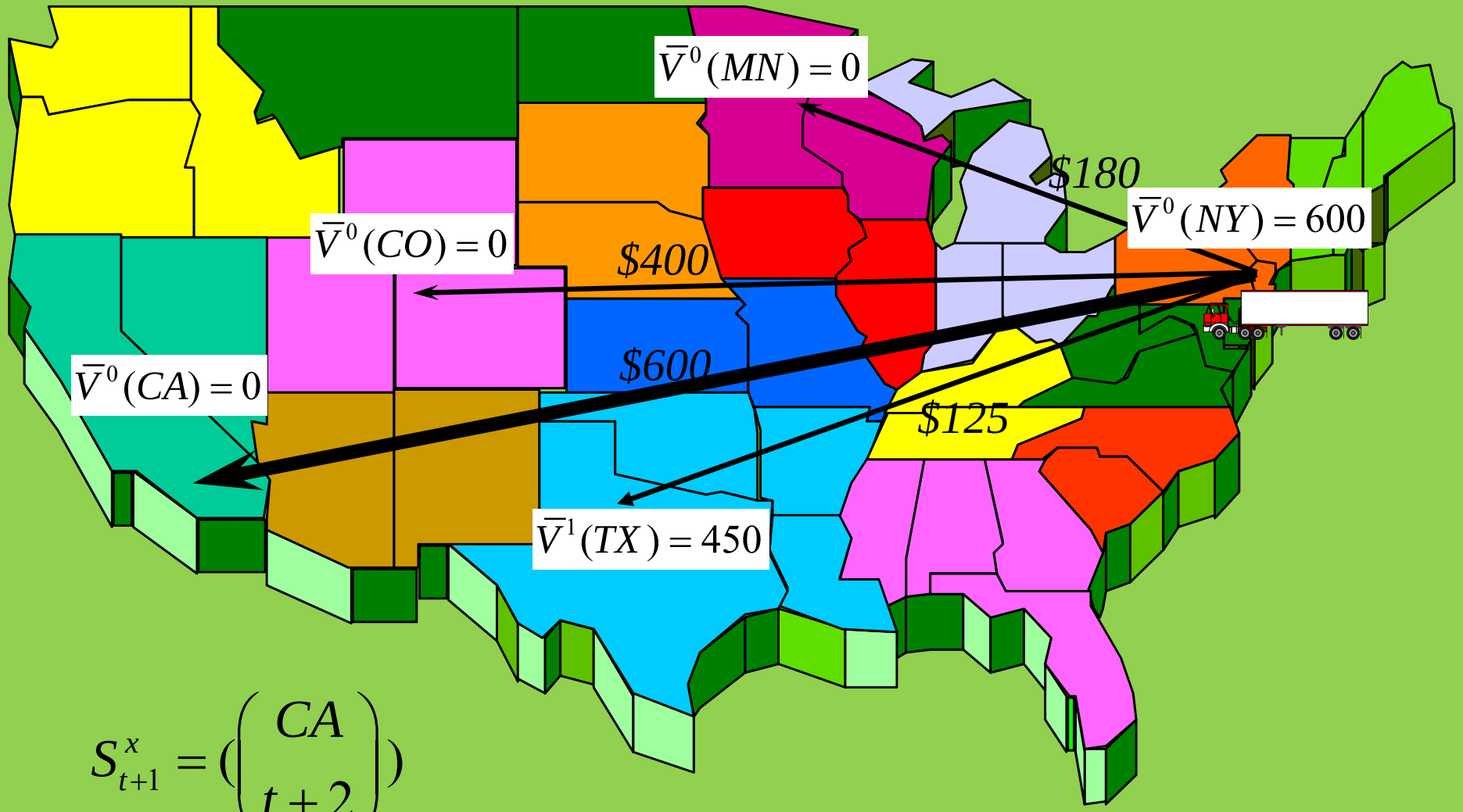
- Now move to the next state, sample new demands and make a new decision



$$S_{t+1} = \left(\begin{pmatrix} NY \\ t+1 \end{pmatrix}, \hat{D}_{t+1}\right)$$

Approximate dynamic programming

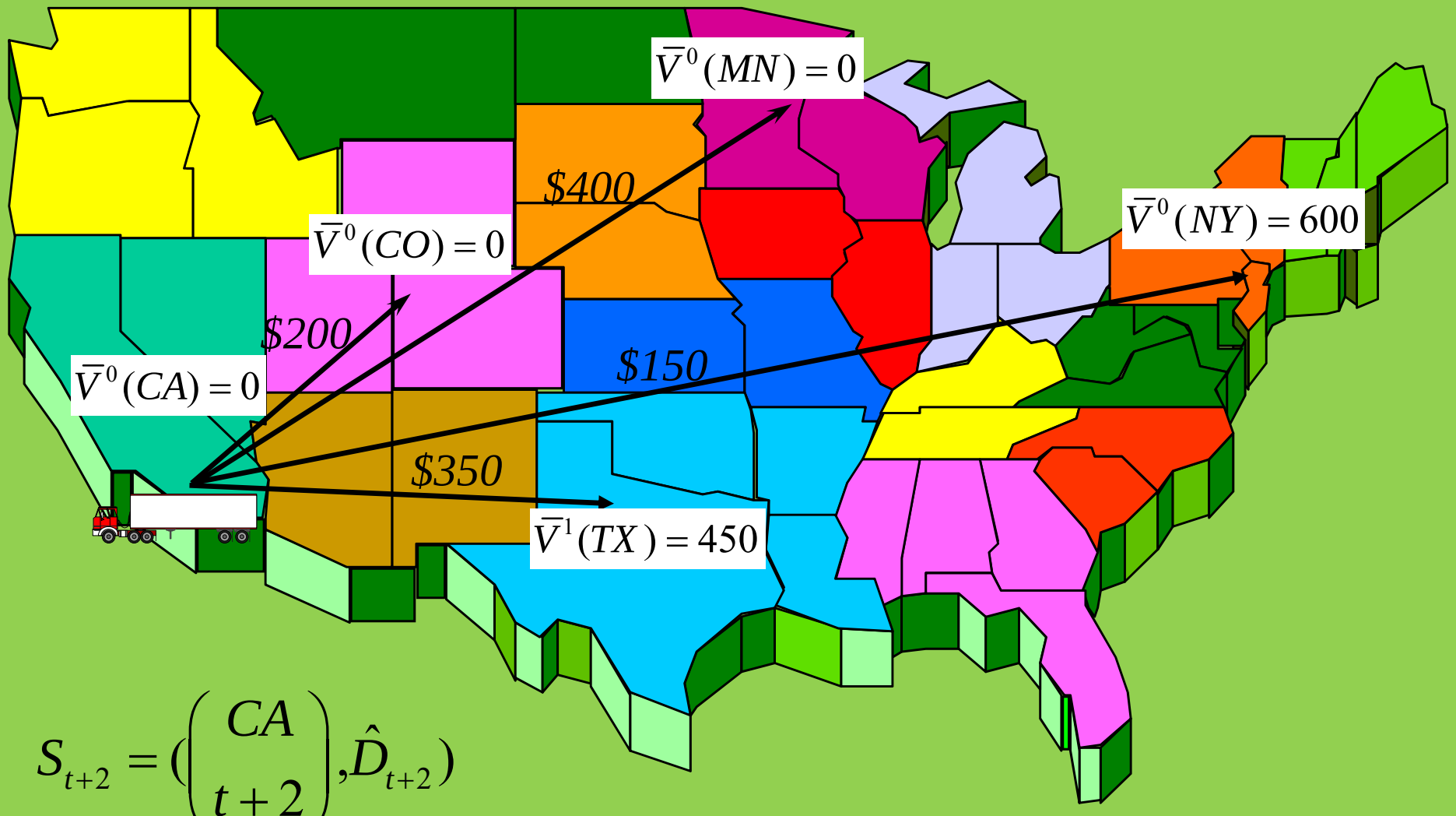
- Update value of being in NY



$$S_{t+1}^x = \begin{pmatrix} CA \\ t+2 \end{pmatrix}$$

Approximate dynamic programming

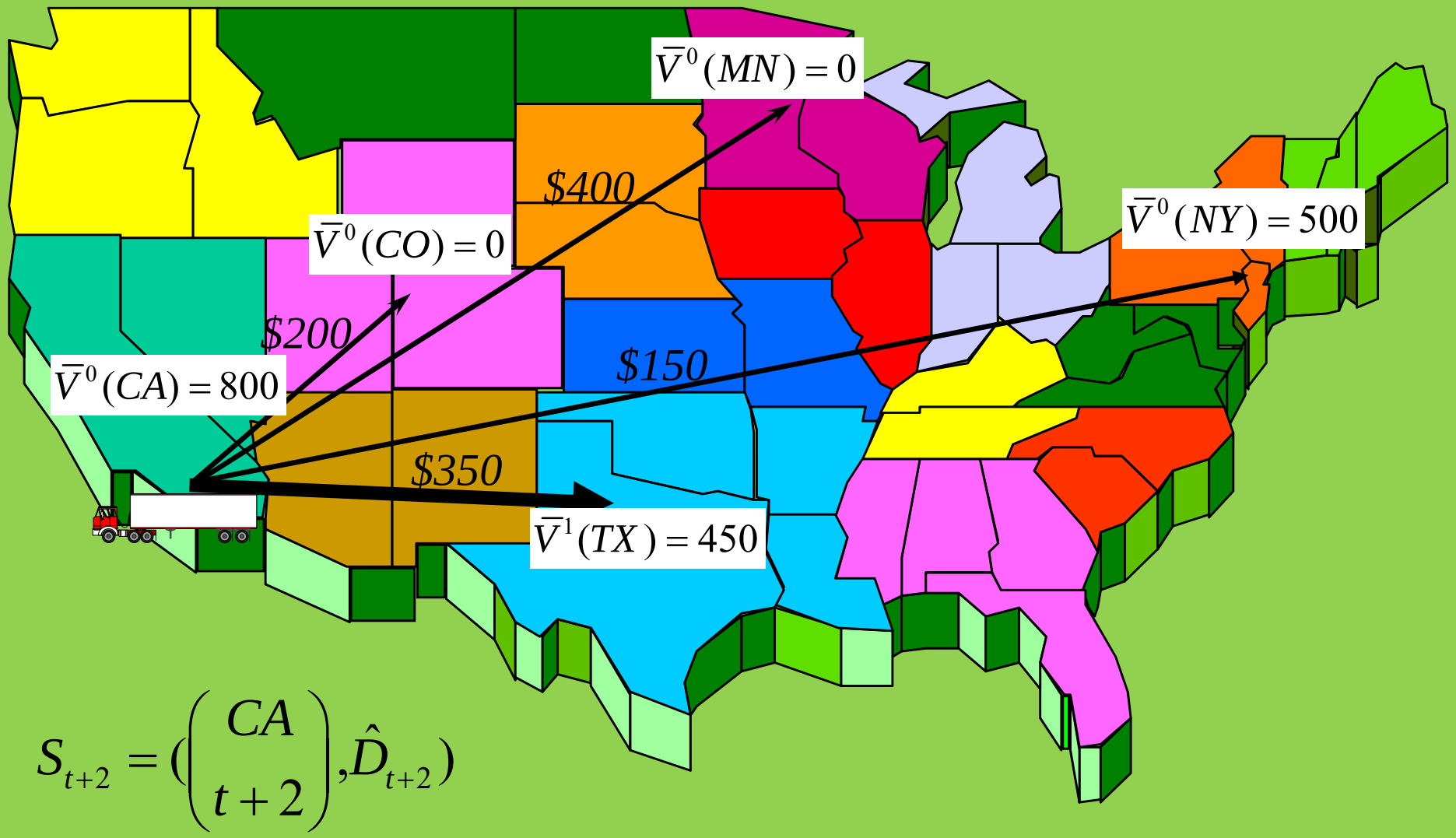
- Move to California.



$$S_{t+2} = \left(\begin{pmatrix} CA \\ t+2 \end{pmatrix}, \hat{D}_{t+2}\right)$$

Approximate dynamic programming

- Make decision to return to TX and update value of being in CA



$$S_{t+2} = \left(\begin{pmatrix} CA \\ t+2 \end{pmatrix}, \hat{D}_{t+2}\right)$$

Approximate dynamic programming

- Updating the value function:

Old value:

$$\bar{V}^1(TX) = \$450$$

New estimate:

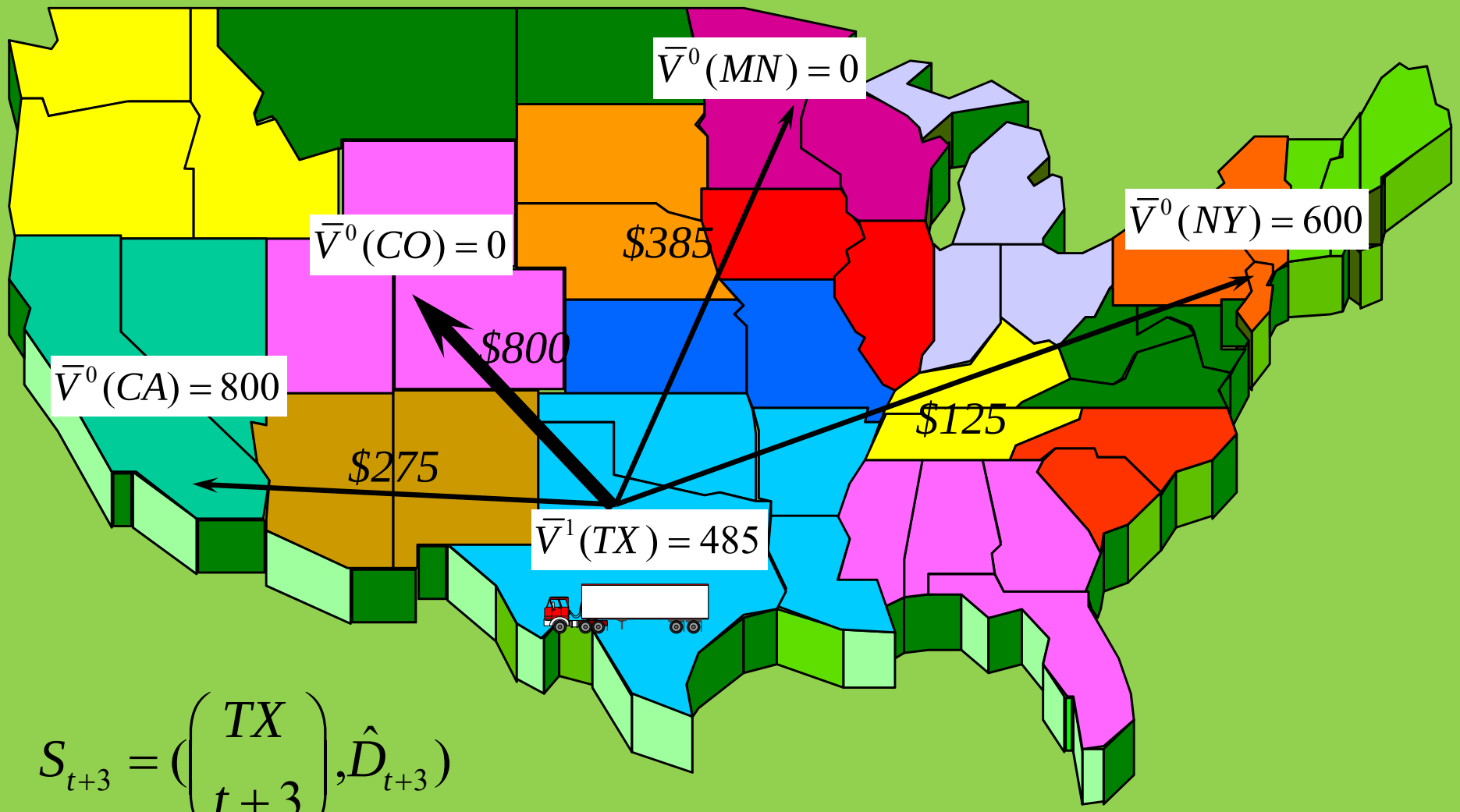
$$\hat{v}^2(TX) = \$800$$

How do we merge old with new?

$$\begin{aligned}\bar{V}^2(TX) &= (1-\alpha)\bar{V}^1(TX) + (\alpha)\hat{v}^2(TX) \\ &= (0.90)\$450 + (0.10)\$800 \\ &= \$485\end{aligned}$$

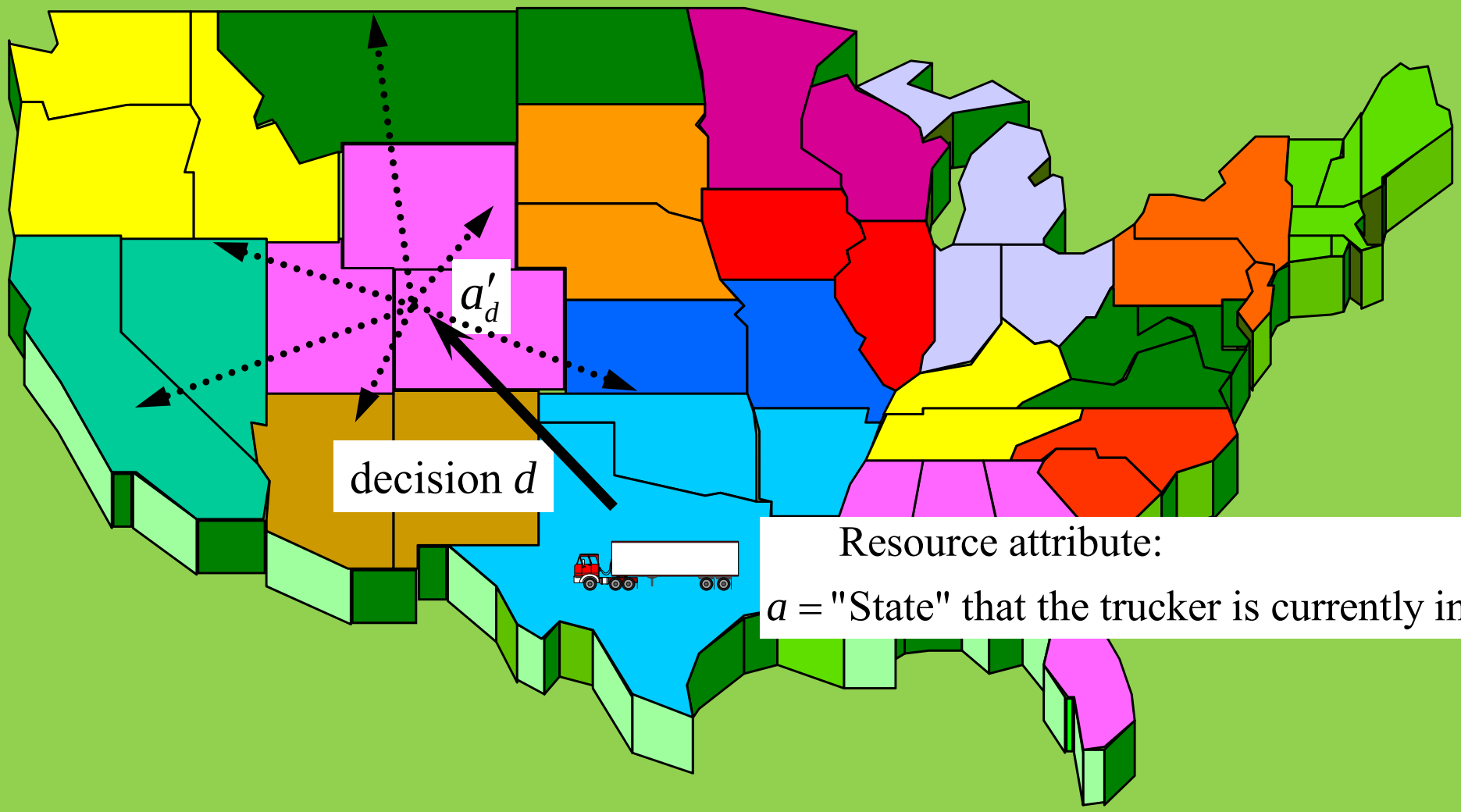
Approximate dynamic programming

- An updated value of being in TX



$$S_{t+3} = \left(\begin{pmatrix} TX \\ t+3 \end{pmatrix}, \hat{D}_{t+3}\right)$$

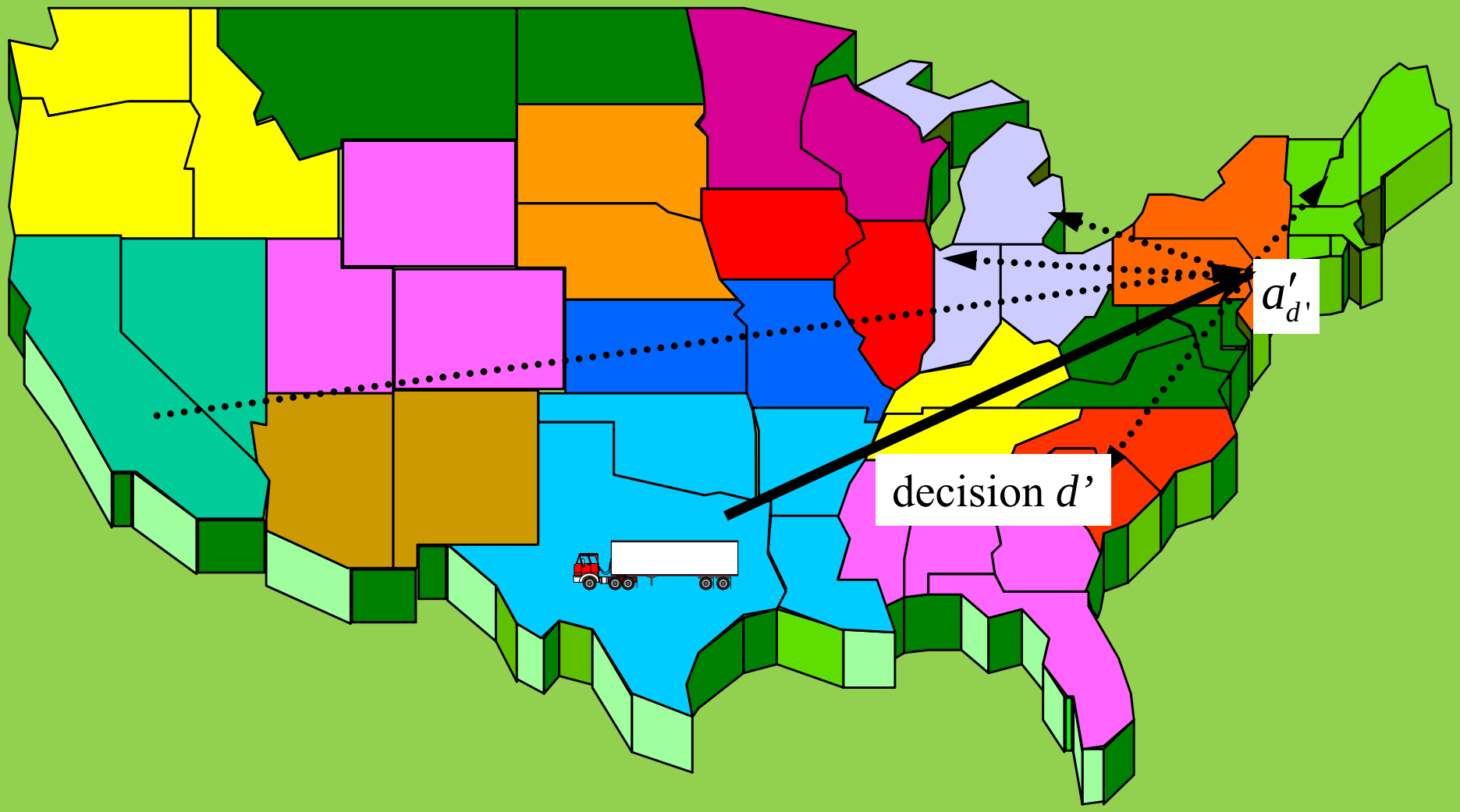
Approximate dynamic programming



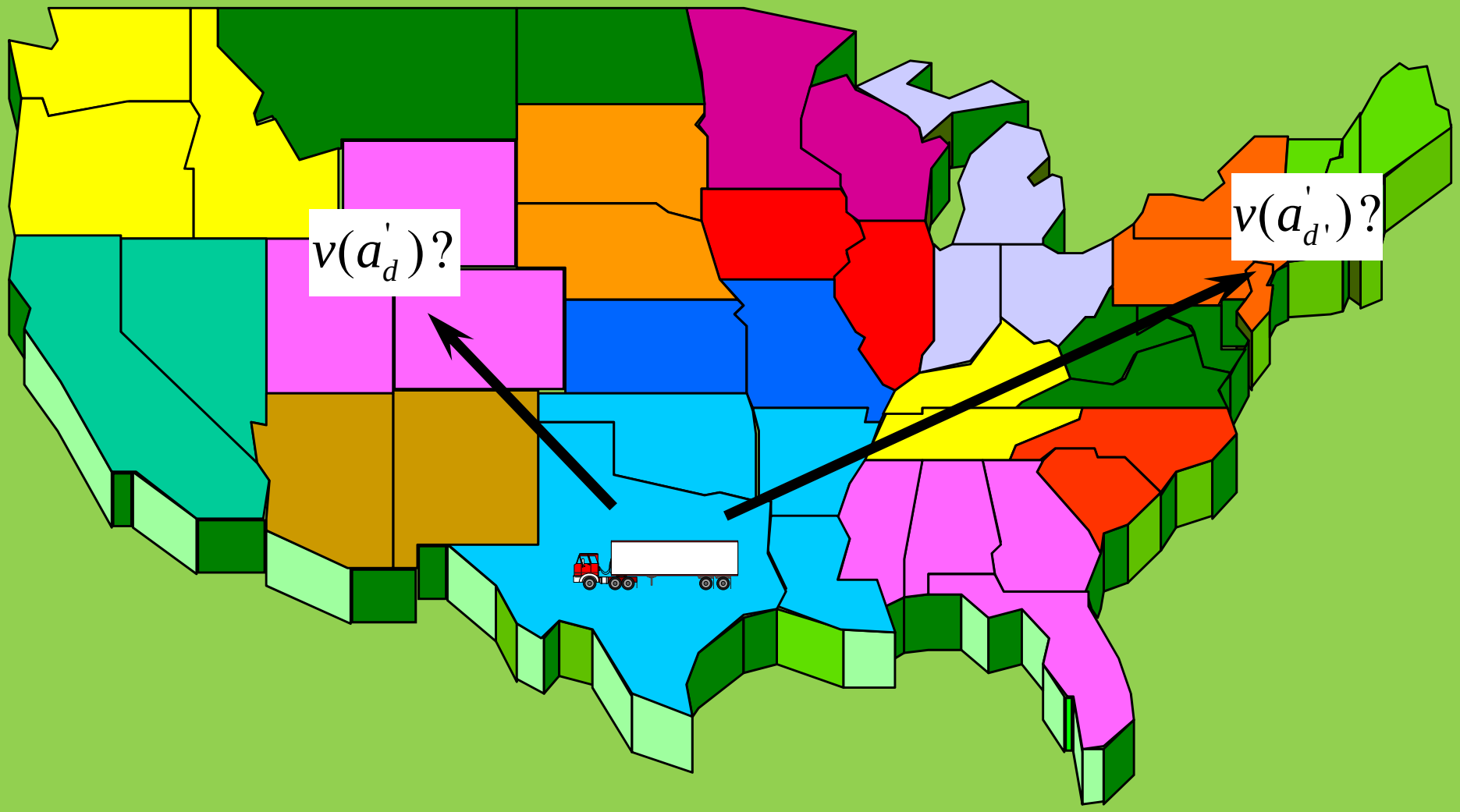
Resource attribute:

a = "State" that the trucker is currently in

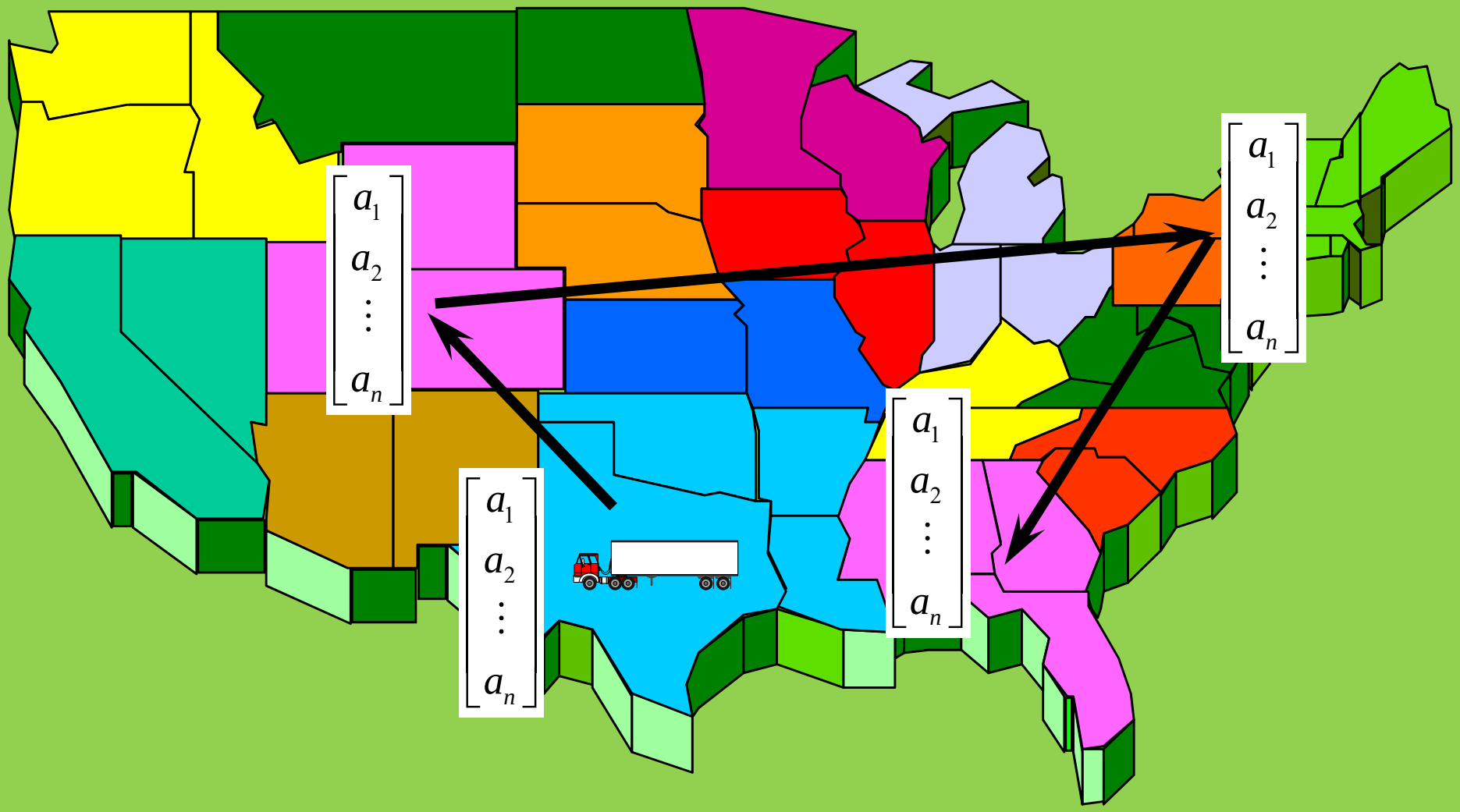
Approximate dynamic programming



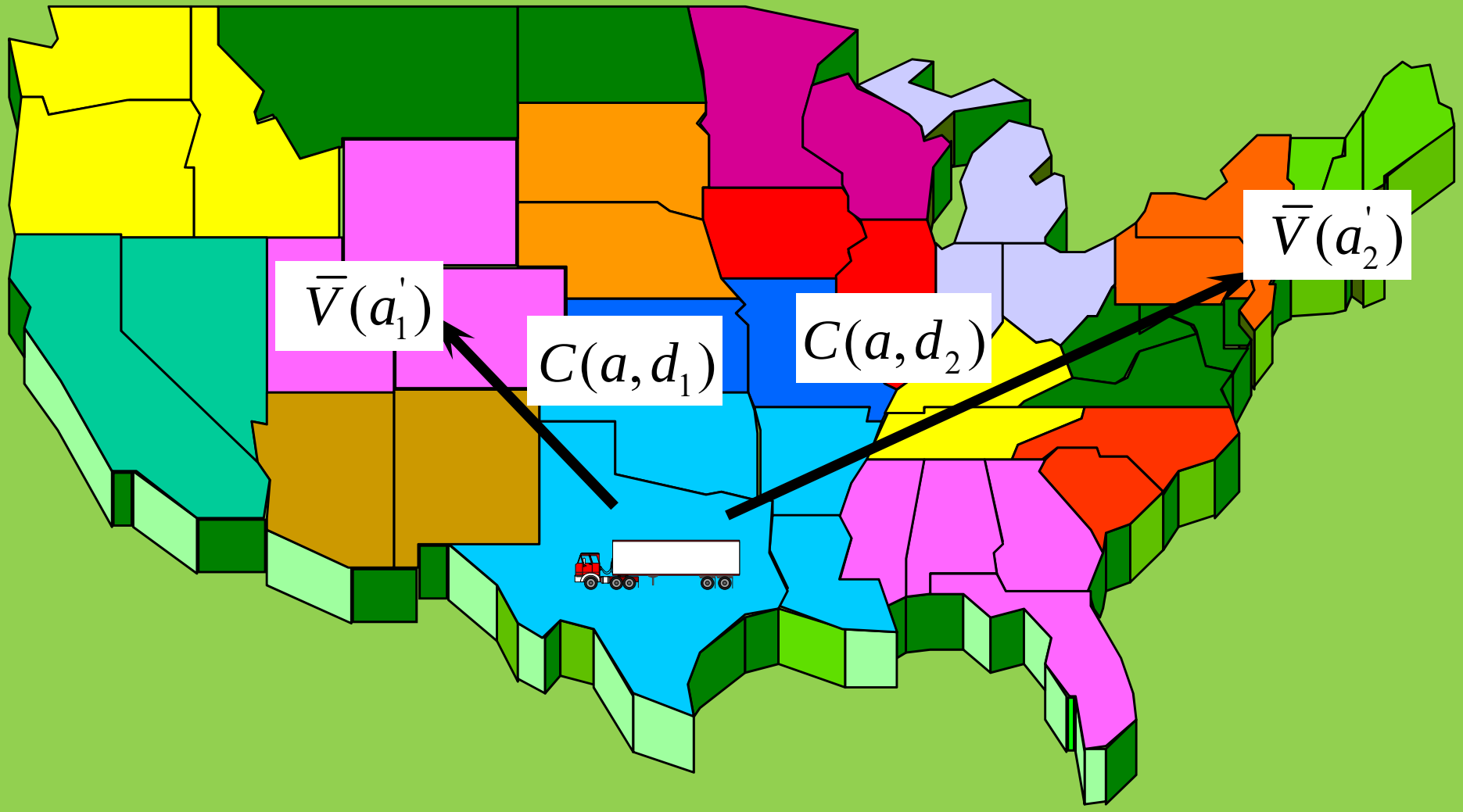
Approximate dynamic programming



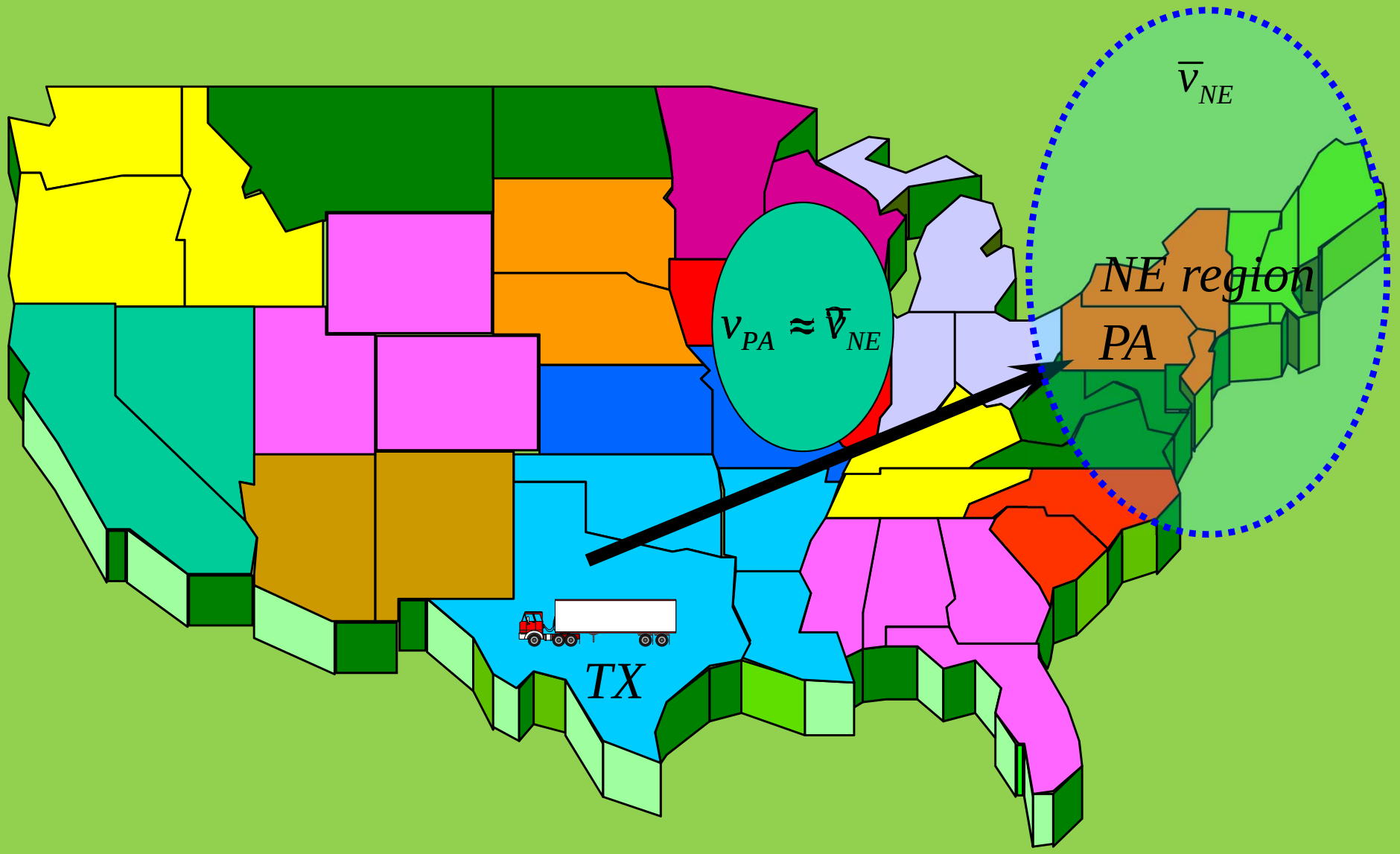
Approximate dynamic programming



Approximate dynamic programming



Approximate dynamic programming



Lookup tables–Hierarchical aggregation

- Discuss:
  - » Curse of dimensionality for lookup tables
  - » Hierarchical aggregation:
    - Ignoring attributes
    - Aggregating attributes
- Examples:
  - » Drugs and drug classes
  - » Spatial
  - » Temporal

Aggregation level	Location	Fleet type	Domicile	Size of state space
0	Sub-region	Fleet	Region	$400 \times 5 \times 100 = 200,000$
1	Region	Fleet	Region	$100 \times 5 \times 100 = 50,000$
2	Region	Fleet	Zone	$100 \times 5 \times 10 = 5,000$
3	Region	Fleet	-	$100 \times 5 \times 1 = 500$
4	Zone	-	-	$10 \times 1 \times 1 = 10$

$$\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix} = \max_a \left(C(s,a) + \overline{V}(S^M(s,a))\right).$$

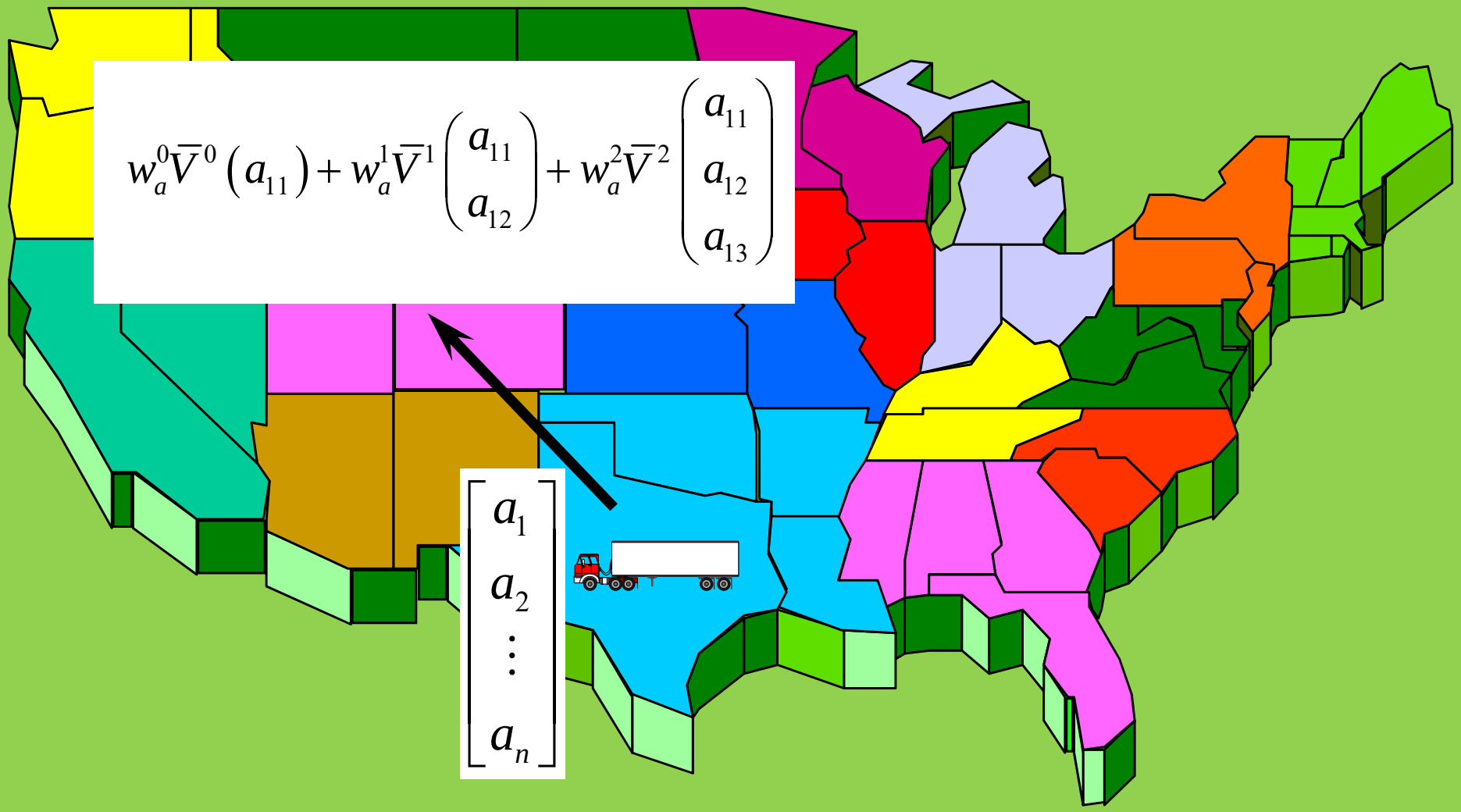
We now wish to use this estimate of the value of a driver with state s to produce value functions at different levels of aggregation. We can do this by simply smoothing this disaggregate estimate in with estimates at different levels of aggregation, as in

$$\overline{v}^{(1,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} = (1-\alpha^{(1)}_{s,n-1})\overline{v}^{(1,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} + \alpha^{(1)}_{s,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},$$

$$\overline{v}^{(2,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} = (1-\alpha^{(2)}_{s,n-1})\overline{v}^{(2,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} + \alpha^{(2)}_{s,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},$$

$$\overline{v}^{(3,n)}\begin{pmatrix}\text{Loc}\end{pmatrix} = (1-\alpha^{(3)}_{s,n-1})\overline{v}^{(3,n-1)}\begin{pmatrix}\text{Loc}\end{pmatrix} + \alpha^{(3)}_{s,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix}.$$

Approximate dynamic programming



Lookup tables–Hierarchical aggregation

3.3.4 Combining multiple levels of aggregation

Rather than try to pick the best level of aggregation, it is intuitively appealing to use a weighted sum of estimates at different levels of aggregation. The simplest strategy is to use

$$\bar{v}_s^n = \sum_{g \in \mathcal{G}} w^{(g)} \bar{v}_s^{(g)}, \tag{3.34}$$

where $w^{(g)}$ is the weight applied to the g^{th} level of aggregation. We would expect the weights to be positive and sum to one, but we can also view these simply as coefficients in a regression function. In such a setting, we would normally write the regression as

$$\overline{V}(s|\theta) = \theta_0 + \sum_{g \in \mathcal{G}} \theta_g \bar{v}_s^{(g)},$$

Lookup tables–Hierarchical aggregation

- State dependent weight:

$$\bar{v}_s^n = \sum_{g \in \mathcal{G}} w_s^{(g)} \bar{v}_s^{(g,n)}.$$

$$w_s^{(g)} \propto \left((\bar{\sigma}_s^2)^{(g,n)} \right)^{-1}.$$

Since the weights should sum to one, we obtain

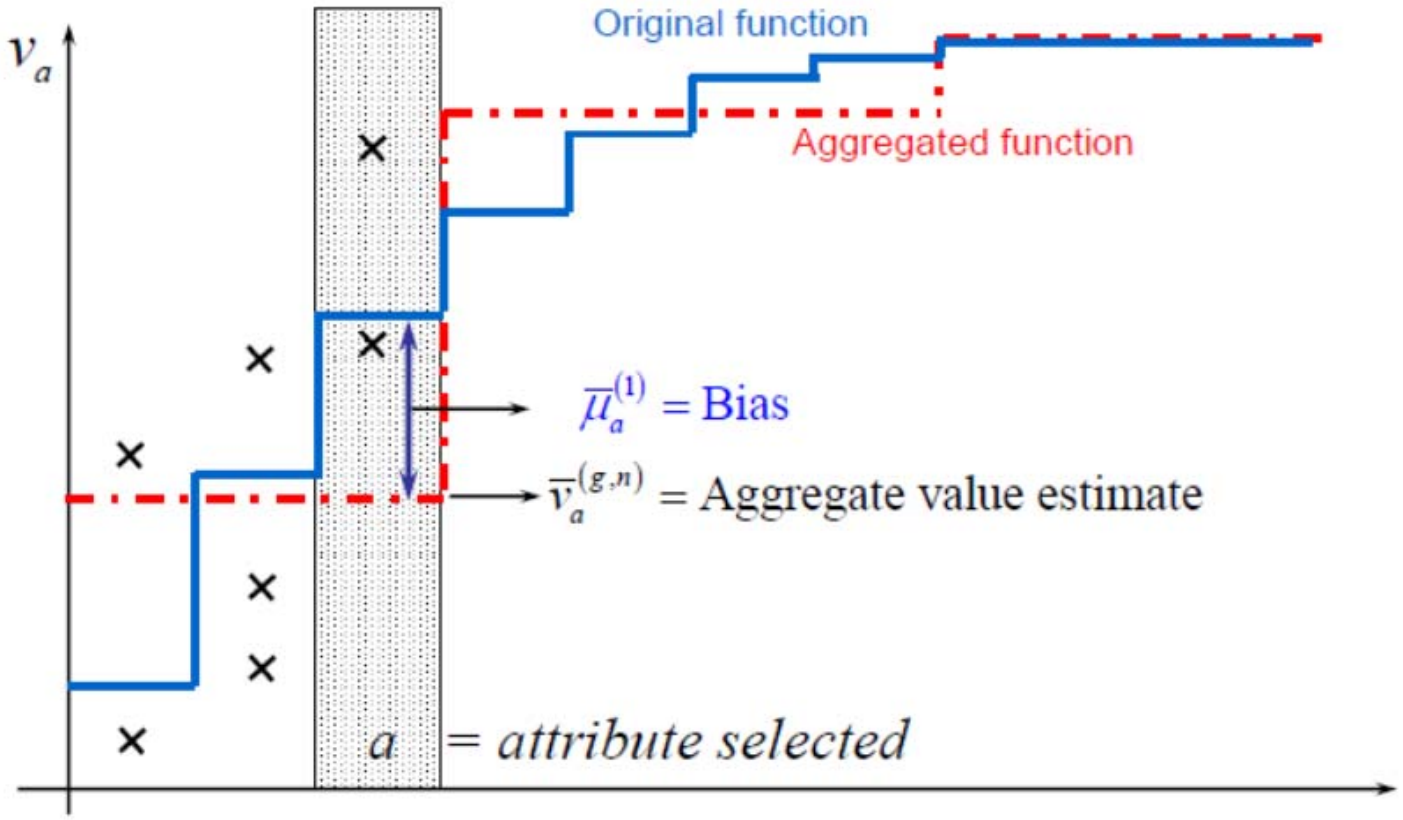
$$w_s^{(g)} = \left(\frac{1}{(\bar{\sigma}_s^2)^{(g,n)}} \right) \left(\sum_{g \in \mathcal{G}} \frac{1}{(\bar{\sigma}_s^2)^{(g,n)}} \right)^{-1}. \tag{3.35}$$

These weights work if the estimates are unbiased, which is clearly not the case. This is easily fixed by using the total variation (variance plus the square of the bias), producing the weights

$$w_s^{(g,n)} = \frac{1}{\left((\bar{\sigma}_s^2)^{(g,n)} + \left(\bar{\mu}_s^{(g,n)} \right)^2 \right)} \left(\sum_{g' \in \mathcal{G}} \frac{1}{\left((\bar{\sigma}_s^2)^{(g',n)} + \left(\bar{\mu}_s^{(g',n)} \right)^2 \right)} \right)^{-1}. \tag{3.36}$$

Lookup tables–Hierarchical aggregation

- Computing bias and variance



Lookup tables–Hierarchical aggregation

- Computing bias and variance:

$$\hat{\mu}^n = \mu(n) + \varepsilon^n,$$

where $\mathbb{E}\varepsilon^n = 0$ and $Var[\varepsilon^n] = \sigma^2$. Previously, ε^n was the error between our previous estimate and our latest observation. Here, we treat this as an exogenous measurement error. With this model, we can compute the variance of μ^n using

$$Var[\bar{\mu}^n] = \lambda^n \sigma^2, \tag{3.22}$$

where λ^n can be computed from the simple recursion

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1, \\ (1-\alpha_{n-1})^2 \lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases} \tag{3.23}$$

To see this, we start with $n = 1$. For a given (deterministic) initial estimate μ^0, we first observe that the variance of $\bar{\mu}^1$ is given by

$$\begin{aligned} Var[\bar{\mu}^1] &= Var[(1-\alpha_0)\bar{\mu}^0 + \alpha_0 \hat{\mu}^1] \\ &= (\alpha_0)^2 Var[\hat{\mu}^1] \\ &= (\alpha_0)^2 \sigma^2. \end{aligned}$$

For $\bar{\mu}^n$ for $n > 1$, we use a proof by induction. Assume that $Var[\bar{\mu}^{n-1}] = \lambda^{n-1}\sigma^2$. Then, since $\bar{\mu}^{n-1}$ and $\hat{\mu}^n$ are independent, we find

$$\begin{aligned} Var[\bar{\mu}^n] &= Var\left[(1-\alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n\right] \\ &= (1-\alpha_{n-1})^2 Var\left[\bar{\mu}^{n-1}\right] + (\alpha_{n-1})^2 Var[\hat{\mu}^n] \\ &= (1-\alpha_{n-1})^2 \lambda^{n-1}\sigma^2 + (\alpha_{n-1})^2 \sigma^2 \qquad (3.24) \\ &= \lambda^n \sigma^2. \qquad (3.25) \end{aligned}$$

Equation (3.24) is true by assumption (in our induction proof), while equation (3.25) establishes the recursion in equation (3.23). This gives us the variance, assuming of course that σ^2 is known.

Lookup tables–Hierarchical aggregation

situation of approximating value functions where the true value is evolving (for example, it may be increasing) over the iterations. To avoid confusion with our static notation, we are going to use $\bar{\mu}_x^n$ as our estimate of this time-varying function at x. We are then going to let $\hat{\mu}_x^n$ be a noisy observation of $mu(n)_x$.

We will use the following recursive formula for $\bar{\mu}^n$ (for any point x)

$$\bar{\mu}^n = (1 - \alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n,$$

where $\hat{\mu}^n = \mu(n) + \epsilon^n$ is an unbiased observation (that is, $\mathbb{E}\hat{\mu}^n = \mu(n)$) which is assumed to be independent of μ^{n-1}. We note that when we are trying to estimate a static function such as $\mathbb{E}F(x, W)$, the parameter we are trying to estimate, $\mu_x(n) = \mu_x = \mathbb{E}F(x, W)$ is static. However, when we are trying to estimate a value function as we illustrated in equation (3.2), then we are trying to estimate a parameter $\bar{\mu}_x^n$ that actually varies with n. We will see this behavior, first in chapter 14 when we introduce basic dynamic programs, and then in chapters 17 and 18 when we cover approximate dynamic programming.

We are interested in estimating the variance of μ^n and its bias, which is defined by $\mu^{n-1} - \mu^n$. We start by computing the variance of μ^n. We assume that our observations of μ^n can be represented using

$$\hat{\mu}^n = \mu(n) + \varepsilon^n,$$

where $\mathbb{E}\varepsilon^n = 0$ and $Var[\varepsilon^n] = \sigma^2$. Previously, ε^n was the error between our previous estimate and our latest observation. Here, we treat this as an exogenous measurement error. With this model, we can compute the variance of μ^n using

$$Var[\bar{\mu}^n] = \lambda^n \sigma^2, \tag{3.22}$$

Lookup tables–Hierarchical aggregation

$$Var[\bar{\mu}^n] = \lambda^n \sigma^2, \tag{3.22}$$

where λ^n can be computed from the simple recursion

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1, \\ (1-\alpha_{n-1})^2 \lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases} \tag{3.23}$$

To see this, we start with $n = 1$. For a given (deterministic) initial estimate μ^0, we first observe that the variance of $\bar{\mu}^1$ is given by

$$\begin{aligned} Var[\bar{\mu}^1] &= Var[(1-\alpha_0)\bar{\mu}^0 + \alpha_0 \hat{\mu}^1] \\ &= (\alpha_0)^2 Var[\hat{\mu}^1] \\ &= (\alpha_0)^2 \sigma^2. \end{aligned}$$

For $\bar{\mu}^n$ for $n > 1$, we use a proof by induction. Assume that $Var[\bar{\mu}^{n-1}] = \lambda^{n-1}\sigma^2$. Then, since $\bar{\mu}^{n-1}$ and $\hat{\mu}^n$ are independent, we find

$$\begin{aligned} Var[\bar{\mu}^n] &= Var\left[(1-\alpha_{n-1})\bar{\mu}^{n-1} + \alpha_{n-1}\hat{\mu}^n\right] \\ &= (1-\alpha_{n-1})^2 Var\left[\bar{\mu}^{n-1}\right] + (\alpha_{n-1})^2 Var[\hat{\mu}^n] \\ &= (1-\alpha_{n-1})^2 \lambda^{n-1}\sigma^2 + (\alpha_{n-1})^2 \sigma^2 \qquad (3.24) \\ &= \lambda^n \sigma^2. \qquad (3.25) \end{aligned}$$

Equation (3.24) is true by assumption (in our induction proof), while equation (3.25) establishes the recursion in equation (3.23). This gives us the variance, assuming of course that σ^2 is known.

Lookup tables–Hierarchical aggregation

The bias of our estimate is the difference between our current estimate and the true value, given by

$$\beta^n = \mathbb{E}[\bar{\mu}^{n-1}] - \mu^n.$$

We note that $\bar{\mu}^{n-1}$ is our estimate of μ^n computed using the information up through iteration $n-1$. Of course, our formula for the bias assumes that $\mu(n)$ is known. These two results for the variance and bias are called the *parameters-known* formulas.

We next require the mean-squared error, which can be computed using

$$\mathbb{E}\left[\left(\bar{\mu}^{n-1} - \mu(n)\right)^2\right] = \lambda^{n-1}\sigma^2 + (\beta^n)^2. \tag{3.26}$$

See exercise 3.3 to prove this. This formula gives the variance around the known mean, μ^n. For our purposes, it is also useful to have the variance around the observations $\hat{\mu}^n$. Let

$$\nu^n = \mathbb{E}\left[\left(\bar{\mu}^{n-1} - \hat{\mu}^n\right)^2\right]$$

be the mean squared error (including noise and bias) between the current estimate $\bar{\mu}^{n-1}$ and the observation $\hat{\mu}^n$. It is possible to show that (see exercise 3.4)

$$\nu^n = (1 + \lambda^{n-1})\sigma^2 + (\beta^n)^2, \tag{3.27}$$

where λ^n is computed using (3.23).

Lookup tables–Hierarchical aggregation

In practice, we do not know σ^2, and we certainly do not know $\mu(n)$. As a result, we have to estimate both parameters from our data. We begin by providing an estimate of the bias using

$$\bar{\beta}^n = (1 - \eta_{n-1})\bar{\beta}^{n-1} + \eta_{n-1}(\bar{\mu}^{n-1} - \hat{\mu}^n),$$

where η_{n-1} is a (typically simple) stepsize rule used for estimating the bias and variance. As a general rule, we should pick a stepsize for η_{n-1} which produces larger stepsizes than α_{n-1} because we are more interested in tracking the true signal than producing an estimate with a low variance. We have found that a constant stepsize such as .10 works quite well on a wide range of problems, but if precise convergence is needed, it is necessary to use a rule where the stepsize goes to zero such as the harmonic stepsize rule (equation (6.12)).

To estimate the variance, we begin by finding an estimate of the total variation ν^n. Let $\bar{\nu}^n$ be the estimate of the total variance which we might compute using

$$\bar{\nu}^n = (1 - \eta_{n-1})\bar{\nu}^{n-1} + \eta_{n-1}(\bar{\mu}^{n-1} - \hat{\mu}^n)^2.$$

Using $\bar{\nu}^n$ as our estimate of the total variance, we can compute an estimate of σ^2 using

$$(\bar{\sigma}^n)^2 = \frac{\bar{\nu}^n - (\bar{\beta}^n)^2}{1 + \lambda^{n-1}}.$$

We can use $(\bar{\sigma}^n)^2$ in equation (3.22) to obtain an estimate of the variance of μ^n.

If we are doing true averaging (as would occur if we use a stepsize of $1/n$), we can get a more precise estimate of the variance for small samples by using the recursive form of the small sample formula for the variance

$$(\hat{\sigma}^2)^n = \frac{n-2}{n-1}(\hat{\sigma}^2)^{n-1} + \frac{1}{n}(\bar{\mu}^{n-1} - \hat{\mu}^n)^2. \tag{3.28}$$

The quantity $(\hat{\sigma}^2)^n$ is an estimate of the variance of $\hat{\mu}^n$. The variance of our estimate $\mu^{(n)}$ is computed using

$$(\bar{\sigma}^2)^n = \frac{1}{n}(\hat{\sigma}^2)^n.$$

Lookup tables–Hierarchical aggregation

$$\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix} = \max_a \left(C(s,a) + \overline{V}(S^M(s,a))\right).$$

We now wish to use this estimate of the value of a driver with state s to produce value functions at different levels of aggregation. We can do this by simply smoothing this disaggregate estimate in with estimates at different levels of aggregation, as in

$$\bar{v}^{(1,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} = (1-\alpha^{(1)}_{s,n-1})\bar{v}^{(1,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\end{pmatrix} + \alpha^{(1)}_{s,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},$$

$$\bar{v}^{(2,n)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} = (1-\alpha^{(2)}_{s,n-1})\bar{v}^{(2,n-1)}\begin{pmatrix}\text{Loc}\\ \text{Equip}\end{pmatrix} + \alpha^{(2)}_{s,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix},$$

$$\bar{v}^{(3,n)}\begin{pmatrix}\text{Loc}\end{pmatrix} = (1-\alpha^{(3)}_{s,n-1})\bar{v}^{(3,n-1)}\begin{pmatrix}\text{Loc}\end{pmatrix} + \alpha^{(3)}_{s,n-1}\hat{v}\begin{pmatrix}\text{Loc}\\ \text{Equip}\\ \text{Home}\\ \text{DOThrs}\\ \text{Days}\end{pmatrix}.$$

Lookup tables–Hierarchical aggregation

We can estimate the variance of $\bar{v}_s^{(g,n)}$ using the techniques described in section 3.3.2. Let

$$(s_s^2)^{(g,n)} = \text{The estimate of the variance of observations made of state } s, \text{ using data from aggregation level } g, \text{ after } n \text{ observations.}$$

$(s_s^2)^{(g,n)}$ is the estimate of the variance of the observations $\hat{v}$ when we observe a state $\hat{s}^n$ which aggregates to state s (that is, $G^g(\hat{s}^n) = s$). We are really interested in the variance of our estimate of the mean, $\bar{v}_s^{(g,n)}$. In section 3.3.2, we showed that

$$\begin{aligned}(\bar{\sigma}_s^2)^{(g,n)} &= Var[\bar{v}_s^{(g,n)}] \\ &= \lambda_s^{(g,n)}(s_s^2)^{(g,n)}, \end{aligned} \tag{3.30}$$

where $(s_s^2)^{(g,n)}$ is an estimate of the variance of the observations $\hat{v}_t^n$ at the g^{th} level of aggregation (computed below), and $\lambda_s^{(g,n)}$ can be computed from the recursion

$$\lambda_s^{(g,n)} = \begin{cases} (\alpha_{s,n-1}^{(g)})^2, & n = 1, \\ (1-\alpha_{s,n-1}^{(g)})^2\lambda_s^{(g,n-1)} + (\alpha_{s,n-1}^{(g)})^2, & n > 1. \end{cases}$$

Note that if the stepsize $\alpha_{s,n-1}^{(g)}$ goes to zero, then $\lambda_s^{(g,n)}$ will also go to zero, as will $(\bar{\sigma}_s^2)^{(g,n)}$. We now need to compute $(s_s^2)^{(g,n)}$ which is the estimate of the variance of observations $\hat{v}^n$ for states $\hat{s}^n$ for which $G^g(\hat{s}^n) = s$ (the observations of states that aggregate up to s). Let $\bar{\nu}_s^{(g,n)}$ be the total variation, given by

$$\bar{\nu}_s^{(g,n)} = (1-\eta_{n-1})\bar{\nu}_s^{(g,n-1)} + \eta_{n-1}(\bar{v}_s^{(g,n-1)} - \hat{v}_s^n)^2,$$

where η_{n-1} follows some stepsize rule (which may be just a constant). We refer to $\bar{\nu}_s^{(g,n)}$ as

Lookup tables–Hierarchical aggregation

where η_{n-1} follows some stepsize rule (which may be just a constant). We refer to $\bar{\nu}_s^{(g,n)}$ as the total variation because it captures deviations that arise both due to measurement noise (the randomness when we compute $\hat{v}_s^n$) and bias (since $\bar{v}_s^{(g,n-1)}$ is a biased estimate of the mean of $\hat{v}_s^n$).

We finally need an estimate of the bias. There are two types of bias. In section 3.3.2 we measured the transient bias that arose from the use of smoothing applied to a nonstationary time series using

$$\bar{\beta}_s^{(g,n)} = (1-\eta_{n-1})\bar{\beta}_s^{(g,n-1)} + \eta_{n-1}(\hat{v}^n - \bar{v}_s^{(g,n-1)}). \tag{3.31}$$

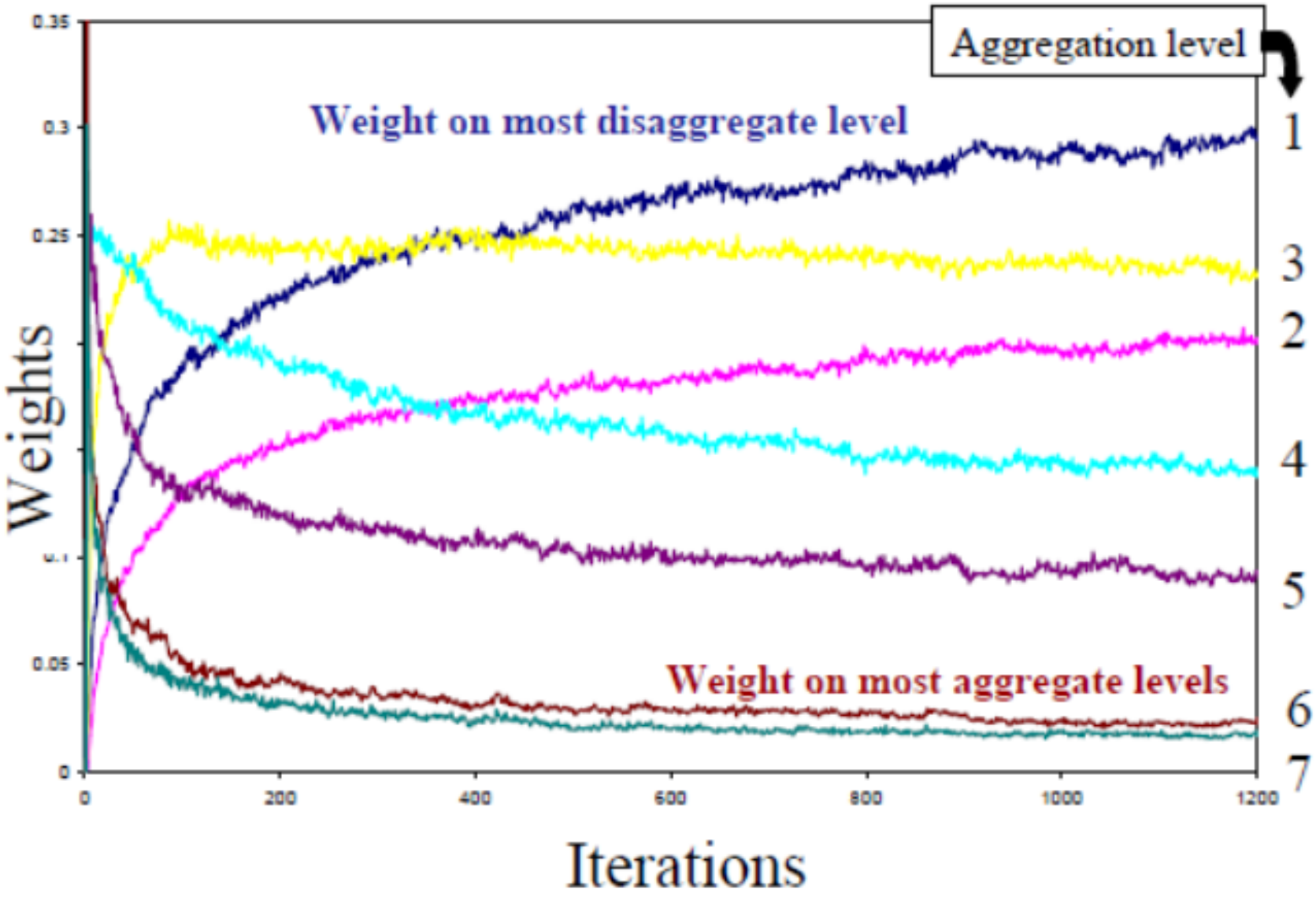
This bias arises because the observations $\hat{v}^n$ may be steadily increasing (or decreasing) with the iterations. When we smooth on past observations, we obtain an estimate $\bar{v}_s^{(g,n-1)}$ that tends to underestimate (overestimate if $\hat{v}^n$ tends to decrease) the true mean of $\hat{v}^n$.

The second form of bias is the aggregation bias given by the difference between the estimate at an aggregate level and the disaggregate level. We compute the aggregation bias using

$$\bar{\mu}_s^{(g,n)} = \bar{v}_s^{(g,n)} - \bar{v}_s^{(0,n)}. \tag{3.32}$$

Using the same reasoning presented in section 3.3.2, we can separate out the effect of bias to obtain an estimate of the variance of the error using

$$(s_s^2)^{(g,n)} = \frac{\bar{\nu}_s^{(g,n)} - (\bar{\beta}_s^{(g,n)})^2}{1+\lambda^{n-1}}. \tag{3.33}$$



Neural networks

Skip for now

Neural networks

- Start by illustrating with linear model

Up to now, we have considered approximation functions of the form

$$\overline{V}(S) = \sum_{f \in \mathcal{F}} \theta_f \phi_f(S),$$

where $\mathcal{F}$ is our set of features, and $(\phi_f(S))_{f \in \mathcal{F}}$ are the basis functions which extract what are felt to be the important characteristics of the state variable which explain the value of being in a state. We have seen that when we use an approximation that is linear in the parameters, we can estimate the parameters θ recursively using standard methods from linear regression. For example, if R_i is the number of resources of type i, our approximation might look like

$$\overline{V}(R|\theta) = \sum_{i \in \mathcal{I}} \left(\theta_{1i} R_i + \theta_{2i} R_i^2\right).$$

Now assume that we feel that the best function might not be quadratic in R_i, but we are not sure of the precise form. We might want to estimate a function of the form

$$\overline{V}(R|\theta) = \sum_{i \in \mathcal{I}} \left(\theta_{1i} R_i + \theta_{2i} R_i^{\theta_3}\right).$$

Now we have a function that is nonlinear in the parameter vector $(\theta_1, \theta_2, \theta_3)$, where θ_1 and θ_2 are vectors and θ_3 is a scalar. If we have a training dataset of state-value observations, $(\hat{v}_n, R_n)_{n=1}^N$, we can find θ by solving

$$\min_{\theta} \sum_{n=1}^{N} \left(\hat{v}_n - \overline{V}(R_n|\theta)\right)^2,$$

which generally requires the use of nonlinear programming algorithms. One challenge is that nonlinear optimization problems do not lend themselves to the simple recursive updating equations that we obtained for linear (in the parameters) functions. But more problematic is that we have to experiment with various functional forms to find the one that fits best.

Neural networks are, ultimately, a form of statistical model which, for our application, predicts the value of being in a state as a function of the state, using a series of observations of states and values. The simplest neural network is nothing more than a linear regression model. If we are in post-decision state S_t^a, we are going to make the random transition $S_{t+1} = S^M(S_t, a_t, W_{t+1}(\omega))$ and then observe a random value $\hat{v}_{t+1}$ from being in state S_{t+1}. We would like to estimate a statistical function $f_t(S_t^a)$ (the same as $\overline{V}_t(S_t^a)$) that predicts $\hat{v}_{t+1}$. To write this in the traditional notation of regression, let $X_t = S_t^a$ where $X_t = (X_{t1}, X_{t2}, \ldots, X_{tI})$ (we assume that all the components X_{ti} are numerical). If we use a linear model, we might write

$$f_t(X_t) = \theta_0 + \sum_{i=1}^{I} \theta_i X_{ti}.$$

In the language of neural networks, we have I inputs (we have $I + 1$ parameters since we also include a constant term), which we wish to use to estimate a single output $\hat{v}_{t+1}$ (a random observations of being in a state). The relationships are illustrated in figure 3.5 where we show the I inputs which are then "flowed" along the links to produce $f_t(X_t)$. After this, we then learn the sample realization $\hat{v}_{t+1}$ that we were trying to predict, which allows us to compute the error $\epsilon_{t+1} = \hat{v}_{t+1} - f_t(X_t)$. We would like to find a vector θ that solves

$$\min_{\theta} \mathbb{E}\frac{1}{2}(f_t(X_t) - \hat{v}_{t+1})^2.$$

Let $F(\theta) = \mathbb{E}(0.5(f_t(X_t) - \hat{v}_{t+1})^2)$, and let $F(\theta, \hat{v}_{t+1}(\omega)) = 0.5(f_t(X_t) - \hat{v}_{t+1}(\omega))^2$ where $\hat{v}_{t+1}(\omega)$ is a sample realization of the random variable $\hat{v}_{t+1}(\omega)$. As before, we can solve this iteratively using a stochastic gradient algorithm

$$\theta_{t+1} = \theta_t - \alpha_t \nabla_\theta F(\theta_t, \hat{v}_{t+1}(\omega)),$$

Neural networks

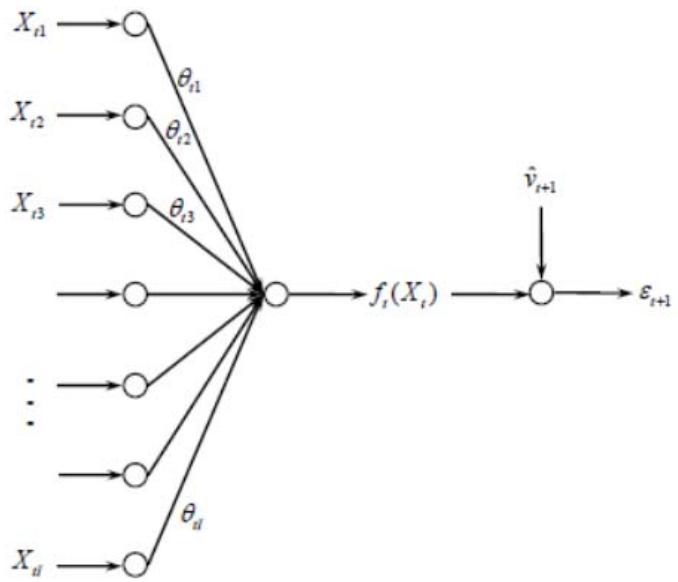
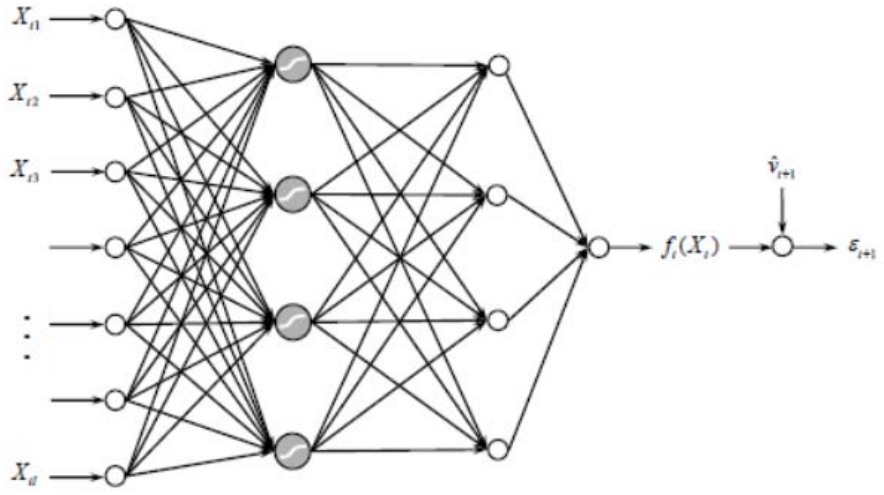


Figure 3.5 Neural networks with a single layer.



Neural networks

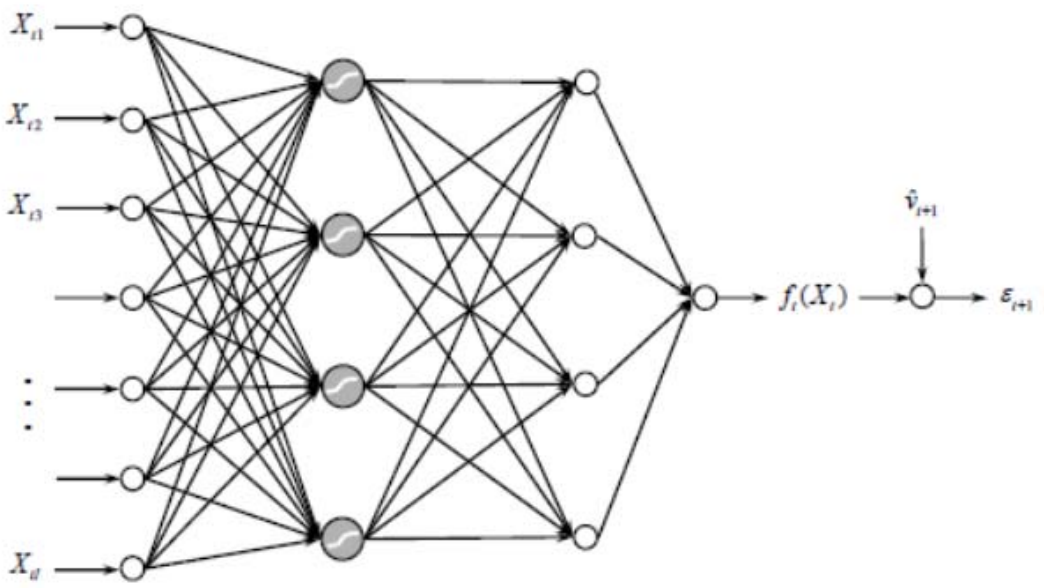


Figure 3.6 A three-layer neural network.

Here, the first linear layer produces J outputs given by

$$Y_{tj}^{(2)} = \sum_{i \in \mathcal{I}^{(1)}} \theta_{ij}^{(1)} X_{ti}^{(1)}, \quad j \in \mathcal{I}^{(2)}.$$

$Y_{tj}^{(2)}$ becomes the input to a nonlinear *perceptron* node which is characterized by a nonlinear function that may dampen or magnify the input. A typical functional form for a perceptron

Neural networks

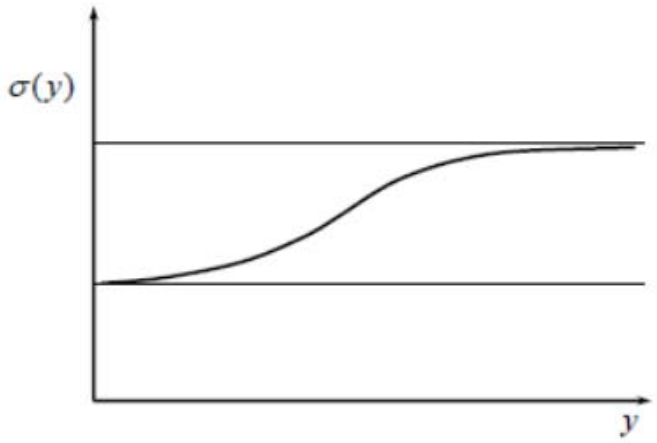


Figure 3.7 Illustrative logistics function for introducing nonlinear behavior into neural networks.

node is the logistics function given by

$$\sigma(y) = \frac{1}{1+e^{-\beta y}},$$

where β is a scaling coefficient. The function $\sigma(y)$ is illustrated in figure 3.7. $\sigma(x)$ introduces nonlinear behavior into the communication of the "signal" X_t.

We next calculate

$$X_{ti}^{(2)} = \sigma(Y_{ti}^{(2)}), \quad i \in \mathcal{I}^{(2)}$$

and use $X_{ti}^{(2)}$ as the input to the second linear layer. We then compute

$$Y_{tj}^{(3)} = \sum_{i \in \mathcal{I}^{(2)}} \theta_{ij}^{(2)} X_{ti}^{(2)}, \quad j \in \mathcal{I}^{(3)}.$$

Finally, we compute the single output using

$$f_t = \sum_{i \in \mathcal{I}^{(3)}} \theta_i^{(3)} X_{ti}^{(3)}.$$

Nonparametric methods

- Definition:
  - » A nonparametric model has the property that as the amount of data goes to infinity, modeling errors go to zero.

- Illustrate:
  - » K-nearest neighbor
  - » Kernel regression

Nonparametric methods

3.7.1 K-nearest neighbor

Perhaps the simplest form of nonparametric regression forms estimates of functions by using a weighted average of the k-nearest neighbors. As above, we assume we have a response y^n corresponding to a measurement $x^n = (x_1^n, x_2^n, \ldots, x_I^n)$. Let $\rho(x, x^n)$ be a distance metric between a query point x (in dynamic programming, this would be a state), and an observation x^n. Then let $\mathcal{N}^n(x)$ be the set of the k-nearest points to the query point x, where clearly we require $k \leq n$. Finally let $\bar{Y}^n(x)$ be the response function, which is our best estimate of the true function $Y(x)$ given the observations $x^1, \ldots, x^n$. When we use a k-nearest neighbor model, this is given by

$$\bar{Y}^n(x) = \frac{1}{k} \sum_{n \in \mathcal{N}^n(x)} y^n. \tag{3.64}$$

Thus, our best estimate of the function $Y(x)$ is made by averaging the k points nearest to the query point x.

Using a k-nearest neighbor model requires, of course, choosing k. Not surprisingly, we obtain a perfect fit of the data by using $k = 1$ if we base our error on the training dataset.

A weakness of this logic is that the estimate $\bar{Y}^n(x)$ can change abruptly as x changes continuously, as the set of nearest neighbors changes. An effective way of avoiding this behavior is using kernel regression, which uses a weighted sum of all data points.

Nonparametric models

Nonparametric methods

3.7.2 Kernel regression

Kernel regression has attracted considerable attention in the statistical learning literature. As with k-nearest neighbor, kernel regression forms an estimate $\bar{Y}(x)$ by using a weighted sum of prior observations which we can write generally as

$$\bar{Y}^n(x) = \frac{\sum_{m=1}^n K_h(x, x^m) y^m}{\sum_{m=1}^n K_h(x, x^m)} \tag{3.66}$$

where $K_h(x, x^m)$ is a weighting function that declines with the distance between the query point x and the measurement x^m. h is referred to as the *bandwidth* which plays an important scaling role. There are many possible choices for the weighting function $K_h(x, x^m)$. One of the most popular is the Gaussian kernel, given by

$$K_h(x, x^m) = e^{-\left(\frac{\|x - x^m\|}{h}\right)^2}.$$

where $\|\cdot\|$ is the Euclidean norm. Here, h plays the role of the standard deviation. Note that the bandwidth h is a tunable parameter that captures the range of influence of a measurement x^m. The Gaussian kernel, often referred to as *radial basis functions* in the ADP literature, provide a smooth, continuous estimate $\bar{Y}^n(x)$. Another popular choice of kernel function is the symmetric Beta family, given by

$$K_h(x, x^m) = \max(0, (1 - \|x - x^m\|)^2)^h.$$

Here, h is a nonnegative integer. $h = 1$ gives the uniform kernel; $h = 2$ gives the Epanechnikov kernel; and $h = 3$ gives the biweight kernel. Figure 3.8 illustrates each of these four kernel functions.

We pause to briefly discuss some issues surrounding k-nearest neighbors and kernel regression. First, it is fairly common in the ADP literature to see k-nearest neighbors

Nonparametric methods

- Kernels

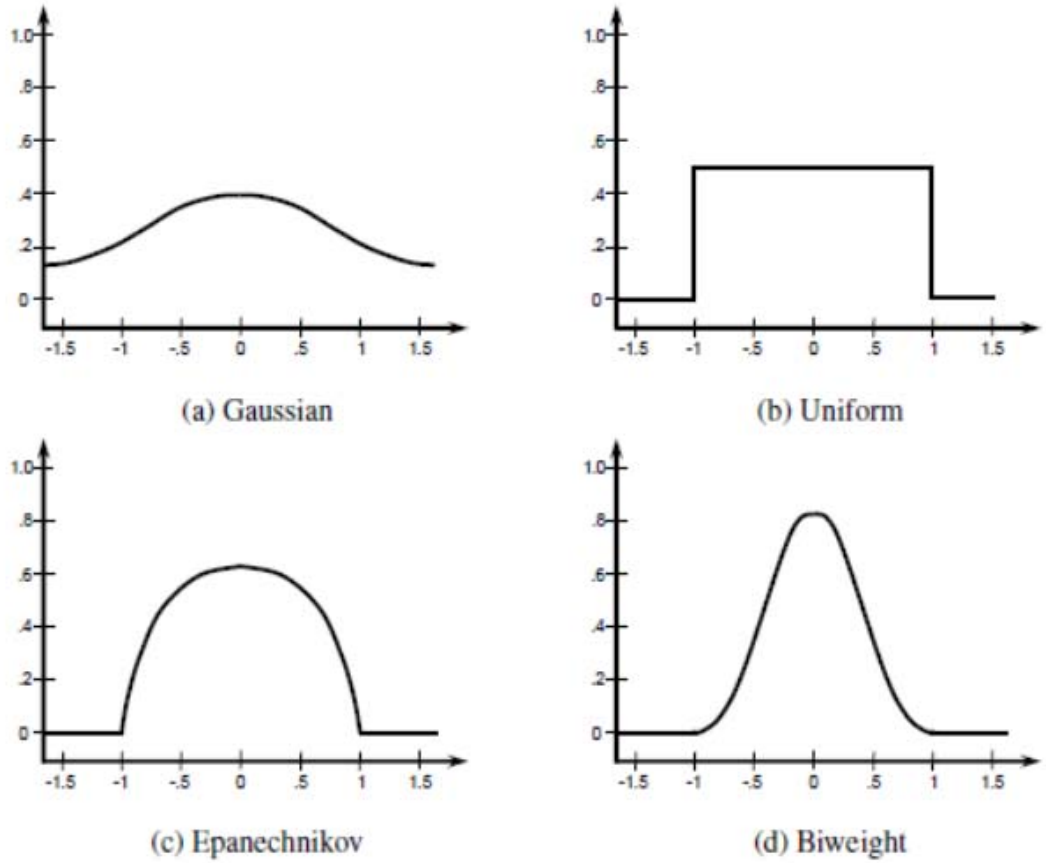
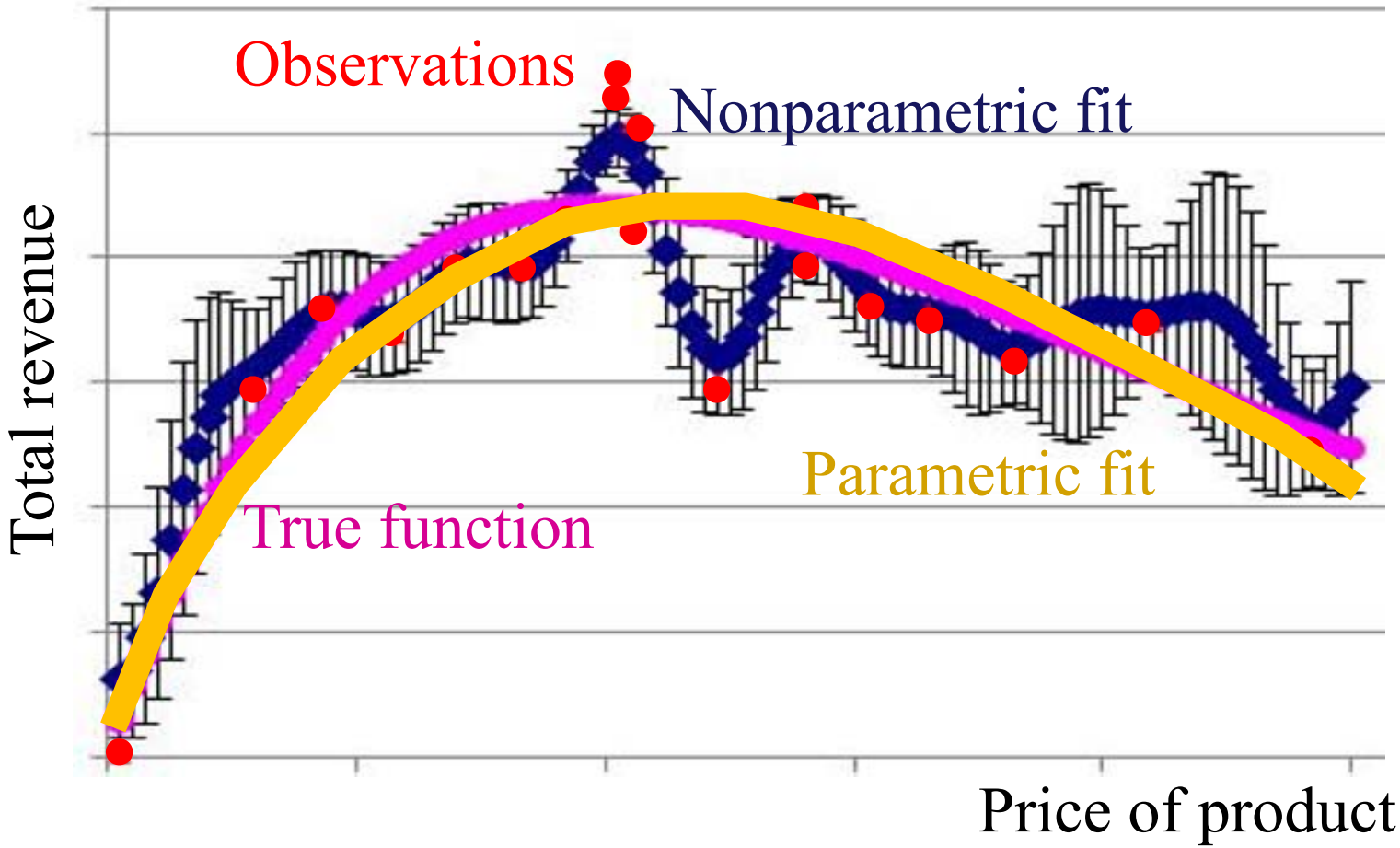


Figure 3.8 Illustration of Gaussian, uniform, Epanechnikov and biweight kernel weighting functions.

Approximating a function

- Parametric vs. nonparametric

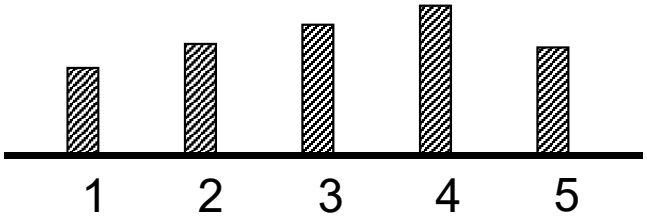


» Robust CFAs are *parametric*

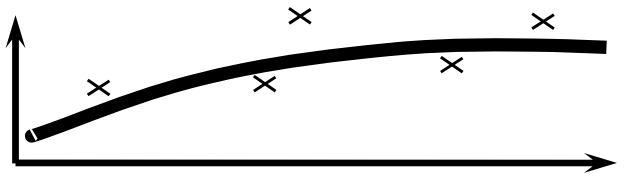
» Scenario trees are *nonparametric*

Nonparametric methods

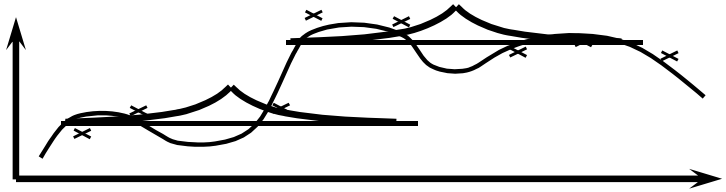
- Lookup table belief models (Frazier)
  - » Independent beliefs
  - » Correlated beliefs



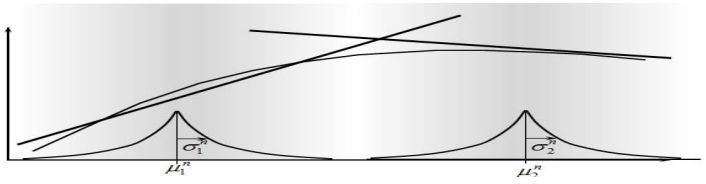
- Linear, parametric belief models (Frazier)



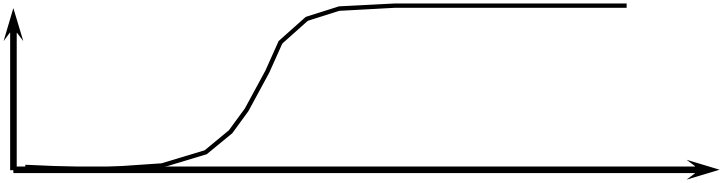
- Nonparametric models
  - » Hierarchical aggregation (Mes)
  - » Kernel regression (Barut)



- Local parametric models
  - » Dirichlet clouds with RBF (Jamshidi)
  - » KG derivation (Harvey Cheng)

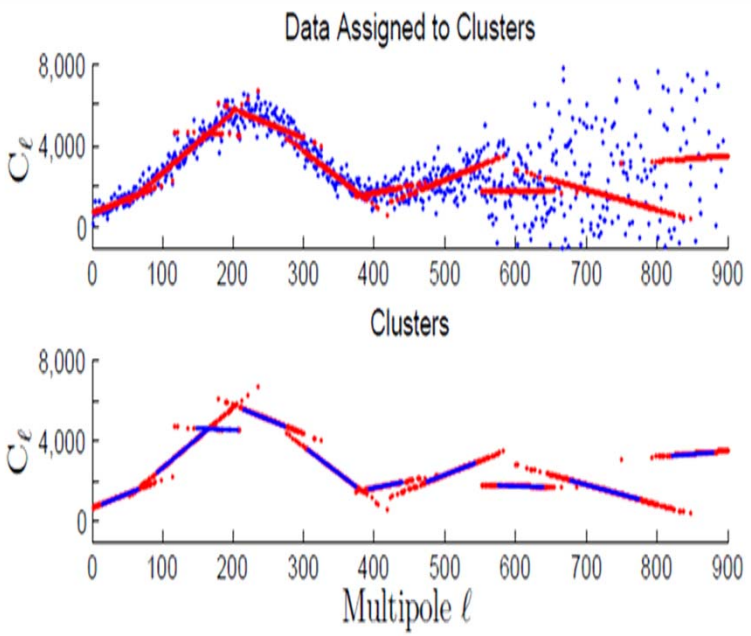


- Generalized linear models
  - » KG derivation (Si Chen)

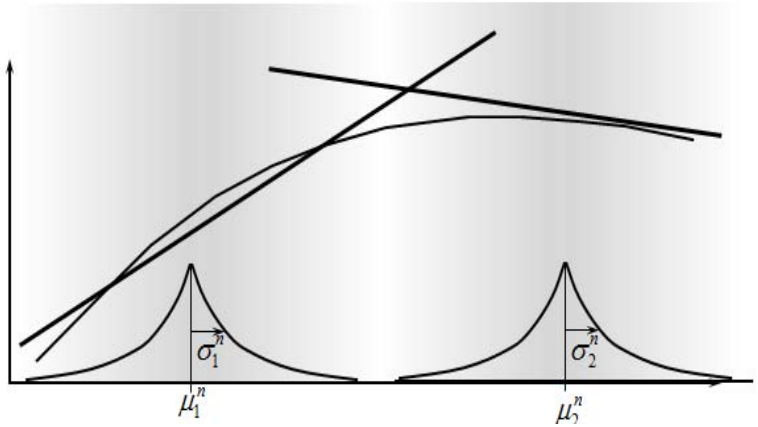


Locally parametric approximations

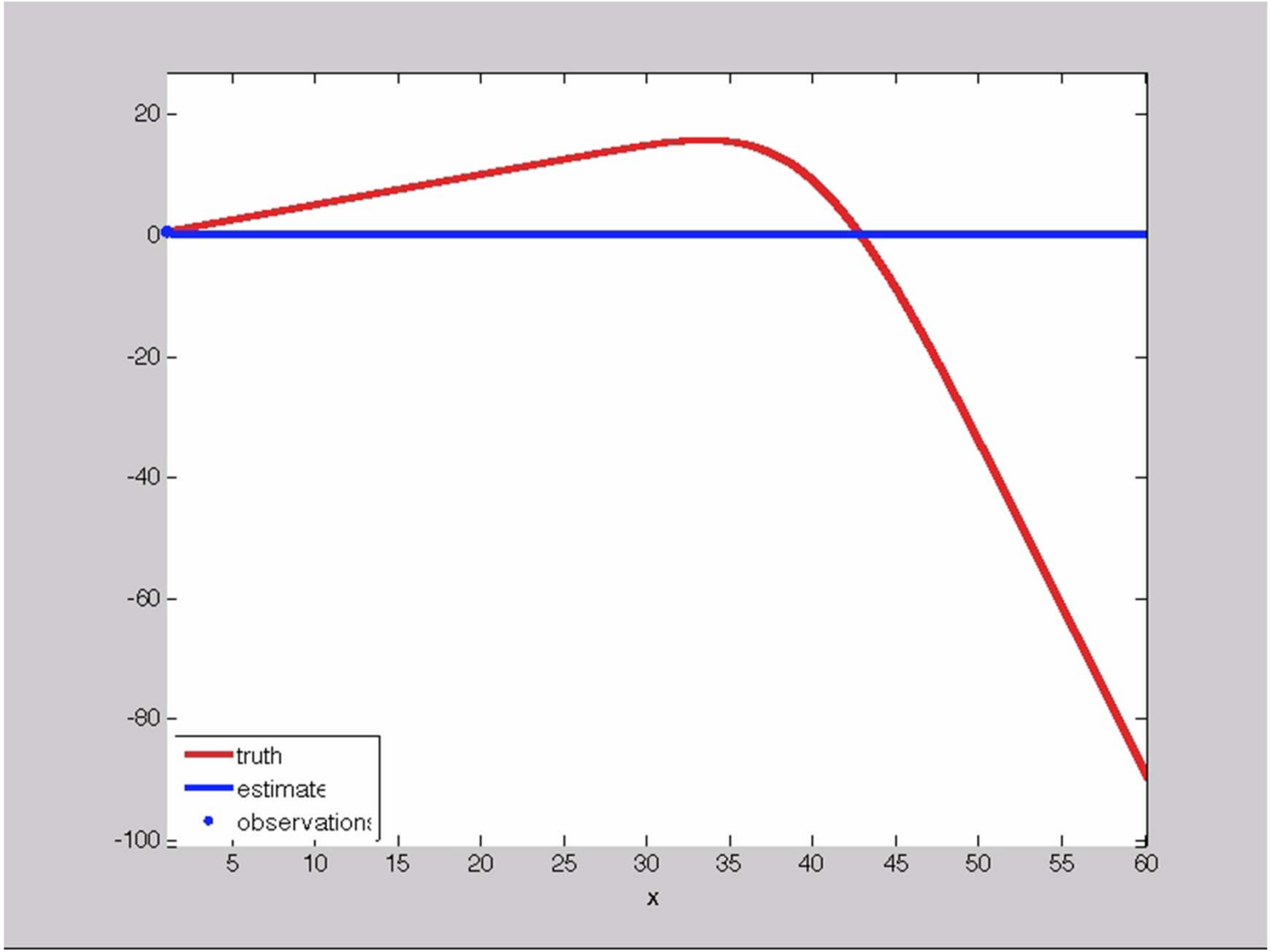
- Dirichlet process mixtures of generalized linear regression models (Hannah, Blei and WBP, 2011).
  - » Very general, but very slow.
  - » Cannot be implemented recursively.



- Dirichlet clouds using radial basis functions (Jamshidi and WBP, 2012)
  - » Local parametric approximations.
  - » Produces variable order approximation that adapts to the function.



Locally parametric approximations



Nonparametric methods

- Discussion:
  - » Local smoothing methods struggle with curse of dimensionality. In high dimensions, no two data points are ever "close."
  - » Nonparametric representations are very high-dimensional, which makes it hard to "store" a model.

Nonparametric methods

- More advanced methods
  - » Deep neural networks
    - Before, we described neural networks as a nonlinear parametric model.
    - Deep neural networks, which have the property of being able to approximate any function, are classified as nonparametric.
    - These have proven to be very powerful on image processing and voice recognition, but not in stochastic optimization.
  - » Support vector machines (classification)
  - » Support vector regression (continuous)
    - We have had surprising but limited success with SVM, but considerably more empirical research is needed.

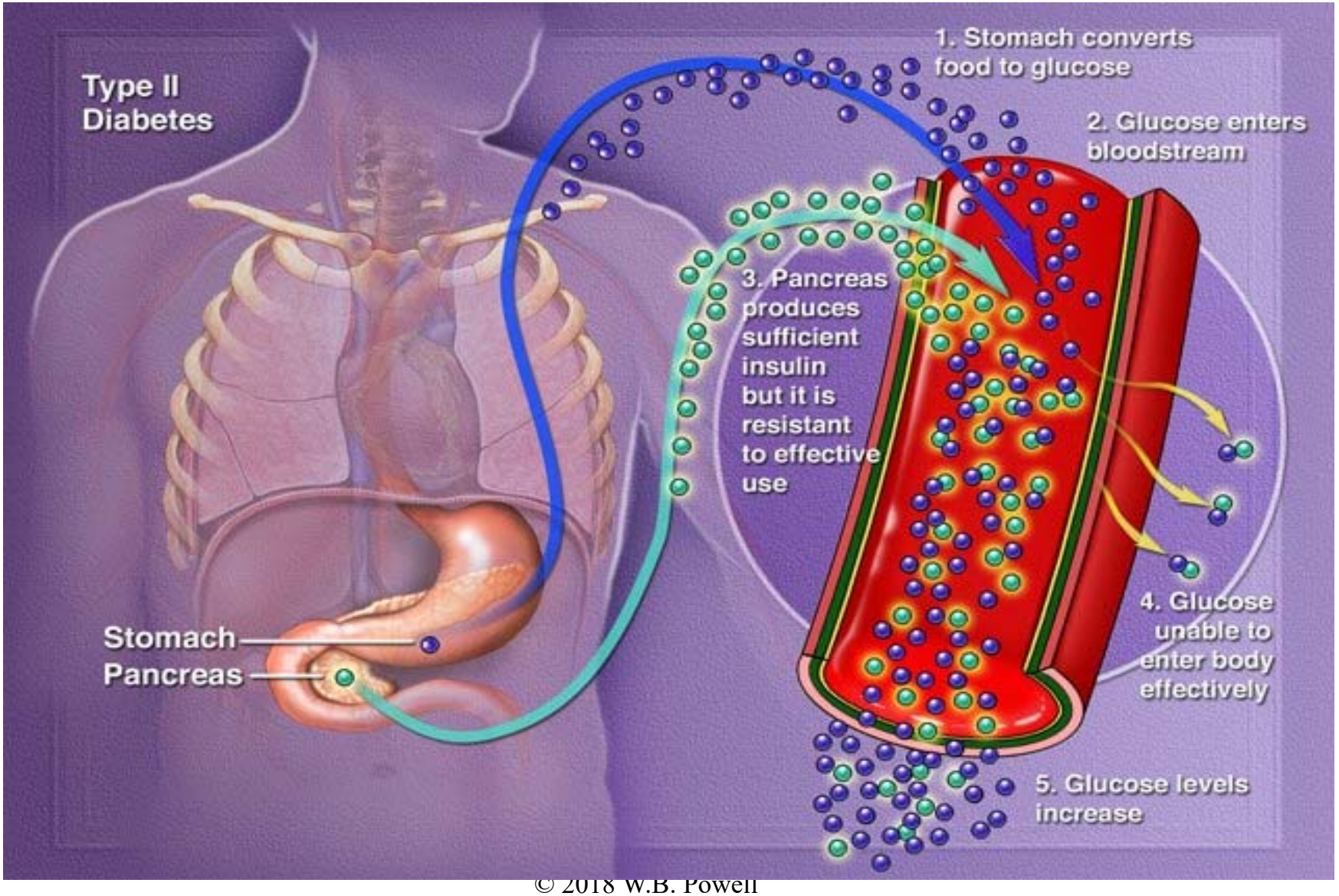
Week 2 - Monday

Learning diabetes medication

Narrative

Narrative

- Type 2 diabetes: Insulin resistance



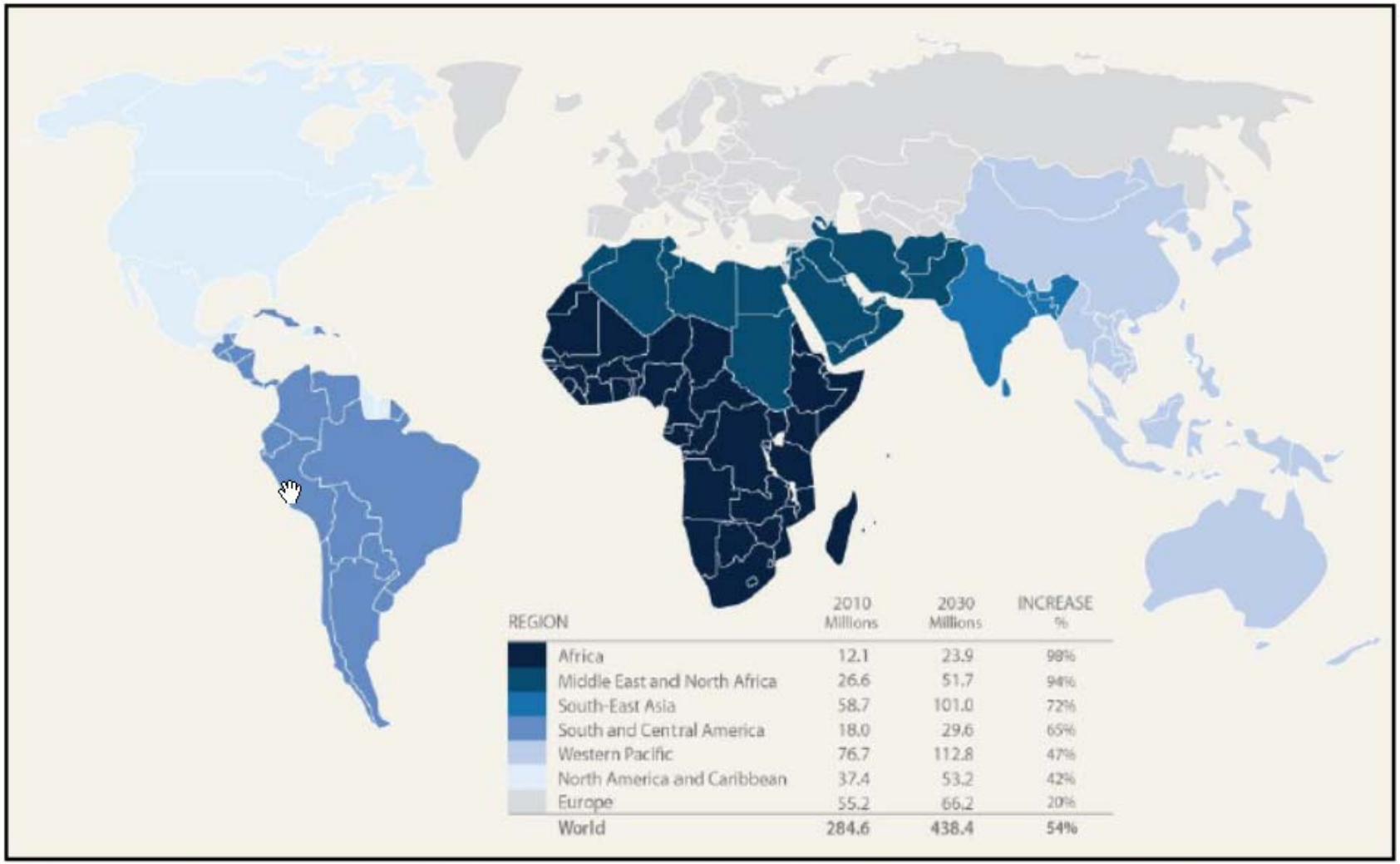


Figure 1.1 International Diabetes Federation (2009) regions and global projections for the number of people with diabetes (20-79 years), 2010-2030.

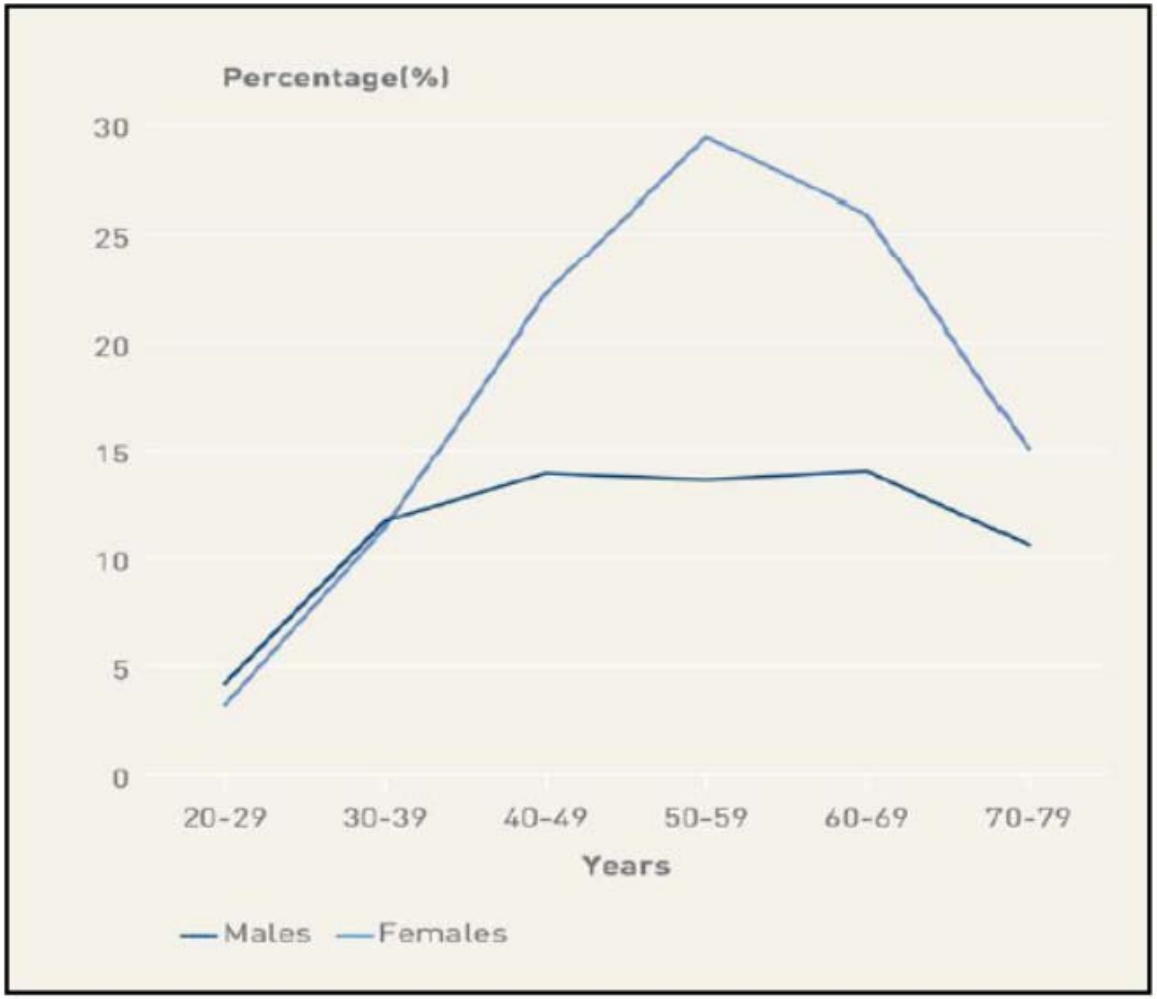


Figure 1.3 Percentage of all-cause mortality attributable to diabetes by age and sex, 2010, North America & Caribbean Region (International Diabetes Federation 2009).

Diabetes medications

Class	Drug	Target Organ	Action	Effect on HbA1c	Side effects	Contraindications
Sensitizer	Metformin	Liver, muscle, and fat	Lowers glucose production, increases insulin receptors	1.5-2%	Bloating, fullness, nausea, cramping	Kidney failure
	TZDs	PPARs in liver, muscle, and fat	Decreases insulin resistance	0.5-1.5%	Weight gain, fluid retention (edema)	Overweightedness, heart/liver failure
Secretagogues	Sulfonylureas	Pancreas β-cells	Increases insulin release (slow-acting)	1-2%	Hypoglycemia, weight gain	Overweightedness, age
	Glinides	Pancreas β-cells	Increases insulin resistance (fast-acting)	0.5-2%		
Alpha-glucosidase	Acarbose	Intestine	Slows down the metabolism of carbohydrates	0.7-1%	Bloating, nausea, diarrhea, abdominal pain	None
Peptide Analogs	GLP-1 agonists, DPP-4 inhibitors	Intestine	Increases GLP-1 activity	0.5-1%	Nausea	Unknown

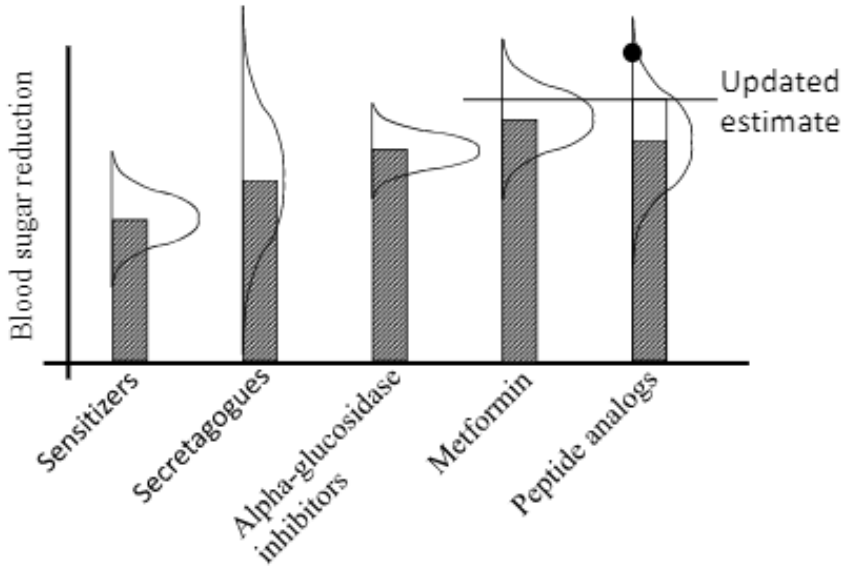


Figure 4.1 Beliefs about the potential that each drug might have on the reduction of blood sugar.

