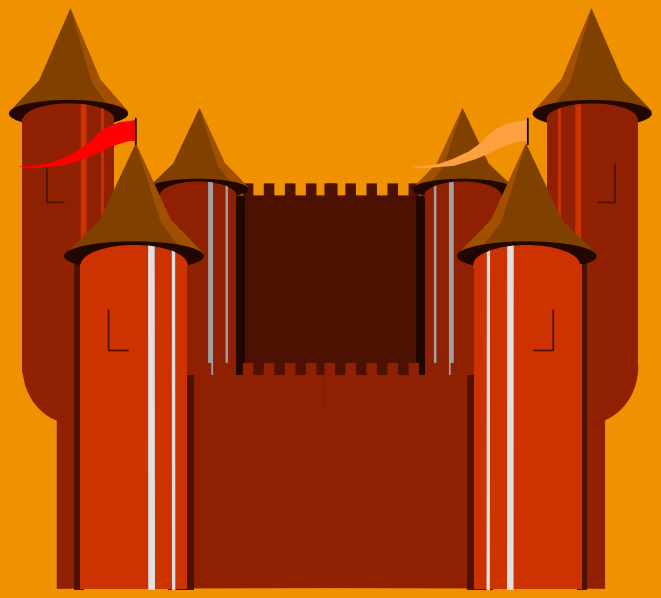
ORF 544

Stochastic Optimization and Learning

Spring, 2019



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

Week 9

Cost function approximations

Cost function approximations

- A general CFA policy:
  - » Define our policy:

$$X_t^\pi(\theta) = \arg\max_x \bar{C}^\pi(S_t, x_t \mid \theta)$$

Parametrically modified costs

subject to

$$Ax = \bar{b}^\pi(\theta)$$

Parametrically modified constraints

  - » We tune θ by optimizing:

$$\min_\theta F^\pi(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X_t^\pi(\theta))$$

Cost function approximations

- A general CFA policy:
  - » Define our policy:

$$X_t^\pi(\theta) = \arg\max_x \left(C(S_t, x_t) + \sum_{f \in F} \theta_f^\phi \phi_f(S_t, x_t) \right)$$

subject to

$$Ax = \theta^c + \theta^b \otimes b$$

  - » We tune θ by optimizing:

$$\min_\theta F^\pi(\theta) = \mathbb{E} \sum_{t=0}^{T} C(S_t, X_t^\pi(\theta))$$

Cost function approximation

- Notes
  - » CFAs can handle high-dimensional decision vectors.
  - » There is no difference doing policy search for PFAs or CFAs.
  - » We can apply the same methods we have used before:
    - Derivative-free
    - Derivative-based
  - » Parameterized policies are easy to compute than the lookahead versions, but policy search remains challenging.

Cost function approximation

Logistics

Cost function approximations

- Inventory management
  - » How much product should I order to anticipate future demands?
  - » Need to accommodate different sources of uncertainty.
    - Market behavior
    - Transit times
    - Supplier uncertainty
    - Product quality

Cost function approximations

- Imagine that we want to purchase parts from different suppliers. Let x_{tp} be the amount of product we purchase at time t from supplier p to meet forecasted demand D_t. We would solve

$$X_t^{\pi}(S_t) = \arg\max_{x_t} \sum_{p \in P} c_p x_{tp}$$

subject to

$$\left.\begin{aligned} \sum_{p \in P} x_{tp} &\geq D_t \\ x_{tp} &\leq u_p \\ x_{tp} &\geq 0 \end{aligned}\right\} \mathcal{X}_t$$

» This assumes our demand forecast D_t is accurate.

Cost function approximations

- Imagine that we want to purchase parts from different suppliers. Let x_{tp} be the amount of product we purchase at time t from supplier p to meet forecasted demand D_t. We would solve

$$X_t^\pi(S_t \mid \theta) = \arg\max_{x_t \in \mathcal{X}^\pi(\theta)} \sum_{p \in P} c_p x_{tp}$$