Drug	A1C reduction	Std. dev.
Metformin	0.32	0.12
Sensitizers	0.28	0.09
Secretagoges	0.30	0.17
Alpha-glucosidase inhibitors	0.26	0.15
Peptide analogs	0.21	0.11

Table 4.1 Metformin and the four drug classes and the average reduction over the full population.

4.1 NARRATIVE

When people find they have high blood sugar, typically evaluated using a metric called the "A1C" level, there are several dozen drugs that fall into four major groups:

- Sensitizers - These target liver, muscle, and fat cells to directly increase insulin sensitivity, but may cause fluid retention and therefore should not be used for patients with a history of kidney failure.

- Secretagoges - These drugs increase insulin sensitivity by targeting the pancreas but often causes hypoglycemia and weight gain.

- Alpha-glucosidase inhibitors - These slow the rate of starch metabolism in the intestine, but can cause digestive problems.

- Peptide analogs - These mimic natural hormones in the body that stimulate insulin production.

The most popular drug is a type of sensitizer called metformin, which is almost always the first medication that is prescribed for a new diabetic, but this does not always work. Prior to working with a particular patient, a physician may have a belief about the potential of metformin, and drugs from each of the four groups, to reduce blood sugar that is illustrated in figure 4.1.

Basic model

State variables

Our state variable is our belief about the random variable μ_x which is the true effect of each drug on a particular patient after n trials. S^0 is the initial state, which we write as

$$S^0 = (\bar{\mu}_x^0, \bar{\sigma}_x^0)_{x \in \mathcal{X}},$$

where we also include in S^0 the assumption of normality, which remains through all the experiments. After n experiments, the state is

$$S^n = (\bar{\mu}_x^n, \bar{\sigma}_x^n)_{x \in \mathcal{X}},$$

where we no longer include the normality assumption because it is captured in our initial state (the distribution is static, so by convention we do not include it in the dynamic state variable).

Later, we are going to find it useful to work with the *precision* of our belief, which is given by

$$\beta_x^n = \frac{1}{(\bar{\sigma}_x^n)^2}.$$

We can then write our state variable as

$$S^n = (\bar{\mu}_x^n, \beta_x^n)_{x \in \mathcal{X}}.$$

Decision variables

The decision is the choice of medication to try for a month, which we write as

$$\begin{aligned} x^n &= \text{The choice of medication,} \\ &\in \mathcal{X} = \{x_1, \ldots, x_M\}. \end{aligned}$$

We are going to determine x^n using a policy $X^\pi(S^n)$ that depends only on the state variable S^n (along with the assumption of the normal distribution in S^0).

Exogenous information

After making the decision x^n, we observe

$$W_x^{n+1} = \text{The reduction in the A1C level resulting from the drug } x = x^n \text{ we prescribed for the } n+1st \text{ trial.}$$

A reader might wonder why we write the information learned from the decision x^n is written W_x^{n+1} rather than W_x^n. We do this to capture the information available in each variable. Thus, the decision x^0 depends only on the initial state S^0. The state S^n for $n \geq 1$ depends on S^0 along with the observations $W_{x^0}^1, \ldots, W_{x^{n-1}}^n$, but not $W_{x^n}^{n+1}$, since we have not yet made the decision x^n. By letting $W_{x^n}^{n+1}$ be the result of the prescription x^n, we know that x^n cannot depend on W^{n+1}, which would be like seeing into the future.

- Discuss sources of uncertainty/randomness
  - » The truth μ_x
    - Prior $\mu_x \sim N(\bar{\mu}^0, \bar{\sigma}^{2,0})$
    - Prior is stored in the initial state S^0
  - » The observation of the truth ϵ_x^{n+1}
  - » Computing an observation:

$$W_x^{n+1} = \mu_x + \epsilon_x^{n+1}$$

  - » Describe how to run simulations
    - Generate μ
    - Generate series of $\epsilon^1, \ldots, \epsilon^N$

Transition function

The transition function captures how the observed reduction in A1C, W_x^{n+1}, affects our belief state S^n. Although it takes a little algebra, it is possible to show that if we try drug $x = x^n$ and observe W_x^{n+1}, we can update our estimate of the mean and precision using

$$\bar{\mu}_x^{n+1} = \frac{\beta_x^n \bar{\mu}_x^n + \beta_x^W W_x^{n+1}}{\beta_x^n + \beta_x^W}, \tag{4.1}$$

$$\beta_x^{n+1} = \beta_x^n + \beta^W. \tag{4.2}$$

For all $x \neq x^n$, $\bar{\mu}_x^n$ and β_x^n remain unchanged.

The transition function which we earlier wrote as a generic function

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

in equation (1.1), is given by equations (4.1) - (4.2).

Objective function

Each time we prescribe a drug $x = x^n$, we observe the reduction in the A1C represented by $W_{x^n}^{n+1}$. We want to find a policy that chooses a drug $x^n = X^\pi(S^n)$ that maximizes the expected total reduction in A1C. Our canonical model used $C(S^n, x^n, W^{n+1})$ as our performance metric. For this problem, this would be

$$C(S^n, x^n, W^{n+1}) \quad = \quad W_{x^n}^{n+1}.$$

We write the problem of finding the best policy as

$$\max_\pi \mathbb{E}\left\{\sum_{n=0}^{N-1} W_{x^n}^{n+1} | S_0\right\}, \tag{4.3}$$

where $x^n = X^\pi(S^n)$, and $S^{n+1} = S^M(S^n, x^n, W^{n+1})$. Here, the conditioning on S_0 is particularly important because it carries the prior distribution of belief.

Uncertainty modeling

4.3 MODELING UNCERTAINTY

Sampling random outcomes for our asset selling problem was relatively simple. For our medical setting, generating outcomes of the random variables $W^1, \ldots, W^n, \ldots$ is a bit more involved.

With the asset selling problem, we were generating random variables with mean 0 and a given variance which we assumed was known. In this medical application, the reduction in the A1C from a particular drug is a noisy observation of the true mean μ_x (for a particular patient) which we can write as

$$W^{n+1} = \mu_x + \varepsilon^{n+1},$$

where ε^{n+1} is normally distributed with mean 0 and a variance (which we assume is known) given by $(\sigma^W)^2$. The real issue is that we do not know μ_x. Given what we know after n experiments with different drugs, we assume that μ_x is normally distributed with mean $\bar{\mu}_x^n$ and precision β_x^n. We write this as

$$\mu_x | S^n \sim N(\bar{\mu}_x^n, \beta_x^n)$$

where we use the precision β_x^n instead of the more customary variance when we write our normal distribution. We then write the distribution of W^{n+1} as conditioned on μ_x using

$$W^{n+1} | \mu_x \sim N(\mu_x, \beta_x^W).$$

This means that we have to simulate two random variables: the true performance of drug x on our patient, given by μ_x (given our beliefs after n experiments), and then the noise ε^{n+1} when we try to observe μ_x. This just means that instead of generating one normally distributed random variable, as we did in our asset selling problem, we have to generate two.

- Notes:
  - » We can think of the truth μ_x as a normally distributed random variable, but we might also think of it as a sampled set of values:

$$(\mu^1, \mu^2, \ldots, \mu^K)$$

  - » Might assume that each is equally likely.
  - » First sample a truth μ^k ...
  - » Then sample the noise ϵ^{n+1}. This is usually normally distributed, but a uniform distribution can be convenient if we want to avoid negative values for W^{n+1}.

Designing policies

Learning policies

- Policies
  - » **Pure exploitation** – Always make the choice that appears to be the best.

$$X^{exploit}(S^n) = argmax_x \bar{\mu}_x^n$$

  - » But we only learn about drugs we try. The actual truth for a drug may be higher than our estimate.

4.4 DESIGNING POLICIES

A popular class of policies for this problem class is organized under the general name of *upper confidence bounding*. One of the first UCB policies had the form

$$X^{UCB}(S^n) = \arg\max_{x\in\mathcal{X}} \left(\bar{\mu}_x^n + 4\sigma^W\sqrt{\frac{\log n}{N_x^n}}\right), \tag{4.4}$$

where N_x^n is the number of times we have tried drug x. It is standard practice to replace the coefficient $4\sigma^W$ by a tunable parameter, giving us

$$X^{UCB}(S^n|\theta^{UCB}) = \arg\max_{x\in\mathcal{X}} \left(\bar{\mu}_x^n + \theta^{UCB}\sqrt{\frac{\log n}{N_x^n}}\right), \tag{4.5}$$

A popular variant that we have found works surprisingly well was originally introduced under the name *interval estimation*, which his given by

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_{x\in\mathcal{X}} \left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right), \tag{4.6}$$

Evaluating policies

4.5 POLICY EVALUATION

We originally wrote our objective function as

$$\max_{\pi} \mathbb{E}\left\{\sum_{n=0}^{N-1} W_{x^n}^{n+1} | S_0\right\},$$

but writing the expectation in this way is a bit vague. Recall that we have two sets of random variables: the true values of μ_x for all $x \in \mathcal{X}$, and the observations $W^1, \ldots, W^N$ (or more precisely, the noise when we try to observe μ_x). We can express this nested dependence by writing the objective function as

$$\max_{\pi} \mathbb{E}_{\mu} \mathbb{E}_{W^1,\ldots,W^N|\mu}\left\{\sum_{n=0}^{N-1} W_{x^n}^{n+1} | S_0\right\}, \tag{4.7}$$

There are two ways to simulate the value of a policy:

Nested sampling First we simulate the value of the truth μ_x for all $x \in \mathcal{X}$ where we let $\psi \in \Psi$ be a sample realization of μ, which we write as $\mu(\psi)$. We then simulation the observations W, where we let $\omega \in \Omega$ be a sample realization of $W^1(\omega), \ldots, W^N(\omega)$, which means that ω is an outcome of all possible observations over all possible drugs $x \in \mathcal{X}$, over all experiments $n = 1, \ldots, N$.

Simultaneous sampling Here, we let ω be a sample realization of both μ_x and the observations $W^1, \ldots, W^N$.

Nested sampling

If we use nested sampling, assume that we generate K samples of the true values $\mu(\psi_k)$, and L samples of the errors $\varepsilon^1(\omega_\ell), \ldots, \varepsilon^N(\omega_\ell)$. For the sampled truth $\mu(\psi_k)$ and noise $\varepsilon^n(\omega_\ell)$, the performance of drug x^n in the $n+1st$ experiment would be

$$W_{x^n}^{n+1}(\psi_k, \omega_\ell) \quad = \quad \mu(\psi_k) + \varepsilon^n(\omega_\ell).$$

We can then compute a simulated estimate of the expected performance of a policy using

$$\overline{F}^\pi \quad = \quad \frac{1}{K}\sum_{k=1}^{K}\left(\frac{1}{L}\sum_{\ell=1}^{L}\sum_{n=0}^{N-1} W_{x^n}^{n+1}(\psi_k, \omega_\ell)\right),$$

where $x^n = X^\pi(S^n)$ and $S^{n+1}(\psi_k, \omega_\ell) = S^M(S^n(\psi_k, \omega_\ell), X^\pi(S^n(\psi_k, \omega_\ell)), W^{n+1}(\psi_k, \omega_\ell)$

Simultaneous sampling

If we use simultaneous sampling, then a sample ω determines both the truth $\mu(\omega)$ and the noise $\varepsilon(\omega)$, allowing us to write a sampled estimate of our observation $W_{x^n}^{n+1}$ as

$$W_{x^n}^{n+1}(\omega_\ell) \quad = \quad \mu(\omega_\ell) + \varepsilon^n(\omega_\ell).$$

The estimated value of a policy is given given by

$$\overline{F}^\pi \quad = \quad \frac{1}{L}\sum_{\ell=1}^{L}\sum_{n=0}^{N-1} W_{x^n}^{n+1}(\omega_\ell),$$

where $x^n = X^\pi(S^n)$ and $S^{n+1}(\omega_\ell) = S^M(S^n(\omega_\ell), X^\pi(S^n(\omega_\ell)), W^{n+1}(\omega_\ell))$.

If we use one of our parameterized policies where θ is the tunable parameter, we might write the expected performance as $\overline{F}^\pi(\theta)$ (technically, the parameter θ) as $\overline{F}^\pi(\theta)$. Then, the optimization problem would be

$$\max_\theta \overline{F}^\pi(\theta), \tag{4.8}$$

using any tunable policy.

Extensions

Extensions

- Contextual learning
  - » Learning about a particular patient
  - » Prior comes from population data
    - Let K^0 be prior knowledge about response to medications.
  - » nth Patient walks in with attribute a:
    - $a^n = (Gender\ G^n, age\ Y^n, ethnicity\ E^n)$
  - » Imagine that we just have gender G^n:
    - Sequence of states, decisions and information:

$$(K^0, G^1, S^1 = (K^0, G^1), x^1, W^1, K^1, G^2, S^2 = (K^1, G^1), \ldots,$$
$$K^{n-1}, G^n, S^n = (K^{n-1}, G^n), x^n, W^n, K^n, G^{n+1}, \ldots)$$

  - » If we use a lookup table belief model, everything we learn and actions we take depend on G.

Extensions

- Notational note:

We pause for a moment and note that our indexing is different from what we used in the basic model. In our basic model, the index n referred to visits by a patient. We make a decision x^n *after* the nth visit using what is known from the first n visits. We let W^{n+1} be the outcome of this treatment, incrementing n to $n+1$ to emphasize that x^n was computed without knowing W^{n+1}.

With our new model, however, n refers to a patient. It makes more sense to let G^n be the gender of the nth patient, at which point we make a decision for the nth patient, and let W^n be the outcome of the treatment for the nth patient. We do not increment n until we see the $n+1st$ patient, at which point we see the gender of the $n+1st$ patient.

Extensions

- Other extensions:
  - » Discuss what happens when we use a vector of patient attributes rather than just gender.
  - » What if we use a linear model:

4.4 Now imagine that our attribute space $\mathcal{A}$ is simply too large to be practical. What we have done up to now is a lookup table representation where we find an estimate $\bar{\mu}^n_{a,x}$, which becomes problematic when the number of possible values of a becomes large. An alternative approach is to use a parametric model. The simplest would be a linear model where we would write

$$\bar{\mu}_{a,x} = \sum_{f \in \mathcal{F}} \theta_f \phi_f(a,x),$$

where $\phi_f(a,x)$ for $f \in \mathcal{F}$ is a set of features that we (as analysts) would have to define. For example, one feature might just be an indicator of gender, or age range, or race. In this case, there would be a feature for each possible gender, each possible age range, and so on.

- If there are L possible values of each K attributes, what is the minimum number of features we would need?
- Suggest more complex features other than just those that indicate the value of each attribute.
- Contrast the strengths and weaknesses of a lookup table representation versus our linear model.

Week 2 - Wednesday

Thompson sampling

Knowledge gradient policies

Overview of policies

Learning policies

- Heuristic learning policies
  - » **Pure exploitation** – Always make the choice that appears to be the best.
  - » **Pure exploration** – Make choices at random so that you are always learning more.
  - » **Epsilon-greedy**
    - Explore with probability ε and exploit with probability $1-\varepsilon$
    - Epsilon-greedy exploration – explore with probability $\varepsilon^n = c/n$. Goes to zero as $n \rightarrow \infty$, but not too quickly.

- Discuss
  - » Convergence
  - » Applying to large problems

Learning policies

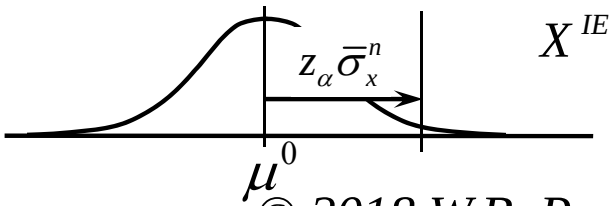
- Heuristic measurement policies
  - » **Boltzmann exploration**
    - Explore choice x with probability $P_x^n(\theta) = \dfrac{e^{\theta \bar{\mu}_x^n}}{\sum_{x'} e^{\theta \bar{\mu}_{x'}^n}}$
    - $X^{Boltz}(S^n|\theta) = \arg\max_x \{x | P_x^n(\theta) \leq U\}.\ U \sim [0,1]$
  - » **Upper confidence bounding**

$$X^{UCB}(S^n \mid \theta^{UCB}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{UCB} \sqrt{\frac{\log n}{N_x^n}} \right)$$

  - » **Thompson sampling – Highlight this**

$$X^{TS}(S^n) = \arg\max_x \hat{\mu}_x^{n+1} \quad \text{where } \hat{\mu}_x^n \sim N(\bar{\mu}_x^n, \beta_x^n)$$

  - » **Interval estimation** (or upper confidence bounding)
    - Choose x which maximizes



$$X^{IE}(S^n \mid \theta^{IE}) = \arg\max_x \bar{\mu}_x^n + \theta^{IE} \bar{\sigma}_x^n$$

- Thompson sampling
  - » Sample from the prior:
    - $\hat{\mu}_x^n \sim N(\bar{\mu}_x^n, \theta^{TS}\bar{\sigma}_x^{2,n})$

    where θ^{TS} is a tunable parameter
  - » Thompson sampling policy:
    - $X^{TS}(S^n|\theta^{TS}) = argmax_x \hat{\mu}_x^n$
  - » Notes:
    - Enjoys good optimality properties
    - Easy to implement, popular in internet applications
    - Suited to applications with high volumes of data (e.g. choosing news articles).

The knowledge gradient

Offline learning/final reward

4.1 THE VALUE OF INFORMATION

There are different ways of estimating the value of information, but one strategy, called the *knowledge gradient*, works as follows. Assume that we have a finite number of discrete alternatives with independent, normally distributed beliefs (the same problem we considered in chapter 3). After n experiments, we let $\bar{\mu}^n$ be our vector of estimates of means and β^n our vector of precisions (inverse variances). We represent our state of knowledge as $S^n = (\bar{\mu}_x^n, \beta_x^n)_{x \in \mathcal{X}}$. If we stop measuring now, we would pick the best option, which we represent by

$$x^n = \max_{x \in \mathcal{X}} \bar{\mu}_x^n.$$

The value of being in state S^n is then given by

$$V^n(S^n) = \bar{\mu}_{x^n}^n.$$

Now let $S^{n+1}(x)$ be the next state if we choose to measure $x^n = x$ right now, allowing us to observe $W_{x^n}^{n+1}$. This allows us to update $\bar{\mu}_x^n$ and β_x^n, giving us an estimate $\bar{\mu}_x^{n+1}$ for the mean and β_x^{n+1} for the precision (using equations (3.2) and (3.3)). Given that we choose

Now let $S^{n+1}(x)$ be the next state if we choose to measure $x^n = x$ right now, allowing us to observe $W_{x^n}^{n+1}$. This allows us to update $\bar{\mu}_x^n$ and β_x^n, giving us an estimate $\bar{\mu}_x^{n+1}$ for the mean and β_x^{n+1} for the precision (using equations (3.2) and (3.3)). Given that we choose to measure $x = x^n$, we transition to a new belief state $S^{n+1}(x)$, and the value of being in this state is now given by

$$V^{n+1}(S^{n+1}(x)) = \max_{x' \in \mathcal{X}} \bar{\mu}_{x'}^{n+1}(x).$$

Let $\bar{\mu}_{x'}^{n+1}(x)$ be the updated estimate of $\bar{\mu}_{x'}^n$ if we run experiment x' and observe $W_{x'}^{n+1}$.

$$\bar{\mu}_{x'}^{n+1}(x) = \begin{cases} \frac{\beta_{x'}^n \bar{\mu}_{x'}^n + \beta^W W_{x'}^{n+1}}{\beta_{x'}^n + \beta^W} & \text{If } x' = x, \\ \bar{\mu}_{x'}^n & \text{Otherwise} \end{cases}. \tag{4.2}$$

At time n when we have chosen $x = x^n$ as our next experiment, but before we have observed W_x^{n+1}, the estimate $\bar{\mu}_{x'}^{n+1}(x)$ is a random variable.

We would like to choose x at iteration n which maximizes the expected value of $V^{n+1}(S^{n+1}(x))$. At time n, we write this expectation as

$$\mathbb{E}\{V^{n+1}(S^{n+1}(x))|S^n\} = \mathbb{E}_\mu \mathbb{E}_{W|\mu}\{V^{n+1}(S^{n+1}(x))|S^n\},$$

where the right hand side brings out that there are two random variables: the truth μ (which is uncertain in a Bayesian belief model) and the outcome $W_x^{n+1} = \mu_x + \epsilon^{n+1}$, which depends on the unknown truth μ. We want to choose an experiment x that

$$\mathbb{E}\{V^{n+1}(S^{n+1}(x))|S^n\} \quad = \quad \mathbb{E}_\mu \mathbb{E}_{W|\mu}\{V^{n+1}(S^{n+1}(x))|S^n\},$$

where the right hand side brings out that there are two random variables: the truth μ (which is uncertain in a Bayesian belief model) and the outcome $W_x^{n+1} = \mu_x + \epsilon^{n+1}$, which depends on the unknown truth μ. We want to choose an experiment x that maximizes $\mathbb{E}\{V^{n+1}(S^{n+1}(x))|S^n\}$, but instead of writing our objective in terms of maximizing the value from an experiment, we are going to write it equivalent as maximizing the incremental value from the experiment, which is given by

$$\begin{aligned} \nu_x^{KG,n} &= \mathbb{E}_\mu \mathbb{E}_{W|\mu} \left\{V^{n+1}(S^{n+1}(x)) - V^n(S^n)|S^n\right\} \\ &= \mathbb{E}_\mu \mathbb{E}_{W|\mu} \left\{V^{n+1}(S^{n+1}(x))|S^n\right\} - V^n(S^n). \end{aligned} \tag{4.3}$$

Keep in mind that the state S^n is our belief about μ after n experiments, which is that $\mu \sim N(\bar{\mu}^n, \beta^n)$. Given S^n (that is, given what we know at time n), the value $V^n(S^n)$ is calculated just using our current estimates $\bar{\mu}^n$, as is done in equation (4.1). Thus, at time n, $V^n(S^n)$ is a number, which is why $\mathbb{E}\{V^n(S^n)|S^n\} = V^n(S^n)$. However, $V^{n+1}(S^{n+1}(x))$ is a random variable since it depends on the outcome of the $n+1st$ experiment W^{n+1}.

The right hand side of (4.3) can be viewed as the derivative (or gradient) of $V^n(S^n)$ with respect to the experiment x. Thus, we are choosing our experiment to maximize the gradient with respect to the knowledge gained from the experiment. For this reason we refer to $\nu_x^{KG,n}$ as the *knowledge gradient*. We write the knowledge

- Knowledge gradient

$$\nu_x^{KG,n} = \mathbb{E}_\mu \mathbb{E}_{W|\mu} \left\{ \max_{x'} \bar{\mu}_{x'}^{n+1}(x) \,|\, S^n \right\} - \max_{x'} \bar{\mu}_{x'}^n$$

» Discuss $\bar{\mu}_{x'}^{n+1}(x)$ as the estimated updated belief if we run experiment x .

» Think of this estimate as looking one step into the future.

» Think about how we might estimate the expectations using simulation:

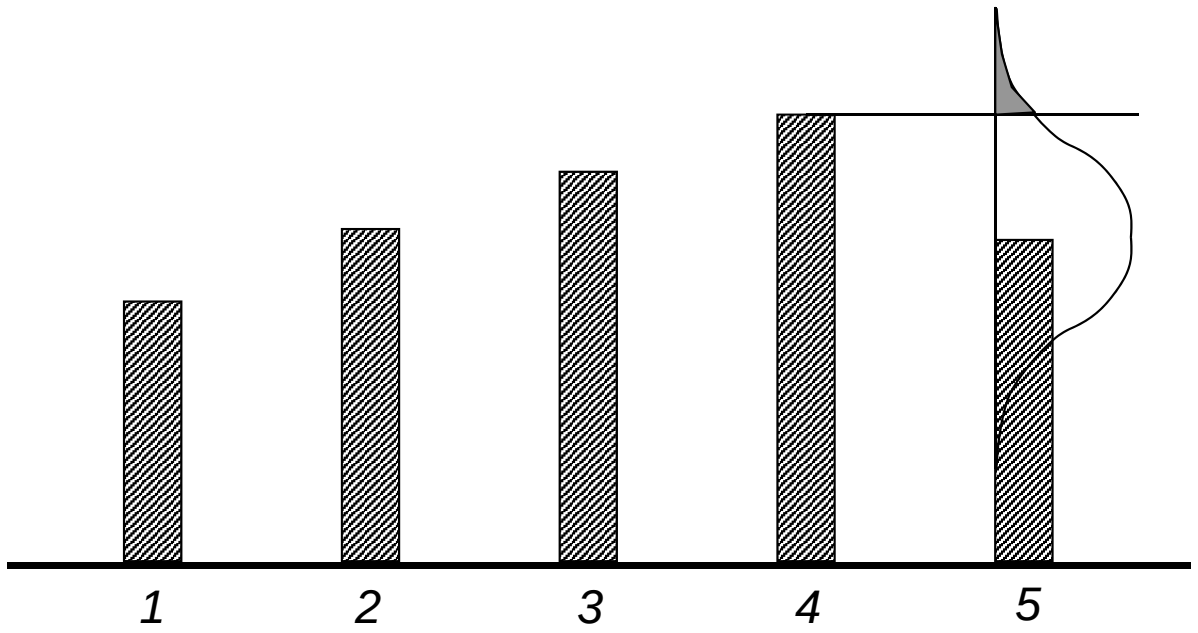
$$\nu_x^{KG,n} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{L}\sum_{l=1}^{L} \max_{x'} \bar{\mu}_{x'}^{n+1}(x \,|\, S^n, W_x^{l|k} = \mu_x^k + \varepsilon^l) - \max_{x'} \bar{\mu}_{x'}^n$$

» Review how means are updated given estimates in S^n (which contains $\bar{\mu}_x^n$) and the observation $W_x^{l|k}$ for a truth μ_x^k and noise ϵ^l

» Remember that μ_x^k is a possible *true value* of μ, not the estimate. We have an estimate, but the random outcome *W* comes from the *truth*.

Ranking and selection

- Knowledge gradient policy



» We want to measure the weighted area under the curve for option 5 that is over the value for option 4.

The knowledge gradient

- Derivation
  - » Notation

$$\beta_x^n = \text{Precision (inverse variance) of our estimate } \theta_x^n \text{ of the value of } x.$$
$$\beta^W = \text{Precision of the measurement noise } (=1/\sigma_W^2)$$
$$w_x^{n+1} = \text{Measurement of } x \text{ in iteration } n+1 \text{ (unknown at } n)$$

  - » We update the precision using

$$\beta_x^{n+1} = \beta_x^n + \beta^W$$

  - » In terms of the variance, this is the same as

$$\left(\sigma_x^{2,n+1}\right)^{-1} = \left(\sigma_x^{2,n}\right)^{-1} + \left(\sigma_W^2\right)^{-1}$$
$$\sigma_x^{2,n+1} = \frac{\sigma_x^{2,n}}{1+\sigma_x^{2,n}/\sigma_W^2}$$

The knowledge gradient

- Derivation
  - » The change in variance can be found to be

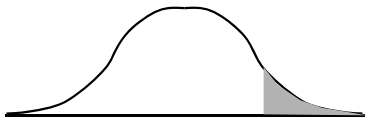
$$\begin{aligned}\tilde{\sigma}_x^{2,n} &= Var\left[\bar{\mu}_x^{n+1} - \bar{\mu}_x^n \mid S^n\right] \\ &= \sigma_x^{2,n} - \sigma_x^{2,n+1} \\ &= \frac{\sigma_x^{2,n}}{1+\sigma_W^2 / \sigma_x^{2,n}}\end{aligned}$$

  - » Next compute the *normalized influence*:

$$\zeta_x^n = -\left|\frac{\bar{\mu}_x^n - \max_{x' \neq x} \bar{\mu}_{x'}^n}{\tilde{\sigma}_x^n}\right|.$$

  - » Let $f(\zeta) = \zeta\Phi(\zeta) + \phi(\zeta)$ $\quad \Phi(\zeta) =$ Cumulative standard normal distribution

    $\phi(\zeta) =$ Standard normal density



  - » Knowledge gradient is computed using

$$\nu_x^{KG} = \tilde{\sigma}_x^n f\left(\zeta_x^n\right)$$

The knowledge gradient

- The normal distribution

We next compute

$$f(\zeta) = \zeta\Phi(\zeta) + \phi(\zeta), \tag{5.9}$$

where $\Phi(\zeta)$ and $\phi(\zeta)$ are, respectively, the cumulative standard normal distribution and the standard normal density. That is,

$$\phi(\zeta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{\zeta^2}{2}},$$

and

$$\Phi(\zeta) = \int_{-\infty}^{\zeta} \phi(x)dx.$$

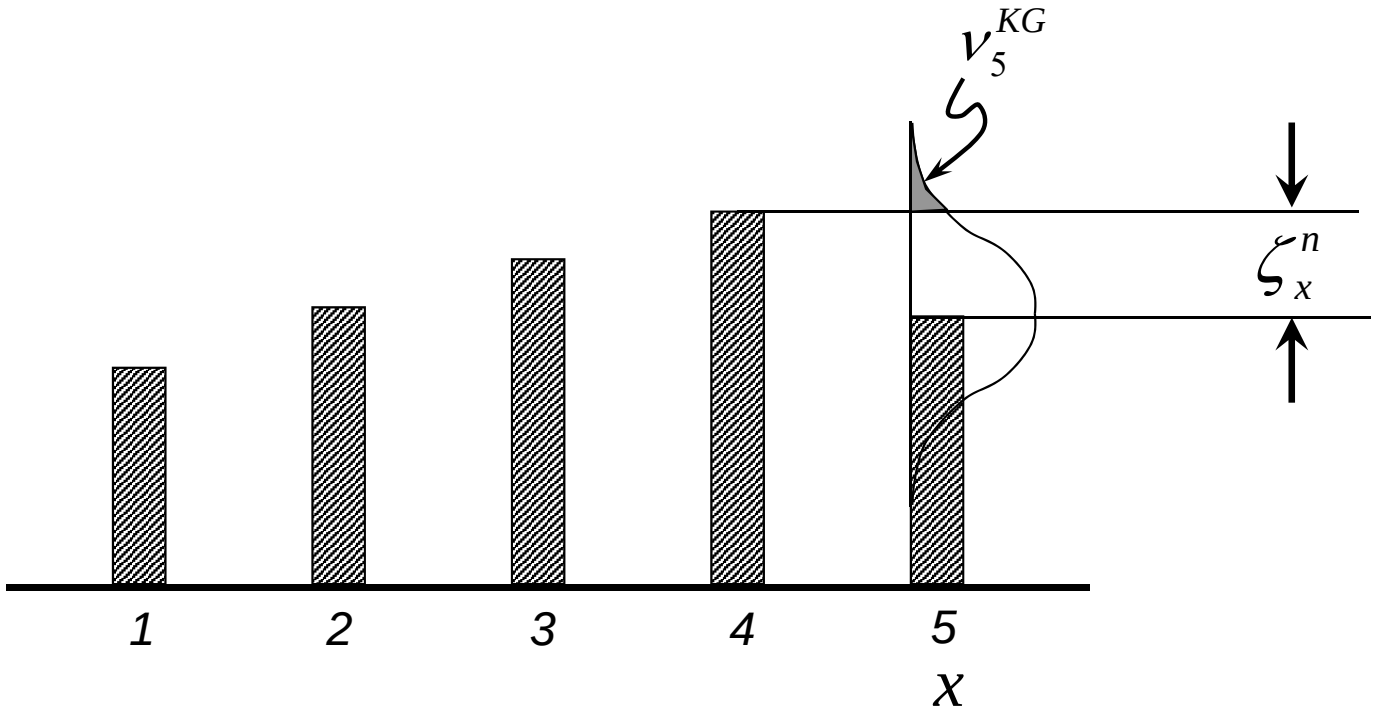
- Computing the cdf
  - » Matlab – normcdf(x,mu,sigma)
  - » Excel – normdist(x,mu,sigma,prob) z = standard normal

The knowledge gradient

- Knowledge gradient

$$\zeta_x^n = -\left|\frac{\bar{\mu}_x^n - \max_{x' \neq x} \bar{\mu}_{x'}^n}{\tilde{\sigma}_x^n}\right|$$

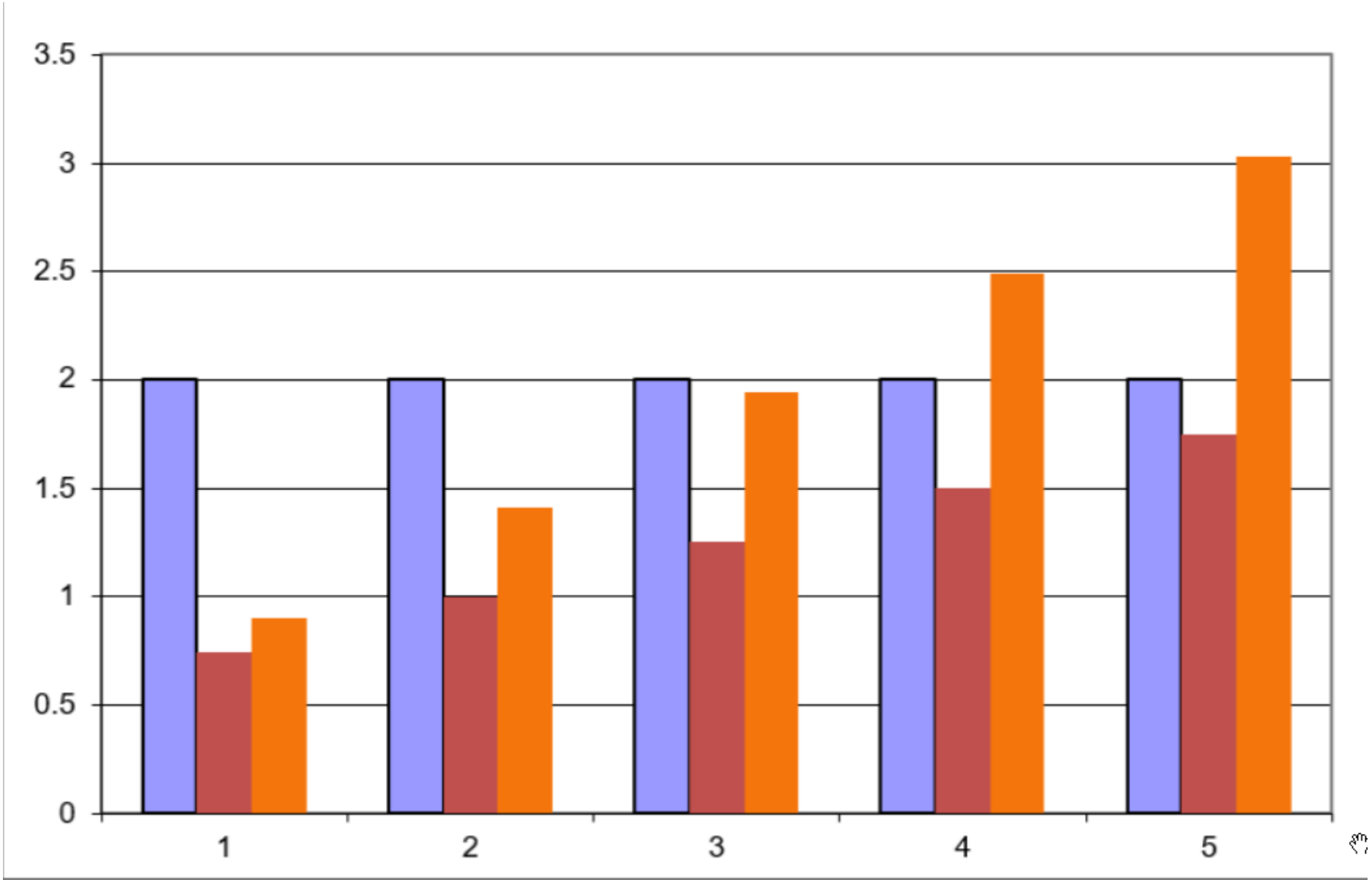
= Normalized distance to best (or second best)

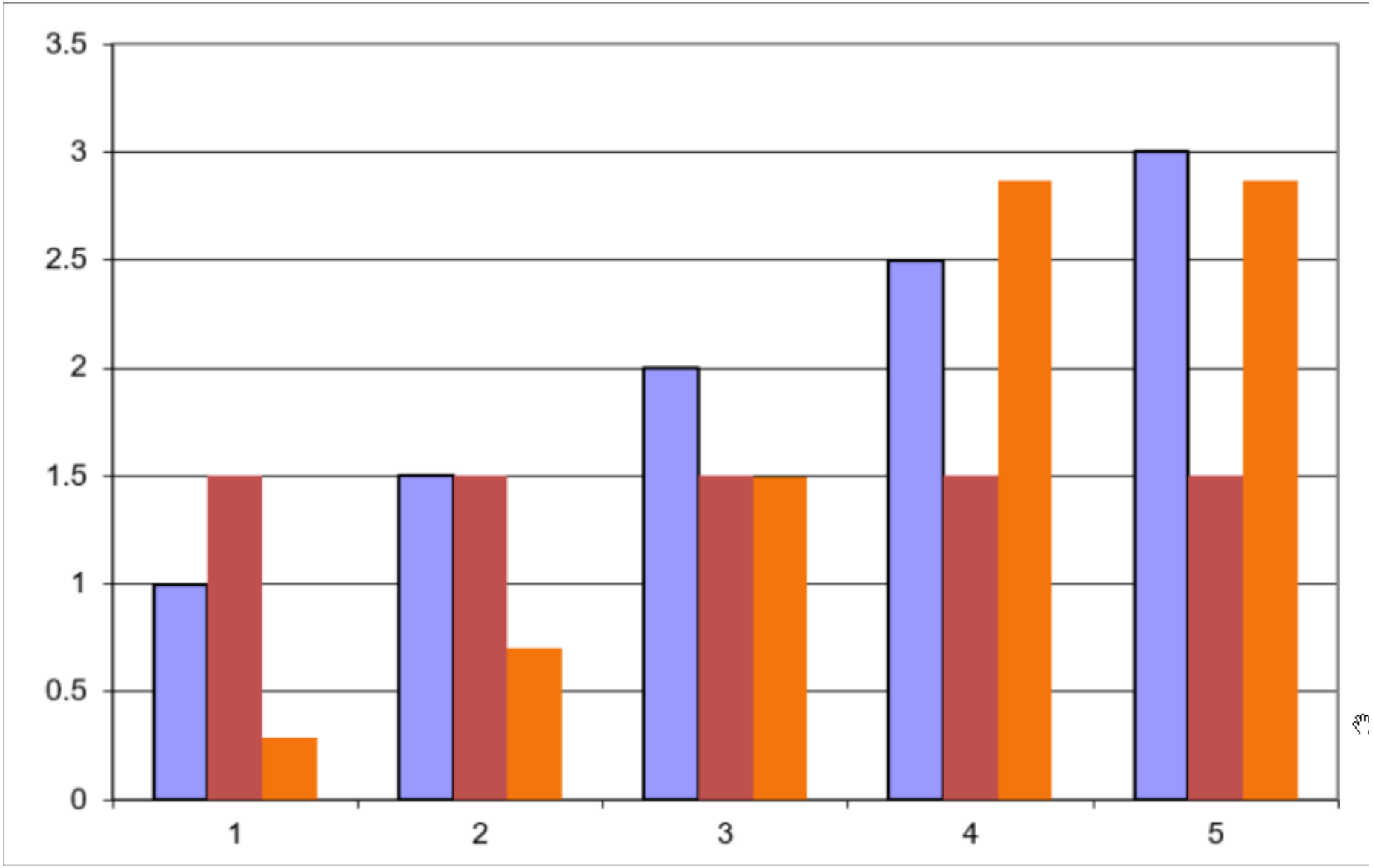


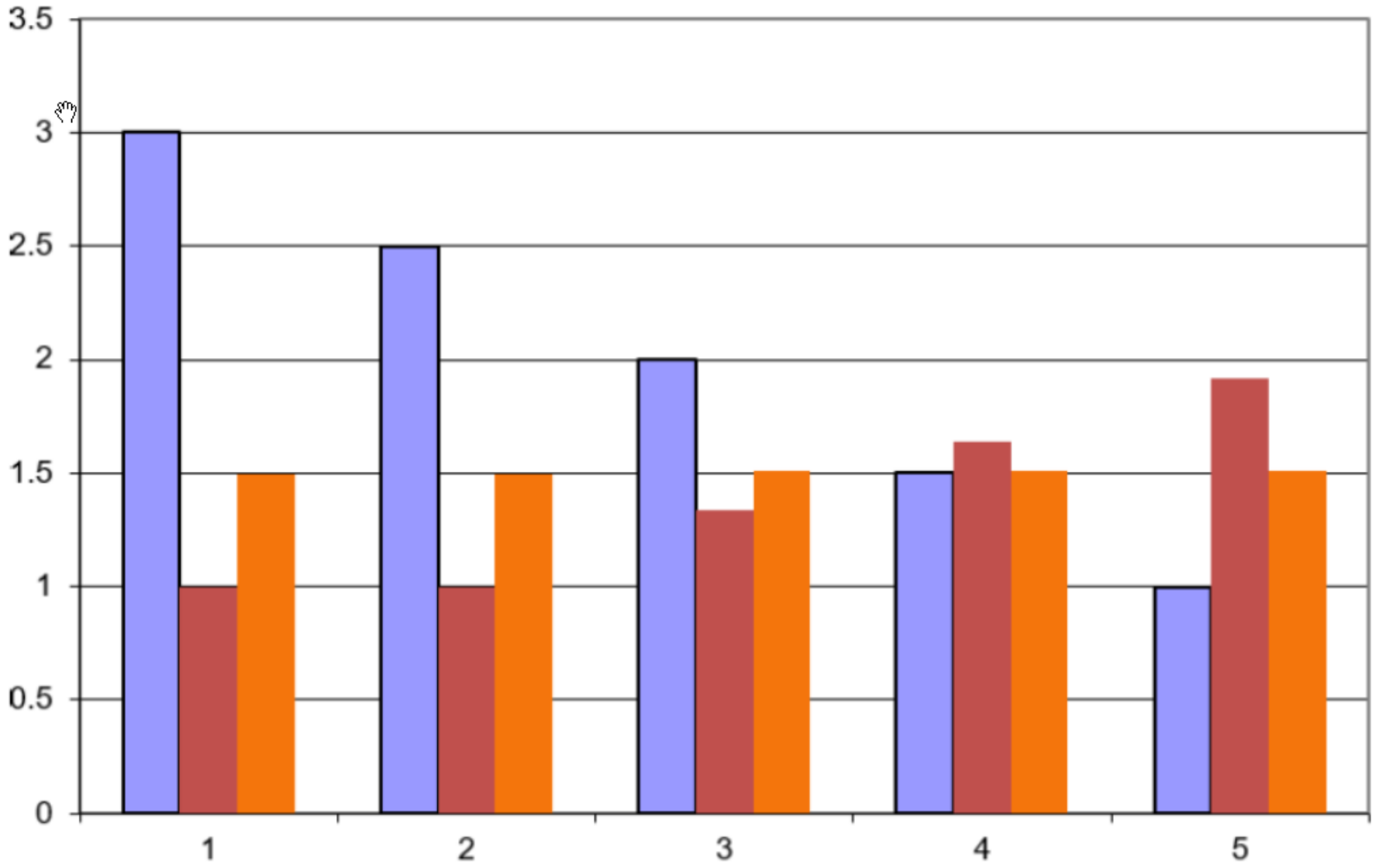
KG calculations

Sigma^n		Decision	&	μ^n	&	β^n	&	β^{n+1}	&	$\sigmatilde$	&	$max_x' x'$	&	ζ	&	$f(z)$	&	$\nu^{KG)_x$
5.00		1	&	3.0	&	0.0400	&	1.0400	&	4.9029	&	5	&	-0.4079	&	0.2277	&	1.1165
8.00		2	&	4.0	&	0.0156	&	1.0156	&	7.9382	&	5	&	-0.1260	&	0.3391	&	2.6920
8.00		3	&	5.0	&	0.0156	&	1.0156	&	7.9382	&	4.5	&	-0.0630	&	0.3682	&	2.9232
9.00		4	&	4.5	&	0.0123	&	1.0123	&	8.9450	&	5	&	-0.0559	&	0.3716	&	3.3241
10.00		5	&	3.5	&	0.0100	&	1.0100	&	9.9504	&	5	&	-0.1507	&	0.3281	&	3.2646

» To link to spreadsheet, click here.







Knowledge gradient

The S-curve effect

4.4.1 The S-curve

What if we perform n_x observations of alternative x, rather than just a single experiment? In this section, we derive the value of n_x experiments to study the marginal value of information. Note that this can be viewed as finding the value of a single experiment with precision $n_x\beta^W$, so below we view this as a single, more accurate experiment.

As before, let $\bar{\mu}_x^0$ and β_x^0 be the mean and precision of our prior distribution of belief about μ_x. Now let $\bar{\mu}_x^1$ and β_x^1 be the updated mean and precision after measuring alternative x a total of n_x times in a row. As before, we let $\beta^W = 1/\sigma_W^2$ be the precision of a single experiment. This means that our updated precision after n_x observations of x is

$$\beta_x^1 = \beta_x^0 + n_x\beta^W.$$

In Section 2.3.1, we showed that

$$\tilde{\sigma}^{2,n} = Var^n[\bar{\mu}^{n+1} - \bar{\mu}^n],$$

where Var^n is the conditional variance given what we know after n iterations. We are interested in the total variance reduction over n experiments. We denote this by $\tilde{\sigma}^{2,0}$, and calculate

$$\begin{aligned}\tilde{\sigma}^{2,0}(n_x) &= \bar{\sigma}^{2,0} - \bar{\sigma}^{2,1} \\ &= (\beta^0)^{-1} - (\beta^0 + n_x\beta^W)^{-1}.\end{aligned}$$

We next take advantage of the same steps we used to create equation (2.14) and write

$$\bar{\mu}_x^1 = \bar{\mu}_x^0 + \tilde{\sigma}_x^0(n_x)Z$$

where Z is a standard normal random variable, and where $\tilde{\sigma}_x^0(n_x) = \sqrt{\tilde{\sigma}_x^{2,0}(n_x)}$ is the standard deviation of the conditional change in the variance of $\bar{\mu}^1$ given that we make n_x observations.

We are now ready to calculate the value of our n_x experiments. Assume we are measuring a single alternative x, so $n_x > 0$ and $n_{x'} = 0$ for $x' \neq x$. Then we can write

$$\nu_x^{KG}(n_x) = \mathbb{E}\left[\max_{x'}(\bar{\mu}_{x'}^0 + \tilde{\sigma}_{x'}^0(n_{x'})Z_{x'})\right] - \max_{x'}\bar{\mu}_{x'}^0.$$

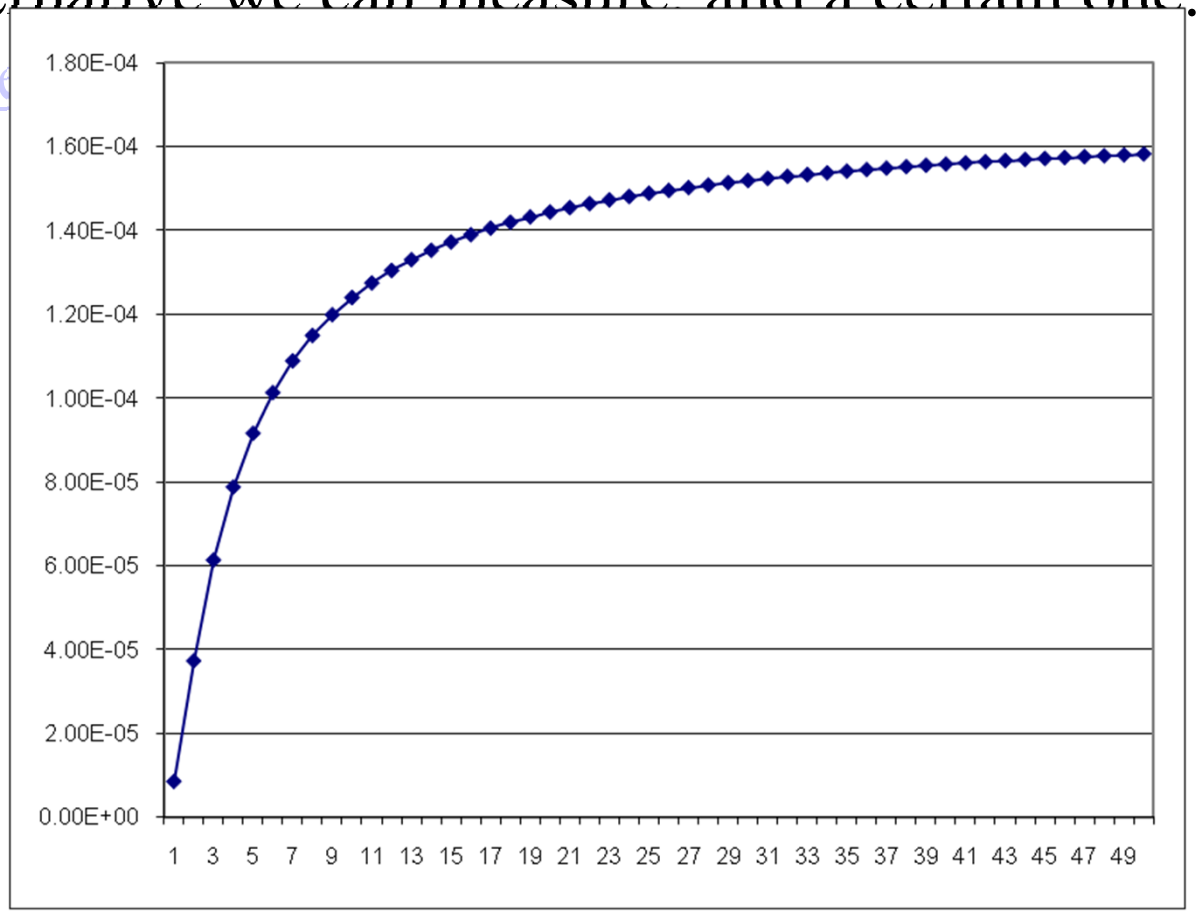
We can compute the value of n_x observations of alternative x using the knowledge gradient formula in equation (4.12),

$$\nu_x^{KG}(n_x) = \tilde{\sigma}_x^0(n_x) f\left(\frac{\bar{\mu}_x^0 - \max_{x' \neq x}\bar{\mu}_{x'}^0}{\tilde{\sigma}_x^0(n_x)}\right),$$

where $f(\zeta)$ is given in equation (4.11).

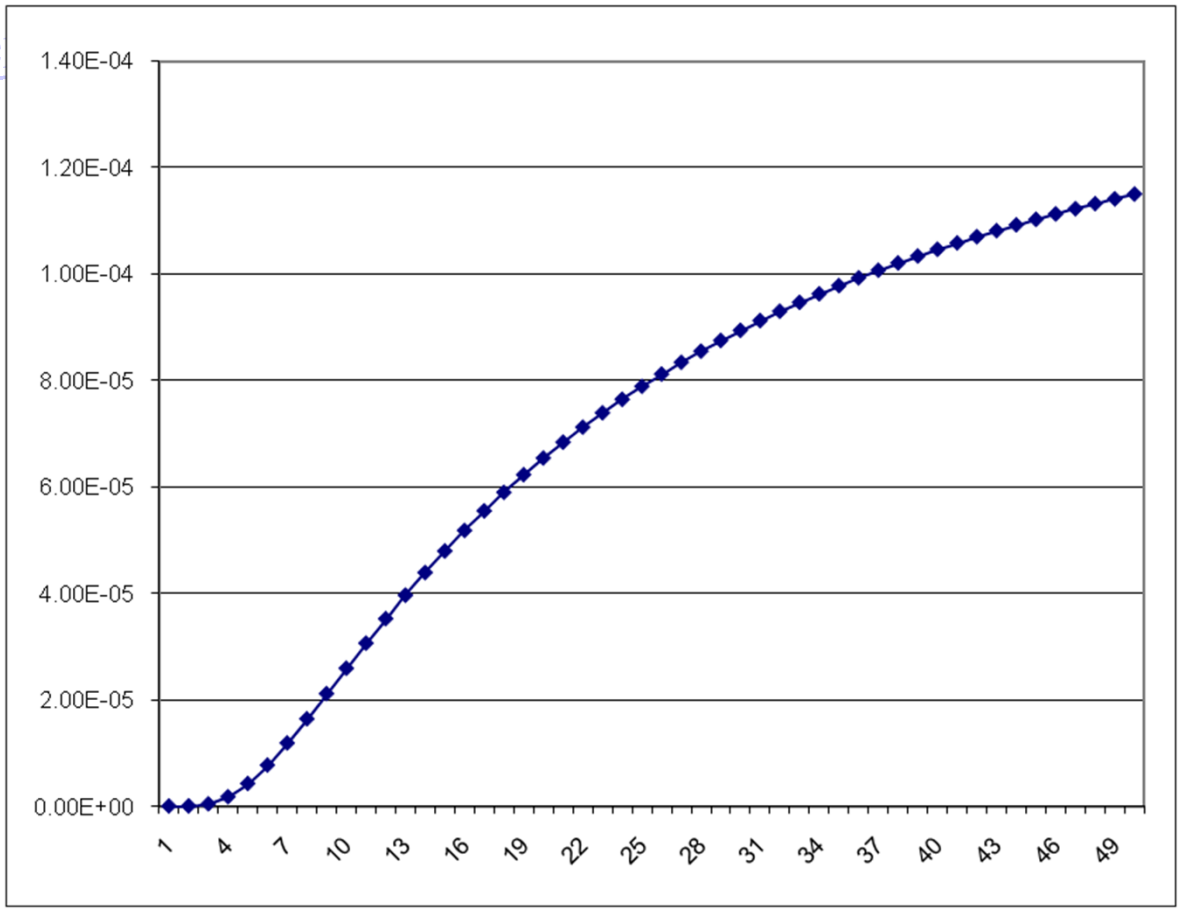
S-curve

- The marginal value of information
  - » Imagine that we are choosing between an uncertain alternative we can measure, and a certain one. Click here



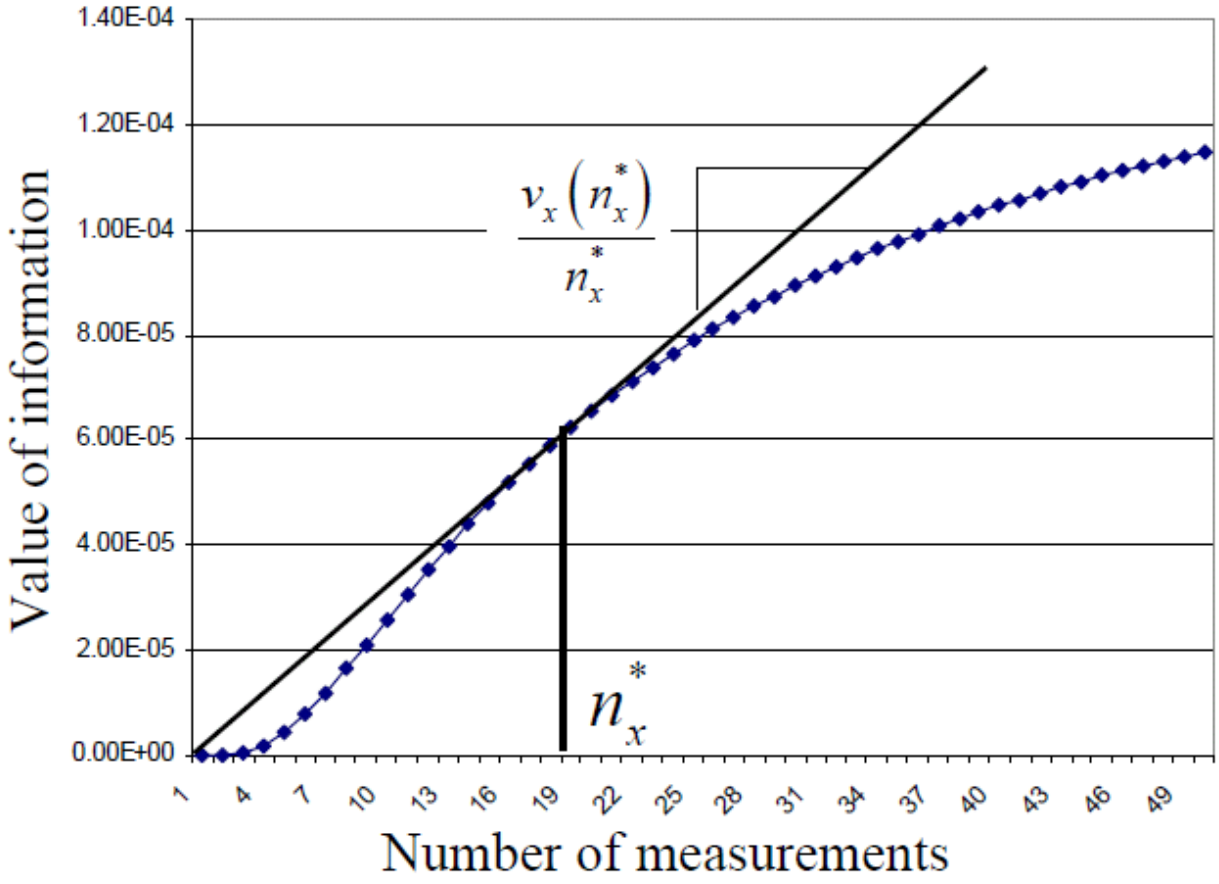
S-curve

- The marginal value of information
  - » Imagine that we are choosing between an uncertain alternative we can measure, and a certain one. Click he



S-curve

- The KG(*) policy
  - » Assume that we are going to make enough measurements to maximize the average value of an



$$n_x^* = \arg\max_{n_x>0} \frac{v_x(n_x)}{n_x}.$$

S-curve

- The KG(*) policy
  - » The KG(*) policy is just like the KG policy with one twist…
  - » Instead of updating the precision with
    $$\beta^{n+1} = \beta^n + \beta^W$$
  - » … we use instead
    $$\beta^{n+1} = \beta^n + n^*\beta^W$$
  - » What we are doing is pretending that the experiment is more precise than it is.
  - » As an alternative, we can introduce a tunable parameter τ:
    $$\beta^{n+1}(\tau) = \beta^n + \tau\beta^W$$
  - » Now let $\nu_x^{KG,n}(\tau)$ be the knowledge gradient computed using precision $\tau\beta^W$ instead of β^W. Now we have a one-step lookahead policy, but with a parameter τ that now has to be tuned just as we did with the earlier heuristic policies.
  - » When the value of information is not concave, this can be quite valuable.

4.4.3 A tunable lookahead policy

The KG(*) policy implicitly assumes that our experimental budget is large enough to sample alternative x roughly n_x^* times. There are many applications where this is just not going to be true. We can mimic the basic idea of KG(*), but replace n_x^* with a tunable parameter. Begin by defining

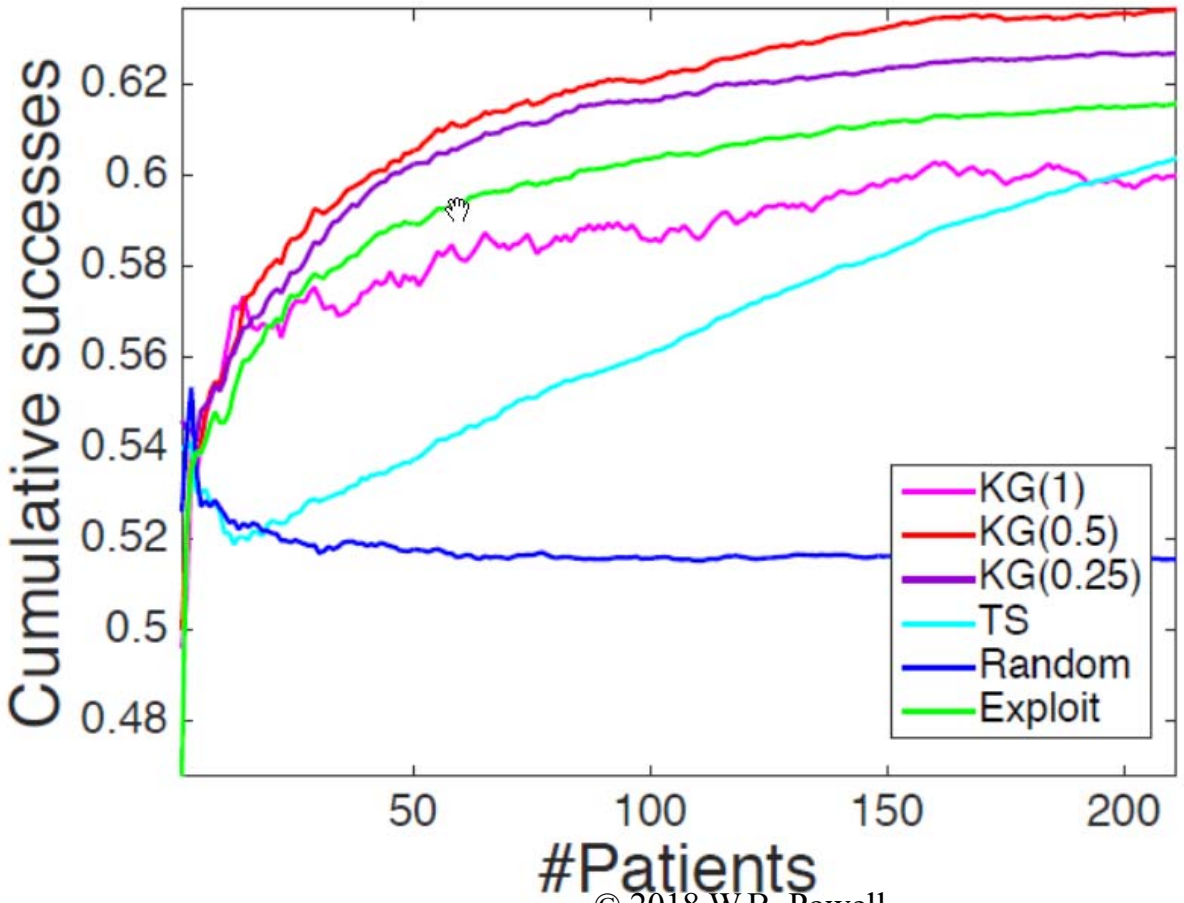
$\nu^{KG,n}(\theta^{KG})$ = The knowledge gradient computed using a precision $\beta^1 = \beta^0 + \theta^{KG}\beta^W$.

We then define our tunable KG policy as

$$X^{KG}(\theta^{KG}) = \arg\max_x \nu_x^{KG,n}(\theta^{KG})$$

We now have to tune our policy to find the best value of θ^{KG} (see section 3.4 for our discussion on tuning). Here is where our policy adapts to our learning budget, as well as other problem characteristics that we choose to model.

- Comparison of policies on health application
  - » Patients arrive at random with a set of characteristics *a*
  - » Choice *x* is a medical decision
  - » otherwise.



Week 3 - Monday

KG for online learning

Bernoulli belief model

Stochastic shortest paths - I

Knowledge gradient

Online learning/cumulative reward

Knowledge gradient

5.3.1 The basic idea

Once again, consider the normal-normal Bayesian learning model. Suppose that we can run N experiments, and that $\gamma = 1$. Furthermore, suppose that we have already run n experiments, and have constructed estimates $\bar{\mu}_x^n$ and β_x^n for each alternative x. Now, as a thought experiment, let us imagine that we will suddenly cease learning, starting at time n. We will still continue to collect rewards, but we will no longer be able to use the updating equations (3.2) and (3.3) to change our beliefs. We are stuck with our time-n beliefs until the end of the time horizon.

If this were to occur, the best course of action would be to choose $x^{n'} = \arg\max_x \bar{\mu}_x^n$ for all times $n \leq n' \leq N$. Since we cannot change our beliefs anymore, all we can really do is choose the alternative that seems to be the best, based on the information that we managed to collect up to this point. The expected total reward that we will collect by doing this, from time n to time N, is given by

$$V^{Stop,n}(S^n) = (N - n + 1) \max_x \bar{\mu}_x^n, \tag{5.16}$$

simply because there are $N - n + 1$ rewards left to collect. Because $\gamma = 1$, each reward is weighted equally. For instance, in the example given in Table 5.4, this quantity is $V^{Stop,n}(S^n) = 6 \cdot 5.5 = 33$.

Knowledge gradient

Consider a different thought experiment. We are still at time n, but now our next decision will change our beliefs as usual. However, starting at time $n+1$, we will cease to learn, and from there on we will be in the situation described above. This means that, starting at time $n+1$, we will always measure the alternative given by $\arg\max_x \bar{\mu}_x^{n+1}$. The problem thus reduces to choosing one single decision x^n to maximize the expected total reward we collect, starting at time n.

This idea is essentially the knowledge gradient concept from a slightly different point of view. In ranking and selection, we chose each decision to maximize the incremental improvement (obtained from a single experiment) in our estimate of the best value. Essentially, we treated each decision as if it were the last time we were allowed to learn. We made each decision in such a way as to get the most benefit out of that single experiment. In the online setting, we do the same thing, only "benefit" is now expressed in terms of the total reward that we can collect from time n to the end of the time horizon.

The KG decision for the bandit problem is given by

$$X^{KG,n} = \arg\max_x \mathbb{E}\left[\mu_x + V^{Stop,n+1}\left(S^{n+1}\right) \mid S^n, x^n = x\right] \tag{5.17}$$

$$= \arg\max_x \bar{\mu}_x^n + (N-n)\,\mathbb{E}\left[\max_{x'} \bar{\mu}_{x'}^{n+1} \mid S^n, x^n = x\right] \tag{5.18}$$

$$= \arg\max_x \bar{\mu}_x^n + (N-n)\,\mathbb{E}\left[\max_{x'} \bar{\mu}_{x'}^{n+1} - \max_{x'} \bar{\mu}_{x'}^n \mid S^n, x^n = x\right] \tag{5.19}$$

$$= \arg\max_x \bar{\mu}_x^n + (N-n)\,\nu_x^{KG,n}, \tag{5.20}$$

where $\nu_x^{KG,n}$ is simply the knowledge gradient for ranking and selection, given by (4.12).

Knowledge gradient

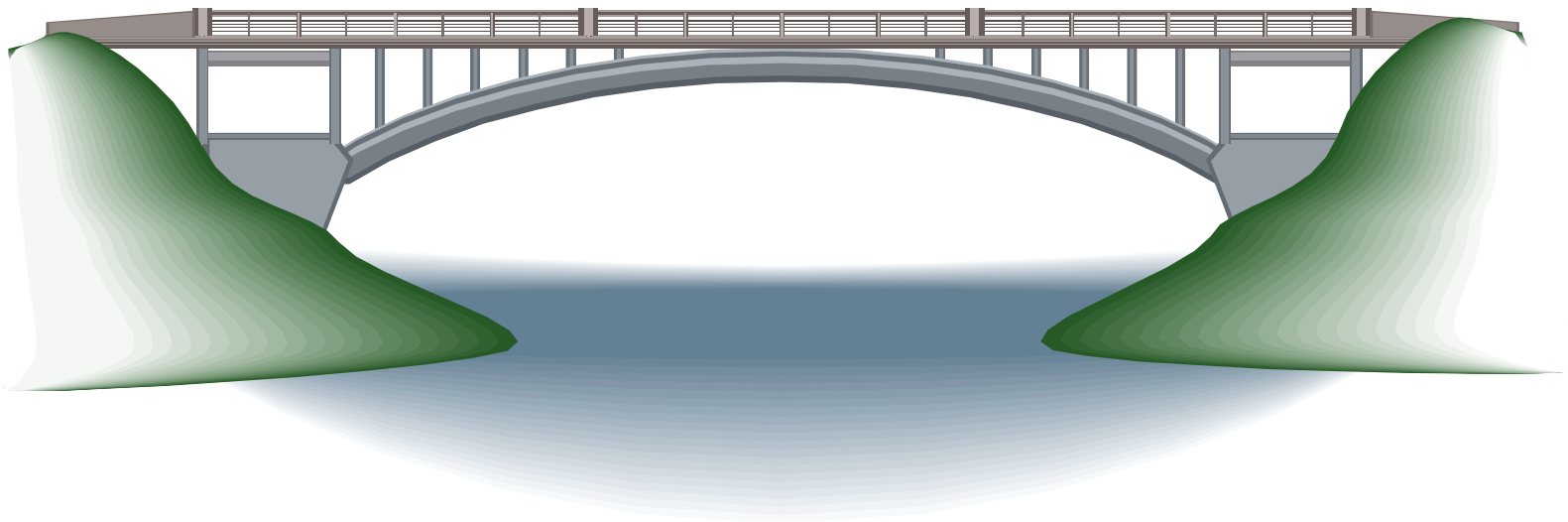
- Notes:
  - » Asymptotically finds optimal arm as $\gamma \to 1$
  - » Knowledge gradient seems to be the only policy with distinct forms for offline (final reward) and online (cumulative reward):

Offline learning

$$\nu_x^{KG,n}$$

Online learning

$$\nu_x^{KG-OL,n} = \bar{\mu}_x^n + (N-n)\nu_x^{KG,n}$$



Knowledge gradient

- Discounted finite horizon

$$X^{KG,n} = \arg\max_x \bar{\mu}_x^n + \gamma \frac{1-\gamma^{N-n}}{1-\gamma} \nu_x^{KG,n}.$$

- Discounted infinite horizon

$$X^{KG,n} = \arg\max_x \bar{\mu}_x^n + \frac{\gamma}{1-\gamma} \nu_x^{KG,n}.$$

- Tunable versions:

$$\beta_x^{n+1} = \beta_x^n + \theta_1 \beta^W.$$

We let $\nu_x^{KG,n}(\theta_1)$ be the offline KG formula using θ_1 as the repeat factor for the precision β^W. Then, we introduce a second parameter θ_2 which replaces the horizon $N-n$ in our online KG formula. Our tunable online KG formula is now given by

$$X^{OLKG}(\theta) = \arg\max_x \left(\bar{\mu}_x^n + \theta_2 \nu_x^{KG,n}(\theta_1)\right),$$

where $\theta = (\theta_1, \theta_2)$ captures our tunable parameters.

Knowledge gradient

- Knowledge gradient policy
  - » For finite-horizon on-line problems:

$$\nu_x^{KG-OL,n} = \bar{\mu}_x^n + (N-n)\nu_x^{KG,n}$$

  - » For infinite-horizon discounted problems:

$$\nu_x^{KG-OL,n} = \bar{\mu}_x^n + \frac{\gamma}{1-\gamma}\nu_x^{KG,n}$$

- Compare to Gittins indices for bandit problems

$$\nu_x^{Gittins} = \bar{\mu}_x^n + \Gamma(n,\gamma)\sigma_W$$

- …interval estimation

$$\nu_x^{IE,n} = \bar{\mu}_x^n + z_\alpha \ \bar{\sigma}_x^n$$

- … and UCB

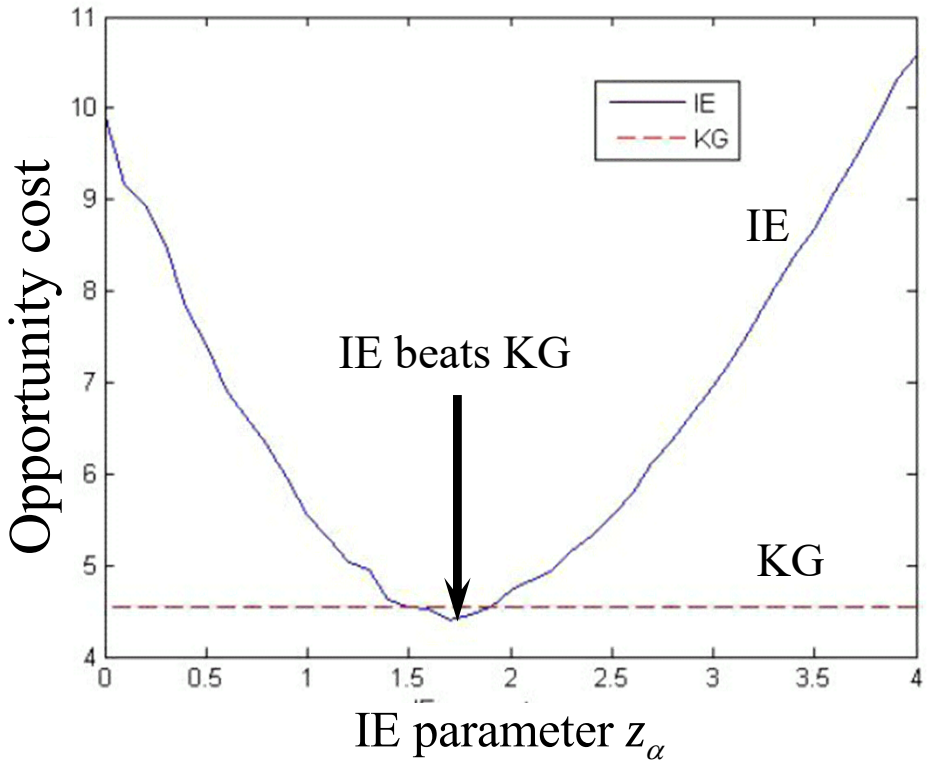
$$\nu_x^{UCB1} = \bar{\mu}_x^n + 4\sigma^\varepsilon \sqrt{\frac{\log n}{N_x^n}}$$

Knowledge gradient

- KG versus interval estimation
  - » Recall that with IE, you choose the alternative with the highest:

$$\nu_x^{IE,n} = \bar{\mu}_x^n + \theta^{IE}\,\bar{\sigma}_x^n$$

Tunable parameter

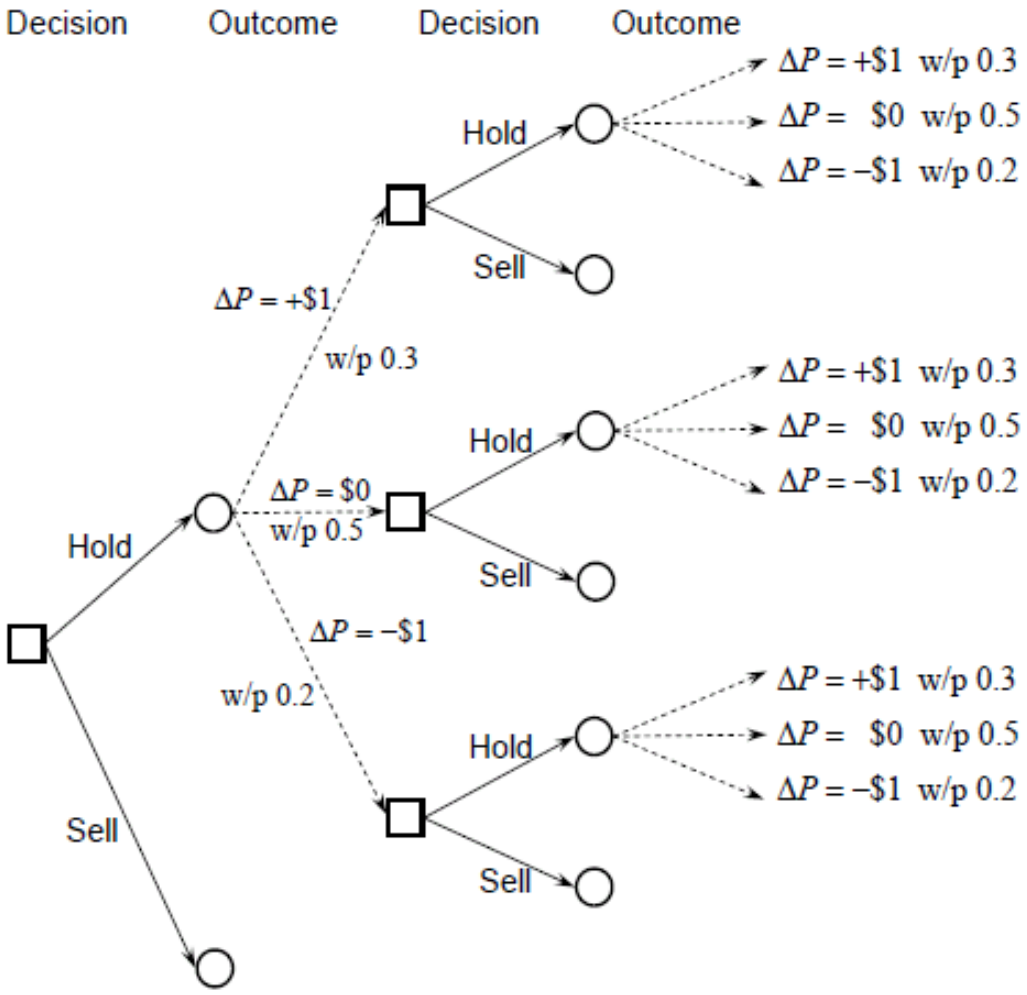


Lookahead learning model

Beta-Bernoulli belief

Learning using decision trees

- Decision tree without learning

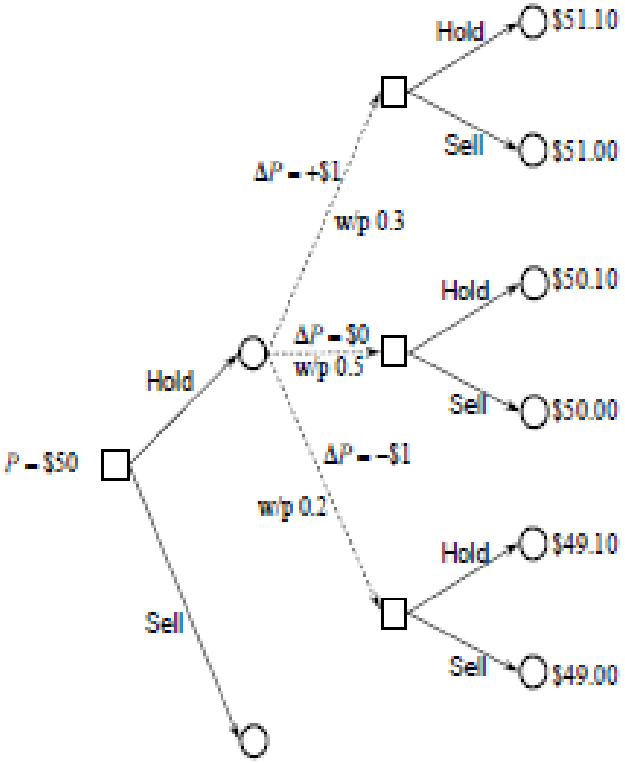


Learning using decision trees

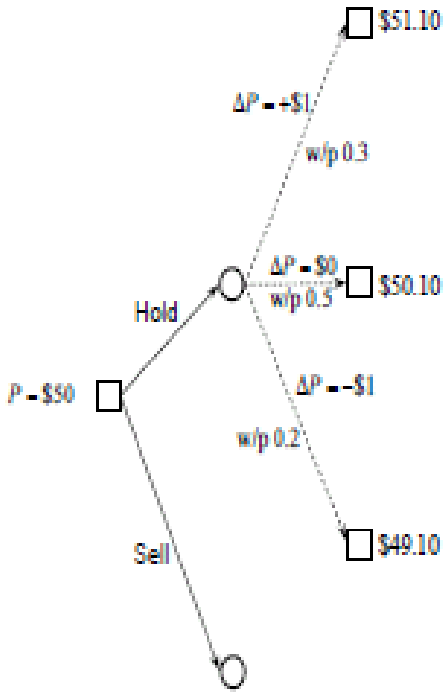
- Some vocabulary:
  - » Square nodes are:
    - Decision nodes – Links emanating from the squares are decisions
    - Also represents the state of our system (it implicitly captures all the information we need to make a decision).
    - Also known as the "pre-decision state."
  - » Circle nodes are:
    - Outcome nodes – Links emanating from circles represent random events.
    - Represents the state after a decision is made.
    - Also known as the "post-decision state."
  - » Nodes in a decision tree always imply the information in the entire path leading up to the node.
  - » We receive a reward whenever we sell.

Learning using decision trees

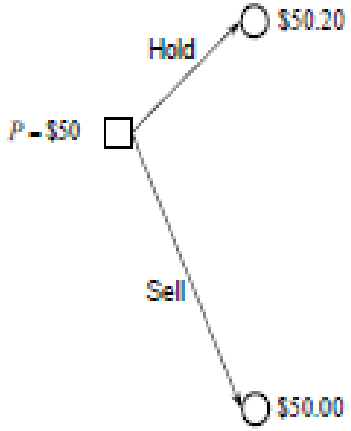
- Rolling back the tree



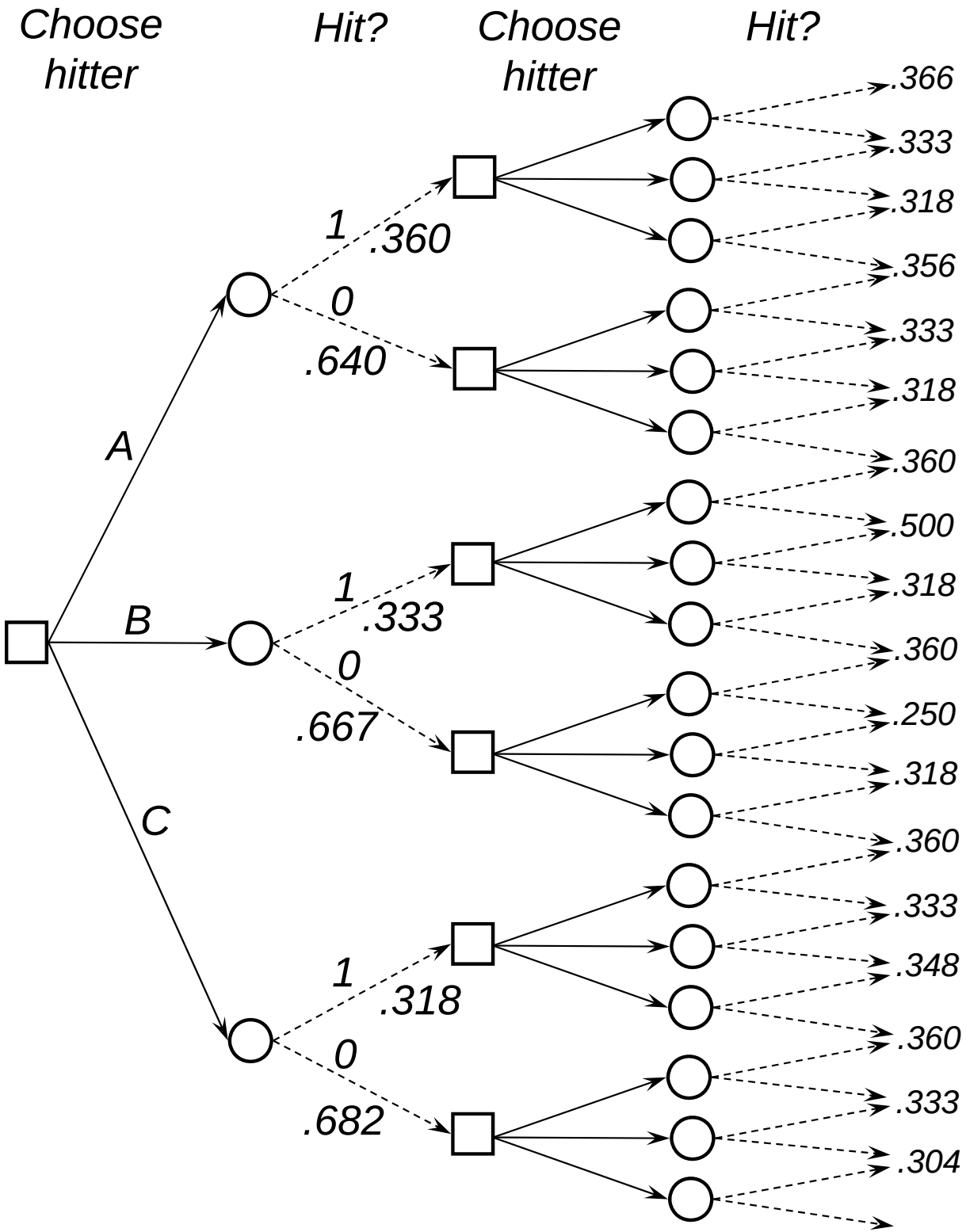
1.7(a)



1.7(b)

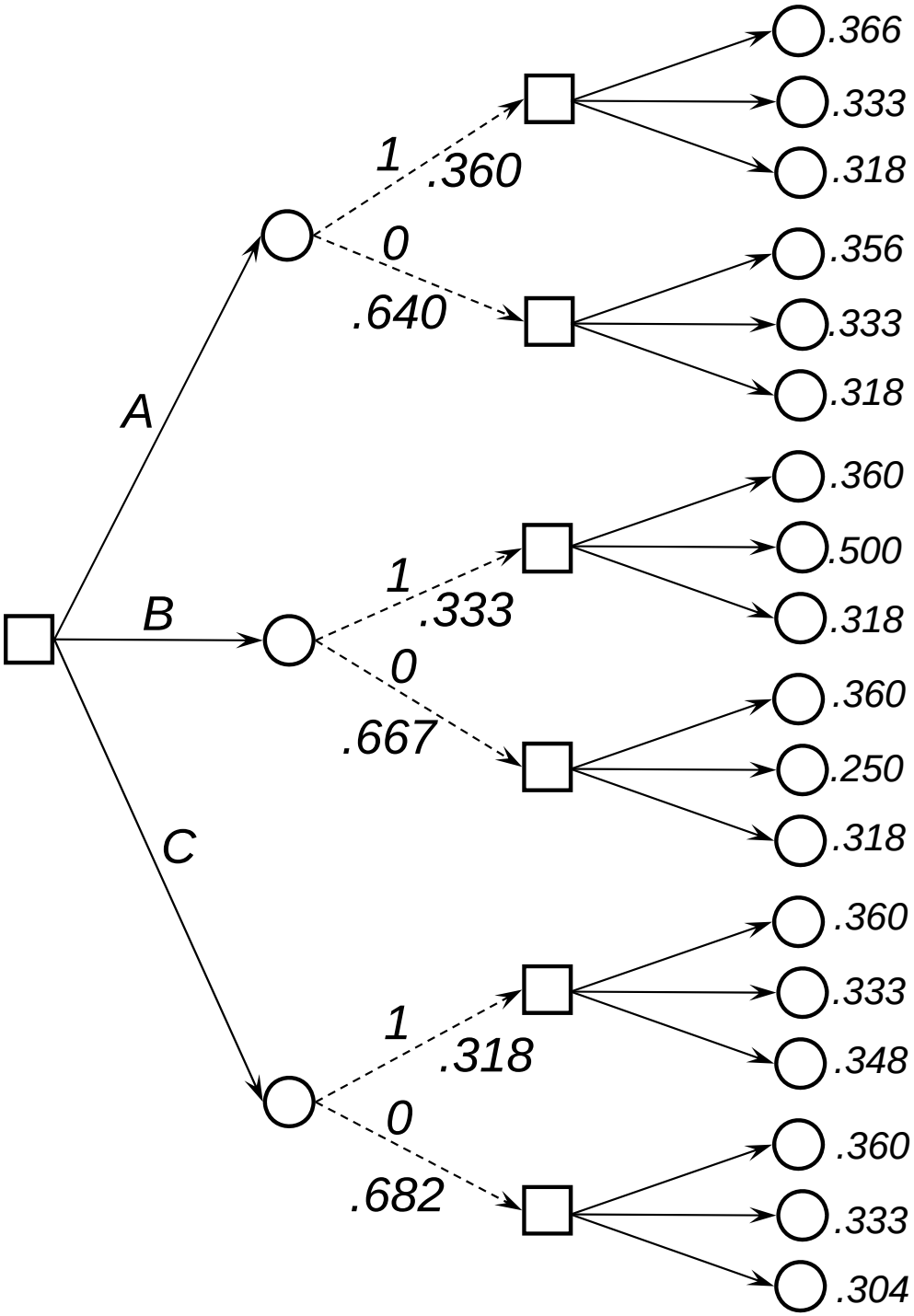


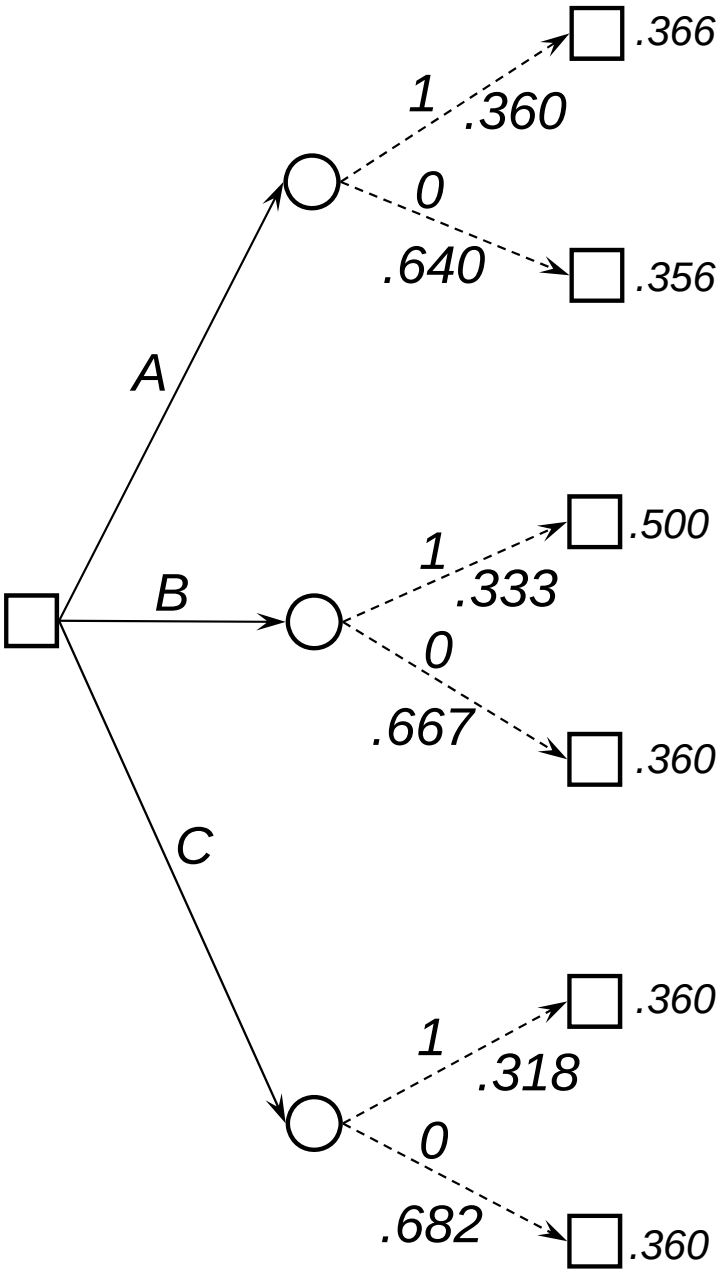
1.7(c)

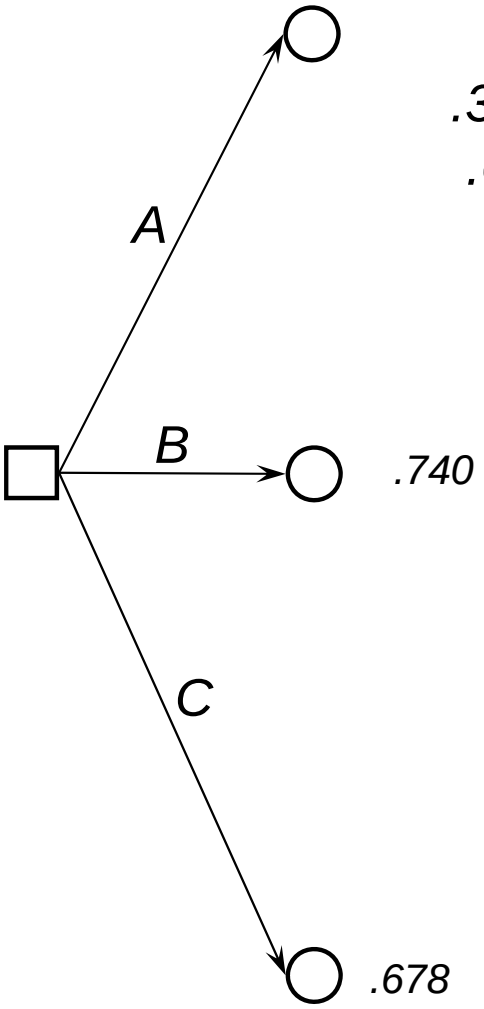


Hidden in each node is the updated belief state after new observations are made.

We could assume that we do not learn, but here we are learning, and new observations produce updated estimates of probabilities.


A
B
C
1
.360
0
.640
1
.333
0
.667
1
.318
0
.682
.366
.333
.318
.356
.333
.318
.360
.500
.318
.360
.250
.318
.360
.333
.348
.360
.333
.304


.366
1
.360
0
.640
.356
A
.500
1
.333
B
0
.667
.360
C
.360
1
.318
0
.682
.360



- Notes:
  - » We choose player B, who does not have the highest average.
  - » Discuss the impact of the uncertainty about player B.
  - » What if all the players had 100's of at bats?

- 0/1 outcomes
  - » Does an internet customer click on a link or buy a product.
  - » Did the baseball player get a hit?
  - » Did the drug work for the patient?
  - » Did the student accept our offer of admission?

Multi-step lookahead policies

- Learning with a beta-Bernoulli model
  - » Each experiment has two outcomes:

$$W_x^{n+1} = \begin{cases} 1 & \text{If success} \\ 0 & \text{Otherwise} \end{cases}$$

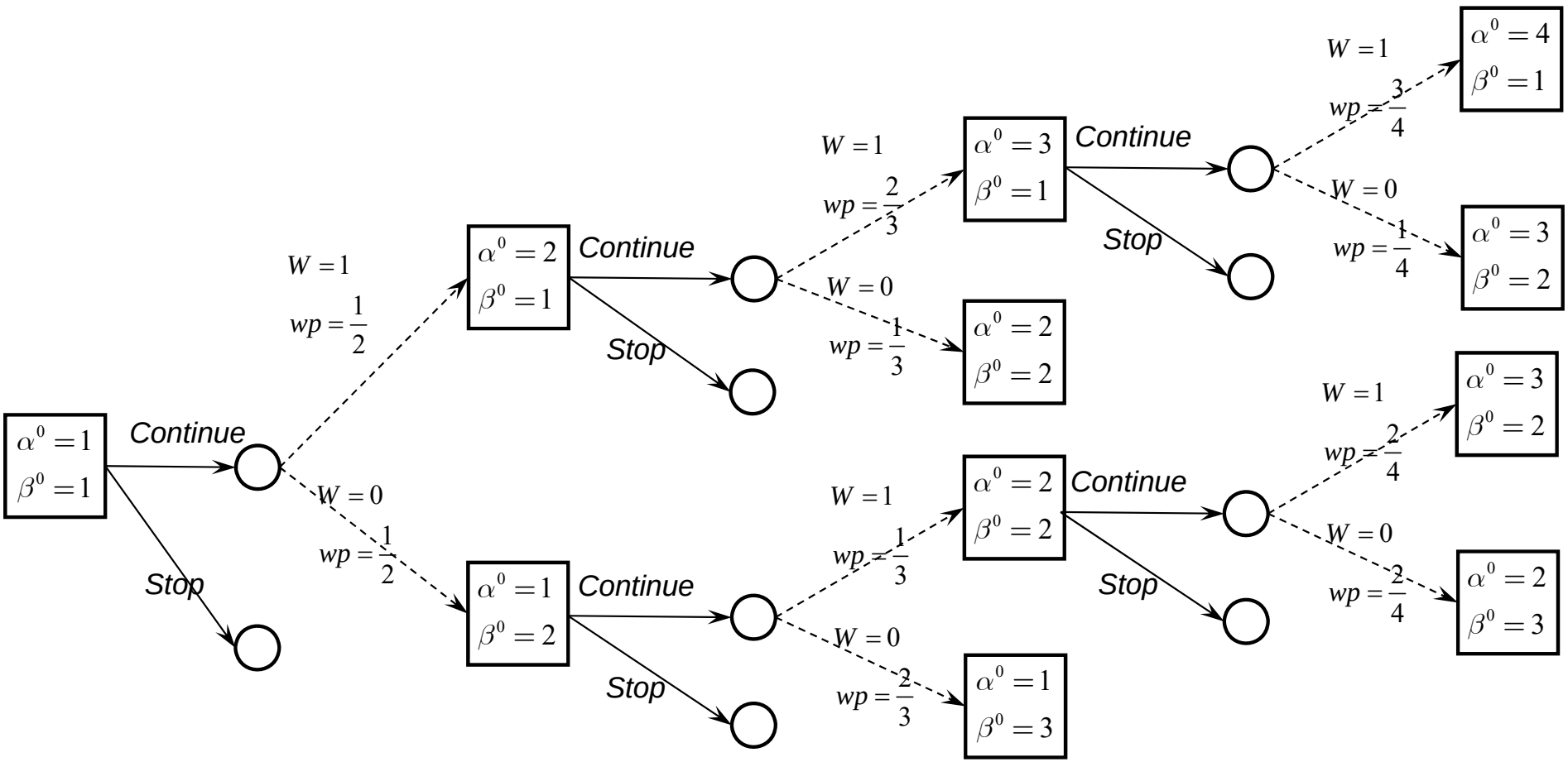
  - » Beta distribution of belief on success probability ρ_x:

$$f(\rho \mid \alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \rho^{\alpha-1}(1-\rho)^{\beta-1} \qquad \Gamma(x) = x!$$

  - » Updating equations

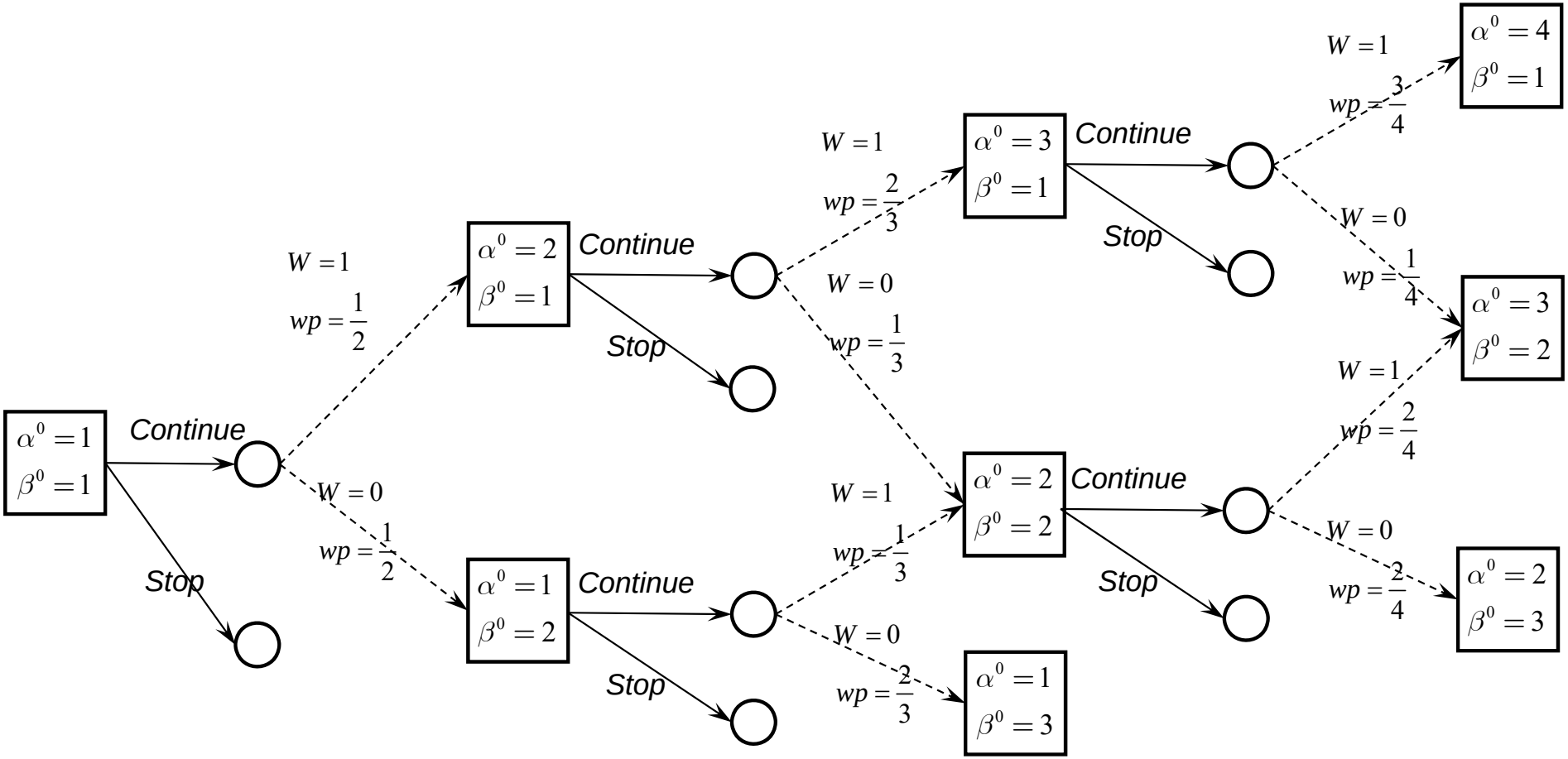
$$\alpha_x^{n+1} = \begin{cases} \alpha_x^n + W_x^{n+1} & \text{if } x^n = x \\ \alpha_x^n & \text{otherwise,} \end{cases}$$

$$\beta_x^{n+1} = \begin{cases} \beta_x^n + \left(1 - W_x^{n+1}\right) & \text{if } x^n = x \\ \beta_x^n & \text{otherwise.} \end{cases}$$



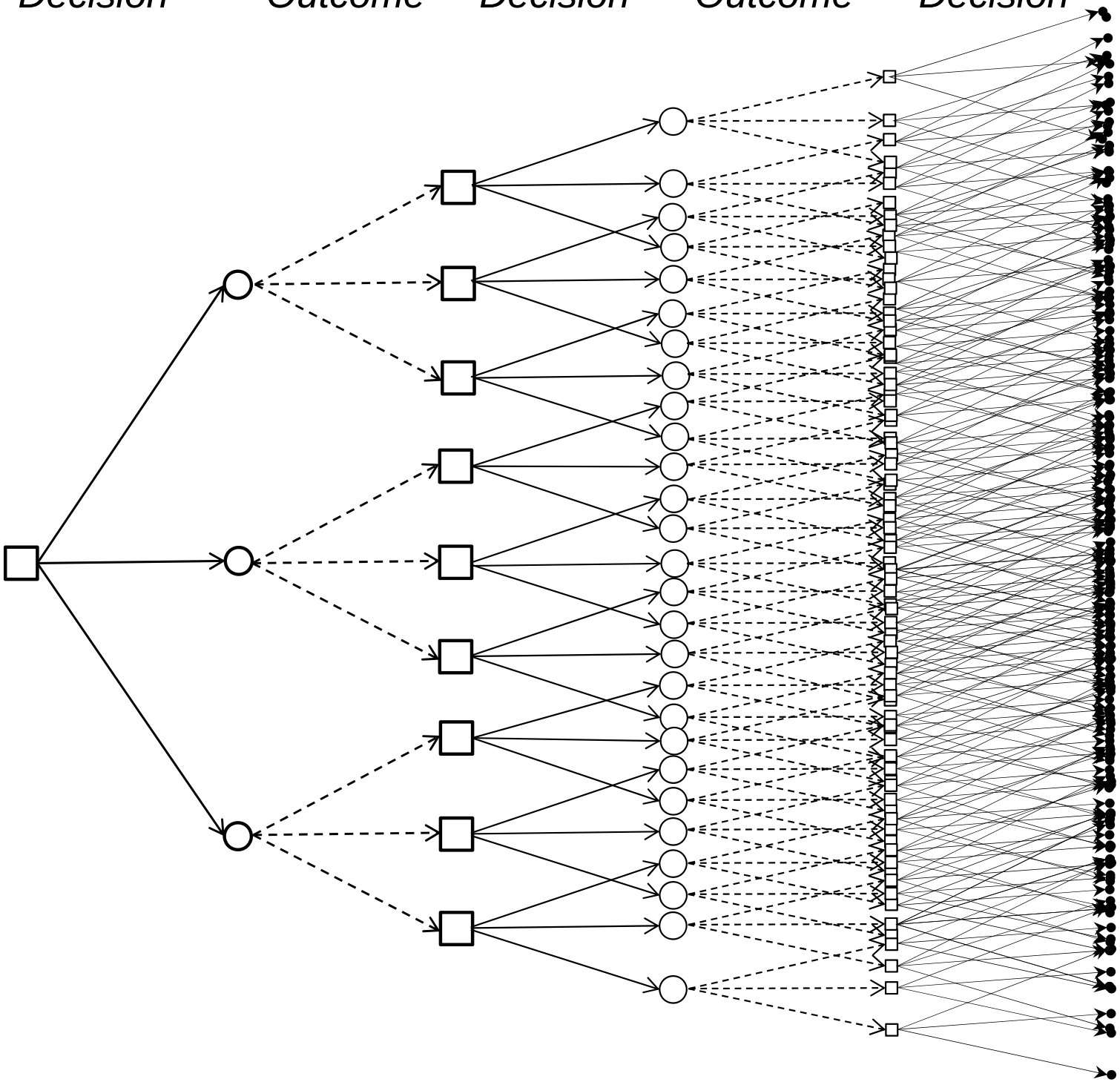
As a decision tree (some states are duplicated)

Decision *Outcome* *Decision* *Outcome*



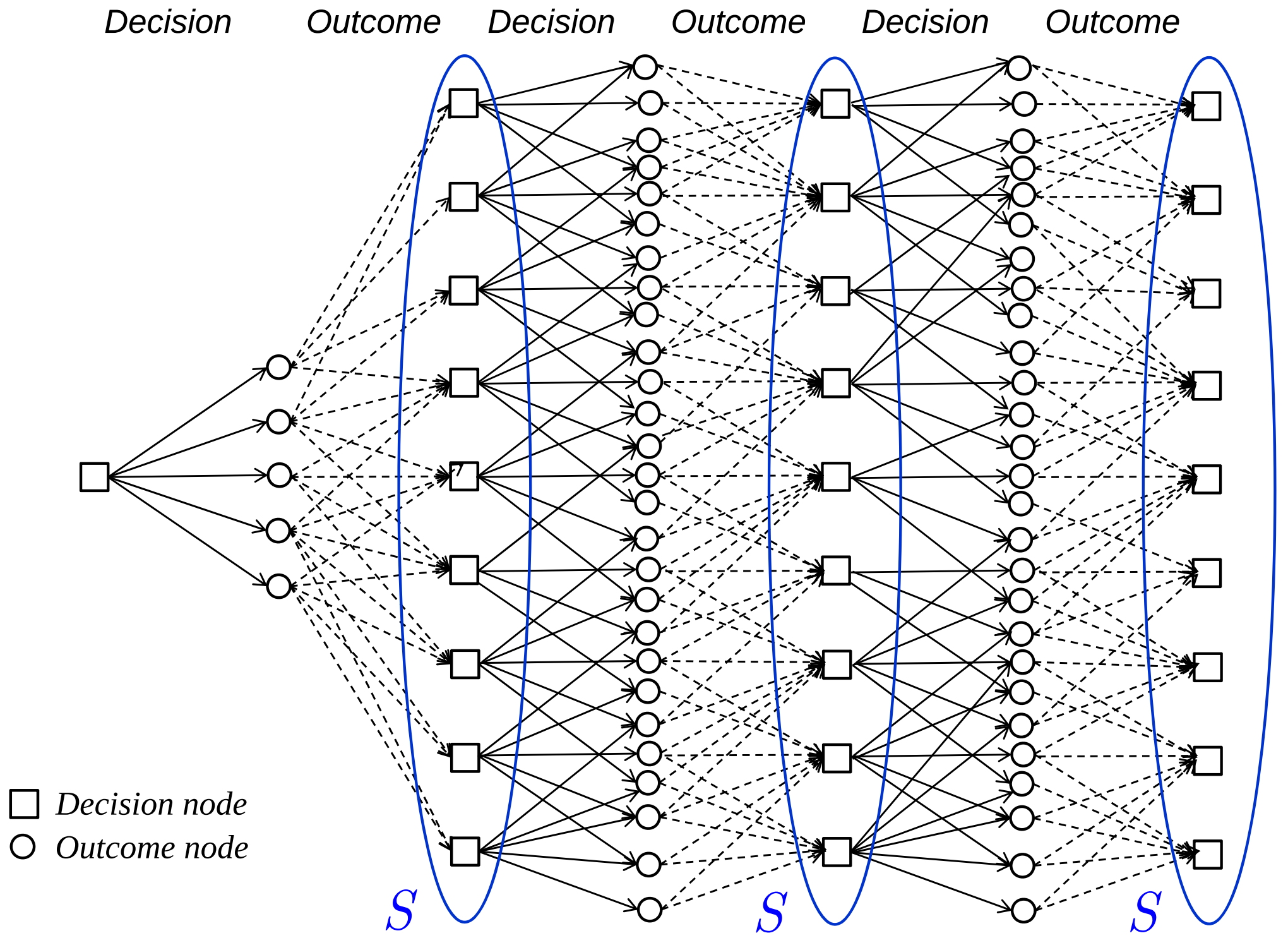
$$V(S_t) = \min_{x_t \in \mathcal{X}} \left(C(S_t, x_t) + \gamma \mathbb{E}\left\{ V(S_{t+1}(S_t, x_t, W_{t+1}) \mid S_t \right\} \right)$$

As a dynamic program (duplicate states removed)



Discrete MDPs

- Decision trees:
  - » Discuss the concept of a "state" in a decision tree.
  - » Implicit in a decision node is the entire history leading up to the decision, which is unique.
  - » We may not need the entire history, but we do not need to articulate this.
  - » The complexity of a decision tree is independent of the complexity of the state variable. Instead, it depends on the number of possible decisions (actions) and outcomes.


Decision
Outcome
Decision
Outcome
Decision
Outcome
S
S
S
Decision node
Outcome node

Shortest path example

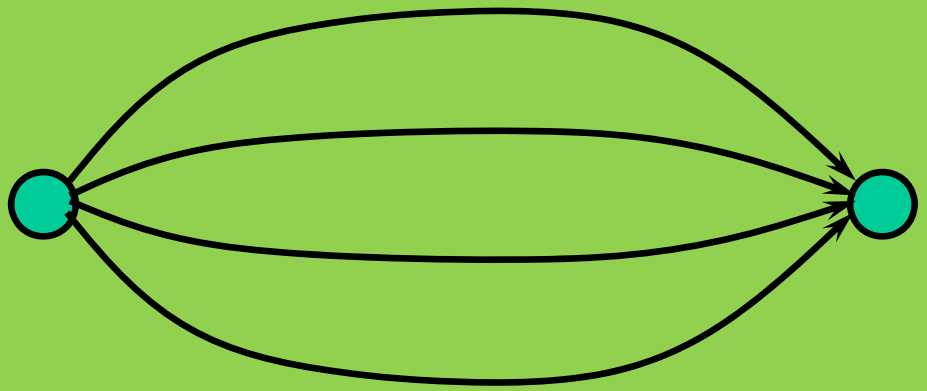
Optional – if time

Finding the best path

- Figure out Manhattan:
  - » Walking
  - » Subway/walking
  - » Taxi
  - » Street bus
  - » Driving
- Paths may be intertwined
  - » Trying one path may inform you about the performance of other paths.

Finding the best path

- Finding the best path to work
  - » Four paths, but everyone time I drive on one, I sample a new time.
  - » I want to choose the path that is best on average.



Information acquisition

- The shortest path game:
  - » Starting with the estimates at the top, choose paths so that you discover the best path.

Path				
1	2	3	4	Chosen path
25.00	24.00	22.00	20.00	1
18.49	24.00	22.00	20.00	1
12.11	24.00	22.00	20.00	1
23.92	24.00	22.00	20.00	4
23.92	24.00	22.00	27.76	3
23.92	24.00	16.31	27.76	3
23.92	24.00	32.95	27.76	1
28.46	24.00	32.95	27.76	2
28.46	24.09	32.95	27.76	2
28.46	27.85	32.95	27.76	4
28.46	27.85	32.95	24.76	4
28.46	27.85	32.95	32.77	

Information acquisition

- The shortest path game:
  - » Starting with the estimates at the top, choose paths so that you discover the best path.

	Path				
Day	1	2	3	4	Chosen path
1	25.00	24.00	22.00	20.00	
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Information acquisition

- The shortest path game:
  - » Starting with the estimates at the top, choose paths so that you discover the best path.

	Path				
Day	1	2	3	4	Chosen path
1	25.00	24.00	22.00	20.00	
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					

Finding the best path

- What do we know?
  - » The real average path times:
    - » Mean time
    - Path 1 20 minutes
    - Path 2 22 minutes
    - Path 3 24 minutes
    - Path 4 26 minutes

    - Errors are +/- 10 minutes
  - » What we think:
    - Path 1 25 minutes
    - Path 2 24 minutes
    - Path 3 22 minutes
    - Path 4 20 minutes
  - » We act by choosing the path that we "think" is the best. The only way we learn anything new is by choosing a path.

Finding the best path

- Illustration of calculations:

True average travel times

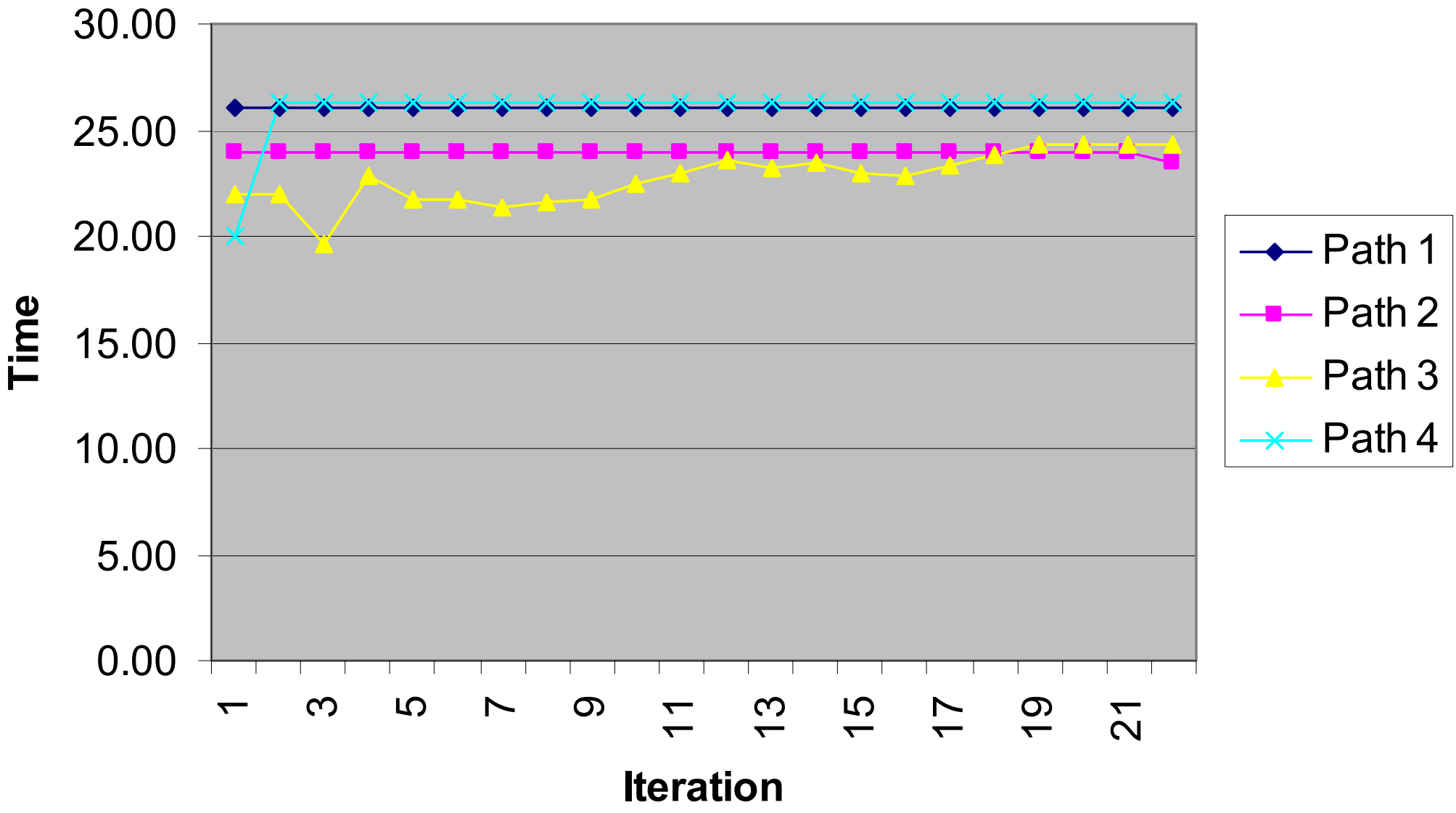
Initial estimates of travel times

Means	20	22	24	26		25	24	22	20
Spread	20					6			
Stepsize	0.1					3			
	Paths					1			
Day	1	2	3	4		Path 1	Path 2	Path 3	Path 4
1	16.91	17.29	14.43	33.46		25.00	24.00	22.00	20.00
2	20.49	12.90	30.19	16.43		25.00	24.00	22.00	26.73
3	20.77	22.57	32.90	17.61		25.00	24.00	24.73	26.73
4	22.50	18.69	25.67	16.72		25.00	23.64	24.73	26.73
5	19.08	16.38	19.15	25.18		25.00	22.65	24.73	26.73

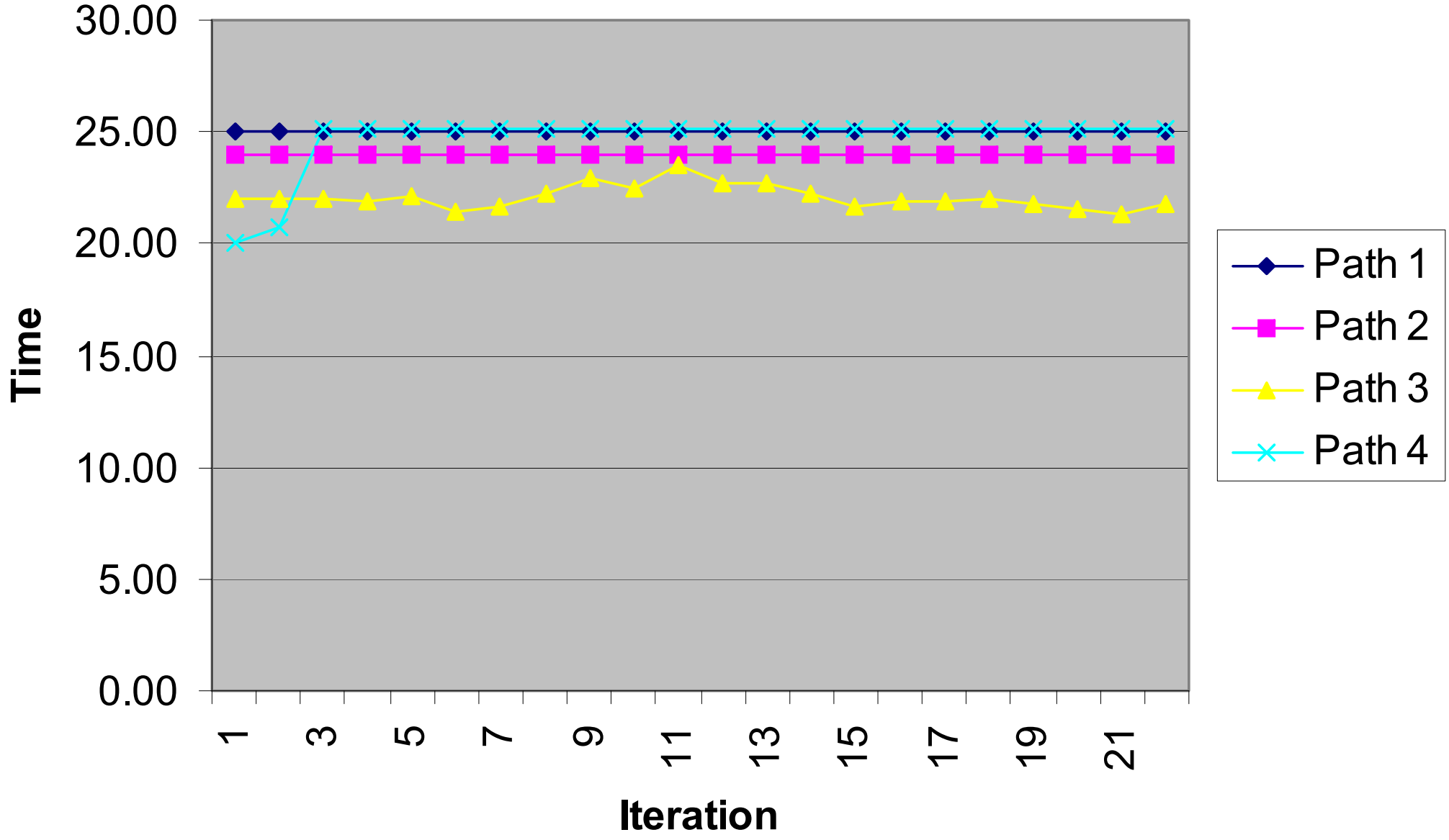
Actual travel times

Estimated travel times

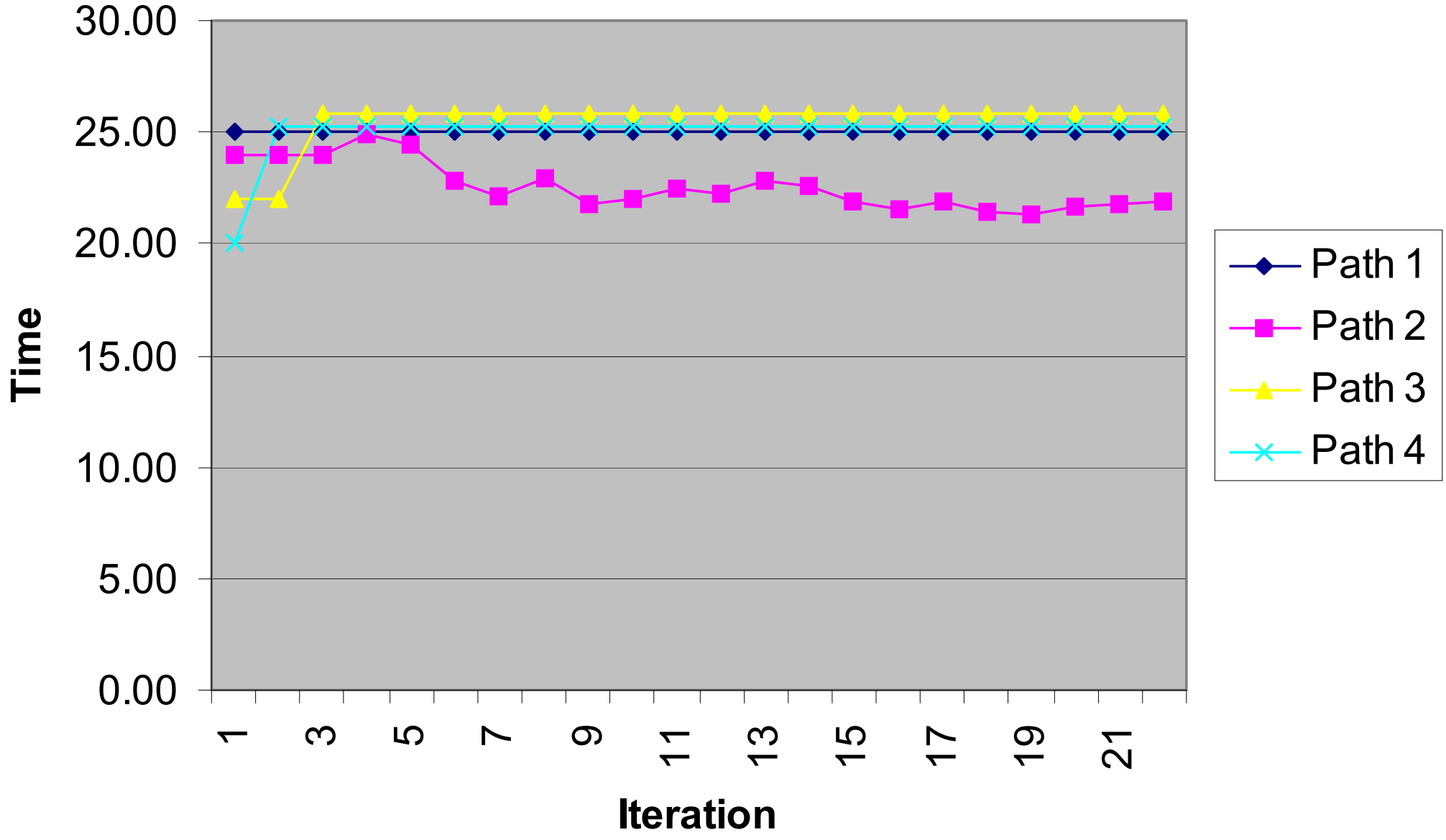
Finding the best path



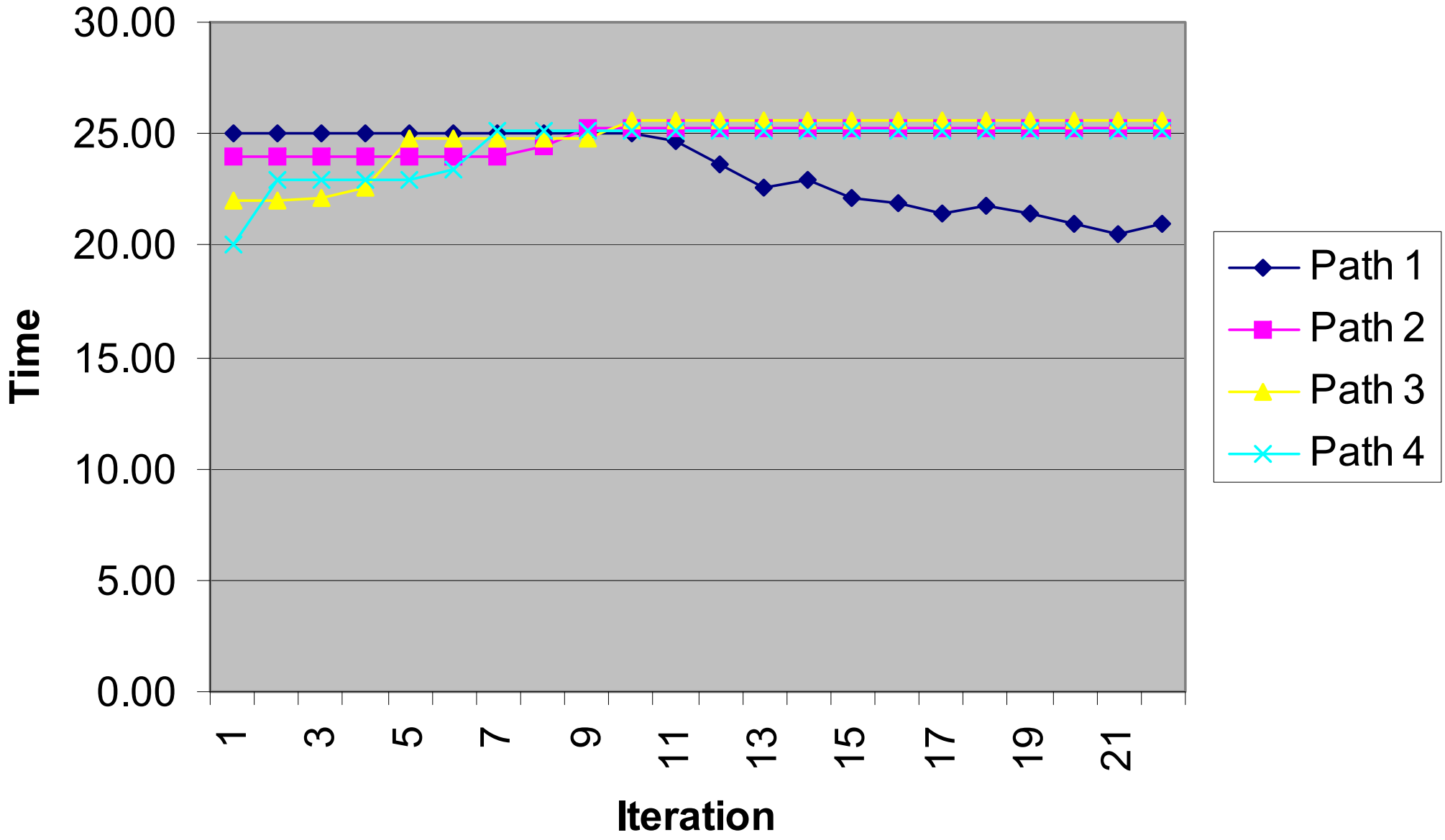
Finding the best path



Finding the best path



Finding the best path



Stochastic shortest paths I

Narrative

A deterministic shortest path problem

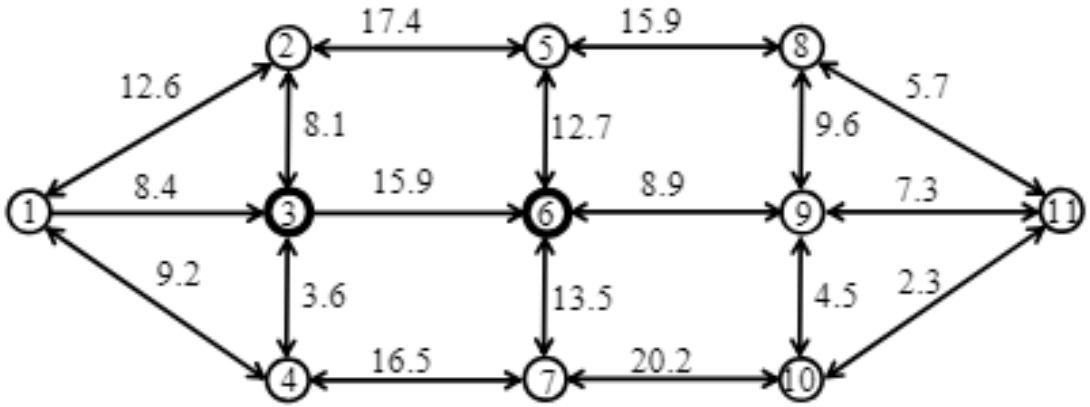


Figure 5.1 Network for a deterministic shortest path problem.

Shortest path problems over graphs are both an important application area (arising in transportation, logistics, and communication), but are also a fundamental problem class that arise in many settings. The most familiar shortest path problem is the classical deterministic problem illustrated in figure 5.1, where we have to find the best path from node 1 to node 11, where the cost of traversing each arc is known in advance.

Shortest path problems building on a fundamental dynamic programming recursion. Let

$$\begin{aligned}
\mathcal{N} &= \text{The set of all nodes in the network (here, this is the nodes } 1, 2, \ldots, 11), \\
\mathcal{N}_i^+ &= \text{The set of all nodes that can be reached direction from node } i, \\
\mathcal{L} &= \text{Set of all links } (i, j) \text{ in the network,} \\
c_{ij} &= \text{The cost of traversing link } (i, j), \text{ where } j \text{ is assumed to be in the set } \mathcal{N}_i^+.
\end{aligned}$$

Now let v_i be the minimum cost to get from node i to the destination node 11. The values v_i for all nodes $i \in \mathcal{N}$ should satisfy

$$v_i = \min_{j \in \mathcal{N}_i^+} (c_{ij} + v_j). \tag{5.1}$$

We can execute equation (5.1) by initializing v_{11} to zero, and setting all other values to some large number. If we loop over every node i and compute v_i using equation (5.1) repeatedly, the values v_i will converge to the optimal value. This is a very inefficient version of a shortest path algorithm.

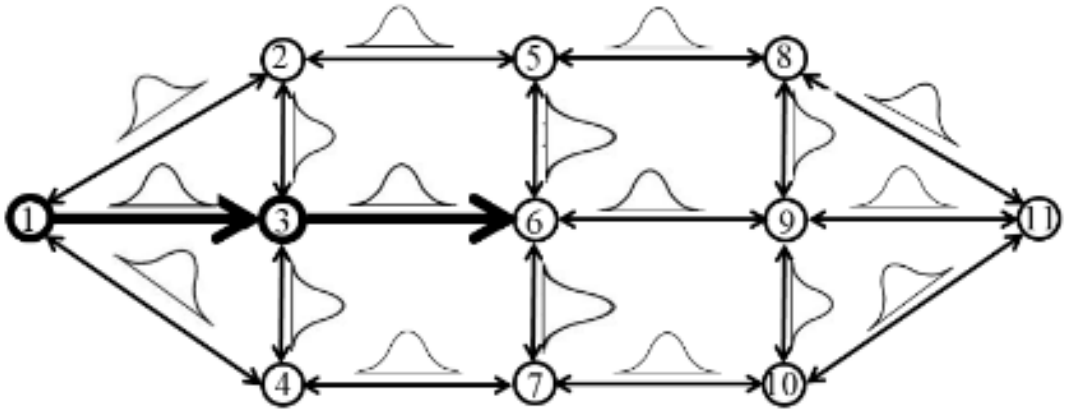


Figure 5.2 Network for a stochastic shortest path problem where distributions are known, but costs are not observed until after decisions are made.

5.1 NARRATIVE

You are trying to create a navigation system that will guide a driverless vehicle to a destination over a congested network. We assume that our system has access to both historical and real-time link costs, from which we can create estimates of the mean and variance of the cost of traversing a link. We can think of this as a shortest path problem where we see distributions rather than actual costs, as depicted in figure 5.2.

We are going to start by assuming that we have to make decisions of which link to traverse based on these distributions. After we traverse a link from i to j, we then experience a sample realization from the distribution. We want to choose a path that minimizes the expected costs.

Basic model

Stochastic network – costs revealed after decisions are made

State variables

In this basic problem, the state S_t is the node where we are located after t link traversals. For the basic problem, it is more natural to say that we are in "state" (node) i, but the state variable will become a lot more interesting in the extensions.

Decision variables

We are modeling decision as the node j that we traverse to given that we are at node i. There is a large community that works on problems that fit this class where the decision is represented as an action a, where a takes on one of a set of discrete values in the set $\mathcal{A}_s$ when we are in state s.

A convenient way to represent decisions is to define

$$x_{ij} = \begin{cases} 1 & \text{If we traverse link } i \text{ to } j \text{ when we are at } i, \\ 0 & \text{Otherwise.} \end{cases}$$

This notation will prove useful when we write our objective function.

Exogenous information

After traversing the link (i, j), we observe

$\hat{c}_{ij}$ = The cost we experience traversing from i to j.

For the moment, we are going to assume that the new observation $\hat{c}_{ij}$ is absorbed in a very large database. Over time, these observations may have the effect of changing $\bar{c}_{ij}$. Later, we are going to introduce problems where the estimates $\bar{c}_{ij}$ may be just rough approximations of the actual average link costs, where the decision to traverse link (i, j) will provide better estimates, but that is for later.

Transition function

For our basic graph problem, if we make decision $x = (i, j)$, the state $S_t = i$ evolves to state $S_{t+1} = j$.

Objective function

We can model our costs using

$\hat{c}_{ij}$ = A random variable giving the cost to traverse from node i to node j,

$\bar{c}_{ij}$ = The expected value of c_{ij} computed by averaging over our database of past travel costs,

= $\mathbb{E}\hat{c}_{ij}$,

$\bar{\sigma}_{ij}$ = Our estimate of the standard deviation of c_{ij} computed using historical data.

We write $\bar{c}_{ij} = \mathbb{E}\hat{c}_{ij}$ as the expected value of the random variable $\hat{c}_{ij}$ which is the actual link cost, although in practice this is really just an average over past observations.

We assume that we have to make the decision of which link to traverse out of a node i before seeing the actual value of the random cost $\hat{c}_{ij}$. This means that we have to make our decision using our best estimate of $\hat{c}_{ij}$, which would be $\bar{c}_{ij}$.

We could write our objective function using

$$\min_{x_{ij},(i,j)\in\mathcal{L}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}_i^+} \hat{c}_{ij}x_{ij}, \tag{5.3}$$

but this formulation would require that we know the realizations $\hat{c}_{ij}$. Instead, we are going to use the expectation, giving us

$$\min_{x_{ij},(i,j)\in\mathcal{L}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}_i^+} \bar{c}_{ij}x_{ij}, \tag{5.4}$$

The objective function

The optimal solution of this problem would be to set all $x_{ij} = 0$, which means we do not get a path. For this reason, we have to introduce *constraints* of the form:

$$\sum_j x_{qj} = 1, \tag{5.5}$$

$$\sum_i x_{ir} = 1, \tag{5.6}$$

$$\sum_i x_{ij} - \sum_k x_{jk} = 0, \text{ for } j \neq q, r. \tag{5.7}$$

$$x_{ij} \geq 0, \ (i,j) \in \mathcal{L}. \tag{5.8}$$

Equations (5.4) - (5.8) represent a *linear program*, and there are powerful packages that can be used to solve this problem when written this way. However, this approach does not provide a path for handling uncertainty. Our basic stochastic shortest path problem. Below, we describe how to solve the problem using our language of designing policies, which will provide a foundation for addressing uncertainty.

» In other words, we can think of this as a well defined problem where we call a black box solver. This could be a linear programming solver, or a shortest path algorithm.

Modeling uncertainty

Modeling uncertainty

5.3 MODELING UNCERTAINTY

For our basic model we are just using the point estimates $\bar{c}_{ij}$ which we assume is just an average of prior observations collected, for example, using travel cost estimates drawn from GPS-enabled smartphones. When estimates are based on field observations, the method is called *data-driven*, which means we do not need a model of uncertainty - we just need to observe it.

For example, let $\bar{c}_{ij}$ be our current estimate of the average travel cost for link (i, j) and assume we just observed a cost of $\hat{c}_{ij}$. We might update our estimate using

$$\bar{c}_{ij} \leftarrow (1 - \alpha)\bar{c}_{ij} + \alpha\hat{c}_{ij},$$

where α is a smoothing parameter (sometimes called a learning rate or a stepsize) that is less than 1.

» Discuss modeling the uncertainty of the realizations:
- Data driven modeling
- Empirical distributions
- Moment fitting for parametrics (e.g. normal)

Designing policies

5.4 DESIGNING POLICIES

Our "policy" for this deterministic problem is a function that maps the "state" (that is, what node we are at) to an action (which link we move over). We can solve this problem by optimizing the linear program represented by equations 5.4 - 5.8, which gives us the vector x^*_{ij} for all links (i, j). We can think of this as a function where given the state (node i) we can an action, which is the link (i, j) for which $x_{ij} = 1$. We can write this policy as a function $X^\pi(S_t)$ using

$$X^\pi(S_t = i) \quad = \quad j \text{ if } x_{ij} = 1.$$

Alternatively, we can solve Bellman's equation as we did initially for our deterministic shortest path problem using equation (5.1). This gives us a value v_i which is the minimum travel cost from each node i to the destination node r. Once these values are computed, we can make decisions using the following policy

$$X^\pi(i) \quad = \quad \arg\min_{j \in \mathcal{N}^+_i} (\bar{c}_{ij} + v_j). \tag{5.9}$$

This means that our "stochastic" shortest path problem can be solved just we solved our deterministic problem. Below we show that with a minor twist, the situation changes dramatically.

We can introduce a slight variant by assuming that our estimated costs depend on our "time" variable. Thus, instead of writing $\bar{c}_{ij}$, we would write $\bar{c}_{tij}$, in which case we would write Bellman's equation as

$$V_t(i) \quad = \quad \min_{j \in \mathcal{N}^+_i} (\bar{c}_{tij} + V_{t+1}(j)). \tag{5.10}$$

Here, we would pick a large time T and set $V_T(r)$ for our destination node r. We would then set $V_T(i)$ to a large number for all other nodes i other than r. Finally, we simply execute equation (5.10) by stepping backward in time. This is not a very efficient shortest path algorithm, but it is easy to accelerate by limiting the loop over all nodes i at time t to nodes that can be reached from nodes j that we have already reached.

- Notes:
  - » This basic problem is just a deterministic shortest path problem, which we can solve using the generic backward DP method we introduced at the beginning.
  - » The policy is optimal, since we are solving the original problem. This means we do not have to do tuning or policy evaluation.

Week 3 - Wednesday

Stochastic shortest paths

Stochastic shortest paths II

Adaptive routing

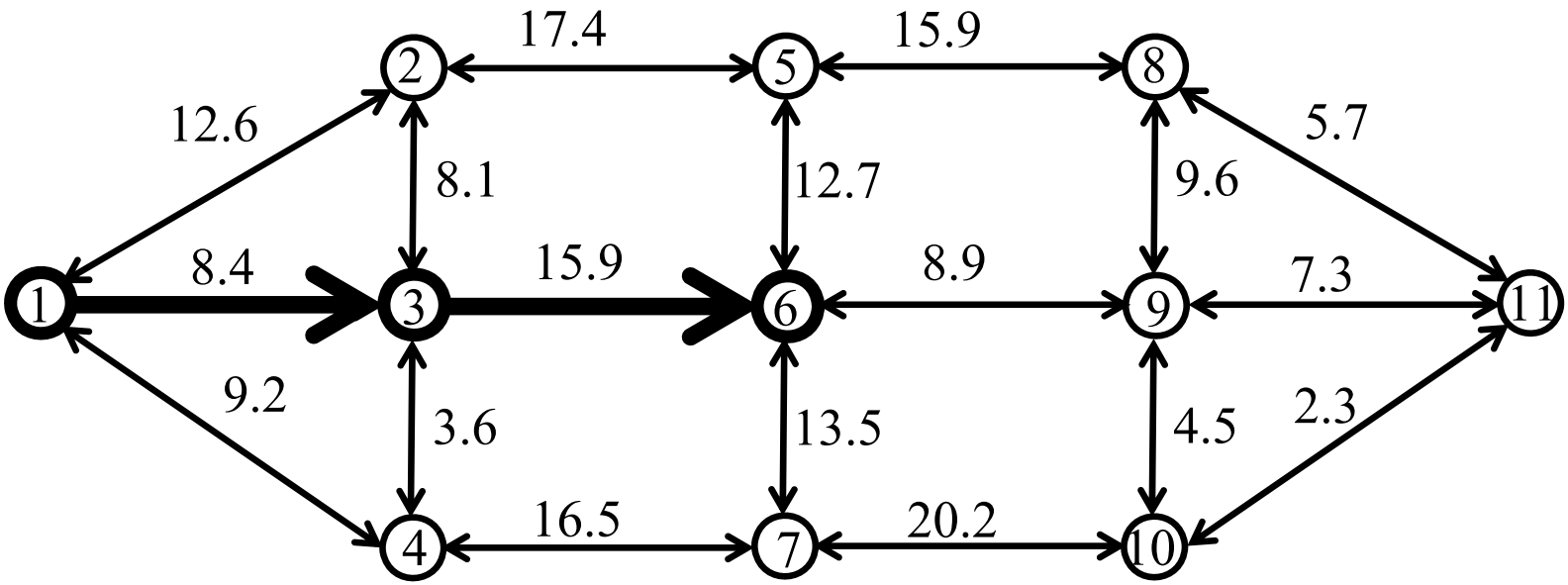
From last time - deterministic

- From deterministic to stochastic
  - » Pure deterministic
    - $v_i = \min_j (c_{ij} + v_j)$
  - » Stochastic, see costs after making decision
    - $v_i = \min_j E(\hat{c}_{ij} + v_j)$

    or
    - $V_t(S_t) = \min_j E\{\hat{c}_{ij} + V_{t+1}(S_{t+1})|S_t\}$
      - – Explain that S_t is state after "t" link transitions.
  - » Stochastic – see costs before making a decision
    - $V_t(S_t) = \min_j \mathrm{E}\{\hat{c}_{ij} + V_{t+1}(S_{t+1})|S_t\}$
    - $V_t(S_t) = \min_j (\hat{c}_{ij} + E\{V_{t+1}(S_{t+1})|S_t\})$
      - – Note that $\hat{c}_{ij}$ is not random given S_t

The state variable

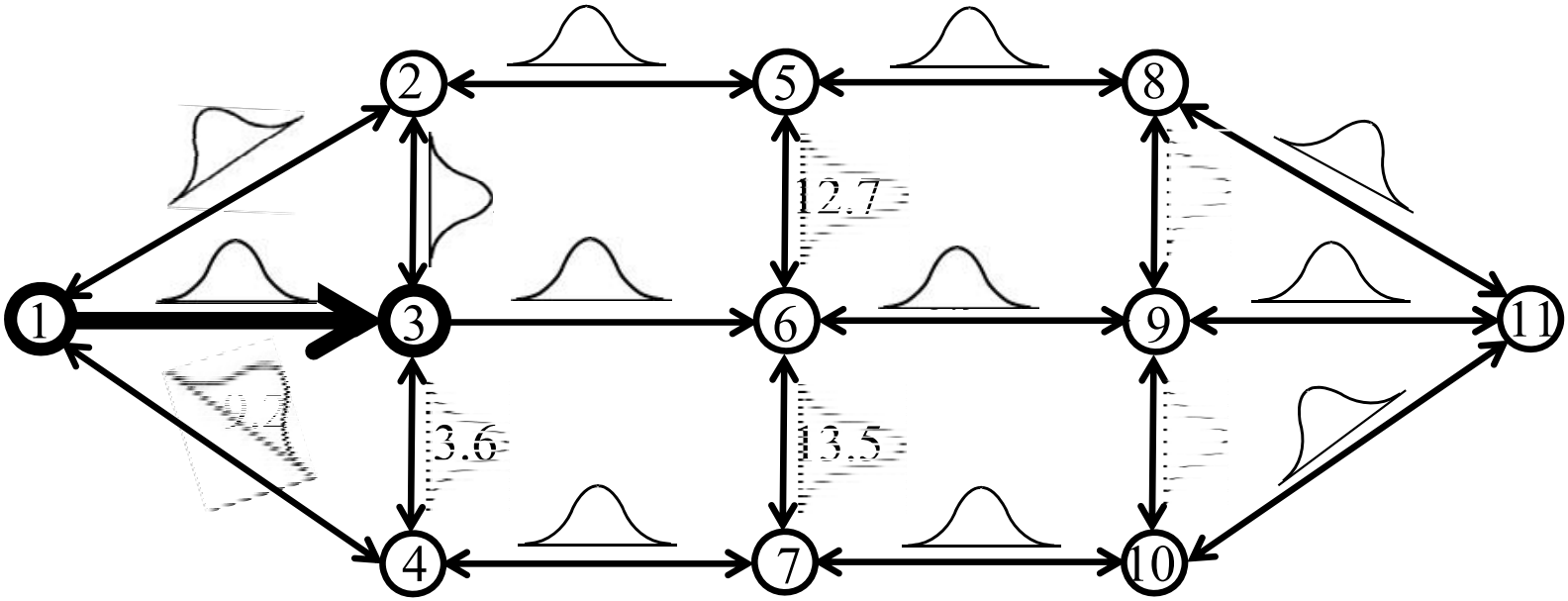
- Illustrating state variables
  - » A deterministic graph



$$S_t = (N_t) = 6$$

The state variable

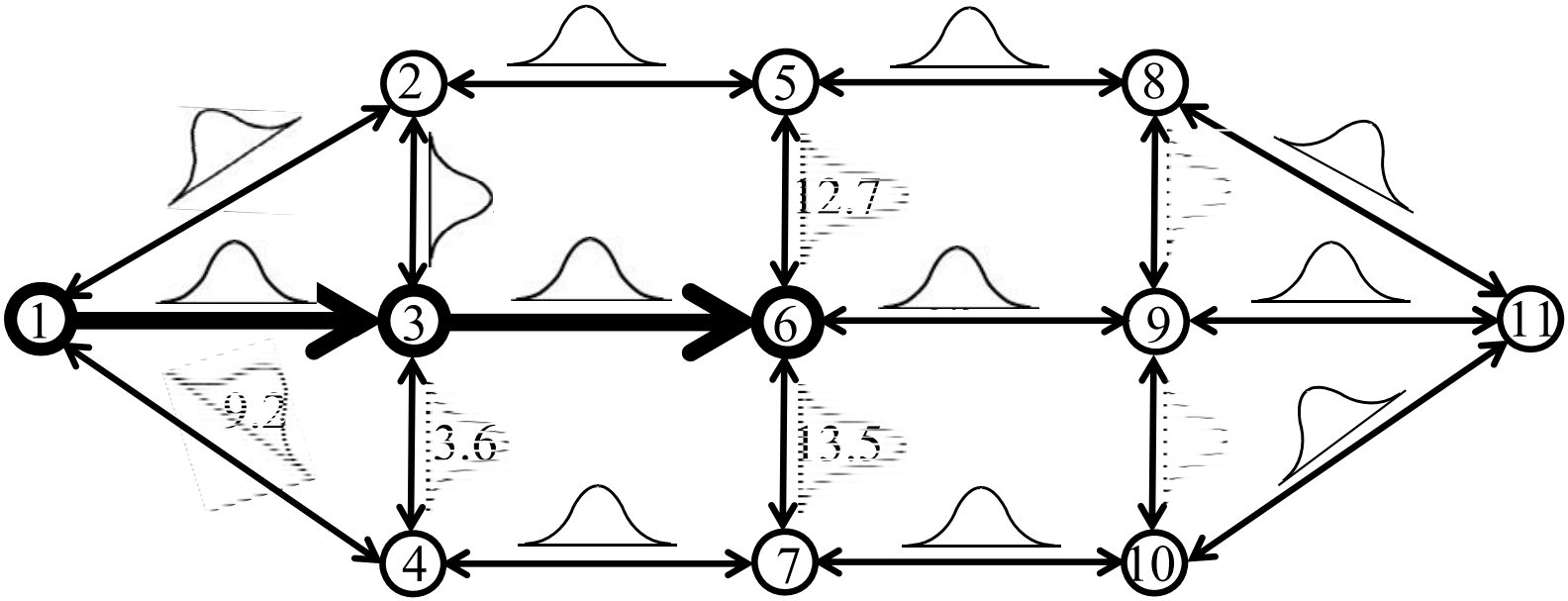
- Illustrating state variables
  - » A stochastic graph



$$S_t = ?$$

The state variable

- Illustrating state variables
  - » A stochastic graph



$$S_t = \Big(\underbrace{N_t}_{R_t}, \underbrace{\big(c_{t,N_t,j}\big)_j}_{I_t} \Big) = \big(6, (12.7, 8.9, 13.5)\big)$$

In our new stochastic model, just capturing the node where our traveler is located is no longer enough. Remember that above, we introduced a state variable as "all the information we need at time t (or after n iterations) from history to model the system from time t (or n) onward." While this description is accurate, it is useful to introduce a more complete definition:

Definition 5.5.1. *A* **state variable** *is: A function of history that is necessary and sufficient to compute the cost/contribution function, the constraints, and any information required to model the evolution of information needed in the cost/contribution function and the constraints.*

Our new stochastic shortest path problem introduces new information that is needed to make a decision: the costs out of the node where we are located. We are going to find it convenient to introduce two new state variables:

$$\begin{aligned} R_t &= \text{The physical state of the system, which is usually controlled directly by decisions,} \\ I_t &= \text{Other information that we need to make a decision.} \end{aligned}$$

In our network problem, our physical state would be the node where we are located, while the "other information" variable I_t would capture the costs on links out of the node where we are located. Assume that at time t that $R_t = i$. We are going to add time t to our index for link costs, which means we will replace $\hat{c}_{ij}$ with $\hat{c}_{tij}$ to refer to the cost when we go from i to j at time t. We might then write

$$\begin{aligned} S_t &= (R_t, I_t) \\ &= \left(i, (\hat{c}_{tij})_{j \in \mathcal{N}_i^+}\right). \end{aligned}$$

To see what this does to our previous way of solving our shortest path problem, take a fresh look at Bellman's equation as we first introduced it in equation (5.2)

$$V_t(S_t) = \min_{x \in \mathcal{X}_s} \big(C(S_t, x) + V_{t+1}(S_{t+1})\big). \tag{5.12}$$

where S_{t+1} would be given by

$$\begin{aligned}
S_{t+1} &= (R_{t+1}, I_{t+1}), \\
R_{t+1} &= \text{The node produced by our decision } x. \text{ So if } x_{ij} = 1, \text{ then } R_{t+1} = j, \\
I_{t+1} &= \text{The costs that are observed out of node } R_{t+1}, \text{ which depends on the decision } x. \text{ If } x \text{ sends us to node } j \text{ so that } R_{t+1} = j, \text{ then } I_{t+1} = (\hat{c}_{t+1,jk_1}, \hat{c}_{t+1,jk_2}, \hat{c}_{t+1,jk_3}) \text{ (assuming there are three links out of node } j).
\end{aligned}$$

Assume that $R_t = i$. The cost function $C(S_t, x)$ is given by

$$C(S_t, x) = \sum_{j \in \mathcal{N}_i^+} \hat{c}_{tij} x_{ij}.$$

Remember that S_t (where $R_t = i$) contains the costs $\hat{c}_{tij}$ for the links (i, j) out of i, so these are known.

But S_{t+1} is random when we are at time t! Need to introduce an expectation:

$$V_t(S_t) = \min_{x \in \mathcal{X}_s} \big(C(S_t, x) + \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t, x\}\big). \tag{5.13}$$

- Finding a policy
  - » We could try using Bellman's equation, but now the transition from S_t to S_{t+1} is stochastic because of revealing the new link times, so we have to write Bellman's equation as:

$$V_t(S_t) = \min_{x \in \mathcal{X}_s} \Big(C(S_t, x) + \mathbb{E}\{V_{t+1}(S_{t+1})|S_t, x\}\Big). \tag{5.13}$$

We can write this more explicitly. Assume that x sends us to node j (which means that $x_{ij} = 1$), and when he arrives he sees $I_{t+1} = (\hat{c}_{t+1,jk_1}, \hat{c}_{t+1,jk_2}, \hat{c}_{t+1,jk_3})$. We learn these costs when we arrive to node j at time $t+1$, but they are random when we are at node i at time t thinking about what to do.

Now we face a new challenge: how do we compute the expectation $\mathbb{E}\{V_{t+1}(S_{t+1})|S_t, x\}$? Assume that each cost $\hat{c}_{jk}$ can take on values $c_1, c_2, \ldots, c_L$ with probabilities $p_{jk}(c_{j\ell})$. For example, c_1 might be 1 minute, c_2 might be 2 minutes, and so on. The probability $p_{jk}(c_\ell)$ is the probability that $\hat{c}_{jk} = c_\ell$. Assume that the decision x takes us to node j, after which we face a choice of traveling over links $(j, k_1), (j, k_2)$ or (j, k_3). We would compute our expectation using

$$\begin{aligned}\mathbb{E}\{V_{t+1}(S_{t+1})|S_t, x\} \quad = \quad & \sum_{\ell_1=1}^{L} p_{jk_1}(c_{\ell_1}) \sum_{\ell_2=1}^{L} p_{jk_2}(c_{\ell_2}) \sum_{\ell_3=1}^{L} p_{jk_3}(c_{\ell_3}) \\ & V_{t+1}(S_{t+1} = (j, (c_{\ell_1}, c_{\ell_2}, c_{\ell_3}))).\end{aligned} \tag{5.14}$$

- Challenges:
  - » Computing Bellman's equation for a single state is hard enough.
  - » We have to compute it for each state S_t. But instead of just being anode, it is now a four dimensional vector. In our network example, our state was:

$$S_t = \left(N_t, \left(c_{t,N_t,j} \right)_j \right) = \left(6, \left(12.7, 8.9, 13.5 \right) \right)$$

  - » If we discretize costs into, say, 20 buckets, then our state space would be $20 \times 20 \times 20 \times$number of nodes.
  - » This is just not going to work.

- Solution approach:
  - » Introduce post-decision state S_t^x =the state immediately after a decision has been made.
  - » In our network example above, the post-decision state when we are at S_t:

$$S_t = \left(6, \left(12.7, 8.9, 13.5\right)\right) \rightarrow S_t^x = (9)$$

  - » Note that we are still at node 6, but we have made the *decision* to go to node 9 (but have not yet arrived there). So we do not yet know the costs on the links out of node 9.

- Bellman's original equation:

$$V_t(S_t) = \min_{x \in \mathcal{X}_t} \left(C(S_t, x) + \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t, x\} \right). \tag{5.13}$$

- From pre- to post-decision state:

$$V_t(S_t) = \min_{x \in \mathcal{X}_t} \left(C(S_t, x) + V_t^x(S_t^x) \right), \tag{5.15}$$

- From post- to pre-decision state:

$$V_t^x(S_t^x) = \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t, x\}. \tag{5.16}$$

- This expectation is still hard to compute, but we have tricks…

- We are going to use an approach called "approximate dynamic programming"

We are going to construct approximations $\overline{V}_t^x(j)$ of the value of being at node j, where

$$\overline{V}_t^{x,n}(j) \approx \mathbb{E}V_{t+1}(S_{t+1}).$$

Let $\overline{V}_t^{x,n}(j)$ be our approximation of $\mathbb{E}V_{t+1}(S_{t+1})$ after n samples. One way to build this approximation is to use samples of the value of being at node j. Imagine that we are going to pass forward through the network, making decisions using approximations $\overline{V}_t^{x,n-1}(S_t)$ obtained from previous iterations, along with sampled costs $\hat{c}_{ij}^n$. We can obtain a sampled estimate of the value of being in state S_t using

$$\hat{v}_t^n(i) = \min_{j \in \mathcal{N}_i^+} \left(\hat{c}_{ij}^n + \overline{V}_t^{x,n-1}(j)\right). \tag{5.17}$$

We are then going to update our estimate of the value of being at node i using

$$\overline{V}_t^{x,n}(i) = (1-\alpha_n)\overline{V}_t^{x,n-1}(i) + \alpha_n \hat{v}_t^n(i). \tag{5.18}$$

- Steps (create notation board to review all notation)
  - » Use $\bar{V}_t^{x,n-1}$ to make a decision. Imagine we are at node i_t^n at time t, while we are simulating the nth path.

$$i_{t+1}^n = argmax_j \, (\hat{c}_{ij} + \bar{V}_{tj}^{x,n-1})$$

  - » This tells us which node to go to, arriving at time $t+1$.
  - » When we arrive at the next node i_{t+1}^n, we then randomly sample the costs on links out of node i_{t+1}^n. This avoids having to enumerate all possible costs.
  - » Compute

$$\hat{v}_{t,i_t^n}^n = max_j \, \hat{c}_{i_t^n,j} + \bar{V}_{tj}^{x,n-1}$$

  - » Now update the previous, post-decision state:

$$\bar{V}_{t-1,i_{t-1}^n}^{x,n} = (1-\alpha)\bar{V}_{t-1,i_{t-1}^n}^{x,n-1} + \alpha\hat{v}_{t,i_t^n}^n$$

- Basic idea
  - » Use the value function approximations around the post-decision state $\bar{V}_{tj}^{x,n-1}$ to create a policy to determine what to do next.
  - » As you transition to a pre-decision state (that is, you reveal the costs), then obtain a sampled estimate of the value of the pre-decision state $\hat{v}_{t,i_t^n}^n$.
  - » Use this to update the *previous post-decision state.*
  - » This is known as (single pass) approximate value iteration.

Stepsizes

- Stepsizes:
  - » Approximate value iteration requires updates of the form:

$$V_{t-1}^{n}(S_{t-1}^{x}) = (1-\alpha_{n-1})V_{t-1}^{n-1}(S_{t-1}^{x}) + \alpha_{n-1}\hat{v}_t^n$$

Updated estimate

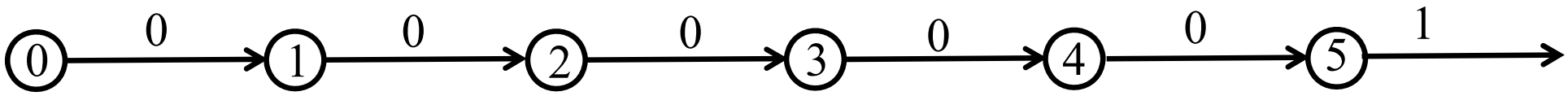
Old estimate

New observation

The stepsize
"Learning rate"
"Smoothing factor"

Stepsizes

- Single state, single action Markov chain
  - » Updating – receives reward=1 at last node. All other rewards = 0.



  - » Same as adding up random rewards with mean of 1. Noise may be zero, or quite high.

Table 9.1 Effect of stepsize on backward learning

Iteration	$\overline{V}_0$	$\hat{v}_1$	$\overline{V}_1$	$\hat{v}_2$	$\overline{V}_2$	$\hat{v}_3$	$\overline{V}_3$	$\hat{v}_4$	$\overline{V}_4$	$\hat{v}_5$
0	0.000		0.000		0.000		0.000		0.000	1
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1
2	0.000	0.000	0.000	0.000	0.000	0.000	0.500	1.000	1.000	1
3	0.000	0.000	0.000	0.000	0.167	0.500	0.667	1.000	1.000	1
4	0.000	0.000	0.042	0.167	0.292	0.667	0.750	1.000	1.000	1
5	0.008	0.042	0.092	0.292	0.383	0.750	0.800	1.000	1.000	1
6	0.022	0.092	0.140	0.383	0.453	0.800	0.833	1.000	1.000	1
7	0.039	0.140	0.185	0.453	0.507	0.833	0.857	1.000	1.000	1
8	0.057	0.185	0.225	0.507	0.551	0.857	0.875	1.000	1.000	1
9	0.076	0.225	0.261	0.551	0.587	0.875	0.889	1.000	1.000	1
10	0.095	0.261	0.294	0.587	0.617	0.889	0.900	1.000	1.000	1

Stepsizes

- Bound on performance using 1/n:

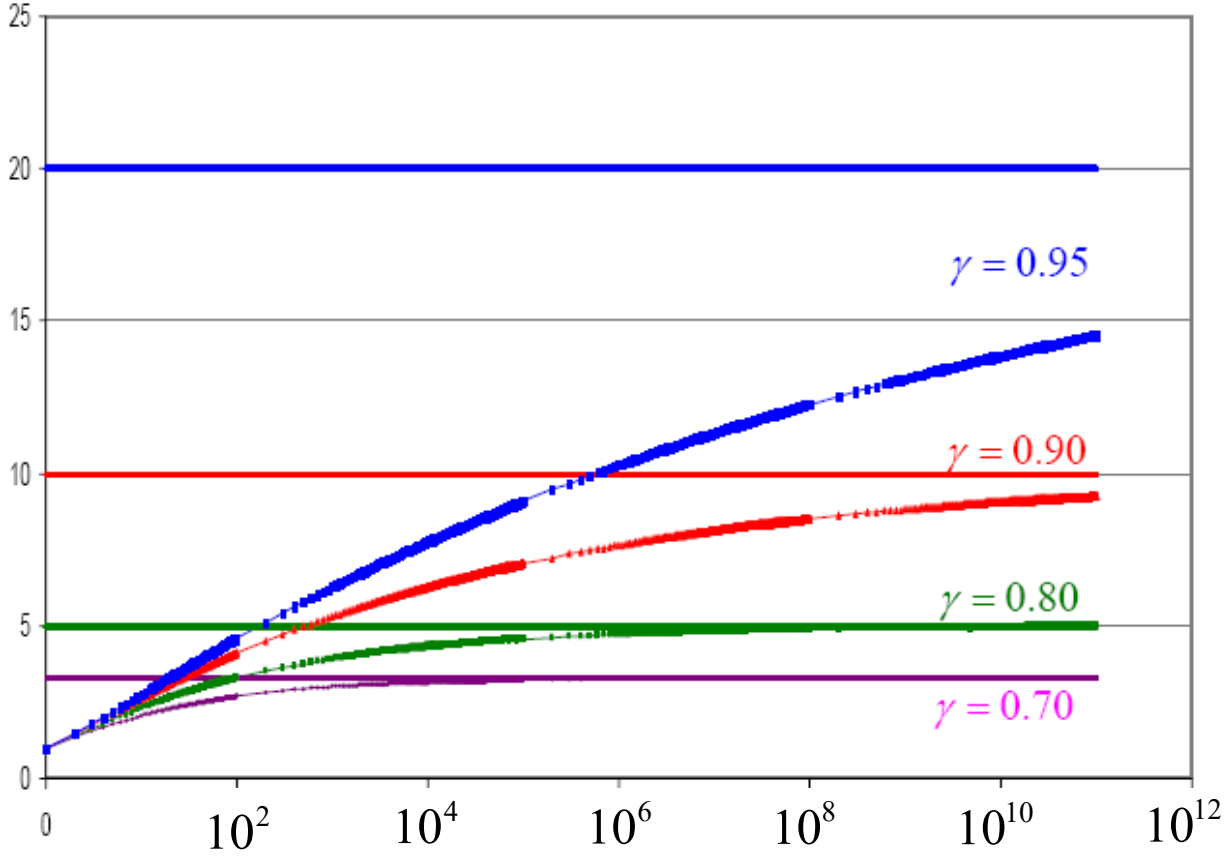
Single state, single action

$$V = \sum_{n=0}^{\infty} \gamma^n c = \frac{c}{1-\gamma}$$

$$\hat{v}^{n+1} = c + \gamma \bar{v}^n$$

$$\bar{v}^{n+1} = (1-\alpha_n)\bar{v}^n + \alpha_n \hat{v}^{n+1}$$

$$\nu^L(n) = \frac{c}{1-\gamma}\left(1 - \left(\frac{1}{1+n}\right)^{1-\gamma}\right)$$



Stepsizes

- Some basic tricks:
  - » Use a constant stepsize, but you have to tune.
  - » Harmonic stepsize, but again – careful tuning.
  - » Use a stepsize = 1 for T iterations (T = horizon), then switch to harmonic.

Stepsize policies

- BAKF stepsize rule (from chapter 6)

It can be shown (see section 6.7.1) that the optimal stepsize is given by

$$\alpha_{n-1} = 1 - \frac{\sigma^2}{(1+\lambda^{n-1})\sigma^2 + (\beta^n)^2}, \tag{6.42}$$

where λ is computed recursively using

$$\lambda^n = \begin{cases} (\alpha_{n-1})^2, & n = 1 \\ (1-\alpha_{n-1})^2\lambda^{n-1} + (\alpha_{n-1})^2, & n > 1. \end{cases} \tag{6.43}$$

The BAKF stepsize formula enjoys several nice properties:

» Properties:

- With static mean, $\alpha_n = \frac{1}{n}$
- With no noise, $\alpha_n = 1$
- At all times, $\alpha_n \geq \frac{1}{n}$

Stepsizes

- Single pass version
  - » For the single-pass version, stepsizes are tricky.
  - » I recommend using a stepsize of $\alpha = 1$ for L steps, where L is roughly the length of a path.
  - » Then switch to something smaller – perhaps 0.20, but the choice depends on how much noise there is in the costs.
    - Less noise, increase α toward 1.
    - More noise, decrease α toward $1/n$.
  - » More sophisticated: BAKF

- A double-pass version
  - » The single pass version, where we update as we progress forward, can exhibit slow backward communication.
  - » A different strategy is to simulate the full path forward without doing any updating.
  - » Then do a backward pass, retracing steps along the same pass while computing:
    - $\hat{v}^n_{t,i^n_t} = max_j \, \hat{c}_{i^n_t,j} + \hat{v}^n_{t,i^n_{t+1}}$
  - » Finally update as we did before:
    - $\bar{V}^{x,n}_{t-1,i^n_{t-1}} = (1-\alpha)\bar{V}^{x,n-1}_{t-1,i^n_{t-1}} + \alpha\hat{v}^n_{t,i^n_t}$

Stepsizes

- Double pass version
  - » Harmonic stepsize rule likely to be best.

- The policy

  » Regardless of strategy, when we are done, the policy looks like

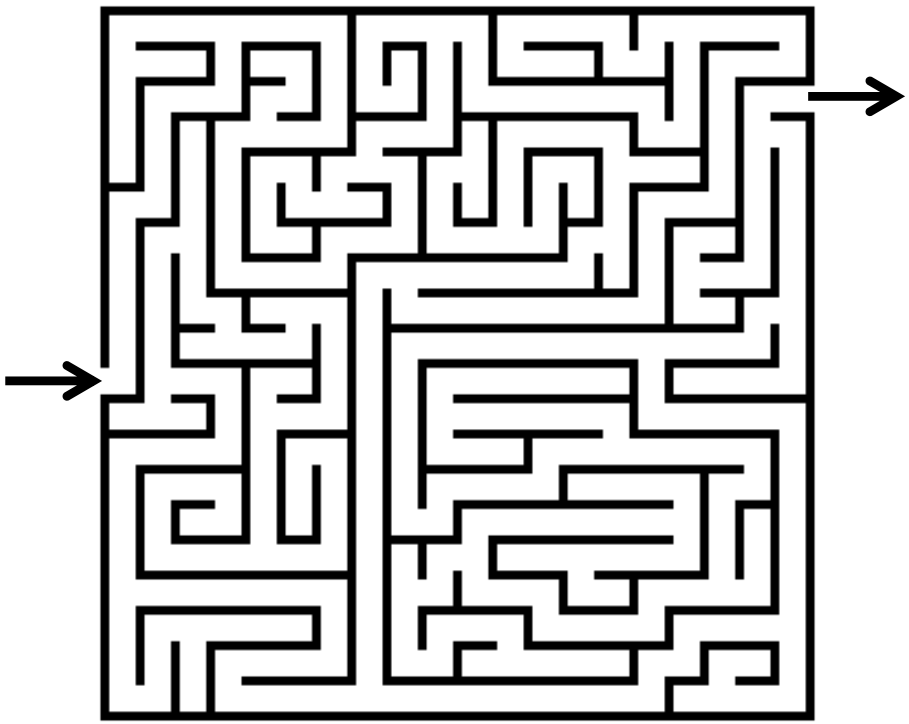
$$X^{\pi}(S_t) = argmax_j \, (\hat{c}_{ij} + \bar{V}_{tj}^{x,n-1})$$

  » The performance of the policy depends on the algorithmic strategies used to compute the value function approximations.

Q-learning

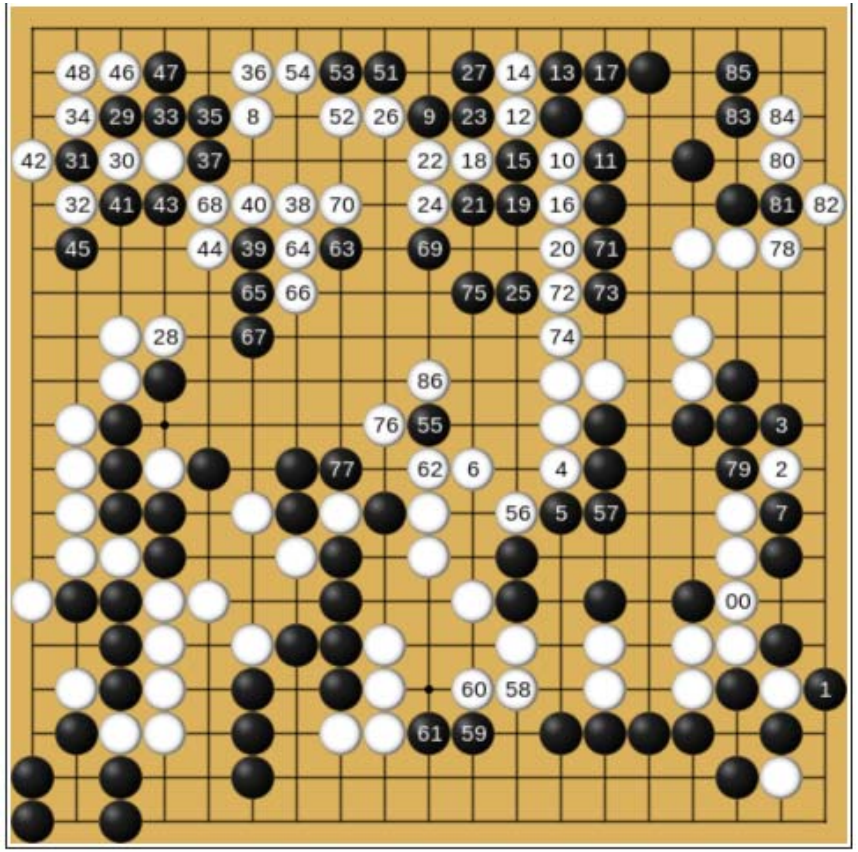
Q-learning

- Mouse in a maze problem



Q-learning

- AlphaGo
  - » Much more complex state space.
  - » Uses hybrid of policies:
    - PFA
    - VFA
    - Lookahead (DLA)



Q-learning

- Basic Q-learning algorithm
  - » Basic update:

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

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

where

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

  - » 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

Q-learning

- 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).

Q-learning

- Learning policy
  - » This is the policy that determines what action to choose as a part of learning the Q-factors.
  - » "Exploitation":

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

  - » 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, …
    - This is learning with a physical state.

Q-learning

- 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.

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

Q-learning

- Implementation policy
  - » This is the policy we are going to follow based on the Q-factors:

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

- The value of the implementation policy:

$$\bar{F}^{\pi,n} = \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1}(\omega)) \quad \text{where} \quad S_{t+1} = S^{M}(S_t, X^{\pi}(S_t), W_{t+1}(\omega))$$

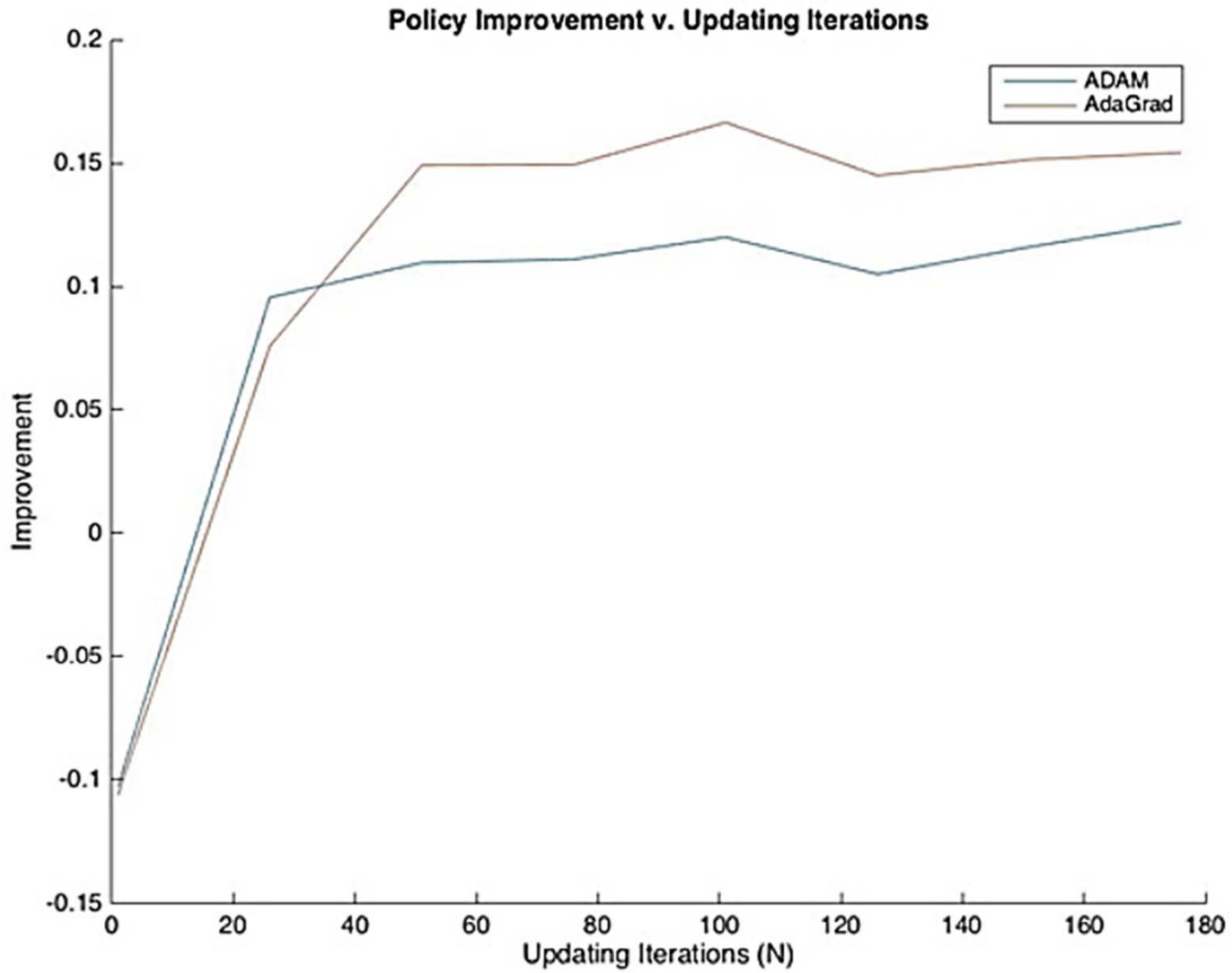
  - » or

$$\bar{F}^{\pi,n} = \frac{1}{N}\sum_{n=1}^{N-1}\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1}(\omega^n)) \quad \text{where} \quad S_{t+1} = S^{M}(S_t, X^{\pi}(S_t), W_{t+1}(\omega^n)))$$

- The goal is to find an effective learning policy so that we obtain the best implementation policy.

Q-learning

- Convergence rates:



Q-learning

- On vs off-policy learning:
  - » On-policy learning – Behavior according to the policy dictated by the Q-factors::

    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.

Q-learning

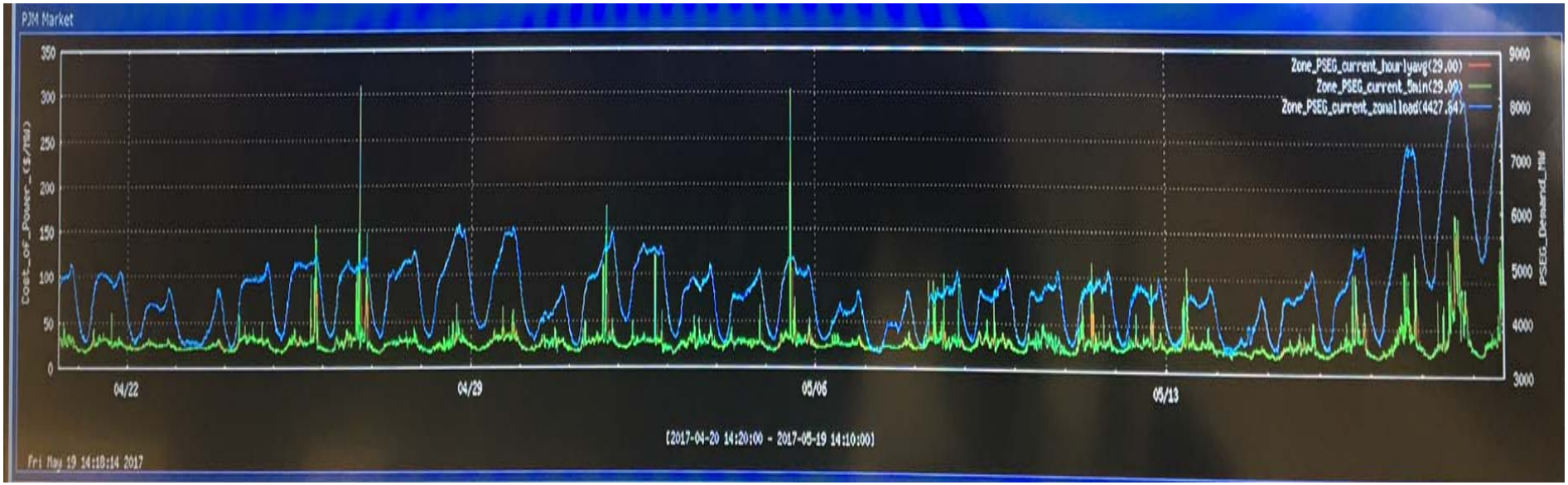
- Model-free vs. model-based
  - » Model-based means we have a mathematical statement of how the problem evolves that can be simulated in the computer.
  - » Model-free refers to a physical process that can be observed, but where we do not have equations describing the evolution over time.
    - The 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

- Notes
  - » Lookup table belief models are most popular, but do not scale (limit of 3 dimensions).
  - » Various smoothing strategies have been suggested (basically nonparametric statistics), but still limited to 3 dimensions.
  - » Need to be very careful with stepsizes. Q-learning is a form of approximate value iteration where the backward learning is slowed by the use of stepsizes.

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

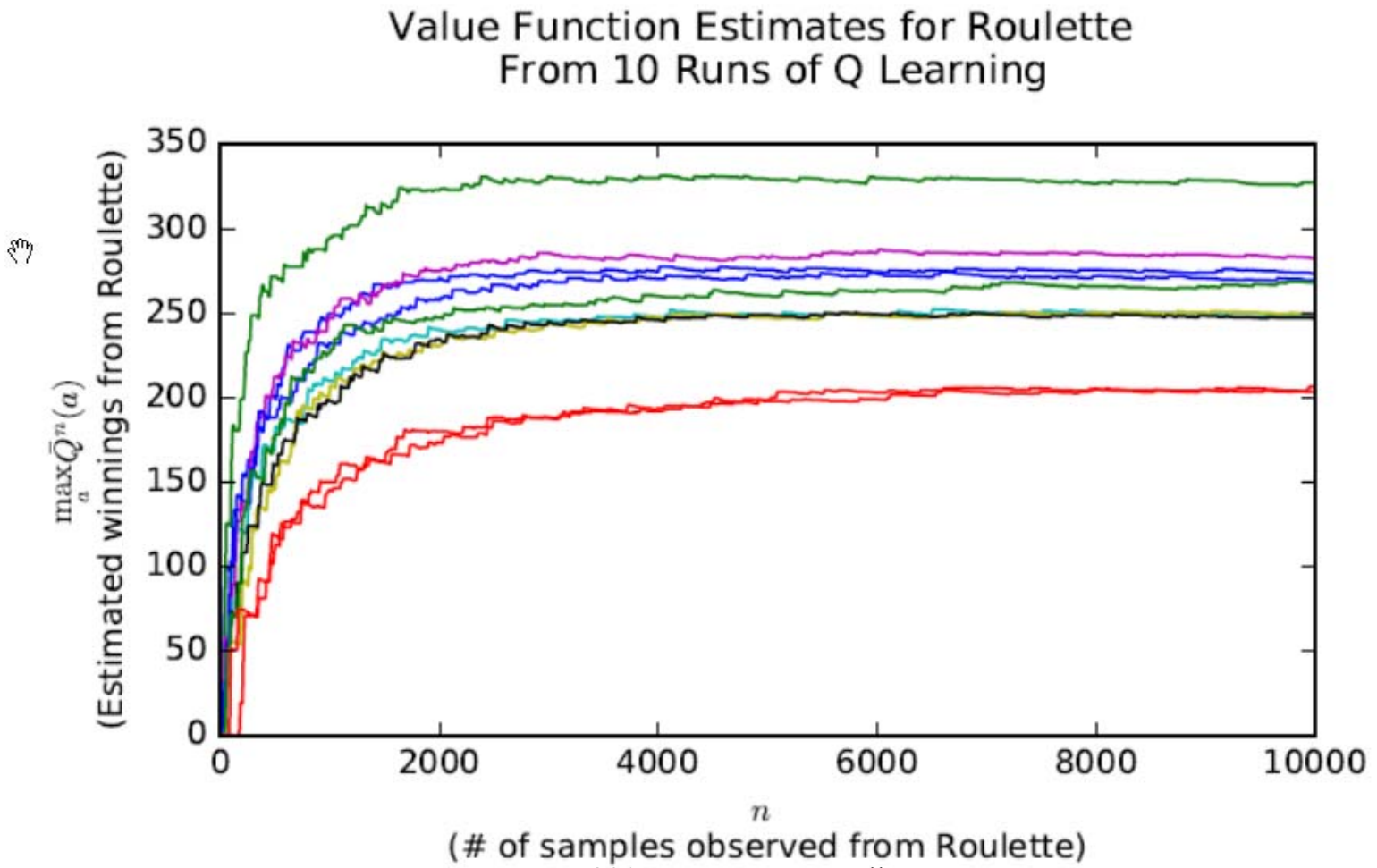
- Max operator bias (cont'd)
  - » This introduces noise in $\hat{q}^n(s^n, a^n)$:

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

  - » 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

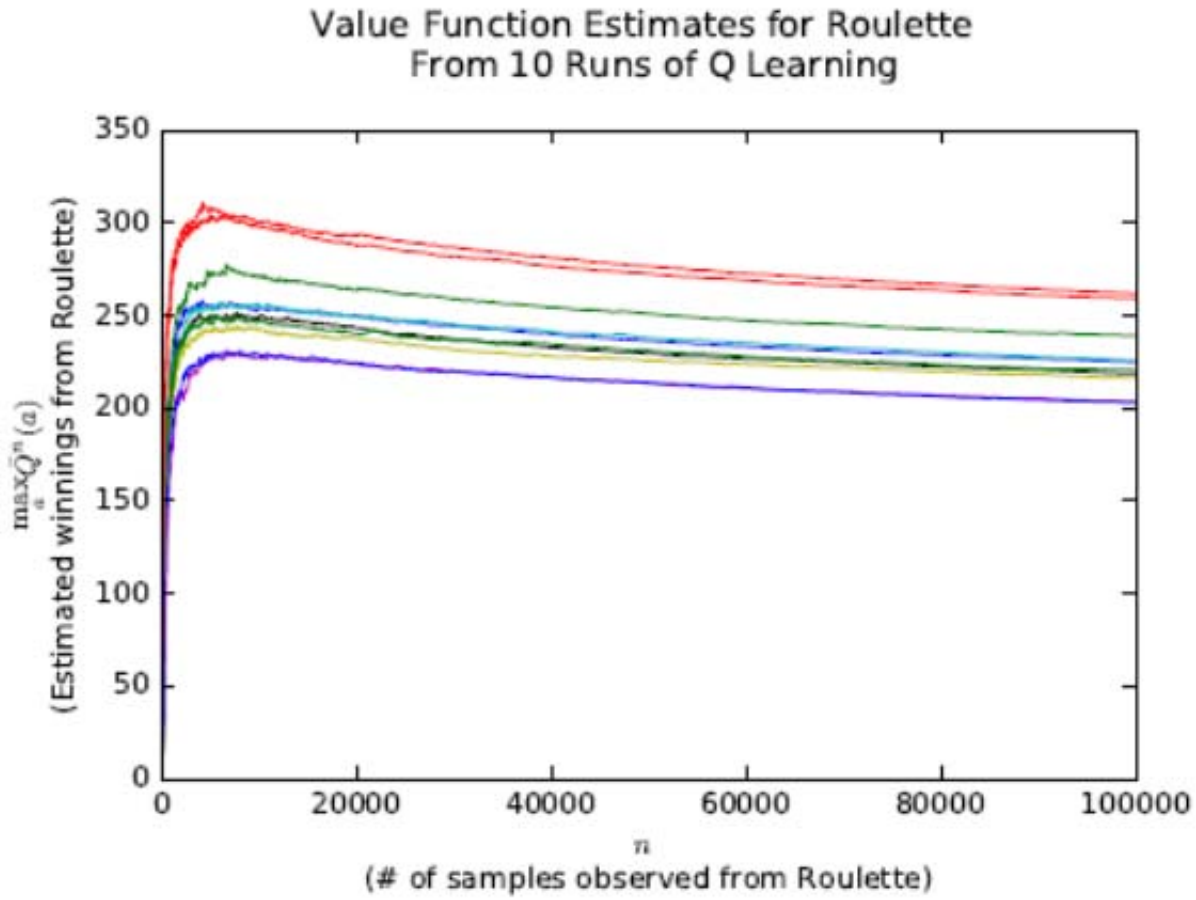
Q-learning

- Roulette
  - » Optimal solution is not to play – optimal value of game is zero
  - » Q-learning over 10,000 iterations

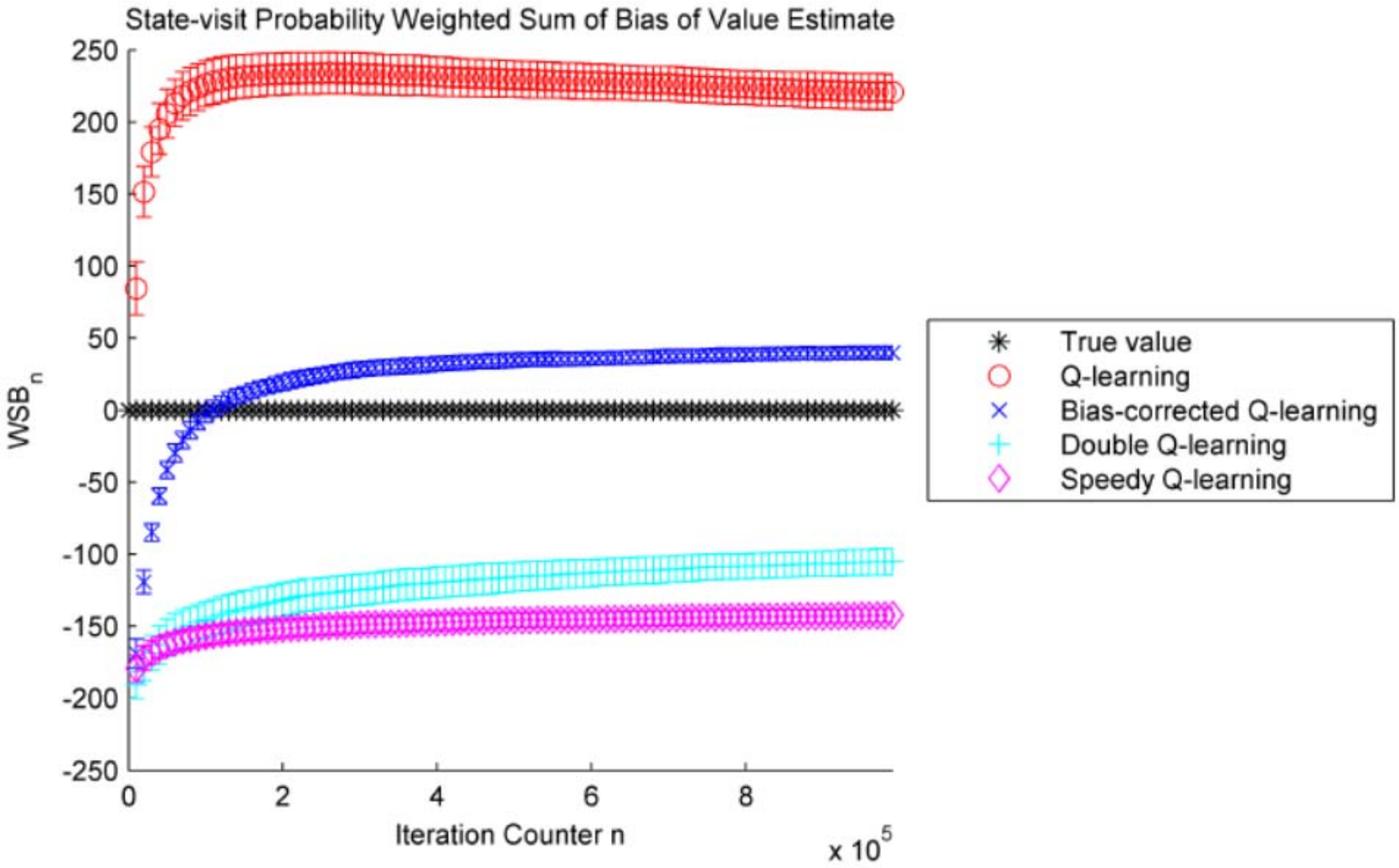


Q-learning

- Roulette
  - » Optimal solution is not to play – optimal value of game is zero
  - » Q-learning over 10,000 iterations



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



Policy evaluation

- Notes:
  - » Policies based on value function approximations should be good, but are not optimal.
  - » We have seen different strategies:
    - Forward pass, backward pass
    - Choice of stepsize formulas
  - » To find the best algorithmic strategy, we need to evaluate the performance of the policy by simulating it.

$$\hat{F}^k(\omega^k) = \sum_{t=0}^{T} C(S_t(\omega^k), X^\pi(S_t))$$

Then average the samples to obtain:

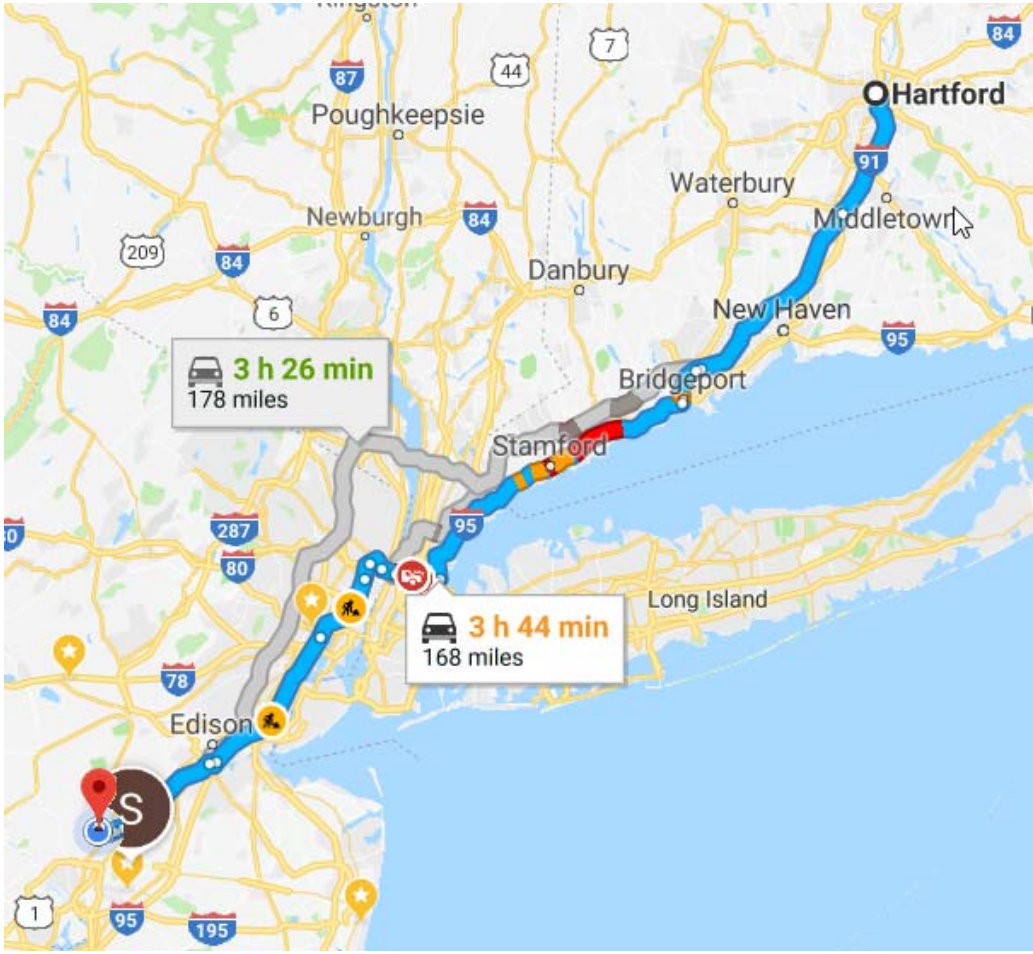
$$\bar{F}^\pi(\theta) = \frac{1}{K}\sum_{k=1}^{K} \hat{F}^k(\omega^k)$$

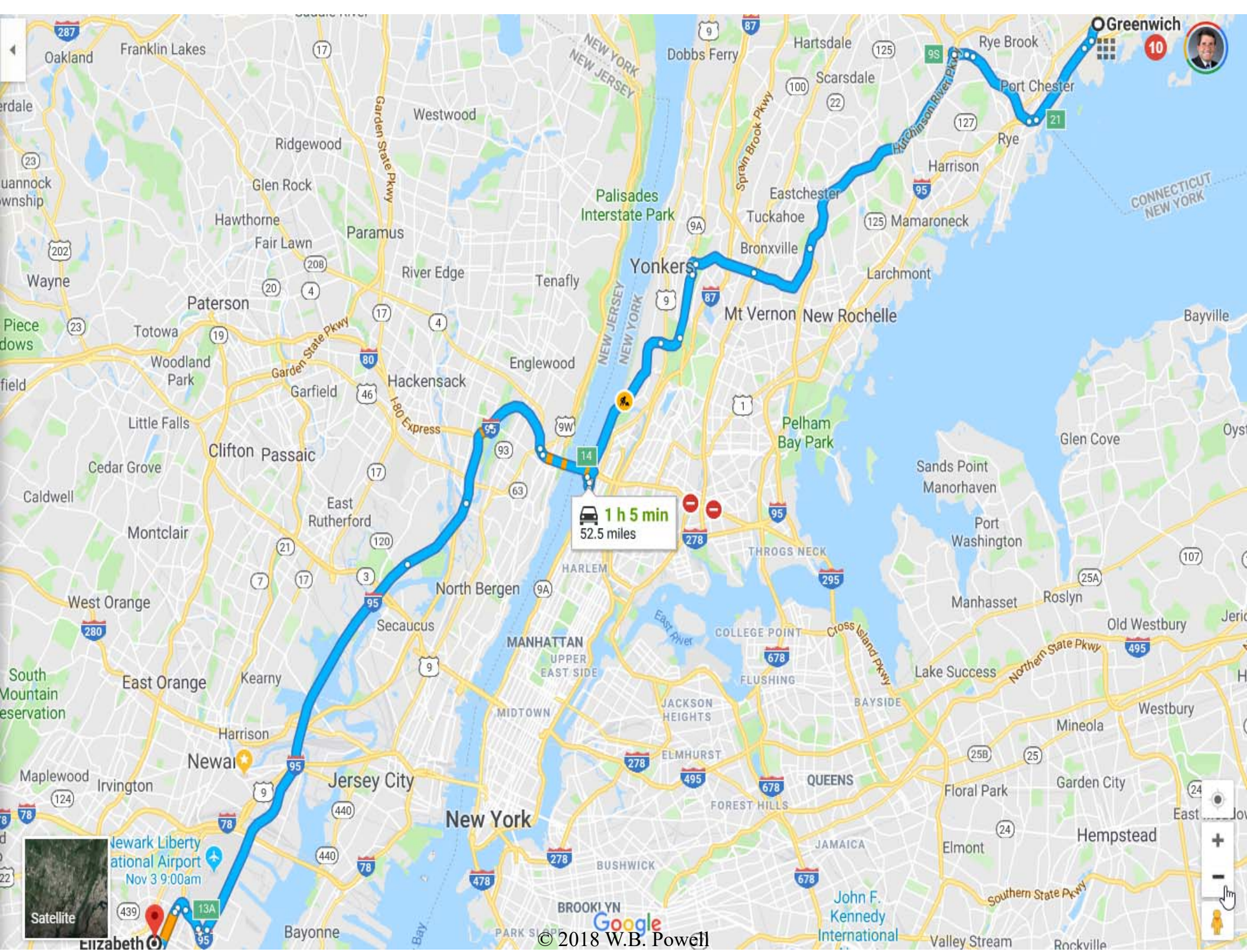
Week 4 - Monday

Dynamic shortest paths

Narrative

- The google maps story:
  - » Use current estimates of travel times on links to find the best path at a time *t*.
  - » As time passes, google gets updated information on travel times.
  - » The shortest path may change over time since costs are dynamic (and of course stochastic).
  - » We want to model this process.





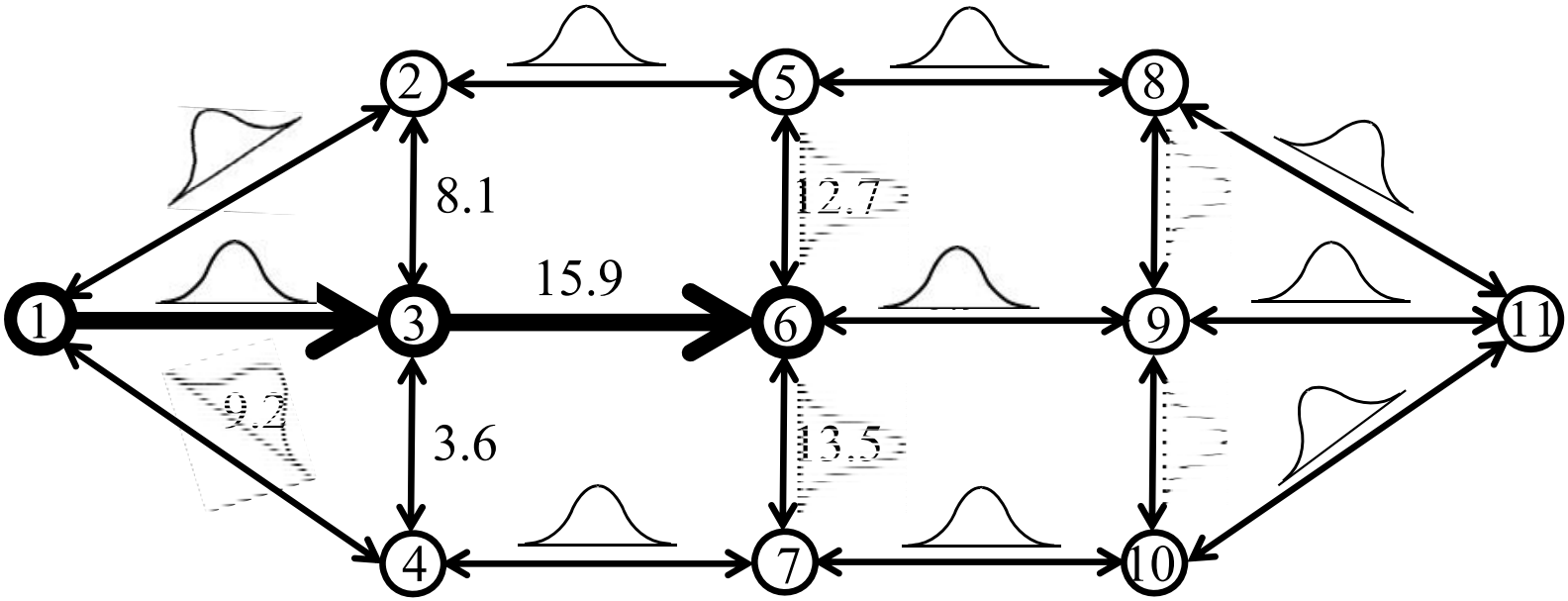
Basic model

- Problem variations
  - » Deterministic base model
    - We assume the real problem does not change over time.
  - » Stochastic base model
    - Use static point estimates of costs that do not change over time, but we experience actual costs as we traverse links.
    - Use estimates of costs that are updated as we traverse links. Estimates of costs evolve over time, but are not time dependent.
    - Use time-dependent estimates of costs that evolve as time passes.

- Stochastic, dynamic lookahead
  - » We could use a stochastic lookahead model. This comes in two flavors:
    - Arc costs are known after we make a decision – This is a deterministic shortest path problem that we can solve as a linear program or a classical deterministic shortest path problem.
    - Arc costs are known before we make a decision – Now we would have to use approximate dynamic programming to solve the lookahead model. This then has to be re-optimized as the mean arc costs are updated.
  - » We are going to design a policy where we traverse the network repeatedly, sampling arc costs as we proceed.

The state variable

- A stochastic network, costs revealed as we arrive to a node:



- Modeling:
  - » State variable (iteration n, time t)
    - $R_t^n = current\ node = i_t^n$ during pass n
    - $I_t^n = information = \left(\bar{c}_{tij}^n\right) = estimates\ of\ costs\ at\ time\ t$ during pass n
    - $S_t^n = (R_t^n, I_t^n)$
  - » Decisions

    $x_{t,i_t^n,j}$=1 if we traverse , (i_t^n, j) at time t.
    - We want a policy $X_t^\pi(S_t) = \left(x_{t,i_t^n=i,j}^\pi\right)_{ij}$ for all links i, j.
  - » Costs
    - $\hat{c}_{tij}$ =Actual realization of costs at time t to traverse (i, j)

- Modeling:
  - » Objective function
    - Cost per period

$$\mathrm{C}(\mathrm{S_t}, \mathrm{X_t^\pi}(S_t)) = (X_t^\pi\ (S_t))^T \hat{c}_t$$

$$= \sum_{i,j} x^\pi_{t,i^n_t,j}\ \hat{c}_{tij}$$

=Costs incurred at time t.

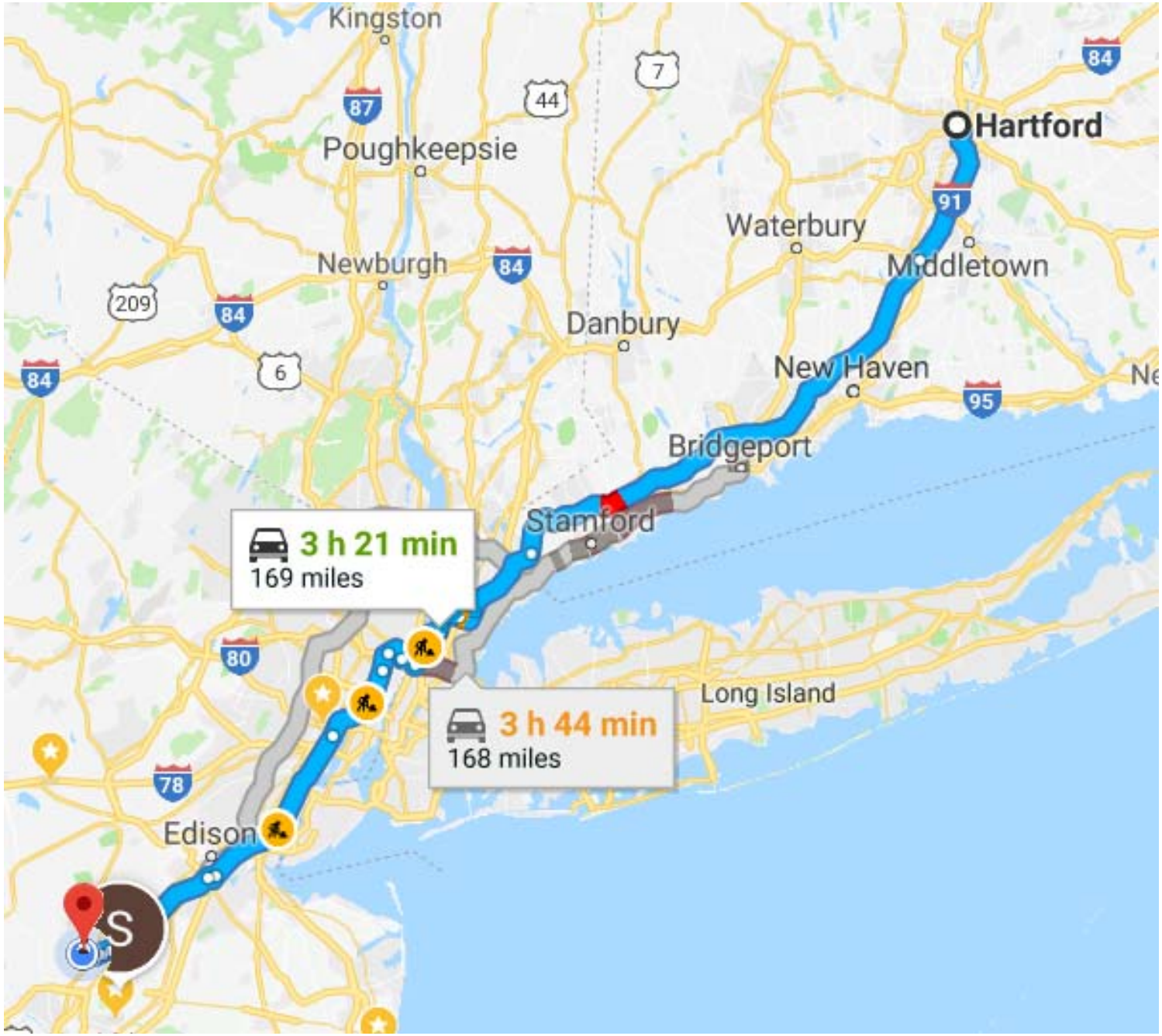
    - Total costs:

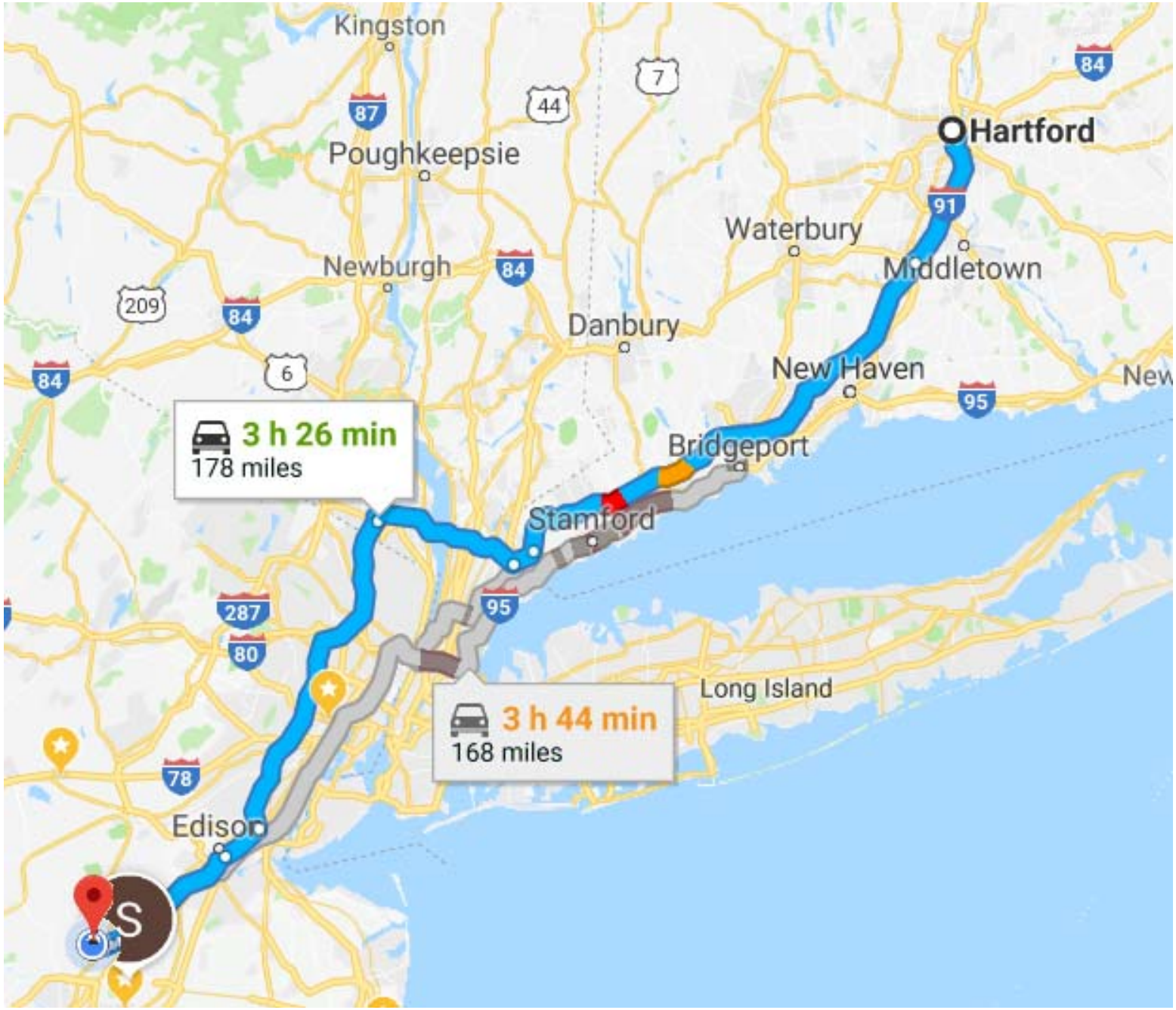
$$\min_\pi \mathbb{E} \sum_{t=0}^{T} \mathrm{C}(\mathrm{S_t}, \mathrm{X_t^\pi}(S_t))$$

  - » This is the base model.

- A policy based on a lookahead model
  - » At each time t we are going to optimize over an estimate of the network that we are going to call the *lookahead model*.
  - » Notation: all variables in the lookahead model have tilde's, and two time indices.
    - First time index, t, is the time at which we are making a decision. This determines the information content of all parameters (e.g. costs) and decisions.
    - A second time index, t', is the time within the lookahead model.
  - » Decisions
    - $\tilde{x}_{tij} = 1$ if we plan on traversing link (i, j) in the lookahead model.
    - $\tilde{c}_{tij} =$ Estimated cost at time t of traversing link (i, j) in the lookahead model.

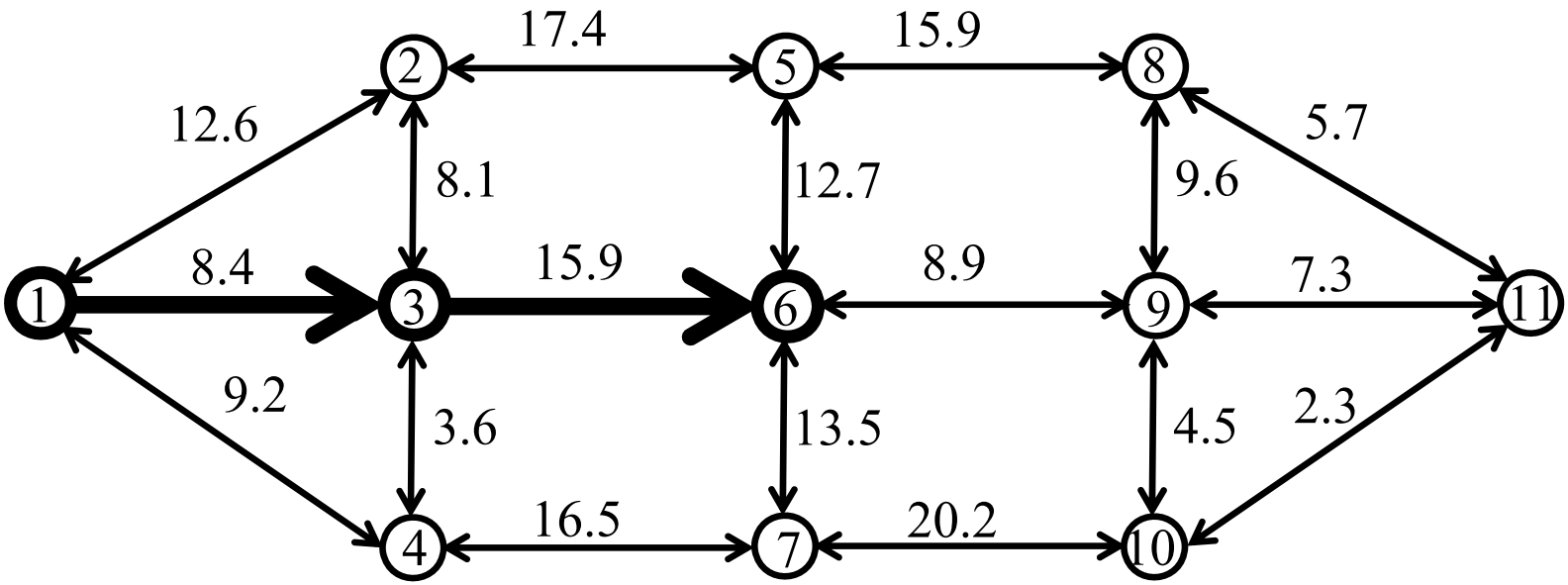
- A policy based on a lookahead model
  - » Static lookahead problem (one set of costs for the entire network.
    - $\min_x \sum_{ij} \tilde{c}_{tij} \tilde{x}_{tij}$
  - » Dynamic lookahead problem (costs depend on the time that we arrive at the link):
    - $\min_x \sum_{t'=t}^{T} \sum_{ij} \tilde{c}_{t,t',ij} \tilde{x}_{t,t',ij}$


Kingston
87
44
7
84
Poughkeepsie
Hartford
91
Waterbury
Middletown
Newburgh
84
209
84
Danbury
84
6
New Haven
95
Bridgeport
Stamford
3 h 21 min
169 miles
80
Long Island
3 h 44 min
168 miles
78
Edison
S


Kingston
87
44
7
84
Hartford
Poughkeepsie
91
Waterbury
Middletown
Newburgh
84
209
84
Danbury
84
6
New Haven
95
3 h 26 min
178 miles
Bridgeport
Stamford
287
95
80
Long Island
3 h 44 min
168 miles
78
Edison
S

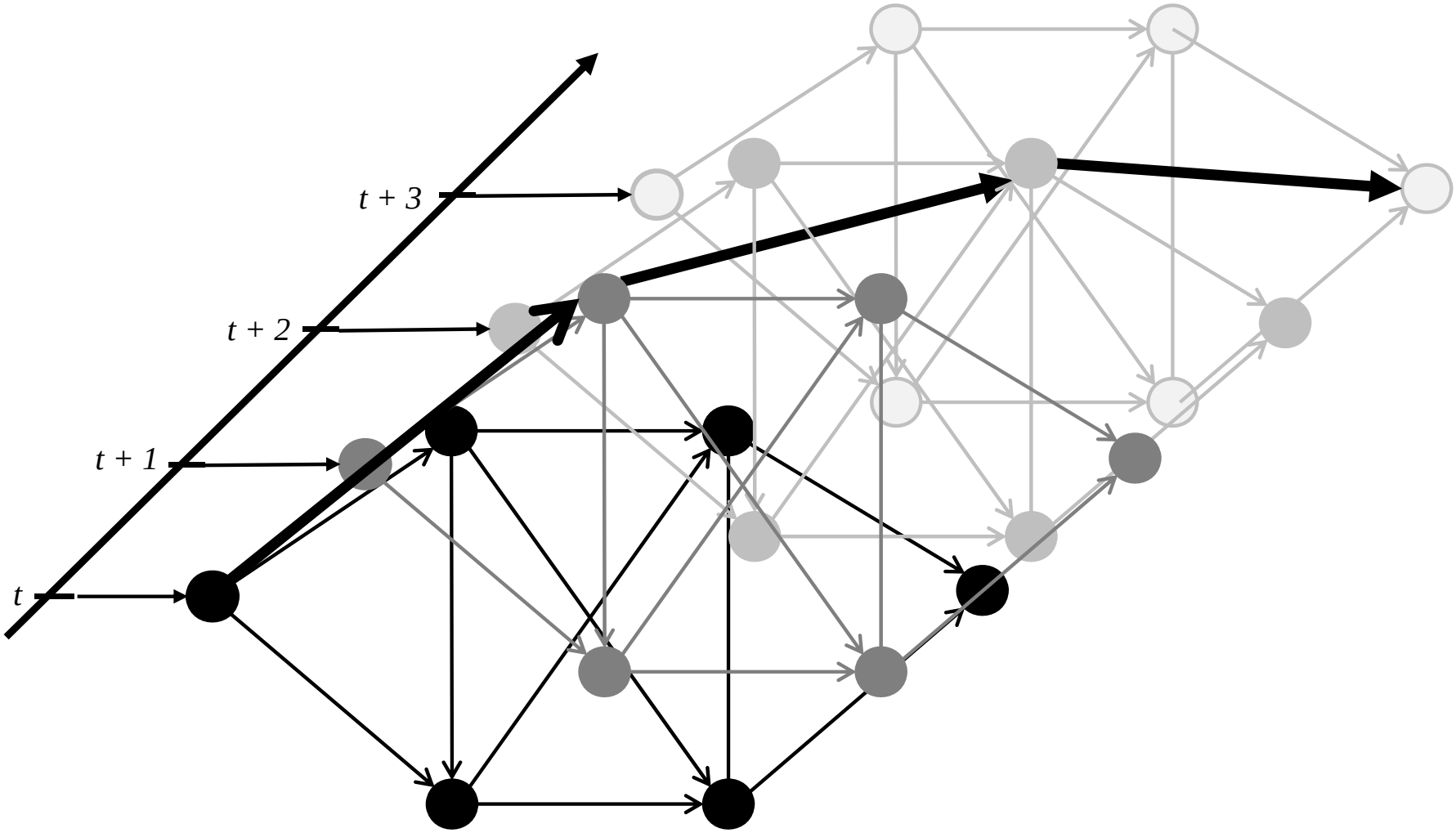
The state variable

- A static, deterministic network



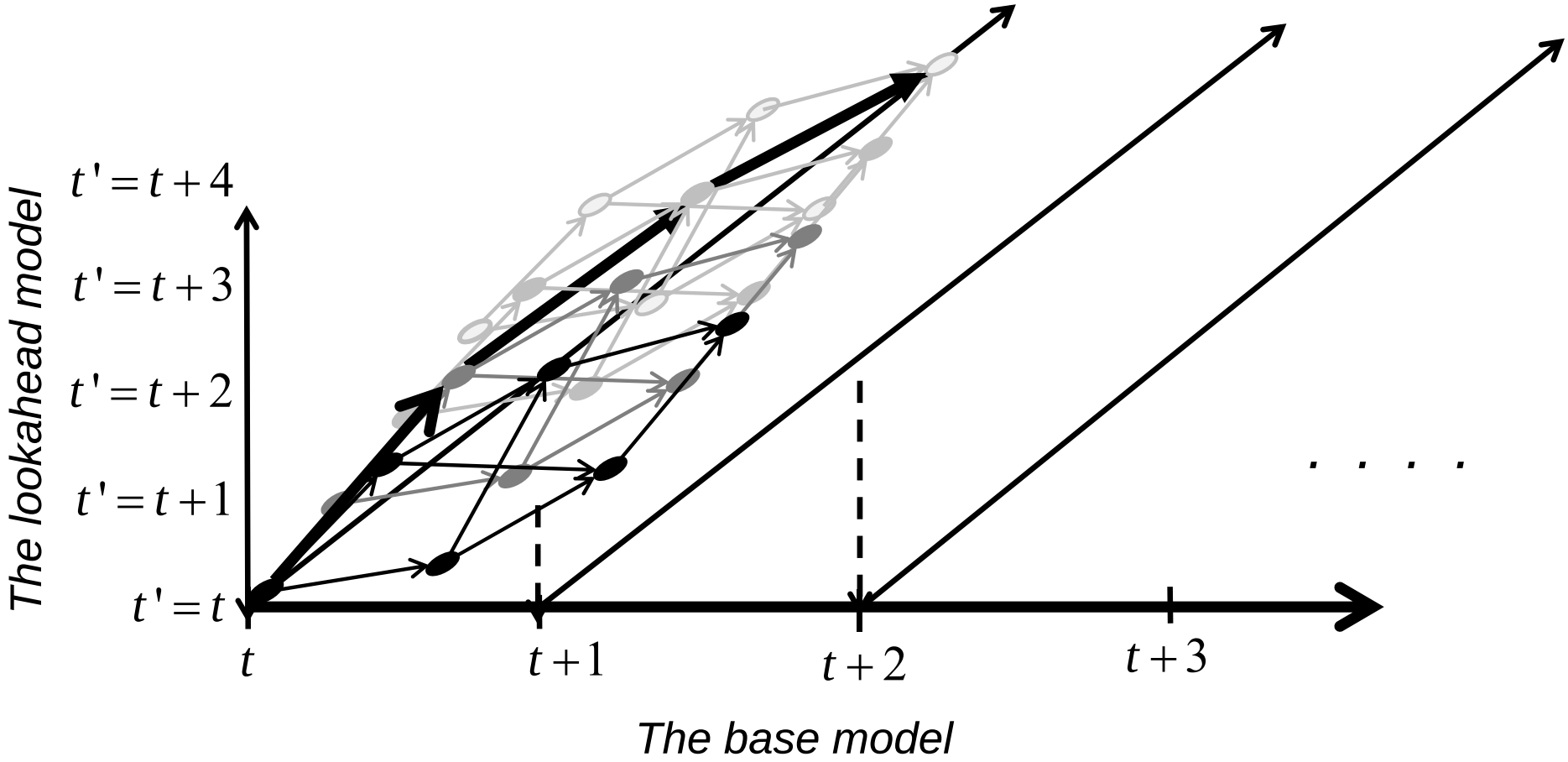
Time dependent representation

- A time-dependent, deterministic network



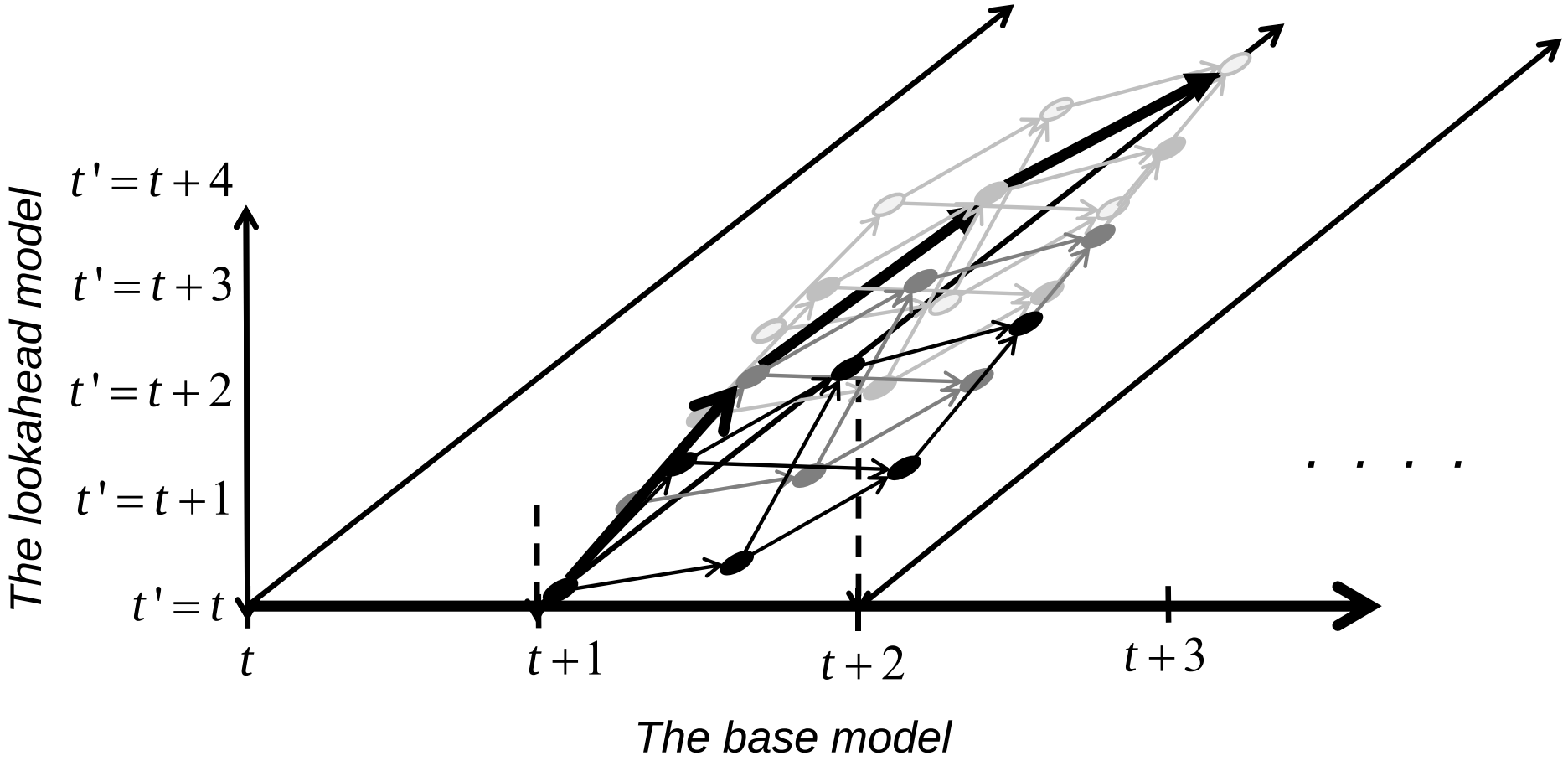
Time dependent representation

- A time-dependent, deterministic lookahead network



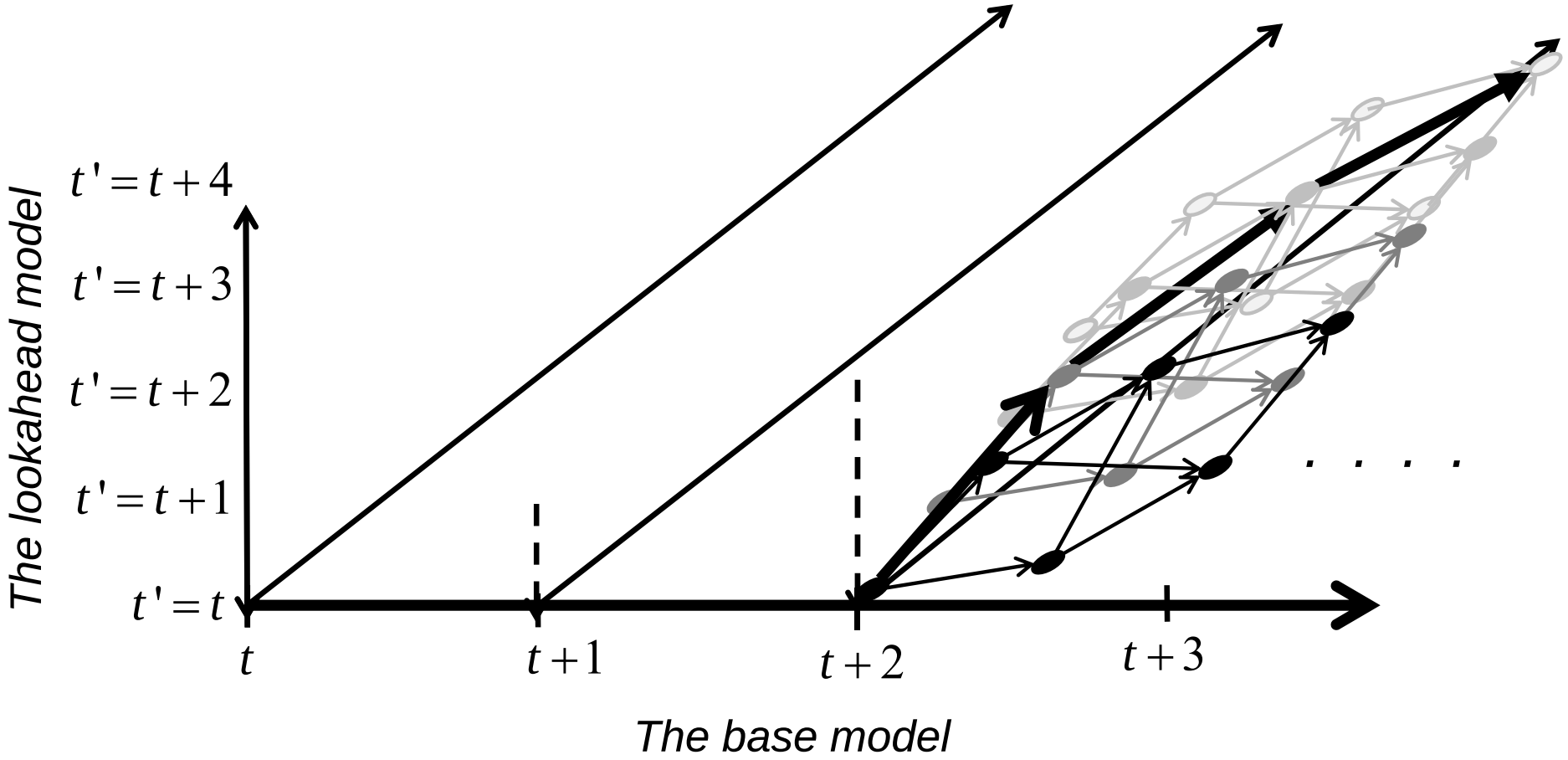
Time dependent representation

- A time-dependent, deterministic lookahead network



Time dependent representation

- A time-dependent, deterministic lookahead network



- Notes:
  - » I suspect that google maps uses a static lookahead, since these can be solved much more quickly.
    - This means if you are traveling from Connecticut to Princeton, passing New York City, leaving at 3pm, the algorithm will use estimates of going through New York at 3pm, rather than at 5pm.
  - » Deterministic, dynamic networks are *much* larger, since you have a cost for every link, for every point in time.
  - » Stochastic, dynamic networks are even harder than deterministic, dynamic networks.