subject to

$$\sum_{p \in P} x_{tp} \geq \theta^{\text{Reserve}} D_t$$

$$x_{tp} \leq u_p$$

$$x_{tp} \geq \theta^{\text{buffer}}$$

$\mathcal{X}_t^\pi(\theta)$

Reserve

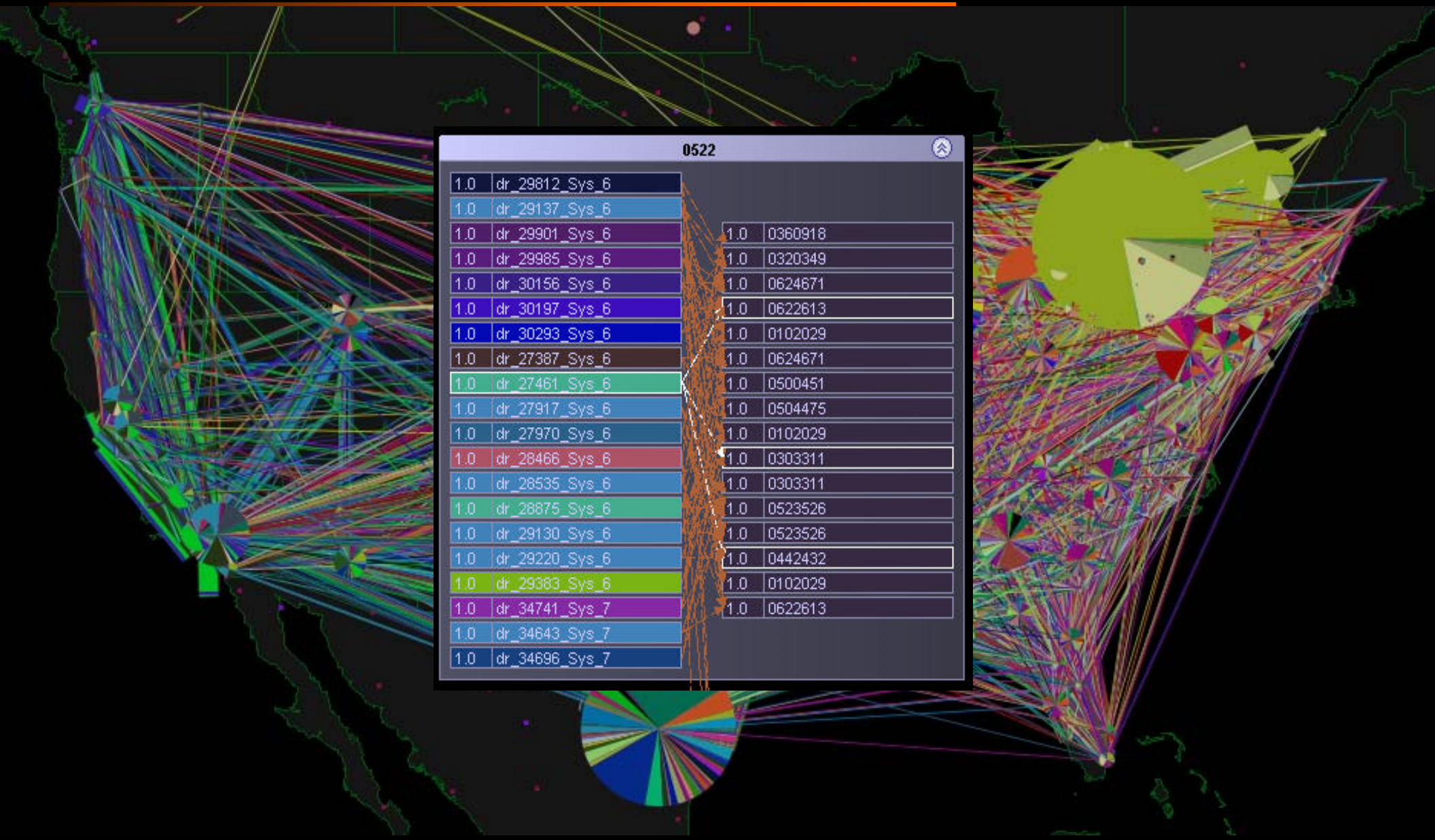
Buffer stock

» This is a "parametric cost function approximation"

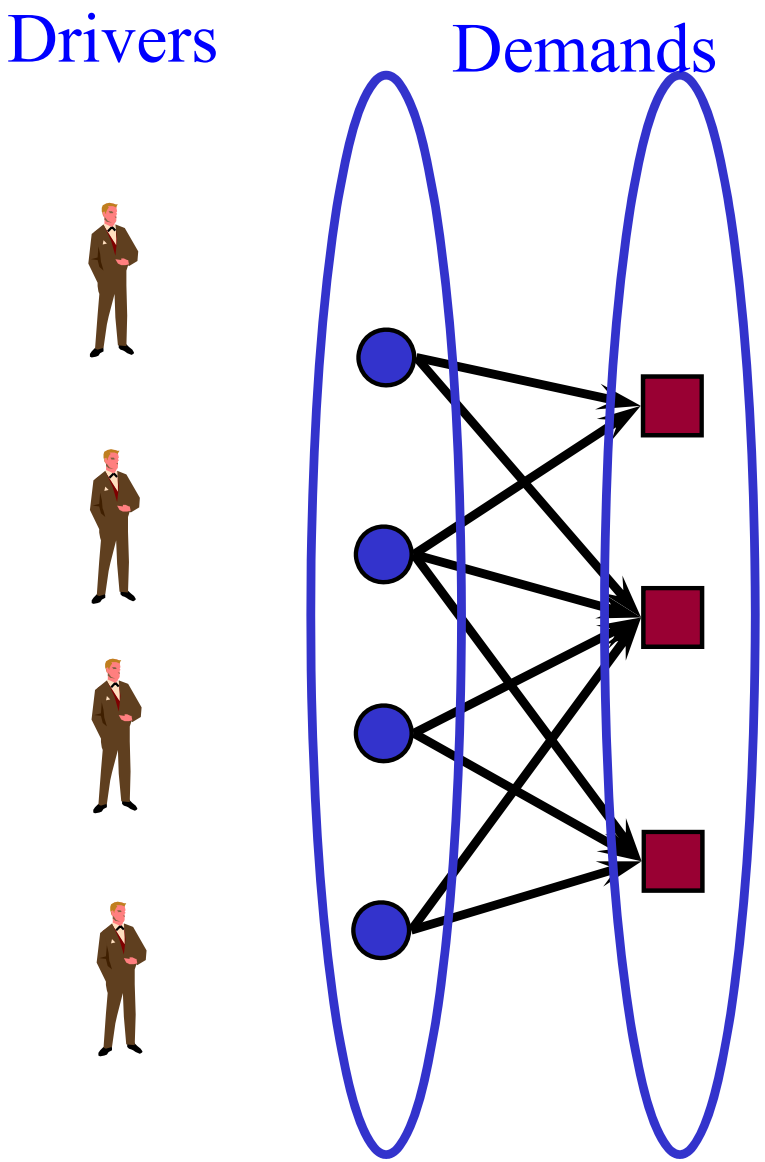
Cost function approximation

Driver dispatching for truckload

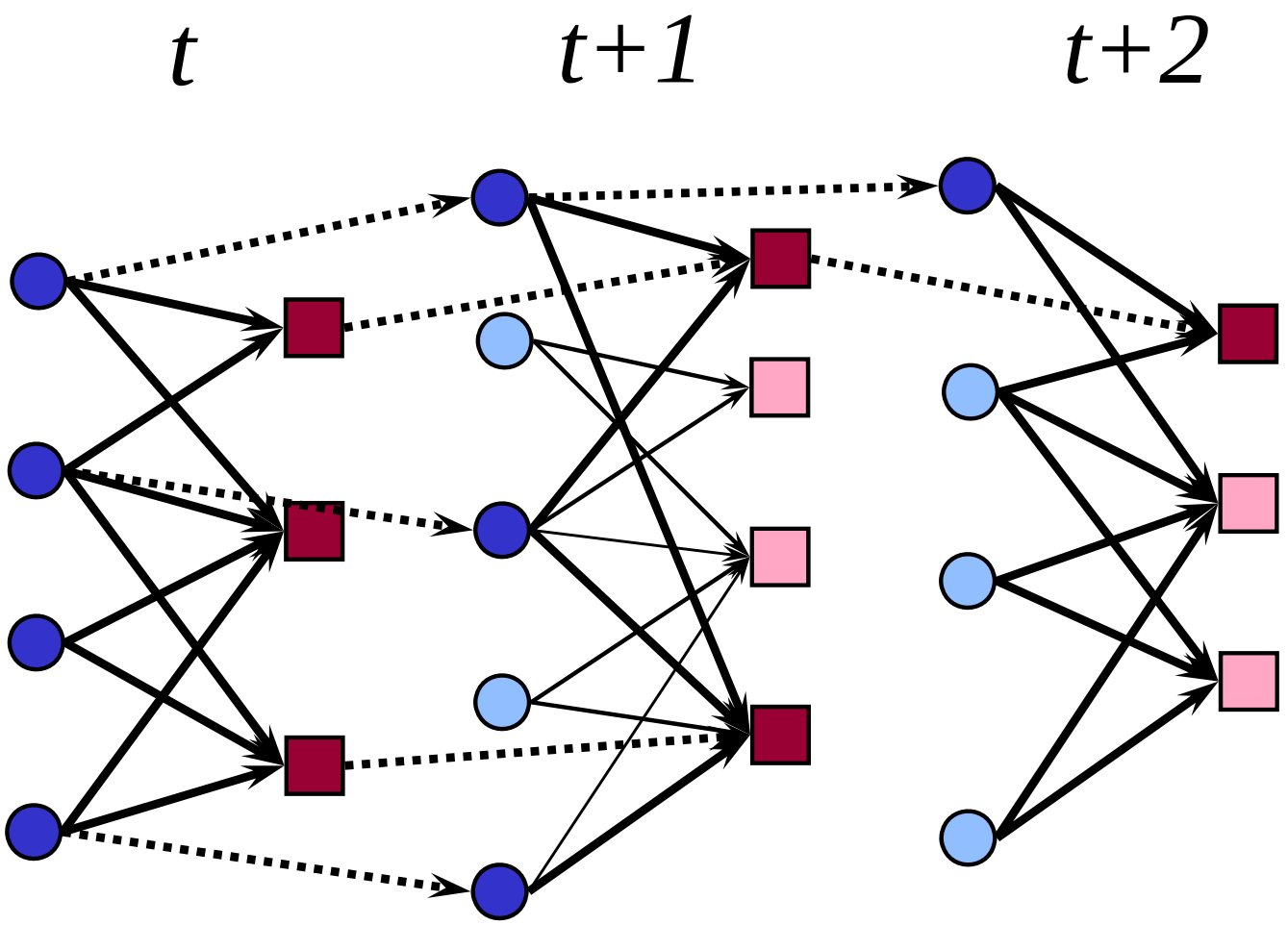
Schneider National



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

$$\min_{\theta} F^{\pi}(\theta) = \mathbb{E}\sum_{t=0}^{T} C(S_t, X_t^{\pi}(S_t \mid \theta))$$

Cost function approximation

SMART-Invest

One-dimensional parameter search

SMART-Invest

Cost-minimized combinations of wind power, solar power and electrochemical storage, powering the grid up to 99.9% of the time

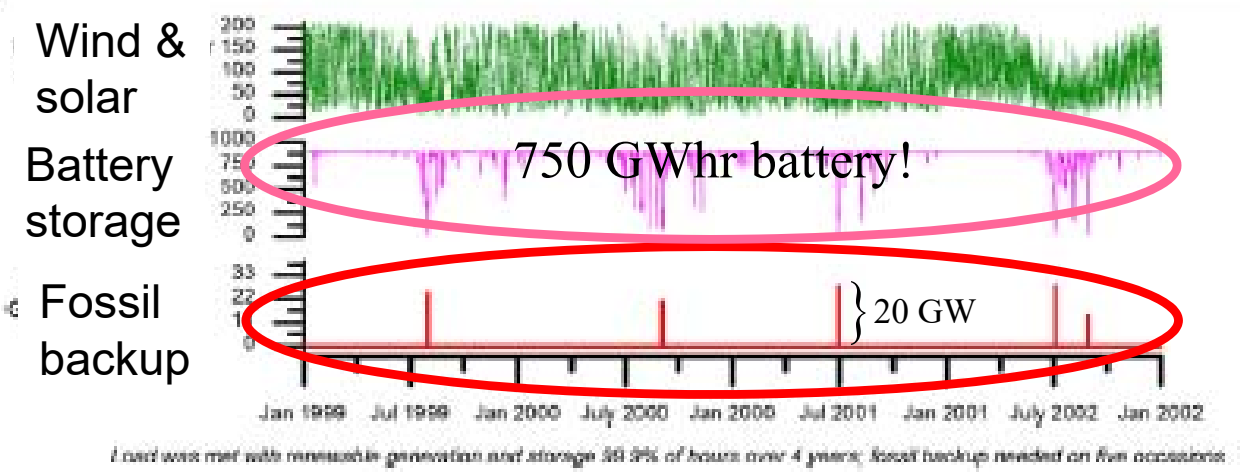
Cory Budischak [a,b,*], DeAnna Sewell [c], Heather Thomson [c], Leon Mach [d], Dana E. Veron [c], Willett Kempton [a,c,e]