Forecasts of travel times

Assume that when we have to make a decision at time t, we have an updated estimate of travel costs based on *current* congestion levels (by tracking the speed at which our smartphones are moving through traffic). We are going to represent these times using

$\tilde{c}_{tt',k\ell}$ = The estimated cost of traversing link (k,ℓ) when we arrive to the link at "time" t', using estimates based on what we know at time t.

» Evolution of costs:

We might detect slower travel times at a link we might traverse in the future, from which we might infer slower travel times over a period of time since it can take time for congestion to dissipate. For this reason, we receive a series of updates for the cost of traversing a link (k,ℓ) at a time t' that can we represent using

$$\left(\tilde{c}_{tt',k\ell}, \tilde{c}_{t+1,t',k\ell}, \tilde{c}_{t+2,t',k\ell}, \cdots, \tilde{c}_{t'-1,t',k\ell}, \tilde{c}_{t',t',k\ell}\right).$$

Static lookahead model In a static lookahead model, we use our best estimate of the time for a link (k, ℓ) as $\bar{c}_{tk\ell} = \tilde{c}_{tt',k\ell}$. We then solve Bellman's equation as

$$\widetilde{V}_{tt'}(j) = \min_{(\tilde{x}_{tt'jk}, k \in \mathcal{N}_j^+)} \left(\sum_{k \in \mathcal{N}_j^+} \bar{c}_{tjk} \tilde{x}_{tt'jk} + \widetilde{V}_{t,t'+1}(k) \right). \quad (5.22)$$

The variables $\tilde{x}_{tjk}$ represent the decisions in this static model using the information as we know it at time t. Once we have our value functions $\widetilde{V}_{tt'}(j)$ for all nodes j, we can compute the optimal decision we would make when we are at node j, which we designate $\tilde{x}^*_{tt'}(j)$ using

$$\tilde{x}^*_{tt'}(j) = \arg\min_{(\tilde{x}_{tt'jk}, k \in \mathcal{N}_j^+)} \left(\sum_{k \in \mathcal{N}_j^+} \bar{c}_{tjk} \tilde{x}_{t,t'+1,jk} + \widetilde{V}_{t,t'+1}(k) \right). \quad (5.23)$$

We run the algorithm by starting at a time $t' = t + H$ for an appropriate horizon H, where H is large enough to ensure that we can get to our destination in H time periods. We then execute equation (5.22) stepping backward in time Since we may

» Can also think of this as a linear program.

Deterministic dynamic model Now assume that we are given cost estimates $\tilde{c}_{tt',k\ell}$ that are projections of the cost that we think we will incur when we traverse link (k,ℓ) at time t' in the future.

$$\widetilde{V}_{tt'}(j) = \min_{(\tilde{x}_{tt',jk},k\in\mathcal{N}_j^+)} \left(\sum_{k\in\mathcal{N}_j^+} \tilde{c}_{tt',jk}\tilde{x}_{tt',jk} + \widetilde{V}_{t,t'+1}(k) \right). \quad (5.24)$$

This is a deterministic model of the future. Although we are trying to solve a stochastic shortest path problem, we are approximating the lookahead model using deterministic estimates of costs (just as we did with the static model).

This works just like the static version, except that now we index the value function by time t'. It is important to recognize that t' is the true time variable in this model, since this is the time at which activities are happening within our model. The index t is just capturing the fact that we are solving the problem at time t, using information as we know it at time t. Since we are currently at node i at time t, our decision would be

$$\tilde{x}^*_{tt'}(i) = \arg\min_{(\tilde{x}_{tt'ij},j\in\mathcal{N}_i^+)} \left(\sum_{j\in\mathcal{N}_i^+} \tilde{c}_{tt'ij}\tilde{x}_{tt'ij} + \widetilde{V}_{t,t'+1}(j) \right). \quad (5.25)$$

- Imagine that the lookahead is just a black box:
  - » Solve the optimization problem

$$X_t^{\pi}(S_t^n) = \arg\min \sum_{i \in N} \sum_{j \in N_i^+} \tilde{c}_{tij}^n \tilde{x}_{tij}$$

  - » subject to

$$\sum_j \tilde{x}_{i_t^n, j} = 1 \quad \text{Flow out of current node where we are located}$$

$$\sum_i \tilde{x}_{i,r} = 1 \quad \text{Flow into destination node } r$$

$$\sum_i \tilde{x}_{i,j} - \sum_k \tilde{x}_{j,k} = 0 \quad \text{for all other nodes.}$$

  - » This is a deterministic shortest path problem that we could solve using Bellman's equation, but for now we will just view it as a black box optimization problem.

Simulating a lookahead policy

We would like to compute

$$F^{\pi} = \mathbb{E}\sum_{t=0}^{T}\sum_{i,j} X^{\pi}_{t,ij}(S_t)\hat{c}_{t,ij}$$

but this is intractable.

Let ω be a sample realization of costs

$$\hat{c}_{t,t',ij}(\omega), \hat{c}_{t+1,t',ij}(\omega), \hat{c}_{t+2,t',ij}(\omega), \ldots$$

Now simulate the policy

$$\hat{F}^{\pi}(\omega^n) = \sum_{t=0}^{T}\sum_{i,j} X^{\pi}_{t,ij}(S_t(\omega^n))\hat{c}_{t,ij}(\omega^n)$$

Talk through how this works.

Finally, get the average performance

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

- Discuss:
  - » Make distinction between sampled costs $\hat{c}_{tij}$ and forecasted costs $\bar{c}_{tij}$
  - » Both are updated with time and from one iteration to another.

Policy for stochastic lookahead

- Solving a stochastic lookahead
  - » Assume that the travel gets to see the costs out of a node once she arrives at the node.
  - » Further assume that we have a forecast, but model the uncertainty around the forecast.
  - » We would have to us approximate dynamic programming to solve the stochastic shortest path problem.
  - » This is computationally very cumbersome – unlikely to ever be used in practice!

- Notes:
  - » The deterministic lookahead is still a policy for a stochastic problem.
  - » Can we make it better?
- Idea:
  - » Instead of using the expected cost, what about using a percentile.
  - » Use pdf of $\hat{c}_{ij}$ to find θ percentile (e.g. $\theta = .8$). Let

$$\tilde{c}_{ij}^{p}(\theta) = \text{The } \theta - \text{percentile of } \hat{c}_{ij}$$

  - » Which means $Prob\left[\hat{c}_{ij} \leq \tilde{c}_{ij}^{p}(\theta)\right] = \theta.$

- The θ −percentile policy.
  - » Solve the linear program (shortest path problem):

$$X_t^\pi(S_t^n \mid \theta) = \arg\min \sum_{i \in N} \sum_{j \in N_i^+} \tilde{c}_{tij}^{\,p}(\theta)\tilde{x}_{tij} \quad \text{(Vector with } x_{tij} = 1 \text{ if decision is to take } (i, j)\text{)}$$

  - » subject to

$$\sum_j \tilde{x}_{t,i_t^n,j} = 1 \quad \text{Flow out of current node where we are located}$$

$$\sum_i \tilde{x}_{tir} = 1 \quad \text{Flow into destination node } r$$

$$\sum_i \tilde{x}_{tij} - \sum_k \tilde{x}_{tjk} = 0 \quad \text{for all other nodes.}$$

  - » This is a deterministic shortest path problem that we could solve using Bellman's equation, but for now we will just view it as a black box optimization problem.

- Simulating a lookahead policy

  Let ω be a sample realization of costs

  $$\hat{c}_{t,t',ij}(\omega), \hat{c}_{t+1,t',ij}(\omega), \hat{c}_{t+2,t',ij}(\omega), \ldots$$

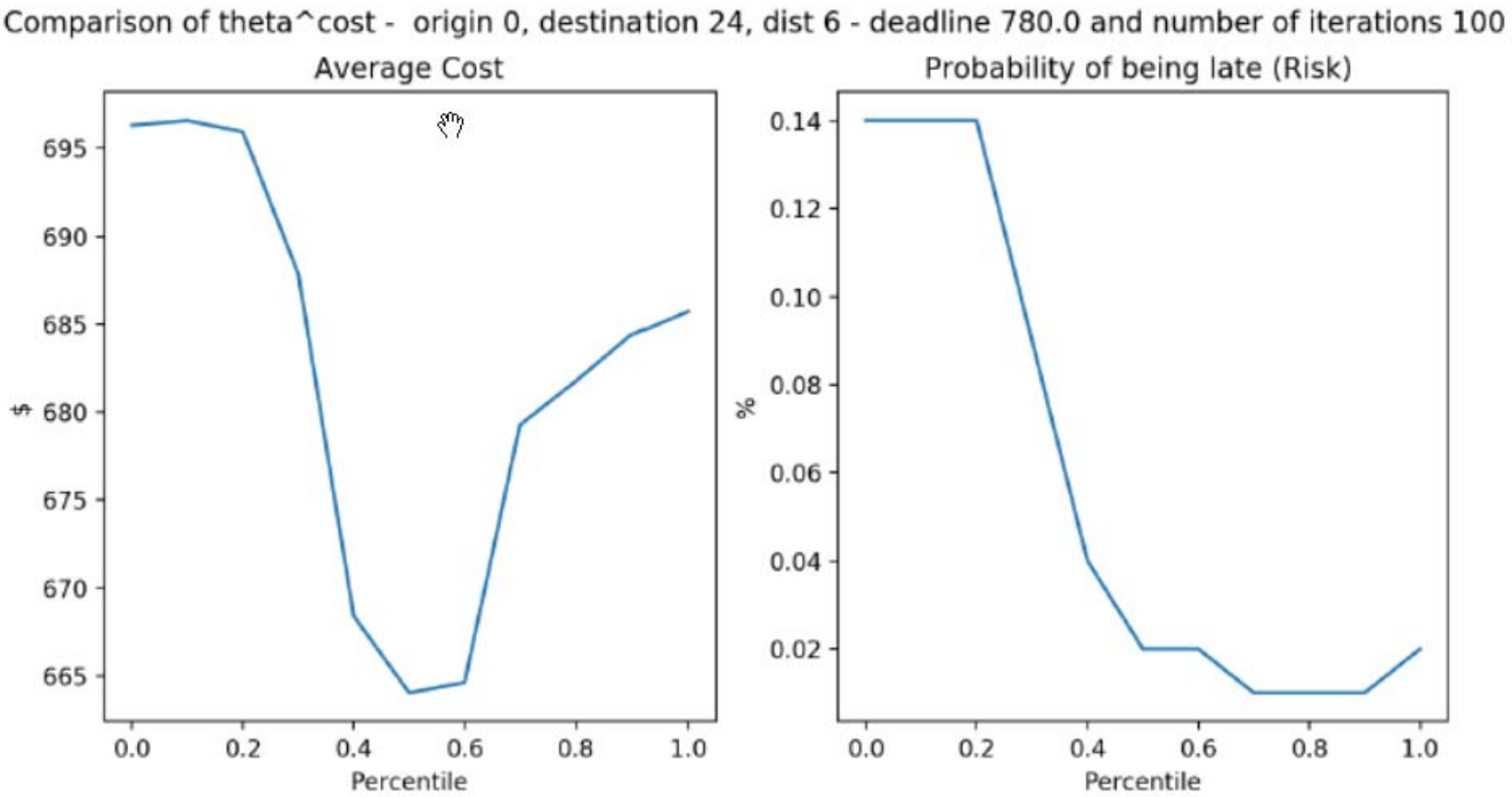
  Now simulate the policy

  $$\hat{F}^{\pi}(\omega^n) = \sum_{t=0}^{T} \sum_{i,j} \hat{c}_{t,t',ij}(\omega) X_t^{\pi}(S_t(\omega^n) \mid \theta)$$

  Finally, get the average performance

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

- Policy tuning
  - » Cost vs. lateness (risk)



Week 4 - Wednesday

Modeling

Modeling frameworks

State variables

The state variables

- What is a state variable?
  - » Surprisingly, the academic community has generally avoided defining a state variable.
  - » Bellman's classic text on dynamic programming (1957) describes the state variable with:
    - "… we have a physical system characterized at any stage by a small set of parameters, the *state variables*."
  - » The most popular book on dynamic programming (Puterman, 2005, p.18) "defines" a state variable with the following sentence:
    - "At each decision epoch, the system occupies a *state*."
  - » Needless to say, these are not "definitions" in any formal sense of the word.

The state variables

- What is a state variable?
  - » Wikipedia:
    - "State commonly refers to either the present condition of a system or entity" or….
    - A state variable is one of the set of variables that are used to describe the mathematical 'state' of a dynamical system
  - » Kirk (2004), an introduction to control theory, offers the definition:
    - A state variable is a set of quantities $x_1(t), x_2(t), \ldots$ which if known at time $t = t_0$ are determined for $t \geq t_0$ by specifying the inputs for the system for $t \geq t_0$.
    - True, but too vague to be useful.
  - » What was the state variable for our cash balance problem?

The state variables

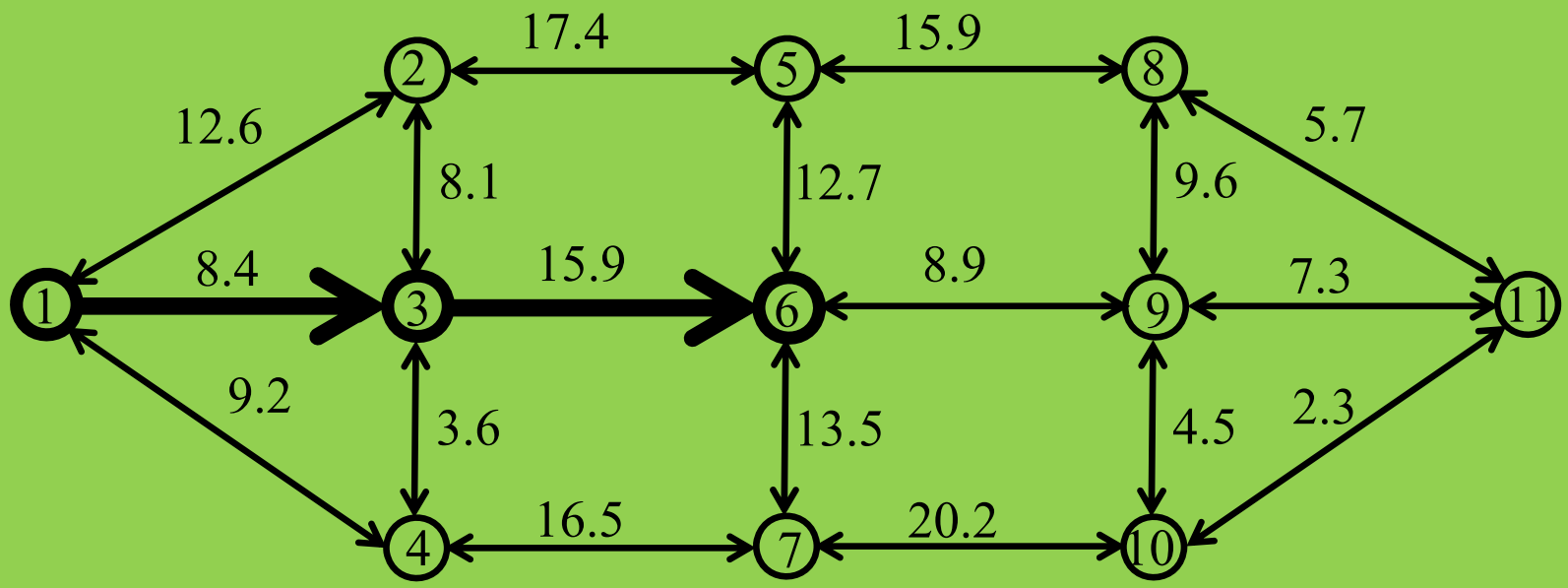
- Dimensions of a state variable:
  - » I often find it useful to use three perspectives of a state variable:
    - Physical state R_t
      - – This is a snapshot of the state of the physical system at a point in time
    - Information state I_t
      - – This is any information needed that is not in the physical state that we need to model the system
    - Belief/knowledge state K_t
      - – This captures what we believe (in the form of probability distributions) about unobservable parameters.
  - » The issue of Markovian vs. history dependent systems arises when people equate state with physical state. Different communities handle this issue differently.

The state variables

- The physical state
  - » What is the price of the stock (right now)?
  - » At what node are you located on a graph?
  - » How much inventory do you have in stock?
  - » These are all variables indexed by *t*.
- The information state
  - » We use this to include all the information other than what is in the physical state:
    - Your model that forecasts future weather, demands, stock prices.
    - Any history (variables index by t-1, t-2, …) needed to make decisions, compute costs or model the future.
- The belief state
  - » This consists of probability distributions about unknown parameters
    - How many books will I sell at price *p*?
    - What is the average global temperature?
    - How will a patient respond to a particular medication?
  - » Each of these are unknown parameters. We might assume that we model our belief using normal distributions. Our belief state would include the normality assumption along with means and variances.

The state variable

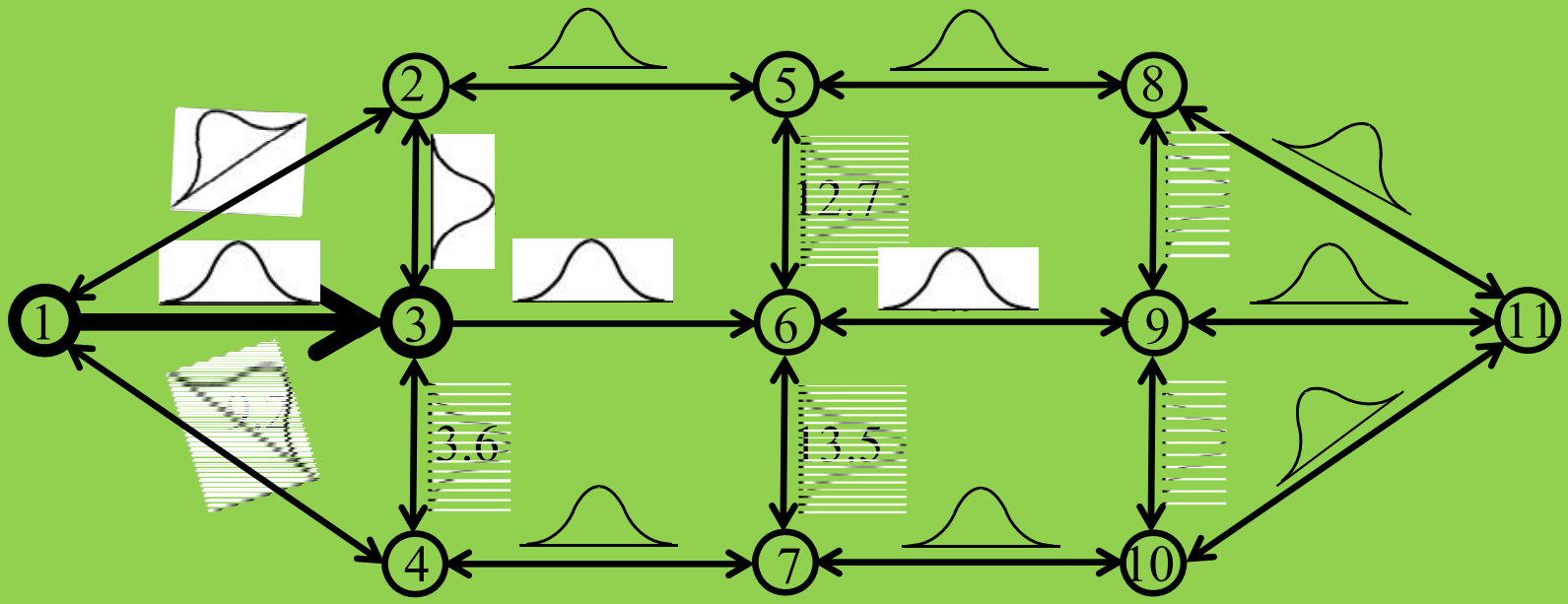
- Illustrating state variables
  - » A deterministic graph



$$S_t = (N_t) = 6$$

The state variable

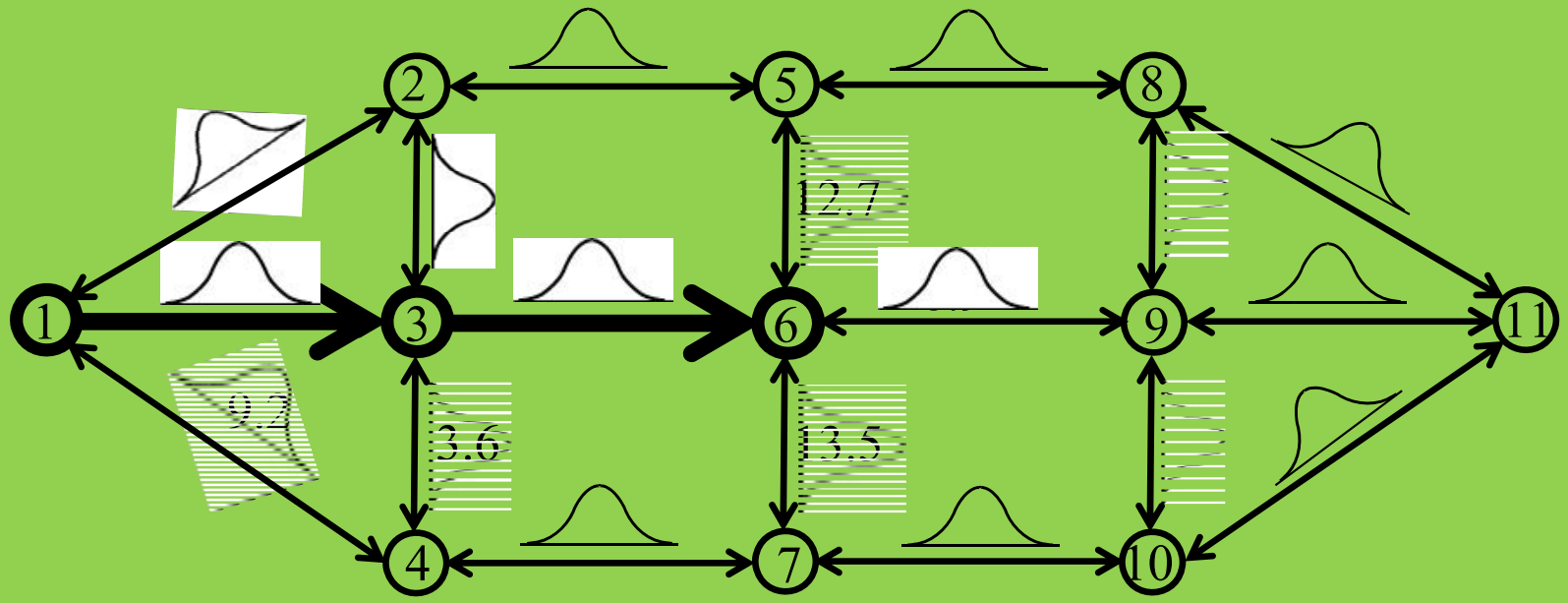
- Illustrating state variables
  - » A stochastic graph



$$S_t = ?$$

The state variable

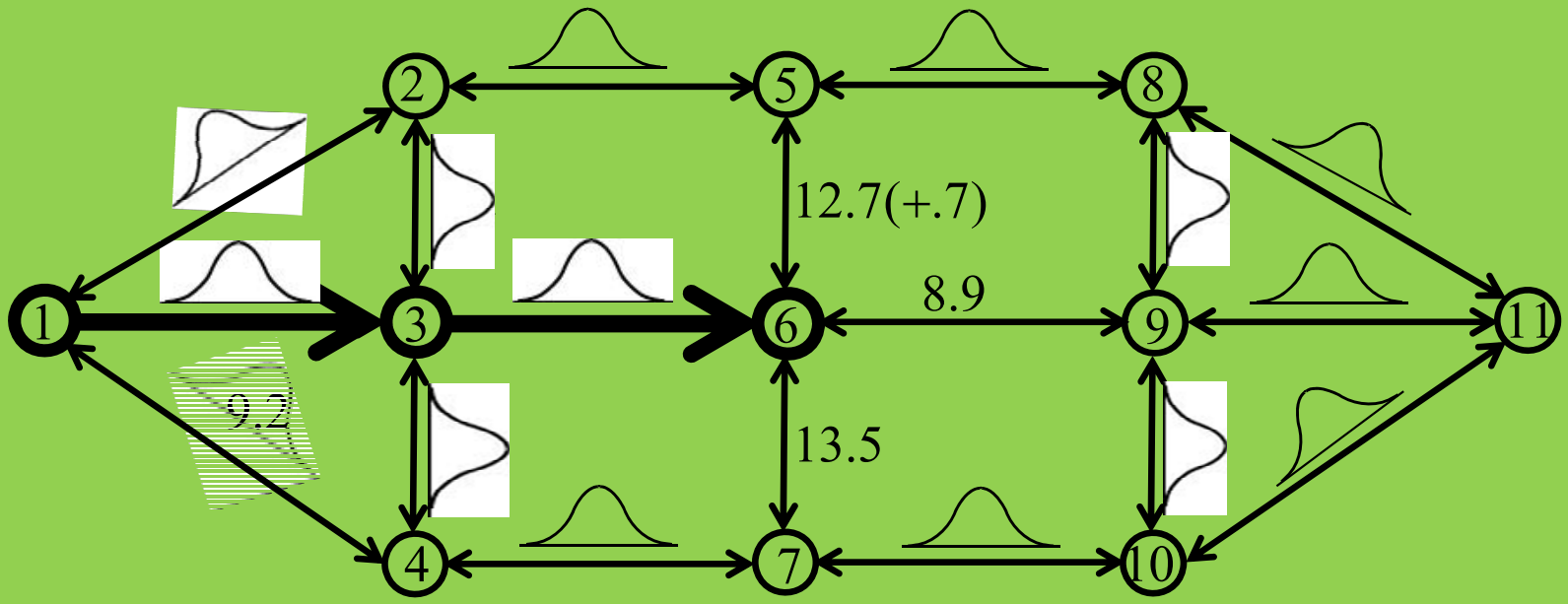
- Illustrating state variables
  - » A stochastic graph



$$S_t = \Big(\underbrace{N_t}_{R_t}, \underbrace{\big(c_{t,N_t,j}\big)_j}_{I_t} \Big) = \big(6, (12.7, 8.9, 13.5)\big)$$

The state variable

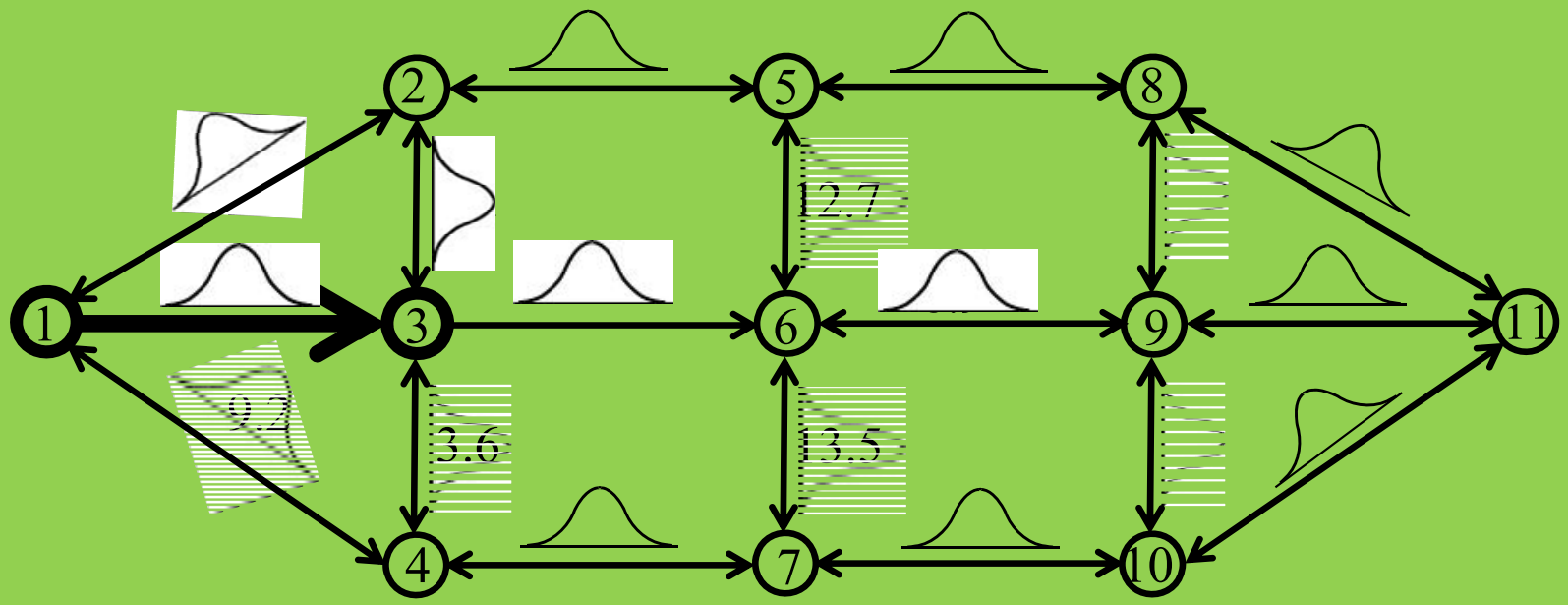
- Illustrating state variables
  - » A stochastic graph with left turn penalties



$$S_t = \Big(\underbrace{N_t}_{R_t}, \underbrace{\big(c_{t,N_t,j}\big)_j, N_{t-1}}_{I_t} \Big) = \big(6, (12,7,8.9,13.5), 3\big)$$

The state variable

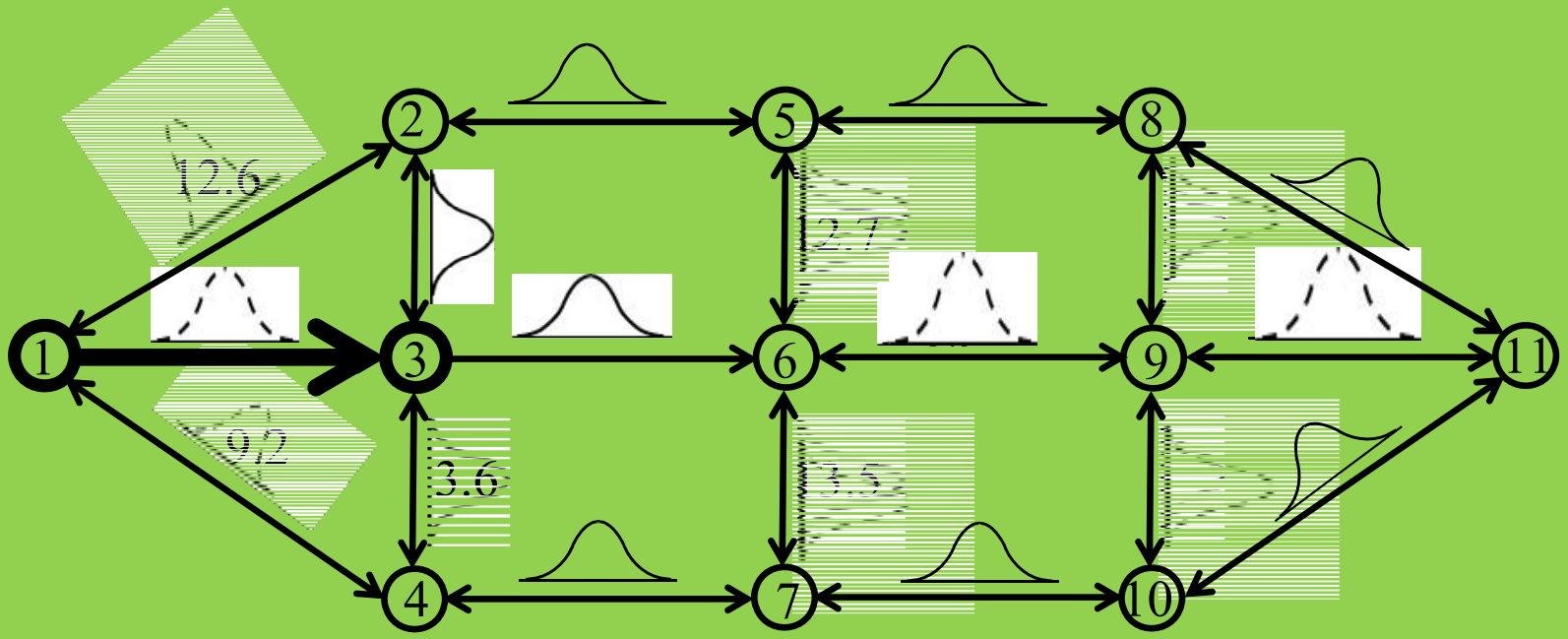
- Illustrating state variables
  - » A stochastic graph with generalized learning



$$S_t = ?$$

The state variable

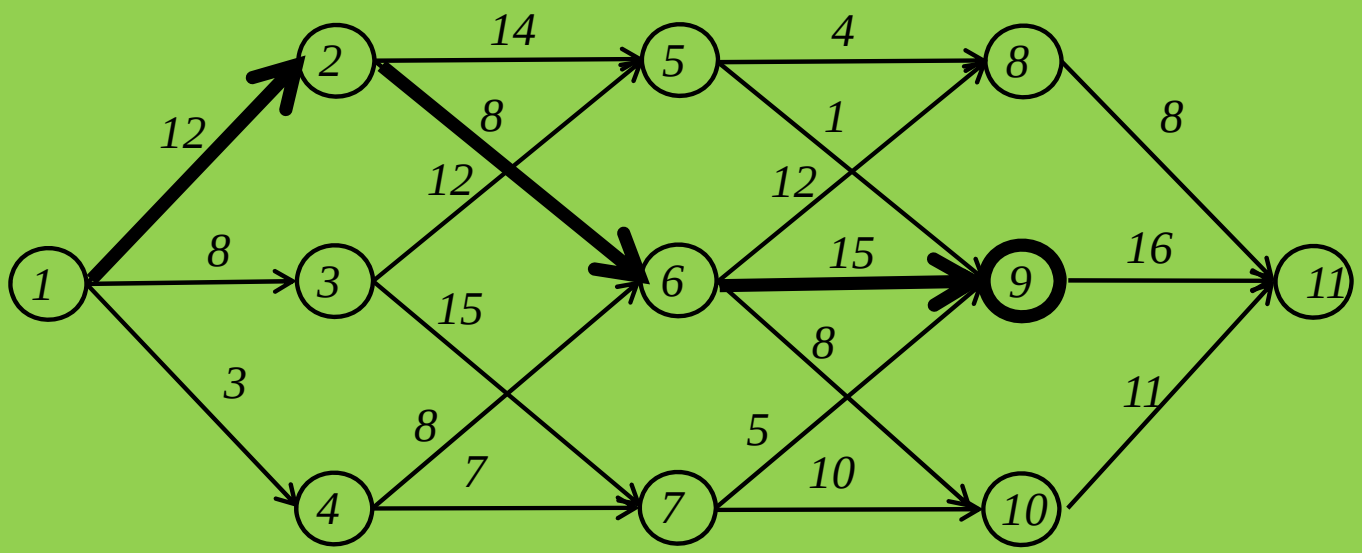
- Illustrating state variables
  - » A stochastic graph with generalized learning



$$S_t = \Big(\underbrace{N_t}_{R_t}, \underbrace{\big(c_{t,N_t,j}\big)_j}_{I_t}, \underbrace{\quad}_{K_t} \Big)$$

The state variable

- Variant of problem in Puterman (2005):
  - » Find best path from 1 to 11 that minimizes the *second highest arc cost* along the path:

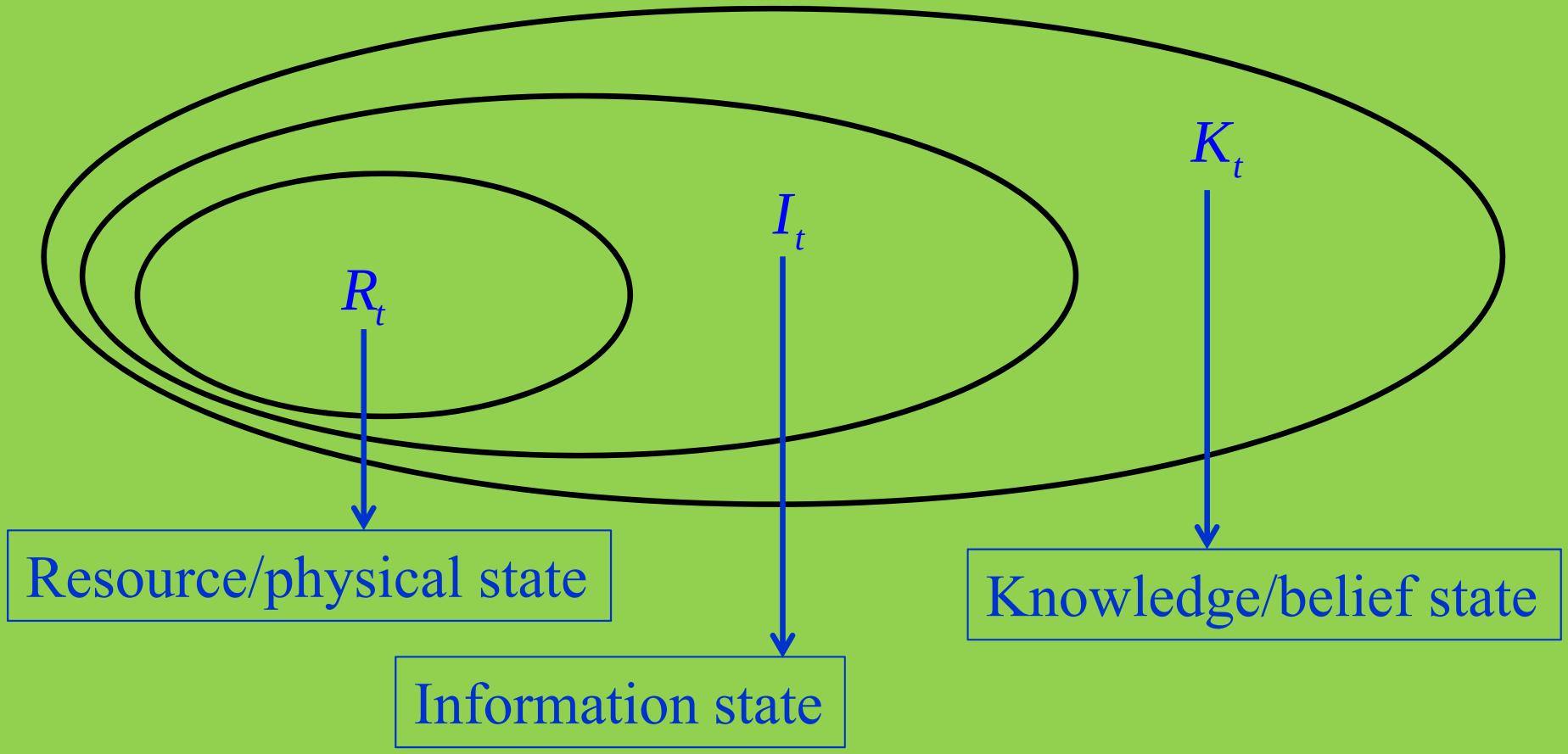


  - » If the traveler is at node 9, what is her state?

$$S_t = (N_t, \text{highest}, \text{second highest}) = (9,15,12)$$

The state variable

- Classes of state variables
  - » Technically these are nested sets….



  - » … but we treat them as if they are distinct.

The state variables

- Let's illustrate our new notation with a new optimization problem… selling a stock.

  Let p_t be the price of the stock at time t. One way to model the evolution of our stock price is using

  $$p_t = p_{t-1} + \hat{p}_t$$

  In this representation, we view $\hat{p}_t$ as the exogenous information. We would write $\hat{p}_t$ as a random variable (that is, it is random before time t), and $\hat{p}_t(\omega)$ as a sample realization when we are following sample path ω.

The state variables

- Selling a stock

  When should we sell our stock? It makes sense to sell when the price gets too much higher than a long term trend.

  Estimate the trend using

  $$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha p_t$$

  For this problem, our state variable is

  $S_t = (p_t, \bar{p}_t, R_t)$ $R_t = 1$ if we are still holding the stock, 0 otherwise

  Let

  $$X(S_t) = \begin{cases} 1 & \text{If we sell at time } t \text{ (requires } R_t = 1) \\ 0 & \text{Otherwise} \end{cases}$$

  One possible rule for selling the stock might be to use

  $$X^{\beta}(S_t) = 1 \text{ if } p_t \geq \bar{p}_t + \beta$$

  So our policy π represents both this type of rule, plus the parameter β.

The state variables

- We can use different ways for tracking prices:
  - » Instead of using the smoothing

$$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha p_t$$

  - » … we used a moving average over the last three periods:

$$\bar{p}_t = \frac{1}{3}\left(p_t + p_{t-1} + p_{t-2}\right)$$

  - » In this case, what is the state of our system?

The state variables

- Moving average pricing problem
  - » State variable

$$S_t = (p_t, p_{t-1}, p_{t-2}, R_t)$$

  - » Transition function

$$p_{t+1} = p_t + \hat{p}_{t+1}$$
$$R_{t+1} = R_t - x_t$$
$$S_{t+1} = (p_{t+1}, p_t, p_{t-1}, R_{t+1})$$

  - » Compare this state and transition with the first pricing problem using exponentially weighted prices.

The state variable

- Two definitions:

  1) *The state S_t is a function of history that, combined with the exogenous information, is necessary and sufficient to calculate cost function, transition function, and decision function (policy) from time t onward.*
  2) *The state S_t is a function of history that, combined with the exogenous information, is necessary and sufficient to calculate costs, constraints and transitions, from time t onward.*

  » Using these definitions, ***all properly modeled problems are Markovian!***

The state variables

- There are two types of state variables
  - » The initial state S_0 - This includes:
    - Static information that does not vary over time
      - The tax rate on long term capital gains.
      - The amount of energy lost when being stored in a battery
      - The speed of an aircraft
    - Probabilistic information about unobservable parameters
      - Distribution of the price of a stock in 6 months
      - Distribution of the possible functions relating demand to price
  - » Dynamic information that varies over time - $S_t,\ t > 0$
    - The demand for a product
    - The price of a stock
  - » By convention, S_t only includes *dynamic information*. Static information is captured implicitly in S_0.

The state variables

- The state variable (dynamic information) comes in three flavors:
  - » Endogenously controllable
    - The cash in a mutual fund
    - The water in a reservoir (which can include exogenous rainfall)
  - » Exogenous information
    - The weather
    - The price of a stock (assuming I cannot influence the market)
    - The number of patients needing blood
  - » Exogenous information that can be endogenously influenced
    - The price of a stock if I am a big player
    - The rate of new HIV infections (if I am working on policies to reduce transmission)

The state variables

- To illustrate
  - » Consider our pricing problem where our decision uses the three most recent prices. The state variable is:

$$S_t = (p_t, p_{t-1}, p_{t-2}, R_t)$$

  - » So what if we chose to write it as:

$$S_t = (p_t, p_{t-1}, R_t)$$

    - Or

$$S_t = (p_t, p_{t-1}, p_{t-2})$$

  - » What if we write it as

$$S_t = (p_t, p_{t-1}, p_{t-2}, p_{t-3}, R_t)$$

    - What is wrong with this? It has all the information we need.

The state variables

- An energy storage problem
  - » In the battery arbitrage problem, we can store energy in a battery from the grid when prices are low, or sell back to the grid when prices are high.
  - » Let

    p_t = Purchase price for electricity from the grid

    x_t = How much we decide to buy (>0) or sell (<0) between t and t+1.

    R_t = Amount of energy in the battery at time t

    $R^{\max}$ = Capacity of our battery
  - » We might make decisions using

$$x_t = \arg\max_{-R_t \le x \le R^{\max} - R_t} \left(-p_t x + V_{t+1}(R_t + x_t)\right)$$

    Our decision depends on p_t and R_t.

Modeling frameworks

Decision variables

The decision variables

- There are three common notational systems for decisions:

Computer science

a_t = Discrete action

Control theory

u_t = Low-dimensional continuous vector

Operations research

x_t = Usually a discrete or continuous but high-dimensional vector of decisions.

The decision variables

- Notes
  - » It is extremely important that making a decision at time t can only use information in the state S_t at time t.
- A definition of a "decision"
  - » An *endogenously controllable* information class.
- How do we make decisions?
  - » We use *policies*, which are rules for making decisions. We will use notation such as:

$A^{\pi}(s)$ = The policy for determining an action a

$U^{\pi}(s)$ = The policy for determining a control u

$X^{\pi}(s)$ = The policy for determining a decision x

Here, π is a label that determines the type of function.

The decision variables

- Defining decisions
  - » In some settings (e.g. finance), it is relatively easy to identify decisions:
    - What stock to buy?
    - How much to allocate into different assets?
    - What price to sell a stock?
  - » In more complex settings (business, policy), it is easier to identify goals (improve cost/service, improve health coverage), but it is not always easy to identify decisions.
    - How should Amazon improve its profits?
    - How should we increase health coverage?

  We know the goals, but we do not know how…

The decision variables

- Take a look at the decisions spreadsheet you compiled over the summer:

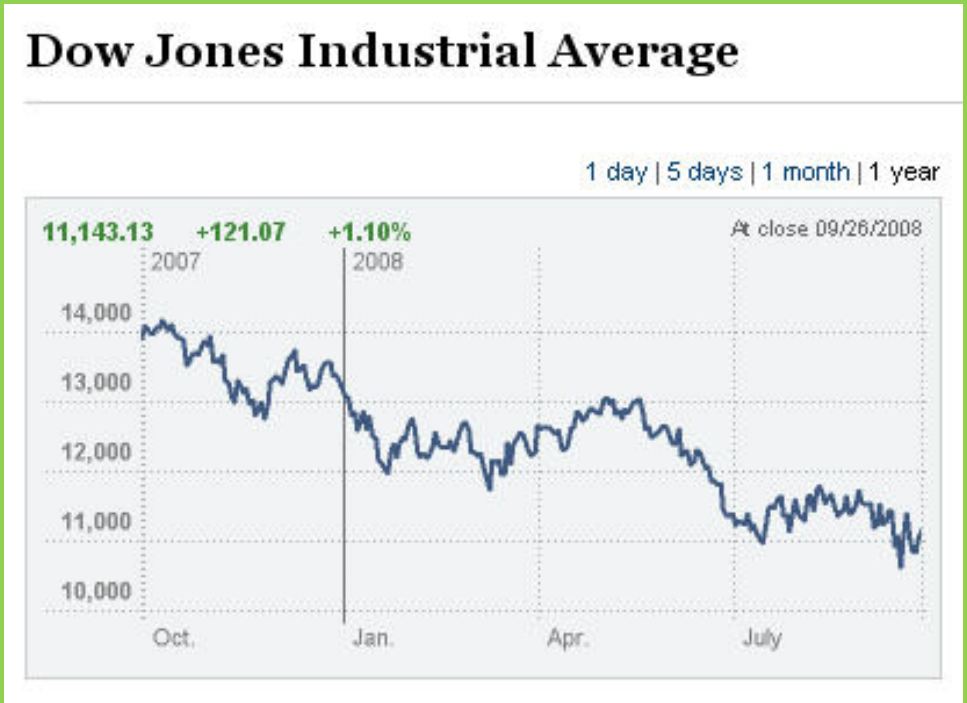
  » Click here for sheet.

Modeling frameworks

Exogenous information

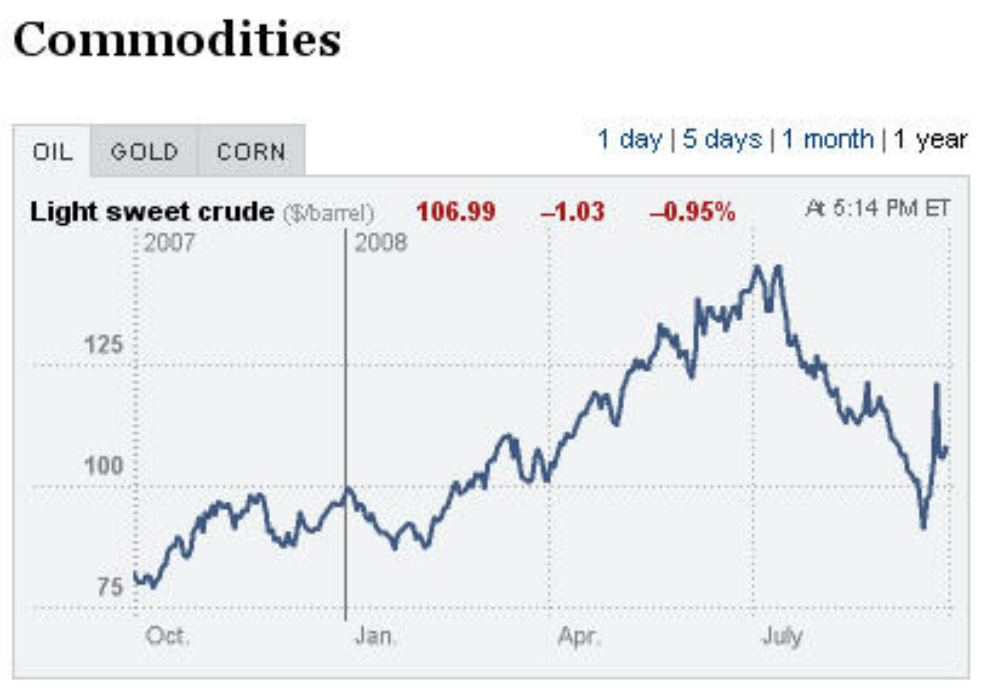
Exogenous information

- Guessing what is happening in the real world
  - » Is the stock market going (further) down?
  - » What do you think?



Exogenous information

- Guessing what is happening in the real world
  - » What about the price of the barrel of crude oil?



Exogenous information

Often, we need to estimate what will happen in the future to make a decision now. Let:

D_t = Our *forecast* of the customer demand in future time period t.

p_t = Our *forecast* of the market price at time t.

The future looks like:

$$\{(D_1, p_1),(D_2, p_2),...,(D_t, p_t)\}$$

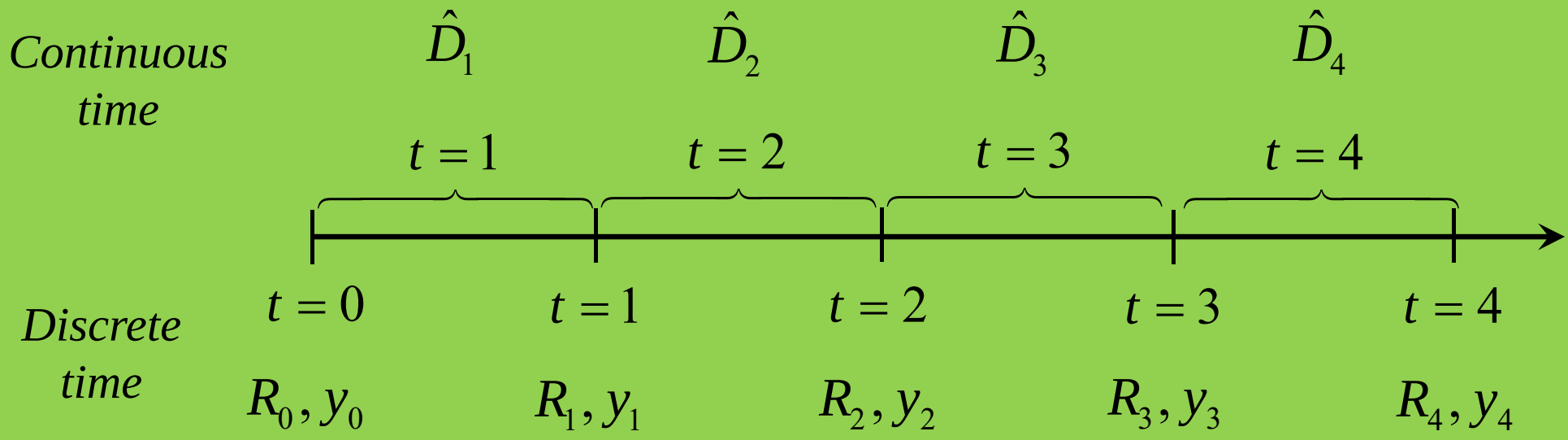
We say that (D_t, p_t) is the *information* arriving at time t. It is sometimes useful to have a single variable to represent the new information arriving to the system at time t. There is not standard notation for modeling information. Some people let:

ω_t = The information arriving in time t.

ω_t represents a realization of the information that arrives in time period t.

Exogenous information

- We need a system for indexing time. In particular, it is important to know the mapping between discrete and continuous time.



It is useful to think of information as arriving continuously over time. Functions (states, decisions) are measured at a point in time.

At time t, anything t' <= t is known, anything t' > t is unknown.

Exogenous information

All the information arriving would then be:

$$\omega = (\omega_1, \omega_2, ..., \omega_t, ...)$$

ω_t is not a random variable (although some people treat it as one). We sometimes need a random variable which is a function providing the information that *might* arrive during time period t. If we do not have a specific variable such as a price, quantity or demand, we can use the generic notation:

W_t = The information arriving in time t.

We use W_t because it "looks like" ω_t. We can also write:

$$\omega_t = W_t(\omega)$$

Exogenous information

For our example, we would write:

$$W_t = (D_t, p_t)$$

In the future, we do not know what might happen. Assume that the only type of new information arriving is customer demands. That is, $W_t = (D_t)$.

Assume that there are only 10 possible sets of demands that might happen in the future:

ω	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9
1	18	16	13	10	17	6	4	15	16
2	12	7	17	15	5	3	4	14	8
3	6	18	7	9	1	13	4	4	7
4	2	11	16	16	1	2	13	0	13
5	18	5	0	6	10	17	8	3	2
6	3	18	5	20	13	16	18	11	10
7	12	14	4	11	19	3	20	19	18
8	6	15	15	14	2	7	14	1	11
9	19	10	5	19	13	14	16	11	17
10	18	15	14	4	6	17	16	10	9

It is absolutely standard notation to index these outcomes by ω (don't ask why). The set of outcomes (the sample space) is referred to as Ω. So an element $\omega \in \Omega$ refers to a particular set of potential outcomes.

Exogenous information

We would say that D_t is a random variable because we do not know the demand D_t right now. We might, for example, assume that D_t follows some probability distribution so that we can describe the range of possible outcomes.

Sometimes we need to refer to a particular realization. For this, we let:

$D_t(\omega)$ = A sample realization of the demand at time t.

$D_t(\omega)$ is not a random variable. Let's say ω=6. Then,

	ω	D_1	D_2	D_3	D_4					
$D_t(6)$ =	6	3	18	5	20	13	16	18	11	10

Exogenous information

- At this point, we have:
  - » Illustrated some actual stochastic optimization problems
  - » Reviewed the core dimensions of a problem
  - » Illustrated some policies
  - » Hinted at how you can find the best policy.

  - » But we have brushed by some concepts that deserve a little more time.

Modeling frameworks

Transition function

The transition function

- The transition function captures the evolution over time:

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

» The transition function goes by many names:

- System model
- Plant model
- Plant equation
- State equation
- Law of motion
- State equation
- Transition law
- Transfer function
- "Model"

The transition function

- The transition function captures the evolution over time:

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

  - » At time t:

    S_t is known (deterministic)

    x_t is a deterministic function of S_t

    W_{t+1} is random

The transition function

- Two frameworks:
  - » Model-based
    - This is where we have a set of equations that describe the transition.
    - In computer science, it refers to problems where the one-step transition matrix is known.
  - » Model-free
    - Here, we do not know the transition function.
    - Typical for complex problems (describing human behavior, the economy, climate, a complex physical problem)
    - In this case, we simply observe the next state without knowing how we got there.

The transition function

- Illustrations
  - » Our deterministic shortest path problem

$$S_t = \text{node} = i$$

$$\text{Decision } x_{ij} = 1$$

$$S_{t+1} = j$$

  - » The stochastic shortest path problem

$$S_t = \left(R_t, \left(\hat{c}_{ij}\right)_j\right)$$

$$\text{Decision } x_{ij} = 1$$

$$R_{t+1} = j$$

$$S_{t+1} = \left(R_{t+1}, \left(\hat{c}_{jk}\right)_k\right)$$

The transition function

- Illustrations
  - » The first pricing problem
    - State variable

$$S_t = (p_t, \bar{p}_t, R_t)$$

    - Transition function

$$p_{t+1} = p_t + \hat{p}_{t+1}$$

$$\bar{p}_{t+1} = (1-\alpha)\,\bar{p}_t + \alpha\, p_{t+1}$$

$$R_{t+1} = R_t - x_t$$

Modeling frameworks

Objective function

The objective function

- Types of objectives:
  - » Rewards, profits, revenues, contributions (business)
  - » Gains, losses (engineering)
  - » Strength, conductivity, diffusivity (materials science)
  - » Stability
  - » Tolerance, toxicity, effectiveness (health)
  - » Risk, volatility (finance)
  - » Utility (finance)

The objective function

- Objective functions
  - » Ways of writing objective functions:

$c_t x_t$ or $f(x)$ – Linear or nonlinear cost (deterministic)

$F(x^n, W^{n+1})$ – Function does not depend on state, but does depend on noisy implementation W

$C(S_t, x_t)$ – Cost/contribution deterministic function of state and action

$C(S_t, x_t, W_{t+1})$ – Cost/cont. depends on state, action and random information.

$C(S_t, x_t, S_{t+1})$ – Cost/cont. depends on state, action and next observed state

» The most general is

$$C(S_t, x_t, W_{t+1}) \quad \text{or} \quad C(S^n, x^n, W^{n+1})$$

where decisions are given by

$$x_t = X^\pi(S_t) \qquad \text{or} \qquad x^n = X^\pi(S^n)$$

The objective function

- Objective functions
  - » Final-reward version – Just look at the performance of the final implementation decision:

$$\max_{\pi} \mathbb{E}\left\{F(X^{\pi,N},W) \mid S_0\right\}$$

  - » Cumulative reward version – You have to add up the rewards as we progress:

$$\max_{\pi} \mathbb{E}\left\{\sum_{n=0}^{N-1} F(X^{\pi}(S^n),W^{n+1}) \mid S_0\right\}$$

or:

$$\max_{\pi} \mathbb{E}\left\{\sum_{t=0}^{T} C_t(S_t, X_t^{\pi}(S_t),W_{t+1}) \mid S_0\right\}$$

The objective function

- Our most general form (up to the expectation)

$$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t), W_{t+1}) \mid S_0\right\}$$

$$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C(S_t, X_t^\pi(S_t)) \mid S_0\right\}$$

$$\max_\pi \mathbb{E}\left\{\sum_{t=0}^{T} C(X_t^\pi(S_t), W_{t+1}) \mid S_0\right\}$$

$$\max_\pi \mathbb{E}\left\{C(X_T^\pi, W) \mid S_0\right\}$$

$$\max_\pi \mathbb{E}\left\{F(X^{\pi,N}, W) \mid S_0\right\}$$

The objective function

- Ways to perform an evaluation:
  - » Offline learning (learning in a simulator)
    - This is where you learn from a simulated system, usually on the computer (for our field).
    - Strengths:
      - Can evaluate many policies in a relatively short period of time.
    - Weaknesses:
      - Good computer simulators can be difficult to build (can take years)
      - Despite this, you still have to live with your modeling assumptions, in particular about uncertainty.

The objective function

- Ways to perform an evaluation:
  - » Online learning (learn as you go)
    - This is where we implement a policy in the field, and watch how well it works.
    - Strengths:
      - You do not need to live with simplified models of complex phenome.
      - You can experience sources of uncertainty that you cannot even identify.
    - Weaknesses:
      - Very slow! Takes a year to simulate a year.
      - Very hard to compare new policies.
      - You have to live with your mistakes.

The objective function

- Characteristics of the objective function
  - » Analytical behavior
    - Concave/convex, unimodal, monotone, smooth,…
  - » Computational cost:
    - Fractions of a second – Analytical functions
    - Minutes – Computer simulations
    - Hours – Laboratory experiments/computer simulations
    - Days (or longer) – Laboratory/field experiments
    - Weeks to months – Field experiments
  - » Startup/switching costs
    - What is involved to observe function for different inputs? Is there a cost to switch to different inputs?
  - » Experimental noise (low to high noise, nature of noise)

The objective function

- There are different uncertainty that we can:
  - » Expectations

$$\min_x \mathbb{E}F(x,W)$$

  - » Risk measures

$$\min_x \mathbb{E}F(x,W) + \theta\mathbb{E}\left[F(x,W) - f_\alpha\right]_+^2$$

$$\min_x \rho\left(F(x,W)\right) \qquad \rho \in \text{Convex/coherent risk measures}$$

  - » Worst case ("robust optimization")

$$\min_x \max_w F(x,w)$$

Modeling

- Deterministic
  - » Objective function

$$\min_{x_0,\ldots,x_T} \sum_{t=0}^{T} c_t x_t$$

  - » Decision variables:

$$(x_0,\ldots,x_T)$$

  - » Constraints:
    - at time t

$$\left.\begin{aligned} A_t x_t &= R_t \\ x_t &\geq 0 \end{aligned}\right\} \mathcal{X}_t$$

    - Transition function

$$R_{t+1} = b_{t+1} + B_t x_t$$

- Stochastic
  - » Objective function

$$\max_\pi E^\pi \left\{ \sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^\pi(S_t), W_{t+1}\right) \mid S_0 \right\}$$

  - » Policy

$$X^\pi : S \mapsto \mathcal{X}$$

  - » Constraints at time t

$$x_t = X_t^\pi(S_t) \in \mathcal{X}_t$$

  - » Transition function

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

  - » Exogenous information

$$(W_1, W_2, \ldots, W_T)$$

Week 4 - Wednesday

Policies

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.

- How do we search over an arbitrary space of policies?

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	

Review of policies in previous problems

- Start:
  - » List all the different policies covered in the class with some explanation (all boards)
  - » End with bulleted list on the left-most boards
  - » Erase remaining boards

- Asset selling
  - » Sell signal:

$$X^{sell-low}(S_t|\theta^{low}) = \begin{cases} 1 & \text{If } p_t < \theta^{low} \text{ and } R_t = 1 \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \quad (2.1)$$

  - » or

$$X^{high-low}(S_t|\theta^{high-low}) = \begin{cases} 1 & \text{If } p_t < \theta^{low} \text{ or } p_t > \theta^{high} \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \quad (2.14)$$

  - » or

$$X^{track}(S_t|\theta^{track}) = \begin{cases} 1 & \text{If } p_t \geq \bar{p}_t + \theta^{track} \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \quad (2.16)$$

$$\bar{p}_t = (1-\alpha)\bar{p}_{t-1} + \alpha\hat{p}_t. \quad (2.15)$$

Adaptive market planning

- Stepsize policies
  - » Harmonic stepsize formula (deterministic

$$\alpha_n(\theta^{step}) = \frac{\theta^{step}}{\theta^{step} + n - 1}, \tag{3.8}$$

  - » Kesten's rule (stochastic)

$$\alpha_n(\theta^{step}) = \frac{\theta^{step}}{\theta^{step} + K^n - 1}, \tag{3.9}$$

$$K^n = \begin{cases} K^n + 1 & \text{If } (\nabla^x F(x^n, W^{n+1}))^T \nabla^x F(x^{n-1}, W^n) < 0, \\ K^n & \text{Otherwise} \end{cases} \tag{3.10}$$

  - » Discuss:
    - Scaling
    - Handling bias (learning) vs. noise (smoothing)

- Choosing diabetes medication
  - » Upper confidence bounding policies

$$X^{UCB}(S^n|\theta^{UCB}) = \arg\max_{x\in\mathcal{X}}\left(\bar{\mu}_x^n + \theta^{UCB}\sqrt{\frac{\log n}{N_x^n}}\right), \tag{4.5}$$

  - » Interval estimation

$$X^{IE}(S^n|\theta^{IE}) = \arg\max_{x\in\mathcal{X}}\left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right), \tag{4.6}$$

- Shortest paths – static/stationary
  - » Deterministic, or stochastic (but we see costs after we choose the link):

$$X^{\pi}(i) = \arg\min_{j \in \mathcal{N}_i^+} (\bar{c}_{ij} + v_j). \quad (5.9)$$

  - » Stochastic (we see costs before we make a decision)

$$X^{\pi}(S_t) = argmax_j \, (\hat{c}_{ij} + \bar{V}_{tj}^{x,n-1})$$

- Knowledge gradient (one step lookahead)

$$\nu_x^{KG,n} = \mathbb{E}_\mu \mathbb{E}_{W|\mu} \left\{ \max_{x'} \bar{\mu}_{x'}^{n+1}(x) \,|\, S^n \right\} - \max_{x'} \bar{\mu}_{x'}^n$$

» Discuss $\bar{\mu}_{x'}^{n+1}(x)$ as the estimated updated belief if we run experiment x .

» Think of this estimate as looking one step into the future.

» Think about how we might estimate the expectations using simulation:

$$\nu_x^{KG,n} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{L}\sum_{l=1}^{L} \max_{x'} \bar{\mu}_{x'}^{n+1}(x \,|\, S^n, W_x^{l|k} = \mu_x^k + \varepsilon^l) - \max_{x'} \bar{\mu}_{x'}^n$$

» Review how means are updated given estimates in S^n (which contains $\bar{\mu}_x^n$) and the observation $W_x^{l|k}$ for a truth μ_x^k and noise ϵ^l

» Remember that μ_x^k is a possible *true value* of μ, not the estimate. We have an estimate, but the random outcome *W* comes from the *truth*.

- Imagine that the lookahead is just a black box:
  - » Solve the optimization problem

$$X_t^{\pi}(S_t^n) = \arg\min \sum_{i \in N} \sum_{j \in N_i^+} \tilde{c}_{tij}^n \tilde{x}_{tij}$$

  - » subject to

$$\sum_j \tilde{x}_{i_t^n, j} = 1 \quad \text{Flow out of current node where we are located}$$

$$\sum_i \tilde{x}_{i,r} = 1 \quad \text{Flow into destination node } r$$

$$\sum_i \tilde{x}_{i,j} - \sum_k \tilde{x}_{j,k} = 0 \quad \text{for all other nodes.}$$

  - » This is a deterministic shortest path problem that we could solve using Bellman's equation, but for now we will just view it as a black box optimization problem.

- The θ −percentile policy.
  - » Solve the linear program (shortest path problem):

$$X_t^\pi(S_t^n \mid \theta) = \arg\min \sum_{i \in N} \sum_{j \in N_i^+} \tilde{c}_{tij}^{\,p}(\theta)\tilde{x}_{tij} \quad \text{(Vector with } x_{tij} = 1 \text{ if decision is to take } (i, j)\text{)}$$

  - » subject to

$$\sum_j \tilde{x}_{t,i_t^n,j} = 1 \quad \text{Flow out of current node where we are located}$$

$$\sum_i \tilde{x}_{tir} = 1 \quad \text{Flow into destination node } r$$

$$\sum_i \tilde{x}_{tij} - \sum_k \tilde{x}_{tjk} = 0 \quad \text{for all other nodes.}$$

  - » This is a deterministic shortest path problem that we could solve using Bellman's equation, but for now we will just view it as a black box optimization problem.

- Erase boards. Put this list of policies on left boards:
  - Sell if price is above a number
  - Buy low, sell high
  - Stepsize policies
  - Upper confidence bounding
  - Interval estimation
  - Value functions for shortest paths
  - Approximate value functions (shortest path)
  - Knowledge gradient
  - Dynamic shortest paths
  - Paramterized stochastic shortest path

Four classes of policies

Designing policies

- Two strategies for designing policies

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.

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

Designing policies

- 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)

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

Designing policies

- Lookahead approximations:

  2a) Policies based on value function approximations (VFAs)

  - Using pre-decision state:

$$X^{VFA}(S_t \mid \theta) = \arg\max_x \left(C(S_t, x_t) + \mathbb{E}\left\{ V_{t+1}(S_{t+1}) \mid S_t \right\} \right)$$

  - Using post-decision state

$$X^{VFA}(S_t \mid \theta) = \arg\max_x \left(C(S_t, x_t) + V_t^x(S_t^x) \right)$$

  2b) Direct lookaheads (DFAs)

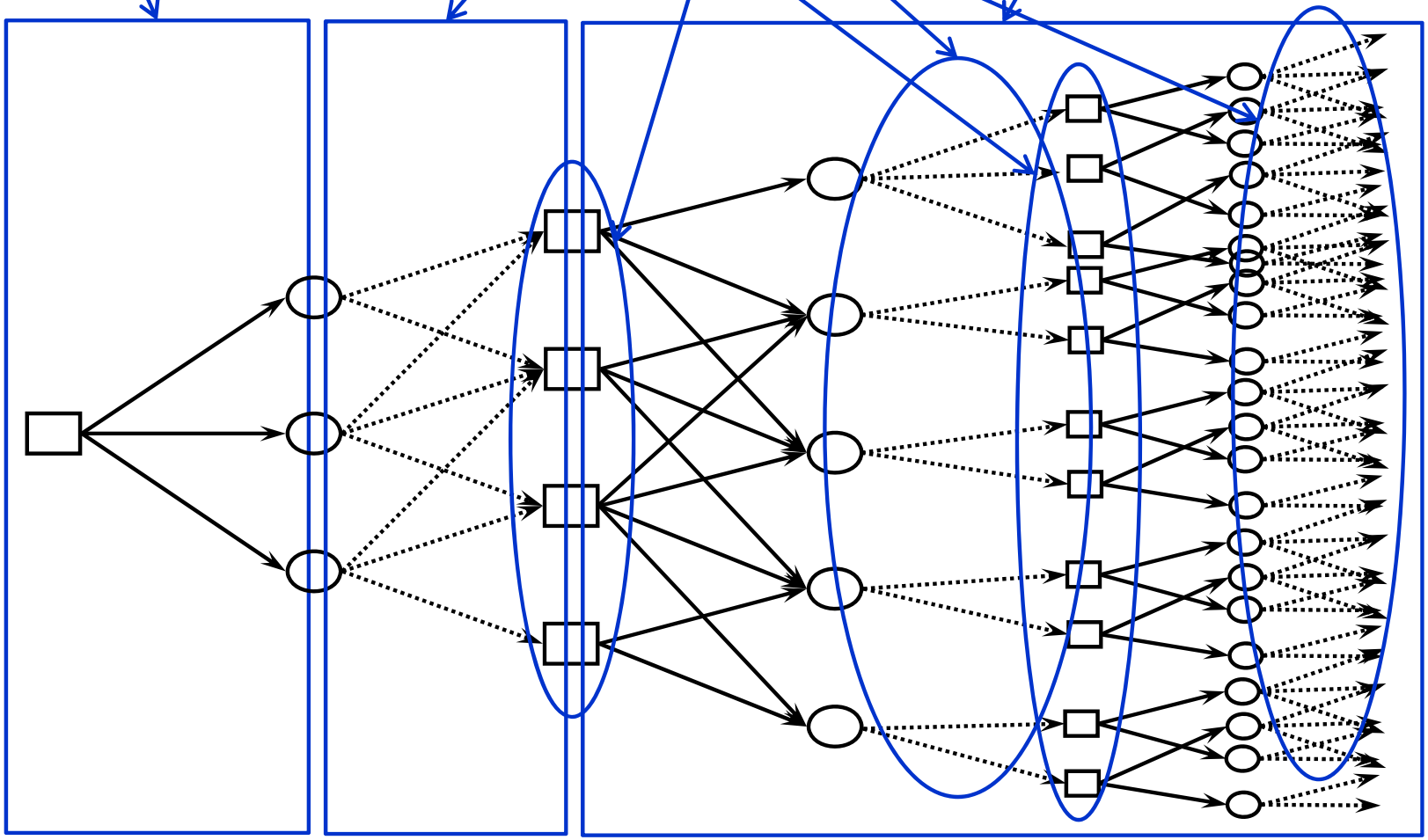
  - Optimizing some approximation of the future (might be deterministic, might be stochastic). A deterministic lookahead might be written

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

Designing policies

- The ultimate lookahead policy is optimal

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



Designing policies

- Lookahead approximations – Approximate the impact of a decision now on the future:
  - » An optimal policy (based on looking ahead):

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

2a) Approximating the value of being in a downstream state using machine learning ("value function approximations")

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

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

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

Designing policies

- Lookahead approximations – Approximate the impact of a decision now on the future:
  - » An optimal policy (based on looking ahead):

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

2a) Approximating the value of being in a downstream state using machine learning ("value function approximations")

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

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

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

Designing policies

- Lookahead approximations – Approximate the impact of a decision now on the future:
  - » An optimal policy (based on looking ahead):

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

2a) Approximating the value of being in a downstream state using machine learning ("value function approximations")

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

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

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

Designing policies

- The ultimate lookahead policy is optimal

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

» 2b) Expectations and max over policies are not computable. Instead, we have to solve an approximation called the *lookahead model:*

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

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

Designing policies

- The ultimate lookahead policy is optimal

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

Maximization that we cannot compute

Expectations that we cannot compute

Designing policies

- Types of lookahead approximations
  - » One-step lookahead – Widely used in pure learning policies:
    - Bayes greedy/naïve Bayes
    - Thompson sampling
    - Value of information (knowledge gradient)
  - » 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.

Lookahead policies

- Lookahead models use five classes of approximations:
  - » Horizon truncation – Replacing a longer horizon problem with a shorter horizon
  - » Stage aggregation – Replacing multistage problems with two-stage approximation.
  - » Outcome aggregation/sampling – Simplifying the exogenous information process
  - » Discretization – Of time, states and decisions
  - » Dimensionality reduction – We may ignore some variables (such as forecasts) in the lookahead model that we capture in the base model (these become *latent* variables in the lookahead model).

- These policies fall into four main groups:
  - » Policy function approximations (CFAs) – parameterized functions that map state to action:
    - Asset selling policies
    - Adaptive marketing planning (gradient-based stochastic optimization)
  - » Cost function approximations (CFAs) – These are parameterized optimization problems
  - » Policies based on value function approximations (VFAs) – Make a decision using an estimate of the value of a downstream state.
  - » Direct lookaheads (DLAs) – We optimize over a future to determine what we should do right now.

- Put four classes of policies on middle boards
  - » Policy search on top board
  - » Lookaheads on bottom board
- Draw lines from list on left to the four classes
- Notes:
  - » Our reason for the four classes is purely computational.
  - » We could just do policy search if we could search over the ultimate family of functions
    - E.g. neural networks – problem is that it does not capture structure. Illustrate with shortest path problem.
  - » We could just do lookahead, if we could just compute that messy lookahead function.

Features of policies

Type of policy	Policy search	Lookahead	Function approximation	Imbedded optimization
PFA	X		X	
CFA	X		X	X
VFA		X	X	X
DLA		X		X

Hybrid policies

- CFA with PFA

$$X^{CFA-PFA}(S_t|\theta) = \arg\max_{x_t} \sum_{a\in\mathcal{A}} \sum_{b\in\mathcal{B}} \left(c_{ab}x_{tab} + \theta(x_{tab} - D_{tb}\rho_{ab})^2\right),$$

- Lookahead with PFA
  - » Tree search with VFA terminal reward

$$X^{\pi}(S_t) = \arg\max_{x_t,\ldots,x_{t+H}} \sum_{t'=t}^{t+H-1} C(S_{t'}, x_{t'}) + \gamma^H \overline{V}_{t+H}(S_{t+H}),$$

- Lookahead with CFA
  - » Shortest path with parameterized cost
  - » Deterministic lookahead with parameterized constraints
- VFA with PFA

$$X^{\pi}(S_t) = \arg\max_{x} \left(C(S_t, x) + \overline{V}(S^{M,x}(S_t, x)) - \beta(\bar{X}(S_t) - x)^2\right).$$

- Fitted VFA with policy search
  - » Fit a value function using sampled estimates of the value of being in a state. Use this to fit a parametric model.

$$X_t^{VFA-hybrid}(S_t) = \arg\max_x \left(C(S_t, x) + \gamma \mathbb{E}\left\{\overline{V}_{t+1}(S_{t+1})|S_t\right\}\right).$$

Now assume we are using a parameterized approximation for the value function approximation such as

$$\overline{V}_t(S_t|\theta) = \sum_{f\in\mathcal{F}} \theta_f \phi_f(S_t), \tag{11.23}$$

  - » Then do policy search

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X^{VFA-hybrid}(S_t|\theta)). \tag{11.24}$$

Policy search vs. Lookahead

- Policy search class
  - » Strengths
    - As a rule simpler than lookaheads – sometimes dramatically simpler. For this reason, these are popular in practice.
    - Able to capture insights and intuition
  - » Weaknesses
    - Typically parametric, which has to be designed by a human.
    - Tuning:
      - Tuning in a model – this means you have to design a realistic model.
      - Tuning in the real world – Slow, and you have to suffer your errors.

- Lookahead class
  - » Strengths
    - Reduced or no tuning.
    - Direct lookaheads produce complex behaviors.
    - This is what you do when all else fails, and all else often fails.
  - » Weaknesses
    - Requires:
      - Solving a direct lookahead model (can be computationally expensive).
      - … or approximating the lookahead model using a value function (this can be difficult and computationally challenging).
    - The lookahead model is… a model. Just as tuning needs a model, lookaheads need either an explicit model (the lookahead) or a source of observations (e.g. for fitting value functions).

Policy search class

Policy search class

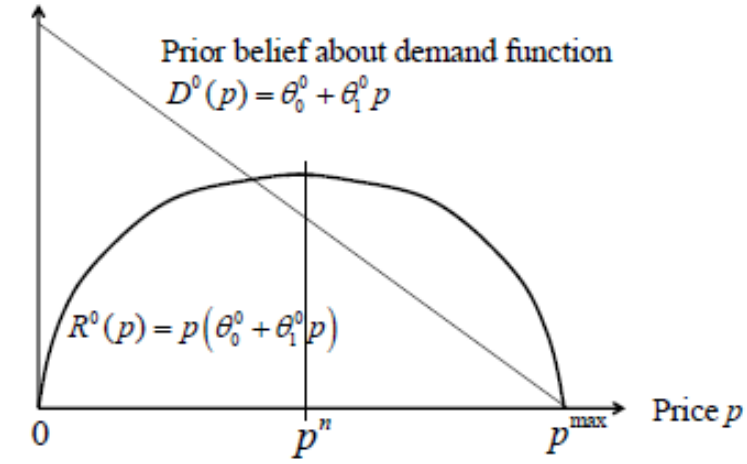
- Parameterized policies may be
  - » Pure PFAs – Some parameterized policy that maps state to action.
  - » CFAs – Parameterized optimization models such as
    - Interval estimation
    - Upper confidence bounding
    - Parameterized lookaheads, as we did with our parameterized policy for dynamic shortest paths
  - » As long as it is a parameterized policy $X^{\pi}(S_t|\theta)$, then we have to search over θ using

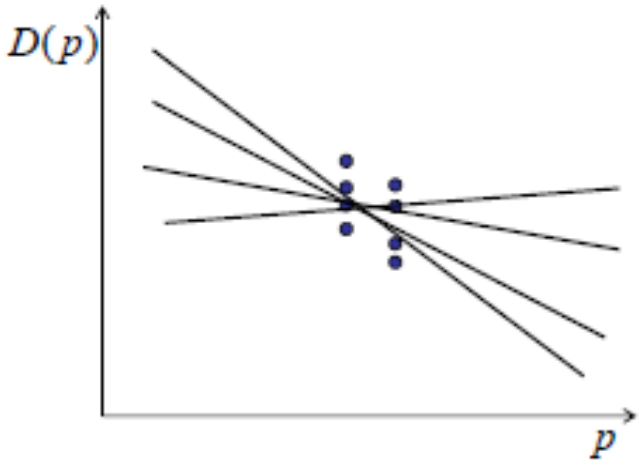
$$\max_{\theta} F(\theta) = \mathbb{E}\sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^{\pi}(S_t \mid \theta)\right)$$

- Notes:
  - » All policy search policies (PFAs and CFAs) require tuning parameters.
  - » There are two broad strategies for tuning parameters:
    - Derivative-based – This draws on the adaptive market planning work.
    - Derivative free – This draws on the diabetes example.
  - » In other words, we have a sequential decision problem (tuning parameters) to solve our sequential decision problem. ☺

Policy function approximation

- Maximizing revenue
  - » $D(p) = \theta_0 - \theta_{1p}$
  - » $R(p) = pD(p) = \theta_0 p - \theta_1 p^2$
- To maximize revenue:
  - » $\frac{dR(p)}{dp} = \theta_0 - 2\theta_1 p = 0$
  - » $p^* = \frac{\theta_0}{2\theta_1}$
- Now assume we have estimates $\bar{\theta}^n$ of the parameters θ_0 and θ_1 after n observations. Tendency will be to sample near the middle. To encourage exploration, add a noise term
  - » $p(\sigma) = \frac{\theta_0}{2\theta_1} + \varepsilon \quad \varepsilon \sim N(0, \sigma^2)$
  - » σ is a tunable parameter for the policy





Policy function approximation

- Asset selling
  - » Sell on drops

$$X^{sell-low}(S_t|\theta^{low}) = \begin{cases} 1 & \text{If } p_t < \theta^{low} \text{ and } R_t = 1 \\ 1 & \text{If } t = T \text{ and } R_t = 1 \\ 0 & \text{Otherwise} \end{cases} \quad (2.1)$$

  - » Need to tune θ^{low}

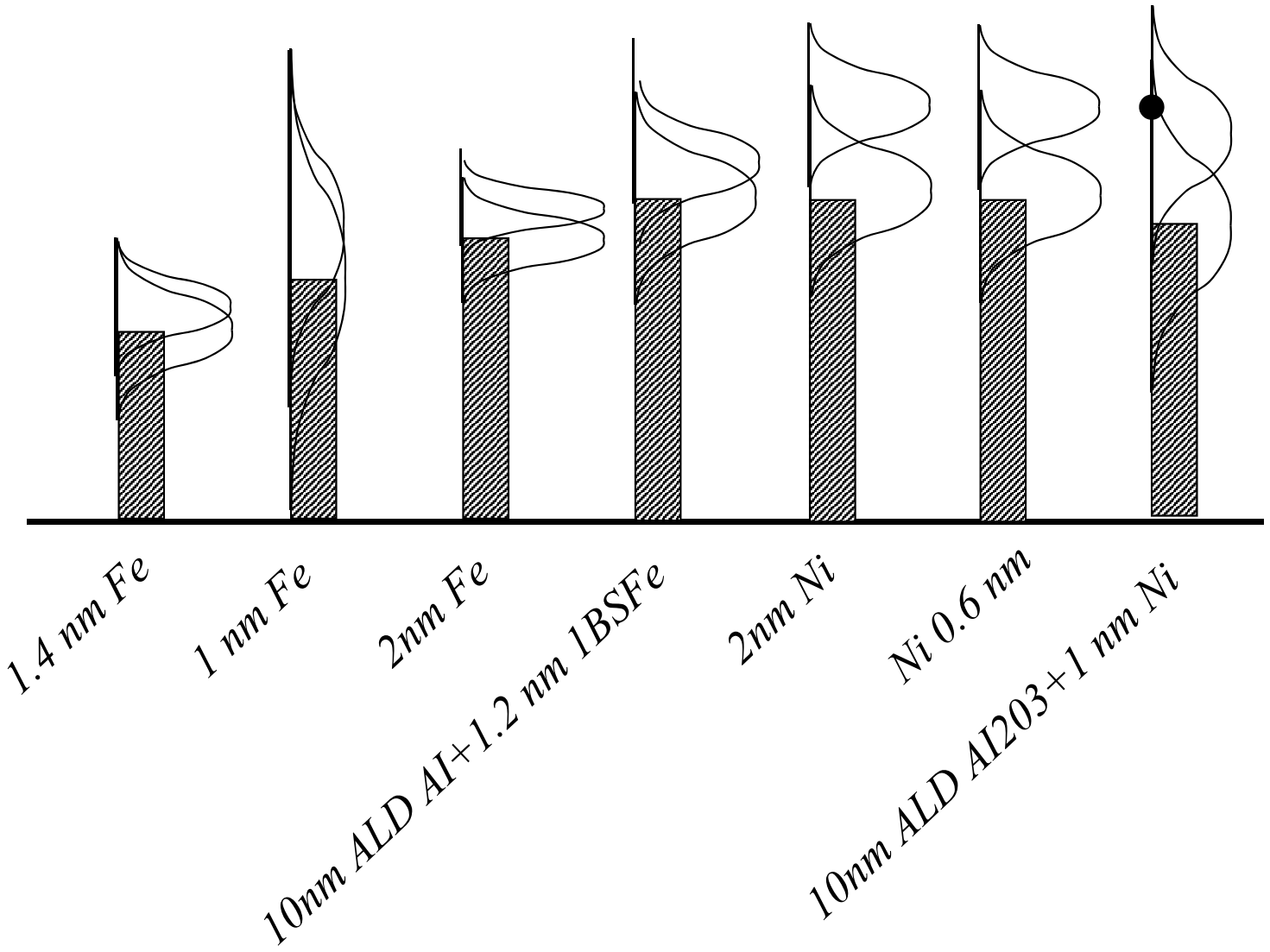
Cost function approximations

- Lookup table
  - » We can organize potential catalysts into groups
  - » Scientists using domain knowledge can estimate correlations in experiments between similar catalysts.

	1.4 nm Fe	1 nm Fe	2nm Fe	10nm ALD Al2O3+1.2 nm IBS Fe	2 nm Ni	Ni 0.6 nm	10nm ALD Al2O3+1 nm Ni
1.4 nm Fe	1	0.7	0.7	0.6	0.4	0.4	0.2
1 nm Fe	0.7	1	0.7	0.6	0.4	0.4	0.2
2nm Fe	0.7	0.7	1	0.6	0.4	0.4	0.2
10nm ALD Al2O3+1.2 nm IBS Fe	0.6	0.6	0.6	1	1	0.3	0
2 nm Ni	0.4	0.4	0.4	1	1	0.7	0.6
Ni 0.6 nm	0.4	0.4	0.4	0.3	0.7	1	0.6
10nm ALD Al2O3+1 nm Ni	0.2	0.2	0.2	0	0.6	0.6	1

Cost function approximations

- Correlated beliefs: Testing one material teaches us about other materials

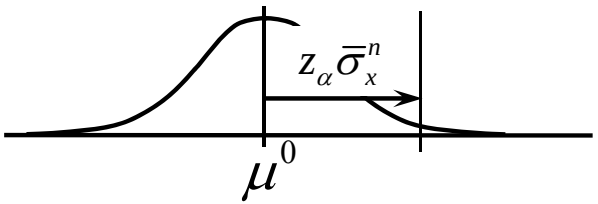


Cost function approximations

- Cost function approximations (CFA)
  - » Upper confidence bounding

$$X^{UCB}(S^n \mid \theta^{UCB}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{UCB} \sqrt{\frac{\log n}{N_x^n}} \right)$$

  - » Interval estimation



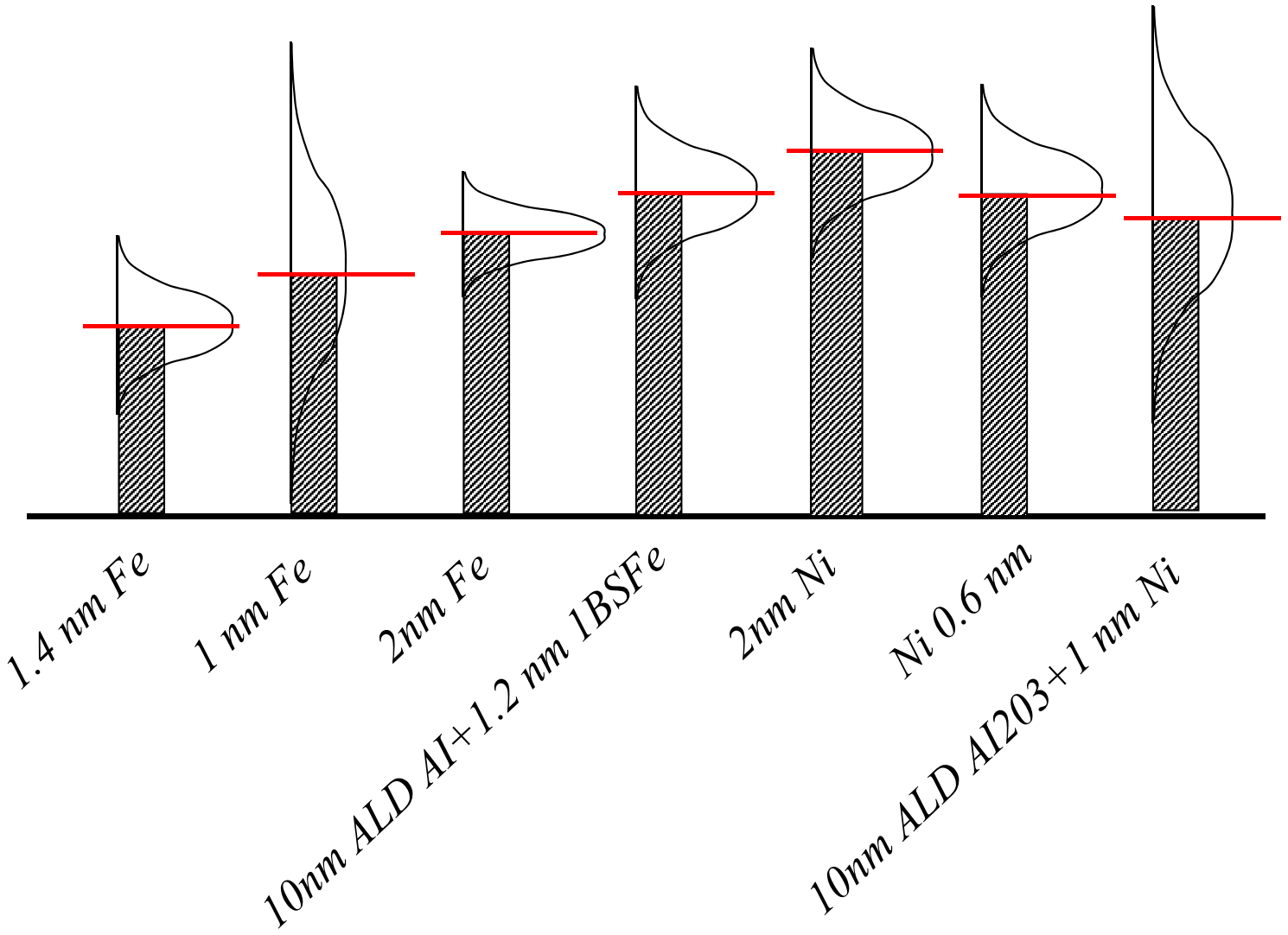
$$X^{IE}(S^n \mid \theta^{IE}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{IE} \bar{\sigma}_x^n \right)$$

  - » Boltzmann exploration ("soft max")
    - Choose x with probability: $P_x^n(\theta) = \dfrac{e^{\theta \bar{\mu}_x^n}}{\sum_{x'} e^{\theta \bar{\mu}_{x'}^n}}$

$$X^{Boltz}(S^n|\theta) = \arg\max_x \{x | P_x^n(\theta) \leq U\}.$$

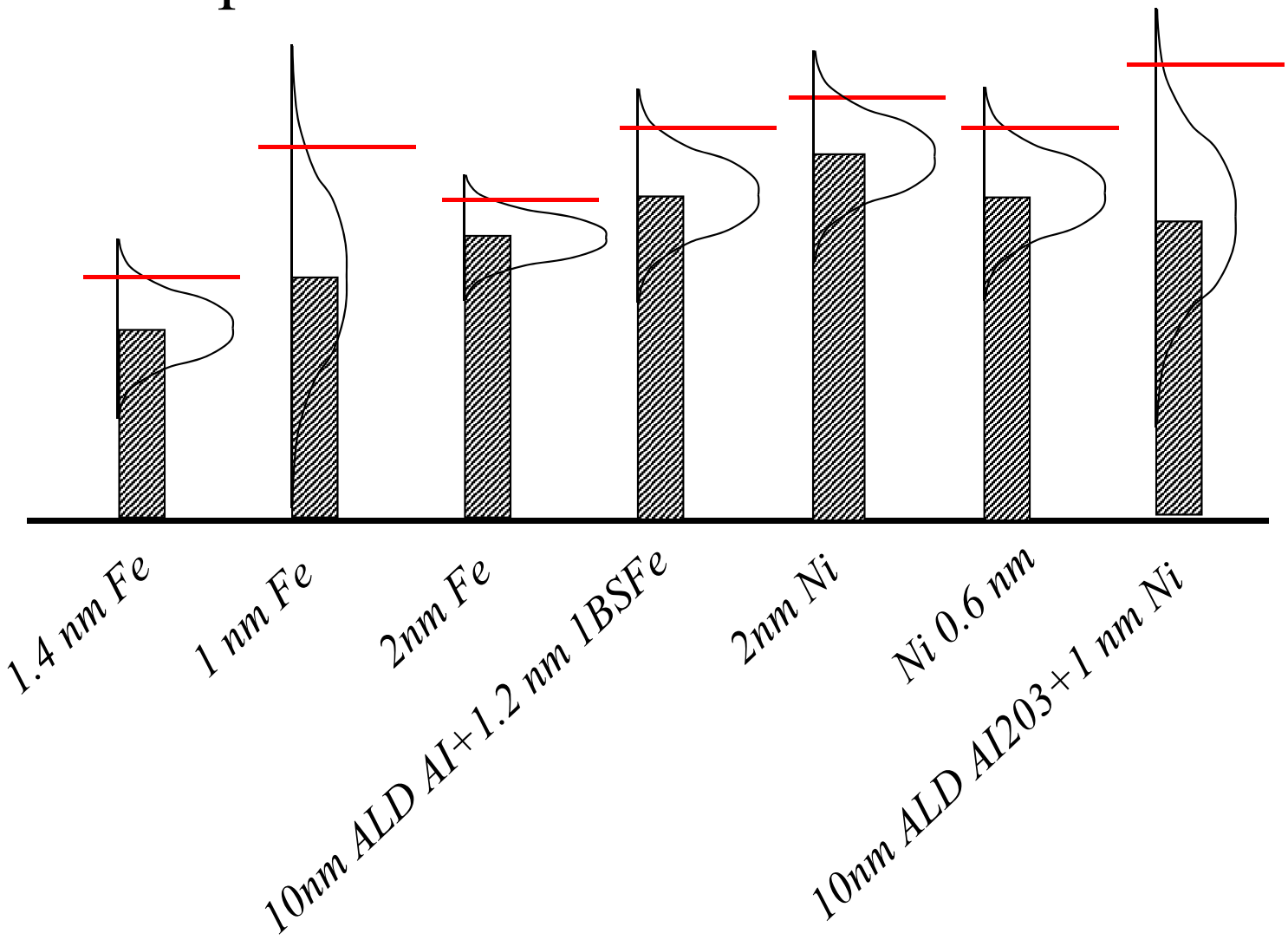
Cost function approximations

- Picking $\theta^{IE} = 0$ means we are evaluating each choice at the mean.



Cost function approximations

- Picking $\theta^{IE} = 2$ means we are evaluating each choice at the 95th percentile.



Cost function approximations

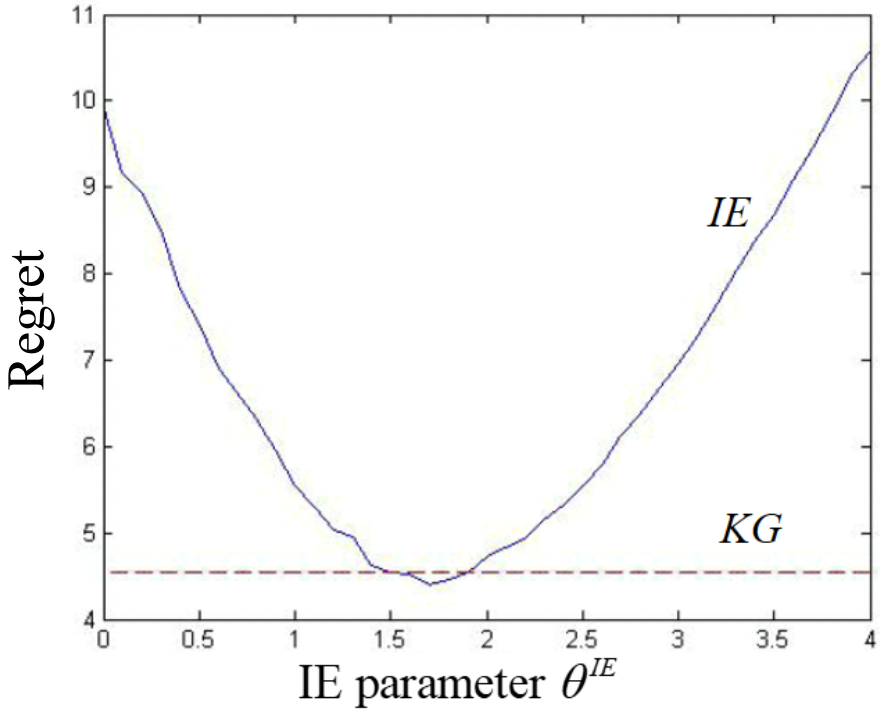
- Optimizing the policy
  - » We optimize θ^{IE} to maximize:

$$\max_{\theta^{IE}} F(\theta^{IE}) = \mathbb{E}F\left(x^{\pi,N}, W\right)$$

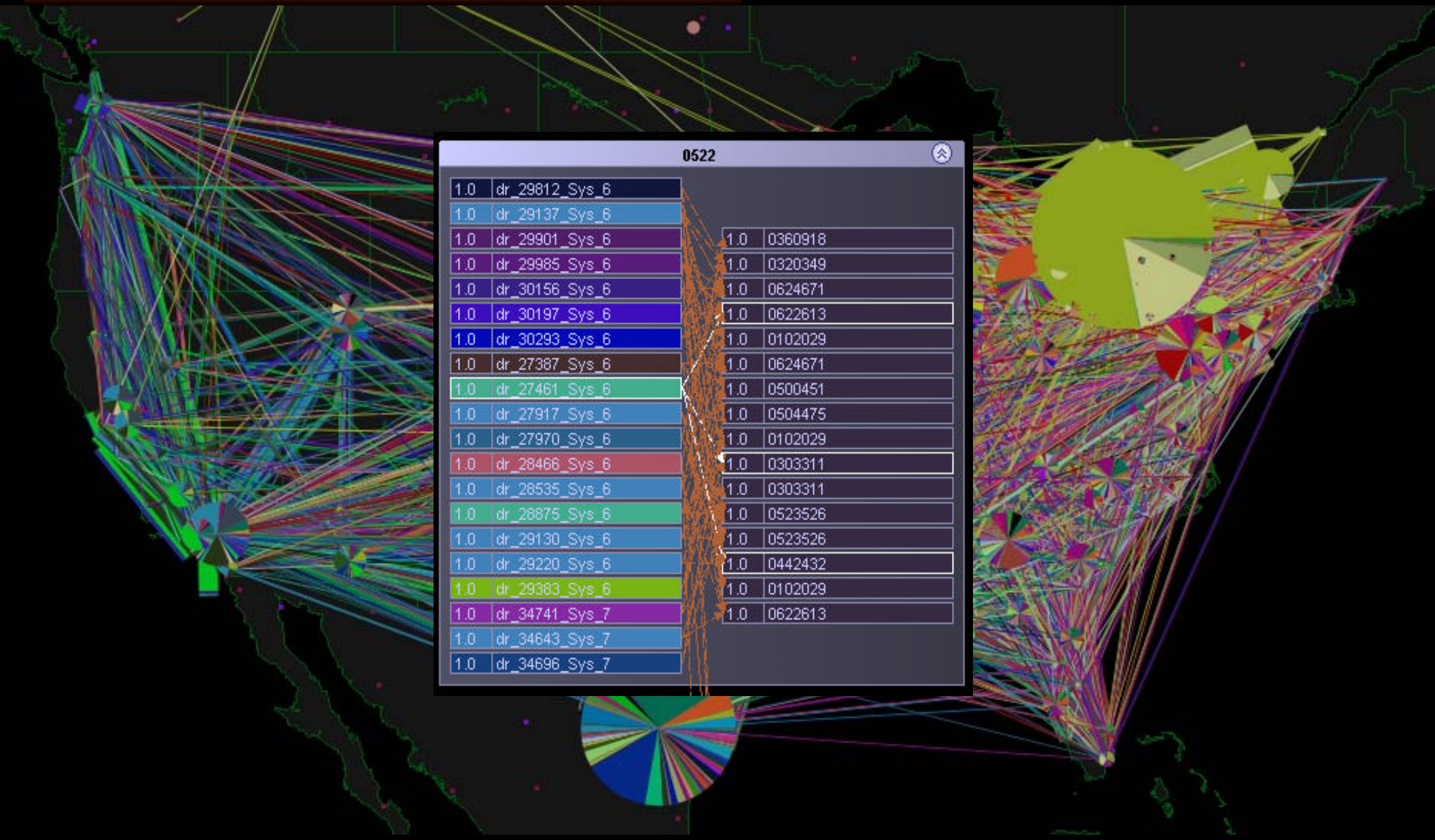
where

$$x^n = X^{IE}(S^n \mid \theta^{IE}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right) \qquad S^n = (\bar{\mu}_x^n, \bar{\sigma}_x^n)$$

- Notes:
  - » This can handle any belief model, including correlated beliefs, nonlinear belief models.
  - » All we require is that we be able to simulate a policy.