[a] Department of Electrical and Computer Engineering, University of Delaware, Newark, DE 19716, USA
[b] Department of Energy Management, Delaware Technical Community College, Newark, DE 19713, USA
[c] Center for Carbon-Free Power Integration, School of Marine Science and Policy, College of Earth Ocean and Environment, University of Delaware, Newark, DE 19716, USA
[d] Energy and Environmental Policy Program, College of Engineering, University of Delaware, Newark, DE 19716, USA
[e] Center for Electric Technology, DTU Elektro, Danmarks Tekniske Universitet, Kgs. Lyngby, Denmark

HIGHLIGHTS

- We modeled wind, solar, and storage to meet demand for 1/5 of the USA electric grid.
- 28 billion combinations of wind, solar and storage were run, seeking least-cost.
- Least-cost combinations have excess generation (3× load), thus require less storage.
- 99.9% of hours of load can be met by renewables with only 9–72 h of storage.
- At 2030 technology costs, 90% of load hours are met at electric costs below today's.

GRAPHICAL ABSTRACT



Load was met with renewable generation and storage 99.9% of hours over 4 years; fossil backup needed on five occasions

SMART-Invest

- Model used in 99 percent paper:
  - » If wind+solar > load, store excess in battery (if possible)
  - » If wind+solar < load, draw from battery (if available)
  - » If outage, assume we can get what we need from the grid.
  - » Simulate over 8,760 hours.
- Now, optimize
  - » Enumerate 28 billion combinations of wind, solar and storage.
  - » Find the least cost combination that meets some level of coverage (e.g. 80 percent, or 99.9 percent)

SMART-Invest

- Limitations:
  - » Ignores the reality that you are going to need a significant portion from slow and fast fossil.
  - » Ignores the cost of uncertainty when planning fossil generations.
  - » Ignores the marginal cost of investment.
    - Choosing the least cost to get 99 percent ignores the marginal cost of getting to 99 percent.

- To fix these problems:
  - » We created "SMART-Invest" using a Lookahead-CFA to plan a full energy portfolio.

SMART-Invest

- Elements of SMART-Invest
  - » Simulates energy planning process each hour over entire year (8760 hours), planning 36 hours into the future.
  - » Properly models advance commitment decisions for slow fossil, fast fossil, and real time ramping and storage.
- It does not model:
  - » The grid
  - » Individual fossil generators

SMART-Invest

- The investment problem:

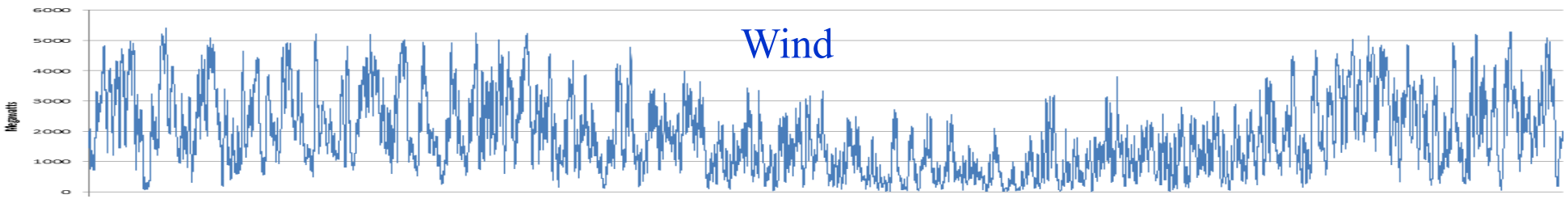
$$\min_{x_i^{Inv},\, i \in I} \left(C^{inv}(x^{inv}) + \sum_{t=1}^{8760} C_t^{opr}\left(S_t, X_t^{opr}(S_t \mid x^{inv})\right) \right)$$

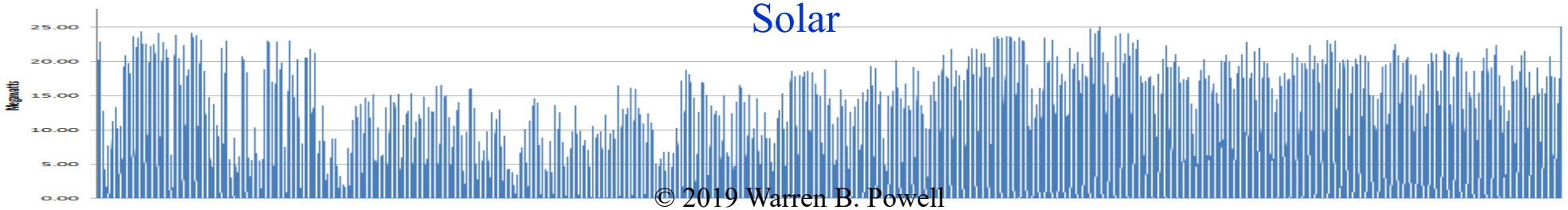
Capital investment cost in wind, solar and storage

Operating costs of fossil generators, energy losses from storage, misc. operating costs of renewables.

Investment cost in wind, solar and storage.