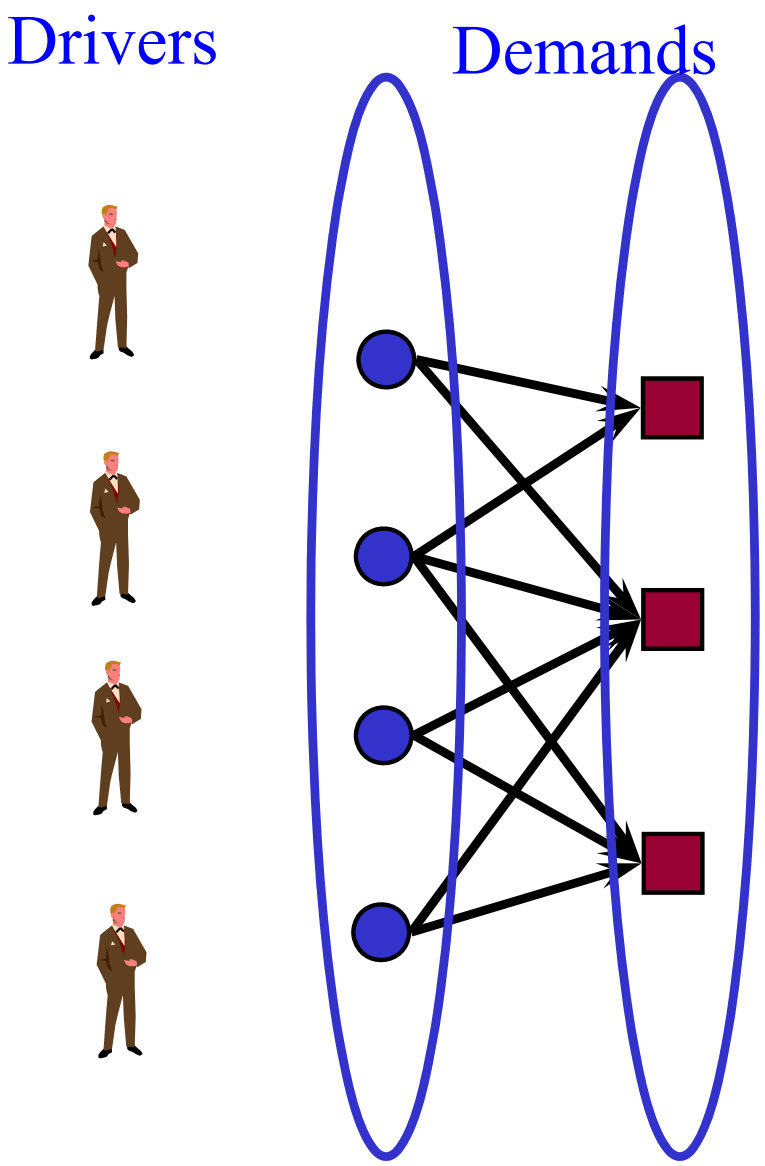
Schneider National



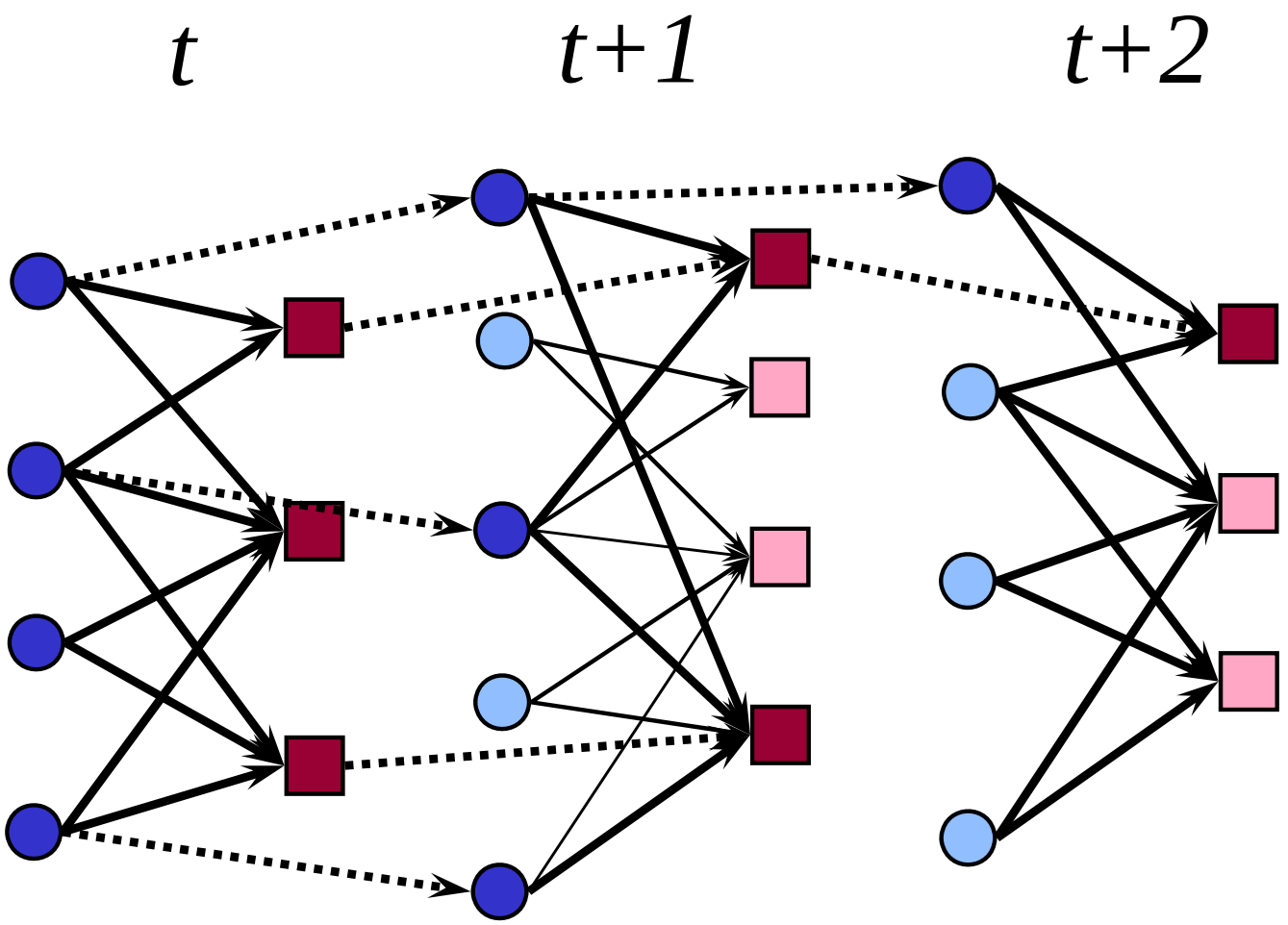

SCHNEIDER
Week 45
1 Way Rank
Dedic Rank
Trkfll Rank
Brok. Rank
Miles
18
4
1
3
1
Working units
18
1
By industry
Auto Asmby
Auto Parts
Retail
Paper
Miles
8
27
1
9
33
Nov YOY Growth
1 Way
Dedic
Trkfll
-5%
-6%
28%
SCHNEIDER

Slide 506

Cost function approximations



Cost function approximations



The assignment of drivers to loads evolves over time, with new loads being called in, along with updates to the status of a driver.

Cost function approximations

- A purely myopic policy would solve this problem using

$$\min_x \sum_d \sum_l c_{tdl} x_{tdl}$$

where

$$x_{tdl} = \begin{cases} 1 & \text{If we assign driver } d \text{ to load } l \\ 0 & \text{Otherwise} \end{cases}$$

$$c_{tdl} = \text{Cost of assigning driver } d \text{ to load } l \text{ at time } t$$

What if a load it not assigned to any driver, and has been delayed for a while? This model ignores the fact that we eventually have to assign someone to the load.

Cost function approximations

- We can minimize delayed loads by solving a modified objective function:

$$\min_x \sum_d \sum_l \left(c_{tdl} - \theta \tau_{tl} \right) x_{tdl}$$

where

τ_{tl} = How long load l has been delayed by time t

θ = Bonus for moving a delayed load

We refer to our modified objective function as a *cost function approximation*.

Cost function approximations

- We now have to tune our policy, which we define as:

$$X^{\pi}(S_t \mid \theta) = \arg\min_x \underbrace{\sum_d \sum_l \left(c_{tdl} - \theta\tau_{tl}\right) x_{tdl}}_{\bar{C}^{\pi}(S_t, x_t \mid \theta)}$$

We can now optimize θ, another form of *policy search*, by solving

$$\min_{\theta} F^{\pi}(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t \mid \theta))$$

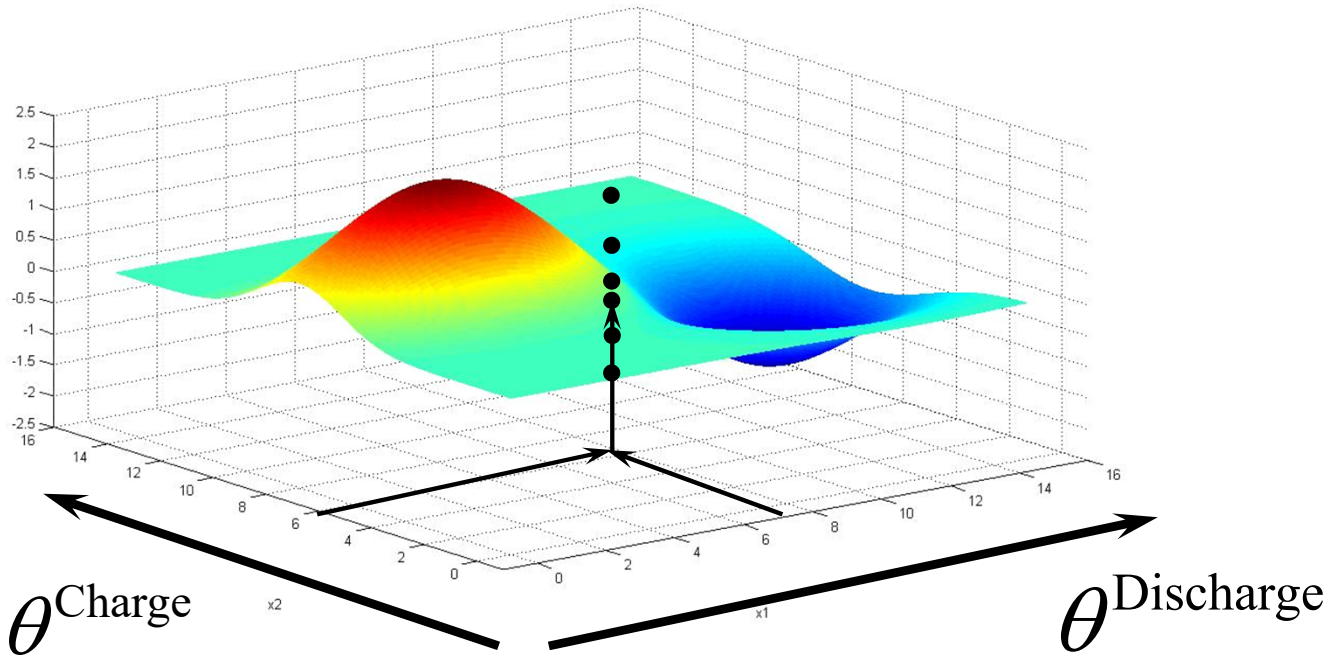
Evaluating policies

Policy search class

- Finding the best policy
  - » We need to maximize

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T}\gamma^t C\left(S_t, X_t^\pi(S_t \mid \theta)\right)$$

  - » We cannot compute the expectation, so we run simulations:



- There are two settings for doing policy search:
  - » Offline
    - In a computer simulation
    - In a lab where we do not mind making mistakes
    - We look to optimize the performance of the final design after a series of experiments
  - » Online
    - This means in the field, where we care about how well we do while we are learning.
    - We look to optimize performance along the way, which means we are maximizing cumulative reward.
  - » Caution: "offline" and "online" are terms used in the machine learning community:
    - "offline" means batch learning
    - "online" means continuous updating

- Offline ("final reward") objective
  - » The final reward process works as follows:
    - States, decisions and exogenous information evolve according to

$$(S^0, x^0, W^1, S^1, x^1, W^2, \dots S^n, x^n, W^{n+1}, \dots)$$

    - Decisions are made according to a policy $x^n = X^\pi(S^n)$.
    - States are updated using

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

    - After N experiments, we call our final decision $x^{\pi,N}$, since it depends on our policy π and the budget N.
    - Once we have the final design, we then have to stop and test the design using new observations that we call $\widehat{W}$.

- The value of the policy
  - » We can start by writing
$$F^{\pi} = \mathbb{E}F(x^{\pi,N}, \hat{W})$$
  - » … but this is not very clear.
  - » There are up to three different sources of uncertainty:
    - The initial state S^0, which might have a Bayesian prior about the performance of the different diabetes drugs.
    - The observations $W^1, W^2, \ldots, W^N$
    - The observations made during the final testing $\hat{W}$.
  - » We can write the value of a policy more clearly as
$$F^{\pi} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,W^2,\ldots,W^N|S^0}\mathbb{E}_{\hat{W}|S^0}F(x^{\pi,N}, \hat{W})$$
  - » This is better, but we won't really understand it until we know how to compute it.

- The value of the policy (cont'd)
  - » To compute
    $$F^{\pi} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,W^2,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N},\widehat{W})$$
  - » We have to simulate the expectations. Let's break down each one:
  - » $\mathbb{E}_{S^0}$: Imagine that S^0 includes a Bayesian prior such as the belief about how a patient responds to a diabetes medication. Let $\mu = (\mu_x)_{x\in X}$ be the truth of the performance of each drug x. Let's just sample $k = 1,\ldots,K$ samples of the vector μ to get $(\mu^1,\mu^2,\ldots,\mu^k,\ldots,\mu^K)$ samples of what each of the truths might be, and let's let $p^0 = \left(p_k^0\right)_{k=1}^K$ be the prior probabilities on these samples (perhaps 1/K).

- The value of the policy (cont'd)
  - » To compute
    $$F^{\pi} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,W^2,\ldots,W^N|S^0}\mathbb{E}_{\widehat{W}|S^0}F(x^{\pi,N},\widehat{W})$$
  - » $\mathbb{E}_{W^1,W^2,\ldots,W^N|S^0}$: Given S^0 means picking one of the $\mu^k = \left(\mu_x^k\right)_{x\in X}$ as the true truth for each drug. Now let's try drug $x = x^n$ after the nth trial, and then observe
    $$W_x^{n+1} = \mu_x^k + \varepsilon_x^{n+1}$$
    where ε_x^{n+1} is the noise in an observation.
  - » Let $W = (W^1, W^2, \ldots, W^N)$ be a sequence of realizations. Again assume we have a series of samples that we will call $W^{(1)}, W^{(2)}, \ldots W^{(\ell)}, \ldots, W^{(L)}$ a sample of all the observations (actually a sample of ε).

- The value of the policy (cont'd)
  - » To compute
    $$F^{\pi} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,W^2,\ldots,W^N|S^0}\mathbb{E}_{\hat{W}|S^0}F(x^{\pi,N},\hat{W})$$
  - » $\mathbb{E}_{\hat{W}|S^0}$: At this point we have our final design $x^{\pi,N}$ which depends on the truth μ^k and the experimental outcomes $W^1, W^2, \ldots, W^L$. We write this as $x^{\pi,N}(k,l)$.
  - » Now we have to test it by observing a new outcome $\hat{W}$. Again assume we sample this, and observe $\hat{W}^1, \hat{W}^2, \ldots, \hat{W}^m, \ldots, \hat{W}^M$ outcomes.
  - » Now simulate F^{π} using
    $$\bar{F}^{\pi} = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{L}\sum_{l=1}^{L}\sum_{m=1}^{M}F\,(x^{\pi,N}(k,l),\hat{W}^m)$$

- Online ("cumulative reward") objective
  - » This is how we evaluated our diabetes policy. Rather than evaluating at the end, we evaluate as we proceed, which makes the formula a bit simpler.
  - » The expected performance of a policy would be written

$$F^{\pi} = \mathbb{E}_{S^0}\mathbb{E}_{W^1,W^2,\ldots,W^N|S^0}\sum_{n=0}^{N-1} F(X^{\pi}(S^n), W_x^{n+1})$$

  - » Using samples to approximate the expectations is just as we did for the offline case. The only difference is that we sum our performance, and we do not have to separate the "learning" from the "evaluating."
  - » A more difficult issue is how we do learning in a field situation. This is active research!

- Tuning environments
  - » In a simulator
    - Requires building a mathematical model of the dynamic process and, most important, the uncertainties.
    - Your policy will only be as good as your model of uncertainty.
  - » In the real world
    - No longer need a stochastic model…
    - But it takes a day to simulate a day. This can be very slow.
    - Examples:
      - FAA
      - Grid operators

Lookahead policies

Value function approximations

Designing policies

- Lookahead approximations – Approximate the impact of a decision now on the future:
  - » An optimal policy (based on looking ahead):

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

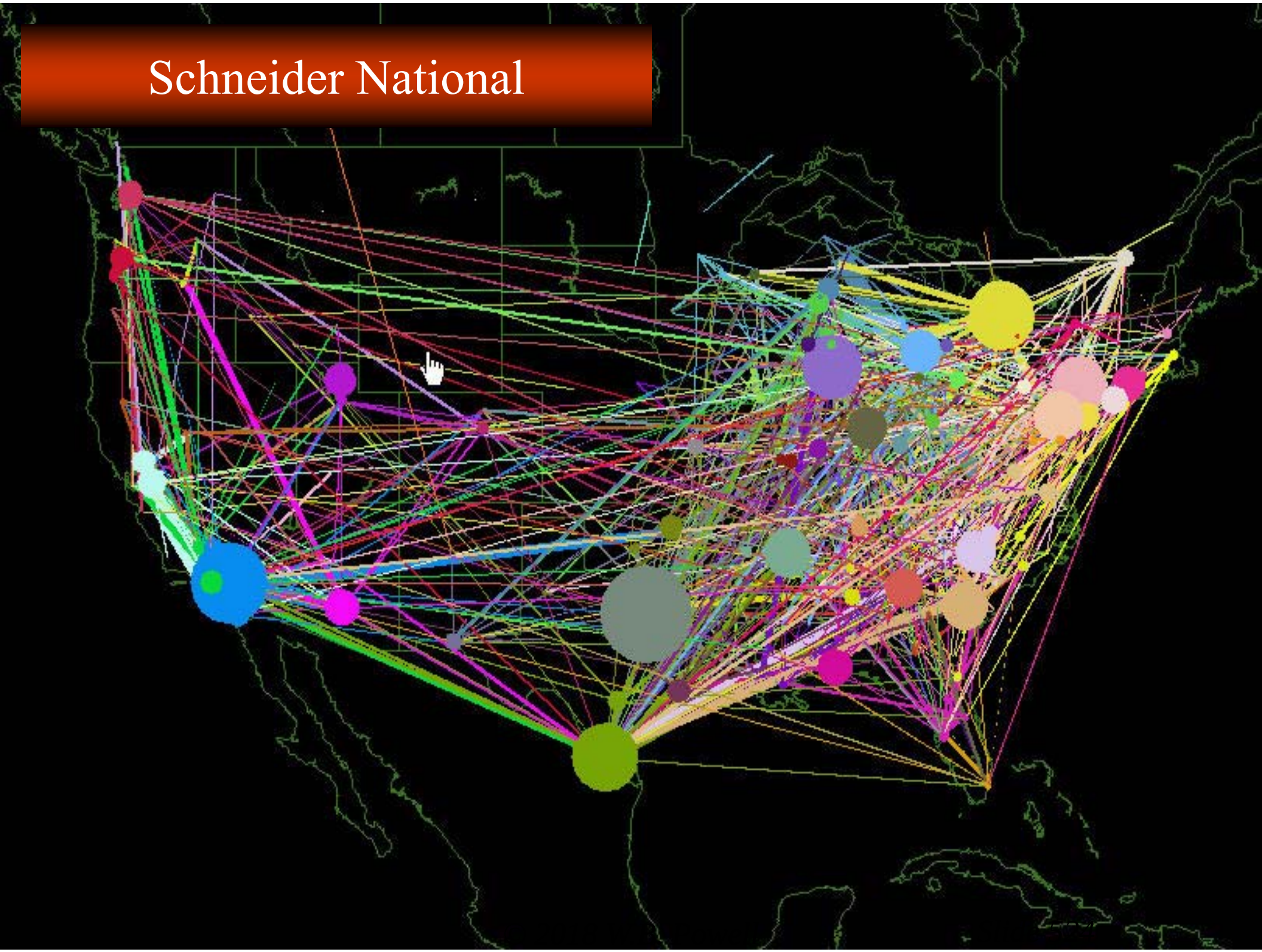
2a) Approximating the value of being in a downstream state using machine learning ("value function approximations")

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

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

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

Schneider National



ADP for trucking

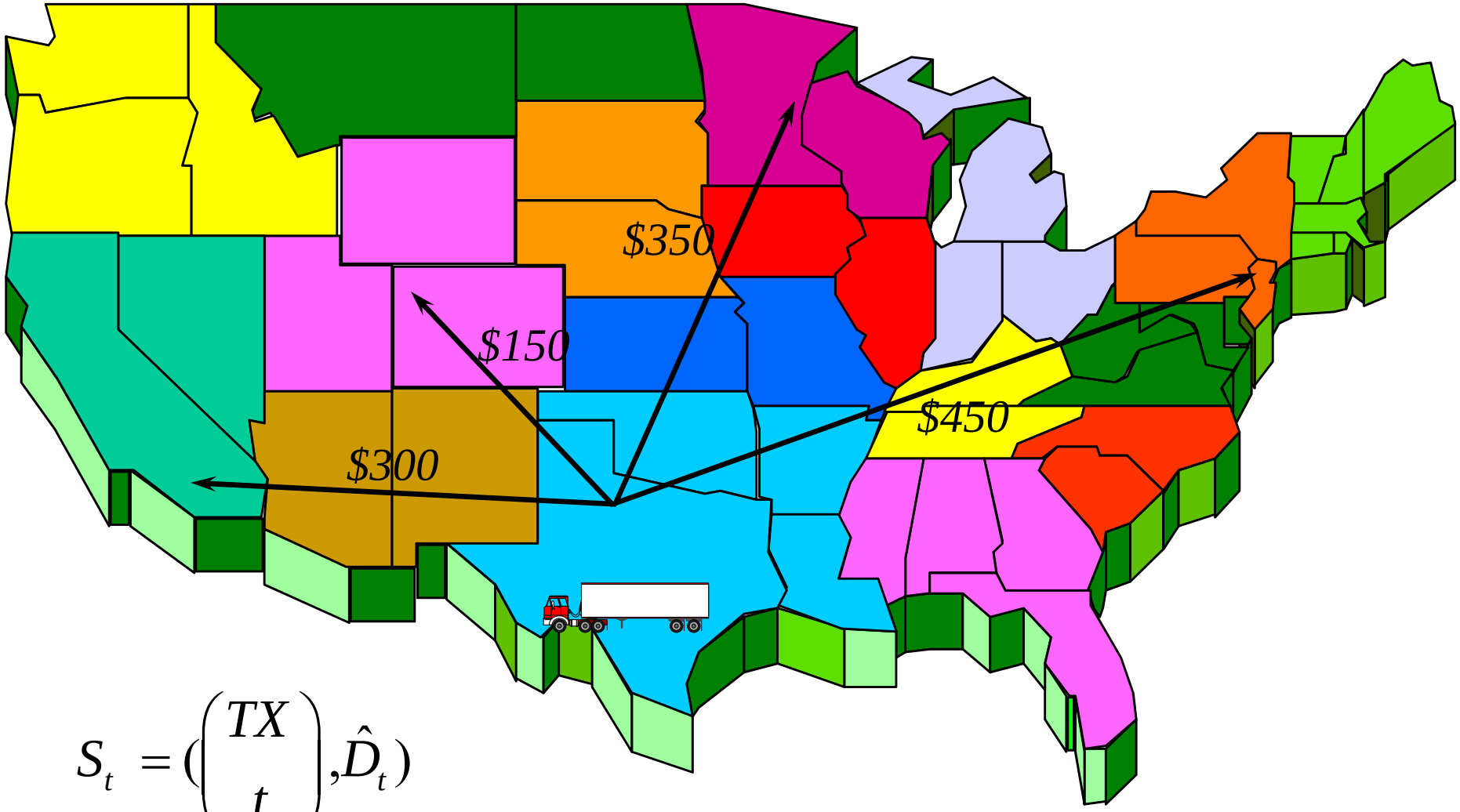
- Our optimization problem at time t looks like:

$$V_t(S_t) = \max_x \left(C_t(S_t, x_t) + \sum_{a \in \mathcal{A}} \overline{v}_{ta} \cdot R^x_{ta} \right)$$

  - » We had to develop novel machine learning strategies to estimate this function, since the attribute space was very large.

Nomadic trucker illustration

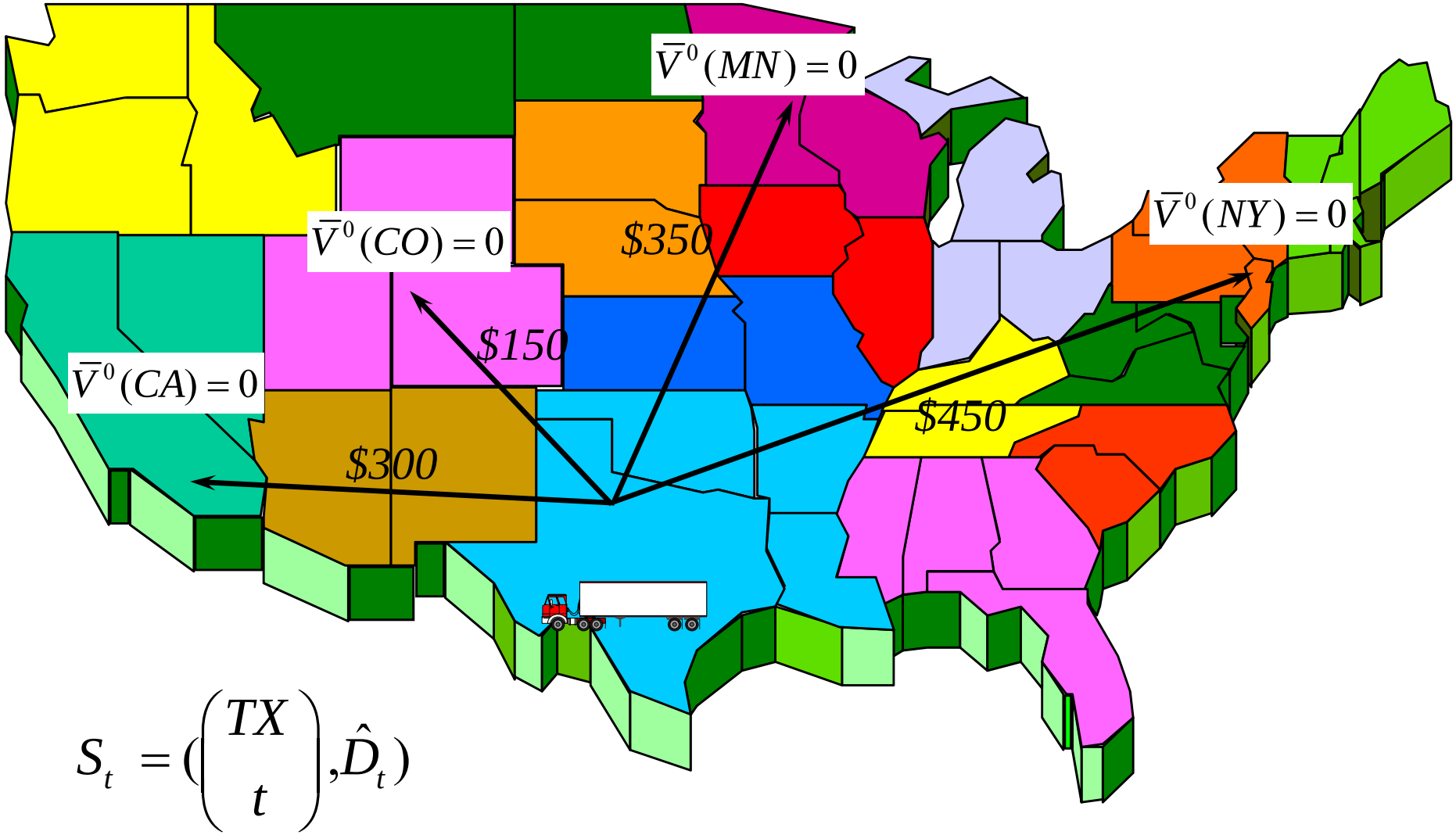
- Pre-decision state: we see the demands



$$S_t = \left(\begin{pmatrix} TX \\ t \end{pmatrix}, \hat{D}_t\right)$$

Nomadic trucker illustration

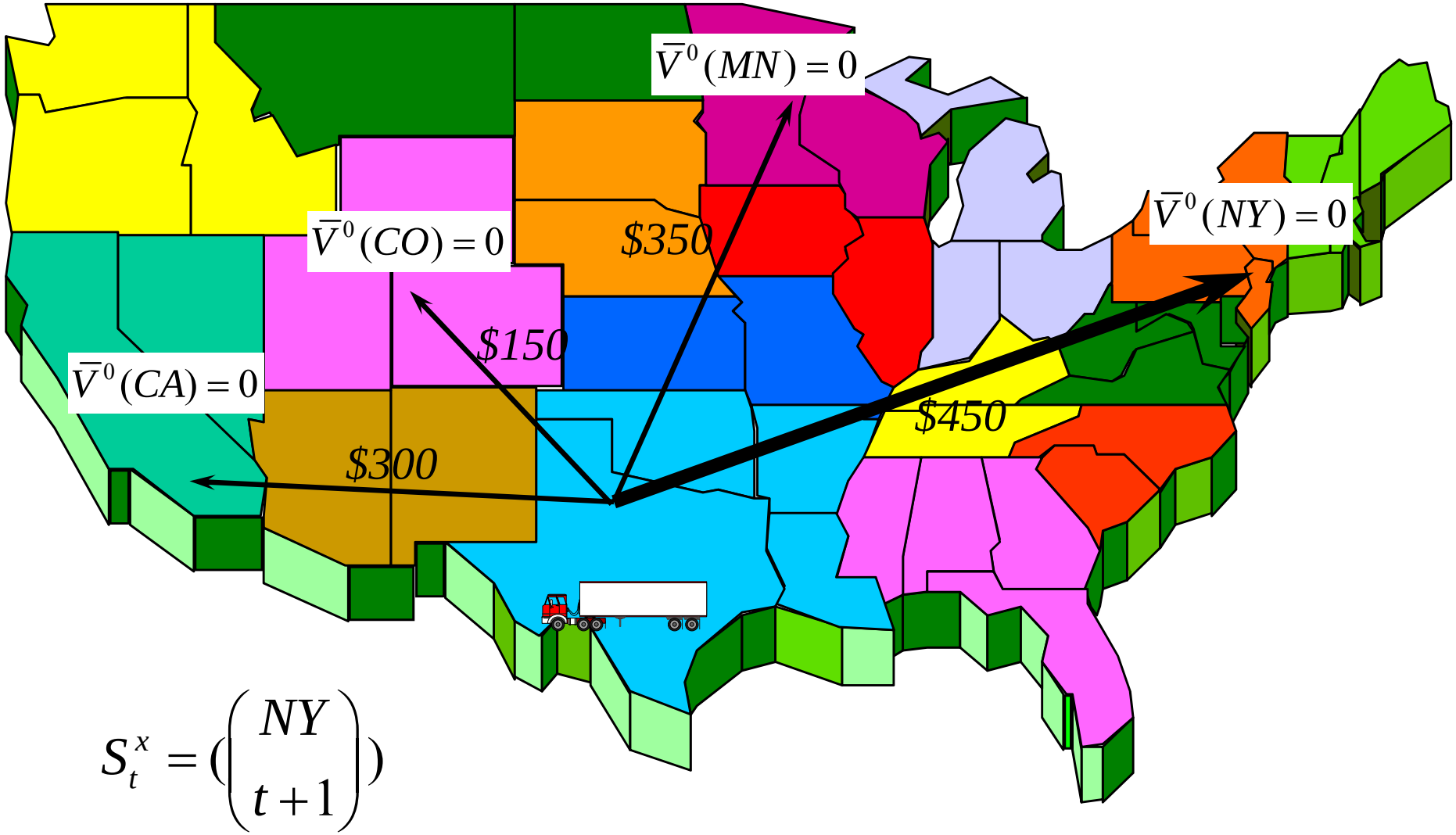
- We use initial value function approximations…



$$S_t = \left(\begin{pmatrix} TX \\ t \end{pmatrix}, \hat{D}_t\right)$$

Nomadic trucker illustration

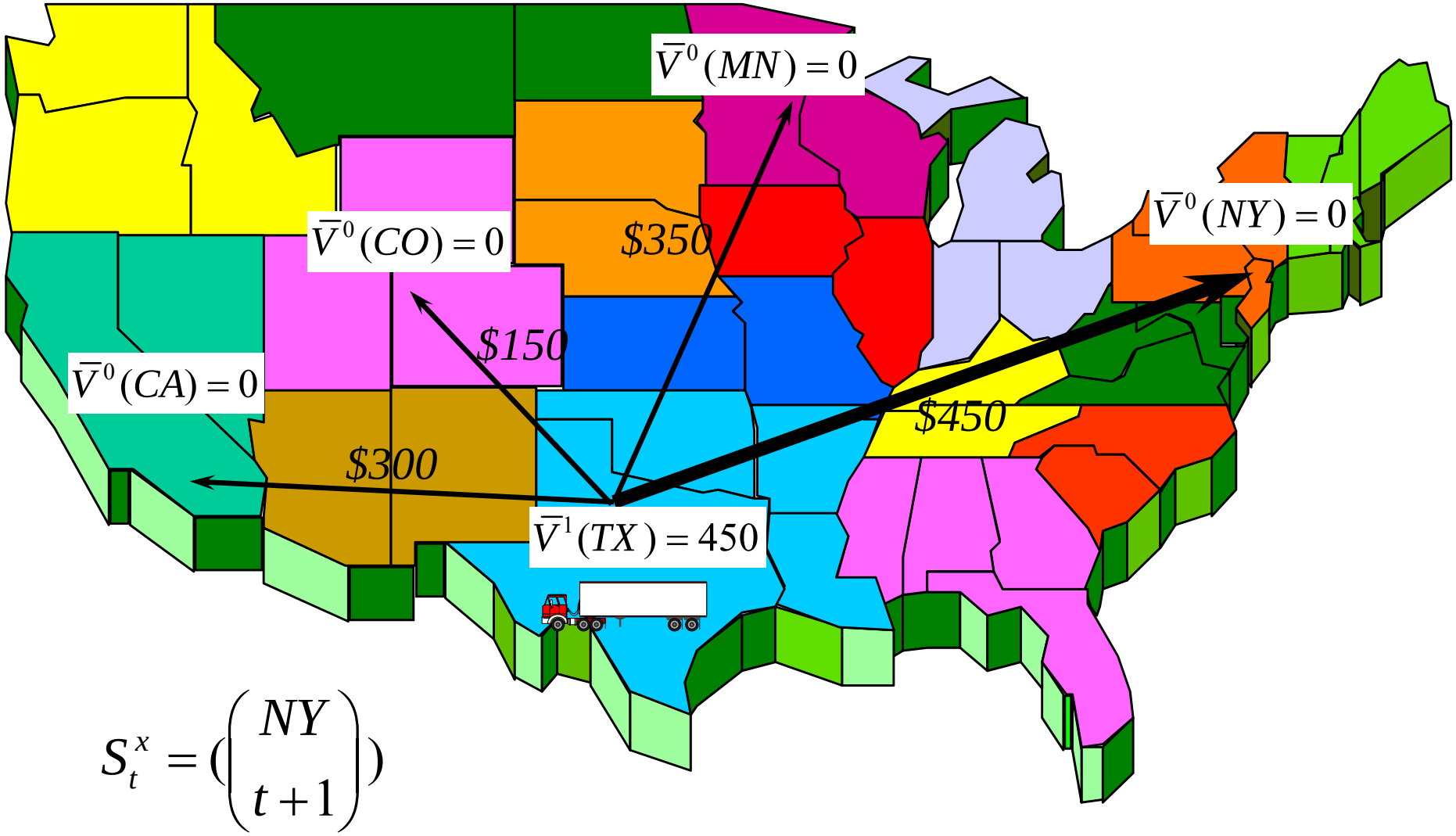
- … and make our first choice: x^1



$$S_t^x = \begin{pmatrix} NY \\ t+1 \end{pmatrix}$$

Nomadic trucker illustration

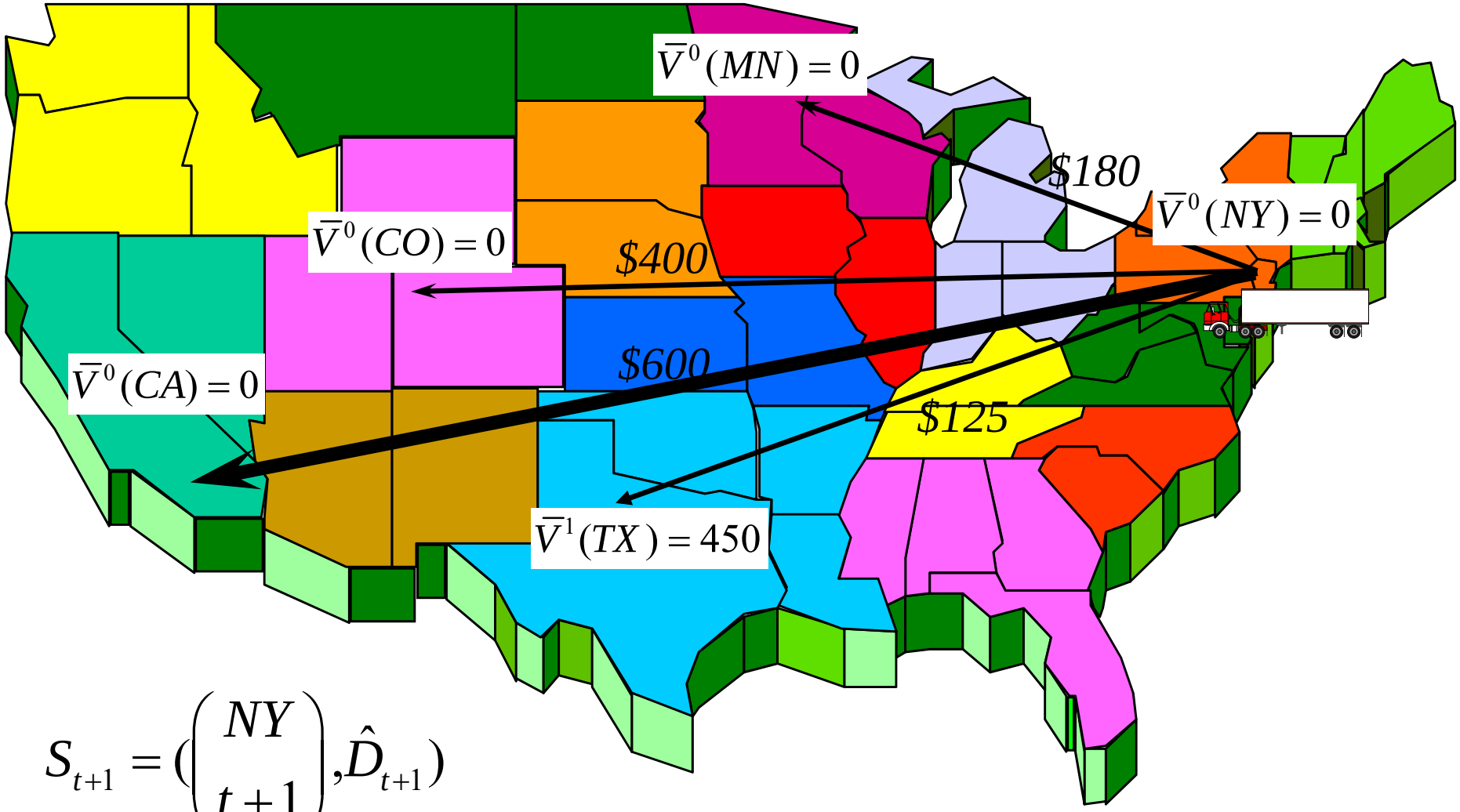
- Update the value of being in Texas.



$$S_t^x = \begin{pmatrix} NY \\ t+1 \end{pmatrix}$$

Nomadic trucker illustration

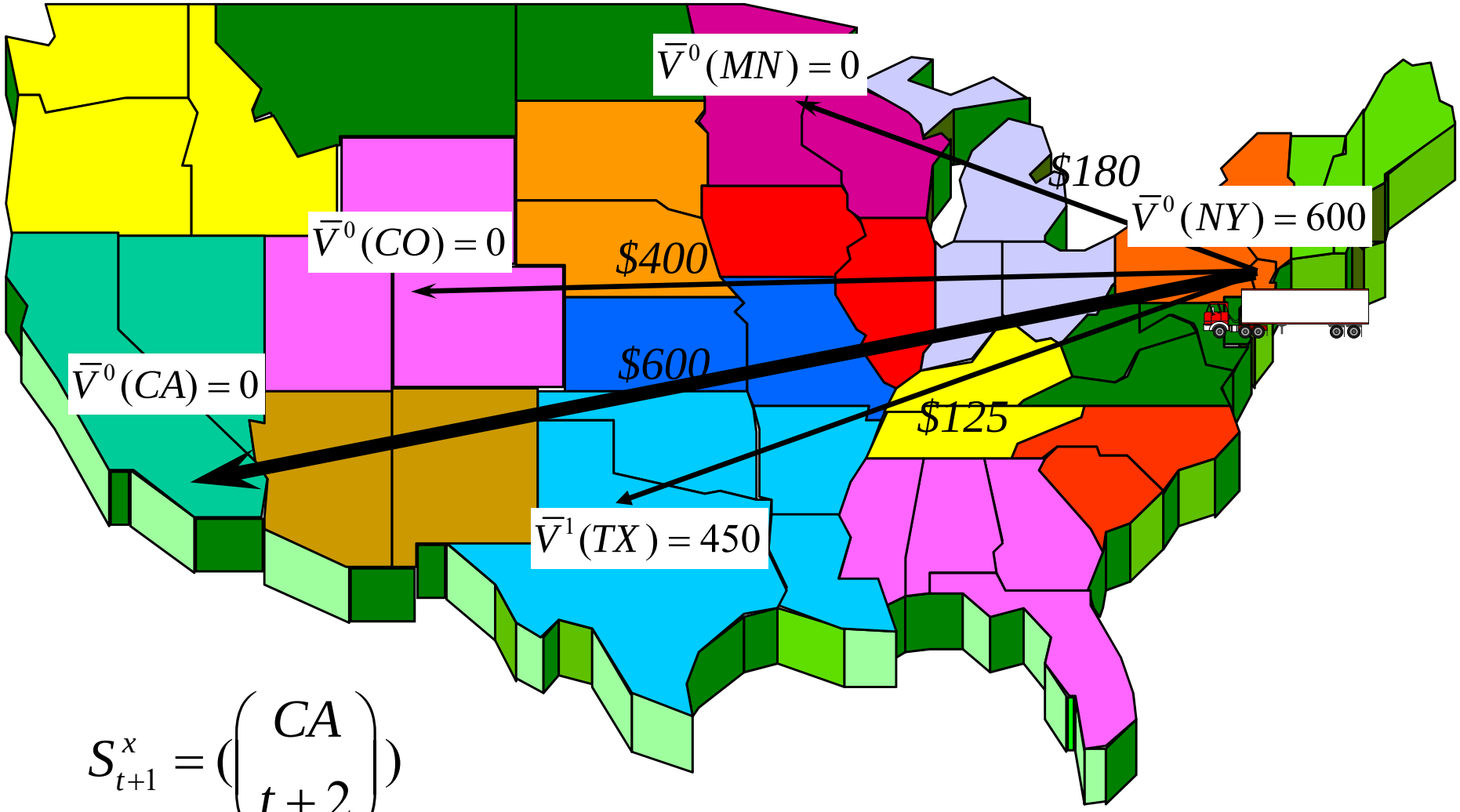
- Now move to the next state, sample new demands and make a new decision



$$S_{t+1} = \left(\begin{pmatrix} NY \\ t+1 \end{pmatrix}, \hat{D}_{t+1}\right)$$

Nomadic trucker illustration

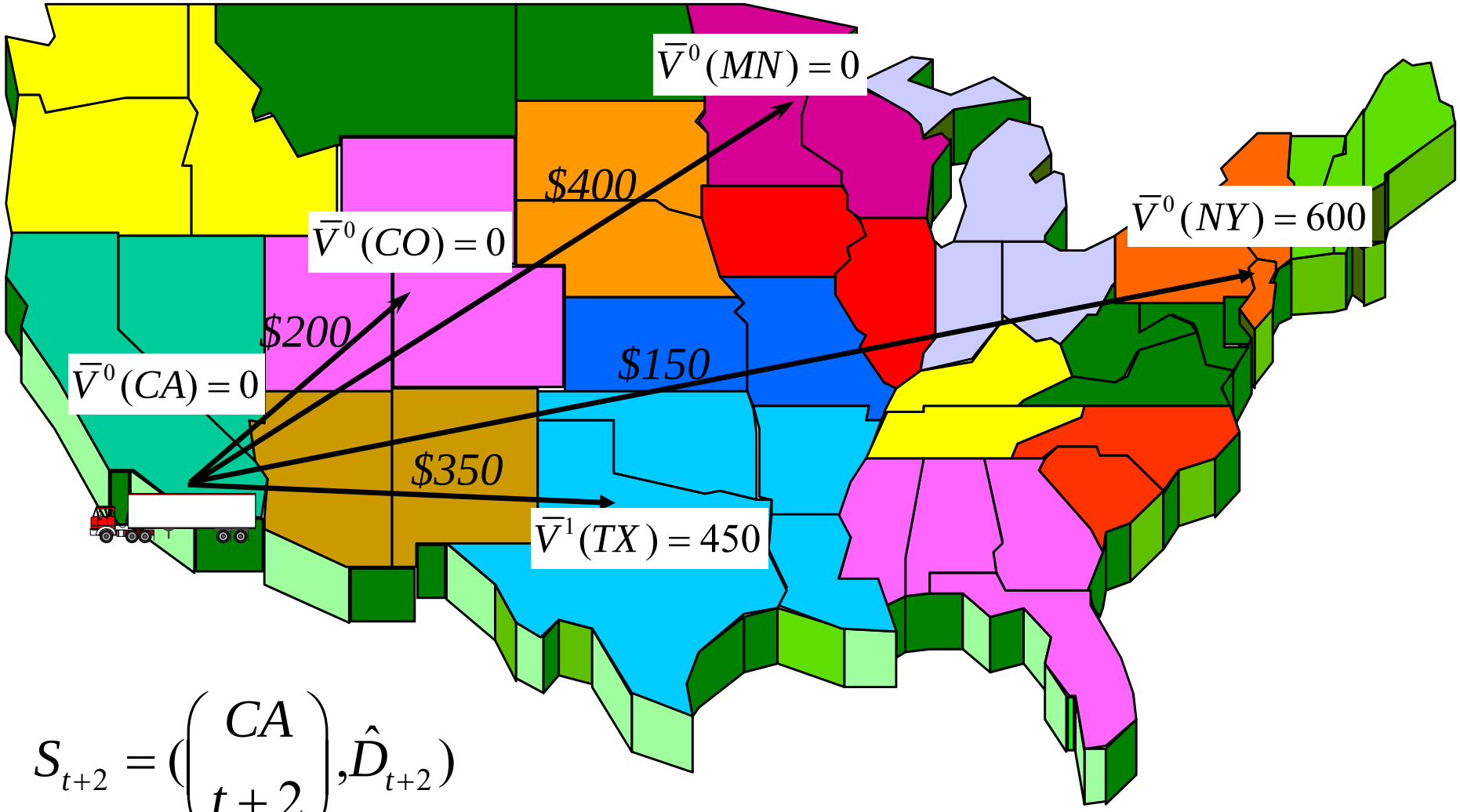
- Update value of being in NY



$$S_{t+1}^x = \begin{pmatrix} CA \\ t+2 \end{pmatrix}$$

Nomadic trucker illustration

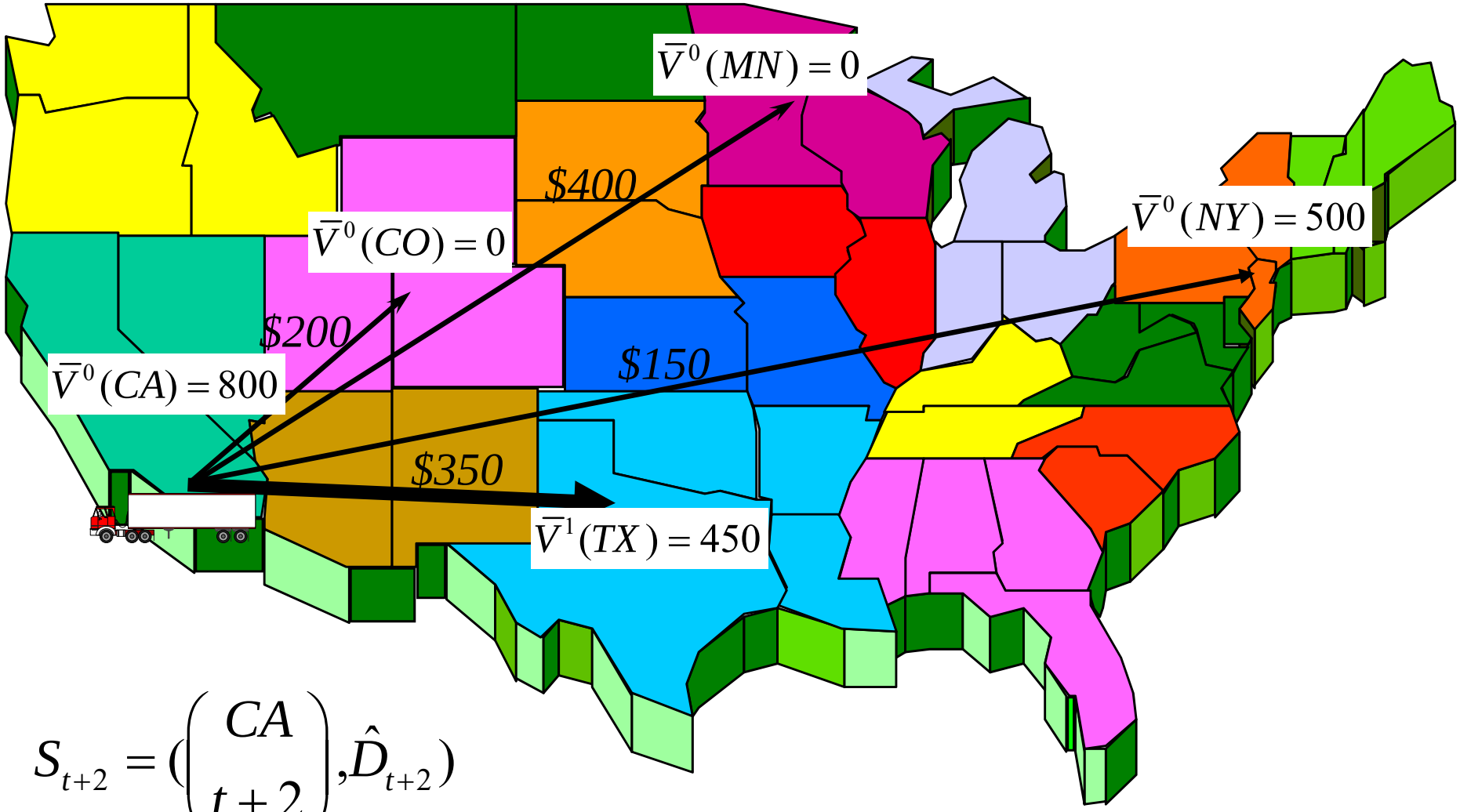
- Move to California.



$$S_{t+2} = \left(\begin{pmatrix} CA \\ t+2 \end{pmatrix}, \hat{D}_{t+2}\right)$$

Nomadic trucker illustration

- Make decision to return to TX and update value of being in CA



$$S_{t+2} = \left(\begin{pmatrix} CA \\ t+2 \end{pmatrix}, \hat{D}_{t+2}\right)$$

Nomadic trucker illustration

- Updating the value function:

Old value:

$$\bar{V}^1(TX) = \$450$$

New estimate:

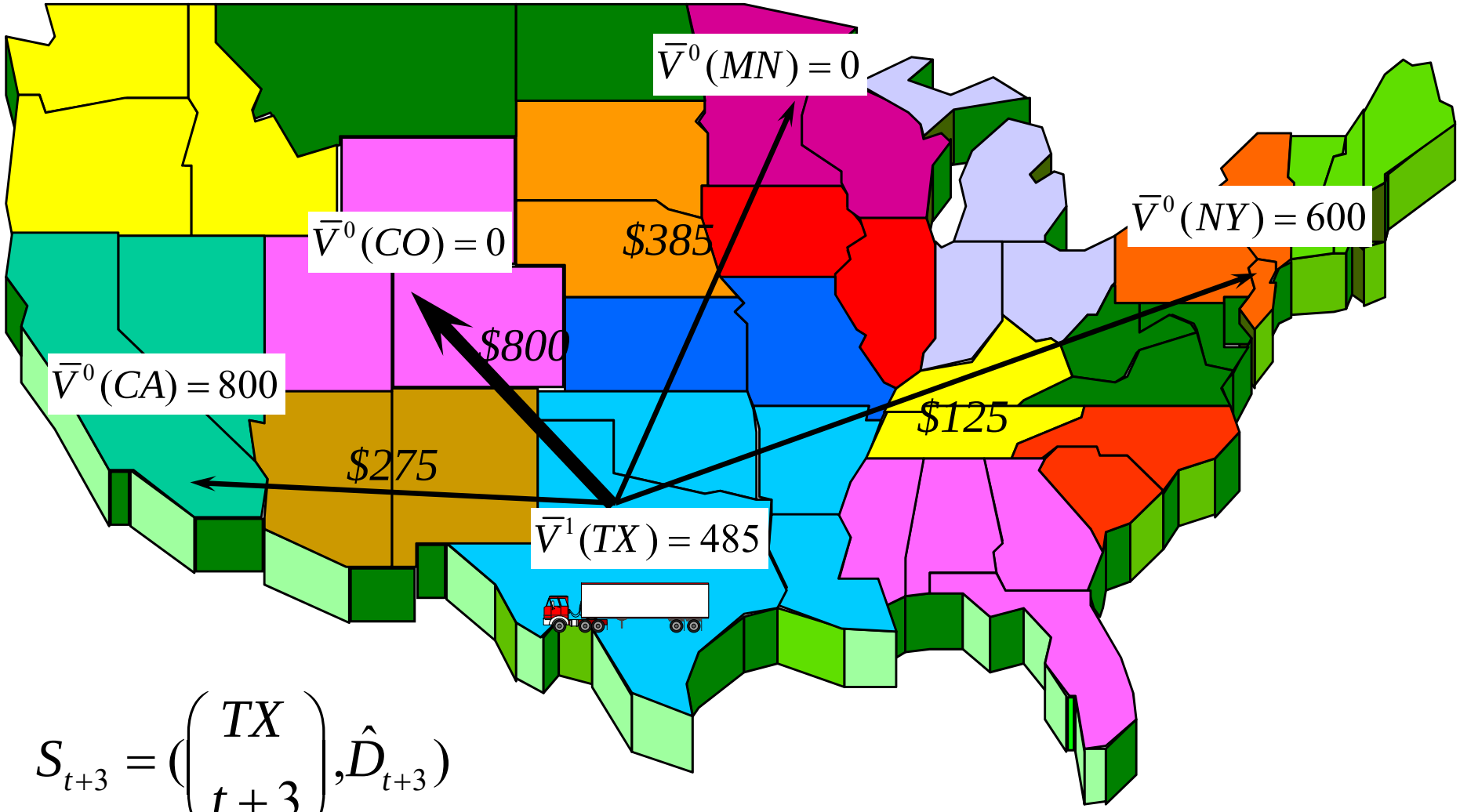
$$\hat{v}^2(TX) = \$800$$

How do we merge old with new?

$$\begin{aligned}\bar{V}^2(TX) &= (1-\alpha)\bar{V}^1(TX) + (\alpha)\hat{v}^2(TX) \\ &= (0.90)\$450 + (0.10)\$800 \\ &= \$485\end{aligned}$$

Nomadic trucker illustration

- An updated value of being in TX



$$S_{t+3} = \left(\begin{pmatrix} TX \\ t+3 \end{pmatrix}, \hat{D}_{t+3}\right)$$

- Notes:
  - » This is an example of *forward approximate dynamic programming*.
  - » We can execute this in two ways:
    - A pure forward pass (which is what we just showed) – You step forward in time, and as you do so, you obtain updated estimates of the value of being in a state based on downstream value function approximations. This is called *bootstrapping*.
    - A two-pass method – Here, you simulate forward using the value functions to define the policy, and then do a backward pass to update value functions (if you are familiar with SDDP, this is just what is done in SDDP).
  - » We are using a lookup table value function approximation (there are other options).

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:

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

to obtain x^n.

Deterministic optimization

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 $W_t(\omega^n)$ and compute the next pre-decision state:

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

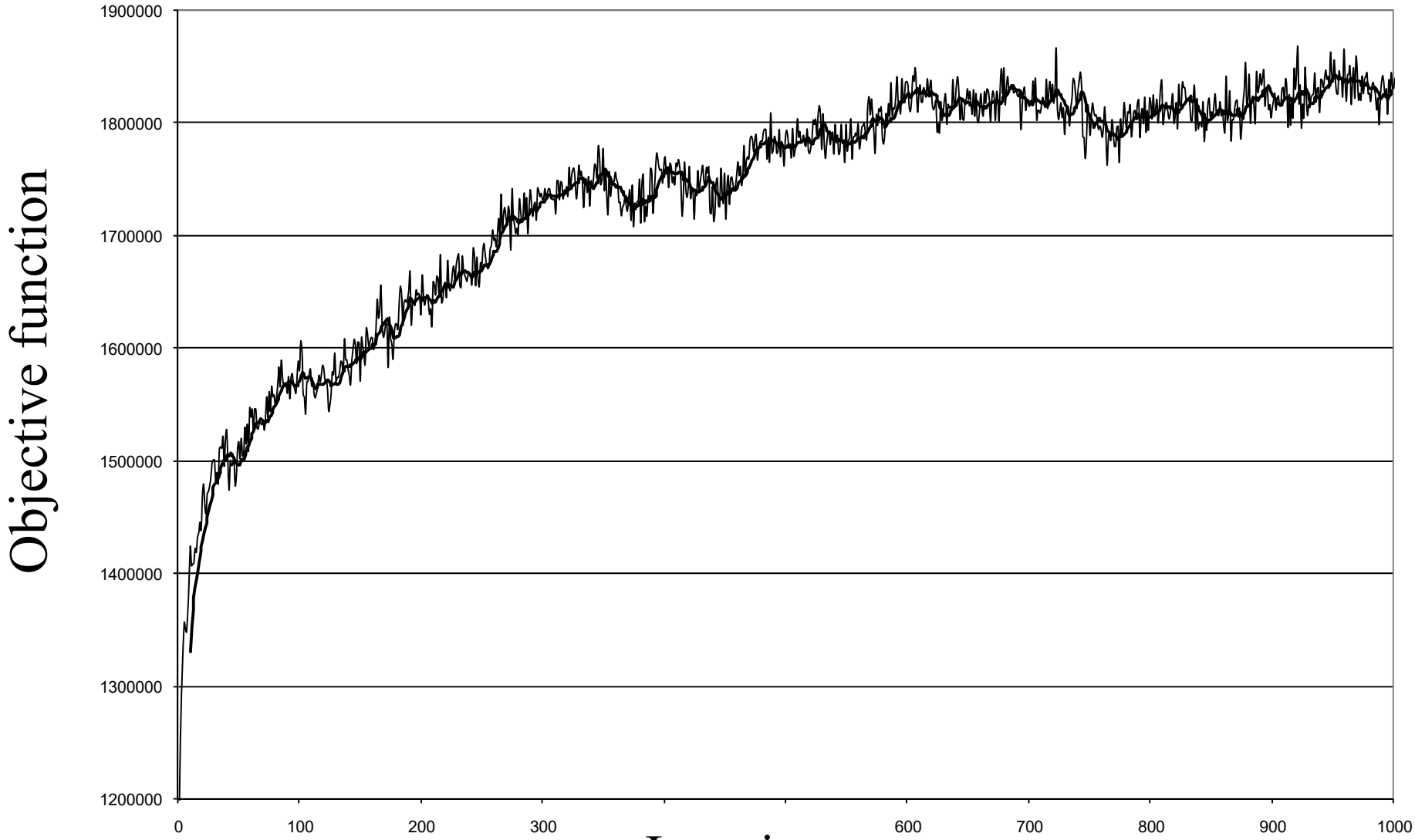
Simulation

Step 5: Return to step 1.

"on policy learning"

Approximate dynamic programming

- With luck, your objective function improves



- Notes:
  - » This method used a pure lookup table value function approximation, which works if the number of "states" is small.
  - » Even here, we have avoided the issue of *exploration*. You may need to visit a state (e.g. "Minnesota") just to learn about this.
  - » The need to explore is reduced if you use a statistical model of the value function that generalizes learning. E.g. visiting Connecticut (a state in the northeastern U.S.) teaches us something about other states in the northeast.

Lookahead policies

Direct lookaheads

- Lookahead policies:
  - » Lookahead policies are based on some sort approximation of the impact of a decision now on the future.
  - » Lookahead policies have two general characteristics:
    - They require little or no tuning.
    - They are almost always harder to compute than the PFA/CFA policies, and are sometimes much harder.
    - … but tuning can be hard too.

Designing policies

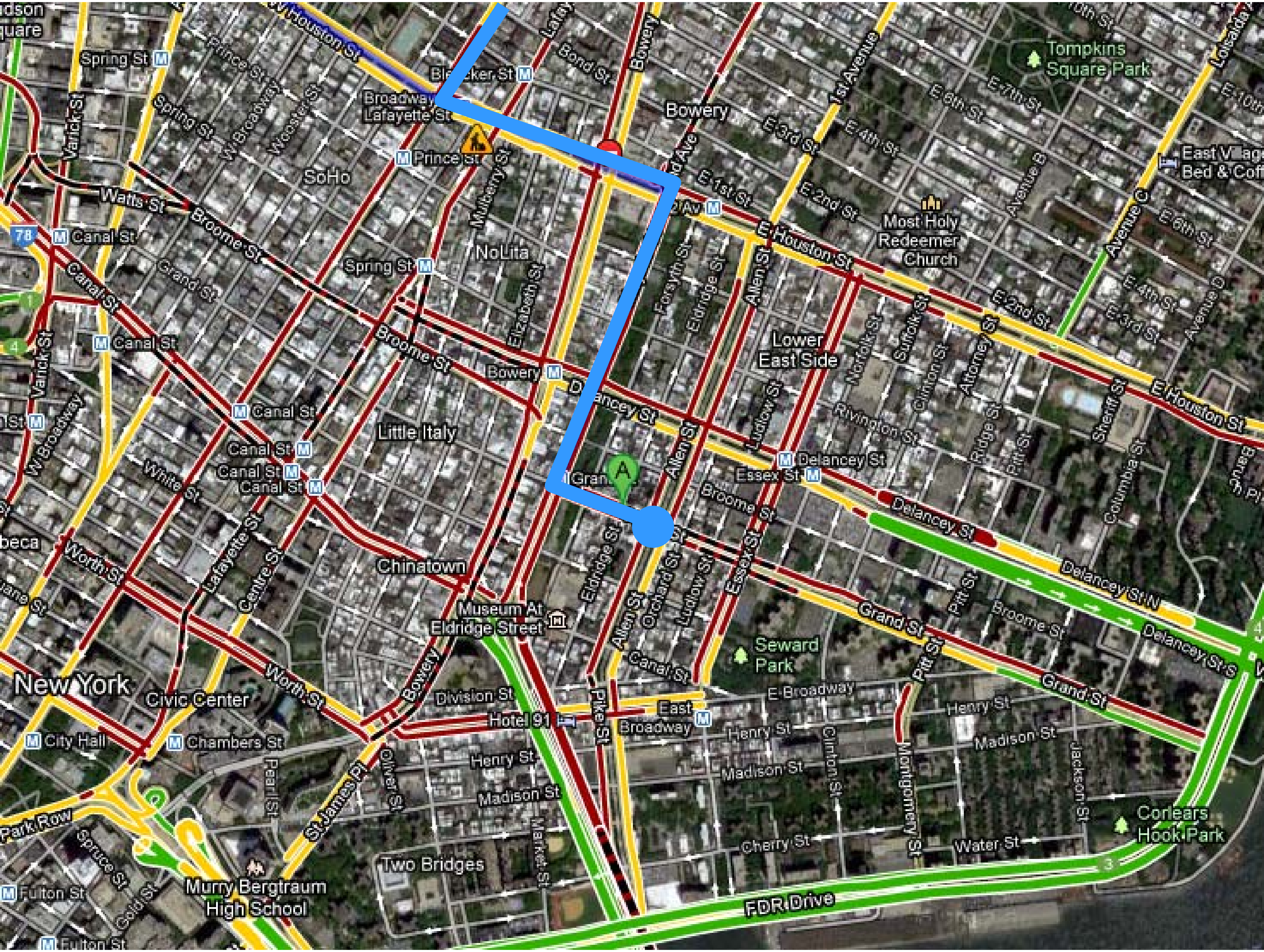
- The ultimate lookahead policy is optimal

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

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

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

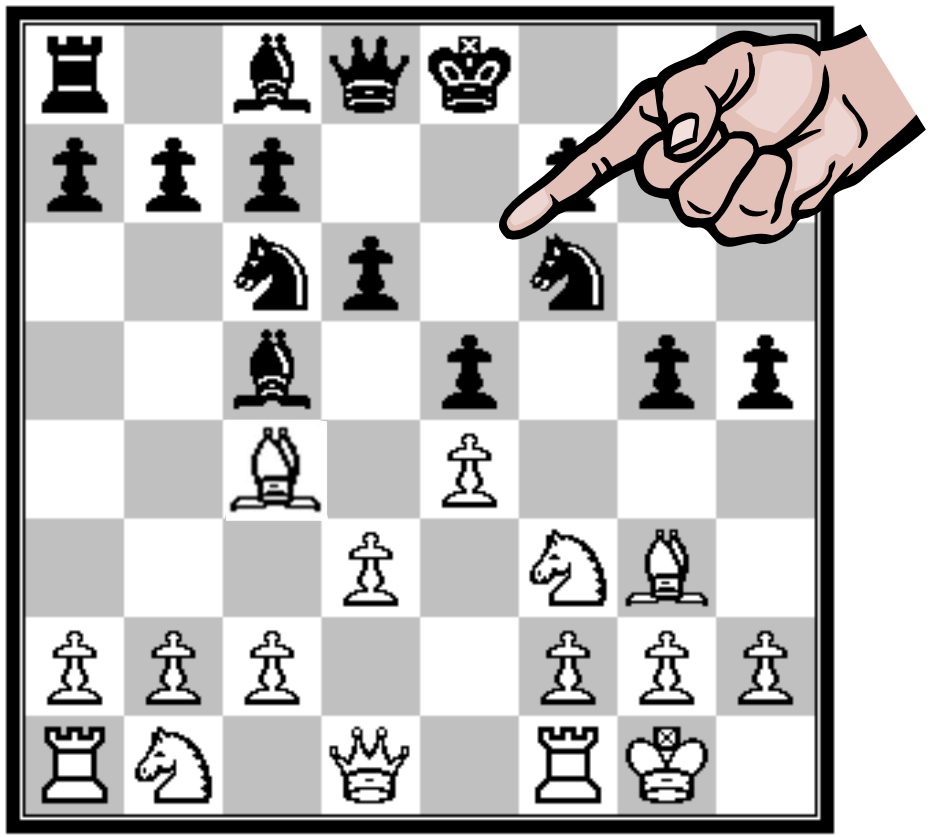
» *A lookahead policy* works by approximating the *lookahead model*.



Spring St
Bleecker St
Broadway Lafayette St
Prince St
Bowery
SoHo
NoLita
Little Italy
Chinatown
Lower East Side
Tompkins Square Park
Most Holy Redeemer Church
E Houston St
Delancey St
Essex St
Canal St
Grand St
Broome St
Watts St
Worth St
Museum At Eldridge Street
Seward Park
East Broadway
Division St
Hotel 91
Henry St
Madison St
Cherry St
Water St
Corlears Hook Park
FDR Drive
Two Bridges
Murry Bergtraum High School
New York
Civic Center
City Hall
Chambers St
Fulton St
Park Row
St James Pl
Pike St
Market St
Oliver St
Pearl St
Centre St
Lafayette St
Mulberry St
Elizabeth St
Forsyth St
Eldridge St
Allen St
Orchard St
Ludlow St
Essex St
Norfolk St
Suffolk St
Clinton St
Attorney St
Ridge St
Pitt St
Sheriff St
Columbia St
Montgomery St
Jackson St
Rivington St
E 1st St
E 2nd St
E 3rd St
E 4th St
E 6th St
E 7th St
1st Avenue
Avenue B
Avenue C
Avenue D
Varick St
W Broadway
Wooster St
Bond St
White St
Tribeca

Lookahead policies

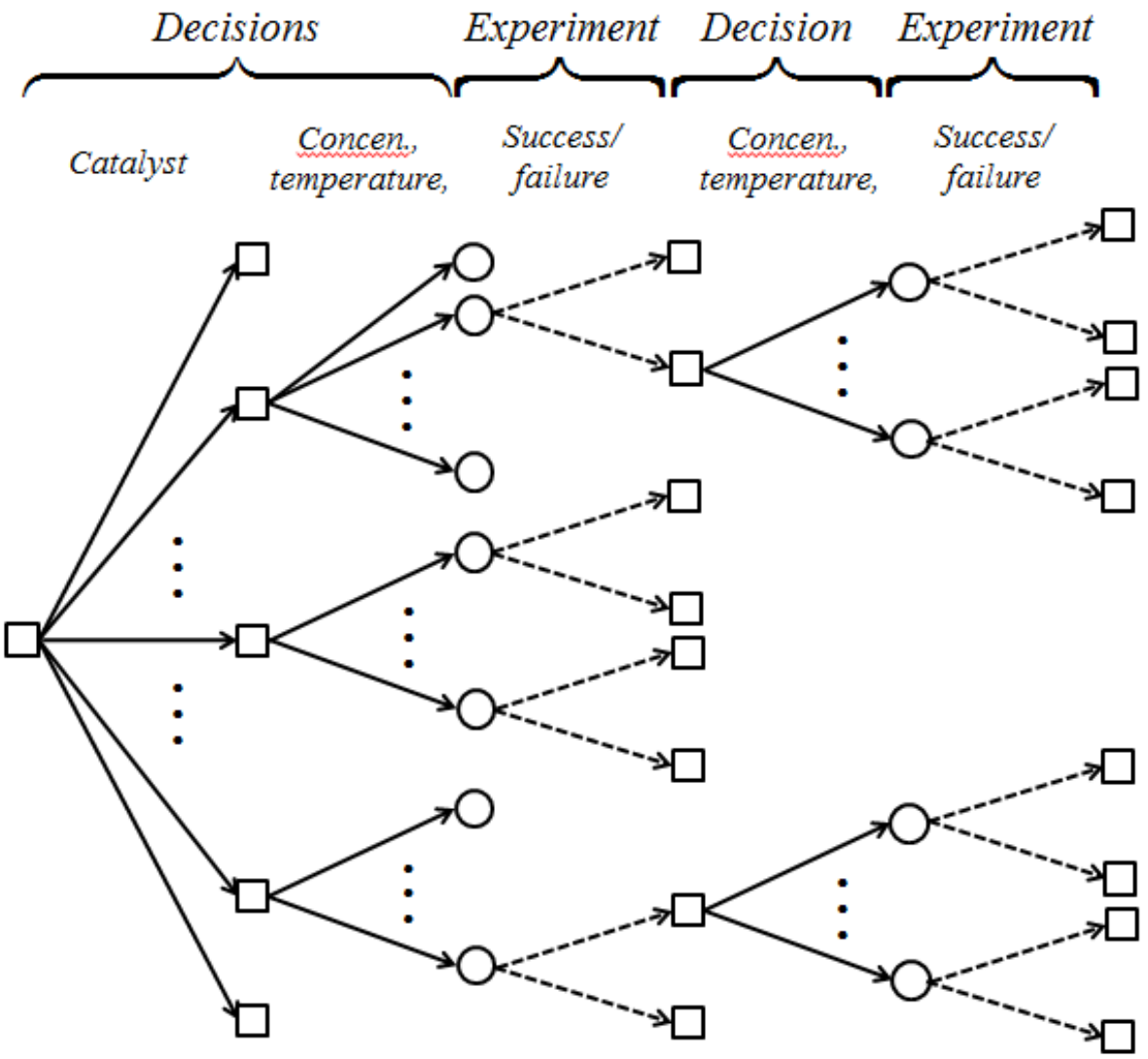
- Planning your next chess move:



» You put your finger on the piece while you think about moves into the future. This is a lookahead policy, illustrated for a problem with discrete actions.

Lookahead policies

- Decision trees:



Lookahead policies

- Lookahead models use five classes of approximations:
  - » Horizon truncation – Replacing a longer horizon problem with a shorter horizon
  - » Stage aggregation – Replacing multistage problems with two-stage approximation.
  - » Outcome aggregation/sampling – Simplifying the exogenous information process
  - » Discretization – Of time, states and decisions
  - » Dimensionality reduction – We may ignore some variables (such as forecasts) in the lookahead model that we capture in the base model (these become *latent* variables in the lookahead model).

Lookahead policies

- Lookahead policies are the trickiest to model:
  - » We create "tilde variables" for the lookahead model:

$\tilde{S}_{t,t'}$ = Approximated state variable (e.g coarse discretization)

$\tilde{x}_{t,t'}$ = Decision we plan on implementing at time t' when we are planning at time t, $t' = t, t+1, \ldots, t+H$

$\tilde{x}_t = \left(\tilde{x}_{t,t}, \tilde{x}_{t,t+1}, \ldots, \tilde{x}_{t,t+H}\right)$

$\tilde{W}_{t,t'}$ = Approximation of information process

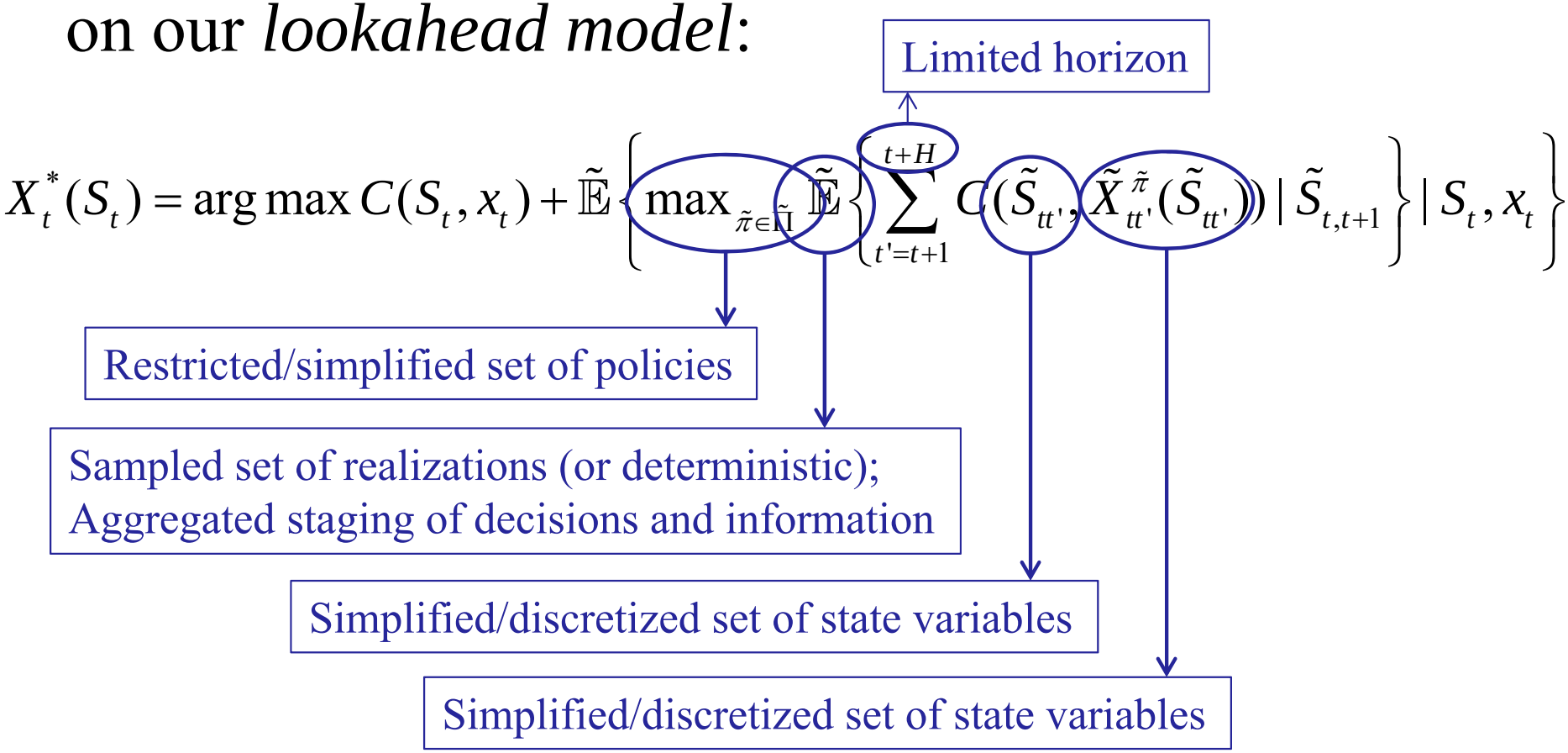
$\tilde{c}_{t,t'}$ = Forecast of costs at time t' made at time t

$\tilde{b}_{t,t'}$ = Forecast of right hand sides for time t' made at time t

  - » All variables are indexed by t (when the lookahead model is being generated) and t' (the time within the lookahead model).

Lookahead policies

- We can use this notation to create a policy based on our *lookahead model*:



» Simplest lookahead is deterministic.

Lookahead policies

- Deterministic lookahead

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

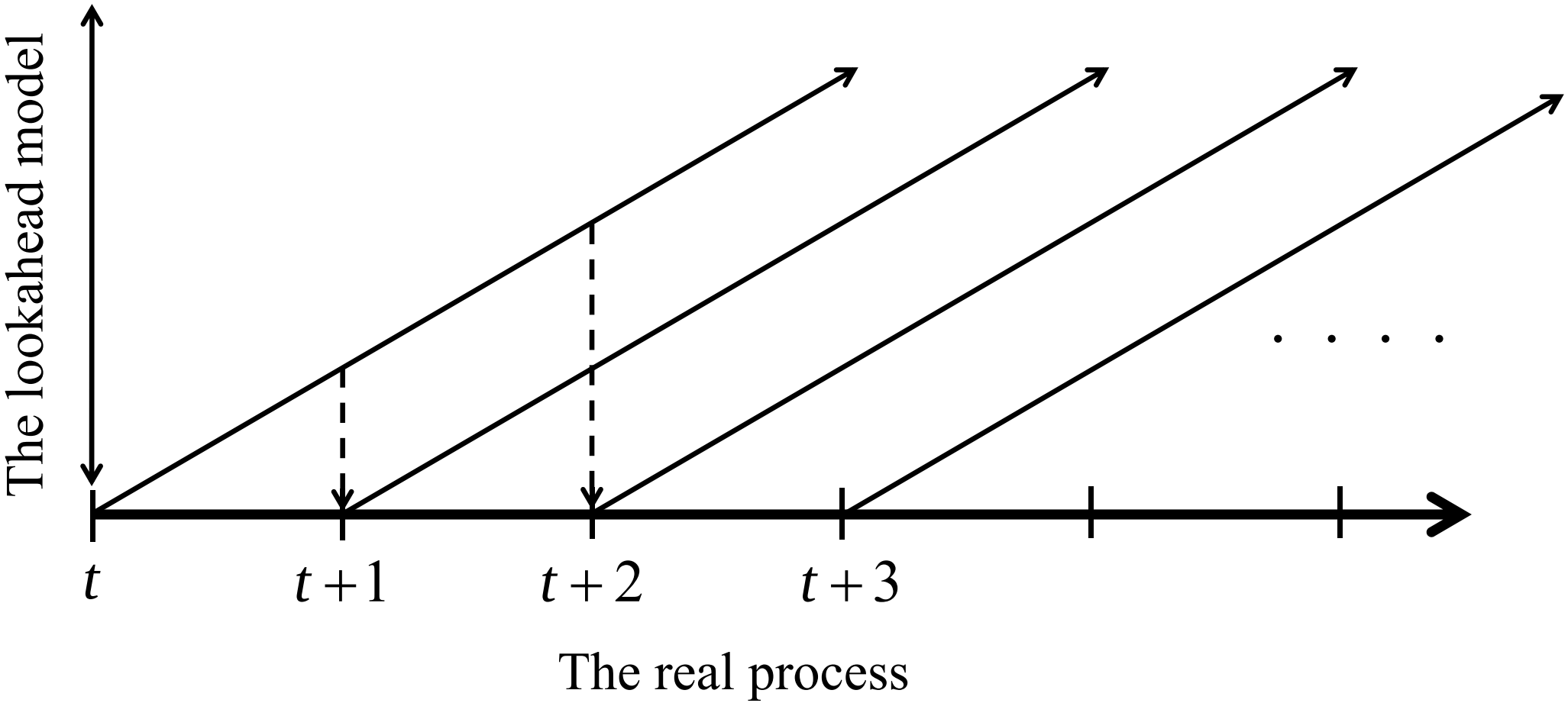
- Stochastic lookahead (with two-stage approximation)

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

Scenario trees

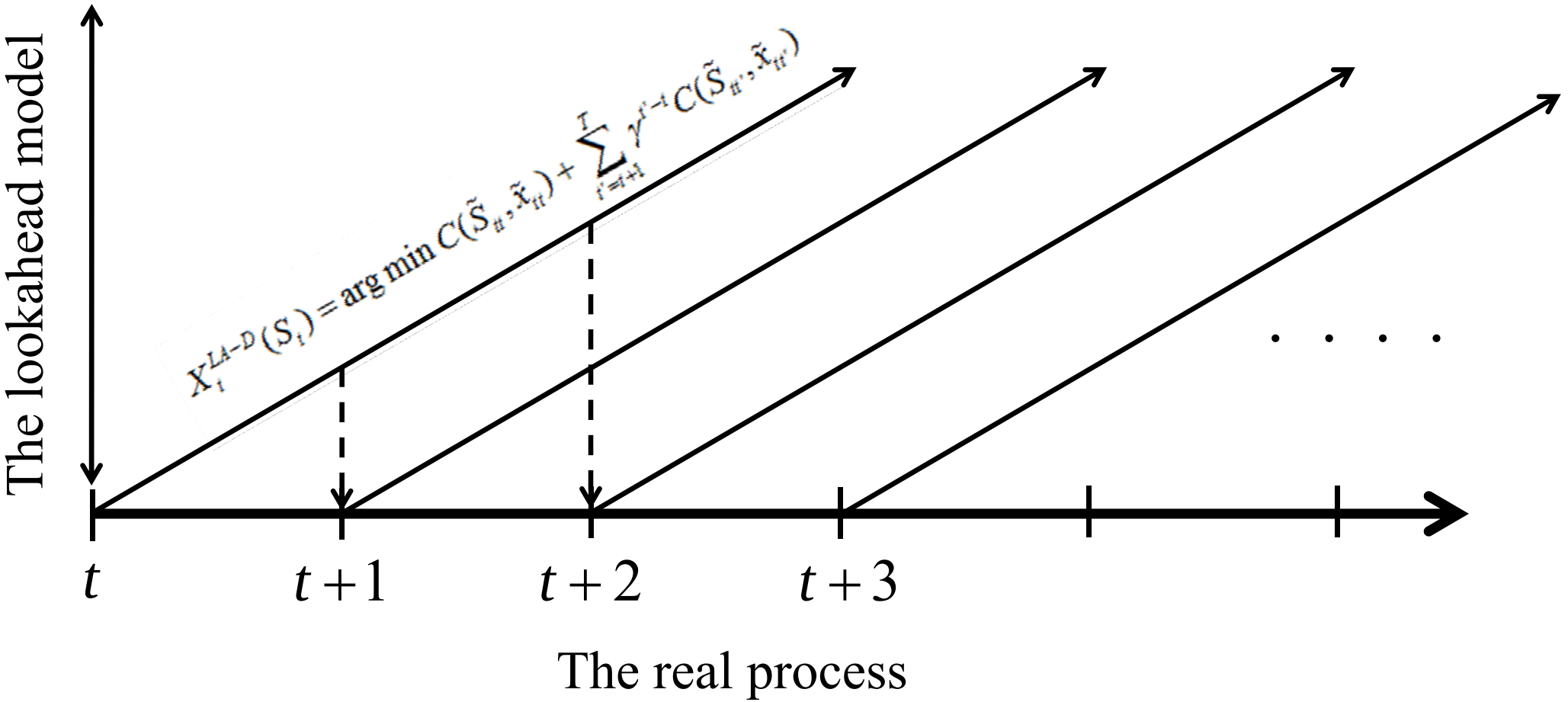
Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



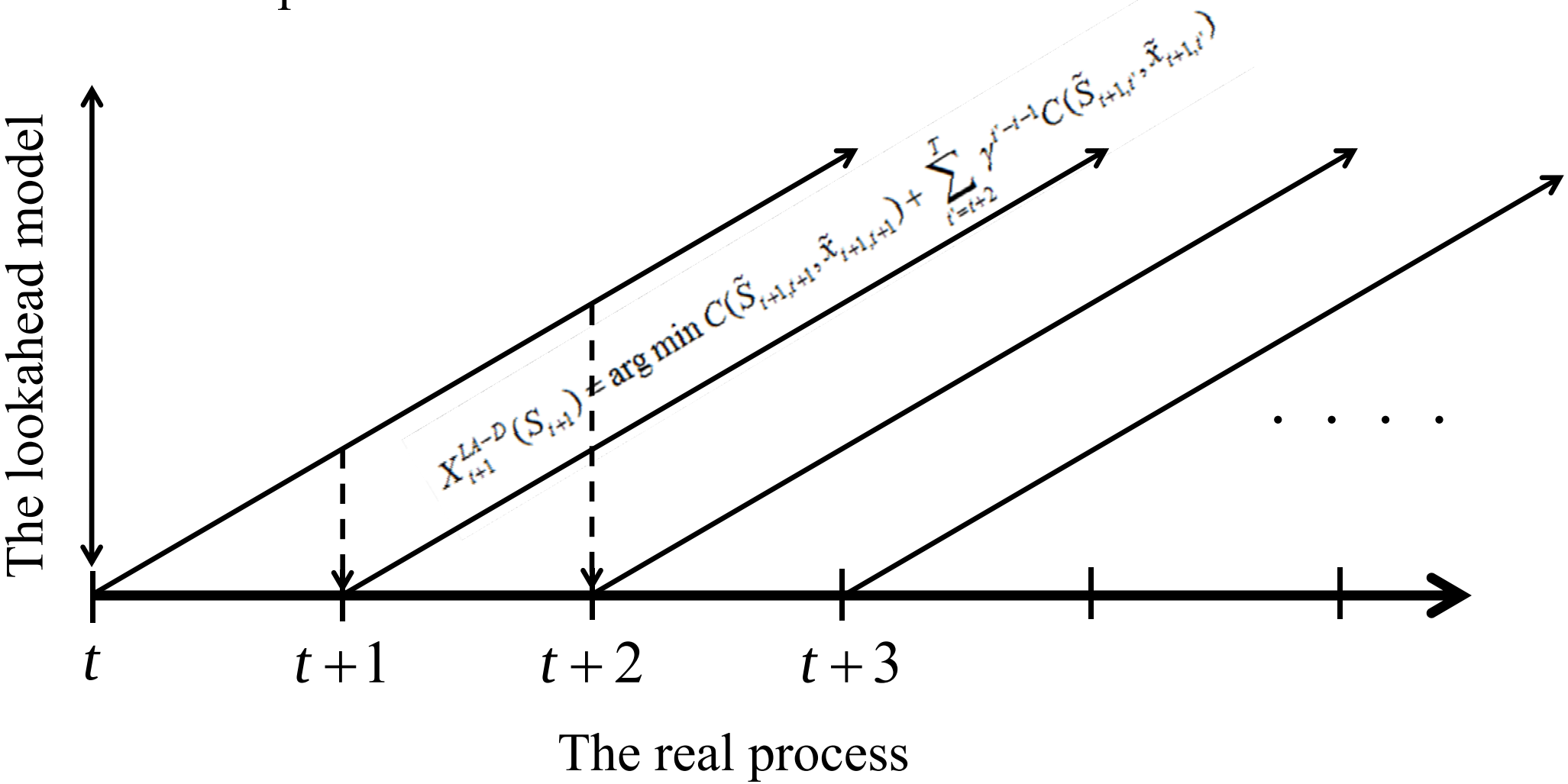
Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



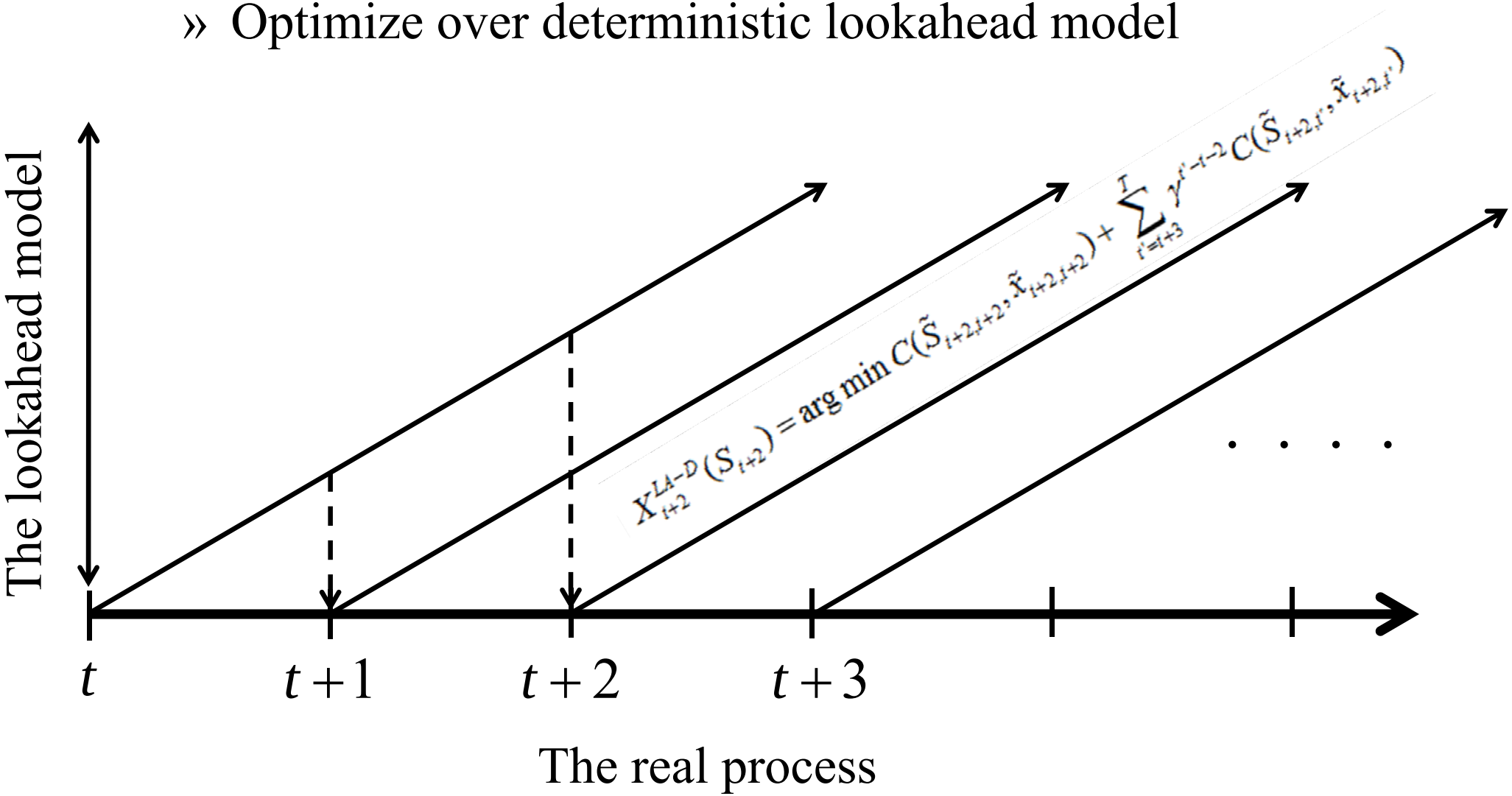
Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



Designing policies

- Types of lookahead approximations
  - » One-step lookahead – Widely used in pure learning policies:
    - Bayes greedy/naïve Bayes
    - Thompson sampling
    - Value of information (knowledge gradient)
  - » 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.

Choosing policies

Modeling sequential decision problems

- **First build your model**
  - » Objective function

$$\max_{\pi} E\left\{\sum_{t=0}^{T}\gamma^{t}C\left(S_t, X_t^{\pi}(S_t)\right)|\, S_0\right\}$$

  - » Policy

$$X^{\pi}: S \mapsto \mathcal{X}$$

  - » Constraints at time t are built into the policy

$$x_t = X_t^{\pi}(S_t) \in \mathcal{X}_t = \text{feasible region}$$

  - » Transition function captures behavior over time

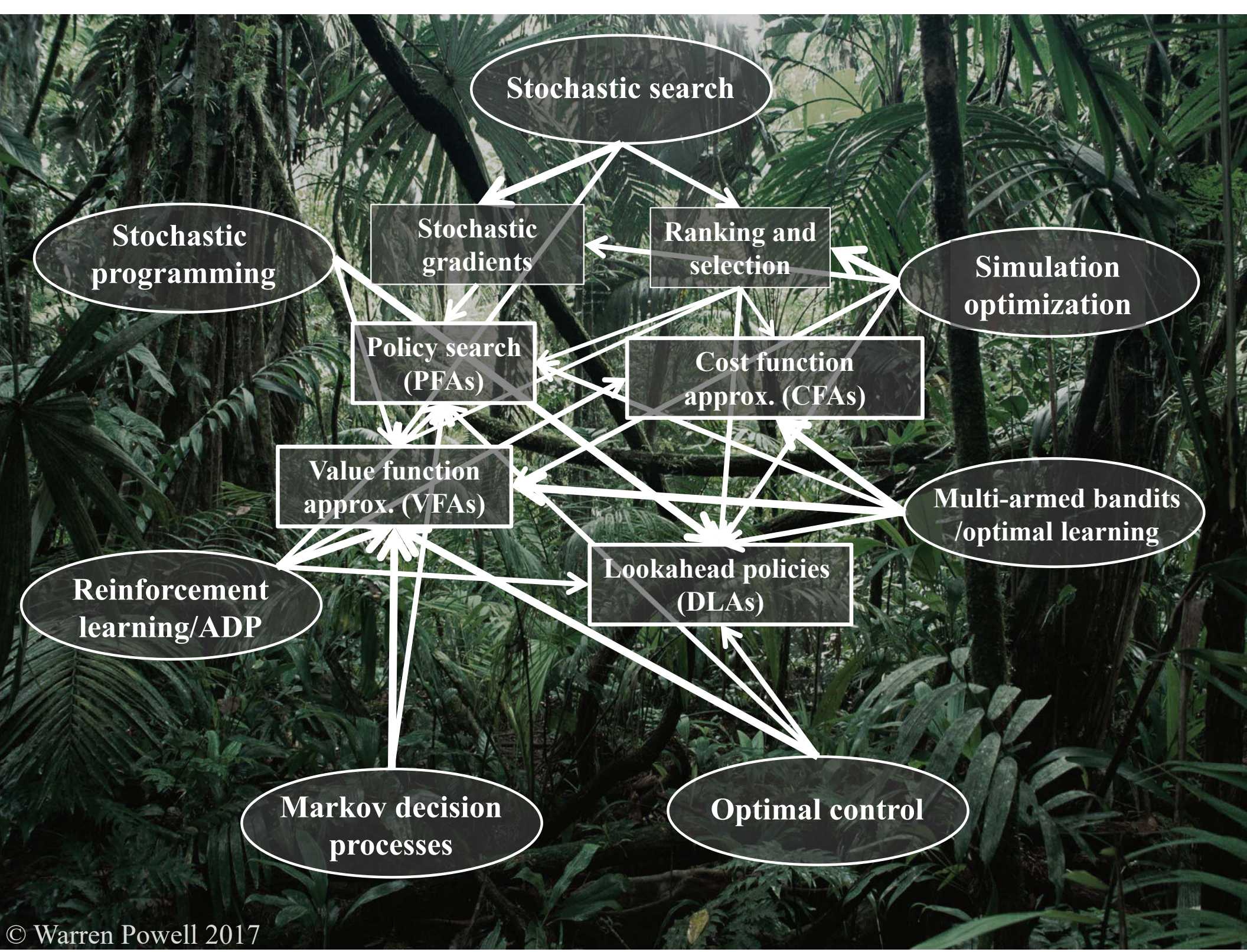
$$S_{t+1} = S^{M}\left(S_t, x_t, W_{t+1}\right)$$

  - » Exogenous information (including initial state).

$$(S_0, W_1, W_2, \ldots, W_T)$$

- Designing policies
  - » Does the way of making decisions seem to have a natural structure? Suggests PFA.
    - E.g. buy low, sell high
    - This can lead to a natural parameterization
  - » For more complex problems (e.g. shortest paths) does there appear to be an obvious parameterization that leads to more robust decisions? Suggests CFA
    - Allowing extra time when planning a trip
    - Maintaining minimum inventories when allocating supplies for emergencies
  - » Does the value of being in a state in the future have an obvious structure that can be exploited? Suggests VFA
    - The value of being at node
  - » Is there information about the future that we need to use? Suggests DLA
    - Forecasts of travel times, demands, prices

- Issues in designing policies
  - » Simplicity
    - Everyone likes a simpler function over a more complicated one. It is easier to understand and implement.
    - Simpler policies are more transparent.
    - Lower risk that you can get it to work "well enough."
  - » Computational complexity
    - Google limits policies to 50 milliseconds.
    - For a grid operator, the limit of 3 hours of CPU time solve a large integer program limits their options for handling uncertainty.
  - » Data requirements
    - A properly tuned policy may work well over time without requiring all the data needed for a lookahead model.
  - » Robustness
    - Avoiding the risk of failures is a major issue.
  - » Performance
    - And of course, everyone wants a policy where they do better.



Week 5 – Monday

Energy storage I

Presentation on energy storage in Brazil

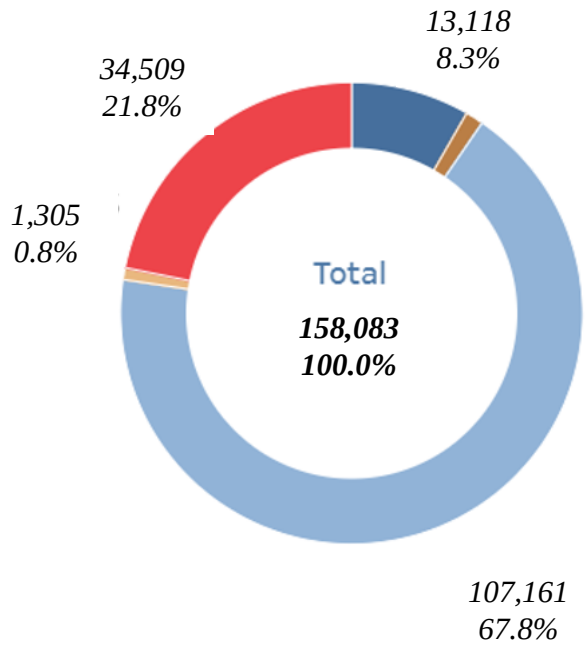
The Brazilian Electricity Sector

Energy mix, grid operation and challenges for integrating intermittent sources to the system

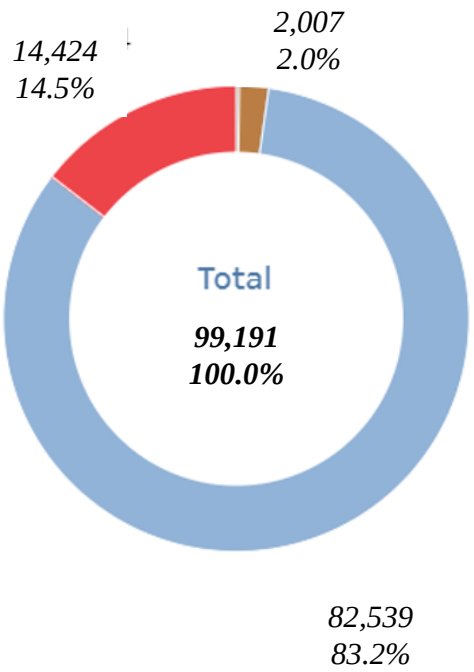
Brazilian Energy Mix

Brazilian Energy Mix

Capacity (MW) per type – August/2018

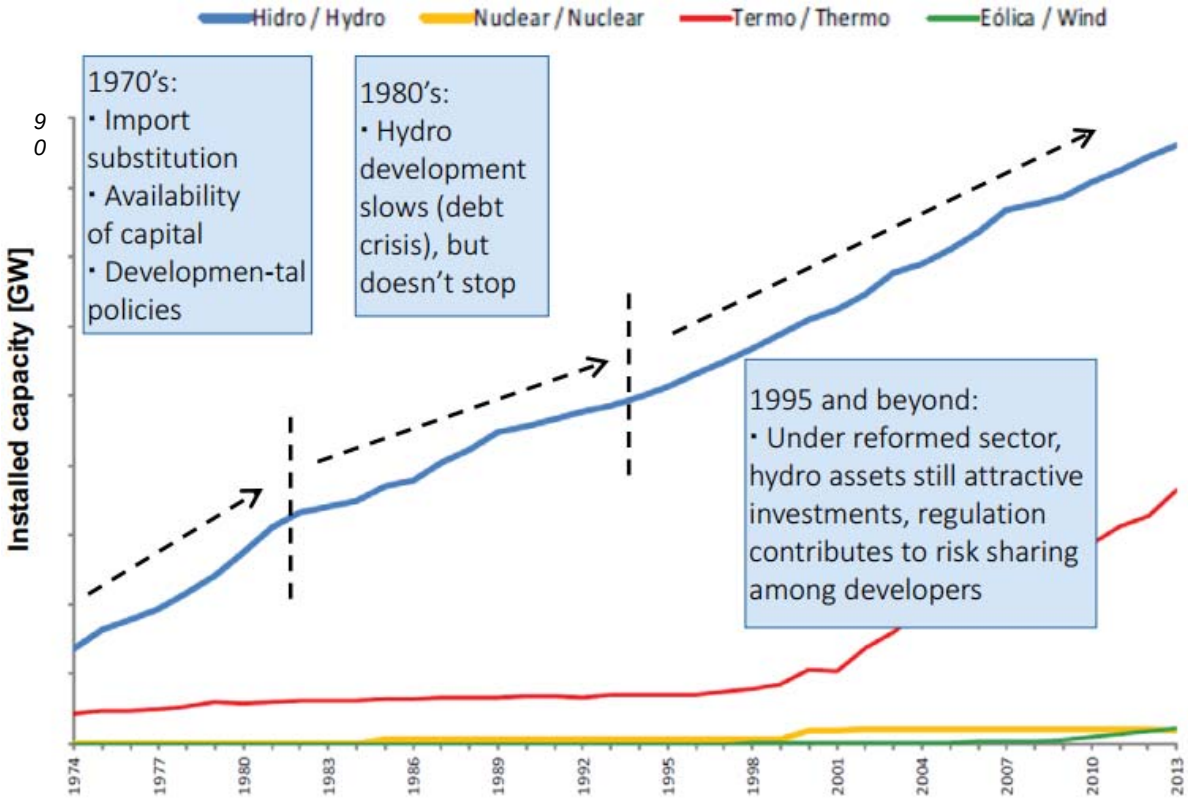


Capacity (MW) per type – August/2008



Source: Adapted from ONS (2018).

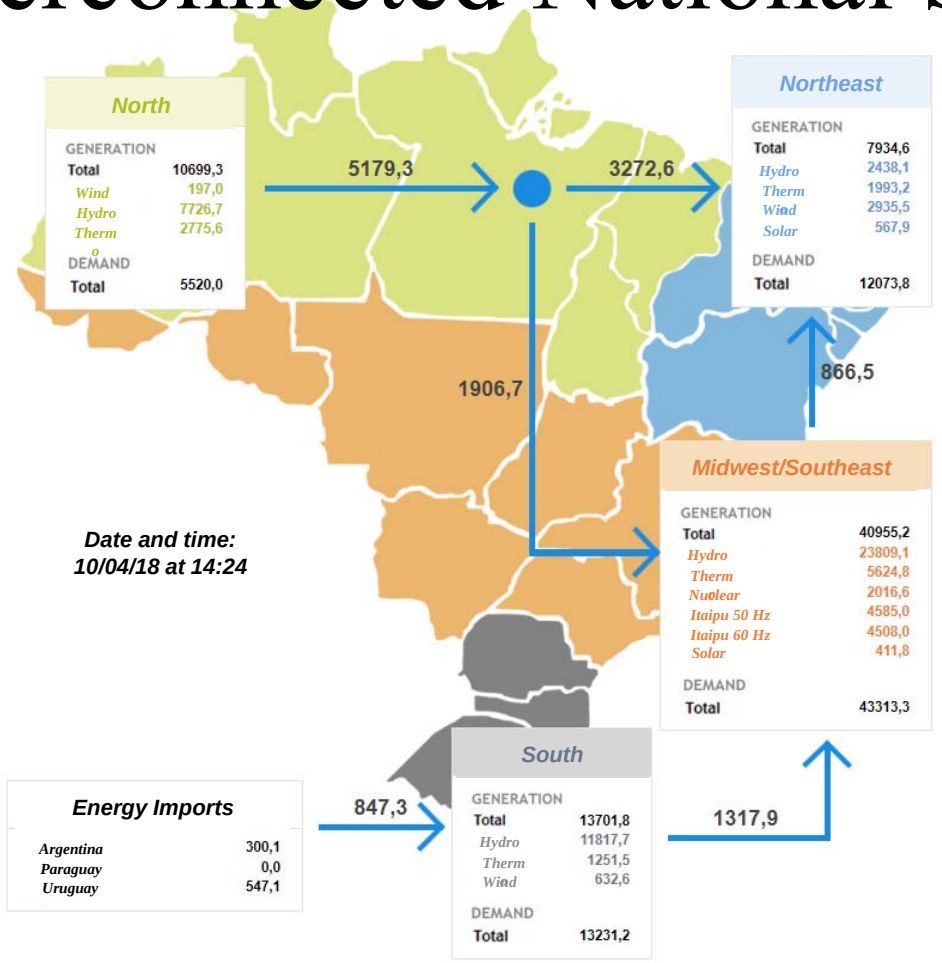
Brazilian Energy Mix



Source: Adapted from EPE (2018).

2014 Brazilian drought

Interconnected National System



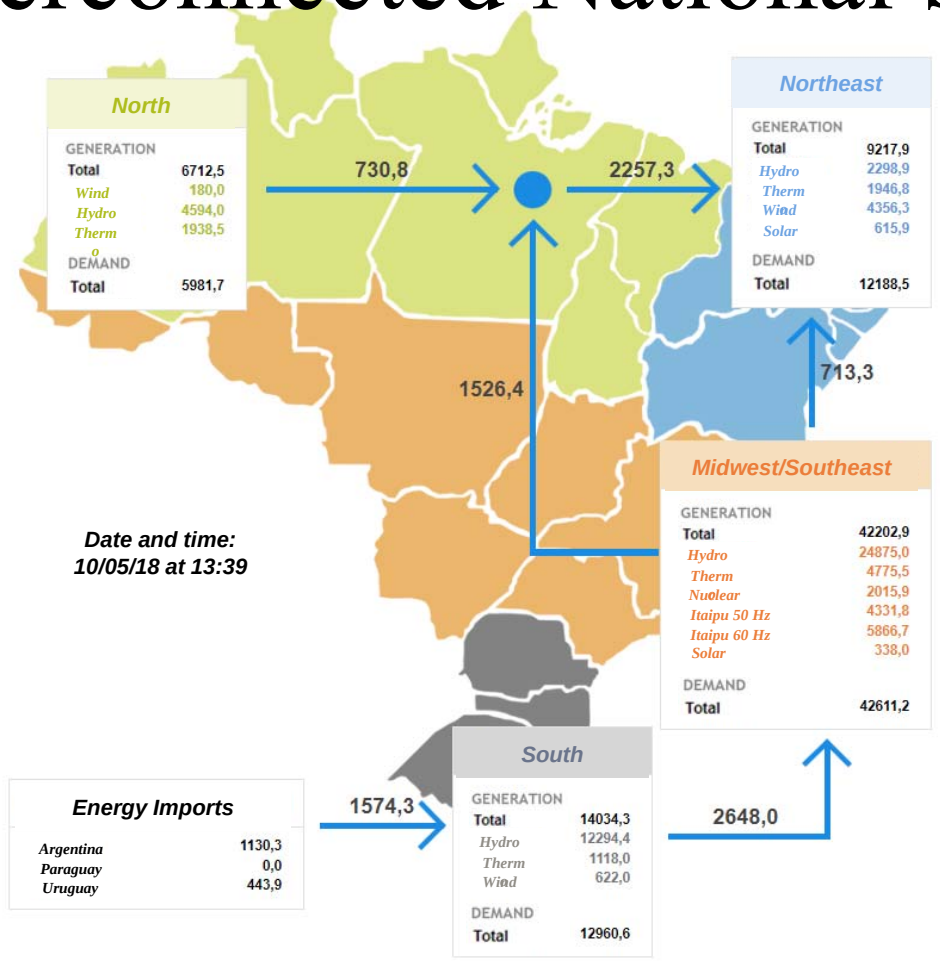
POWER IN MW

Northeast

GENERATION	
Total	7934,6
Hydro	2438,1
Thermo	1993,2
Wind	2935,5
Solar	567,9
DEMAND	
Total	12073,8

Source: Adapted from ONS (2018).

Interconnected National System



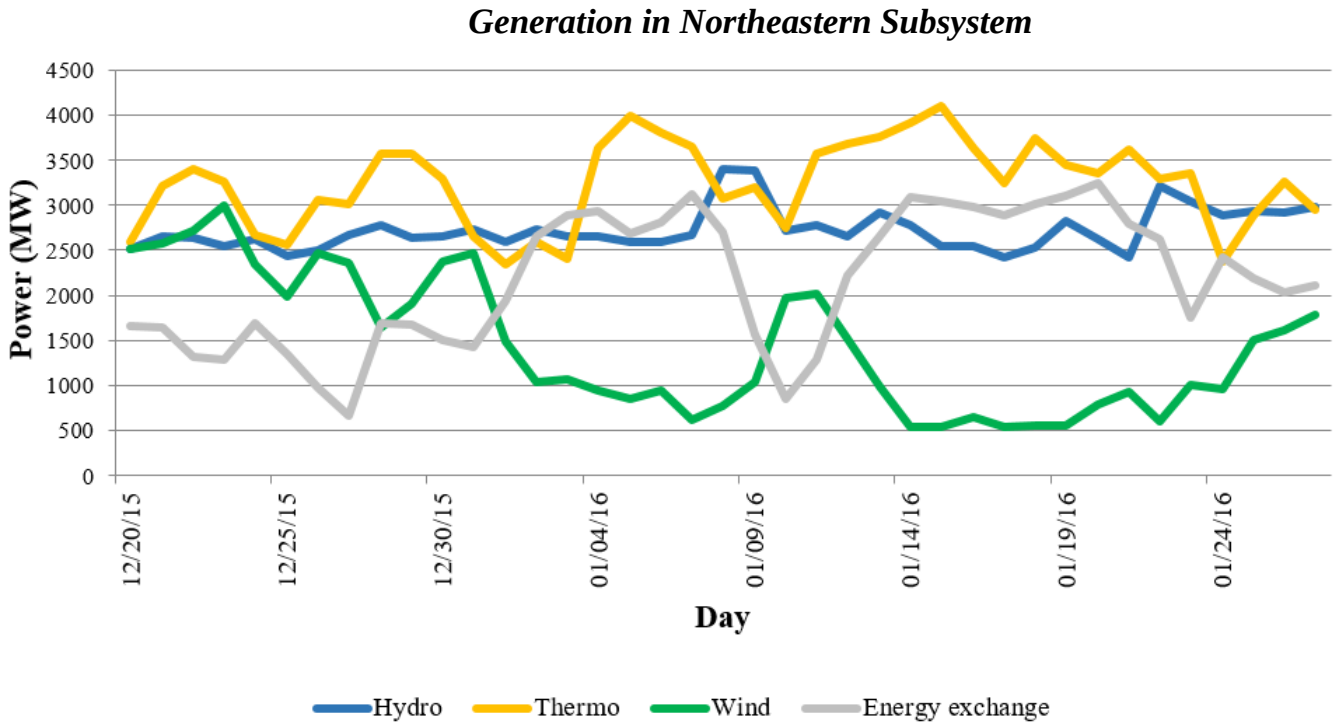
POWER IN MW

Northeast

GENERATION	
Total	9217,9
Hydro	2298,9
Thermo	1946,8
Wind	4356,3
Solar	615,9
DEMAND	
Total	12188,5

Source: Adapted from ONS (2018).

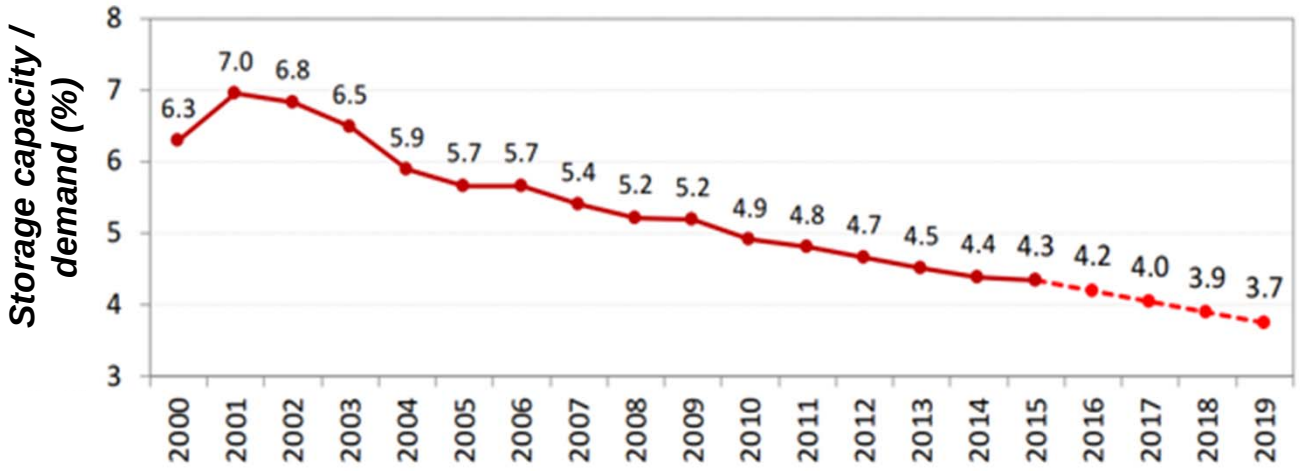
Intermittent generation



Source: Adapted from EPE (2018).

Current storage options

- Hydro with reservoirs
- Thermo fuel



Source: Adapted from EPE (2018).

Energy Storage Systems

- Energy storage systems can help mitigating wind intermittence

Source: DOE (2010).

Narrative

An energy storage problem

- Consider a basic energy storage problem:

» We are going to show that with minor variations in the characteristics of this problem, we can make *each* class of policy work best.

- There are many variations of "storage problems."
  - » Inventory planning at Amazon (or any retailer)
  - » How much cash to keep in a mutual fund.
  - » How much blood to keep in a blood bank.
  - » How many vaccines to keep in inventory

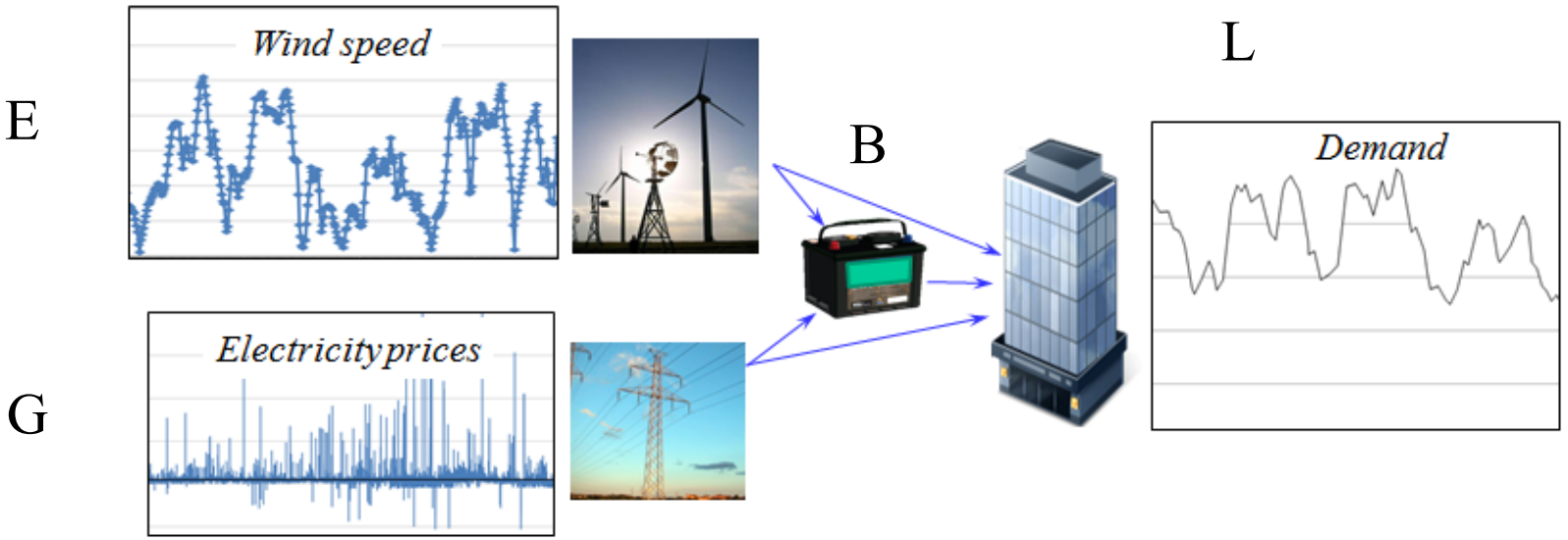
Basic model

An energy storage problem

- A model of our problem
  - » State variables
  - » Decision variables
  - » Exogenous information
  - » Transition function
  - » Objective function

An energy storage problem

- State variables

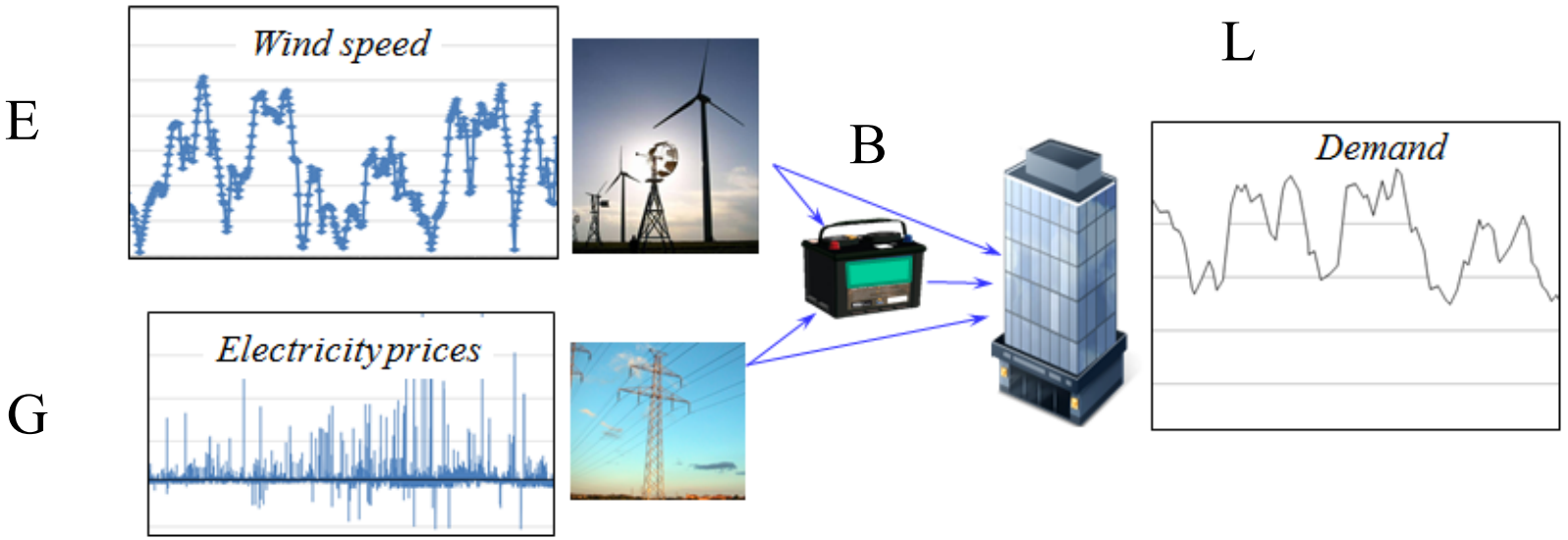


» We will present the full model, accumulating the information we need in the state variable.

» We will highlight information we need as we proceed. This information will make up our state variable.

An energy storage problem

- Decision variables



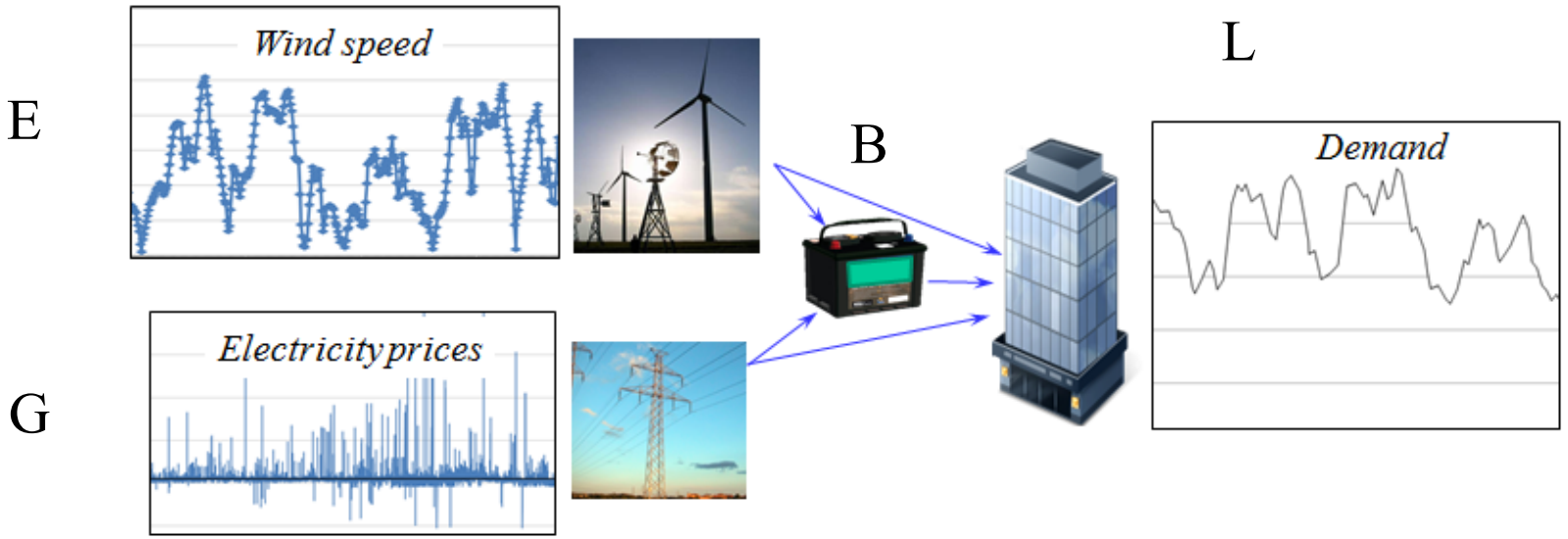
$$x_t = \left(x_t^{EL}, x_t^{EB}, x_t^{GL}, x_t^{GB}, x_t^{BL}, \right)$$

» Constraints;

$$\begin{aligned} x_t^{EL} + x_t^{EB} &\leq E_t, \\ (x_t^{GL} + x_t^{EL} + x_t^{BL}) &= L_t, \\ x_t^{BL} &\leq R_t, \end{aligned}$$

An energy storage problem

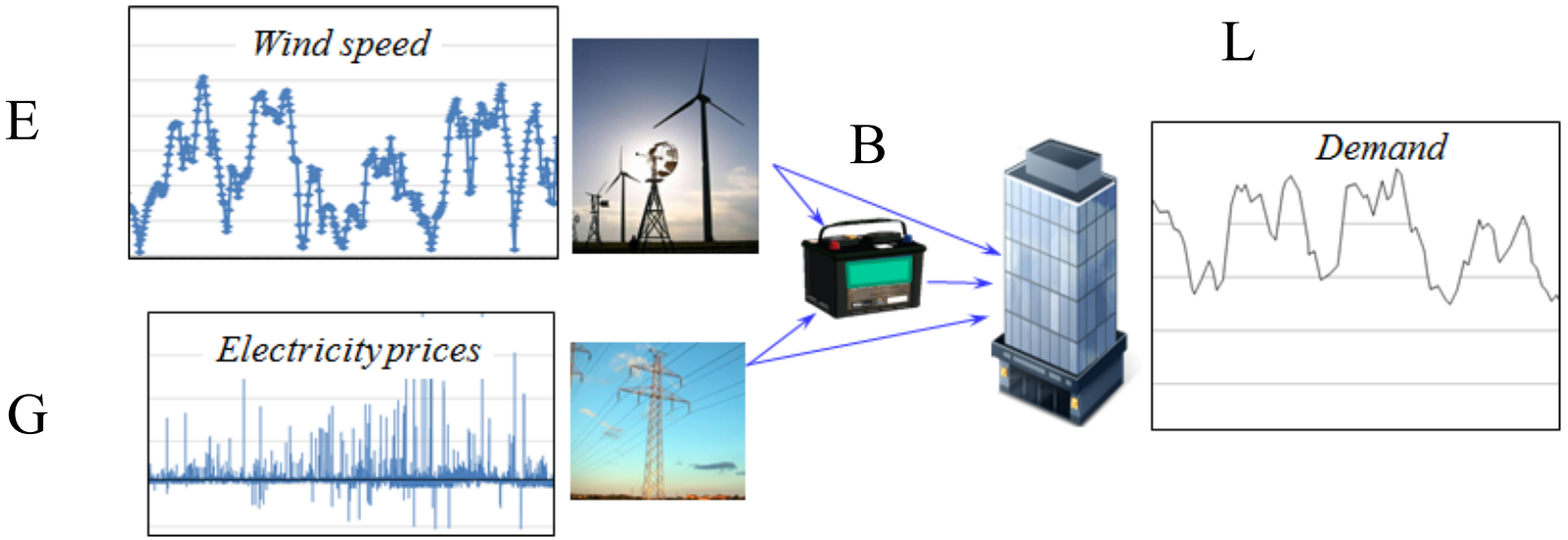
- Exogenous information



$$W_t = \begin{cases} \hat{E}_t = \text{Change in energy from wind between } t-1 \text{ and } t \\ \varepsilon_t^p = \text{Noise in the price process between } t-1 \text{ and } t \\ f_{tt'}^D = \text{Forecast of demand } D_{t'} \text{ provided by vendor at time } t \\ f_t^D = \left(f_{tt'}^D \right)_{t'>t} \quad \text{Provided exogenously} \\ \varepsilon_t^D = \text{Difference between actual demand and forecast} \end{cases}$$

An energy storage problem

- Transition function



$$E_{t+1} = E_t + \hat{E}_{t+1}$$

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon^p_{t+1}$$

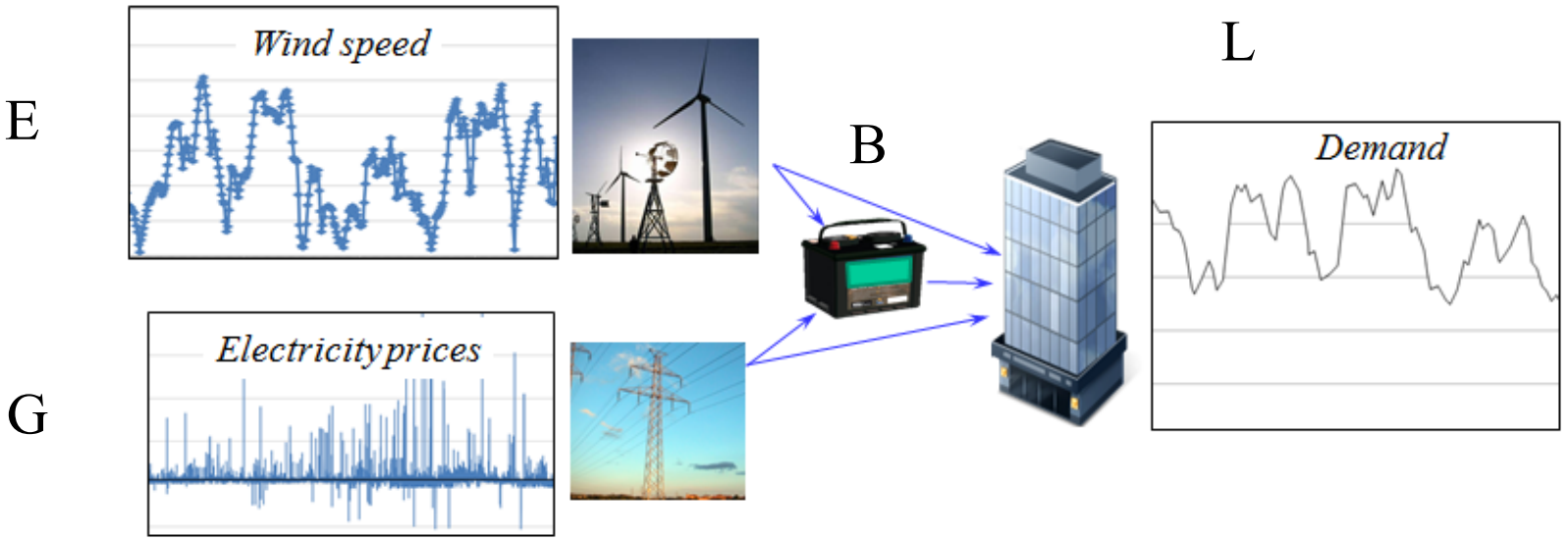
$$D_{t+1} = D_t + \hat{D}_{t+1}$$

$$f^L_t = \text{Provided exogenously}$$

$$R^{battery}_{t+1} = R^{battery}_t + x_t$$

An energy storage problem

- Objective function



$$C(S_t, x_t) = p_t \left(x_t^{GB} + x_t^{GL} \right)$$

$$\min_\pi \mathbb{E} \left\{ \sum_{t=0}^{T} C_t(S_t, X_t^\pi(S_t), W_{t+1}) \,|\, S_0 \right\}$$

An energy storage problem

- State variables
  - » Cost function

    p_t = Price of electricity

  - » Decision function

    Constraints:

$$
\begin{aligned}
x_t^{EL} + x_t^{EB} &\leq E_t \\
(x_t^{GL} + x_t^{EL} + x_t^{BL}) &= L_t \\
x_t^{BL} &\leq R_t
\end{aligned}
$$

$S_t = \left(R_t, E_t, L_t, (p_t, p_{t-1}, p_{t-2}), f_t^L\right)$

f_t^L = Needed if we use a lookahead policy

  - » Transition function

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon_{t+1}^p$$

Designing policies

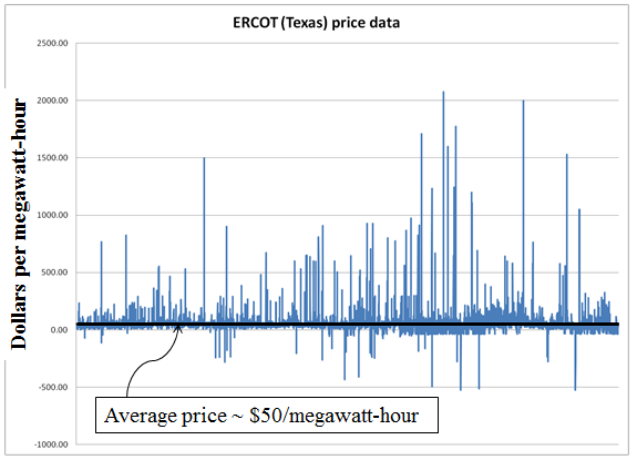
PFA for:

a) Simple battery arbitrage

b) More complex storage problem

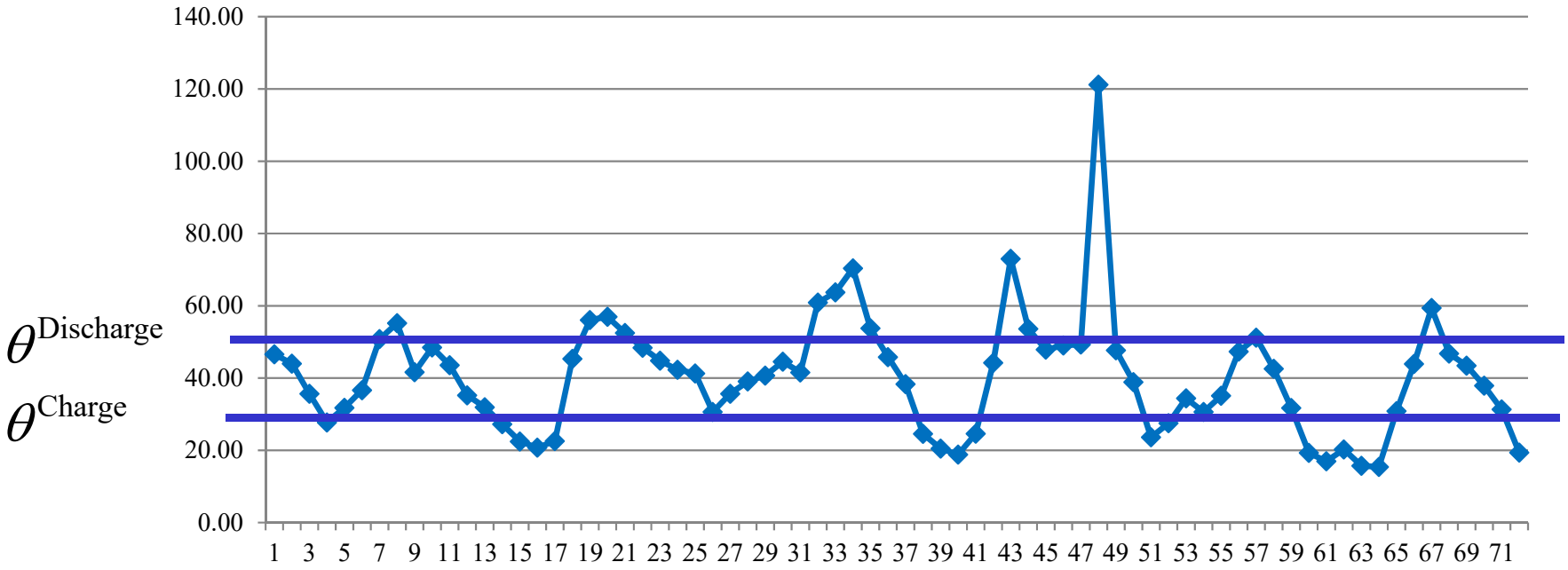
Policy function approximations

- Battery arbitrage – When to charge, when to discharge, given volatile LMPs



Policy function approximations

- Grid operators require that batteries bid charge and discharge prices, an hour in advance.



- We have to search for the best values for the policy parameters θ^{Charge} and $\theta^{\text{Discharge}}$.

8.4.1 Buy low, sell high

A buy low, sell high policy works on the simple principle of charging the battery when the price falls below a lower limit, and then sells when the price goes above an upper limit. The policy can be written as

$$X^{low-high}(S_t|\theta) = \begin{cases} -1 & \text{If } p_t \leq \theta^{buy} \\ 0 & \text{If } \theta^{buy} < p_t < \theta^{sell} \\ +1 & \text{If } p_t \geq \theta^{sell}. \end{cases} \tag{8.11}$$

We now have to tune $\theta = (\theta^{buy}, \theta^{sell})$. We evaluate our policy by following a sample path of prices $p_t(\omega)$ (or we may be observing the changes in prices $\hat{p}(\omega)$). Assuming we are generating sample paths from a mathematical model, we can generate sample paths $\omega^1, \ldots, \omega^N$. We can then simulate the performance of the policy over each sample path and take an average using

$$\overline{F}^{low-high} = \frac{1}{N}\sum_{n=1}^{N} C\big(S_t(\omega^n), X^{low-high}(S_t(\omega^n)|\theta)\big).$$

Tuning θ requires solving the problem

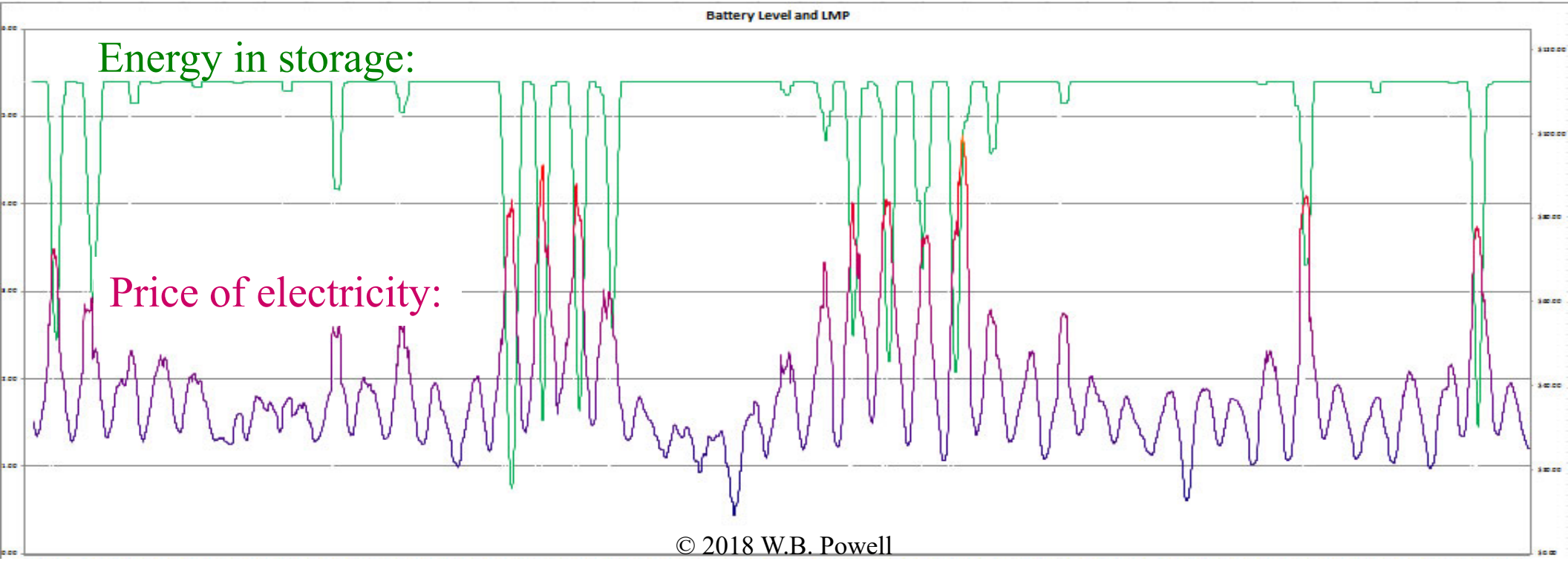
$$\max_{\theta} \overline{F}^{\pi}(\theta). \tag{8.12}$$

Since θ has only two dimensions, one strategy is to do a full grid search by discretizing each dimension, and then searching over all possible values of the two dimensions. A common discretization is to divide a region into 5-percent increments. Including the boundaries, this means we have to represent 21 values of each parameter, creating a grid of size 441 points, which is manageable (although not trivial) for most problems.

Policy function approximations

- Our policy function might be the parametric model (this is nonlinear in the parameters):

$$X^{\pi}(S_t \mid \theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

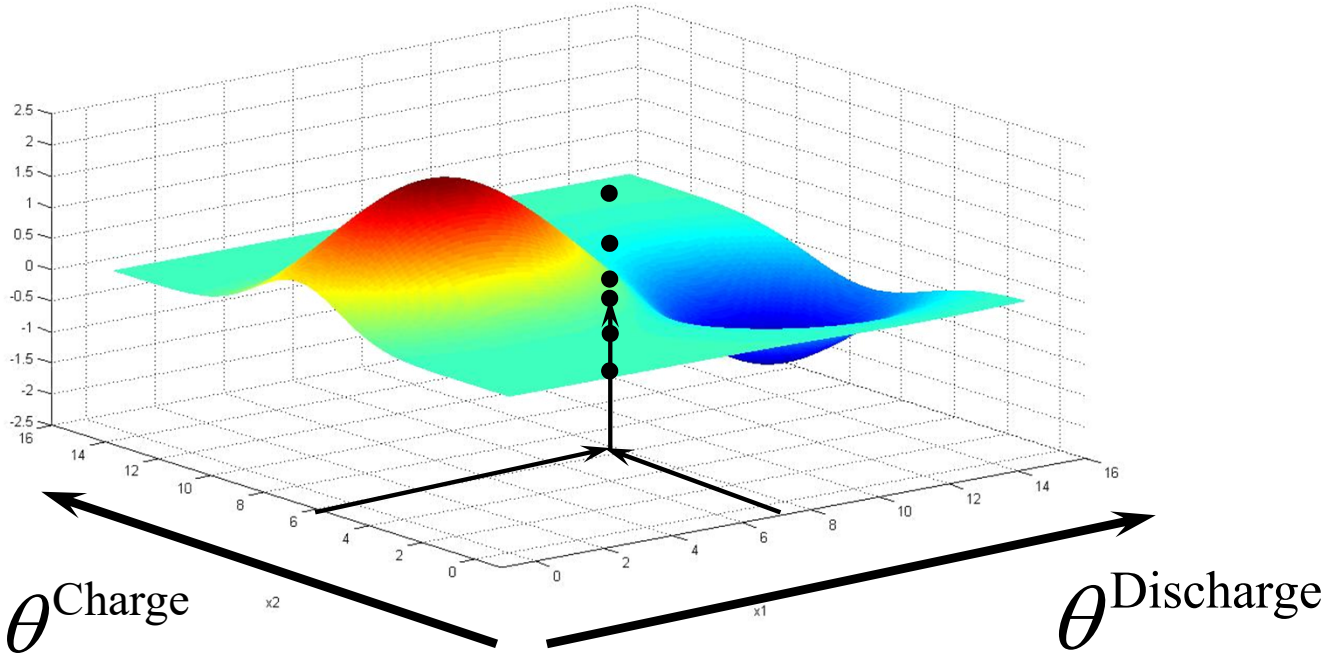


Policy function approximations

- Finding the best policy
  - » We need to maximize

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T}\gamma^t C\left(S_t, X_t^\pi(S_t \mid \theta)\right)$$

  - » We cannot compute the expectation, so we run simulations:



Policy function approximations

- Performing policy search
  - » Searching for the best values of θ is its own sequential decision problem.
  - » There are two settings for doing policy search:
    - Offline using a simulator
      - 1) Using real data – this would be a "data-driven" simulator
      - 2) Using observations from a mathematical model (see lecture for next week)
    - Online in the field
      - Need a policy that learns while earning
      - This is the setting of multiarmed bandit problems

Week 5 – Wednesday

Energy storage II

Policy search for PFAs

Derivative-based stochastic search

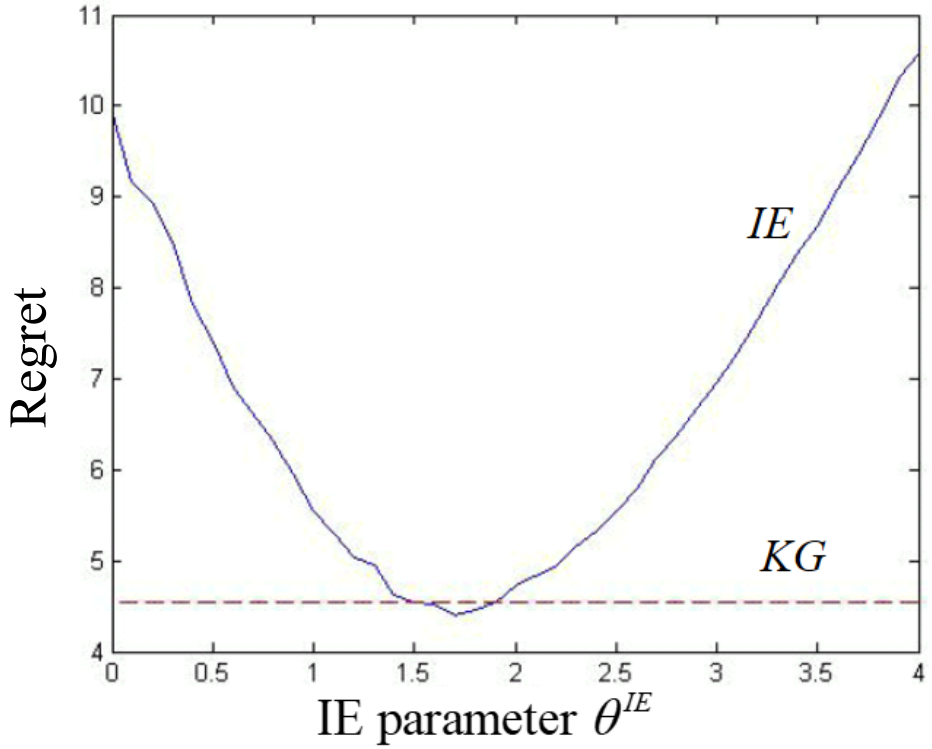
Cost function approximations

- Optimizing the policy
  - » We optimize θ^{IE} to maximize:

$$\max_{\theta^{IE}} F(\theta^{IE}) = \mathbb{E}F\left(x^{\pi,N}, W\right)$$

$$x^n = X^{IE}(S^n \mid \theta^{IE}) = \arg\max_x \left(\bar{\mu}_x^n + \theta^{IE}\bar{\sigma}_x^n\right) \qquad S^n = (\bar{\mu}_x^n, \bar{\sigma}_x^n)$$

  - » Policy search matters:



Policy function approximations

- Derivative-based
  - » Consider basic newsvendor problem

$$F(x,W) = p\min\{x,W\} - cx.$$

  - » If we can compute the gradient

$$\frac{\partial F(x^{n-1},W^n)}{\partial x} = \begin{cases} p-c & \text{If } x^{n-1} < W^n, \\ -c & \text{If } x^{n-1} > W^n. \end{cases} \tag{5.13}$$

  - » Then we can use a stochastic gradient algorithm

$$x^n = x^{n-1} + \alpha_{n-1}\nabla_x F(x^{n-1},W^n). \tag{5.12}$$

Policy function approximations

- Derivative-based
  - » In our newsvendor problem, we are optimizing over x. Now, we are optimizing over θ. We might write our problem as

$$F^{\pi}(\theta) = \mathbb{E}_W F^{\pi}(\theta, W) = \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t|\theta))$$

where $S_{t+1} = S^M(S_t, X^{\pi}(S_t|\theta), W_{t+1})$

  - » In practice, we often cannot compute gradients.
  - » Instead, try using numerical derivatives.

Derivative-based stochastic search

- Finite differences

makes sense to estimate the derivative using finite differences. In this setting, we can approximate gradients using finite differences. Assume that x is a K-dimensional vector, and let e_k be a K-dimensional column vector of zeroes with a 1 in the kth position. Now assume that we can run two simulations for each dimension, $F(x^n + \delta x^n e_k, W_k^{n+1,+})$ and $F(x^n - \delta x^n e_k, W_k^{n+1,-})$ where $\delta x^n e_k$ is the change in x^n, multiplied by e_k so that we are only changing the kth dimension. We use $W_k^{n+1,+}$ and $W_k^{n+1,}$ to represent the sequences of random variables that are generated when we run each simulation, which would be run in the $n+1st$ iteration. Think of $F(x^n + \delta x^n e_k, W_k^{n+1,+})$ and $F(x^n - \delta x^n e_k, W_k^{n+1,-})$ as calls to a black-box simulator where we start with a set of parameters x^n, and then perturb it to $x^n + \delta x^n e_k$ and $x^n - \delta x^n e_k$ and run two separate, independent simulations. We then have to do this for each dimension k, allowing us to compute

$$g_k^n(x^n, W^{n+1,+}, W^{n+1,-}) = \frac{F(x^n + \delta x^n e_k, W_k^{n+1,+}) - F(x^n - \delta x^n e_k, W_k^{n+1,-})}{2\delta x^n}, \tag{5.22}$$

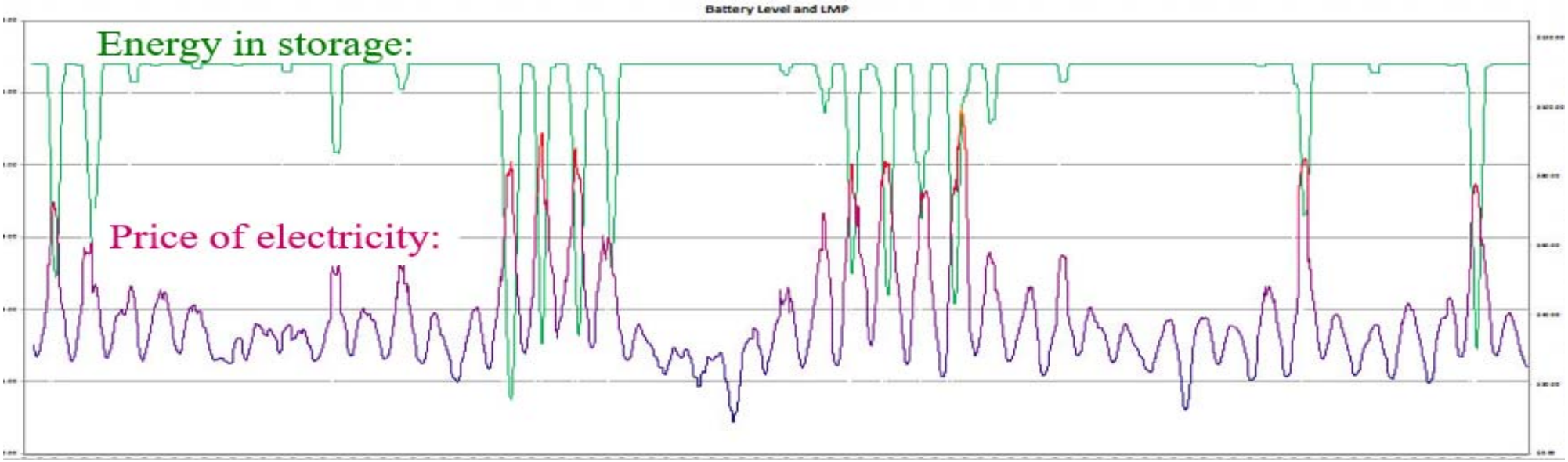
where we divide the difference by the width of the change which is $2\delta x^n$ to get the slope.

The calculation of the derivative (for one dimension) is illustrated in figure 5.1. We see from figure 5.1 that shrinking δx can introduce a lot of noise in the estimate of the gradient. At the same time, as we increase δx, we introduce bias, which we see in the difference between the dashed line showing $\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$, and the dotted line that depicts $\partial \mathbb{E}F(x^n, W^{n+1})/\partial x^n$. If we want an algorithm that converges asymptotically in the limit, we need δx^n decreasing, but in practice it is often set to a constant δx, which is then handled as a tunable parameter.

Derivative-based stochastic search

- Finite differences
  - » We can just simulate the policy (even in a spreadsheet):

$$X^{\pi}(S_t \mid \theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

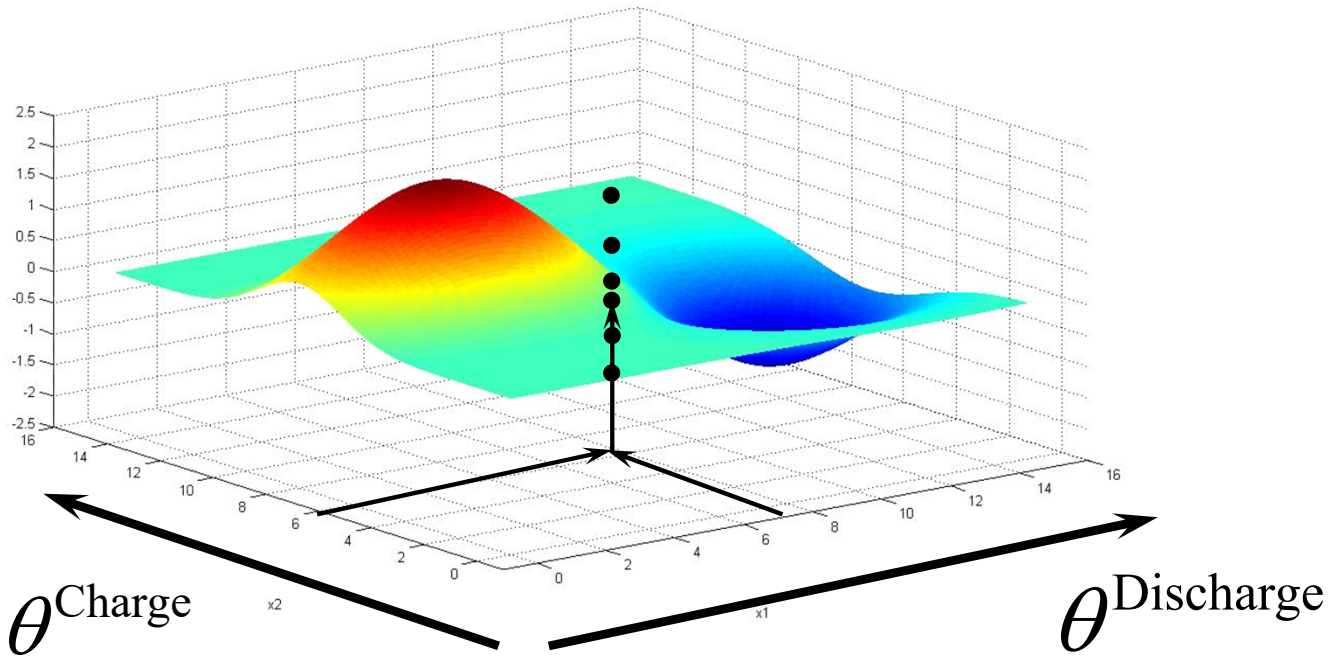


Derivative-based stochastic search

- Finding the best policy
  - » We need to maximize

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^\pi(S_t \mid \theta)\right)$$

  - » We cannot compute the expectation, so we run simulations:



Simulating the numerical derivative

Let ω be a sample path of $W_1(\omega), W_2(\omega), \ldots$ with a sequence of states $S_{t+1}(\omega) = S^M(S_t(\omega), X_t^\pi(S_t(\omega)\,|\,\theta), W_{t+1}(\omega))$.

Let

$$F^\pi(\theta,\omega) = \sum_{t=0}^{T} C\left(S_t(\omega), X_t^\pi(S_t(\omega)\,|\,\theta)\right)$$

Now compute the stochastic numerical derivative using

$$g(\omega) = \frac{F^\pi(\theta+\delta,\omega) - F^\pi(\theta,\omega)}{\delta}$$

We are writing this assuming that $F^\pi(\theta+\delta,\omega)$ and $F^\pi(\theta,\omega)$ are both computed with the same sample path ω. This is best, but not required, and not always possible.

Derivative-based stochastic search

- Finite differences

Replace x^n with θ^n

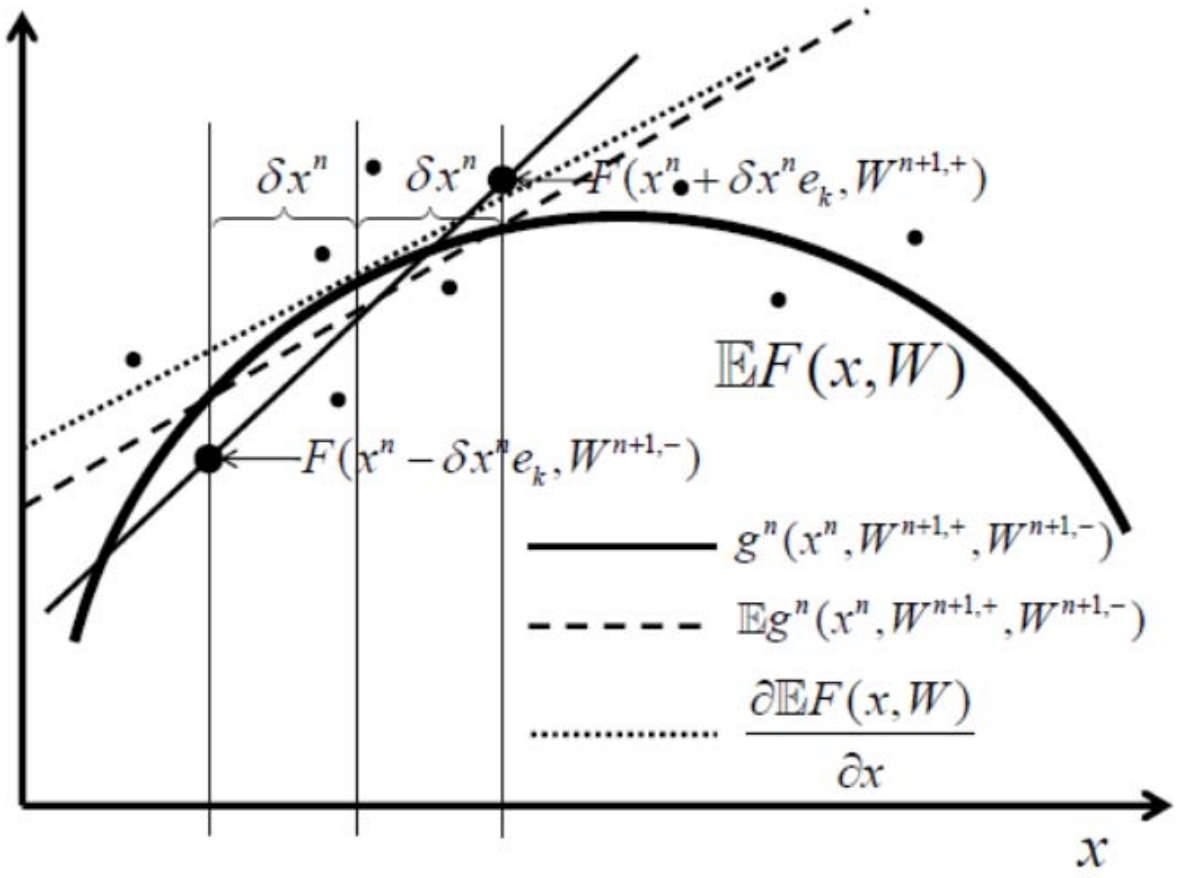


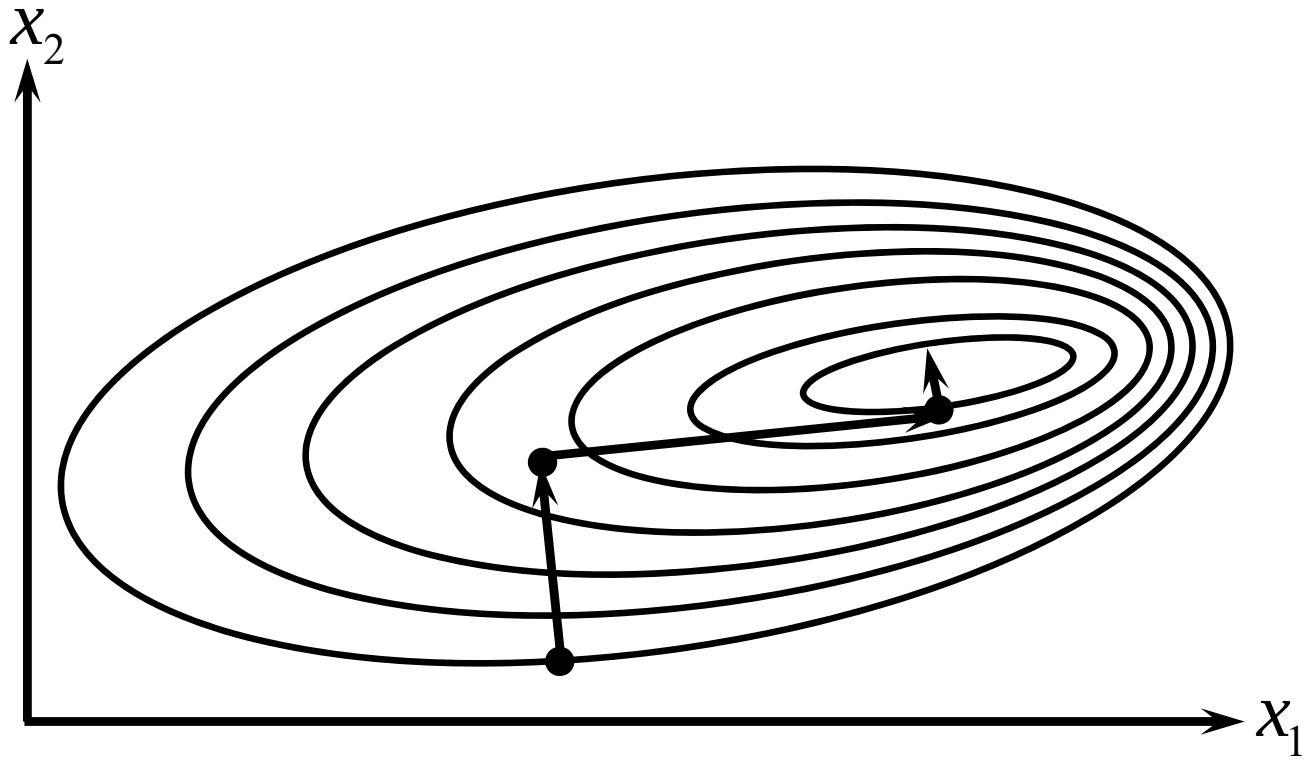
Figure 5.1 Different estimates of the gradient of $F(x,W)$ with a) the stochastic gradient $g^n(x^n, W^{n+1,+}, W^{n+1,-})$ (solid line), the expected finite difference $\mathbb{E}g^n(x^n, W^{n+1,+}, W^{n+1,-})$ (dashed line), and the exact slope at x^n, $\partial \mathbb{E}F(x^n, W^{n+1})/\partial x^n$.

Finite-difference estimate of derivative

													Perturb lower price			
loss	0.70			Battery size	8.00			Derivatives:		Battery size	8.00					
Smoothing	1.00		Buy	30.00				0.62		32.00						Derivative
delta	2		Sell	50.00		Total profit/hr	$16.69	0.39		50.00		Total profit/hr	$17.92			0.615115
					Amt in						Amt in					
	Time period	Hour of day	Price	Buy-sell	storage	Bought/sold	Revenue			Buy-sell	storage	Bought/sold	Revenue			
					0.00						0.00					
1/1/05	1	1	292.54	-1.00	0.00	0.00	0.00		292.54	-1.00	0.00	0.00	0.00			
1/1/05	2	2	86.97	-1.00	0.00	0.00	0.00		86.97	-1.00	0.00	0.00	0.00			
1/1/05	3	3	65.50	-1.00	0.00	0.00	0.00		65.50	-1.00	0.00	0.00	0.00			
1/1/05	4	4	123.52	-1.00	0.00	0.00	0.00		123.52	-1.00	0.00	0.00	0.00			
1/1/05	5	5	76.08	-1.00	0.00	0.00	0.00		76.08	-1.00	0.00	0.00	0.00			
1/1/05	6	6	43.34	0.00	0.00	0.00	0.00		43.34	0.00	0.00	0.00	0.00			
1/1/05	7	7	6.81	1.00	1.00	1.00	-6.81		6.81	1.00	1.00	1.00	-6.81			
1/1/05	8	8	5.42	1.00	2.00	1.00	-5.42		5.42	1.00	2.00	1.00	-5.42			
1/1/05	9	9	2.77	1.00	3.00	1.00	-2.77		2.77	1.00	3.00	1.00	-2.77			
1/1/05	10	10	22.80	1.00	4.00	1.00	-22.80		22.80	1.00	4.00	1.00	-22.80			
1/1/05	11	11	141.56	-1.00	3.00	-0.70	99.09		141.56	-1.00	3.00	-0.70	99.09			
1/1/05	12	12	101.64	-1.00	2.00	-0.70	71.15		101.64	-1.00	2.00	-0.70	71.15			
1/1/05	13	13	2.88	1.00	3.00	1.00	-2.88		2.88	1.00	3.00	1.00	-2.88			
1/1/05	14	14	7.10	1.00	4.00	1.00	-7.10		7.10	1.00	4.00	1.00	-7.10			
1/1/05	15	15	91.74	-1.00	3.00	-0.70	64.22		91.74	-1.00	3.00	-0.70	64.22			
1/1/05	16	16	46.01	0.00	3.00	0.00	0.00		46.01	0.00	3.00	0.00	0.00			
1/1/05	17	17	52.54	-1.00	2.00	-0.70	36.78		52.54	-1.00	2.00	-0.70	36.78			

Stochastic gradient algorithm

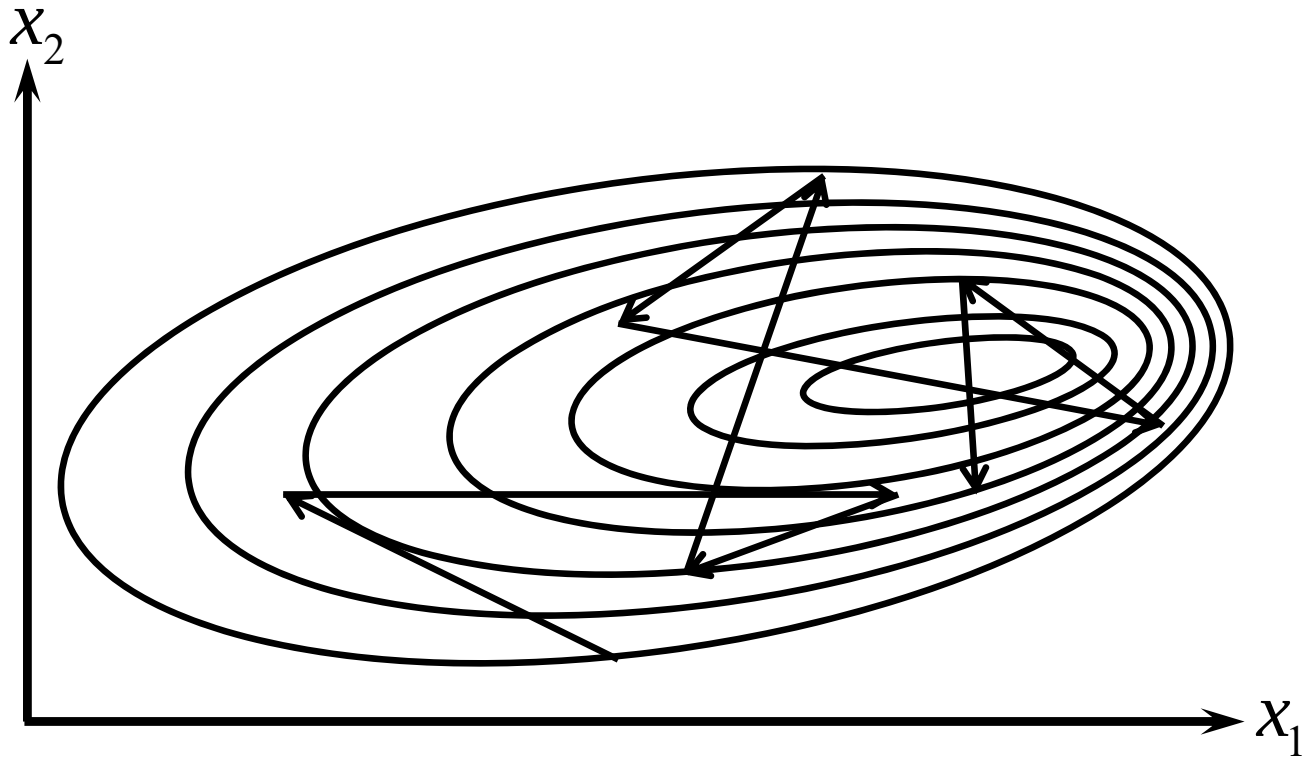
- Steepest ascent for a deterministic problem



$$\theta^{n+1} = \theta^n + \alpha^n \nabla f(\theta^n)$$

Stochastic gradient algorithm

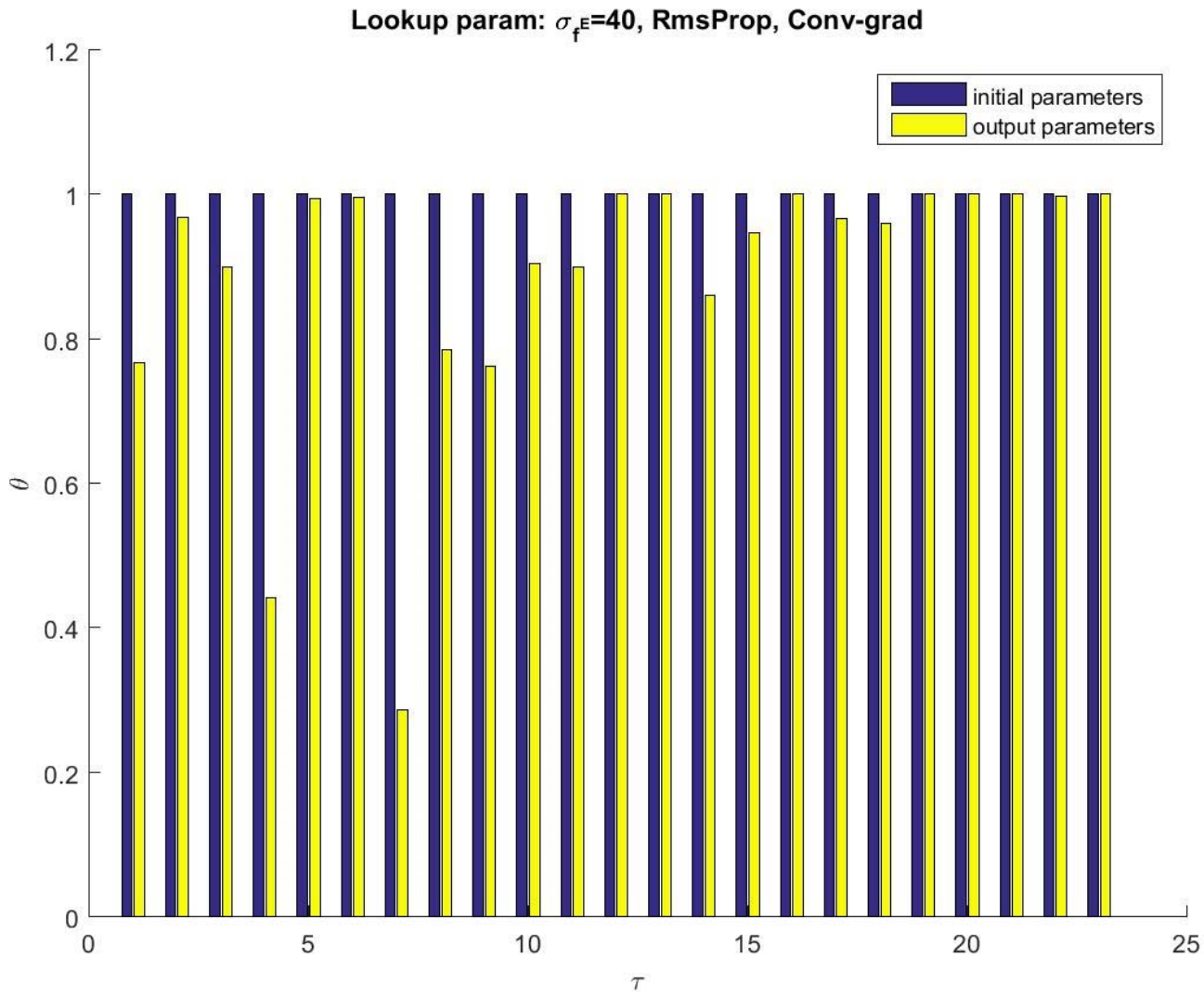
- How a stochastic gradient algorithm may behave

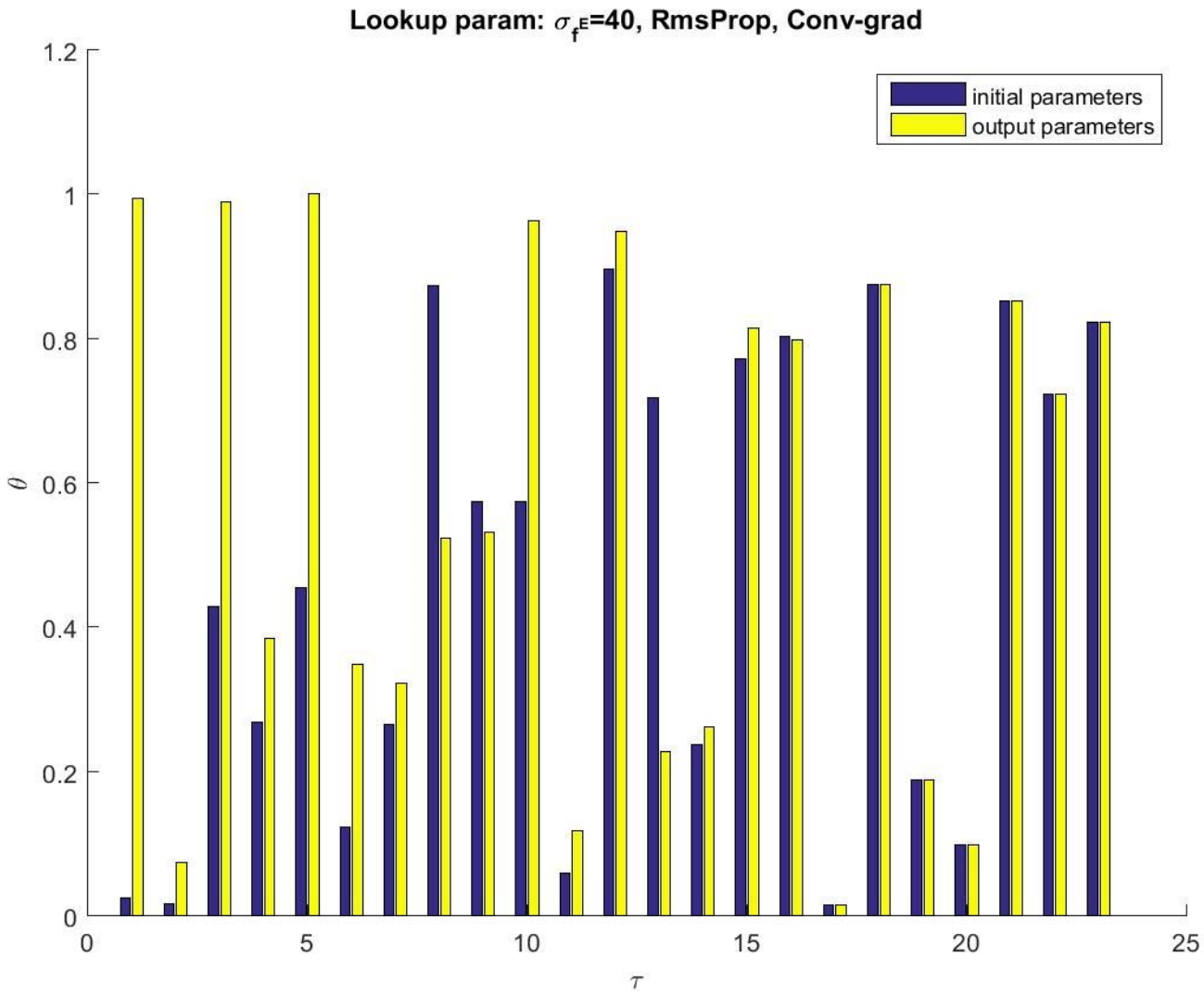


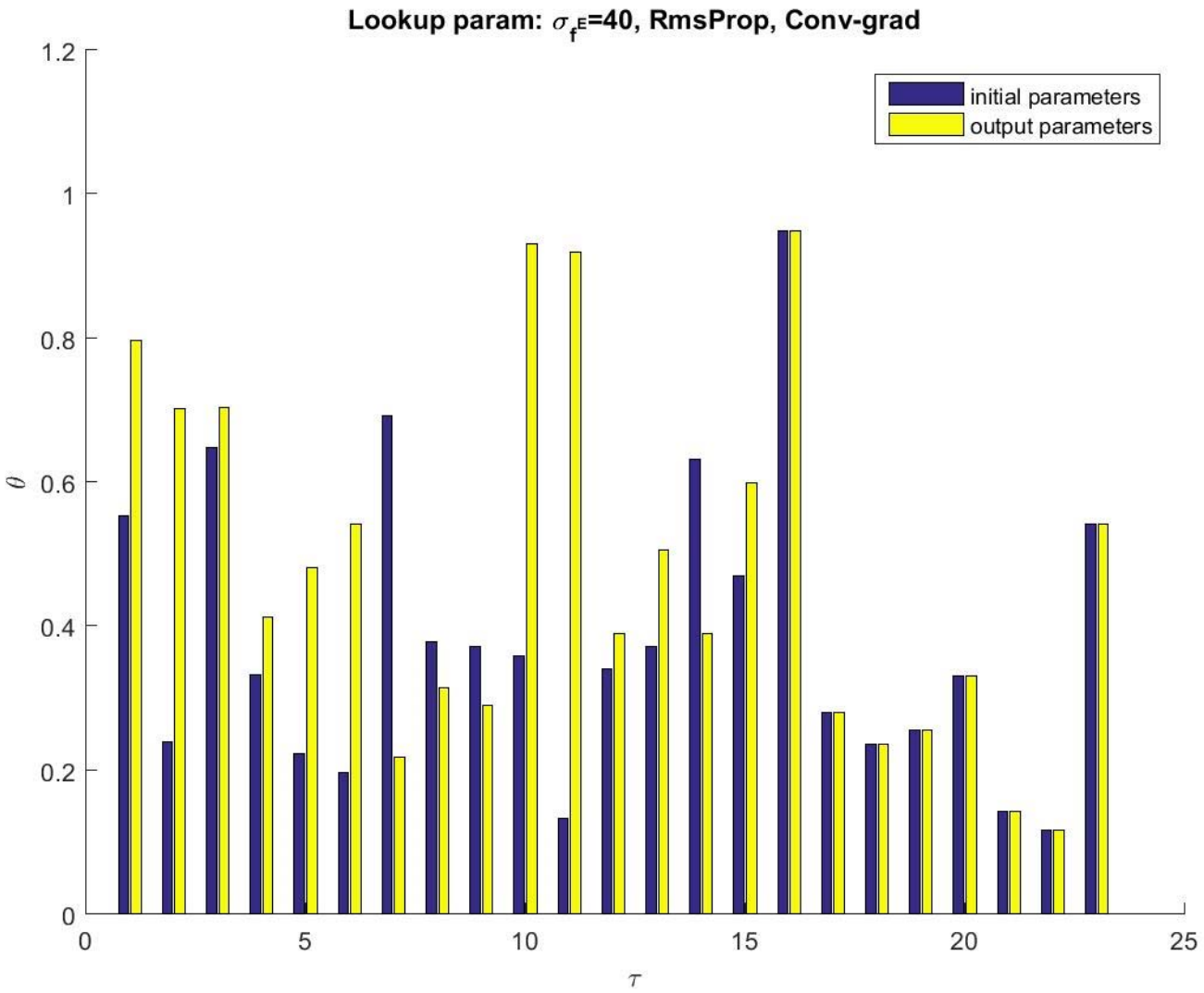
$$\theta^{n+1} = \theta^n + \alpha^n \nabla f(\theta^n, W^{n+1})$$

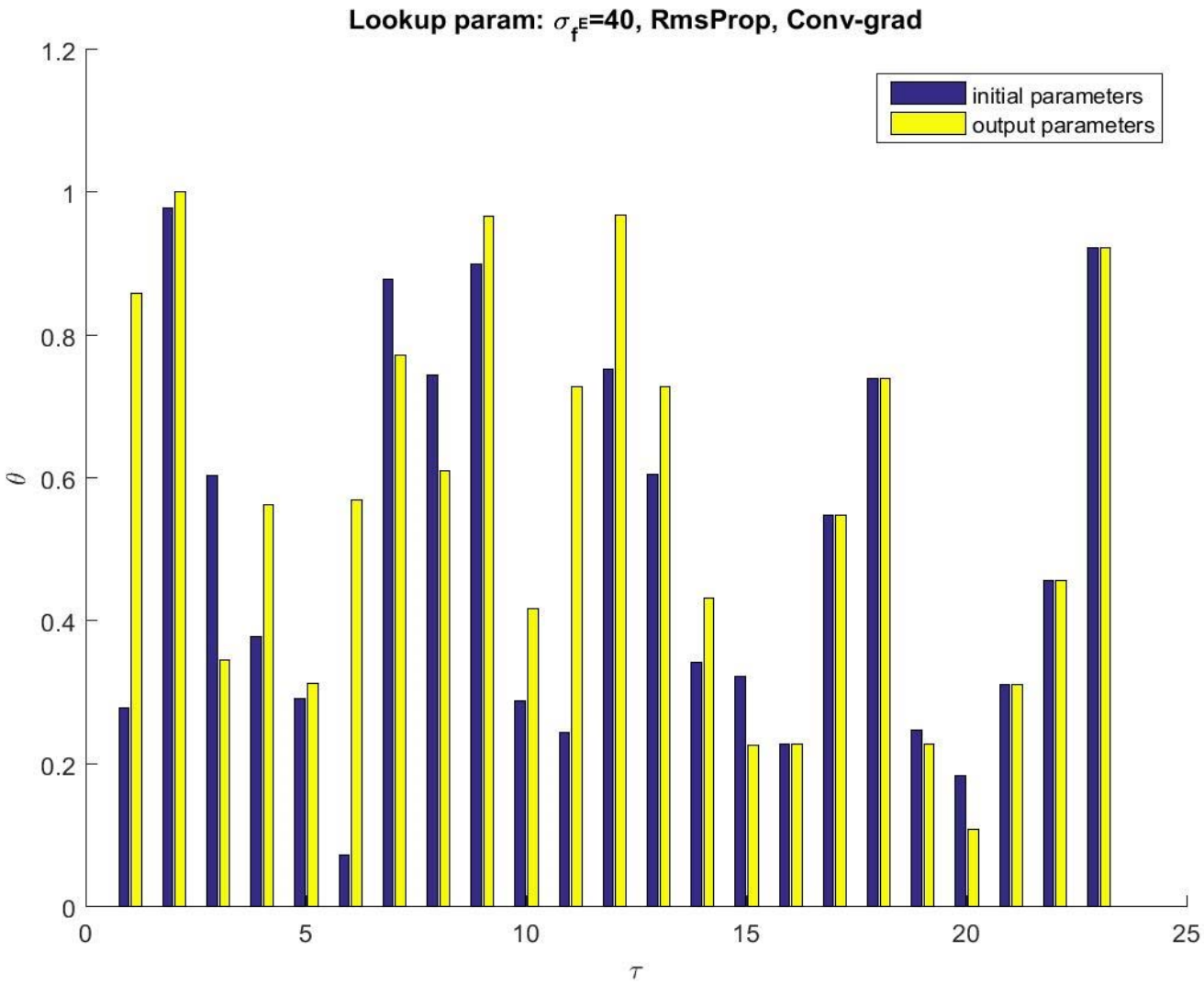
603

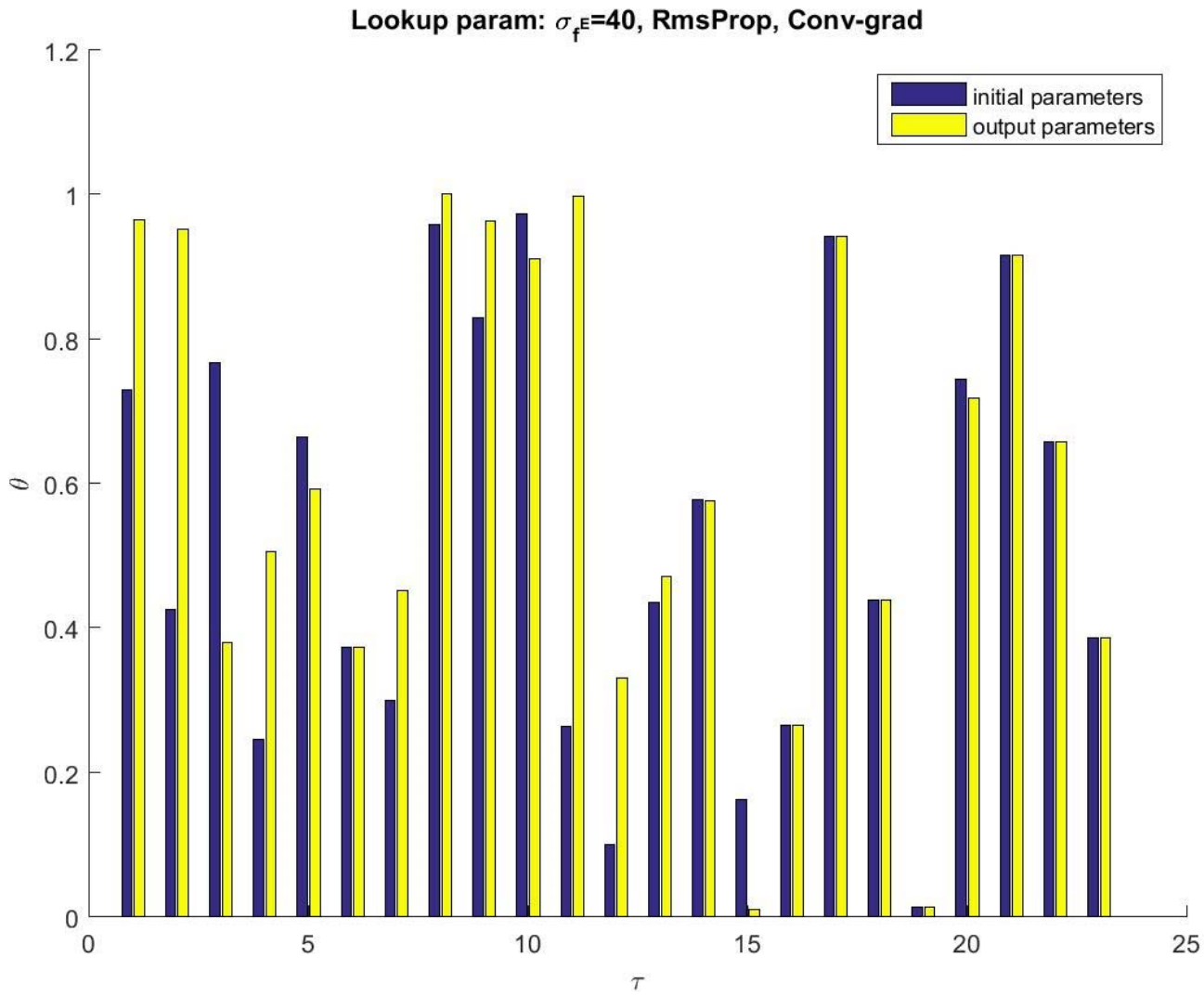
- Notes:
  - The profit function $F(\theta)$ is not concave in θ.
  - Standard practice is to run stochastic gradients from multiple starting points and pick the best final solution.
  - Slides that follow show different initial values of θ, and then the final θ after running a stochastic gradient algorithm.
  - We will return to these results later.

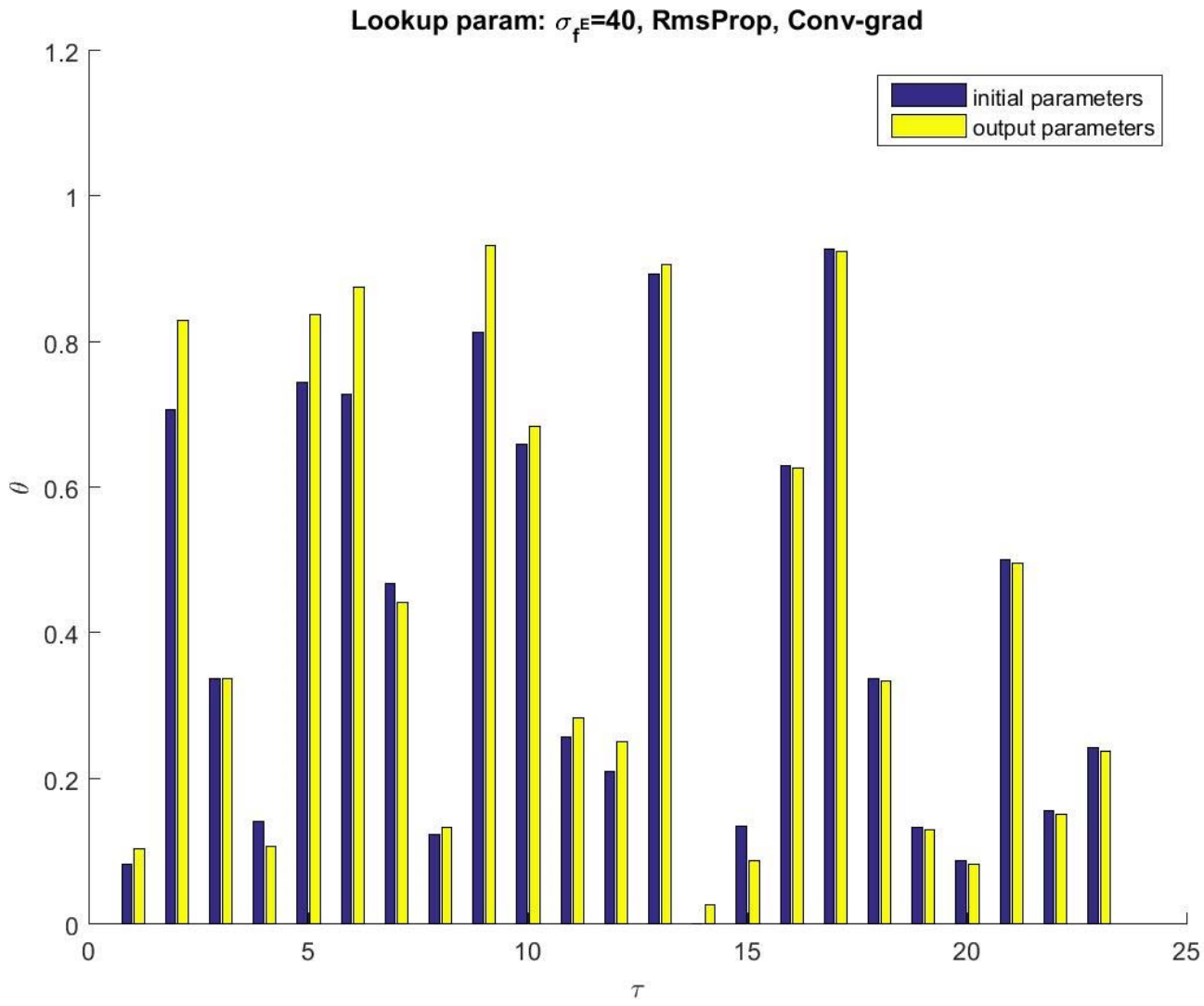




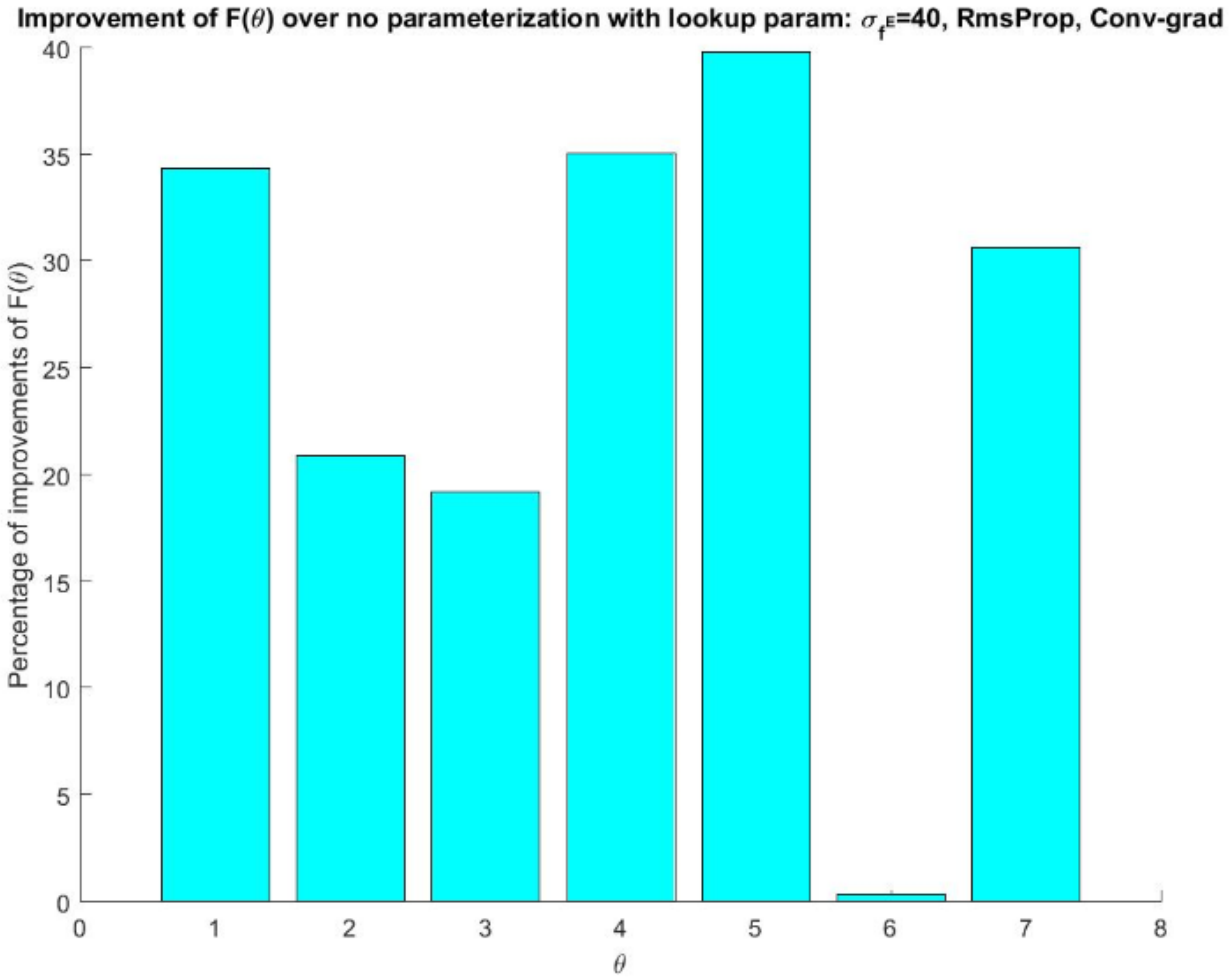








- Improvement between final solution and initial starting point



Policy search for PFAs

Derivative-free stochastic search

Derivative-free policies

- Derivative-free
  - » Here, we are going to assume that we can only choose among a set of finite alternatives

$$(\theta_1, \theta_2, \ldots, \theta_M)$$

  - » Since we no longer have a gradient, we need some other rule, or policy, for choosing

$$\theta^n = \Theta^\pi(S^n \mid \lambda) \quad \lambda \text{ is our "learning parameter"}$$

where
- S^n is our "state of knowledge"
- θ is a set of tunable parameters

  - » Throughout, this works just like our previous algorithms where we were searching for the best x.

- Note:
  - » Everything in this section is the same as what we saw earlier for the diabetes problem.
  - » The difference is that instead of finding the best drug x, we are looking for the best tunable parameter θ to optimize our policy $X^{\pi}(S^n|\theta)$ for finding the best drug.
  - » So, we still need a policy for choosing the best drug, but in this section, we are looking for the best policy to find the drug.
  - » This means we are going to take all of our methods for finding x, and replace them with methods for finding θ.
  - » We are going to be finding a policy $\Theta^{\pi}(S^n|\lambda)$ for finding θ^n. This policy will have its own learning parameter that we are going to call λ (for learning… got it?).
  - » This means we need a policy $\Theta^{\pi}(S^n|\lambda)$ to learn the best parameter θ that determines the performance of the policy $X^{\pi}(S^n|\theta)$.

Derivative-free policies

- The belief model:
  - » Start by assuming that we generate a discrete set of possible values of θ:

$$\theta \in \{\theta_1, \theta_2, ..., \theta_M\}$$

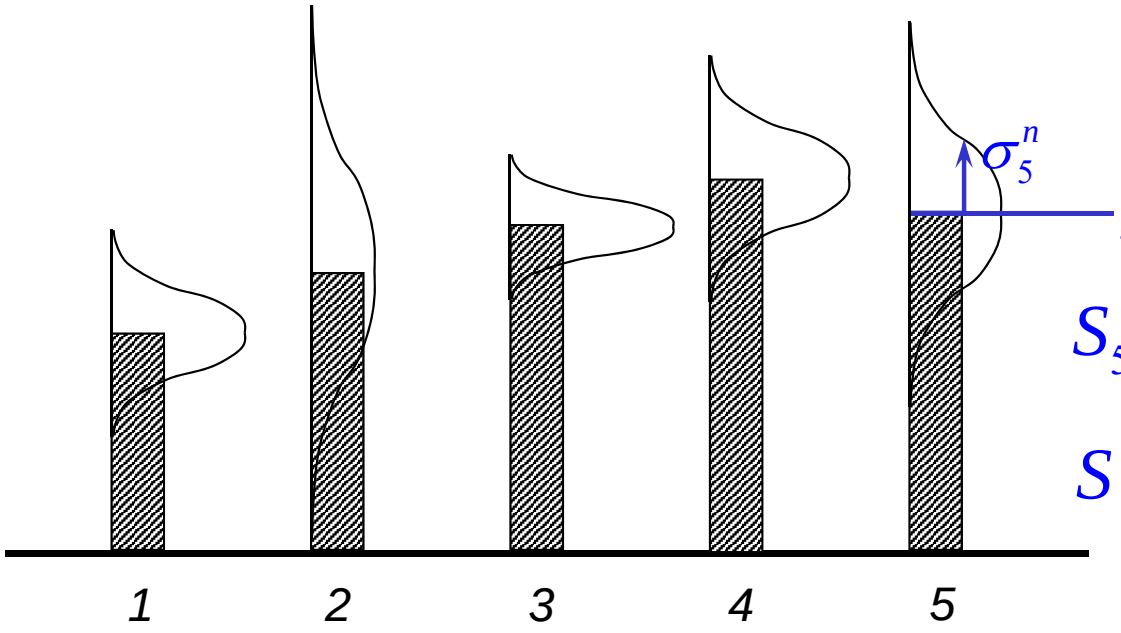
  - » Assume we have some belief about our function (say, lookup table). Using a Bayesian model, we assume we have a distribution of belief about $F(\theta) = \mathbb{E}F(\theta, W)$ given by

$$\mathbb{E}F(\theta, W) = \mu_\theta \sim N\left(\mu_\theta^0, \beta_\theta^0\right)$$

where β_θ^0 is the *precision* in our estimate μ_θ^0 of the value of $F(\theta) = \mathbb{E}F(\theta, W)$.

Derivative-free policies

- Belief state for ranking and selection
  - » S is our "state of knowledge"



$$S_5 = N\left(\mu_5^n, \sigma_5^{2,n}\right)$$

$$S^n = \left(S_1^n, \ldots, S_5^n\right)$$

Derivative-free policies

$$\mu_x \sim N\left(\mu_x^n, \beta_x^n\right)$$

- Updating beliefs
  - » After n experiments, our belief is $\mu_\theta \sim N(\bar{\mu}_\theta^n, \beta_\theta^n)$
  - » Assume that based on this belief, we choose $\theta = \theta^n$ to run for our next experiment (experiment $n+1$), and we observe

$$W_{\theta^n}^{n+1} = \mu_{\theta^n} + \varepsilon^{n+1}$$

  - » We update our beliefs using

$$\bar{\mu}_\theta^{n+1} = \frac{\beta_\theta^n \bar{\mu}_\theta^n + \beta^W W_\theta^{n+1}}{\beta_\theta^n + \beta^W}$$

$$\beta_\theta^{n+1} = \beta_\theta^n + \beta^W$$

  - » This is our *transition function* $S^{n+1} = S^M(S^n, \theta^n, W^{n+1})$ for our state of knowledge using a lookup table belief model.
  - » We could use other (e.g. parametric) belief models.

Derivative-free policies

- Notes:
  - » With derivative-based optimization, we have tended to use the language of algorithms:
    - Choosing a decent vector
    - Choosing a stepsize rule

    We then study the behavior of this "algorithm."
  - » With derivative-free optimization, we have to use a belief model that captures what we know about a problem, and a policy that guides us in how to make the next decision.
  - » In practice, we model both using the language of finding the best "policy."

Derivative-free policies

- Derivative-free policies
  - » Some examples of policies are:
    - Interval estimation:

$$\Theta^{IE}(S^n, \lambda^{IE}) = \arg\max_\theta \left(\mu_\theta^n + \lambda^{IE} \sigma^n \right) \qquad \sigma_\theta^n = \text{Std. dev. of } \mu_\theta^n$$

    - Upper confidence bounding

$$\Theta^{UCB}(S^n, \lambda^{UCB}) = \arg\max_\theta \left(\bar{\mu}_\theta^n + \lambda^{UCB} \sqrt{\frac{\log n}{N_\theta^n}} \right) \qquad N_x^n = \text{No. of times } \theta \text{ is tested.}$$

    - Thompson sampling:

$$\Theta^{TS}(S^n) = \arg\max_\theta \hat{\mu}_\theta^n \qquad \hat{\mu}_\theta^n \sim N(\mu_\theta^n, \beta_\theta^n)$$

Derivative-free policies

- How do we find the best policy?
  - » We use our objective function

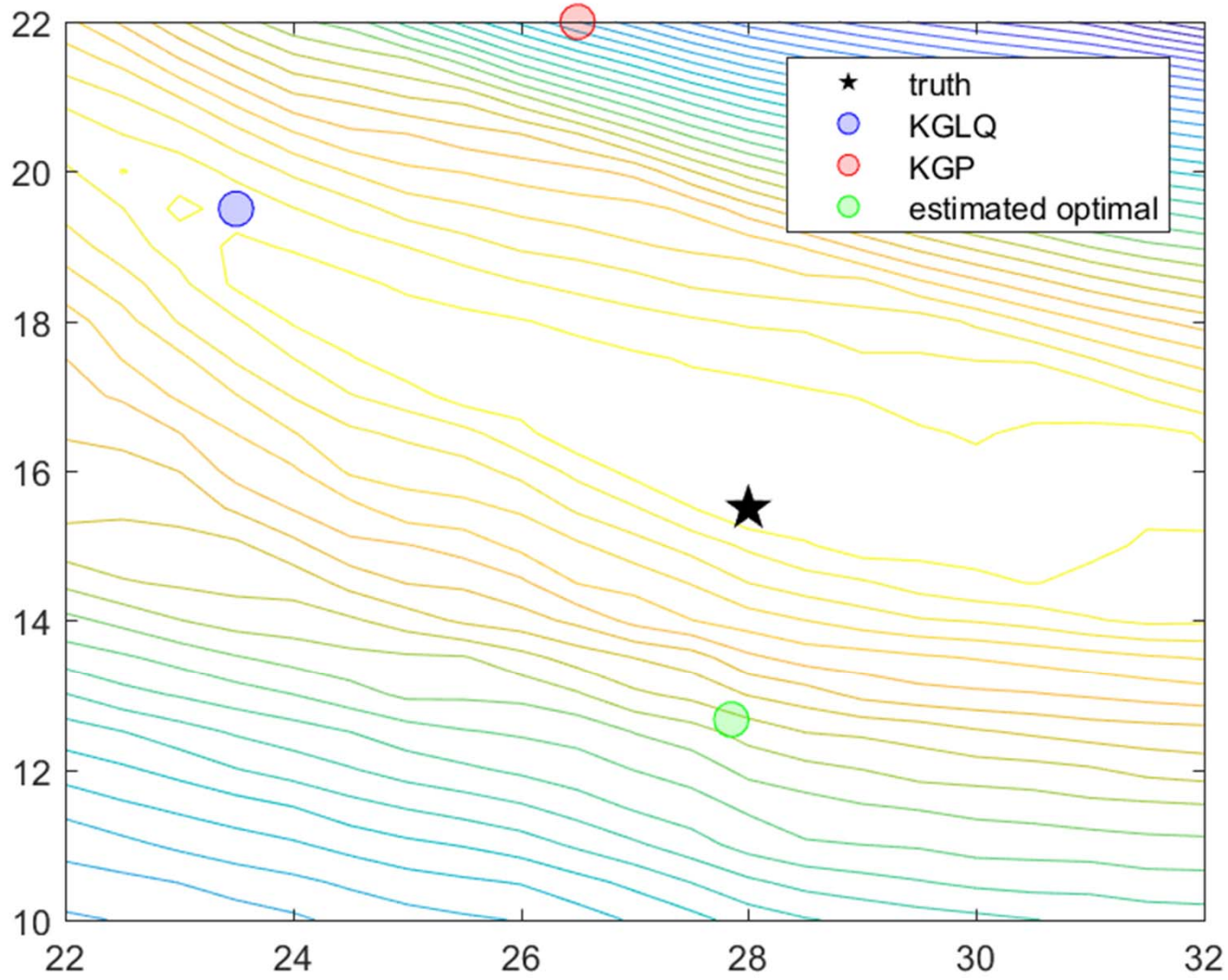
$$\max_{\pi} \mathbb{E}\left\{F(\theta^{\pi,N},W) \mid S_0\right\}$$

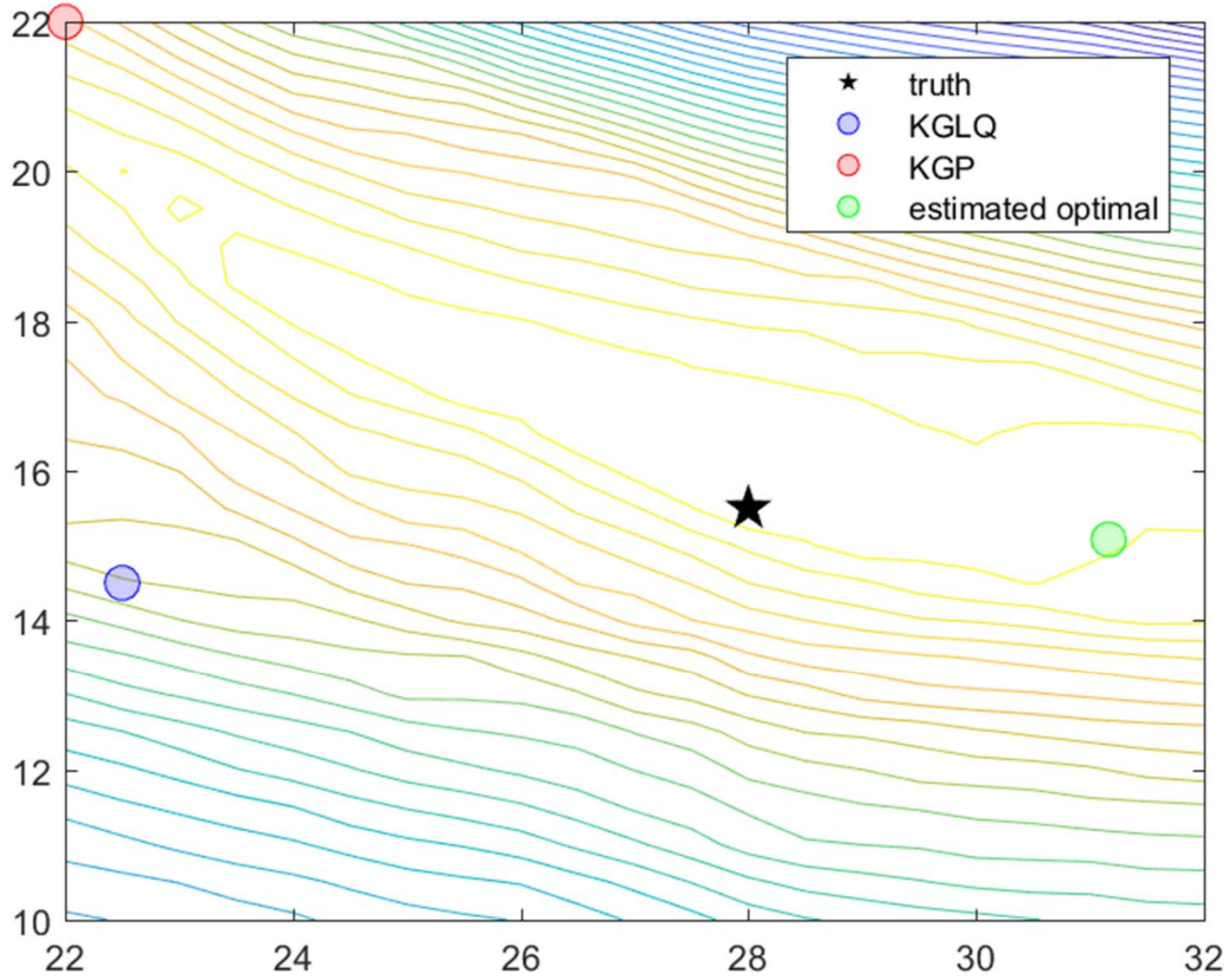
  - » However, this compact form is not very useful. We need to go back to the full form:

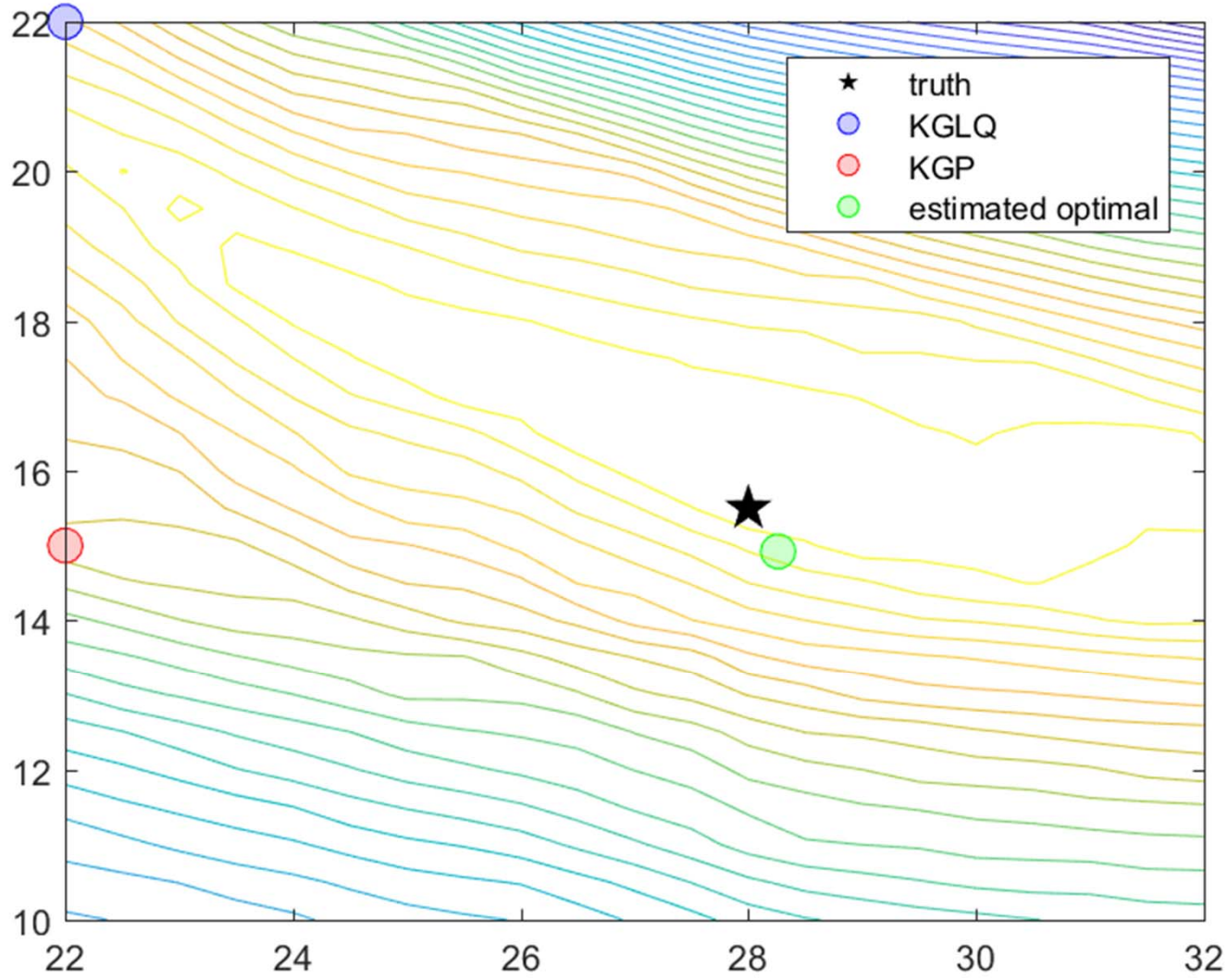
$$\max_{\pi} \mathbb{E}\left\{\mathbb{E}_{W^1,\ldots,W^N}\left\{\mathbb{E}_{\theta^{\pi,N}|W^1,\ldots,W^N}\left\{\mathbb{E}_{W|\theta^{\pi,N}}\left\{F(\theta^{\pi,N},W) \mid \theta^{\pi,N}\right\} \mid W^1,\ldots,W^N\right\}\right\} \mid S_0\right\}$$

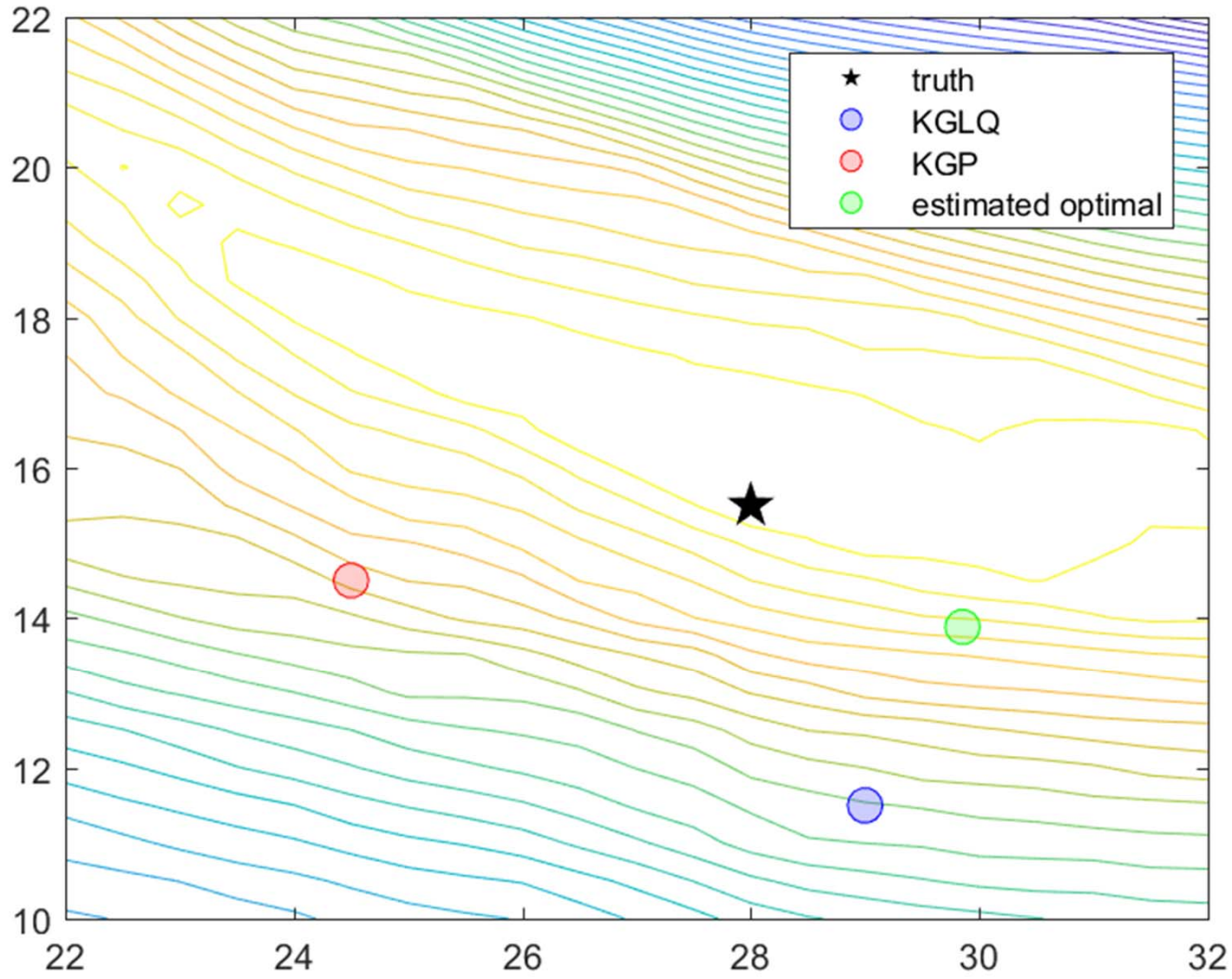
  - » Now, we just simulate each policy, generating many sample paths and taking an average.
  - » The first formulation is the most compact, but you need to understand that it translates to the nested formulation.