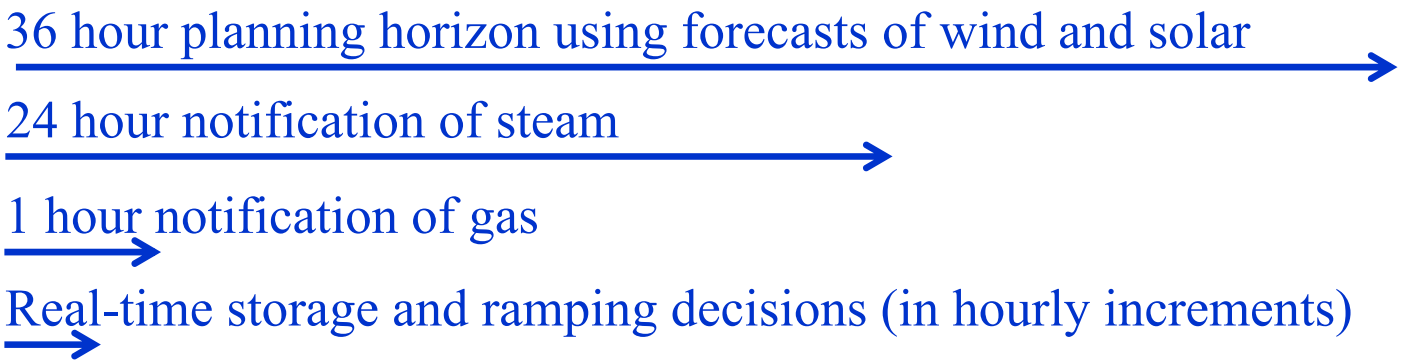
$X^{opr}(S_t)$ is the operating policy.





SMART-Invest

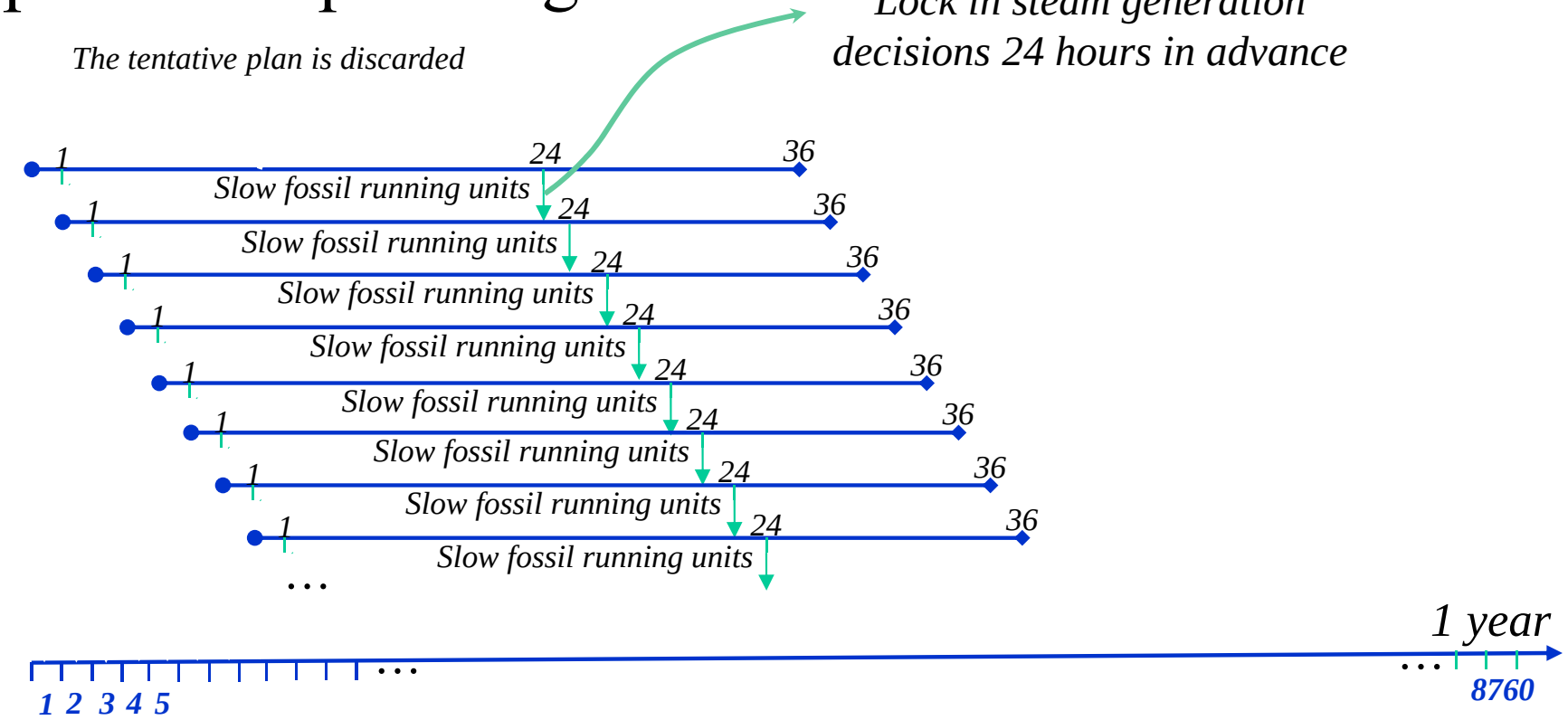
- Operational planning



  - » Meet demand while minimizing operating costs
  - » Observe day-ahead notification requirements for generators
  - » Includes reserve constraints to manage uncertainty
  - » Meet aggregate ramping constraints (but does not schedule individual generators)

SMART-Invest

- Operational planning model



» Model plans using rolling 36 hour horizon

» Steam plants are locked in 24 hours in advance

» Gas turbines are decided 1 hour in advance

SMART-Invest

- Lookahead model:
  - » Objective function

$$X_t^\pi(S_t|\theta,(x_g^{inv})_{g\in\mathcal{G}}) = \arg\min_{(\tilde{x}_{t,t'})_{t'}} \sum_{t'=t}^{t+n^T} \sum_{g\in G} \left(c_g^{opr} \tilde{x}_{t,t',g}^{gen} + c^{cb} \tilde{x}_{t,t'}^{bc} + \frac{c^{db}}{e^d} \tilde{x}_{t,t'}^{bd} \right)$$

  - » Reserve constraint:

$$\sum_{g\in G} \tilde{x}_{t,t',g}^{gen} = \theta_{t,t'}^{rd}$$

Tunable policy parameter

  - » Other constraints:
    - Ramping
    - Capacity constraints
    - Conservation of flow in storage
    - ….