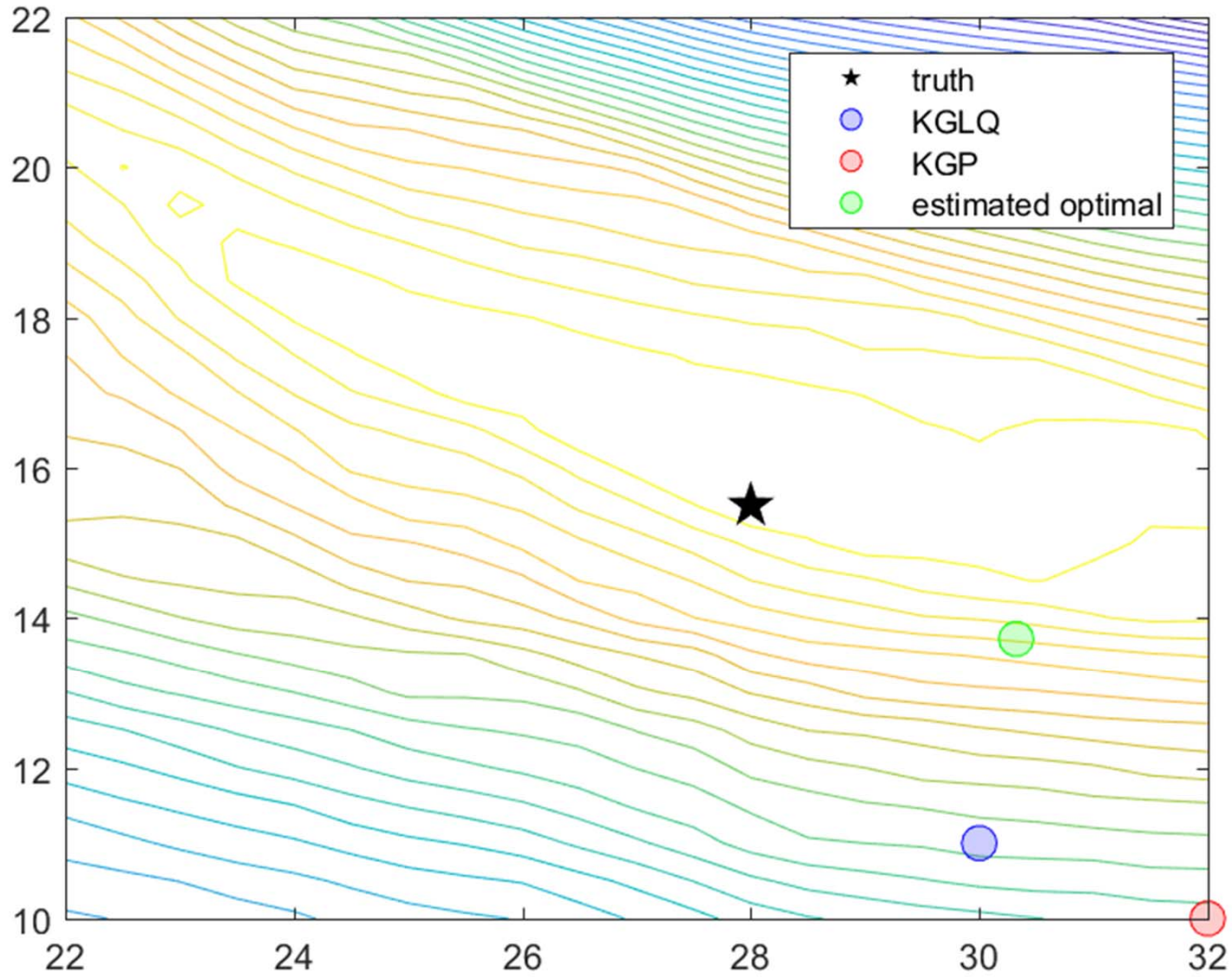
- Tests of a derivative-free policy search procedure
  - » Work of Michael Li ‘20

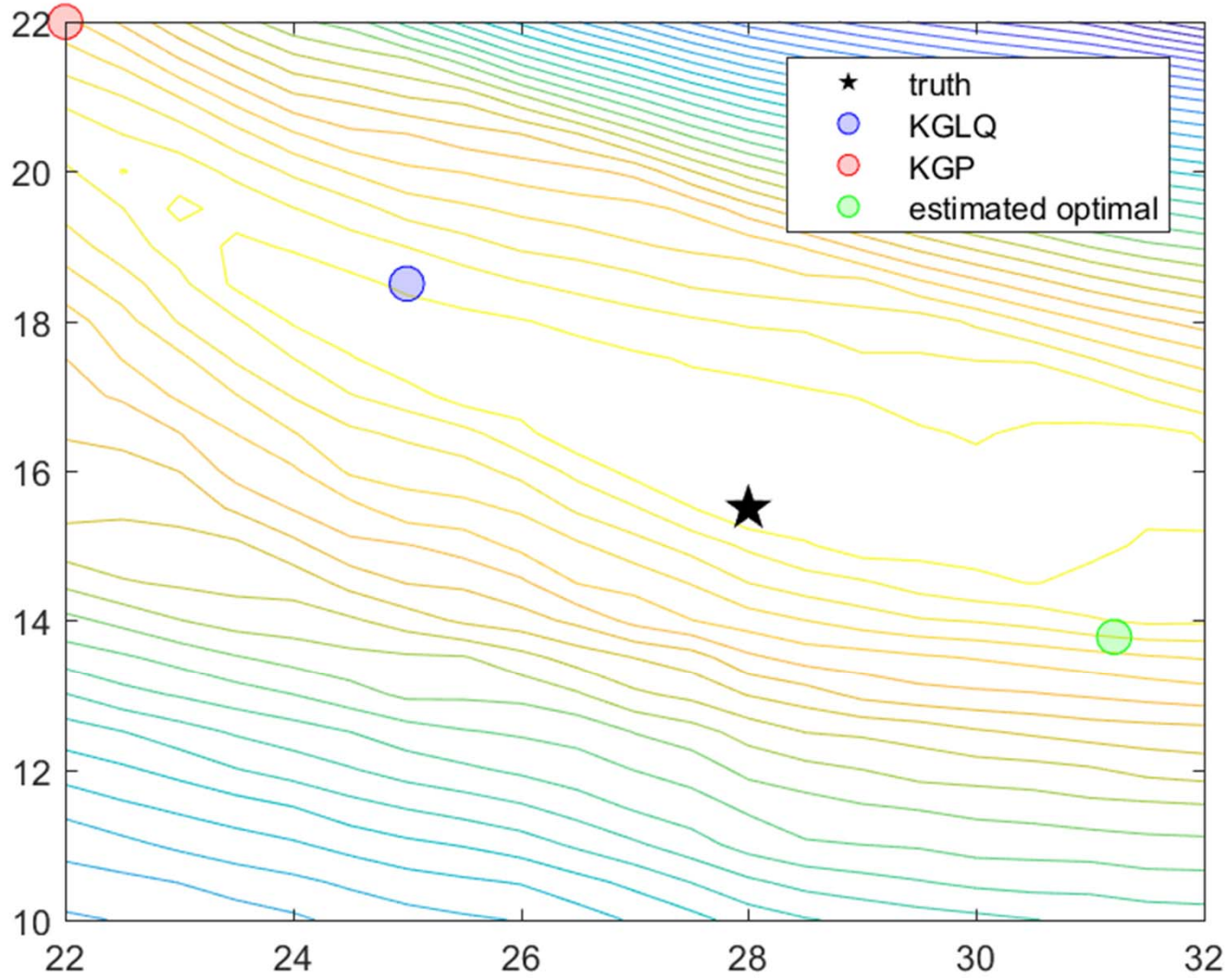

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

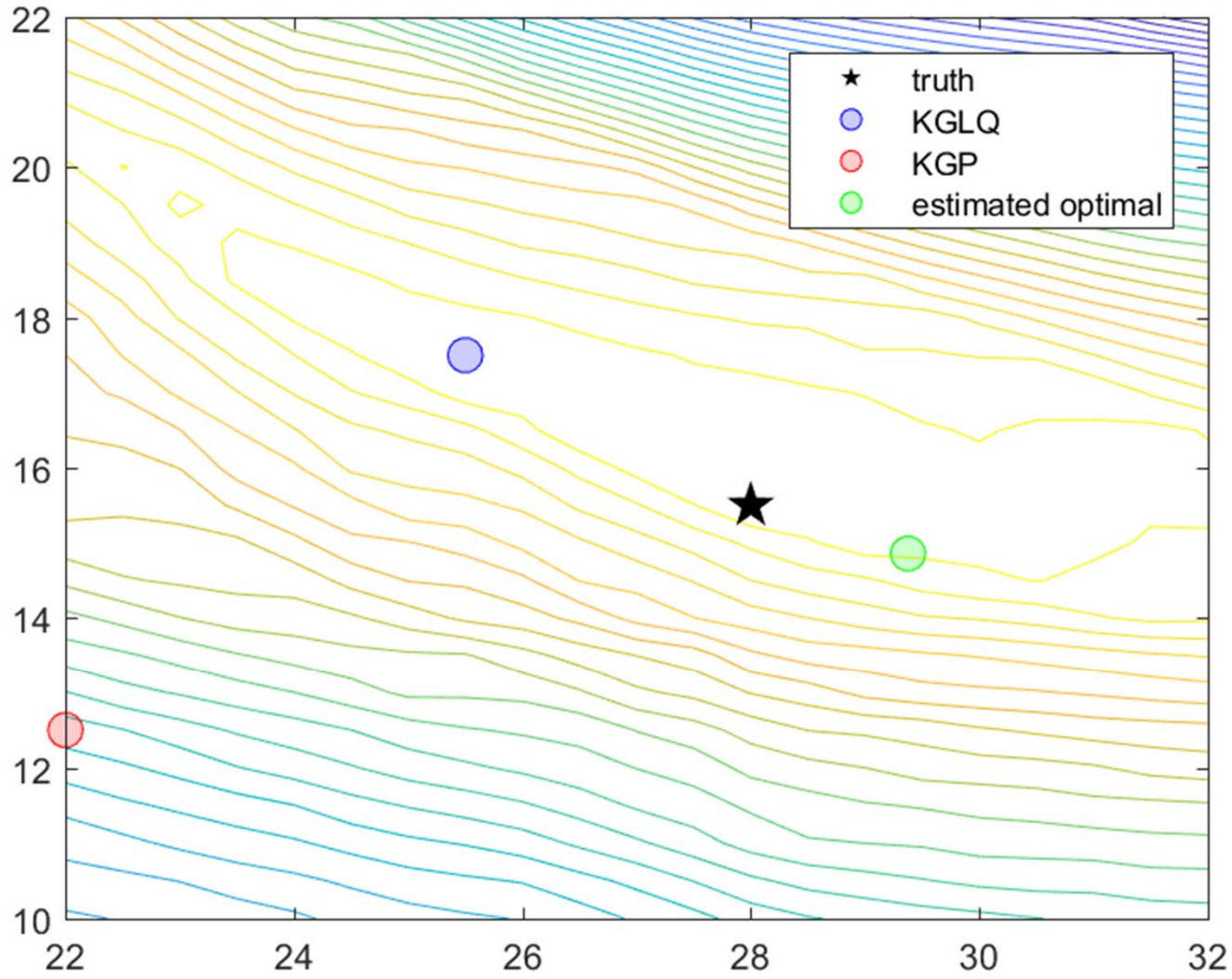

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

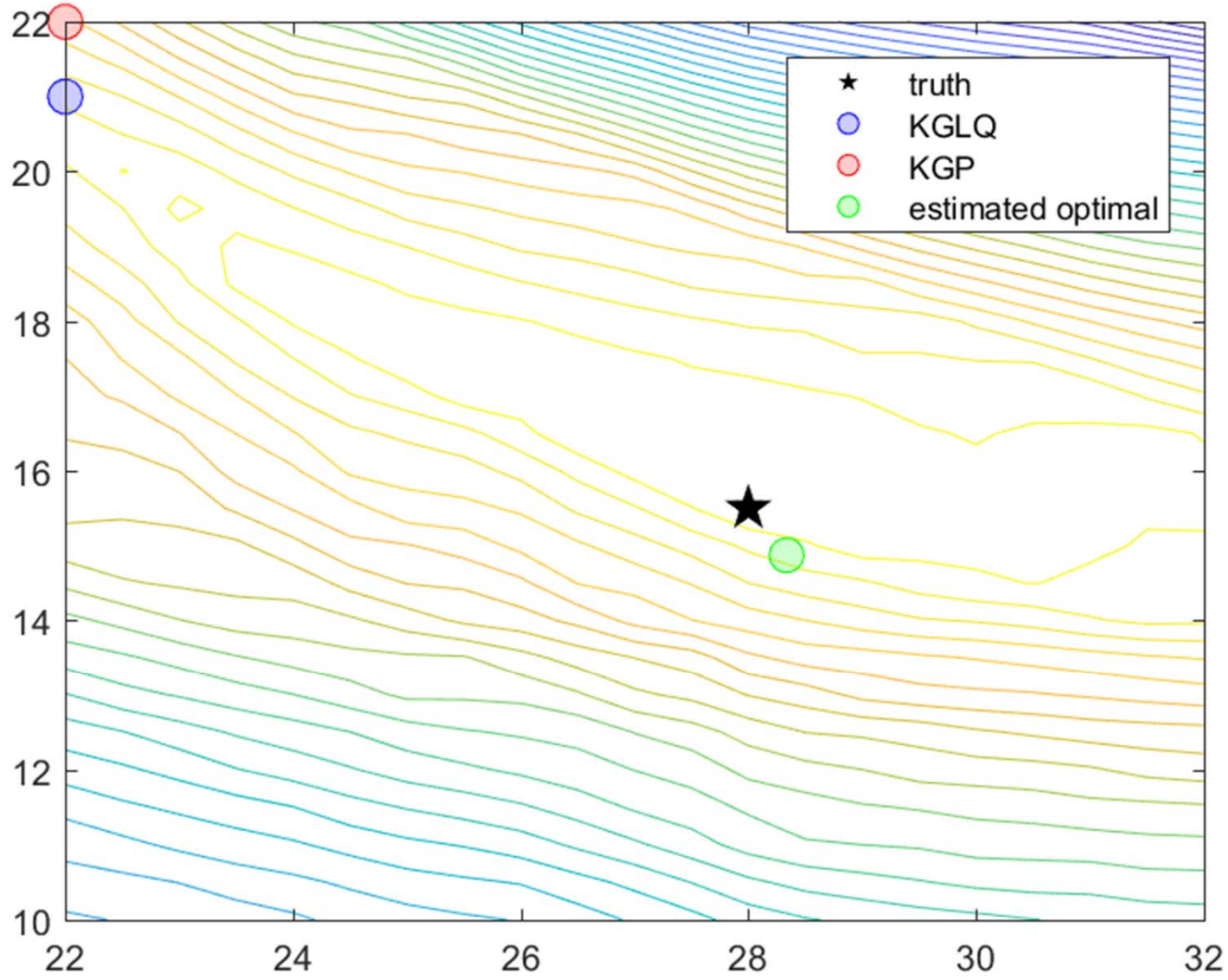

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

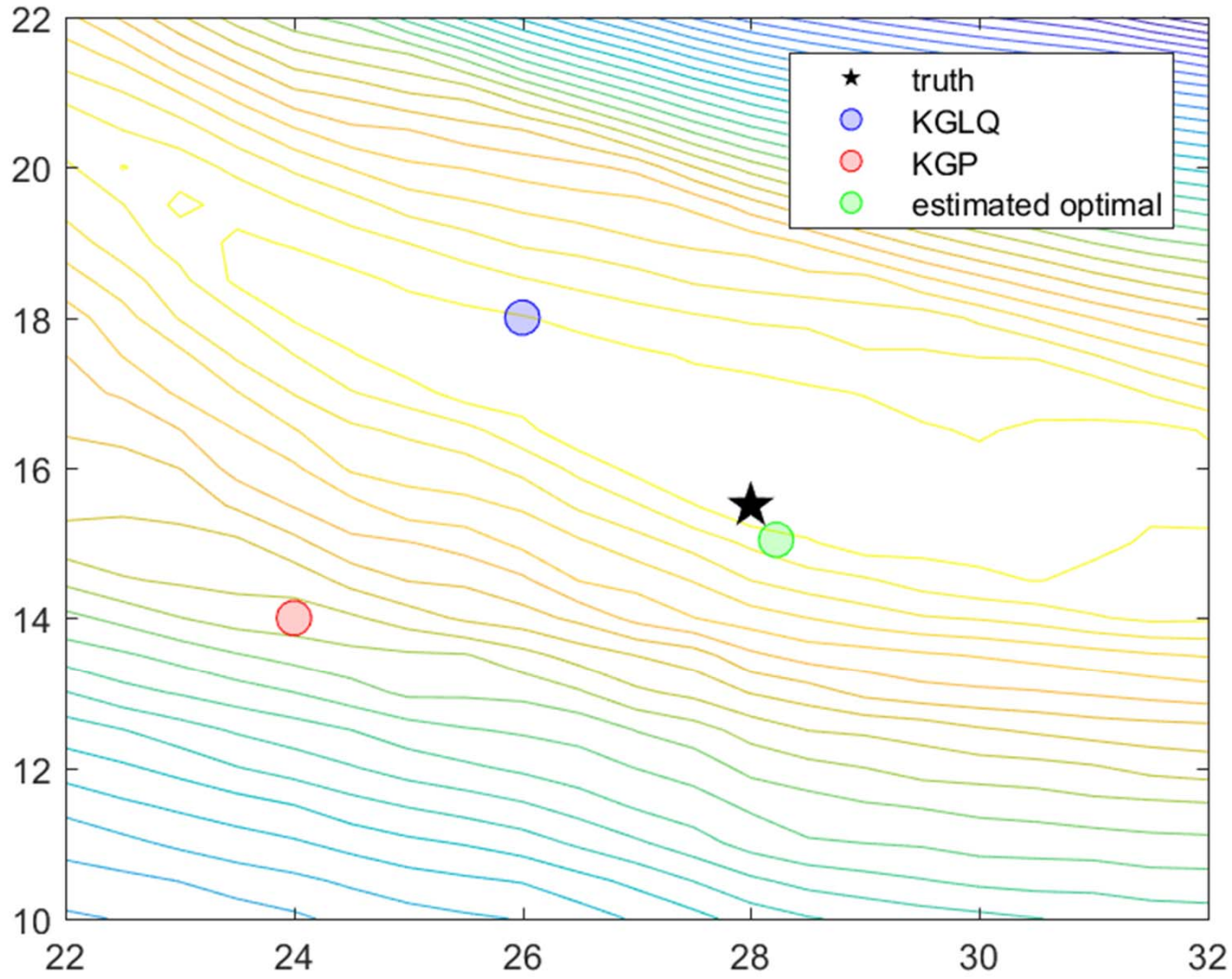

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

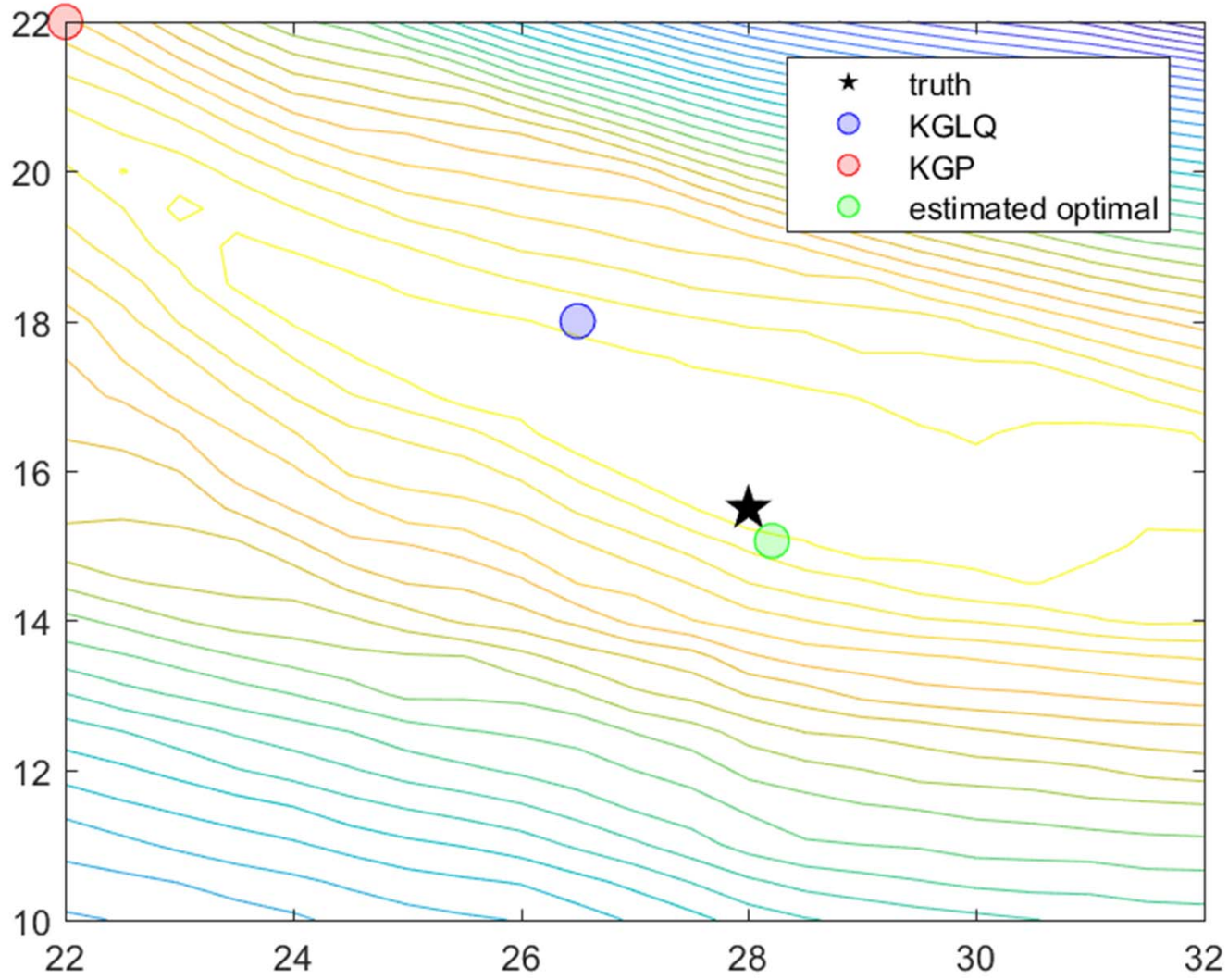

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

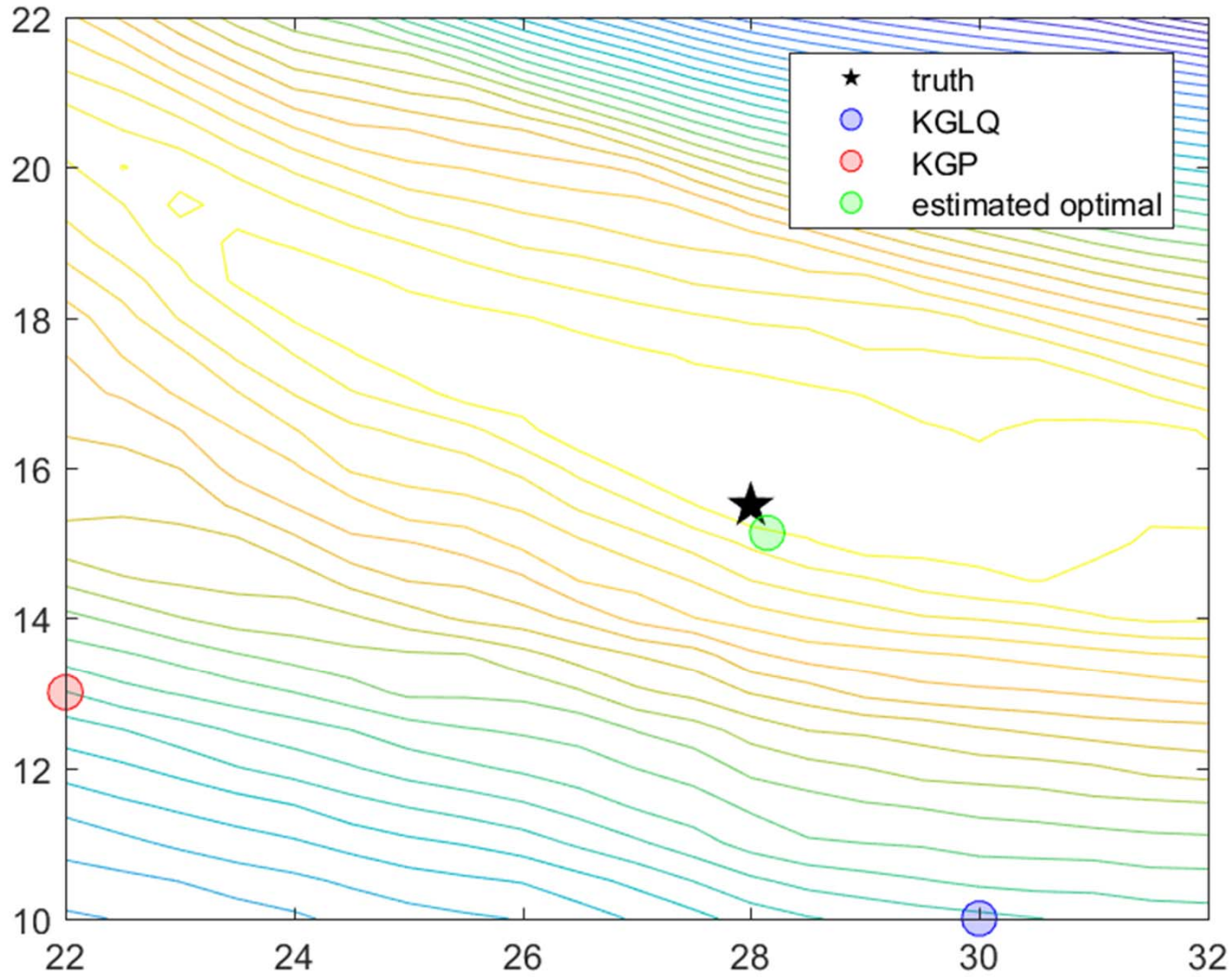

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

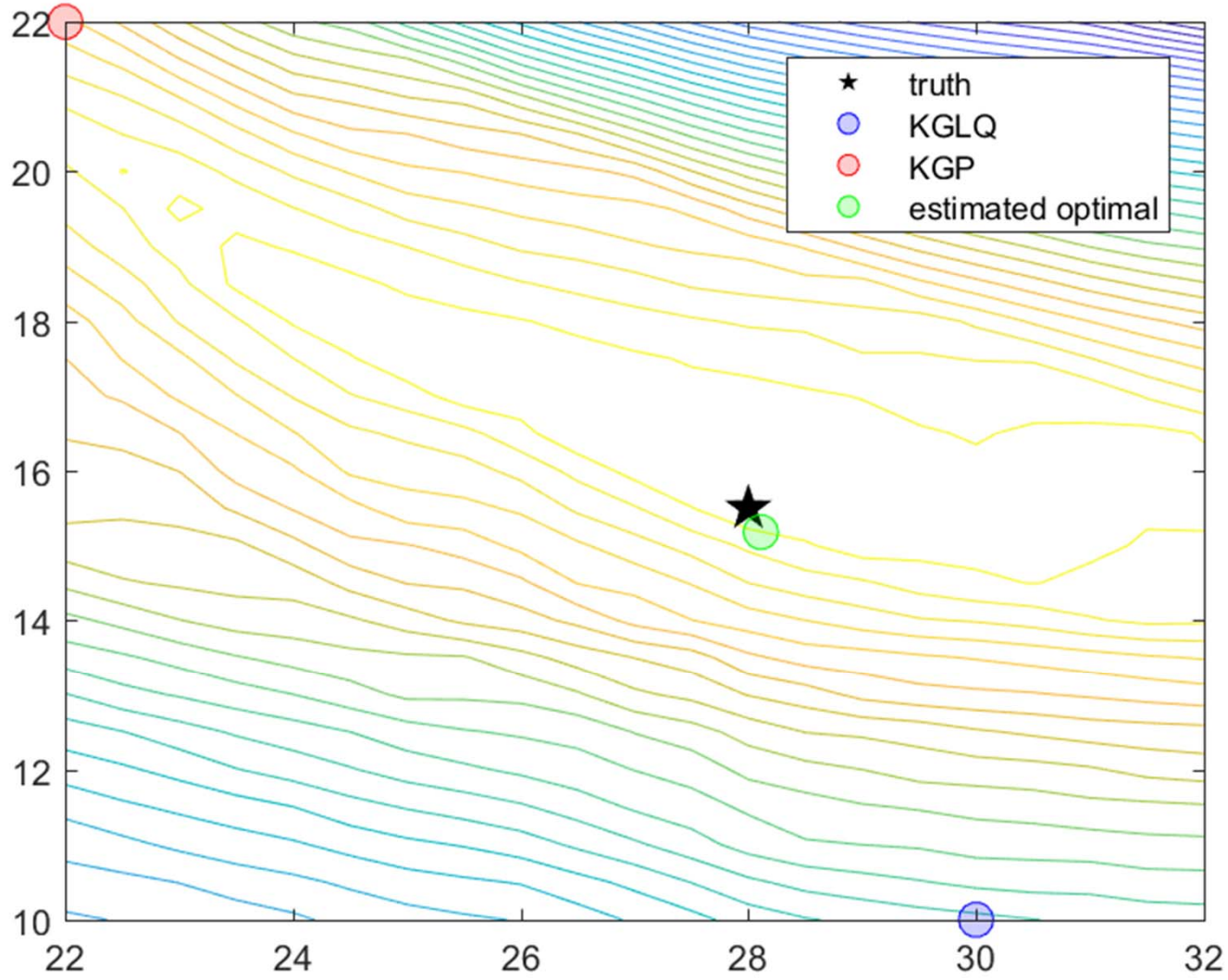

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

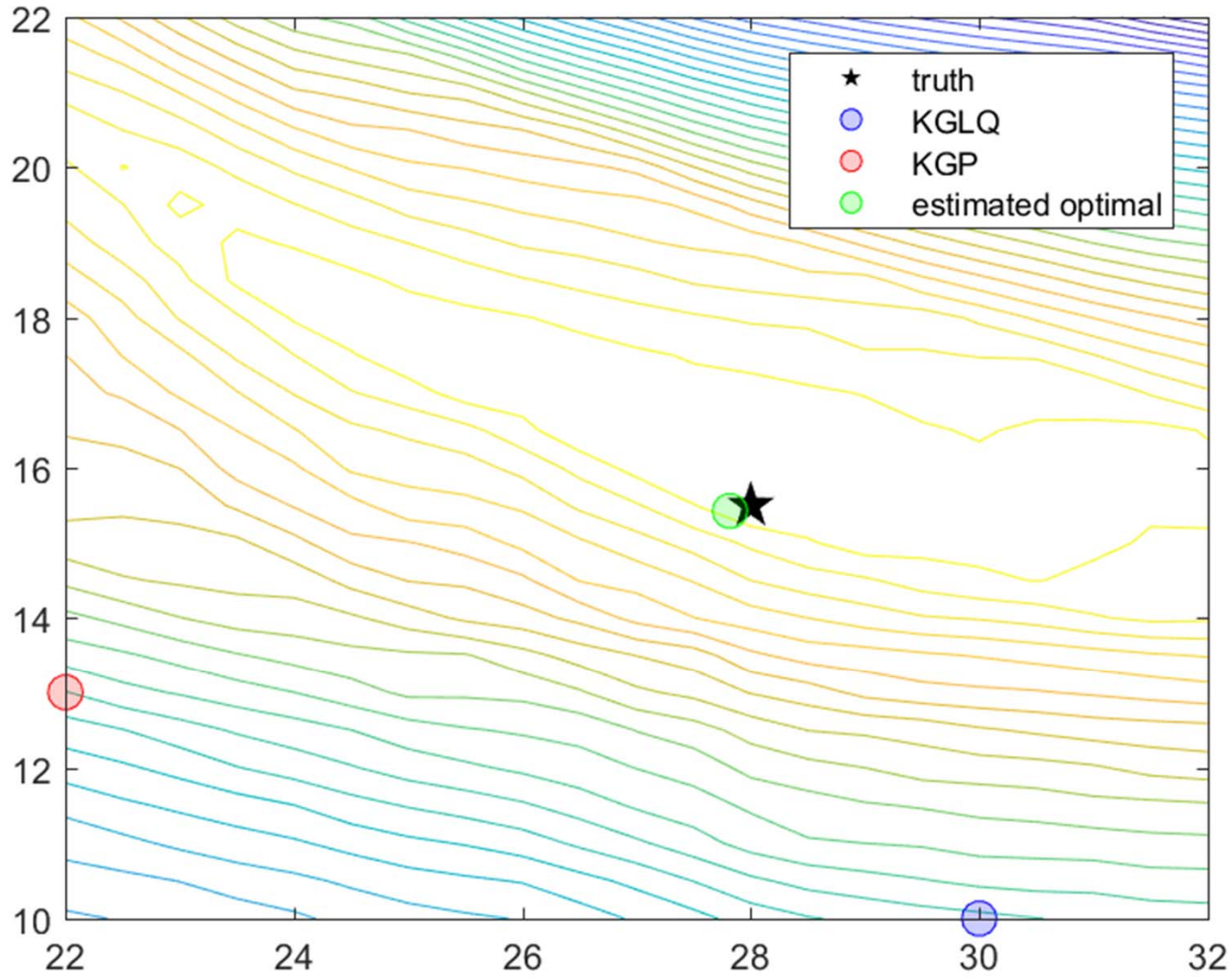

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

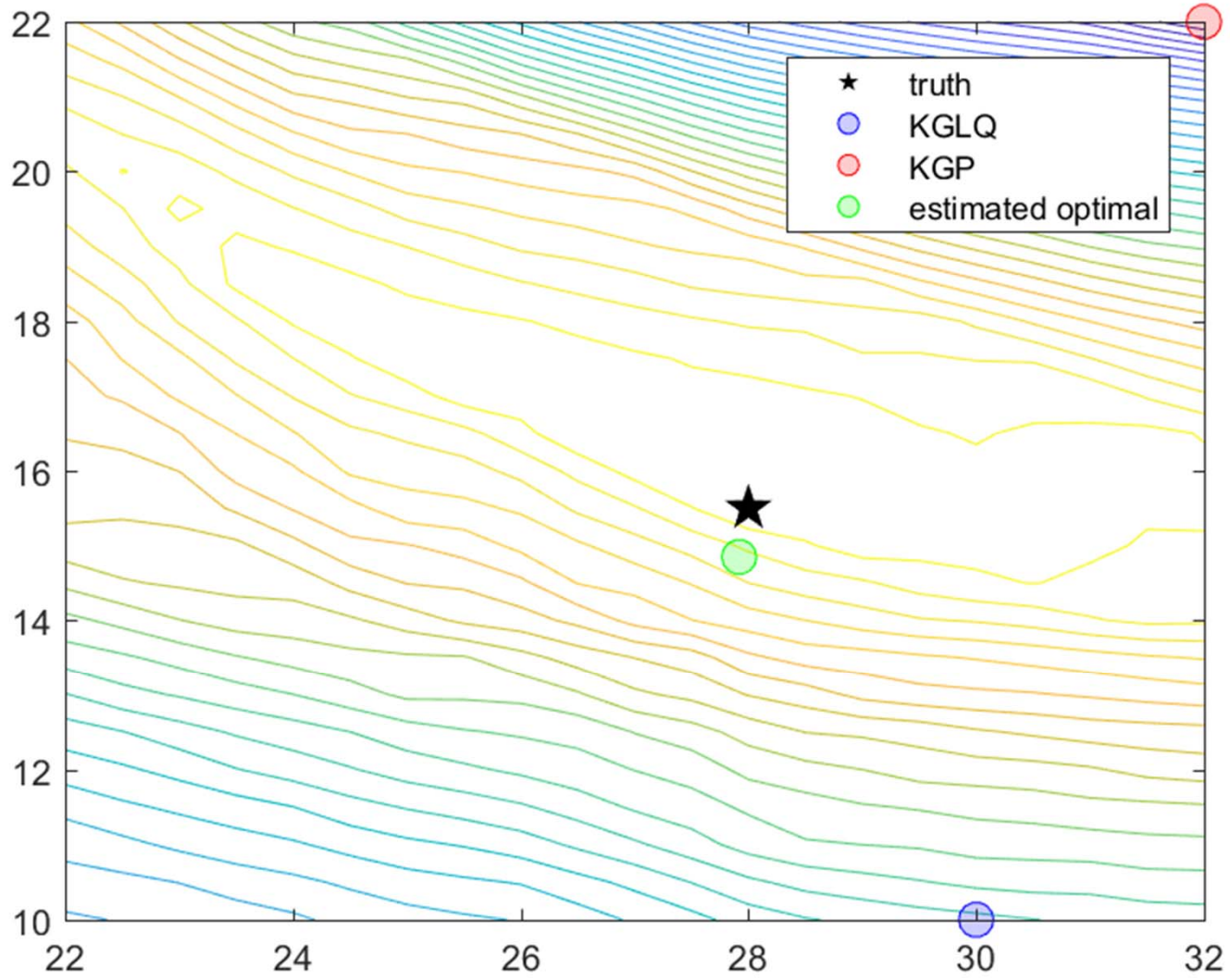

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

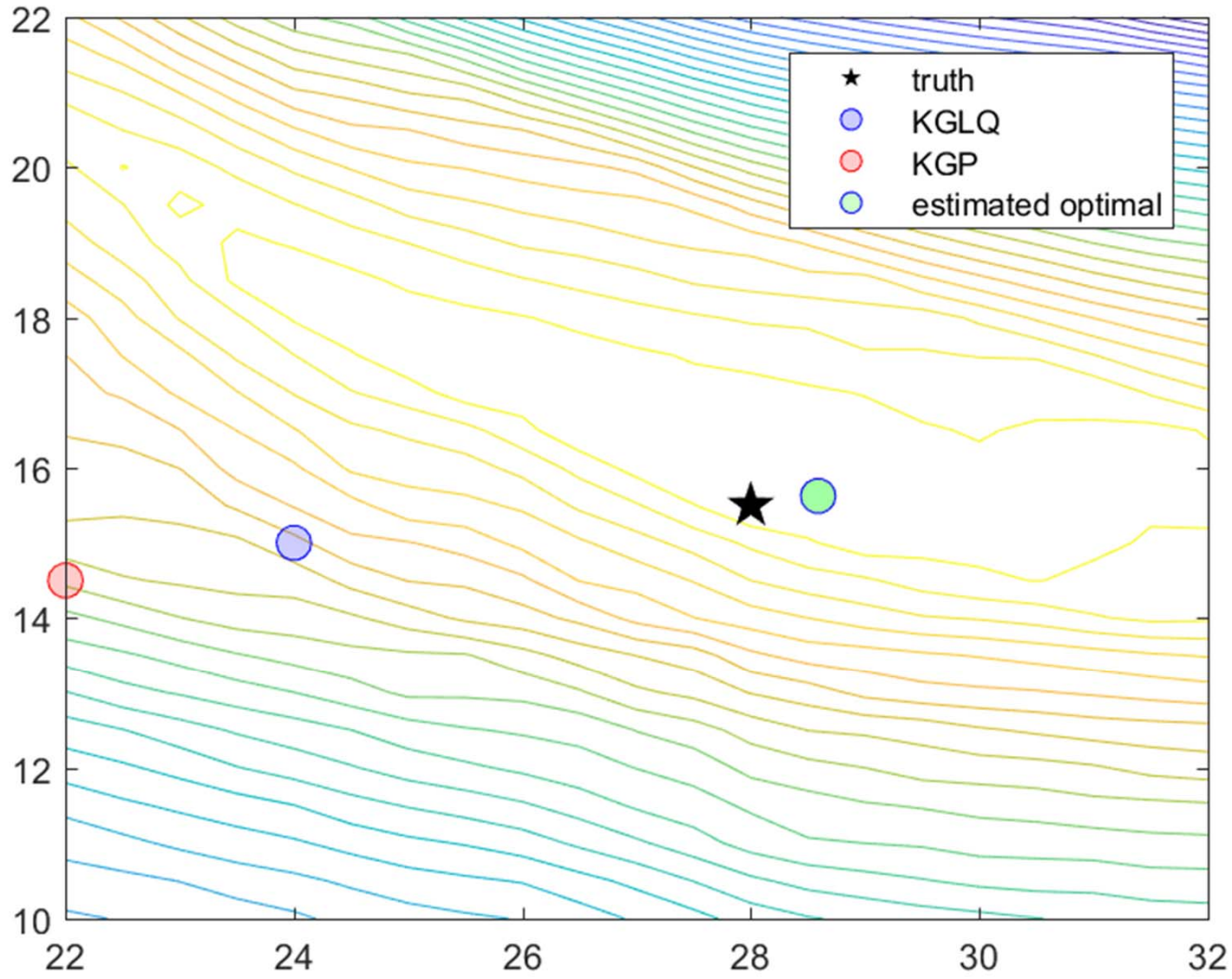

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32

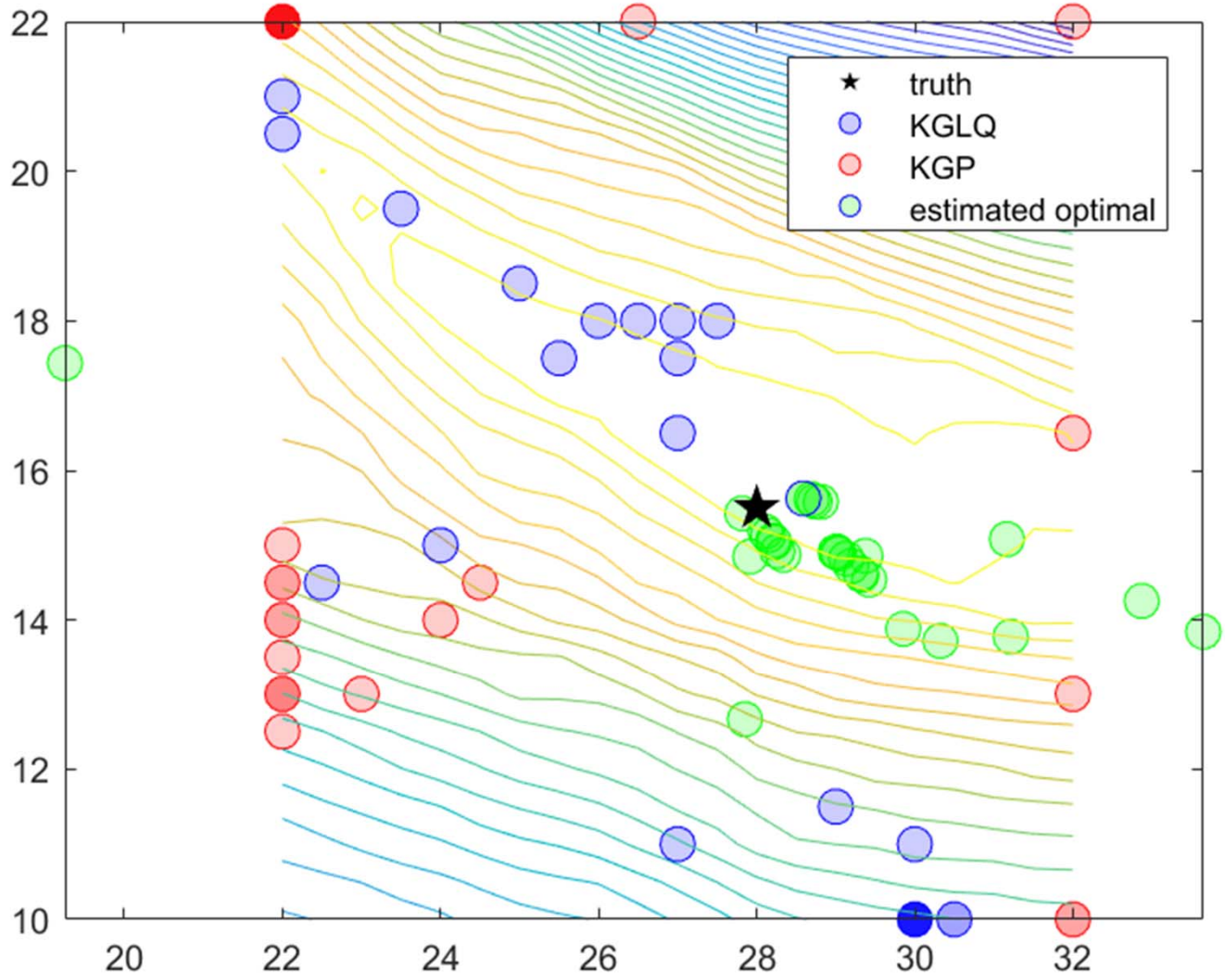

truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32


truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32


truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32


truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32


truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
22
24
26
28
30
32


truth
KGLQ
KGP
estimated optimal
22
20
18
16
14
12
10
20
22
24
26
28
30
32

- Closing notes:
  - » Making decisions over time, under uncertainty, is the universal problem. Absolutely guaranteed you will have to do this in a professional setting.
  - » "Policy search" policies are the simplest, so they are the ones that you are going to most likely encounter in practice (or which you might make up on your own).
  - » The price of simplicity is tunable parameters. Sometimes it takes a little time to recognize them, but they are there.
  - » Tuning is hard.

Designing policies

CFA (illustrative)

- A type of CFA
  - » Myopic policy:

$$X^{\pi}(S_t) = argmax_x C(S_t, x_t)$$

  - » Parameterized myopic:

$$X^{\pi}(S_t|\theta) = argmax_x C(S_t, x_t) + \sum_{f \in F} \theta_f \phi_f(S_t, x_t)$$

where we might have

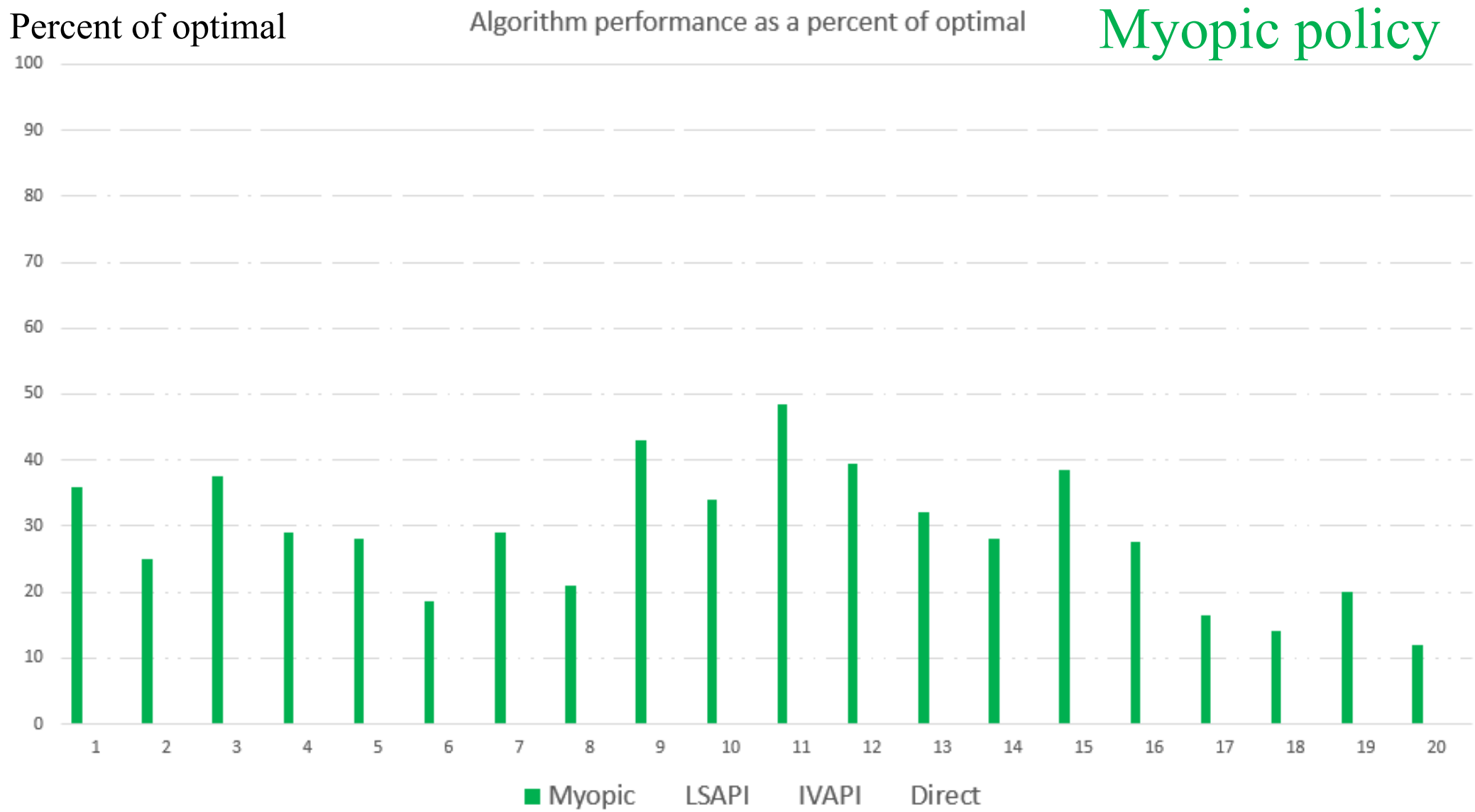
$$\sum_{f \in F} \theta_f \phi_f(S_t, x_t) = \theta_1 p_t + \theta_2 p_t^2 + \theta_3 x_t + \theta_4 x_t^2 + \theta_5 p_t x_t$$

  - » We then have to tune θ by solving

$$\max_{\theta} F(\theta) = \mathbb{E} \sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^{\pi}(S_t \mid \theta)\right)$$

- Notes
  - » The policy does not attempt to do any form of explicit lookahead. We will do this later.
- Derivative-free stochastic search
  - » Use same search methods as with basic PFA. The presence of the "arg max" within the policy does not affect how you perform policy search.

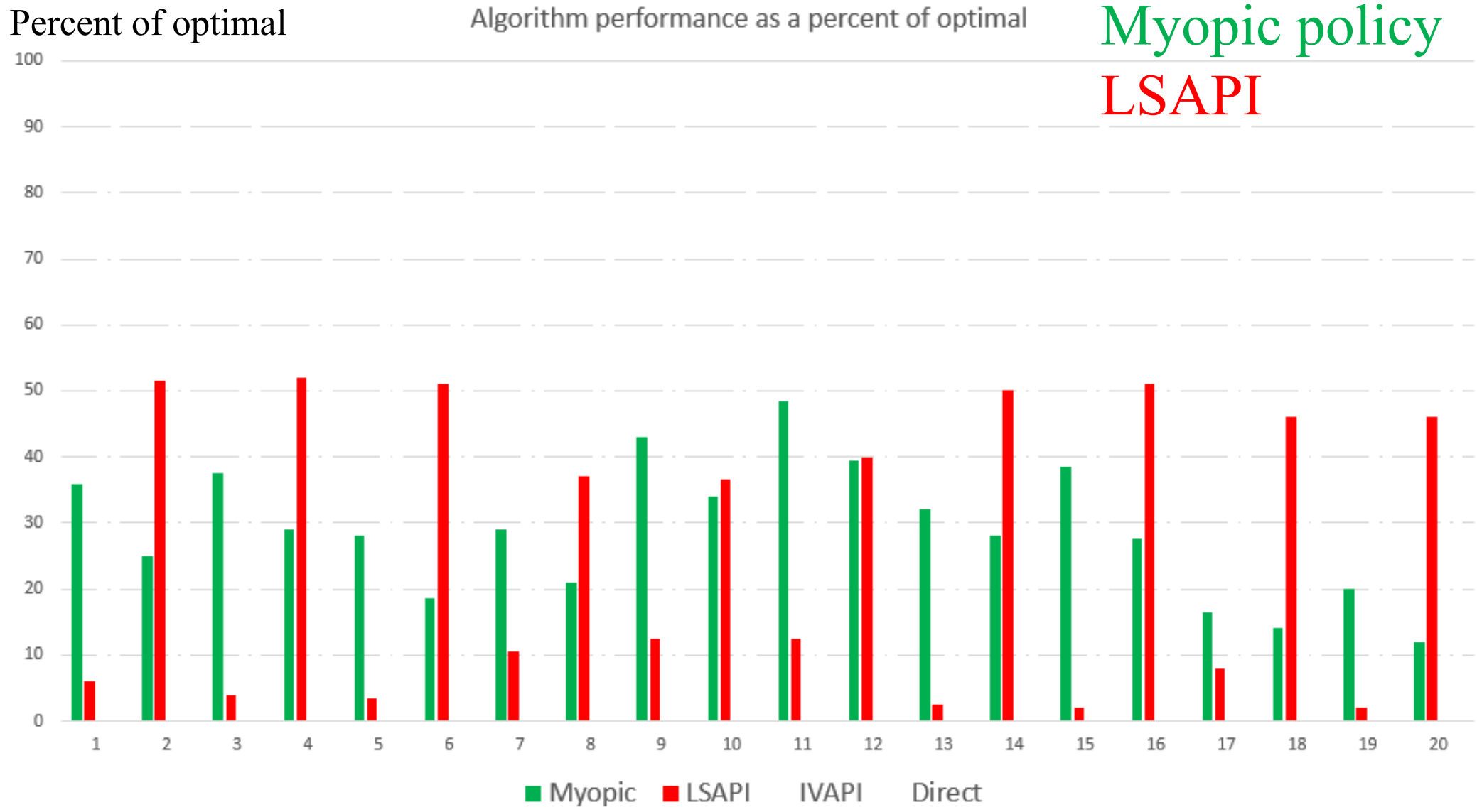
Optimizing storage



Myopic policy

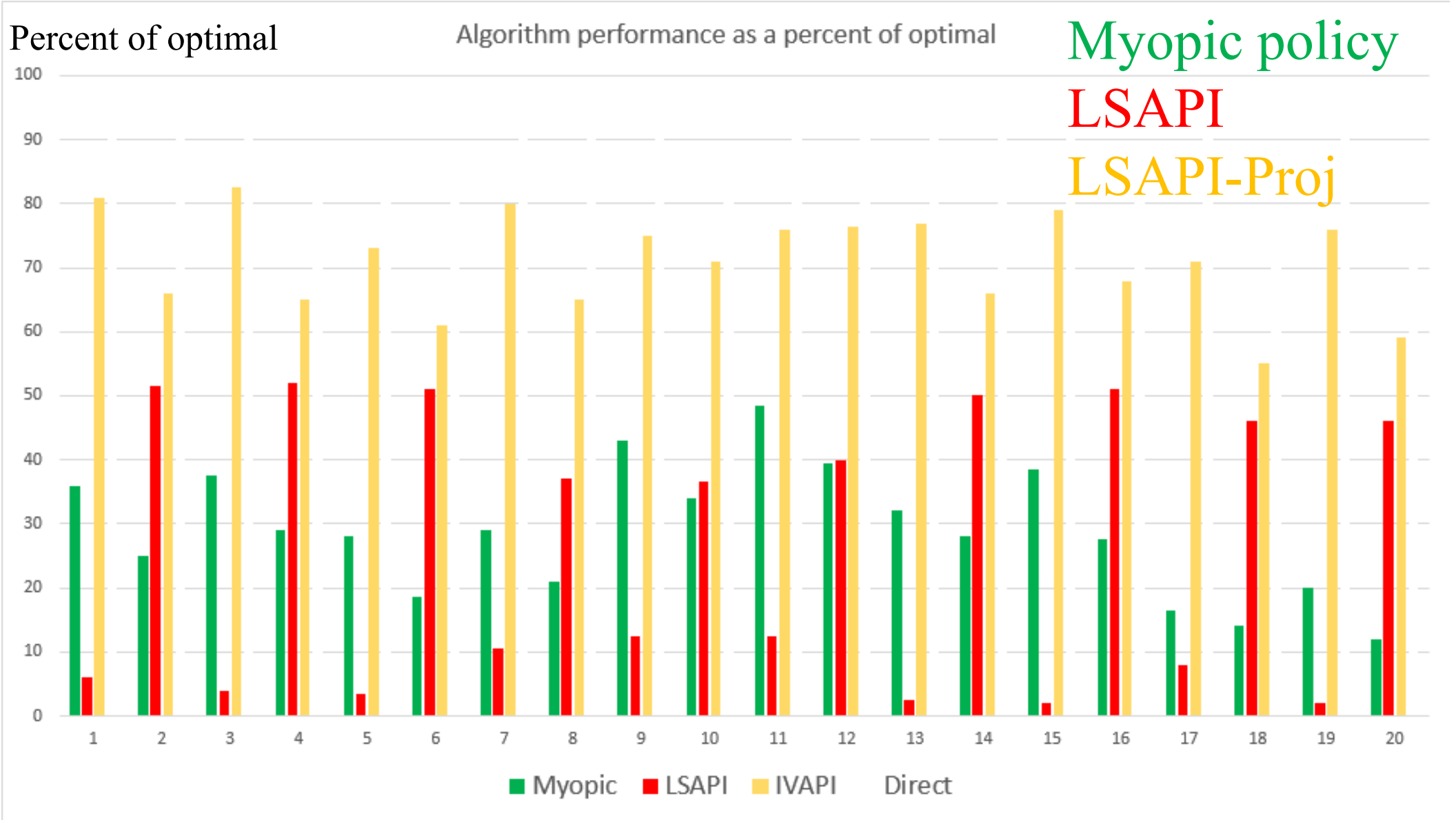
For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Optimizing storage



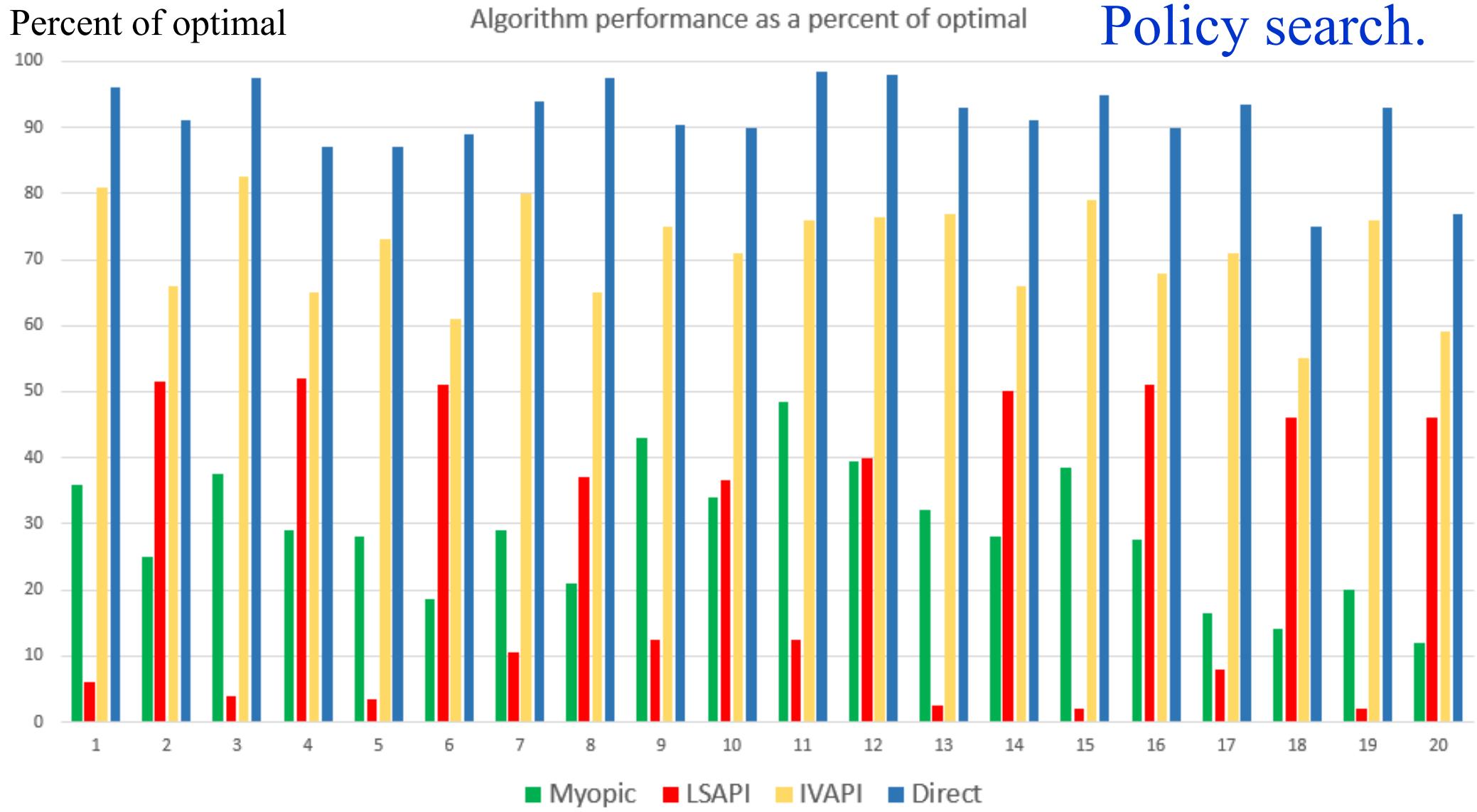
For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Optimizing storage



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Policy search



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Designing policies

Review from last week

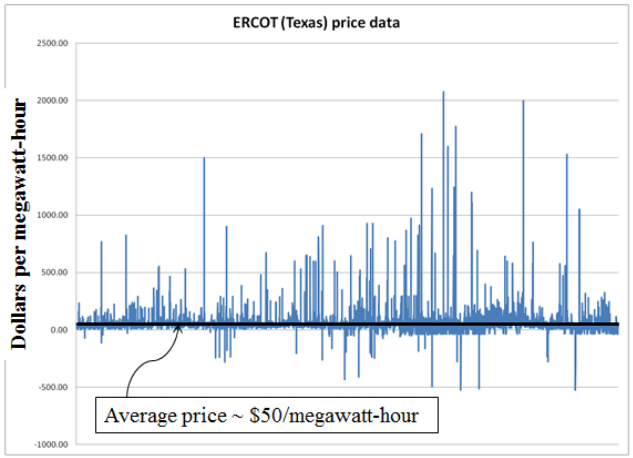
An energy storage problem

- Consider a basic energy storage problem:

» We are going to show that with minor variations in the characteristics of this problem, we can make *each* class of policy work best.

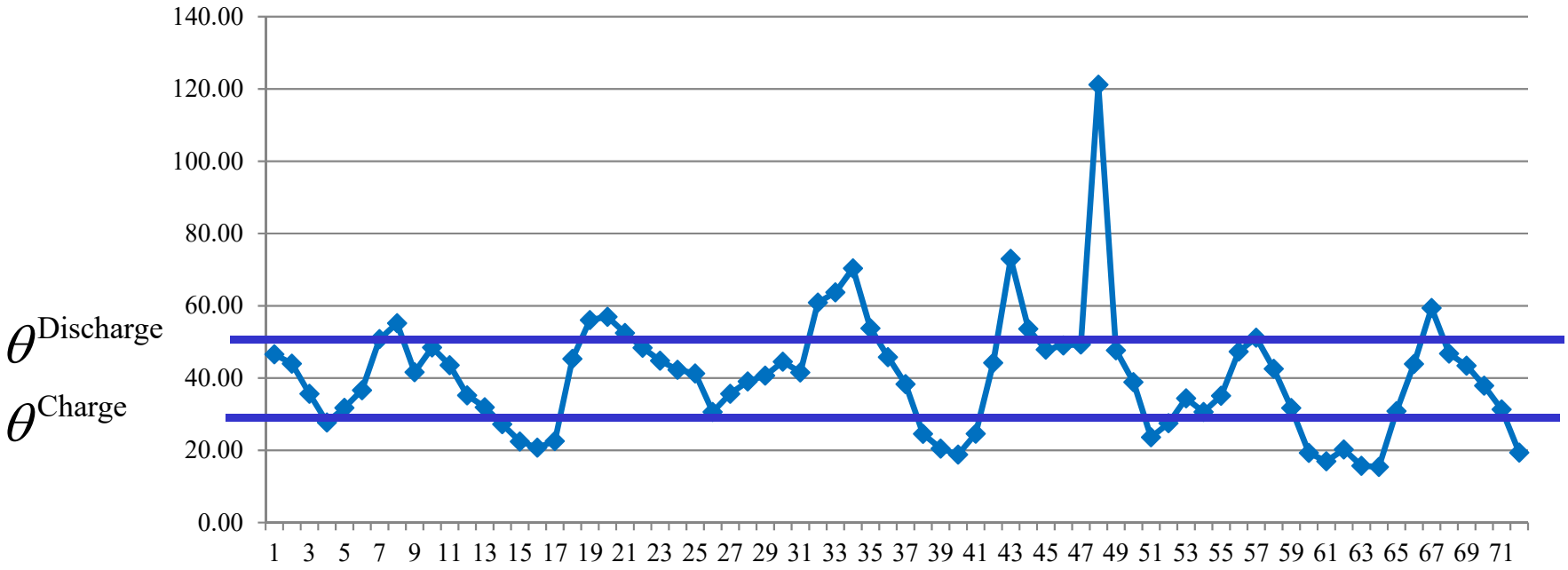
Policy function approximations

- Battery arbitrage – When to charge, when to discharge, given volatile LMPs



Policy function approximations

- Grid operators require that batteries bid charge and discharge prices, an hour in advance.



- We have to search for the best values for the policy parameters θ^{Charge} and $\theta^{\text{Discharge}}$.

8.4.1 Buy low, sell high

A buy low, sell high policy works on the simple principle of charging the battery when the price falls below a lower limit, and then sells when the price goes above an upper limit. The policy can be written as

$$X^{low-high}(S_t|\theta) = \begin{cases} -1 & \text{If } p_t \leq \theta^{buy} \\ 0 & \text{If } \theta^{buy} < p_t < \theta^{sell} \\ +1 & \text{If } p_t \geq \theta^{sell}. \end{cases} \tag{8.11}$$

We now have to tune $\theta = (\theta^{buy}, \theta^{sell})$. We evaluate our policy by following a sample path of prices $p_t(\omega)$ (or we may be observing the changes in prices $\hat{p}(\omega)$). Assuming we are generating sample paths from a mathematical model, we can generate sample paths $\omega^1, \ldots, \omega^N$. We can then simulate the performance of the policy over each sample path and take an average using

$$\overline{F}^{low-high} = \frac{1}{N}\sum_{n=1}^{N} C\big(S_t(\omega^n), X^{low-high}(S_t(\omega^n)|\theta)\big).$$

Tuning θ requires solving the problem

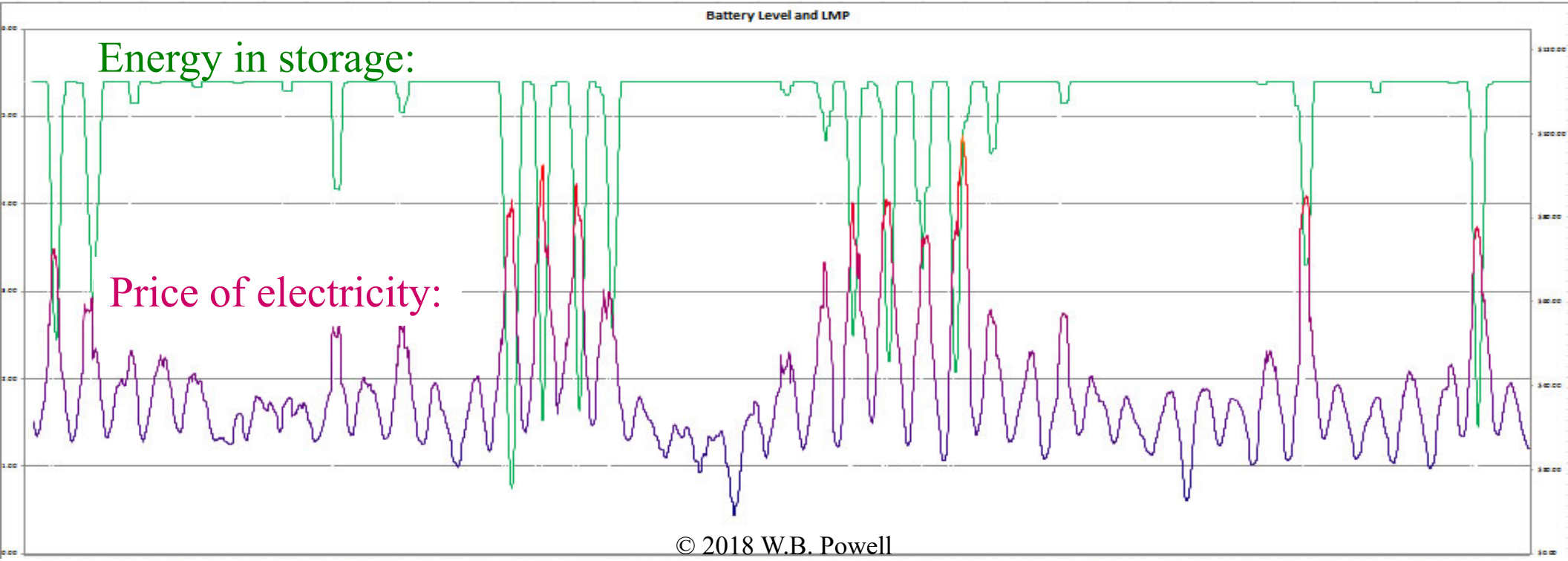
$$\max_{\theta} \overline{F}^{\pi}(\theta). \tag{8.12}$$

Since θ has only two dimensions, one strategy is to do a full grid search by discretizing each dimension, and then searching over all possible values of the two dimensions. A common discretization is to divide a region into 5-percent increments. Including the boundaries, this means we have to represent 21 values of each parameter, creating a grid of size 441 points, which is manageable (although not trivial) for most problems.

Policy function approximations

- Our policy function might be the parametric model (this is nonlinear in the parameters):

$$X^{\pi}(S_t \mid \theta) = \begin{cases} +1 & \text{if } p_t < \theta^{\text{charge}} \\ 0 & \text{if } \theta^{\text{charge}} < p_t < \theta^{\text{discharge}} \\ -1 & \text{if } p_t > \theta^{\text{charge}} \end{cases}$$

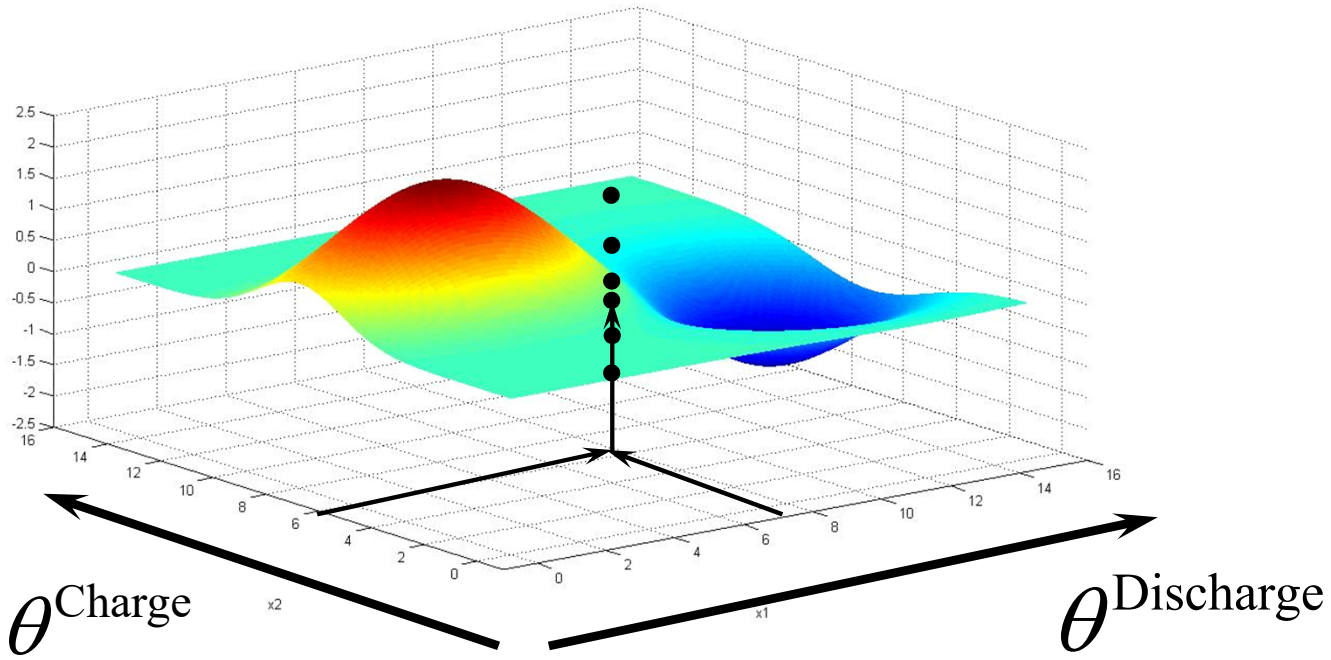


Policy function approximations

- Finding the best policy
  - » We need to maximize

$$\max_\theta F(\theta) = \mathbb{E}\sum_{t=0}^{T}\gamma^t C\left(S_t, X_t^\pi(S_t \mid \theta)\right)$$

  - » We cannot compute the expectation, so we run simulations:



Derivative-free policies

- Derivative-free policies
  - » Some examples of policies are:
    - Interval estimation:

$$\Theta^{IE}(S^n, \lambda^{IE}) = \arg\max_\theta \left(\mu_\theta^n + \lambda^{IE} \sigma^n \right) \qquad \sigma_\theta^n = \text{Std. dev. of } \mu_\theta^n$$

    - Upper confidence bounding

$$\Theta^{UCB}(S^n, \lambda^{UCB}) = \arg\max_\theta \left(\bar{\mu}_\theta^n + \lambda^{UCB} \sqrt{\frac{\log n}{N_\theta^n}} \right) \qquad N_x^n = \text{No. of times } \theta \text{ is tested.}$$

    - Thompson sampling:

$$\Theta^{TS}(S^n) = \arg\max_\theta \hat{\mu}_\theta^n \qquad \hat{\mu}_\theta^n \sim N(\mu_\theta^n, \beta_\theta^n)$$

- A type of CFA
  - » Myopic policy:

$$X^{\pi}(S_t) = argmax_x C(S_t, x_t)$$

  - » Parameterized myopic:

$$X^{\pi}(S_t|\theta) = argmax_x C(S_t, x_t) + \sum_{f \in F} \theta_f \phi_f(S_t, x_t)$$

where we might have

$$\sum_{f \in F} \theta_f \phi_f(S_t, x_t) = \theta_1 p_t + \theta_2 p_t^2 + \theta_3 x_t + \theta_4 x_t^2 + \theta_5 p_t x_t$$

  - » We then have to tune θ by solving

$$\max_{\theta} F(\theta) = \mathbb{E} \sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^{\pi}(S_t \mid \theta)\right)$$

Designing policies

VFA-backward MDP

A storage problem

- Bellman's optimality equation

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

$$\begin{bmatrix} E_t^{wind} \\ P_t^{grid} \\ D_t^{load} \\ R_t^{battery} \end{bmatrix}$$

$$\begin{bmatrix} x_t^{wind-battery} \\ x_t^{wind-load} \\ x_t^{grid-battery} \\ x_t^{grid-load} \\ x_t^{battery-load} \end{bmatrix}$$

$$\begin{bmatrix} \hat{E}_{t+1}^{wind} \\ \hat{P}_{t+1}^{grid} \\ \hat{D}_{t+1}^{load} \end{bmatrix}$$

8.4.2 Backward dynamic programming

Step 0. Initialization:

Initialize the terminal contribution $V_{T+1}(S_{T+1}) = 0$.

Step 1. Do for $t = T, T-1, \ldots, 1, 0$:

Step 2. For all $s \in \mathcal{S}$, compute

$$V_t(s) = \max_x \left(C_t(s,x) + \sum_{w \in \mathcal{W}} f^W(w|s,a_t) V_{t+1}(S^M(s,x,w)) \right)$$

Figure 8.6 A backward dynamic programming algorithm.

One way to get an optimal solution (that is, an optimal policy) is to use what is known as backward dynamic programming. A vanilla implementation of backward dynamic programming is given in figure 8.6 which exhibits four loops:

1) The loop stepping backward in time from T to time 0.

2) The loop over all possible states $s \in \mathcal{S}$ (more precisely, this is the set of possible values of the state S_t at time t).

3) The loop that would be required to search over all possible decisions x in order to solve the maximization problem.

4) The loop over all possible values of the random variable W that is captured in the summation required to compute $V_t(s)$.

Optimizing a one-dimensional problem

- Backward dynamic programming in one dimension

Step 0: Initialize $V_{T+1}(R_{T+1}) = 0$ for $R_{T+1} = 0,1,...,100$

Step 1: Step backward $t = T, T-1, T-2, ...$

Step 2: Loop over $R_t = 0,1,...,100$

Step 3: Loop over all decisions $-(R^{\max} - R_t) \le x_t \le R_t$

Step 4: Take the expectation over exogenous information:

$$\text{Compute } Q(R_t, x_t) = C(R_t, x_t) + \sum_{w=0}^{100} V_{t+1}(\min\left\{R^{\max}, R_t - x + w\right\}) P^W(w)$$

End step 4;

End Step 3;

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

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

End Step 2;

End Step 1;

Optimizing a multi-dimensional problem

- Dynamic programming in multiple dimensions

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;

Dynamic programming

- Notes:
  - » There are potentially three "curses of dimensionality" when using backward dynamic programming:
    - The state variable – We need to enumerate all states. If the state variable is a vector with more than two dimensions, the state space gets very big, very quickly.
    - The random information – We have to sum over all possible realizations of the random variable. If this has more than two dimensions, this gets very big, very quickly.
    - The decisions – Again, decisions may be vectors (our energy example has five dimensions). Same problem as the other two.
  - » Some problems fit this framework, but not very. However, when we can use this framework, we obtain something quite rare: an optimal policy.

Dynamic programming

- Strategies for computing/approximating value functions:
  - » Backward dynamic programming
    - Exact using lookup tables
    - Backward approximate dynamic programming:
      - – Linear regression
      - – Low rank approximations
  - » Forward approximate dynamic programming
    - Pure forward pass (online or offline)
    - Double pass – Forward simulation with backward learning

Week 6 – Monday

Energy storage III

Designing policies

VFA-backward ADP

Backward ADP

- Classical backward dynamic programming
  - » Uses lookup table representations of value functions
  - » Assumes the one-step transition matrix can be computed (which is also lookup table).
  - » "Dynamic programming" does *not* suffer from the curse of dimensionality (as we show below), but lookup tables do.
  - » There are three curses of dimensionality, but often it is the state variable that causes the most problems.
  - » Before we jump to forward ADP, consider doing backward ADP.

Backward ADP

- Backward approximate dynamic programming
  - » Basic idea is to step backward in time, just as we do with classical backward dynamic programming.
  - » Instead of looping over all the states, loop over a random sample (much smaller than all states).
  - » Now, use the sample of values and states to produce an approximate value function:

    $$\bar{V}_t(S_t|\theta_t) = \theta_{t1}\phi_1(S_t) + \theta_{t2}\phi_2(S_t) + \theta_{t3}\phi_3(S_t) + \ldots$$
  - » Features $\phi_f(S_t)$ might be
    - Energy in storage R_t, price p_t, energy from wind E_t,…
    - Square of any of the above
    - Any cross products
    - Any other transformations you think might capture the behavior.

Optimizing a multi-dimensional problem

Backward ADP

Step 0: Initialize $\bar{V}_{T+1}(S_{T+1}) = 0$ for all states.

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

Step 2: Loop over a random sample of states $\hat{s}_t = (R_t, D_t, p_t, E_t)$ (one loop)

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

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

Compute $Q(\hat{s}_t, x_t) = C(\hat{s}_t, x_t) +$

$$\sum_{w_1=0}^{100}\sum_{w_2=0}^{100}\sum_{w_3=0}^{100}\bar{V}_{t+1}\left(S^M\left(\hat{s}_t, x_t, W_{t+1} = (w_1, w_2, w_3)\right)\right)P^W(w_1, w_2, w_3)$$

End step 4;

End Step 3; $\bar{V}_{t+1}(S_{t+1}|\theta_{t+1}) = \theta_{t+1,1}\phi_1(S_{t+1}) + \theta_{t+1,2}\phi_2(S_{t+1}) + \theta_{t+1,3}\phi_3(S_{t+1}) + \ldots$

Find $\hat{v}_t(\hat{s}_t) = \max_{x_t} Q(\hat{s}_t, x_t)$

End Step 2;

Use sampled $\hat{v}_t(\hat{s}_t)'s$ to find an approximate $\bar{V}_t(s)$.

End Step 1;

- Notes:
  - » You can replace the triple sum over the three random variables with a random sample.
  - » Let $\Omega = (w^1, w^2, \ldots, w^k, \ldots, w^K)$ where w^k is a random sample of the random variables $W = (w_1, w_2, w_3)$ where
    - $w_1 = \widehat{D}$,
    - $w_2 = \hat{p}$,
    - $w_3 = \widehat{E}$
  - » Now approximate the value of the state-action pair (s, x) using
    - $Q(\hat{s}_t, x_t) = C(\hat{s}_t, x_t) + \frac{1}{K}\sum_{k=1}^{K} \bar{V}_{t+1}(S_{t+1} = S^M(\hat{s}_t, x_t, w^k))$
  - » You will need to test different values of K to find the smallest value that produces a high quality policy.

8.4.3 Backward approximate dynamic programming

A powerful algorithmic strategy is known as "backward approximation dynamic programming." This approach progresses exactly as we just did in figure 8.6, with one difference. Instead of looping over all the states, we choose a random sample $\hat{S}$. We then compute the value of being in state $s \in \hat{S}$ just as we originally did, and compute the corresponding value $\hat{v}$. Assume we repeat this N times, and acquire a dataset $(\hat{s}^n, \hat{v}^n), n = 1, \ldots, N$. We can then use this to fit a linear approximation of the form

$$\overline{V}(s) = \theta_0 + \theta_1\phi_1(s) + \theta_2\phi_2(s) + \ldots + \theta_F\phi_F(s), \tag{8.13}$$

where $\phi_f(s), f = 1, \ldots, F$ is a set of appropriately chosen features.

We have found backward ADP to work exceptionally well on a small set of problems, but there are no guarantees, and its performance clearly depends on choosing an effective set of features. In one application, we reduced a run time of one month for a standard backward MDP algorithm, down to 20 minutes, with a solution that was within 5 percent of the optimal (produced by the one-month run time). However, there are no bounds or guarantees.

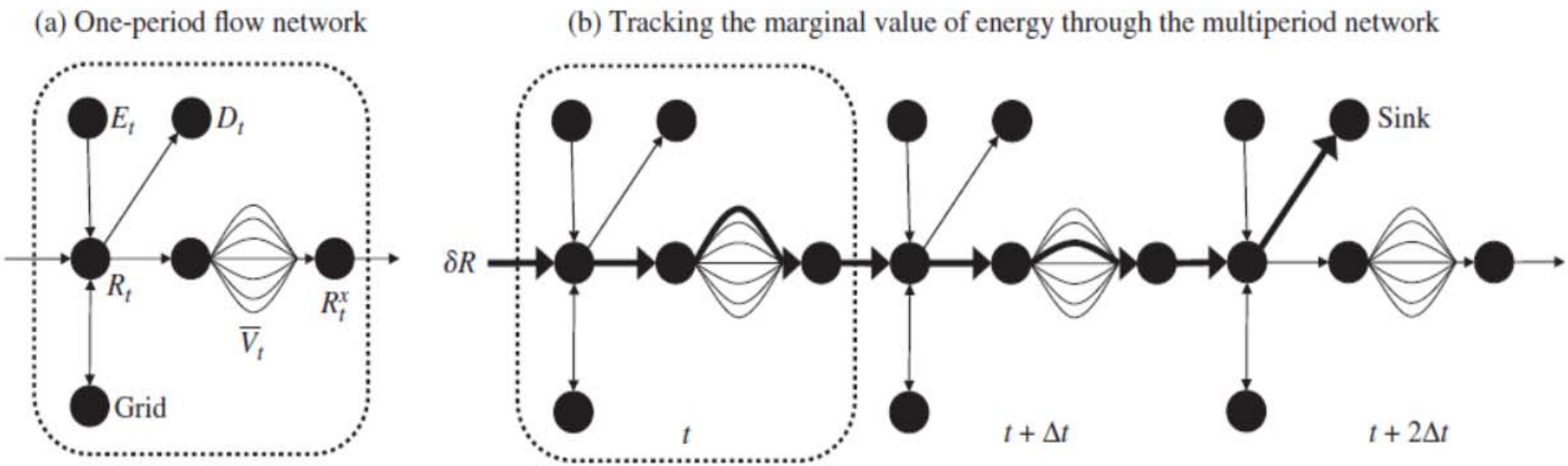
- Notes:
  - » We have been using backward ADP for the past few years with surprisingly good results – around 95 percent of the optimal policy computed using classical backward MDP.
  - » Backward ADP can be quite fast. In one application, it reduced CPU times from a *month* for classical backward MDP, to 20 *minutes*, with a solution only 4-5 percent below optimal.
  - » We have no theory to support these results.

Designing policies

VFA-forward ADP exploiting convexity

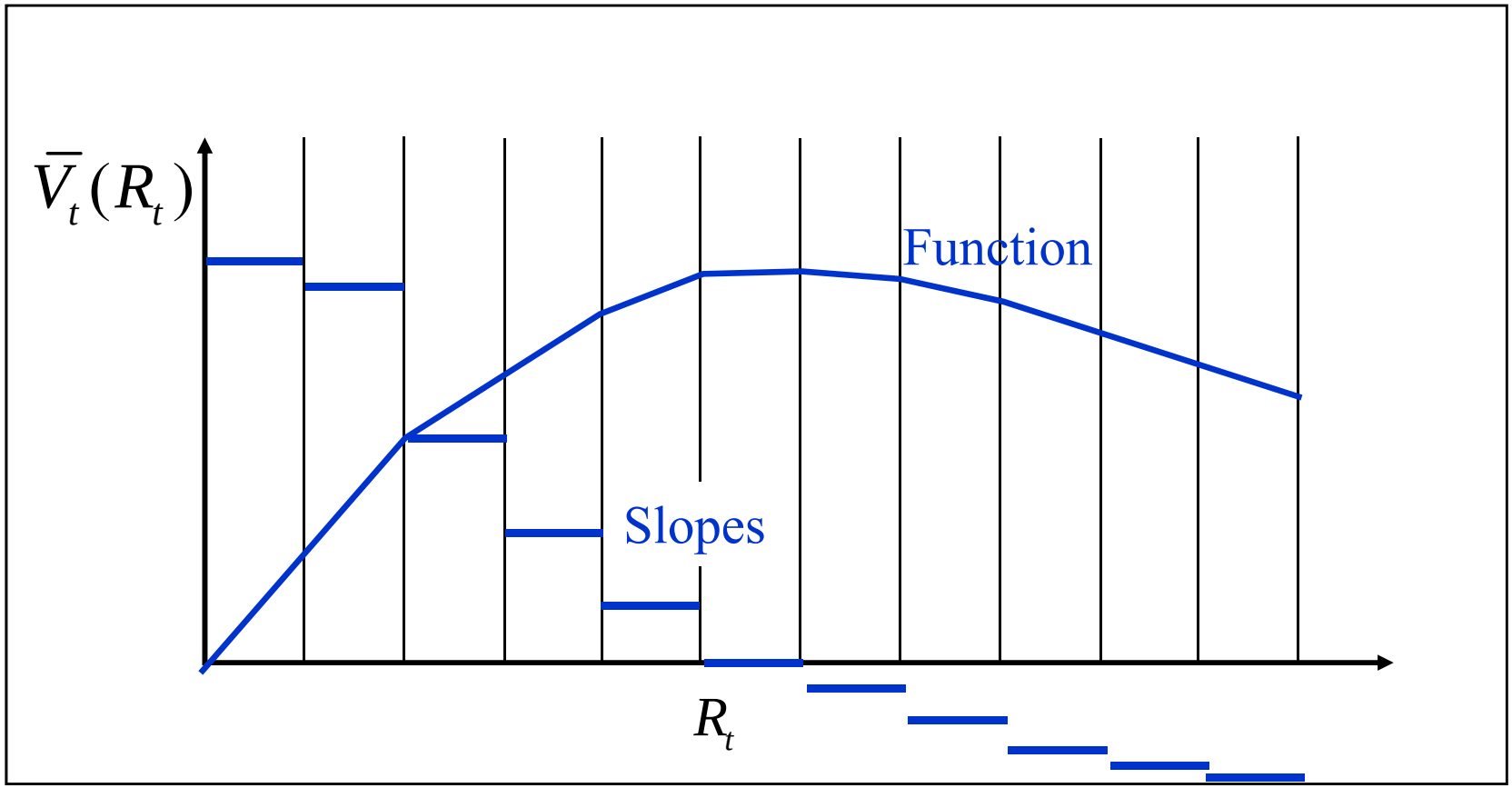
- Exploiting concavity
  - » For a single storage device, we can optimize the decision to store/charge/discharge by fixing the VFA, simulating forward in time, and then asking what we would do with a marginal increment of energy in storage.

Figure 1. Tracking the Marginal Value of Energy Through the Multiperiod Network



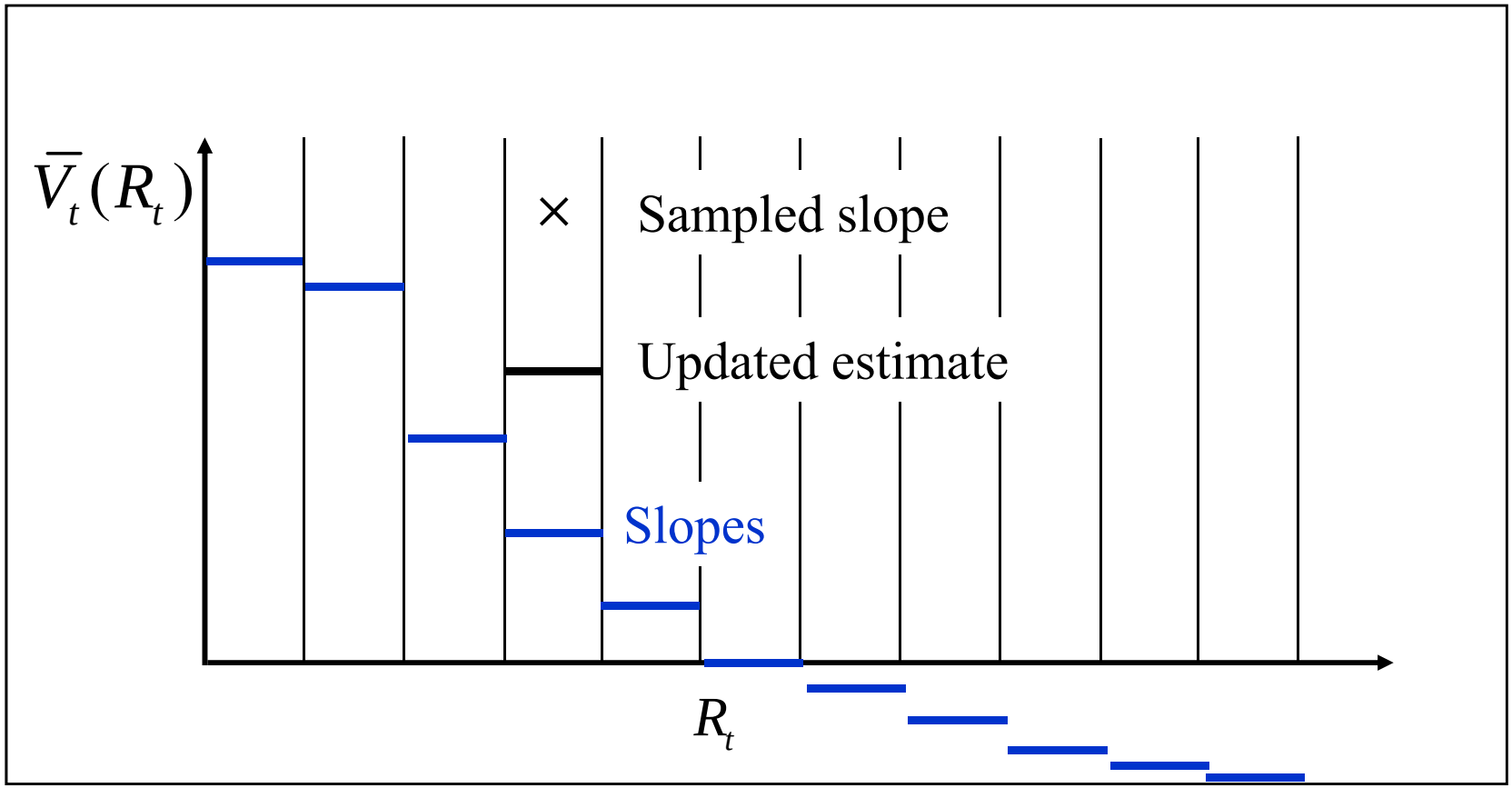
Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation, where we maintain concavity at each step.



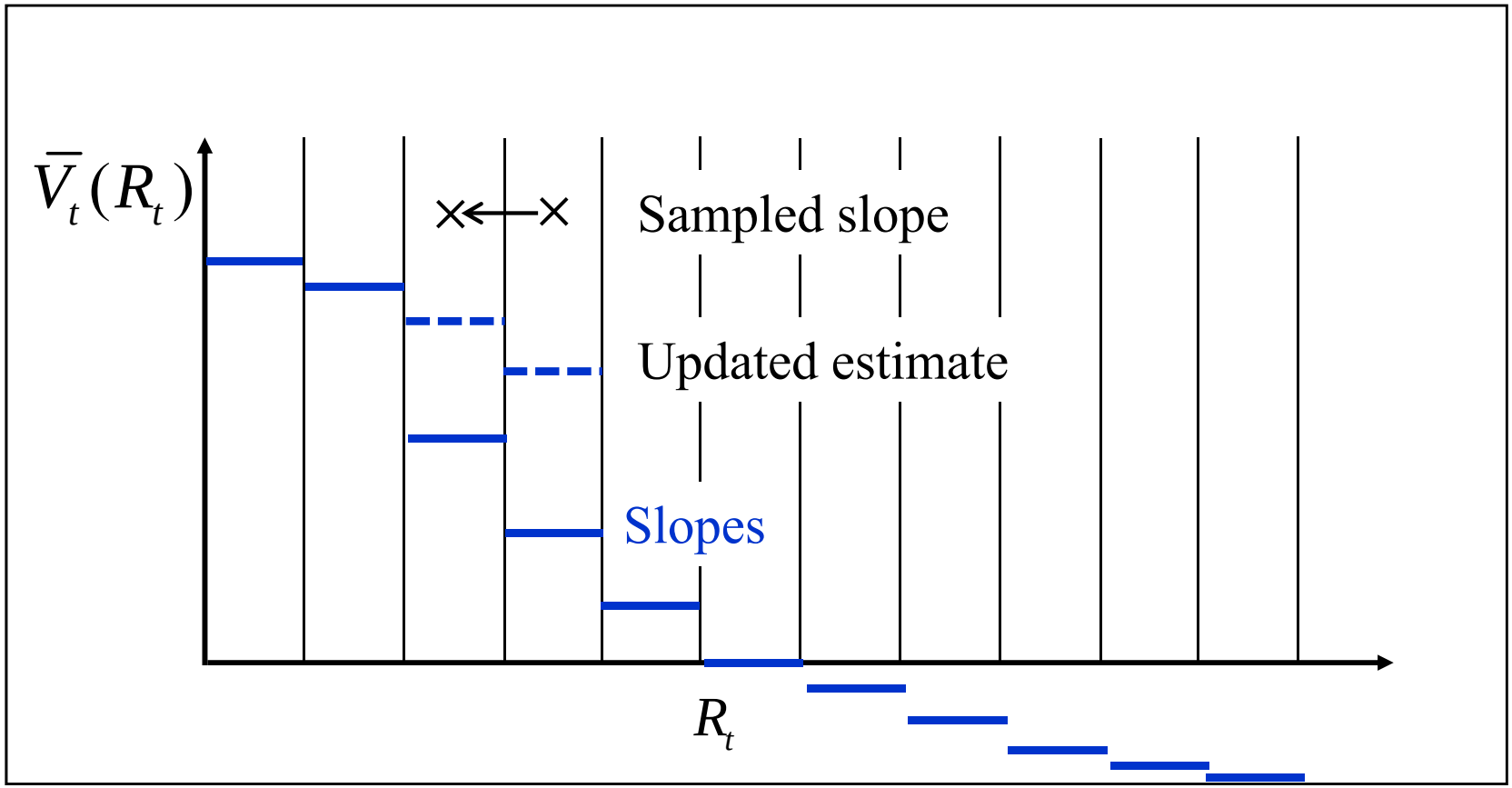
Exploiting concavity

- Maintaining monotonicity in the slopes



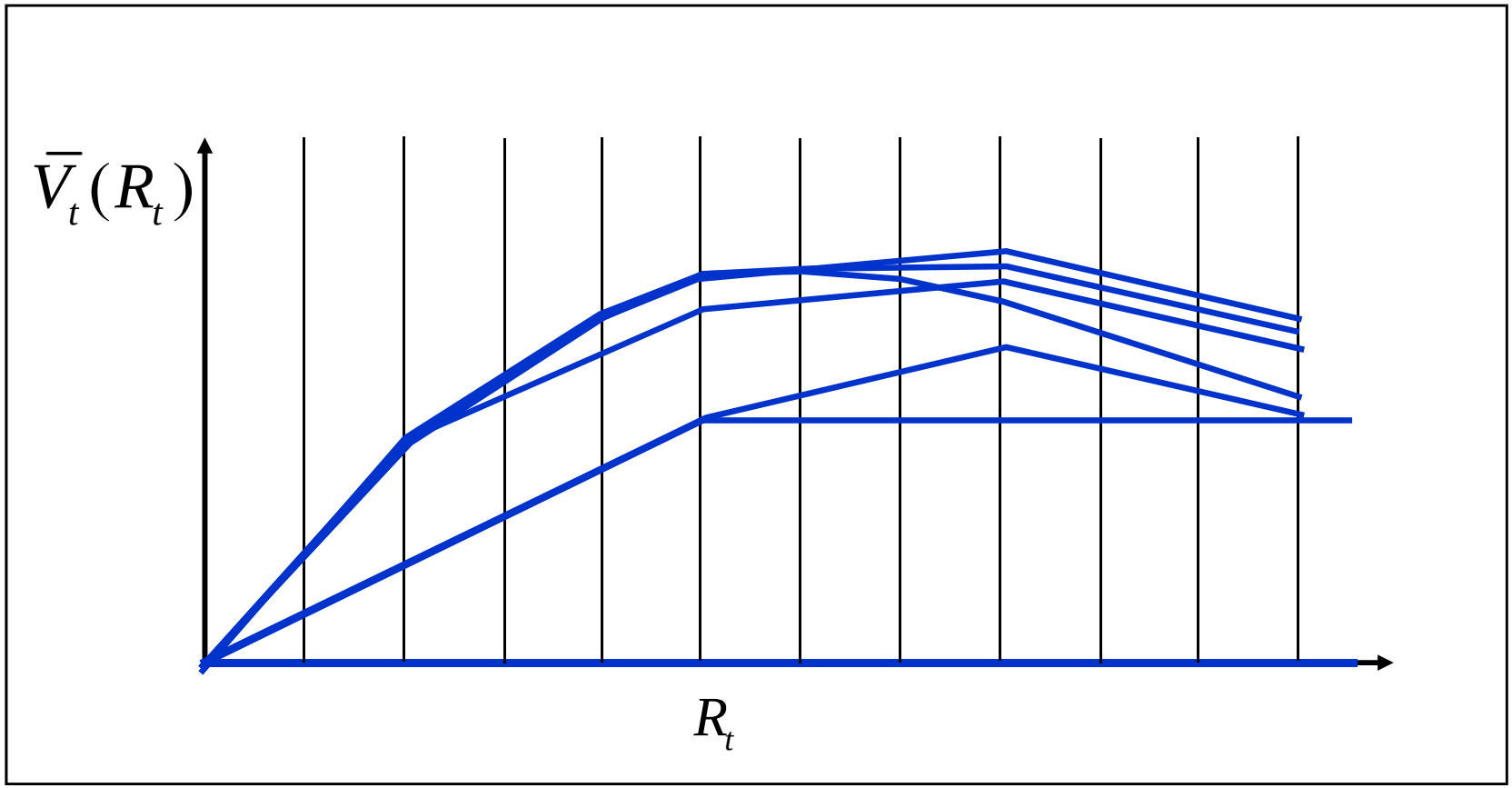
Exploiting concavity

- Maintaining monotonicity in the slopes.
  - » If there is a montonicity violation, keep smoothing the observation to the left (or right as necessary) until monotonicity is restored.



Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation, where we maintain concavity at each step.



- Forward ADP exploiting concavity
  - » If our only state variable is energy in the battery, we can exploit this when computing the value function approximation.
  - » Results are consistently near optimal:

Label	$\bar{F}^*$	F^0 (%)	F^{50} (%)	F^{250} (%)	F^{500} (%)	$F^{1,500}$ (%)	$F^{3,000}$ (%)	$F^{3,500}$ (%)	$s^{3,500}$ (%)
S1	19,480.11	44.88	91.87	98.77	99.22	99.49	99.61	99.63	±0.25
S2	18,915.05	44.60	90.37	98.63	99.16	99.32	99.47	99.52	±0.22
S3	19,730.28	44.63	90.70	98.83	99.20	99.49	99.59	99.61	±0.25
S4	19,831.66	44.20	90.70	98.88	99.32	99.53	99.60	99.61	±0.65
S5	16,806.49	57.24	97.41	98.92	99.02	98.98	99.14	99.14	±0.32
S6	17,089.29	57.17	97.14	98.78	98.84	99.18	99.55	99.58	±0.33
S7	18,104.07	56.77	97.46	98.69	98.78	99.31	99.57	99.59	±0.34
S8	19,011.24	55.85	97.72	98.65	99.02	99.42	99.58	99.59	±0.34
S9	17,963.79	57.54	96.99	98.74	98.97	99.36	99.43	99.44	±0.35
S10	19,079.16	55.88	97.91	98.85	99.02	99.42	99.55	99.56	±0.34

250 iterations 1500 iterations

Benchmarking of ADP

Table 2 Mean performance of policies in the various test cases over 100 trials, reported as a percent of the optimal policy. Shown on the far right is the average time in hours needed to compute value functions for each algorithm (or time allotted for policy search), but note that this is highly problem-dependent and can vary greatly between test cases. The optimal policy took an average of 11.3 hours to compute.

Case	1	2	3	4	5	6	7	8	9	10	11	12	Avg	CPU (hrs)
Lookup-.01	99.1	96.7	98.0	88.8	98.0	100.0	93.2	98.4	99.6	99.4	98.8	98.9	97.4	0.41
Lin-.01	96.2	96.4	96.9	91.8	88.2	95.2	95.4	98.9	98.9	98.2	98.3	99.5	96.2	1.62
Lookup-.10	100.1	99.5	99.3	97.1	99.7	100.2	97.2	99.4	100.0	100.1	99.6	99.8	99.3	0.67
Lin-.10	96.3	96.5	98.1	91.1	88.3	95.2	94.9	98.9	98.9	98.2	99.0	99.2	96.2	2.72
API	82.7	77.9	79.5	57.3	76.2	90.8	50.5	80.4	94.7	90.0	86.4	86.9	79.5	12.0
PFA	93.6	92.3	93.4	71.4	80.6	91.3	72.3	93.4	97.3	95.2	96.0	94.6	89.3	5.14
DLA-24-.05	87.7	Need	to	73.5	84.8	93.3	90.4	89.9	96.8	92.2	92.2	95.3	91.1	N/A
DLA-72-.01	91.1	Run	These	86.1	99.4	95.5	101.7	93.0	97.9	94.4	94.2	94.3	94.2	N/A

» Backward ADP is first four lines:

- Lookup = lookup table approximation for VFA
- Linear = linear model
- .01 or .10 refers to the discretization
- Backward MDP (which produced optimal policy) took over 10 hours to compute.

- Slides that follow illustrate exploiting concavity on a network of storage devices for the grid.
- We will revisit this method later in the course to show that we can use this method to solve high-dimensional problems.
- For now, we are just showing off.

- Imagine 25 large storage devices spread around the PJM grid:





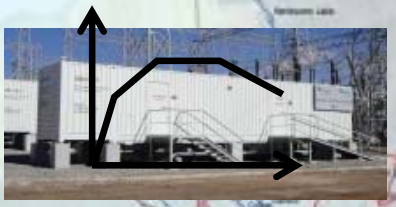


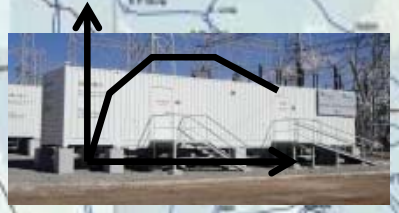


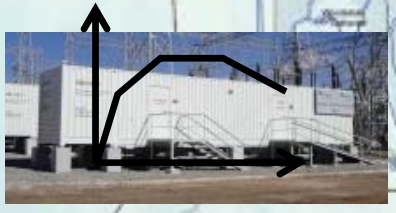
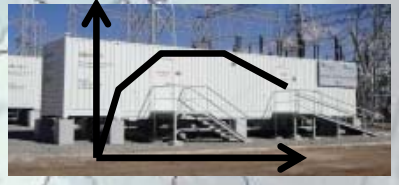



- Imagine 25 large storage devices spread around the PJM grid:

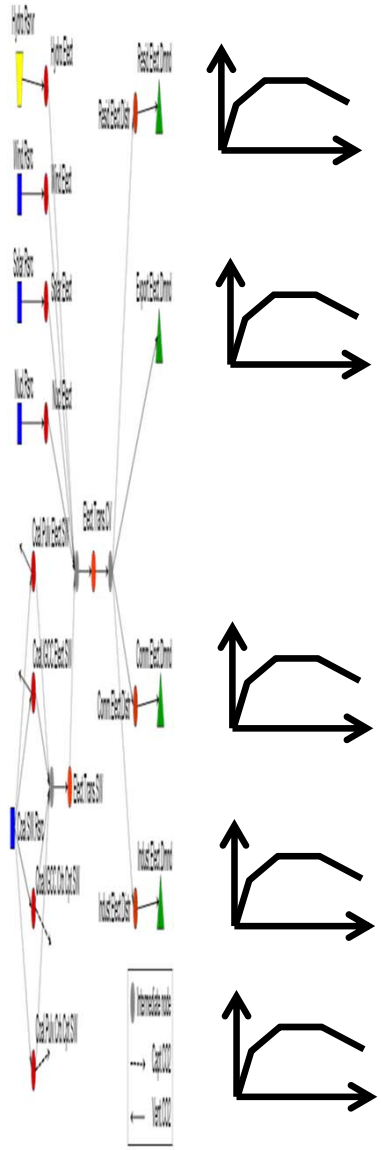




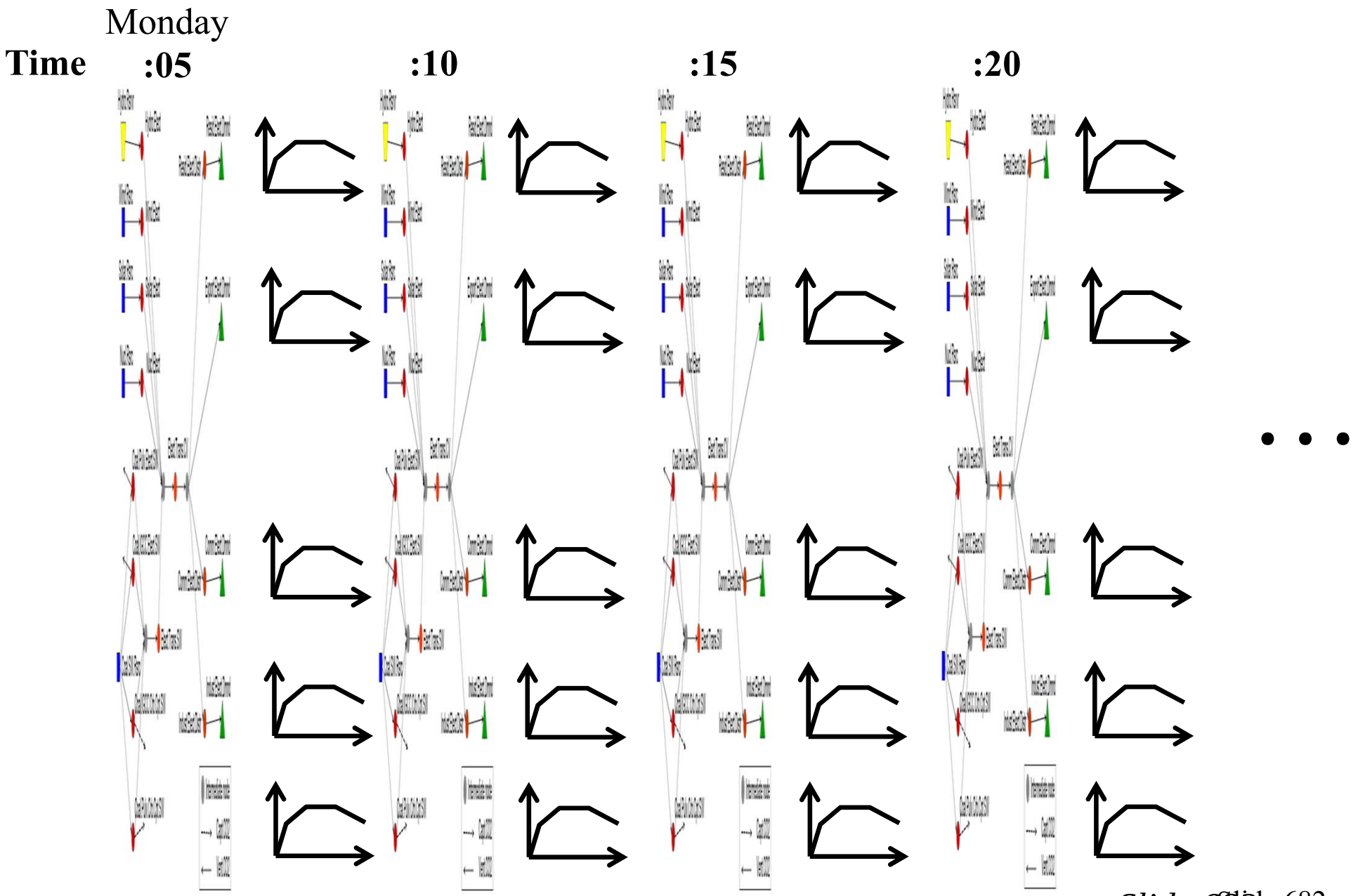



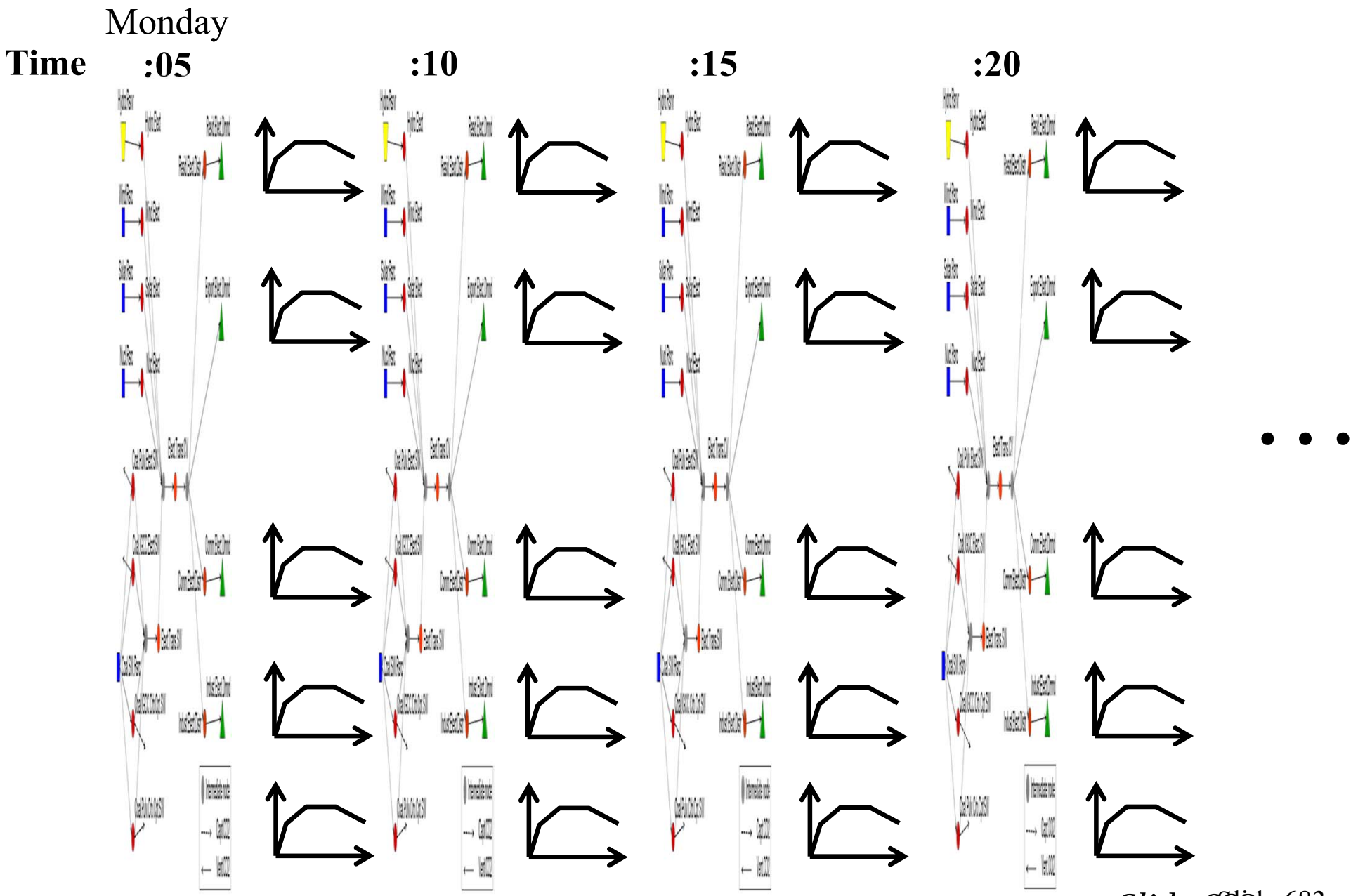

Value function approximations



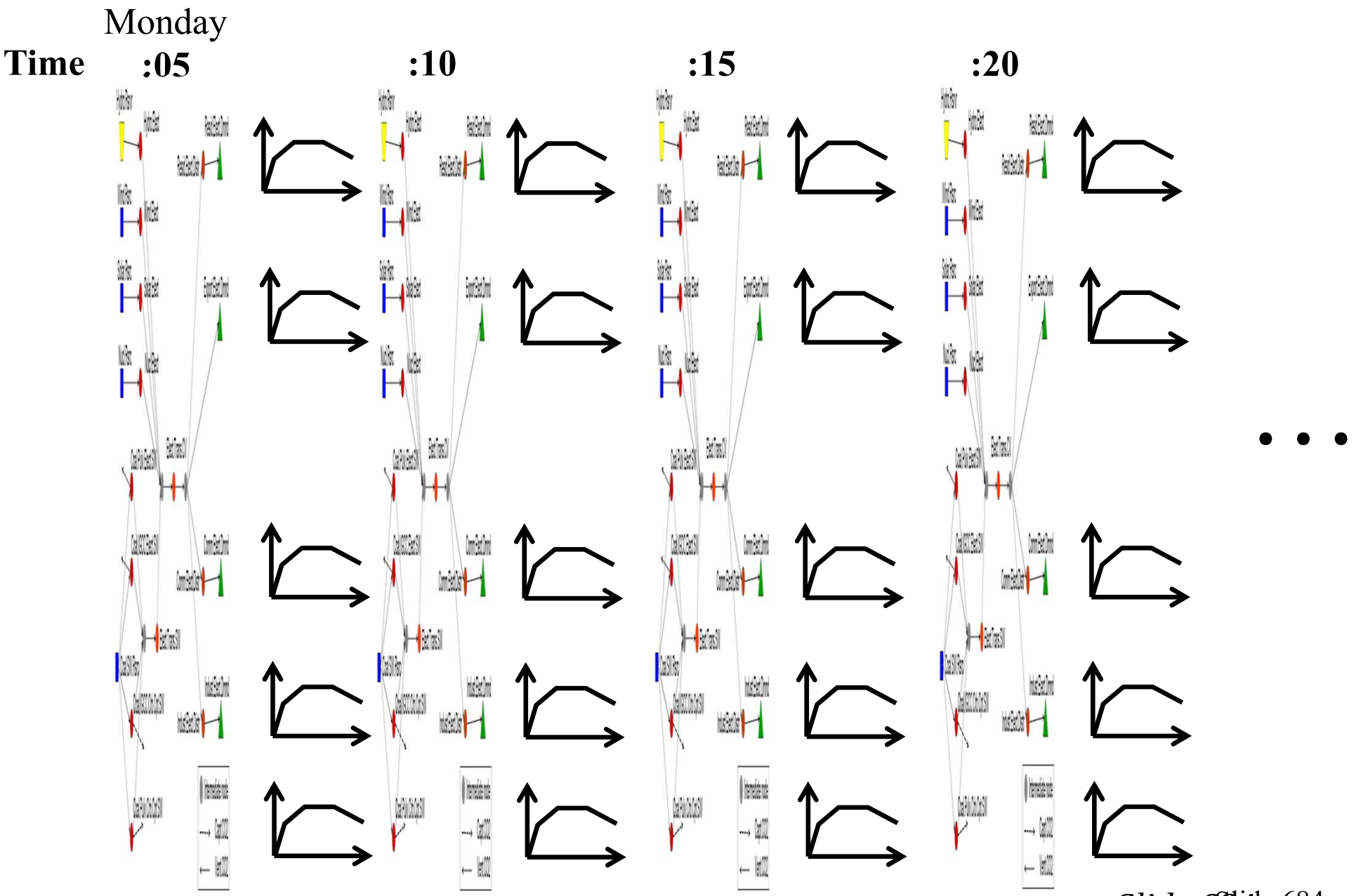
Value function approximations



Value function approximations

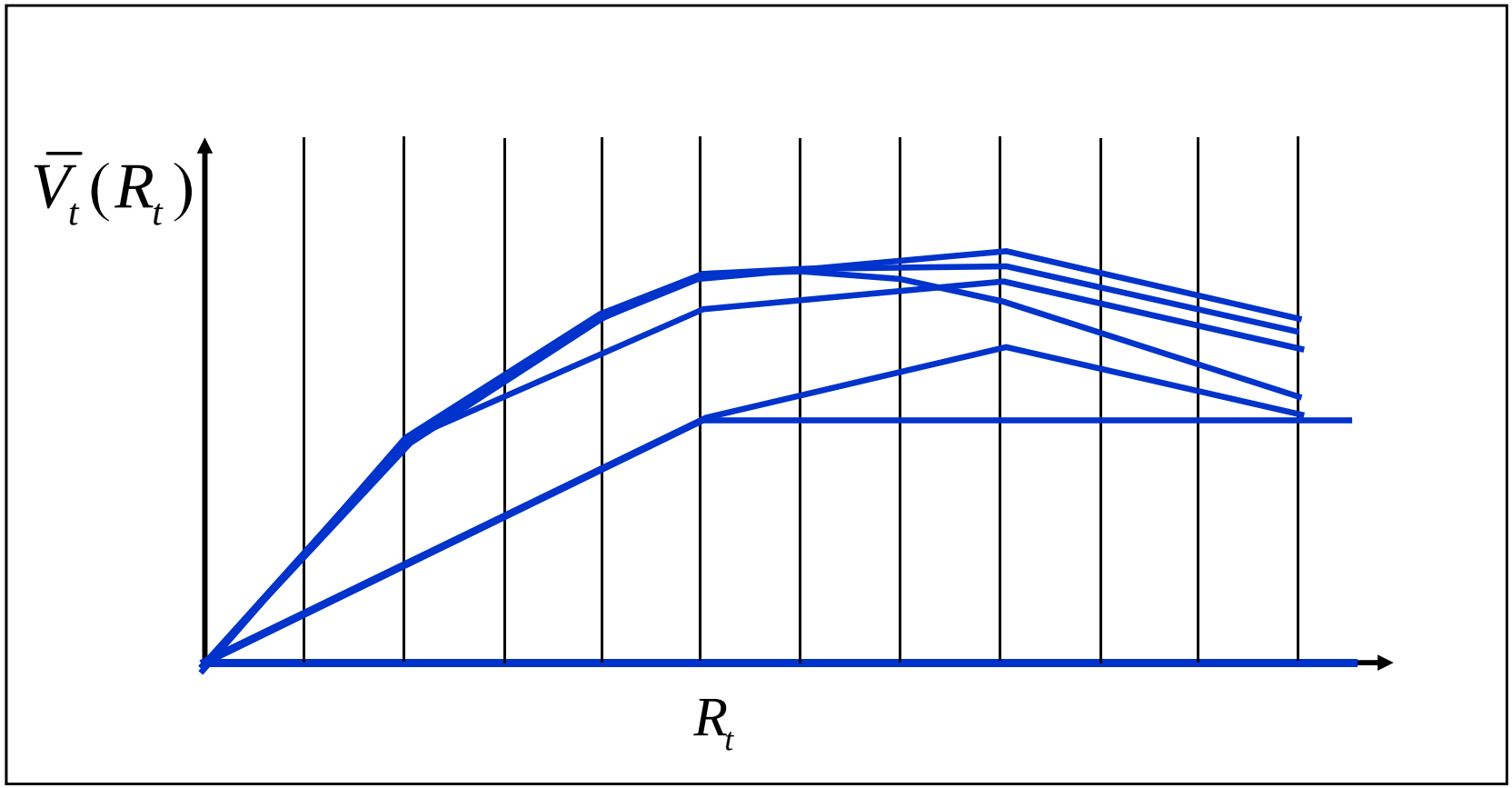


Value function approximations



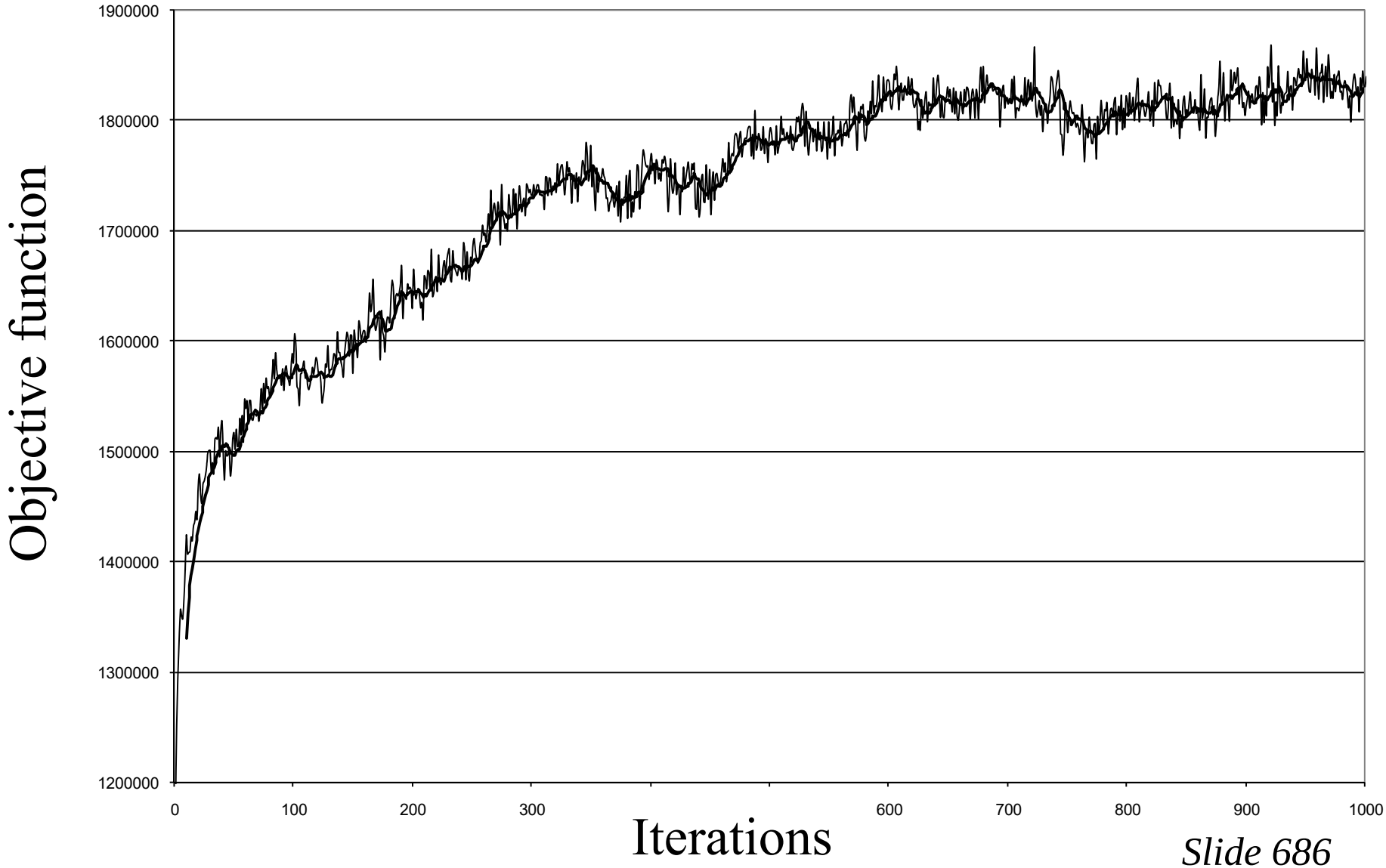
Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation



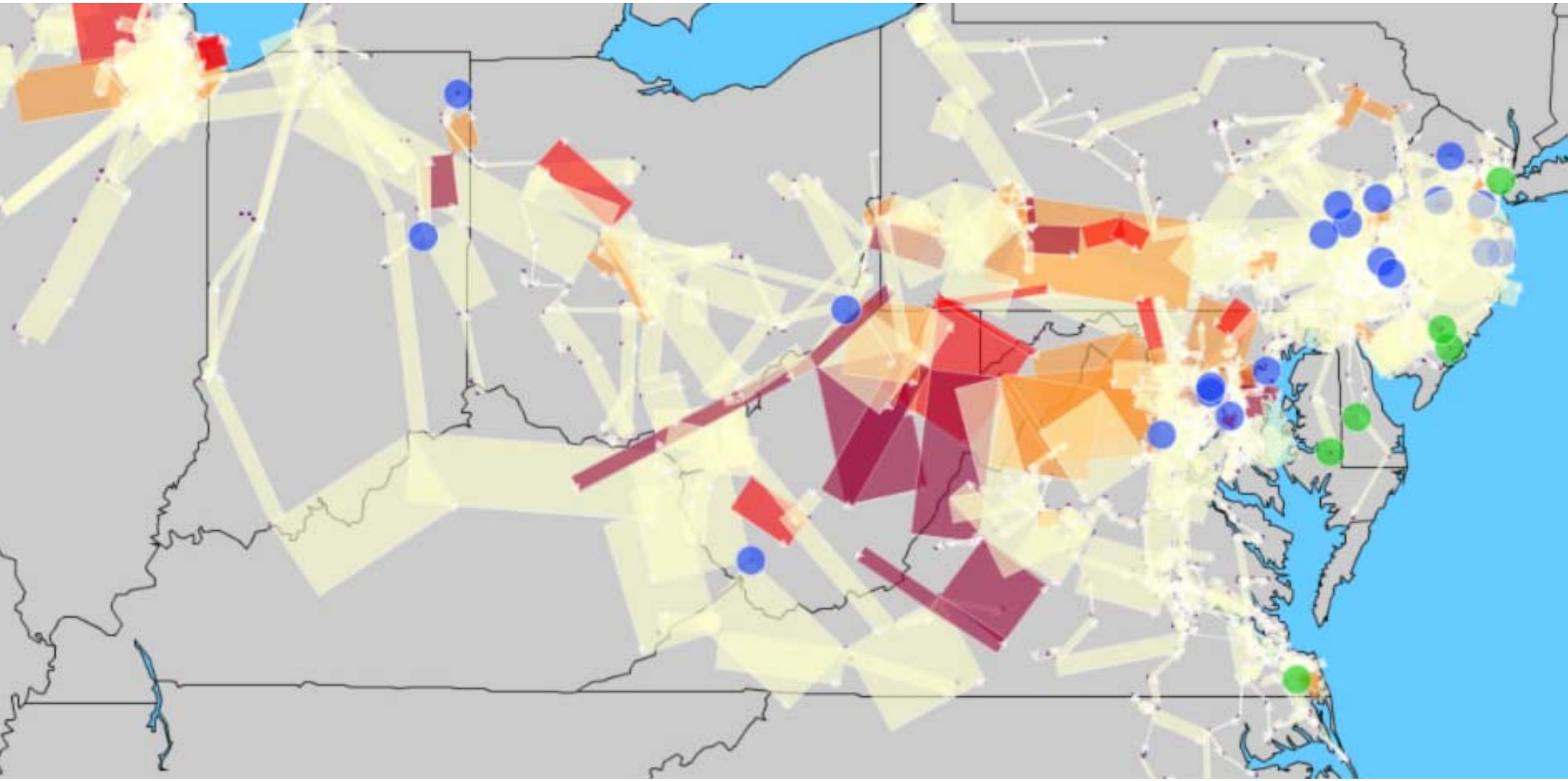
Approximate dynamic programming

- With luck, your objective function improves



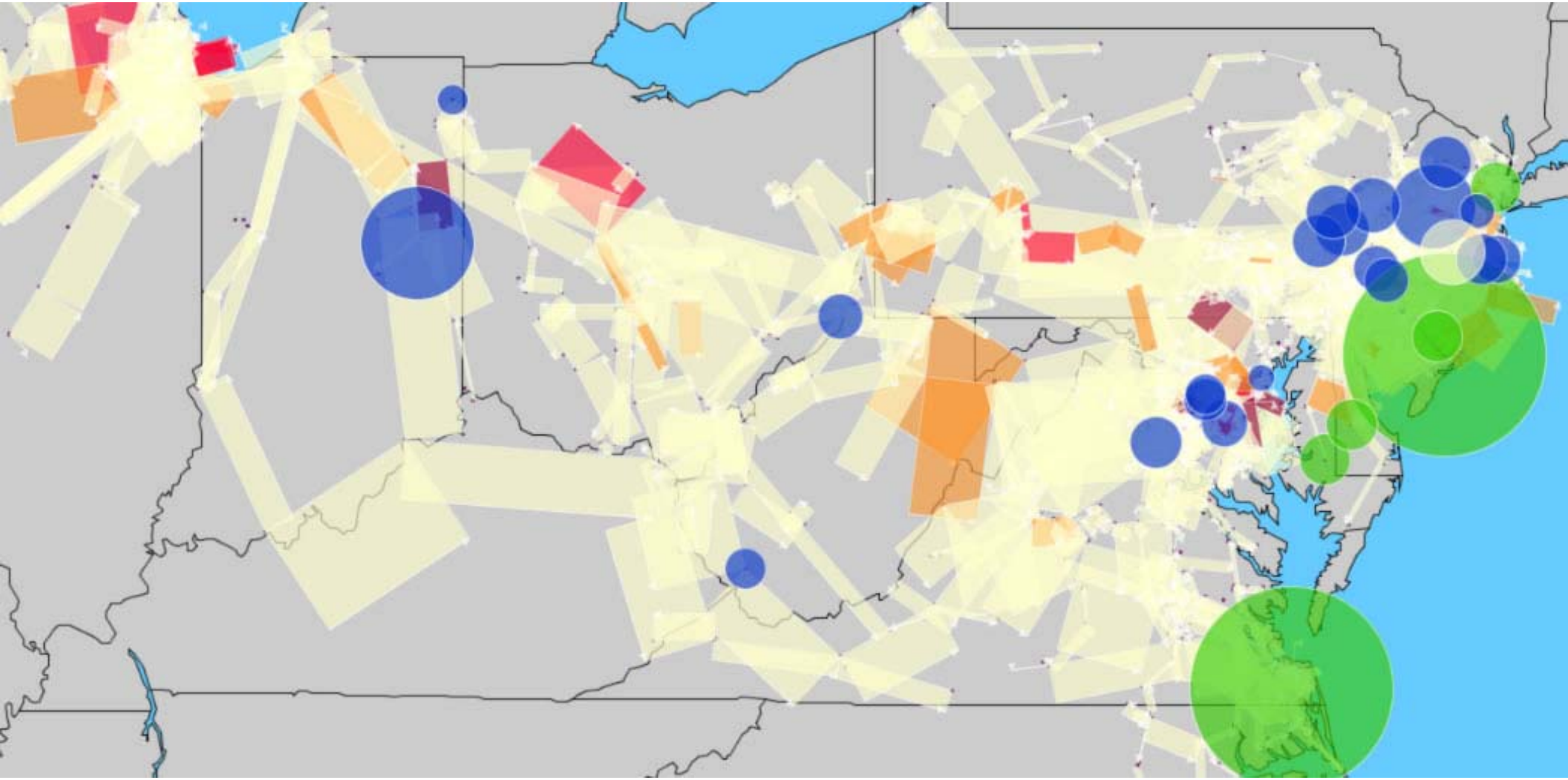
Grid level storage

- Congested grid:
  - » Green and blue circles indicate energy storage



SMART-Solar

- Congested grid:
  - » Green and blue circles indicate energy storage



- Notes:
  - » There are many "resource allocation problems" where the state variable is a quantity R_t giving the quantity of resources (energy in storage, water in a reservoir, money held in cash in a mutual fund).
  - » We can exploit convexity (concavity if maximizing) to get very accurate approximations of value functions.
  - » This logic scales to high dimensions, where $R_t = (R_{ti})$ where R_{ti} is the number of resources of "type" i, where i might be:
    - Location of resources (warehouse, battery, …)
    - Blood type
    - Type of asset money is invested in.