SMART-Invest

- The objective function

$$\min_{\pi} E^{\pi} \left\{ \sum_{t=0}^{T} \gamma^t C\left(S_t, X_t^{\pi}(S_t)\right) \right\}$$

Given a *system model* (transition function)

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

And exogenous information process:

$$(S_0, x_0, W_1, S_1, \ldots, x_{t-1}, W_t, S_t)$$

This involves running a simulator.

SMART-Invest

- For this project, we used numerical derivatives:
  - » Objective function is

$$\bar{F}(\theta,\omega)=\sum_{t=0}^{T}C\left(S_t(\omega),X_t^{\pi}(S_t(\omega)\mid\theta)\right)$$

  - » Numerical derivative is

$$\nabla_x F(\omega)=\frac{F^{\pi}(\theta+\delta,\omega)-F^{\pi}(\theta,\omega)}{\delta}$$

  - » or

$$\nabla_x F(\omega)=\frac{F^{\pi}(\theta+\delta,\omega)-F^{\pi}(\theta-\delta,\omega)}{2\delta}$$

SMART-Invest

- Optimizing the investment parameters
  - » The model optimizes investment in:
    - Wind
    - Solar
    - Storage
    - Fast and slow fossil (for some times of runs)
  - » Algorithm
    - Stochastic gradient using numerical derivatives
    - Custom stepsize formula
  - » Research opportunity:
    - Use adaptive learning model using some sort of parametric or local parametric belief model.

SMART-Invest

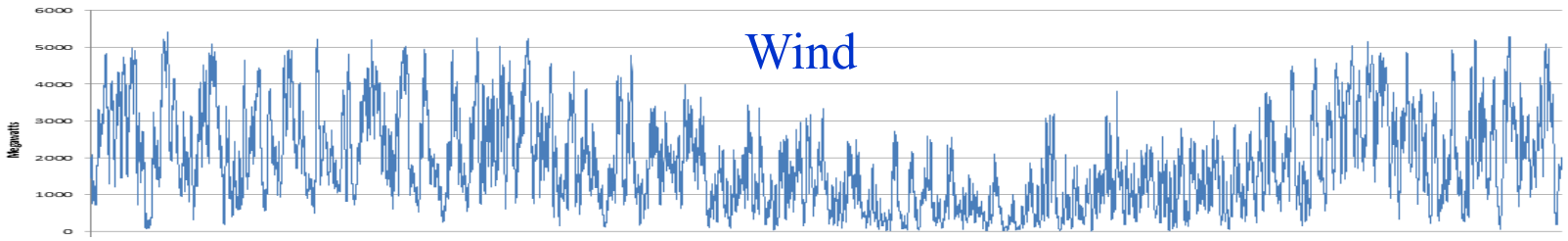
Update investments

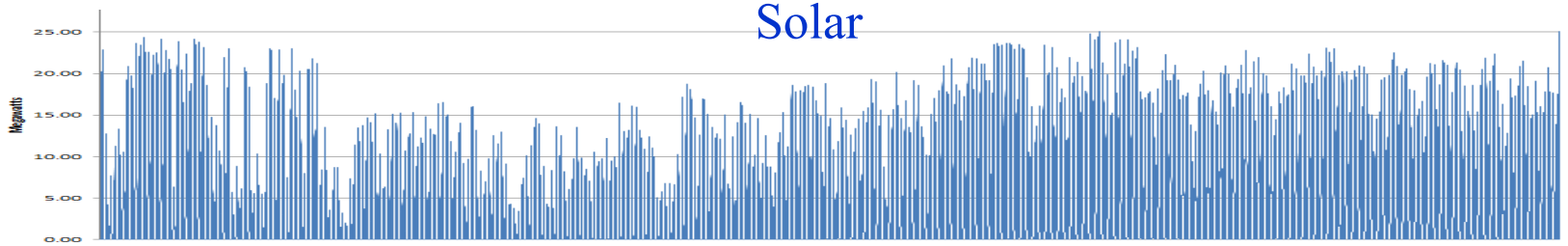
Stage 1: Investment

Wind Capacity
Solar Capacity
Battery Capacity
Fossil Capacity

Find search direction

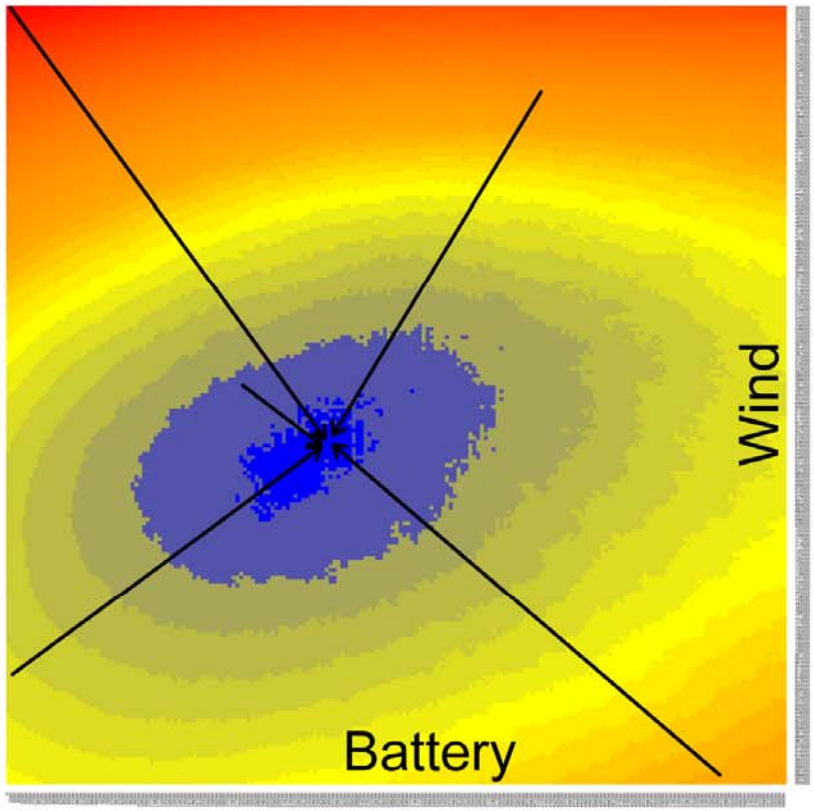
Hour 1 *Hour 2* *Hour 3* … *Hr 8758* *Hr 8759* *Hr 8760*



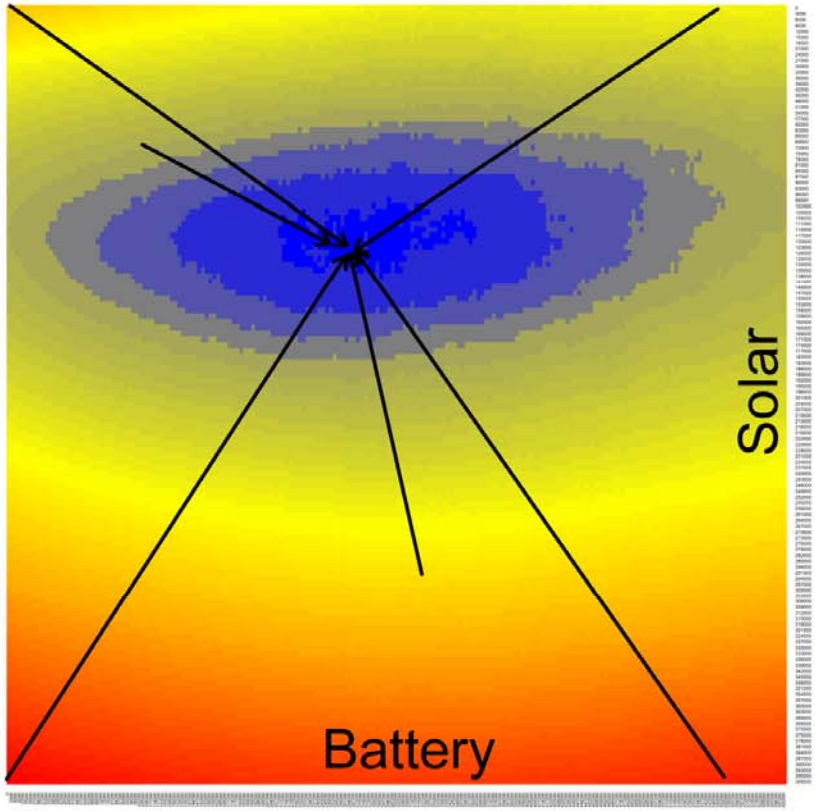


The search algorithm

- Results
  - » The stochastic search algorithm reliably finds the optimum from different starting points.



(a) Solar capacity is fixed.



(b) Wind capacity is fixed.

Different Starting Points

The search algorithm

- Empirical performance of algorithm
  - » Starting the algorithm from different starting points appears to reliably find the optimal (determined using a full grid search).
  - » Algorithm tended to require < 15 iterations:
    - Each iteration required 4-5 simulations to compute the complete gradient
    - Required 1-8 evaluations to find the best stepsize
    - Worst case number of function evaluations is 15 x (8+5) = 195.
    - Budischak paper required "28 billion" for full enumeration (used super computer)
  - » Run times
    - Aggregated supply stack: ~100 minutes
    - Full supply stack: ~100 hours

SMART-Invest

- Policy search – Optimizing reserve parameter

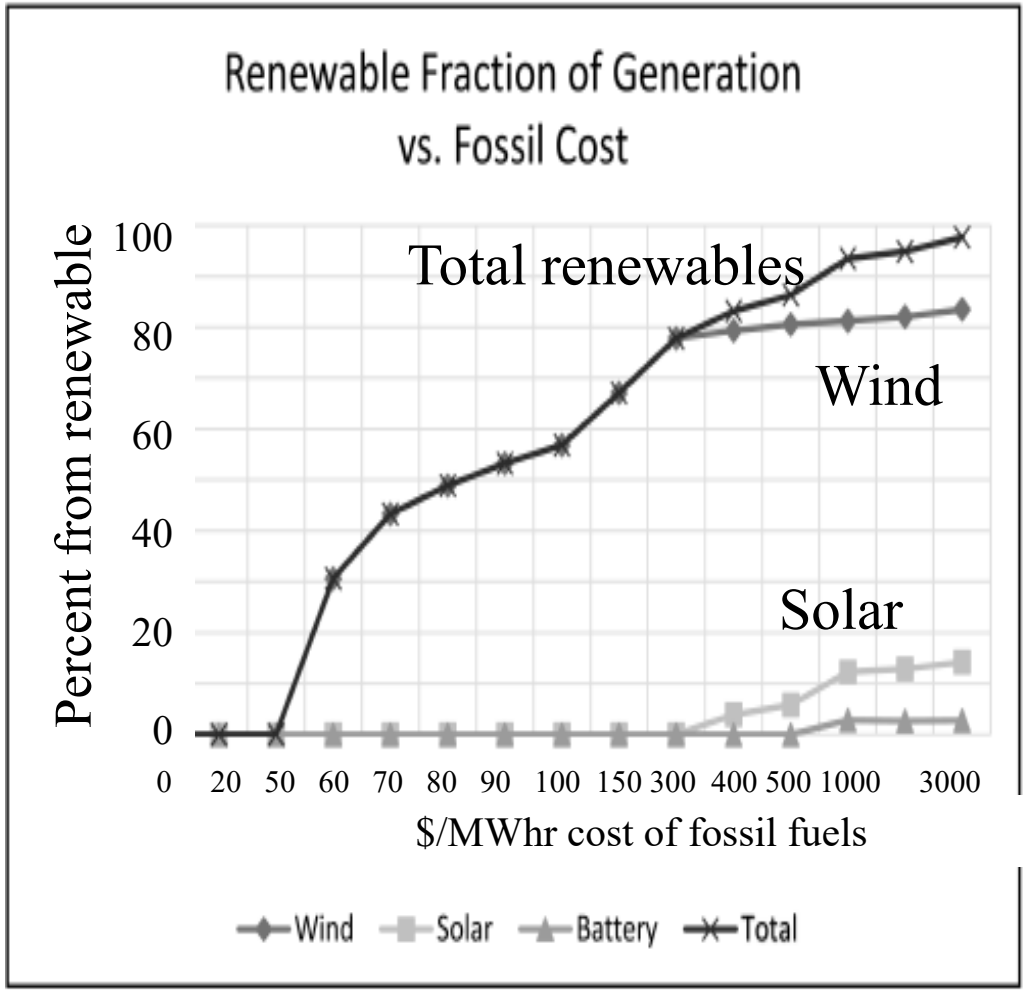
CO_2 tax \$/tonne	Wind MW	Solar MW	Batt. MWh	Wind Gen. (%)	Opt. θ
0	0	0	0	0.0	1.000
50	51284	170	19738	19.0	0.985
80	82066	3741	4556	29.8	0.998
100	80250	6584	6005	29.2	1.001
150	120251	33372	30939	42.0	1.020
0	0	0	0	0.0	1.000
0	51284	170	19738	18.8	1.191
0	80250	6584	6005	29.2	1.041
0	120251	33372	30939	40.7	1.197
0	120251	33372	30939	40.7	1.197
50	120251	33372	30939	41.0	1.134
80	120251	33372	30939	42.2	1.011
100	120251	33372	30939	42.1	1.011