Designing policies

VFA-forward ADP

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:

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

to obtain x^n.

Deterministic optimization

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 $W_t(\omega^n)$ 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

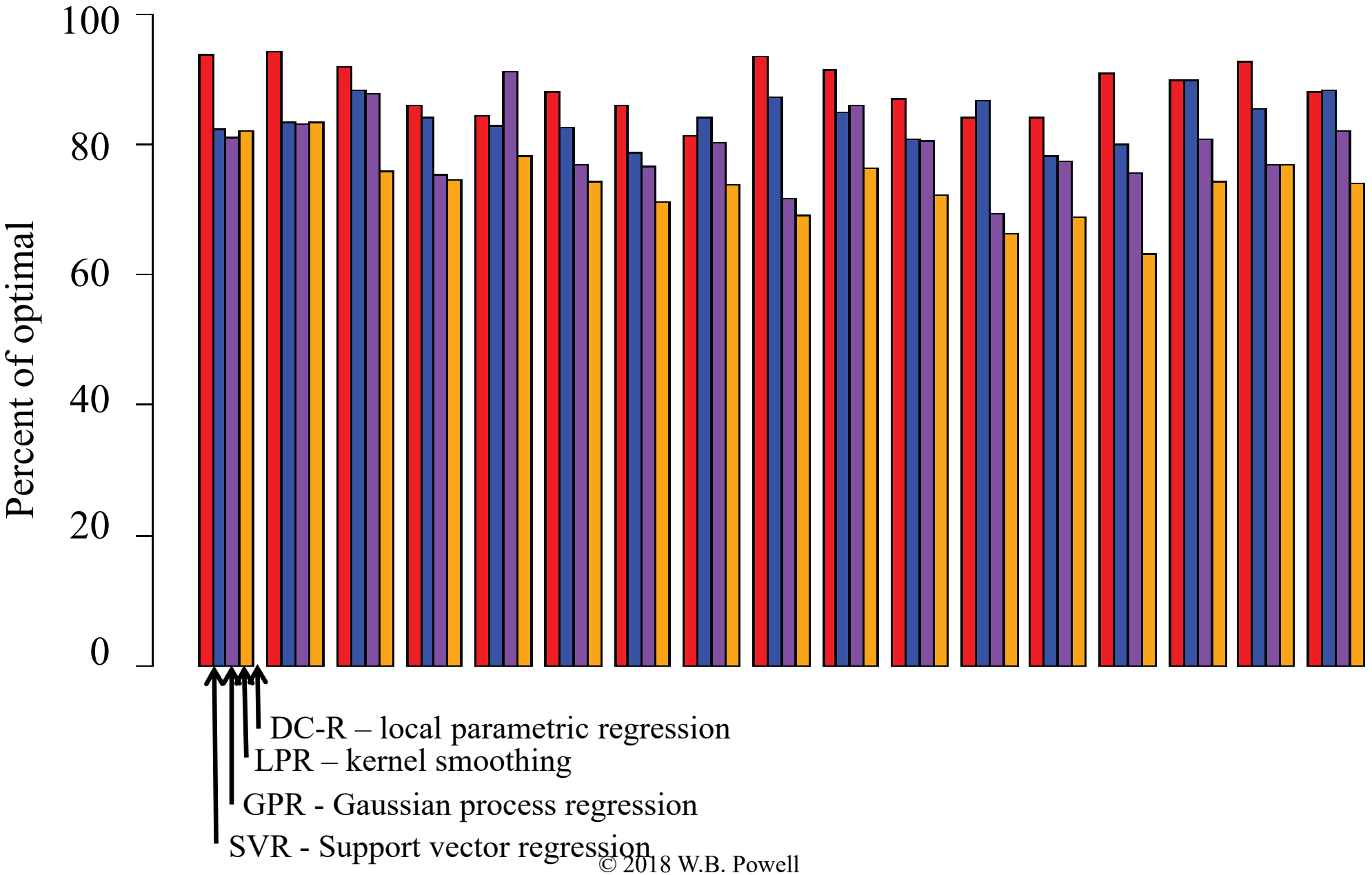
Step 5: Return to step 1.

Approximate policy iteration

- Forward ADP using general statistical models (ignores problem structure)
- Approximation methods
  - » **SVR -** Support vector regression with Gaussian radial basis kernel
  - » **LBF** – Weighted linear combination of polynomial basis functions
  - » **GPR** – Gaussian process regression with Gaussian RBF
  - » **LPR** – Kernel smoothing with second-order local polynomial fit
  - » **DC-R** – Dirichlet clouds – Local parametric regression.
  - » **TRE** – Regression trees with constant local fit.

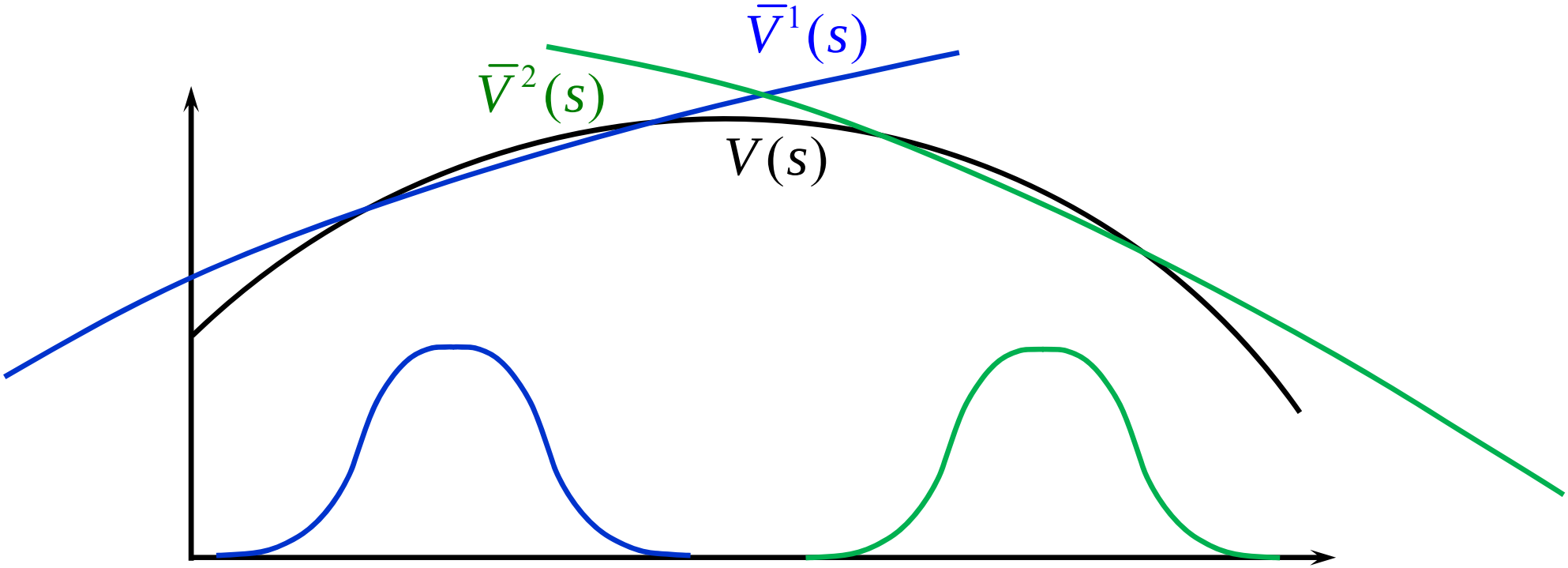
- Notes:
  - » None of these statistical models explicitly captures structure such as convexity/concavity.
  - » We ran thousands of iterations to test all of these machine learning methods to approximate the value function.
  - » The results were then compared against an optimal policy using value functions computed using classical MDP.

Energy storage applications



Approximate policy iteration

- A tale of two distributions
  - » The *sampling distribution*, which governs the likelihood that we sample a state.
  - » The *learning distribution*, which is the distribution of states we would visit given the current policy



Forward ADP

- The story changes when we exploit structure…
  - » The value function is concave in the resource variable.
  - » But we have no structural properties with respect to the other dimensions of the state variable.

Designing policies

Direct lookahead

Parametric cost function approximation

- An energy storage problem:

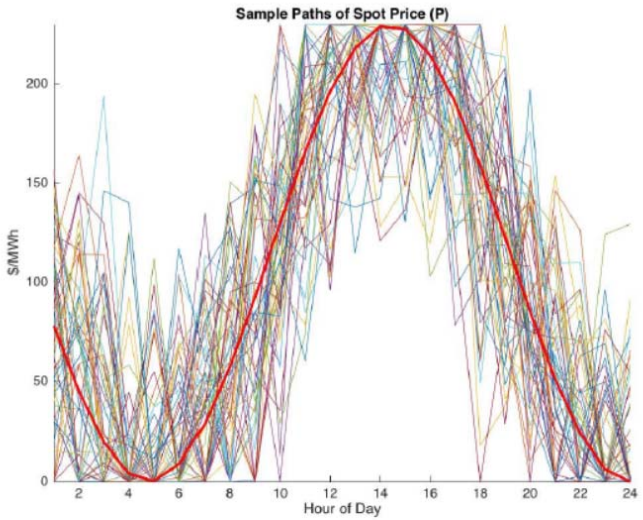


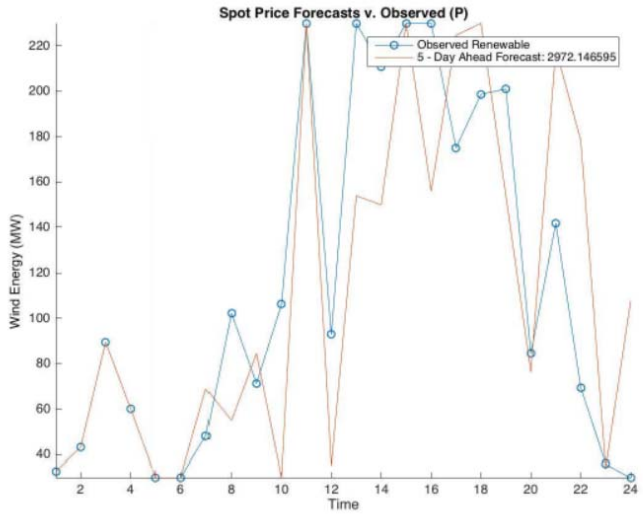
Figure: Sample paths of spot prices (P_t)

The state of our system is given by:

$$S_t = (R_t, E_t, P_t, D_t, f_t^E)$$

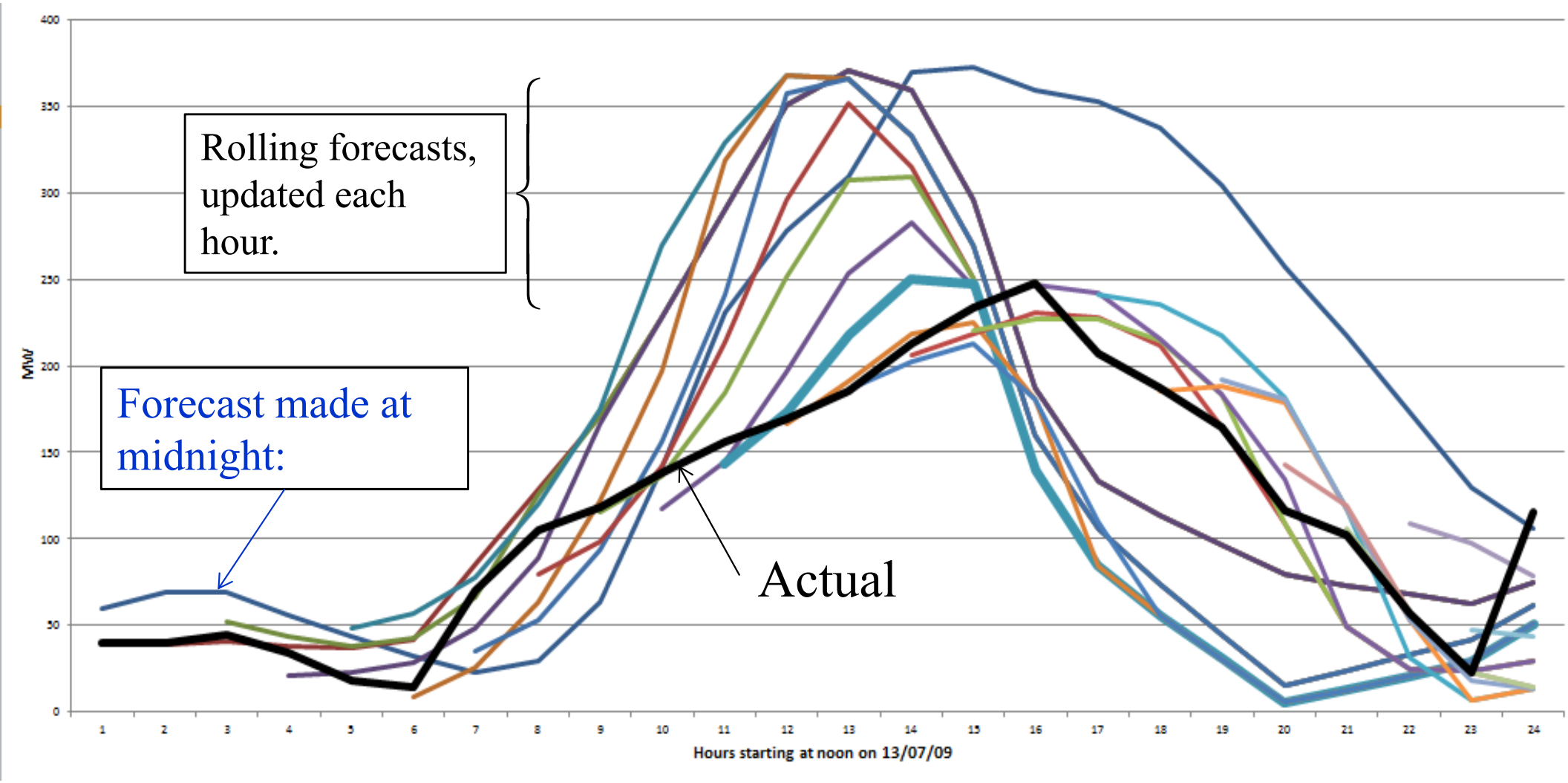
where

R_t =Energy in storage
E_t^W =Energy available from wind
P_t =Grid price
D_t =Demand
$f_t^E = \left(f_{tt'}^E\right), t' \geq t,$ = Forecasts



Parametric cost function approximation

- Forecasts evolve over time as new information arrives:



Parametric cost function approximation

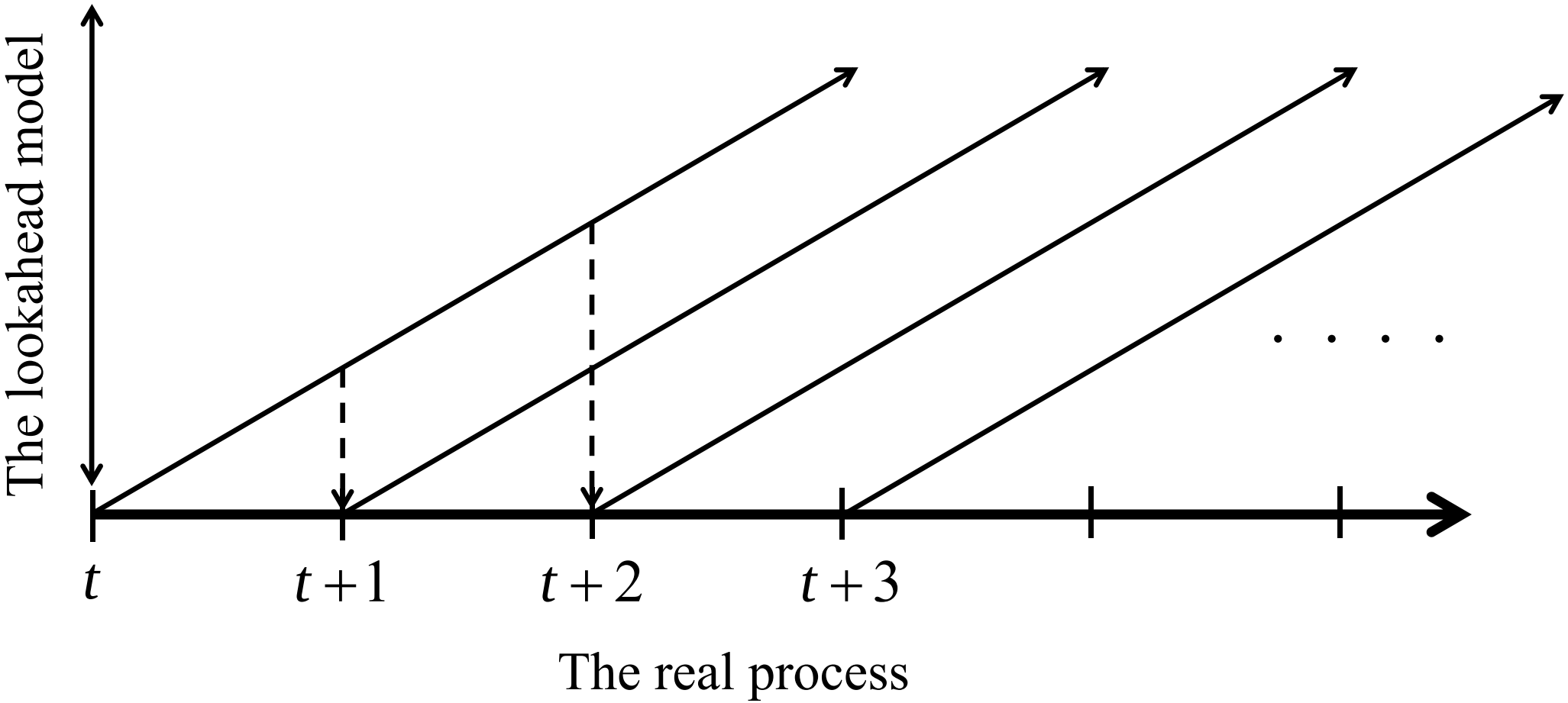
- Benchmark policy – Deterministic lookahead

$$X_t^{\text{D-LA}}(S_t) = \underset{x_t,(\tilde{x}_{tt'},t'=t+1,\ldots,t+H)}{\text{argmin}} \left(C(S_t,x_t) + \left[\sum_{t'=t+1}^{t+H} \tilde{c}_{tt'}\tilde{x}_{tt'} \right] \right)$$

$$\begin{aligned}
\tilde{x}_{tt'}^{wd} + \beta\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{gd} &\le f_{tt'}^{D} \\
\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{rg} &\le \tilde{R}_{tt'} \\
\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{gr} &\le R^{\max} - \tilde{R}_{tt'} \\
\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{wd} &\le f_{tt'}^{E} \\
\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{gr} &\le \gamma^{\text{charge}} \\
\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{rg} &\le \gamma^{\text{discharge}}
\end{aligned}$$

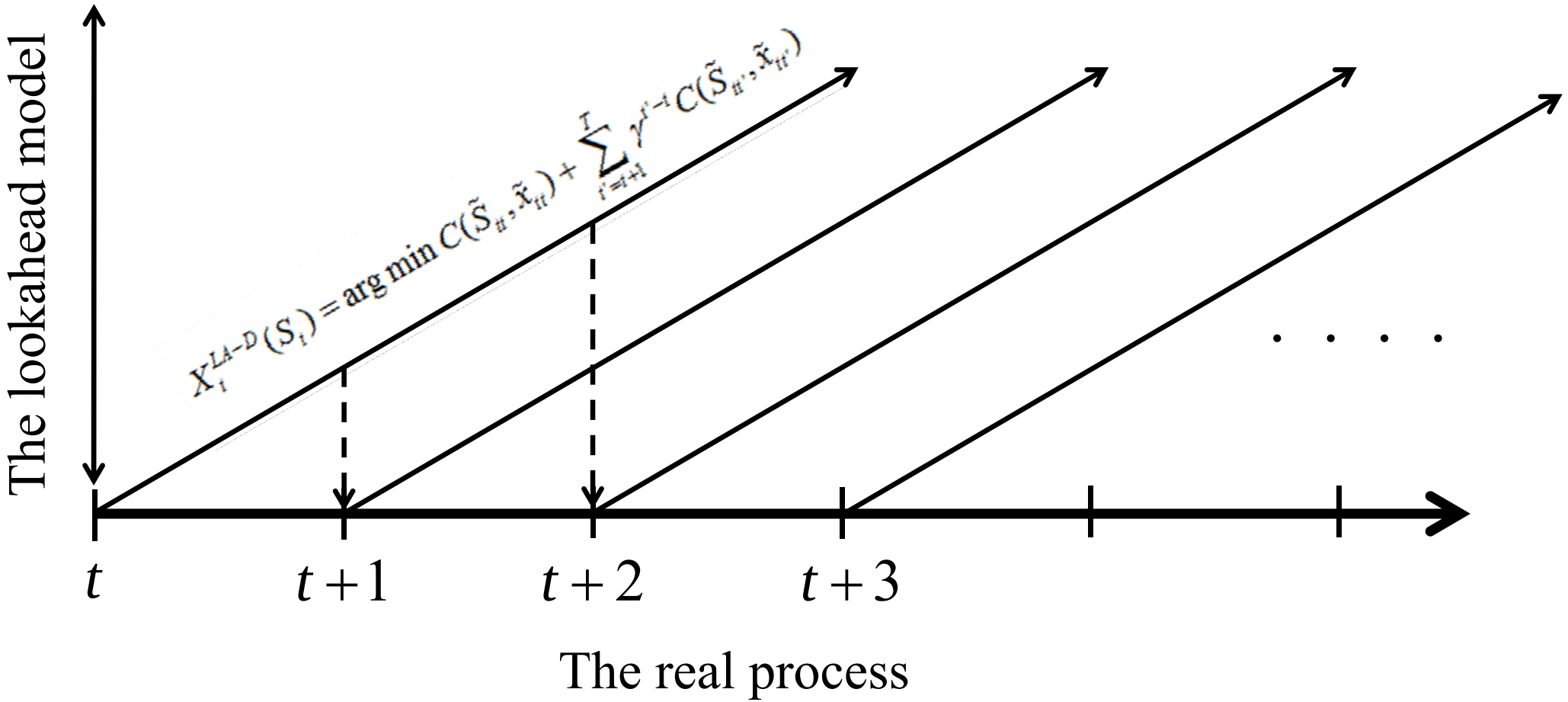
Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



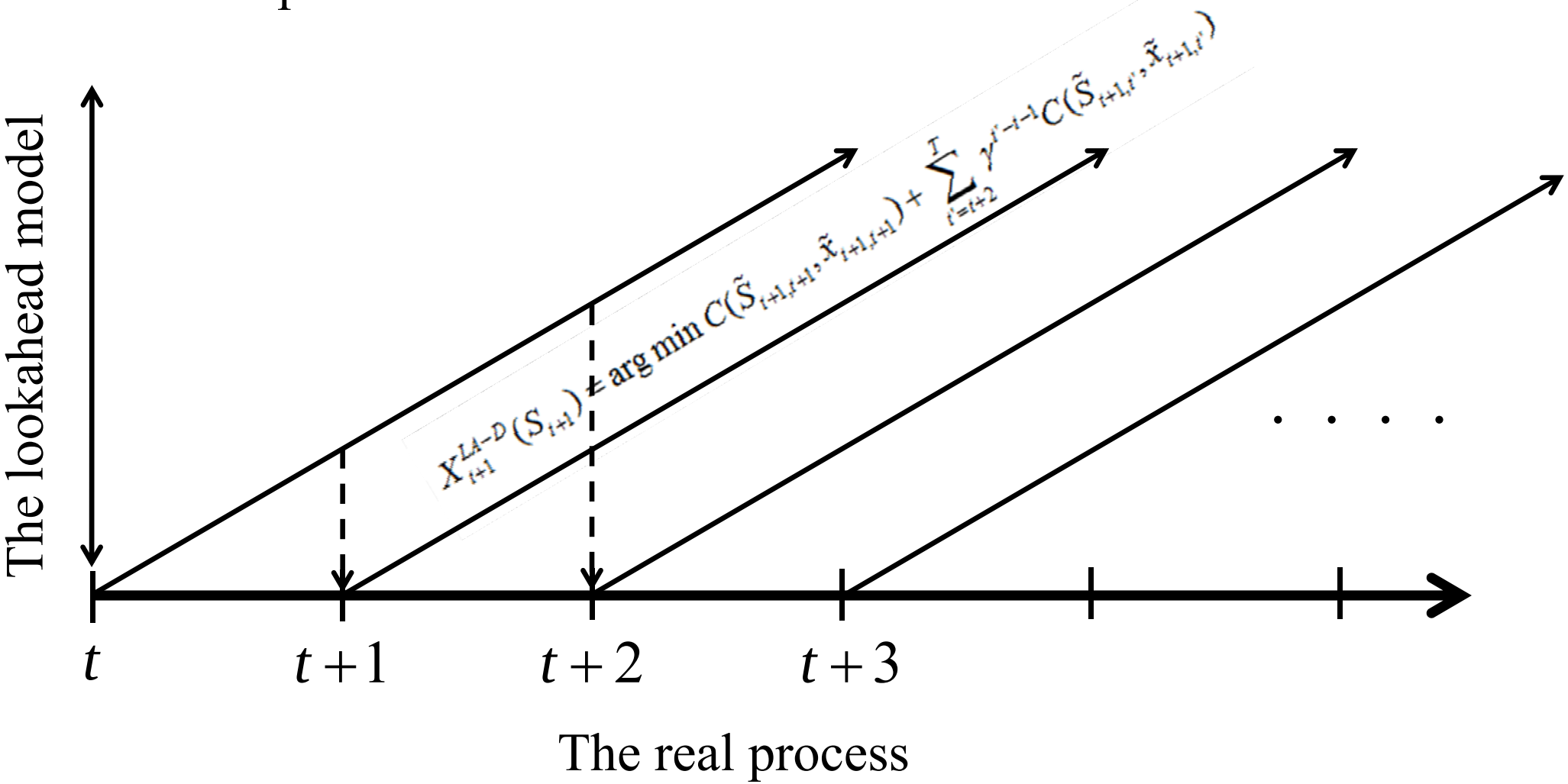
Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



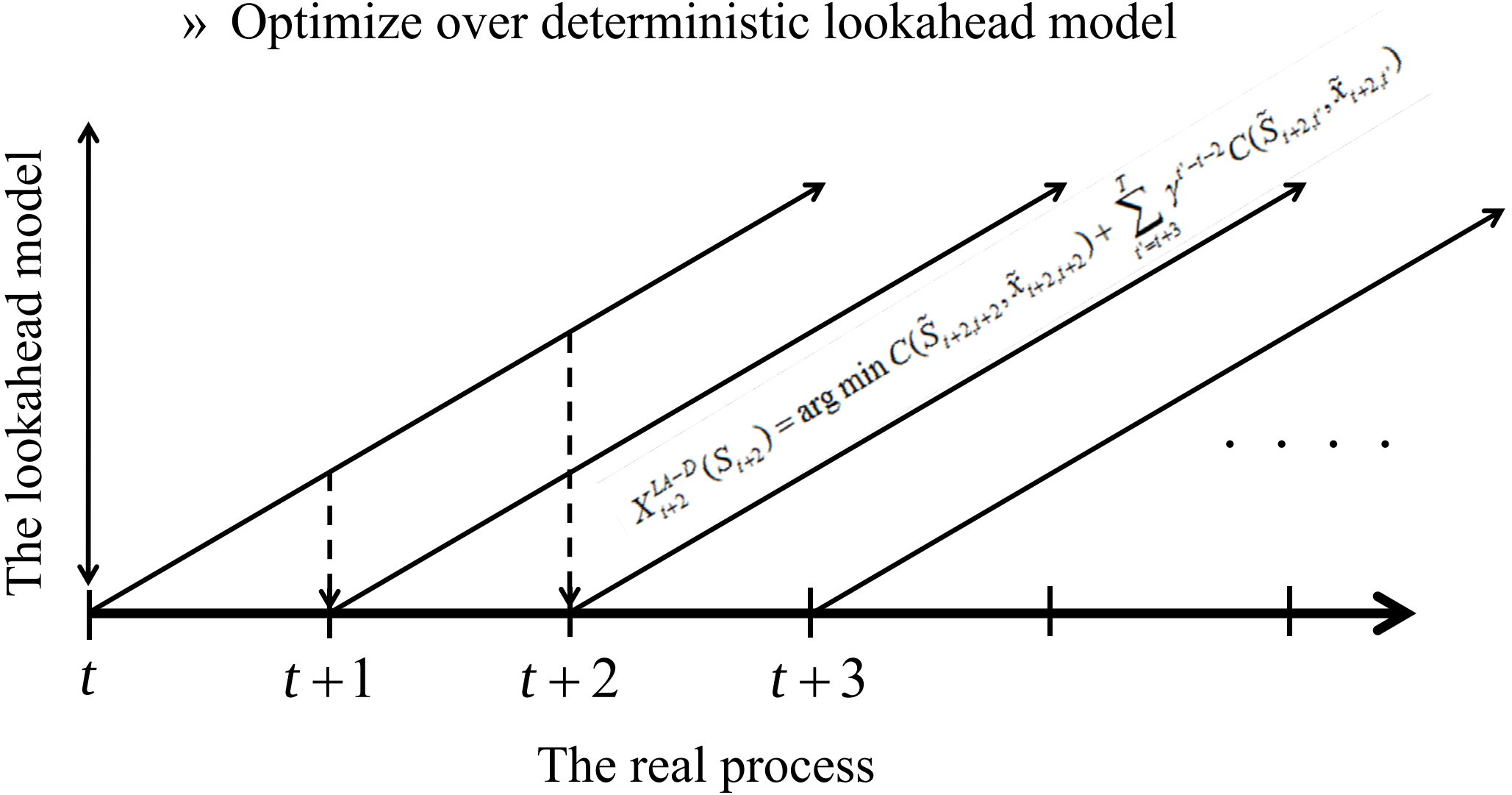
Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



Lookahead policies

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



Hybrid policies

- Simulating the policy:
  - » Let $\bar{F}(\omega)$ be a simulation of our policy following sample path ω:

$$\bar{F}(\omega) = \sum_{t=0}^{T} C\left(S_t(\omega), X_t^{\pi}(S_t(\omega))\right)$$

where

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

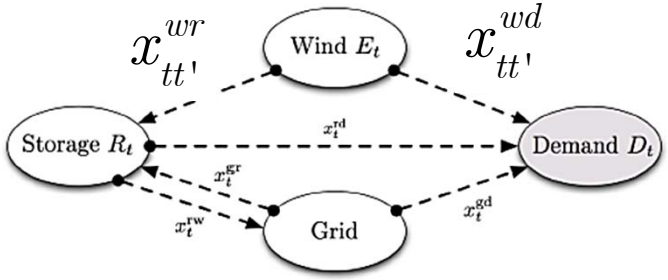
Designing policies

Hybrid – parameterized lookahead (CFA/DLA)

Parametric cost function approximation

- Parametric cost function approximations
  - » Replace the constraint

$$\tilde{x}^{wr}_{tt'} + \tilde{x}^{wd}_{tt'} \leq f^{E}_{tt'}$$



with:

  - » Lookup table modified forecasts (one adjustment term for each time $\tau = t' - t$ in the future):

$$x^{wr}_{tt'} + x^{wd}_{tt'} \leq \theta_{t'-t} f^{E}_{tt'}$$

  - » Exponential function for adjustments (just two parameters)

$$x^{wr}_{tt'} + x^{wd}_{tt'} \leq \theta_1 e^{\theta_2 (t'-t)} f^{E}_{tt'}$$

  - » Constant adjustment (one parameter)

$$x^{wr}_{tt'} + x^{wd}_{tt'} \leq \theta f^{E}_{tt'}$$

Hybrid policies

- Optimizing the CFA:
  - » Let $\bar{F}(\theta,\omega)$ be a simulation of our policy given by

$$\bar{F}(\theta,\omega)=\sum_{t=0}^{T}C\left(S_t(\omega),X_t^{\pi}(S_t(\omega)\mid\theta)\right)$$

  - » We then compute the gradient with respect to θ

$$\nabla_\theta F(\theta,\omega)=\nabla_\theta \bar{F}(\theta,\omega)$$

  - » The parameter θ is found using a classical stochastic gradient algorithm:

$$\theta^{n+1}=\theta^n+\alpha_n\nabla_\theta F(\theta^n,\omega^{n+1})$$

We tested several stepsize formulas and found that ADAGRAD worked best:

$$\alpha_n=\frac{\eta}{\sqrt{G_t+\varepsilon}}\quad G_t=\sum_{t'=0}^{t}\left(\nabla_x F(x_t,W_{t+1})^2\right)$$

Simulating the numerical derivative

Let ω be a sample path of $W_1(\omega), W_2(\omega), \ldots$ with a sequence of states $S_{t+1}(\omega) = S^M(S_t(\omega), X_t^\pi(S_t(\omega)\,|\,\theta), W_{t+1}(\omega))$.

Let

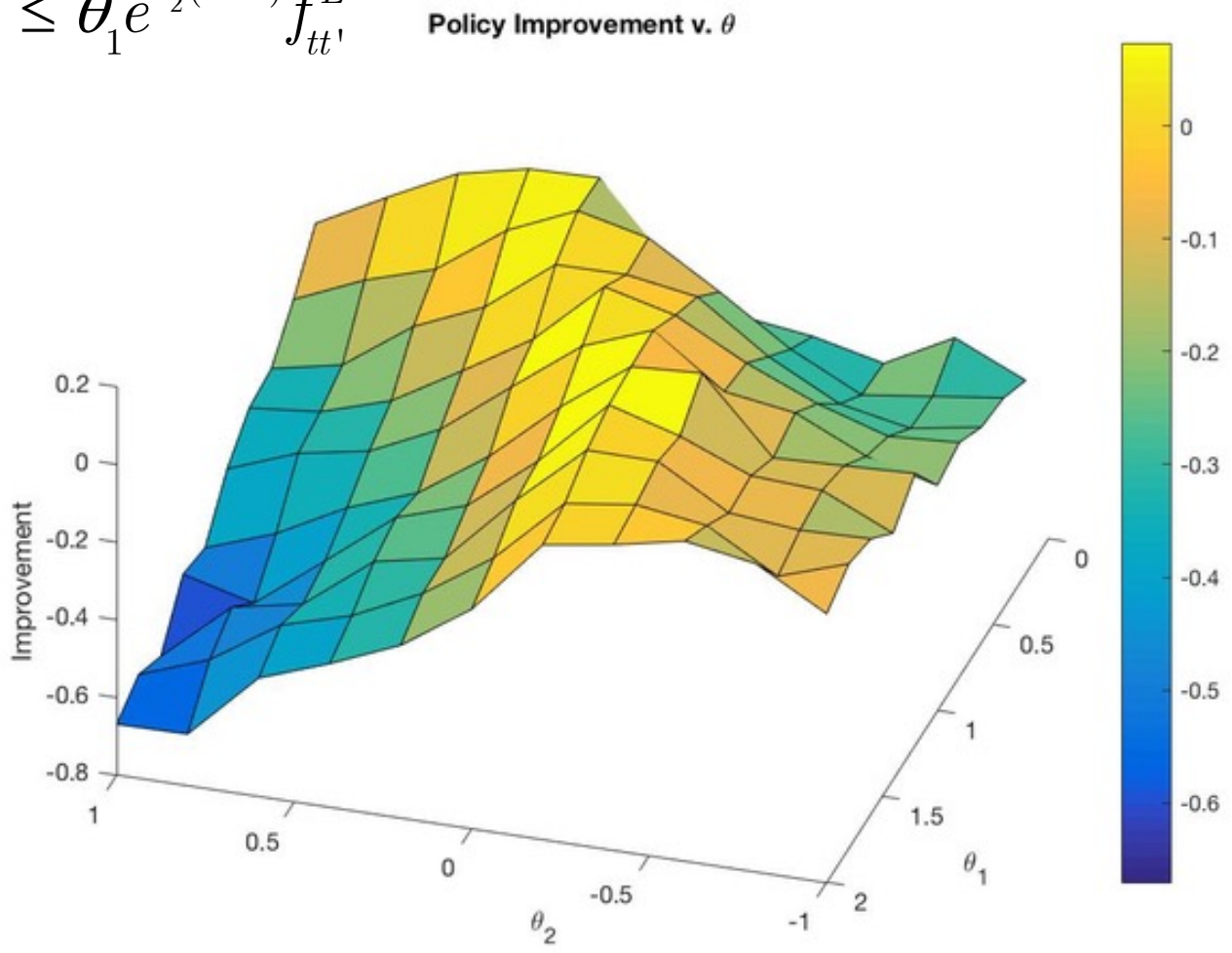
$$F^\pi(\theta,\omega) = \sum_{t=0}^{T} C\left(S_t(\omega), X_t^\pi(S_t(\omega)\,|\,\theta)\right)$$

Now compute the stochastic numerical derivative using

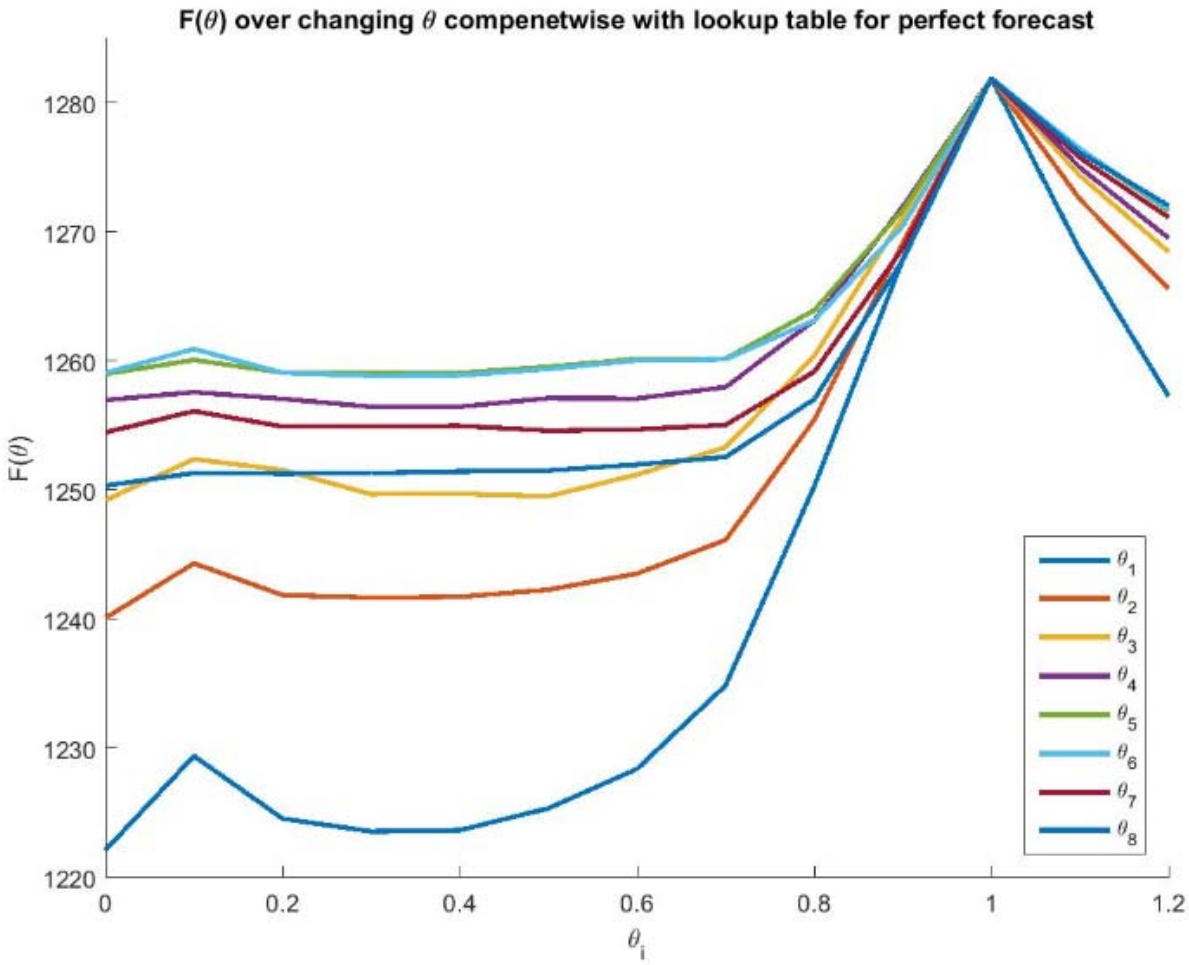
$$g(\omega) = \frac{F^\pi(\theta+\delta,\omega) - F^\pi(\theta,\omega)}{\delta}$$

We are writing this assuming that $F^\pi(\theta+\delta,\omega)$ and $F^\pi(\theta,\omega)$ are both computed with the same sample path ω. This is best, but not required, and not always possible.

$$x_{tt'}^{wr} + x_{tt'}^{wd} \leq \theta_1 e^{\theta_2 (t'-t)} f_{tt'}^{E}$$

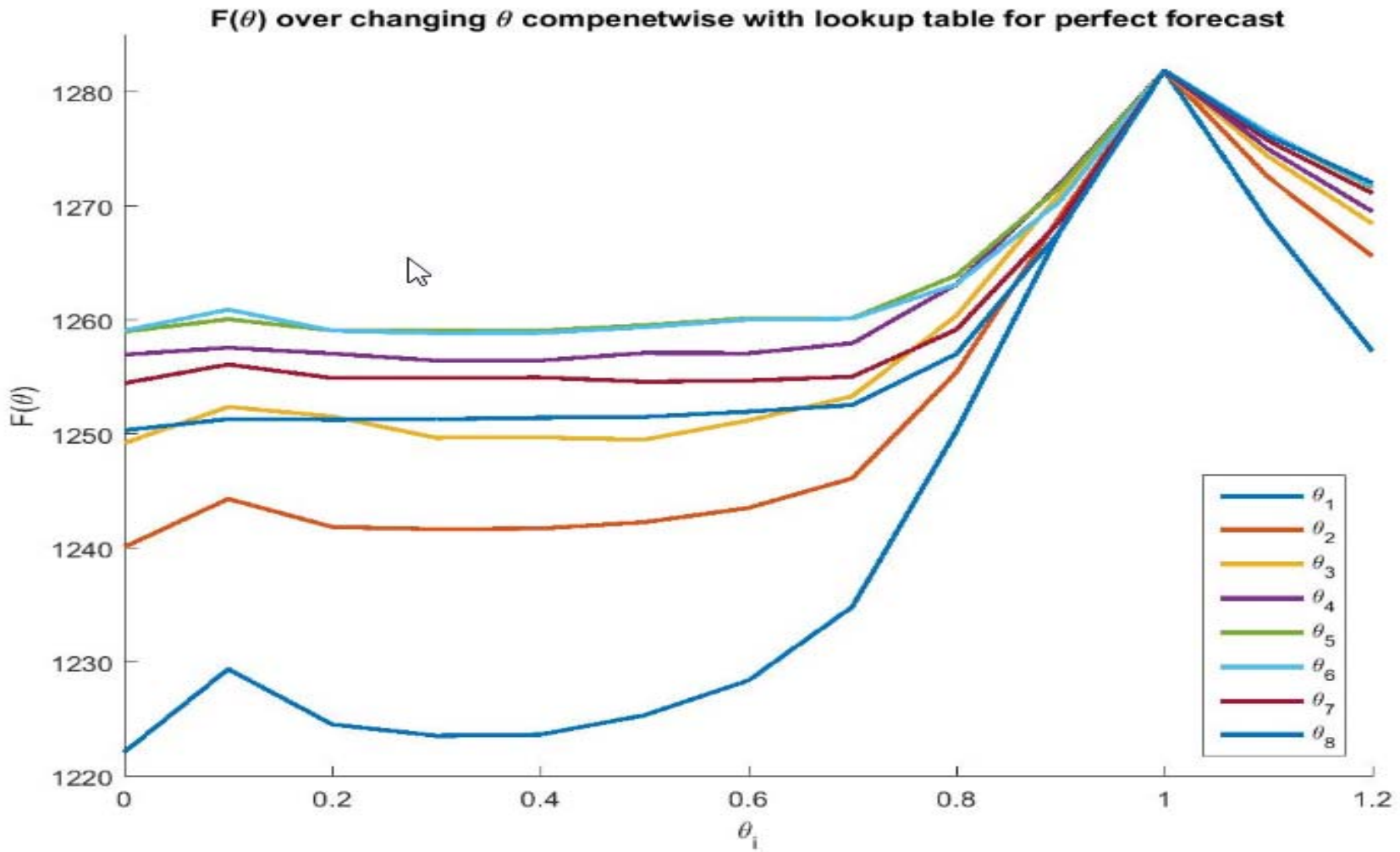


- Perfect forecasts:
  - » When the forecast is perfect, the optimal coefficients $\theta = 1$.
  - » Below we fix all $\theta s = 1$, but then vary each θ_i one at a time.

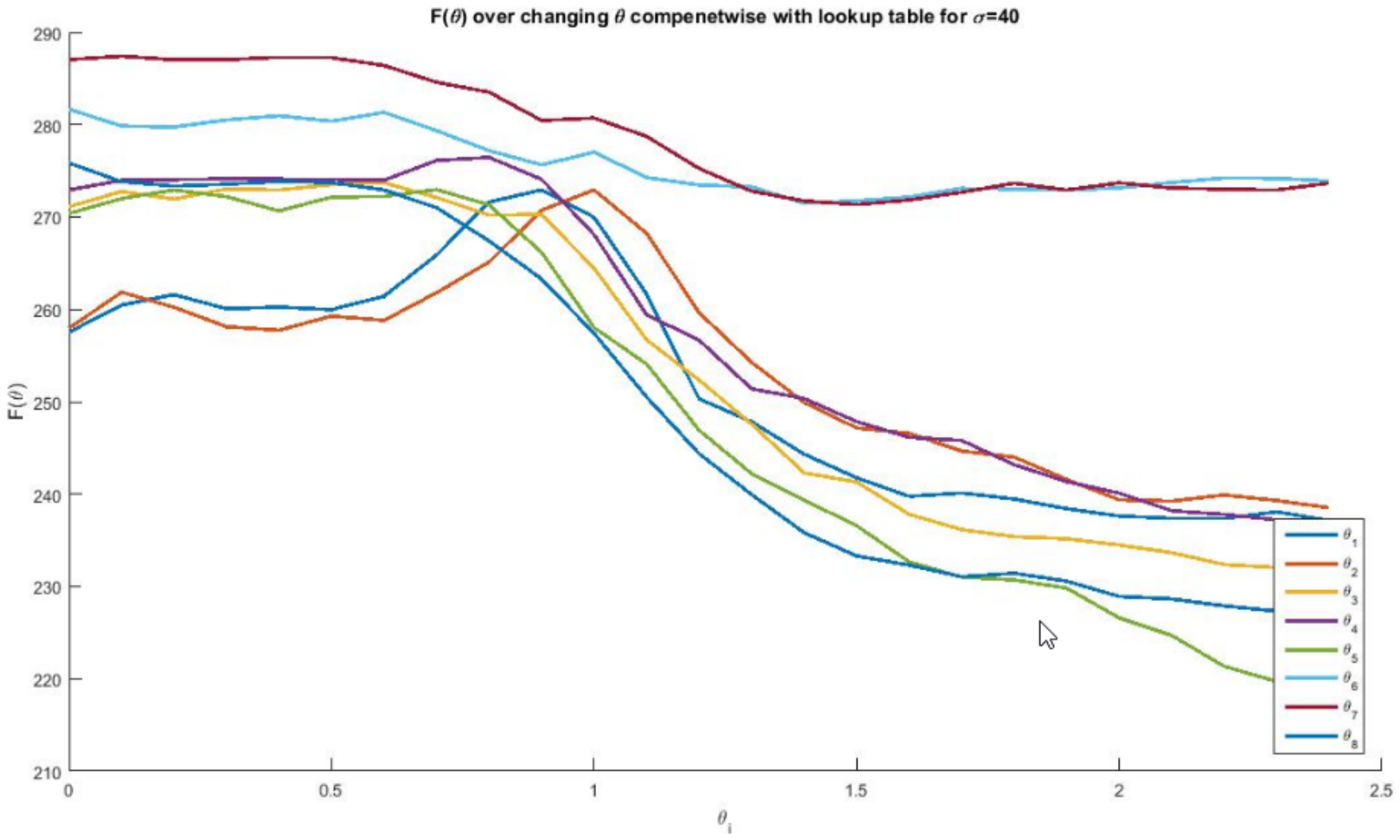


- Notes
  - » In the slides that follow, we optimized the θs, fixed each θ_i to the optimized point we found, and then varied each θ_i, one at a time.
  - » The results were somewhat unexpected…

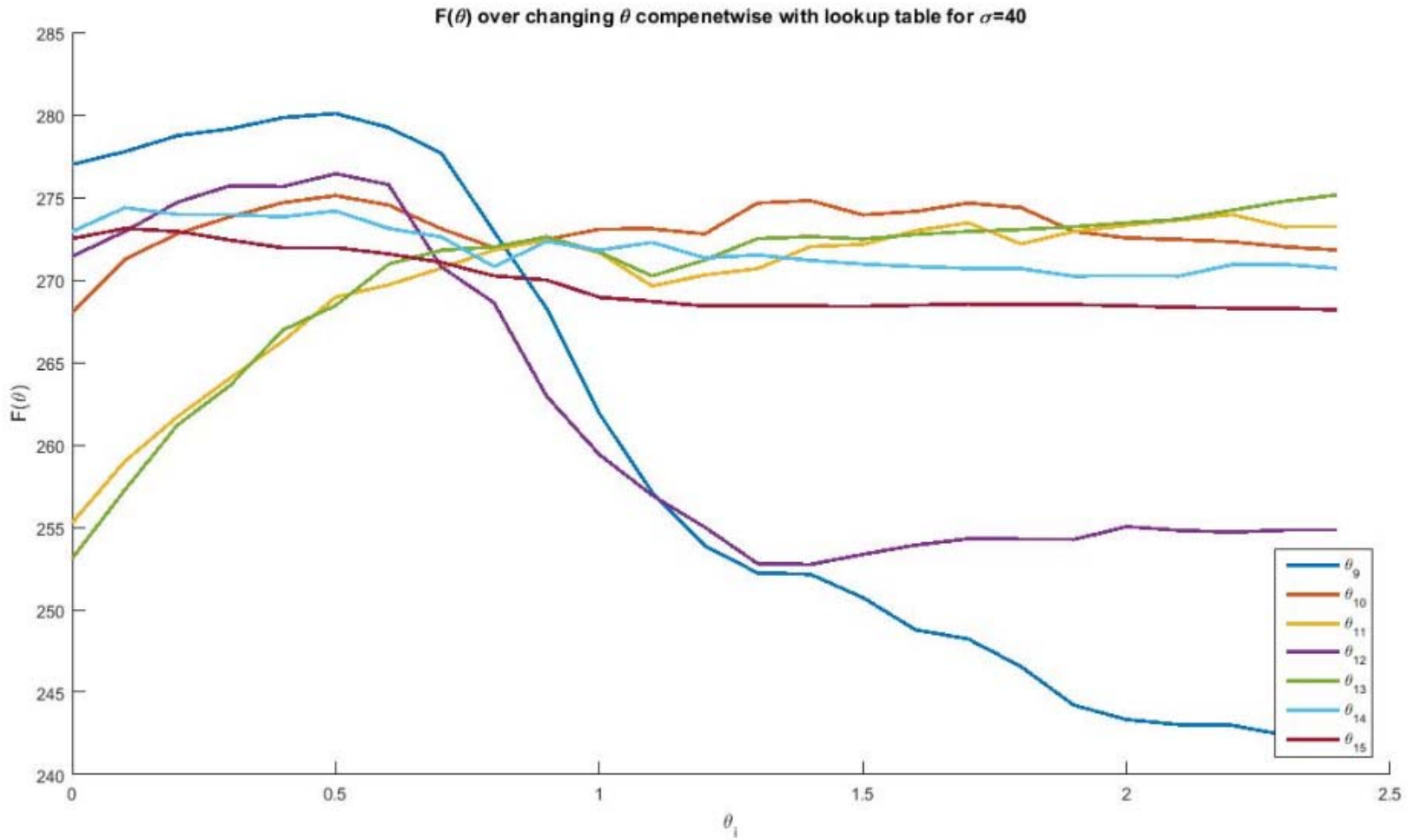
- One-dimensional contour plots – perfect forecast
  - » θ_i for i=1,…, 8 hours into the future.

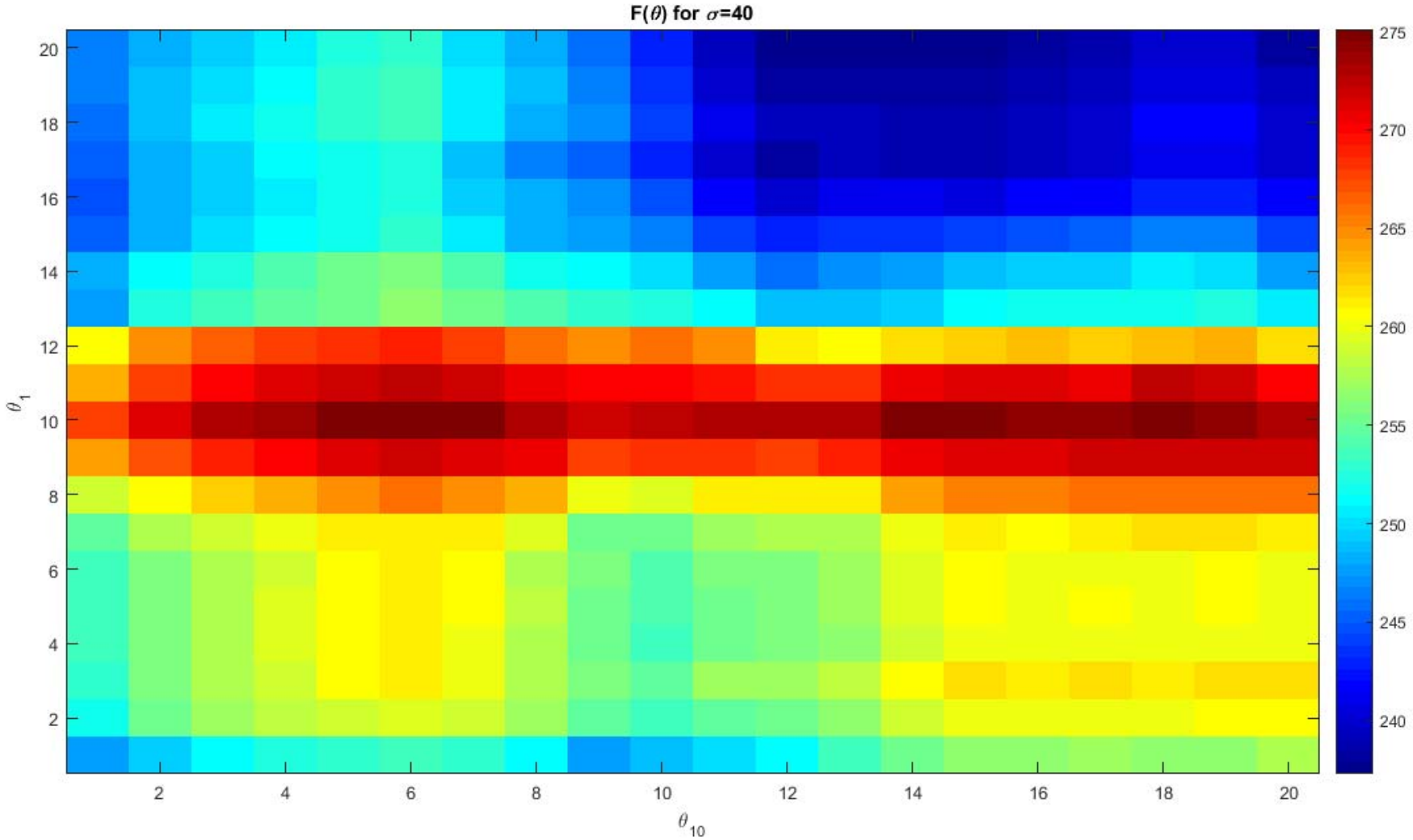


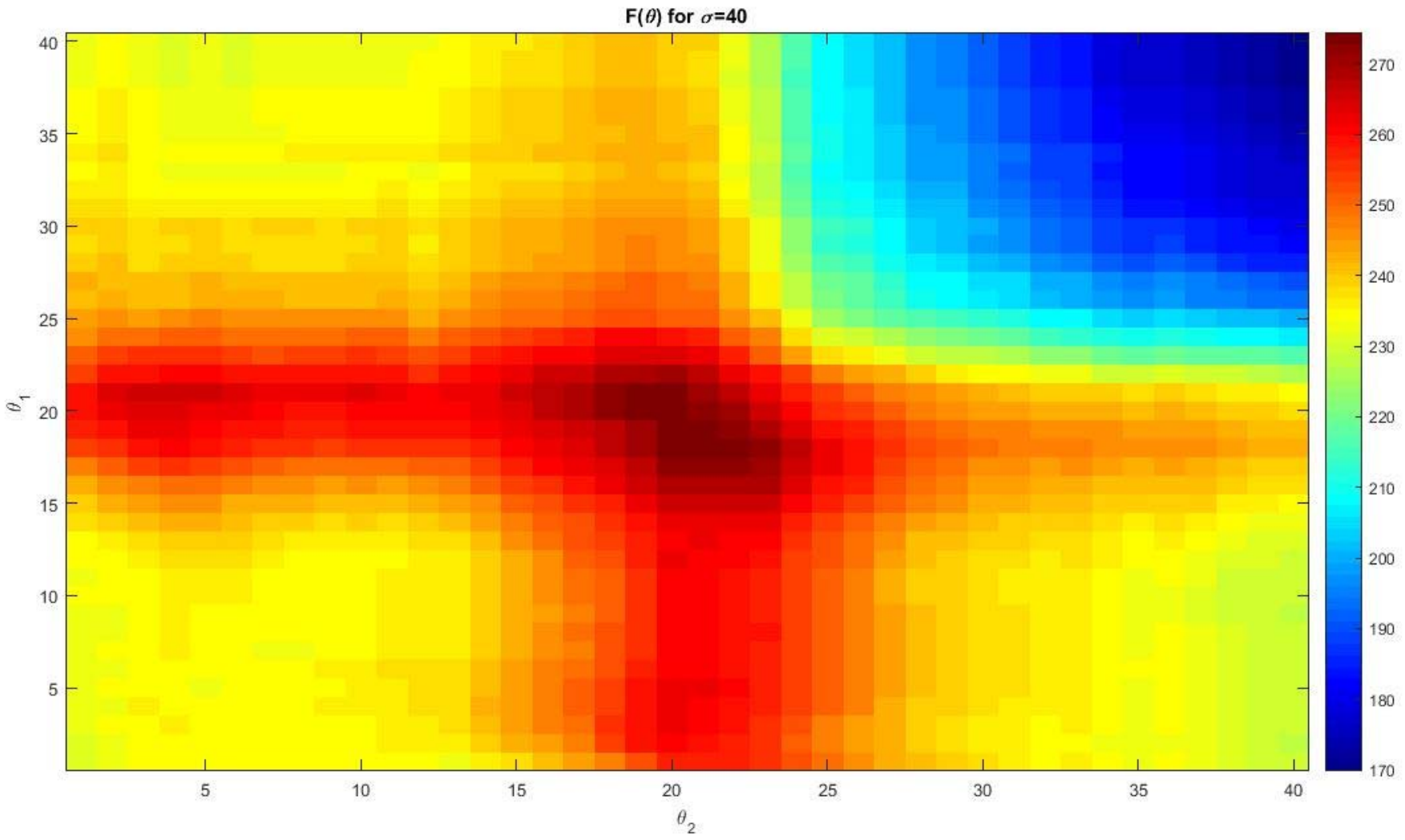
- One-dimensional contour plots-uncertain forecast
  - » θ_i for i=1,…, 8 hours into the future.

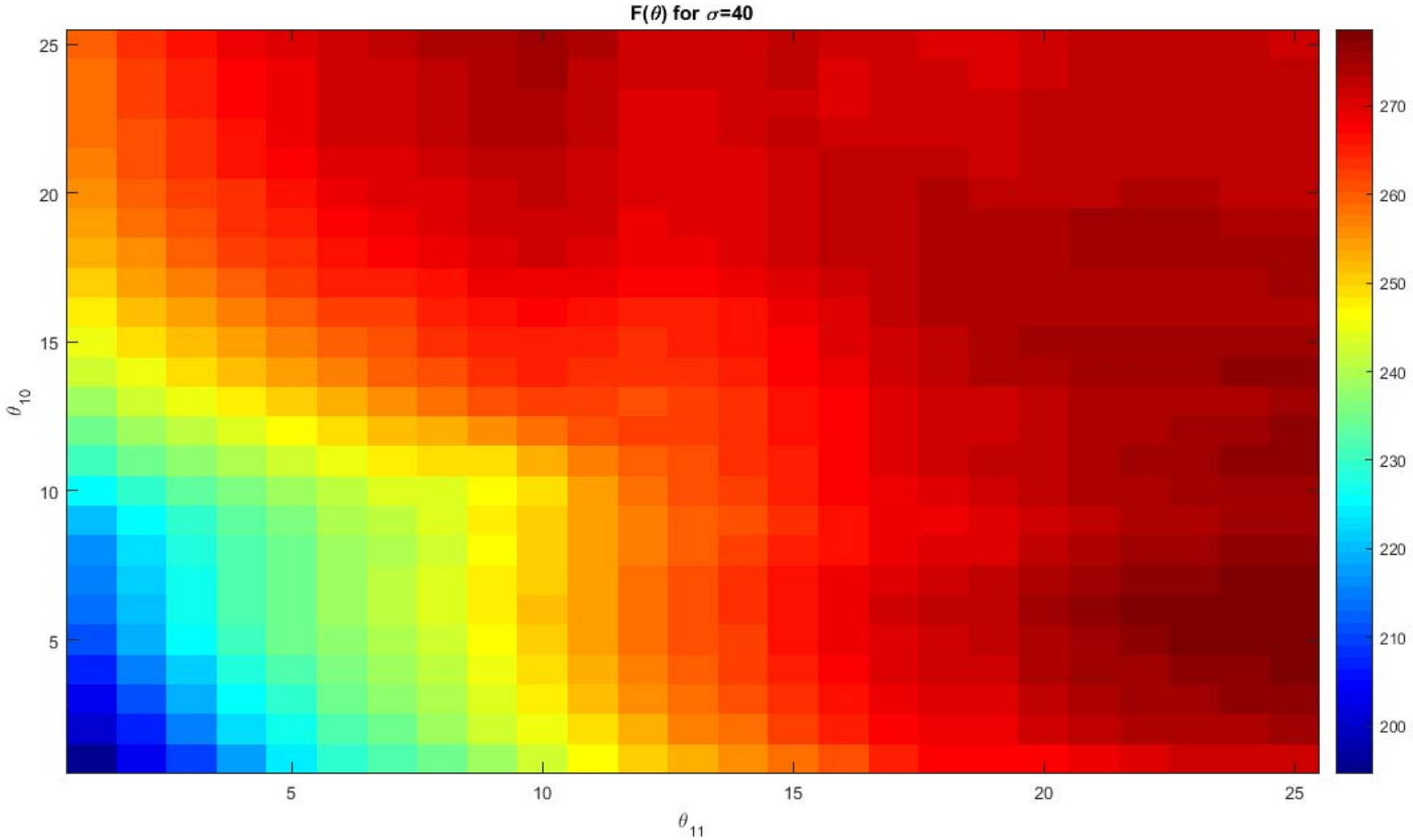


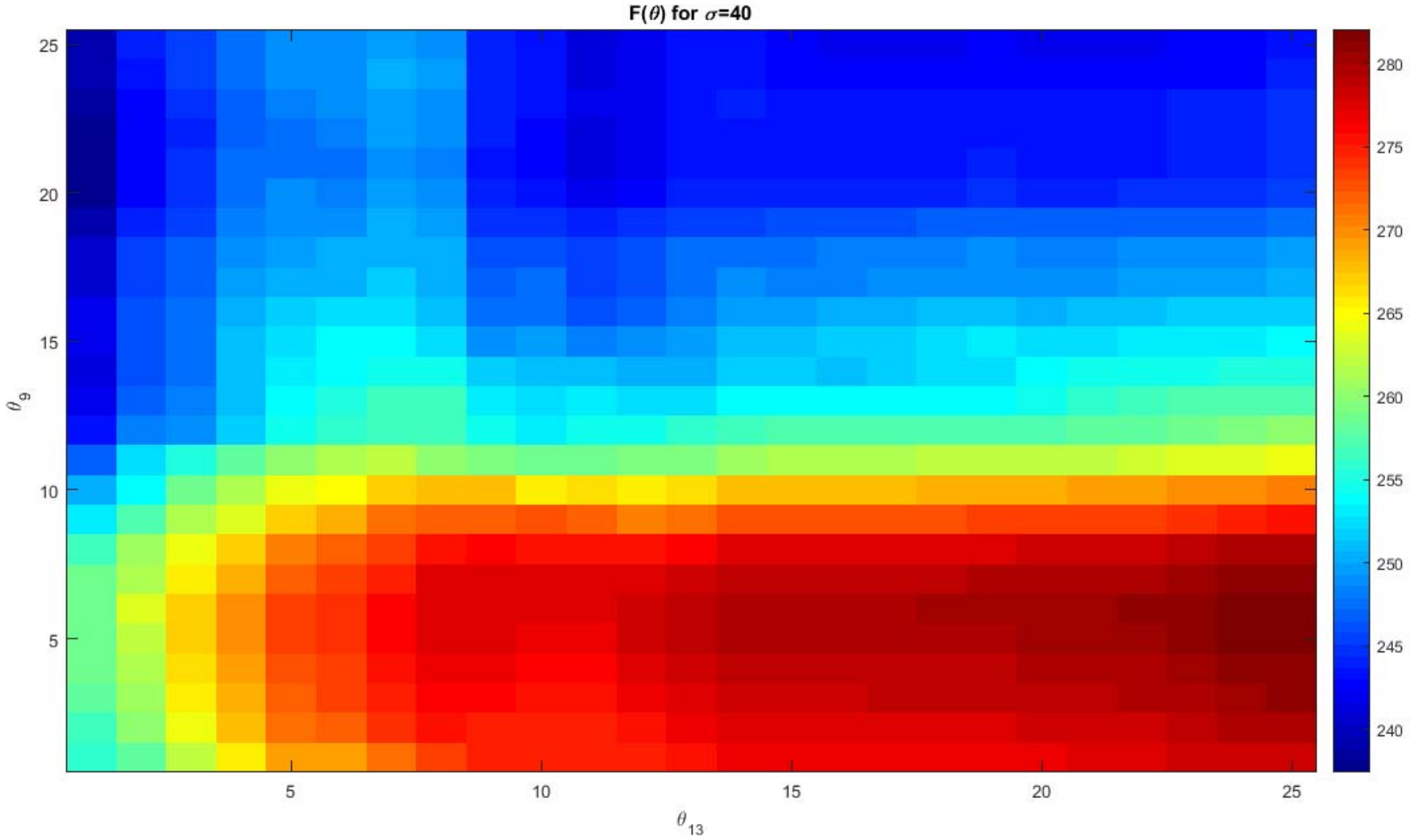
- One-dimensional contour plots-uncertain forecast
  - » θ_i for i=9,…, 15 hours into the future

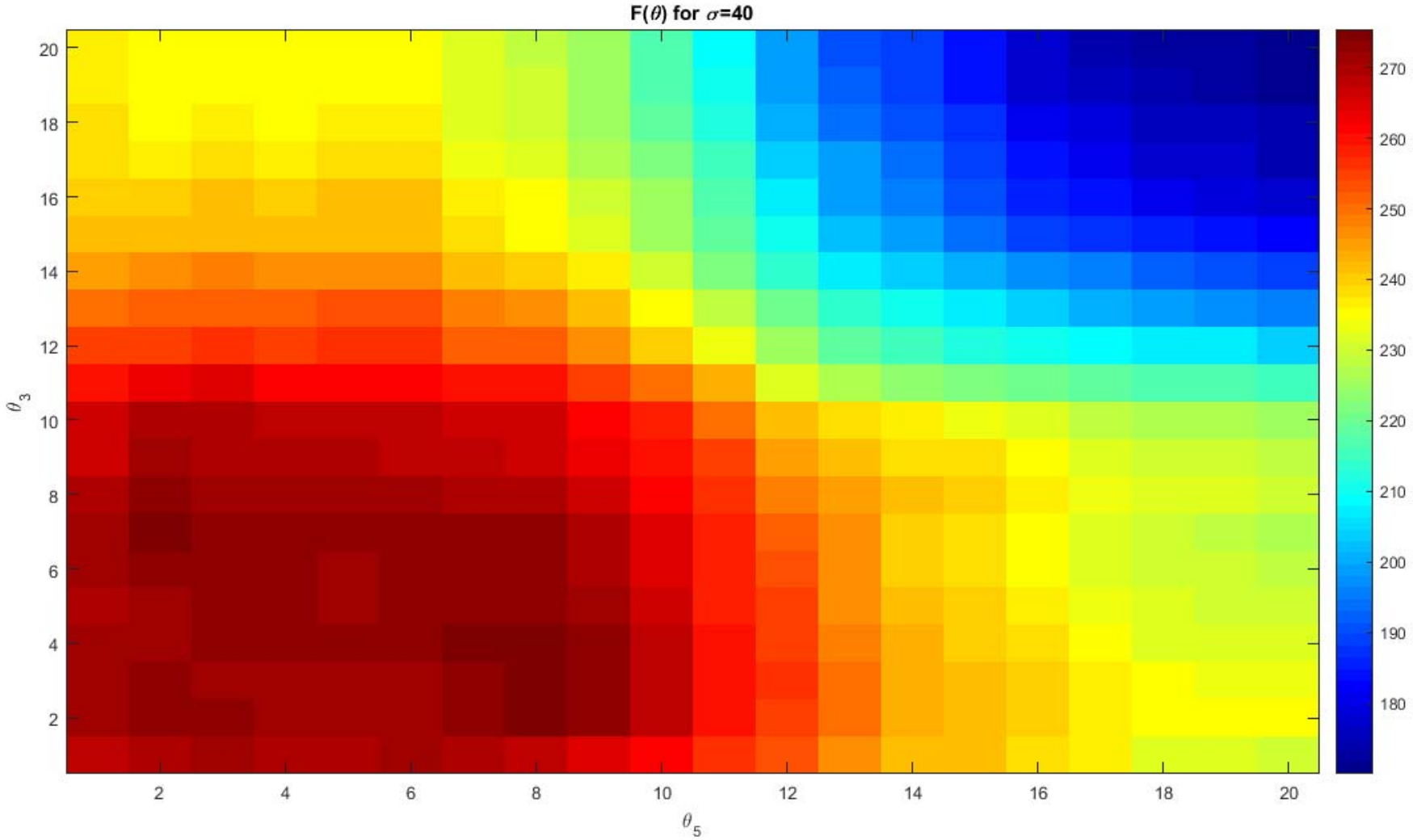


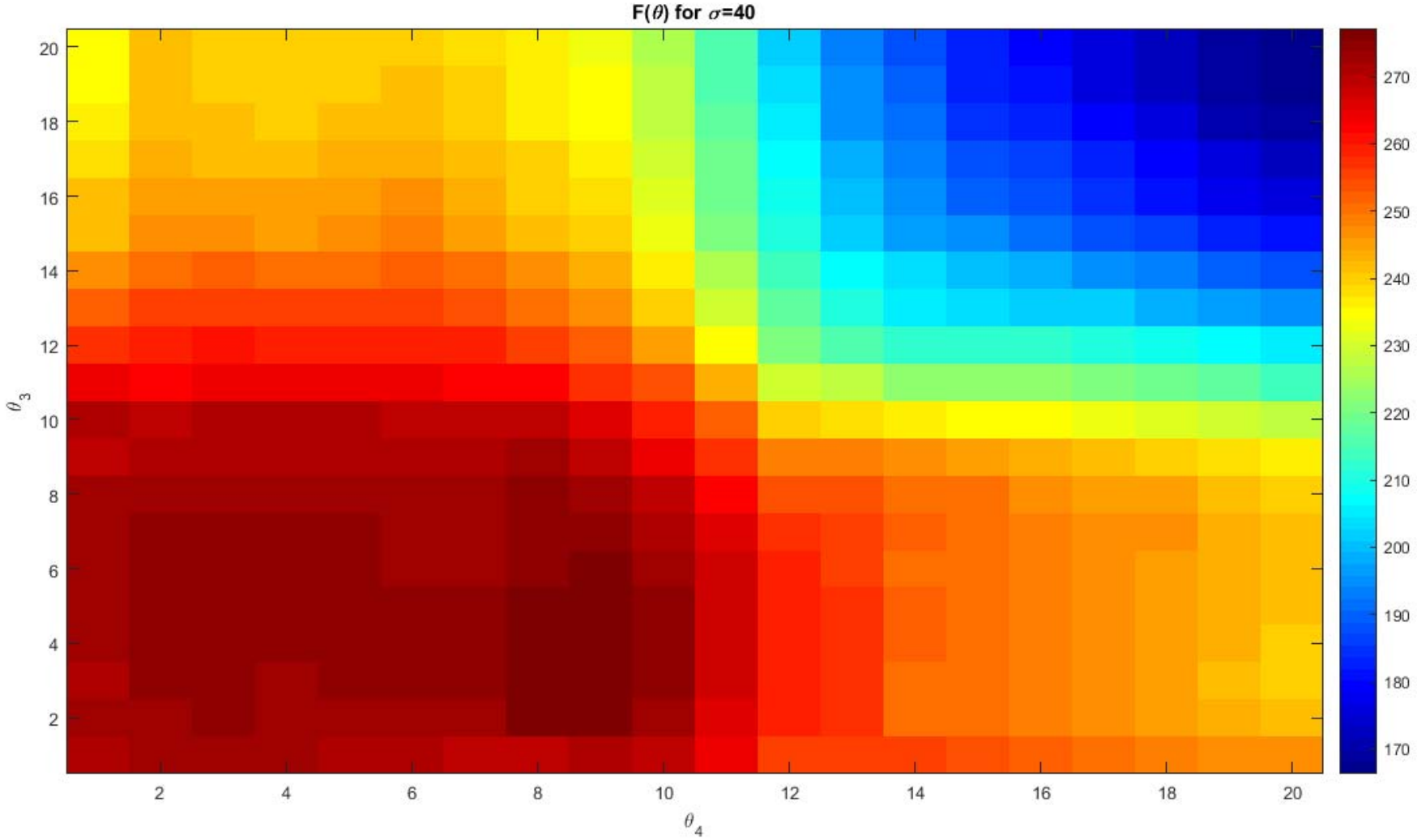

F(θ) for σ=40
20
18
16
14
12
10
8
6
4
2
θ1
2
4
6
8
10
12
14
16
18
20
θ10
275
270
265
260
255
250
245
240

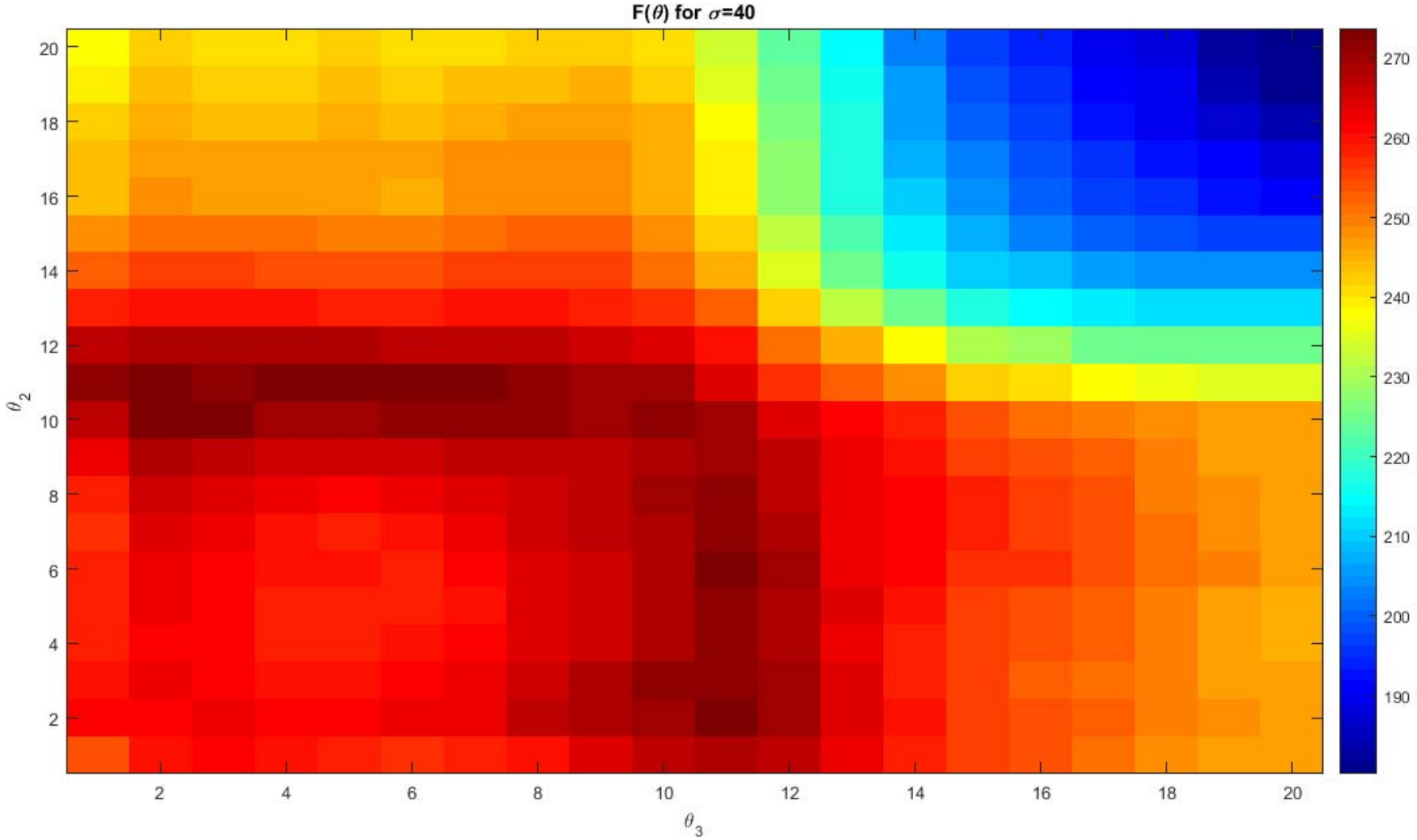

F(θ) for σ=40
40
35
30
25
20
15
10
5
θ1
5
10
15
20
25
30
35
40
θ2
270
260
250
240
230
220
210
200
190
180
170

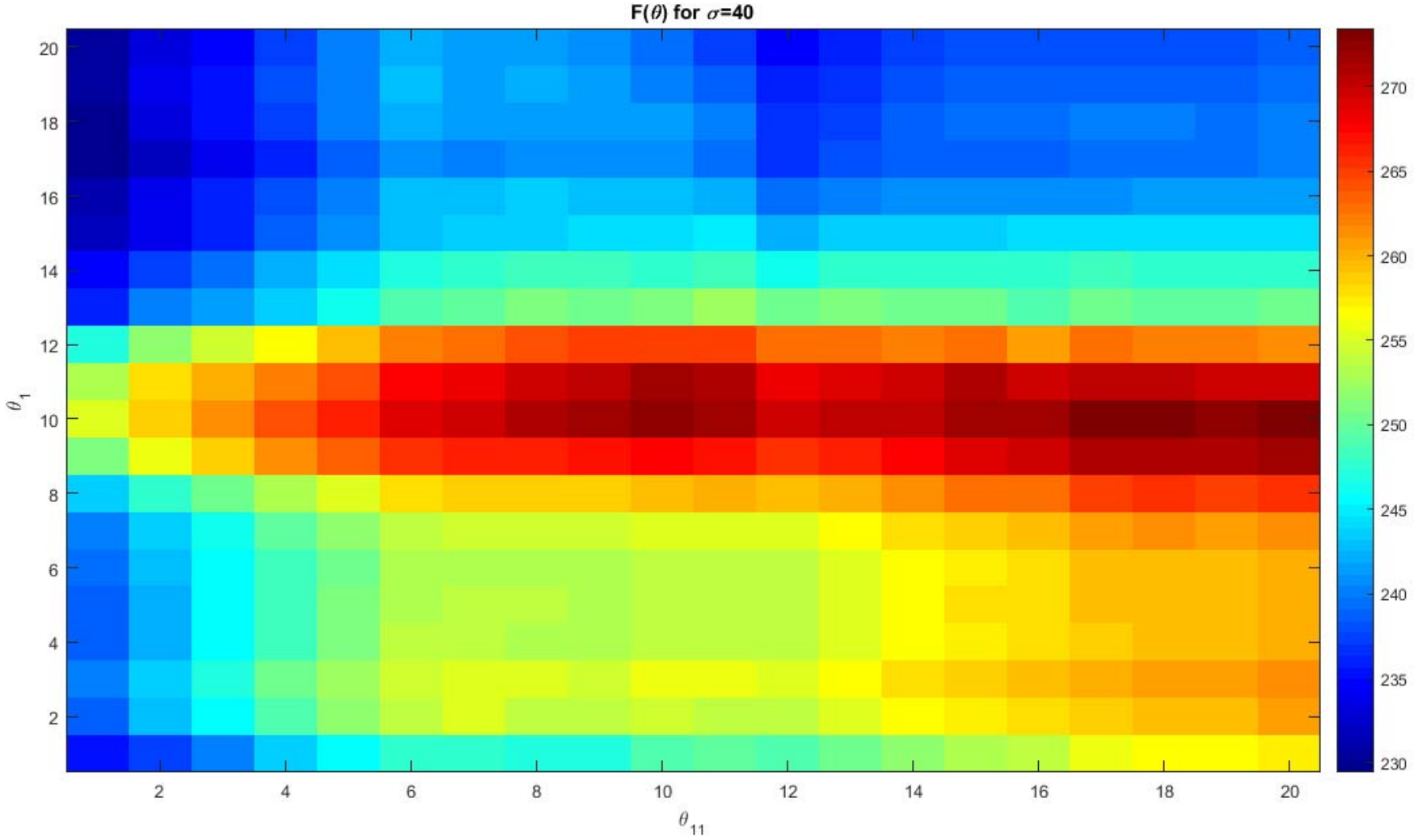

F(θ) for σ=40
25
20
15
10
5
θ10
5
10
15
20
25
θ11
270
260
250
240
230
220
210
200


F(θ) for σ=40
25
20
15
10
5
θ9
5
10
15
20
25
θ13
280
275
270
265
260
255
250
245
240


F(θ) for σ=40
θ3
θ5
20
18
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16
18
20
270
260
250
240
230
220
210
200
190
180


F(θ) for σ=40
θ3
θ4
20
18
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16
18
20
270
260
250
240
230
220
210
200
190
180
170


F(θ) for σ=40
θ2
θ3
2
4
6
8
10
12
14
16
18
20
190
200
210
220
230
240
250
260
270


F(θ) for σ=40
θ1
θ11
20
18
16
14
12
10
8
6
4
2
2
4
6
8
10
12
14
16
18
20
270
265
260
255
250
245
240
235
230

Any policy may work best

An energy storage problem

- Consider a basic energy storage problem:

» We are going to show that with minor variations in the characteristics of this problem, we can make *each* class of policy work best.

An energy storage problem

- We can create distinct flavors of this problem:
  - » Problem class 1 – Best for PFAs
    - Highly stochastic (heavy tailed) electricity prices
    - Stationary data
  - » Problem class 2 – Best for CFAs
    - Stochastic prices and wind (but not heavy tailed)
    - Stationary data
  - » Problem class 3 - Best for VFAs
    - Stochastic wind and prices (but not too random)
    - Time varying loads, but inaccurate wind forecasts
  - » Problem class 4 – Best for deterministic lookaheads
    - Relatively low noise problem with accurate forecasts
  - » Problem class 5 – A hybrid policy worked best here
    - Stochastic prices and wind, nonstationary data, noisy forecasts.

An energy storage problem

- The policies
  - » The PFA:
    - Charge battery when price is below p1
    - Discharge when price is above p2
  - » The CFA
    - Optimize over a horizon H; maintain upper and lower bounds (u, l) for every time period except the first (note that this is a hybrid with a lookahead).
  - » The VFA
    - Piecewise linear, concave value function in terms of energy, indexed by time.
  - » The lookahead (deterministic)
    - Optimize over a horizon H (only tunable parameter) using forecasts of demand, prices and wind energy
  - » The lookahead CFA
    - Use a lookahead policy (deterministic), but with a tunable parameter that improves robustness.

- PFA:

For the PFA policy, we use the following rule:

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

- CFA:

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

$$X^{CFA-EC}(S_t|\theta) = \arg\min_{x_t \in \mathcal{X}_t} C(S_t, x_t) + \theta * \left(x_t^{GB} + x_t^{EB} + x_t^{BL}\right) \quad (18)$$

where $\mathcal{X}_t$ is defined by (7)–(11). We use a linear correction term for simplicity (and it sometimes works).

- VFA:

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

$$X^{VFA}(S_t) = \arg \min_{x_t \in \mathcal{X}_t} C(S_t, x_t) + \bar{V}_t^x \left(R_t^x\right) \qquad (19)$$

DLA:

The deterministic lookahead optimizes over a horizon H using forecasts of exogenous information:

$$X_t^{LA-DET}(S_t|H) = \arg \min_{(\tilde{x}_{tt},\ldots,\tilde{x}_{t,t+H})} \sum_{t'=t}^{t+H} C(S_{tt'}, \tilde{x}_{tt'}) \quad (20)$$

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

$$\tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{EB} \leq f_{tt'}^{E}, \quad (21)$$

$$f_{tt'}^{\eta}\left(\tilde{x}_{tt'}^{GL} + \tilde{x}_{tt'}^{EL} + \tilde{x}_{tt'}^{BL}\right) = f_{tt'}^{L}, \quad (22)$$

$$\tilde{x}_{tt'}^{BL} \leq \tilde{R}_{tt'}, \quad (23)$$

$$\tilde{R}_{t,t'+1} - \left(\tilde{R}_{tt'} + f_{t,t'+1}^{\eta}\left(\tilde{x}_{tt'}^{GB} + \tilde{x}_{tt'}^{EB}\right) - \tilde{x}_{tt'}^{BL}\right) = f_{t,t'+1}^{R}, \quad (24)$$

$$\tilde{x}_{tt'} \geq 0. \quad (25)$$

- Hybrid DLA-CFA
  - » Use the DLA with two new constraints:

The last policy, $X_t^{LA-CFA}(S_t|\theta^L,\theta^U)$, is a hybrid lookahead with a form of cost function approximation in the form of two additional constraints for $t' = t+1,\ldots,T$:

$$\tilde{R}_{tt'} \geq \theta^L, \tag{26}$$

$$\tilde{R}_{tt'} \leq \theta^U. \tag{27}$$

  - » These are designed to avoid allowing the storage to hit upper and lower bounds so that it can respond to sudden bursts or gaps in the energy from wind.

An energy storage problem

- Each policy is best on certain problems
  - » Results are percent of *posterior* optimal solution

Problem:	Problem description	PFA	CFA Error correction	VFA	Determ. Lookahead	CFA Lookahead
A	A stationary problem with heavy-tailed prices, relatively low noise, moderately accurate forecasts.	**0.959**	**0.839**	**0.936**	**0.887**	**0.887**
B	A time-dependent problem with daily load patterns, no seasonalities in energy and price, relatively low noise, less accurate forecasts.	**0.714**	**0.752**	**0.712**	**0.746**	**0.746**
C	A time-dependent problem with daily load, energy and price patterns, relatively high noise, forecast errors increase over horizon.	**0.865**	**0.590**	**0.914**	**0.886**	**0.886**
D	A time-dependent problem, relatively low noise, very accurate forecasts.	**0.962**	**0.749**	**0.971**	**0.997**	**0.997**
E	Same as (C), but the forecast errors are stationary over the planning horizon.	**0.865**	**0.590**	**0.914**	**0.922**	**0.934**

  - » … any policy might be best depending on the data.

Joint research with Prof. Stephan Meisel, University of Muenster, Germany.

An energy storage problem

- From the paper:

Tutorial on Stochastic Optimization in Energy—Part II: An Energy Storage Illustration

Warren B. Powell, *Member, IEEE*, and Stephan Meisel

***Abstract*—In Part I of this tutorial, we provided a canonical modeling framework for sequential, stochastic optimization (control) problems. A major feature of this framework is a clear separation of the process of modeling a problem, versus the design of policies to solve the problem. In Part II, we provide additional discus-**

TRANSACTIONS ON POWER SYSTEMS approach the modeling of stochastic optimization problems, where we highlight what we feel are differences in the approaches used by the literature versus the modeling strategy we are proposing.

We then illustrate our modeling framework in

Extension

Active learning

An energy storage problem

- Types of learning:
  - » No learning ($\theta's$ are known)

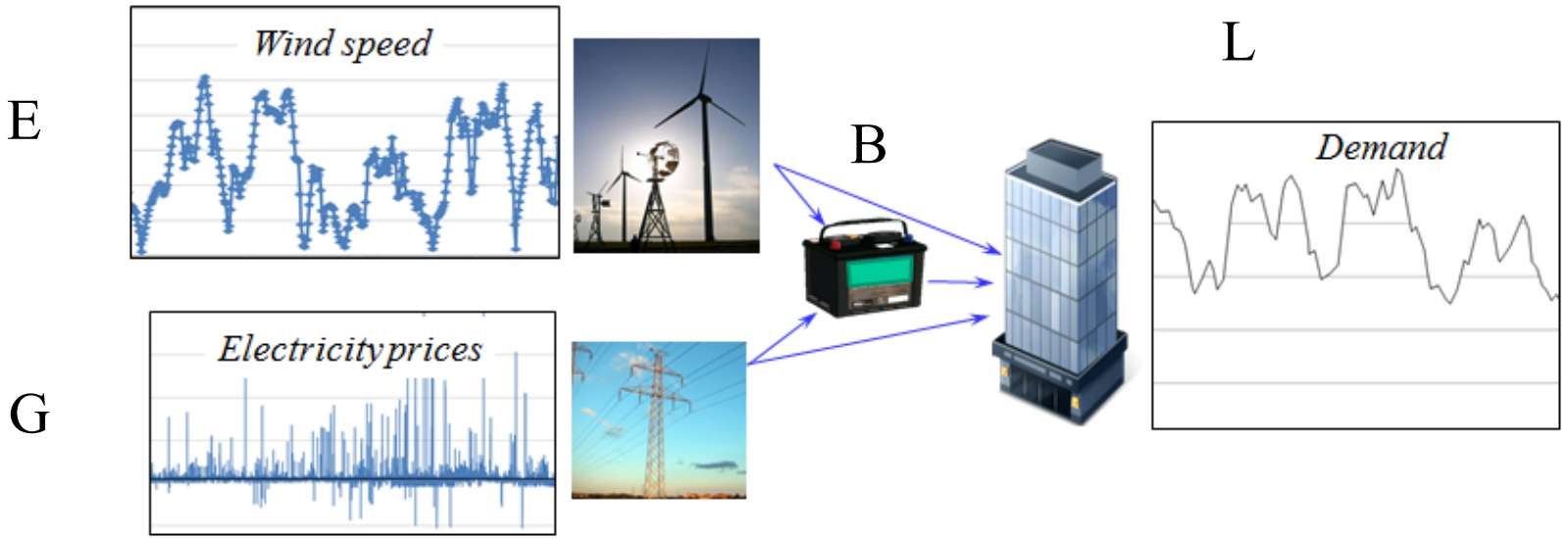
$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon^p_{t+1}$$

  - » Passive learning (learn $\bar{\theta}s$ from price data)

$$p_{t+1} = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2} + \varepsilon^p_{t+1}$$

An energy storage problem

- Transition function



$$E_{t+1} = E_t + \hat{E}_{t+1}$$

$$p_{t+1} = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2} + \varepsilon^p_{t+1} = (\bar{\theta}_t)^T \bar{p}_t + \varepsilon^p_{t+1} \quad \bar{p}_t = \begin{pmatrix} p_t \\ p_{t-1} \\ p_{t-2} \end{pmatrix}$$

$$D_{t+1} = f^D_{t,t+1} + \varepsilon^D_{t+1}$$

$$R^{battery}_{t+1} = R^{battery}_t + x_t$$

Learning in stochastic optimization

- Updating the demand parameter
  - » Let p_{t+1} be the new price and let

$$\bar{F}_t^{price}(\bar{p}_t \mid \bar{\theta}_t) = (\bar{\theta}_t)^T \bar{p}_t = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2}$$

  - » We update our estimate $\bar{\theta}_t$ using our recursive least squares equations:

$$\bar{\theta}_{t+1} = \bar{\theta}_t - \frac{1}{\gamma_{t+1}} B_t \bar{p}_t \varepsilon_{t+1}$$

$$\varepsilon_{t+1} = \bar{F}_t^{price}(\bar{p}_t \mid \bar{\theta}_t) - p_{t+1},$$

$$B_{t+1} = B_t - \frac{1}{\gamma_{t+1}} \left(B_t \bar{p}_t (\bar{p}_t)^T B_t \right)$$

$$\gamma_{t+1} = 1 + (\bar{p}_t)^T B_t \bar{p}_t$$

An energy storage problem

- Types of learning:
  - » No learning ($\theta's$ are known)

$$p_{t+1} = \theta_0 p_t + \theta_1 p_{t-1} + \theta_2 p_{t-2} + \varepsilon^p_{t+1}$$

  - » Passive learning (learn $\bar{\theta}s$ from price data)

$$p_{t+1} = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2} + \varepsilon^p_{t+1}$$

  - » Active learning ("bandit problems") – What we learn depends on the decisions x^{GB}_t

$$p_{t+1} = \bar{\theta}_{t0} p_t + \bar{\theta}_{t1} p_{t-1} + \bar{\theta}_{t2} p_{t-2} + \bar{\theta}_{t3} x^{GB}_t + \varepsilon^p_{t+1}$$

Buy/sell decisions

Revenue management

- Earning vs. learning
  - » You earn the most with prices near the middle.
  - » You learn the most with extreme prices.
  - » Challenge is to strike a balance.

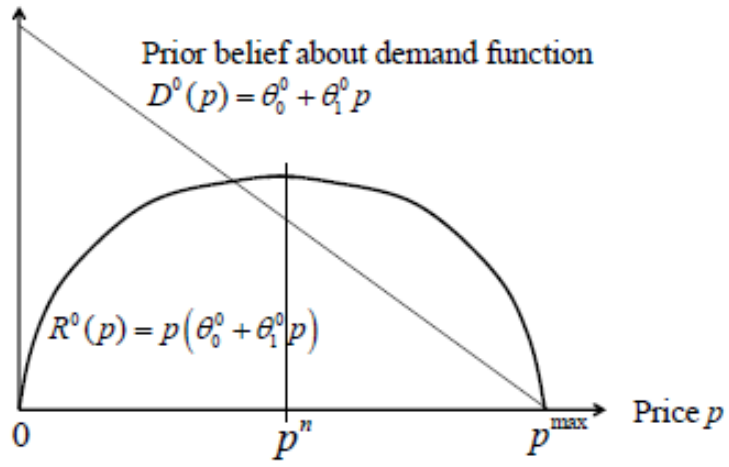


Figure 8.5 Prior demand function and revenue curve.

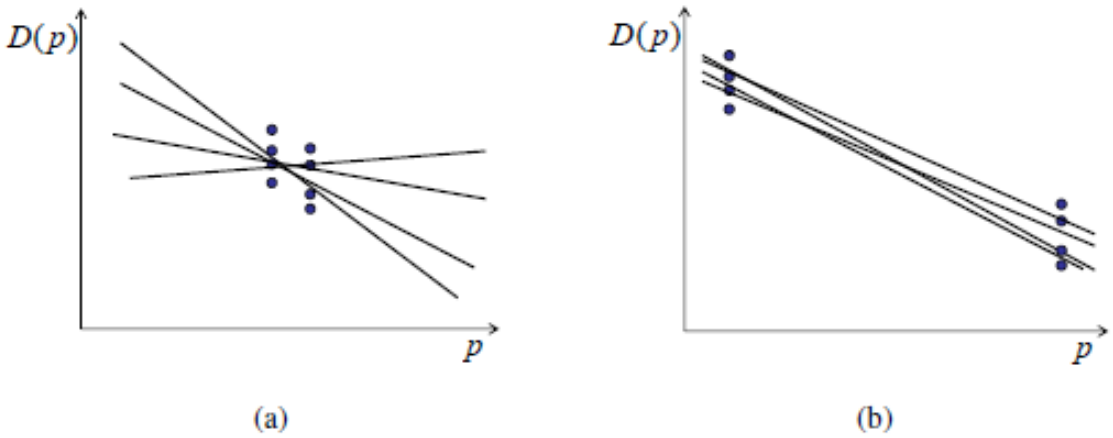


Figure 8.6 Estimating the demand function using (a) observations near the middle and (b) observations near the endpoints.

- Notes
  - » We started with a pure stochastic optimization problem with no learning.
  - » The previous slide (optimizing prices) is a pure learning problem.
  - » We can also have a mixed problem.
  - » With the last model, our decisions to charge/discharge energy from/to the grid affects our learning of $\bar{\theta}_{t3}$.
  - » The model (e.g. state variables) are the same.
  - » We would expect the choice of best policy (or the best parameters for a parameterized policy) would change when there is active learning, but how much depends on the relative value of information.

Extension

Learning

Week 6 – Wednesday

Midterm