Low carbon tax, increased usage of slow fossil, requires higher reserve margin ~19 percent

High carbon tax, shift from slow to fast fossil, requires minimal reserve margin ~1 pct

Policy studies

- Renewables as a function of cost of fossil fuels



Cost function approximation

Energy storage example

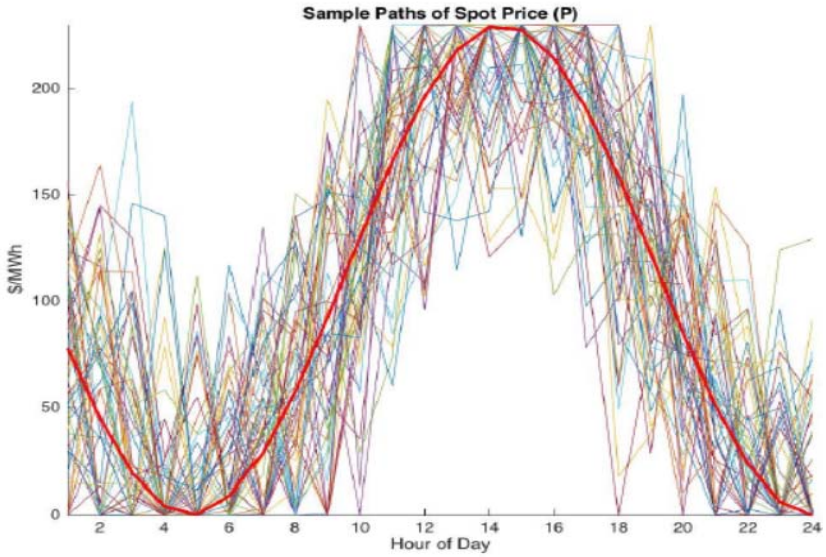
Multi-dimensional parameter space

Energy storage optimization

- Notes:
  - » In this set of slides, we are going to illustrate the use of a parametrically modified lookahead policy.
  - » This is designed to handle a nonstationary problem with rolling forecasts. This is just what you do when you plan your path with google maps.
  - » The lookahead policy is modified with a set of parameters that factor the forecasts. These parameters have to be tuned using the basic methods of policy search.

Energy storage optimization

- An energy storage problem:



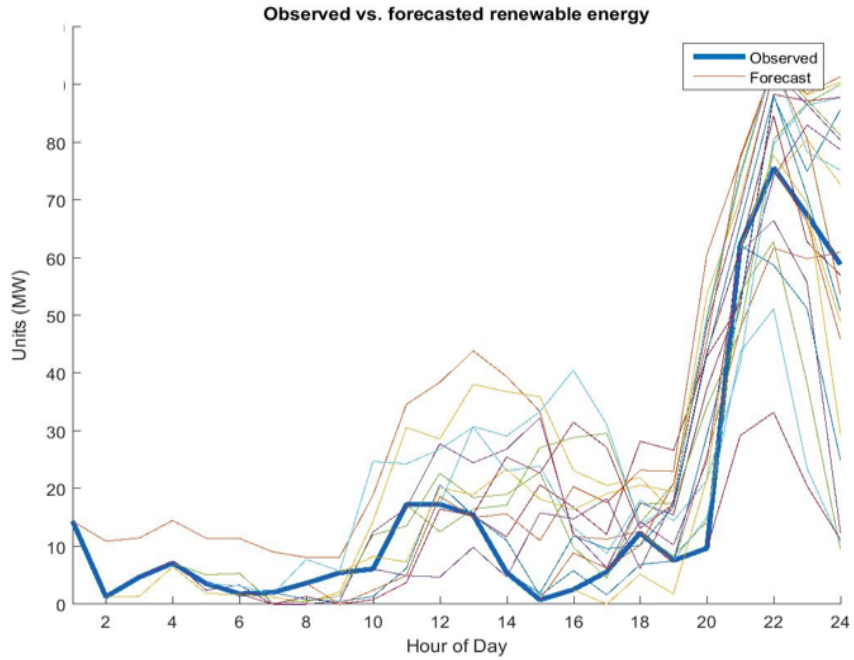
The state of the system can be represented by

r,

$$S_t = (R_t, E_t, P_t, D_t, G_t)$$

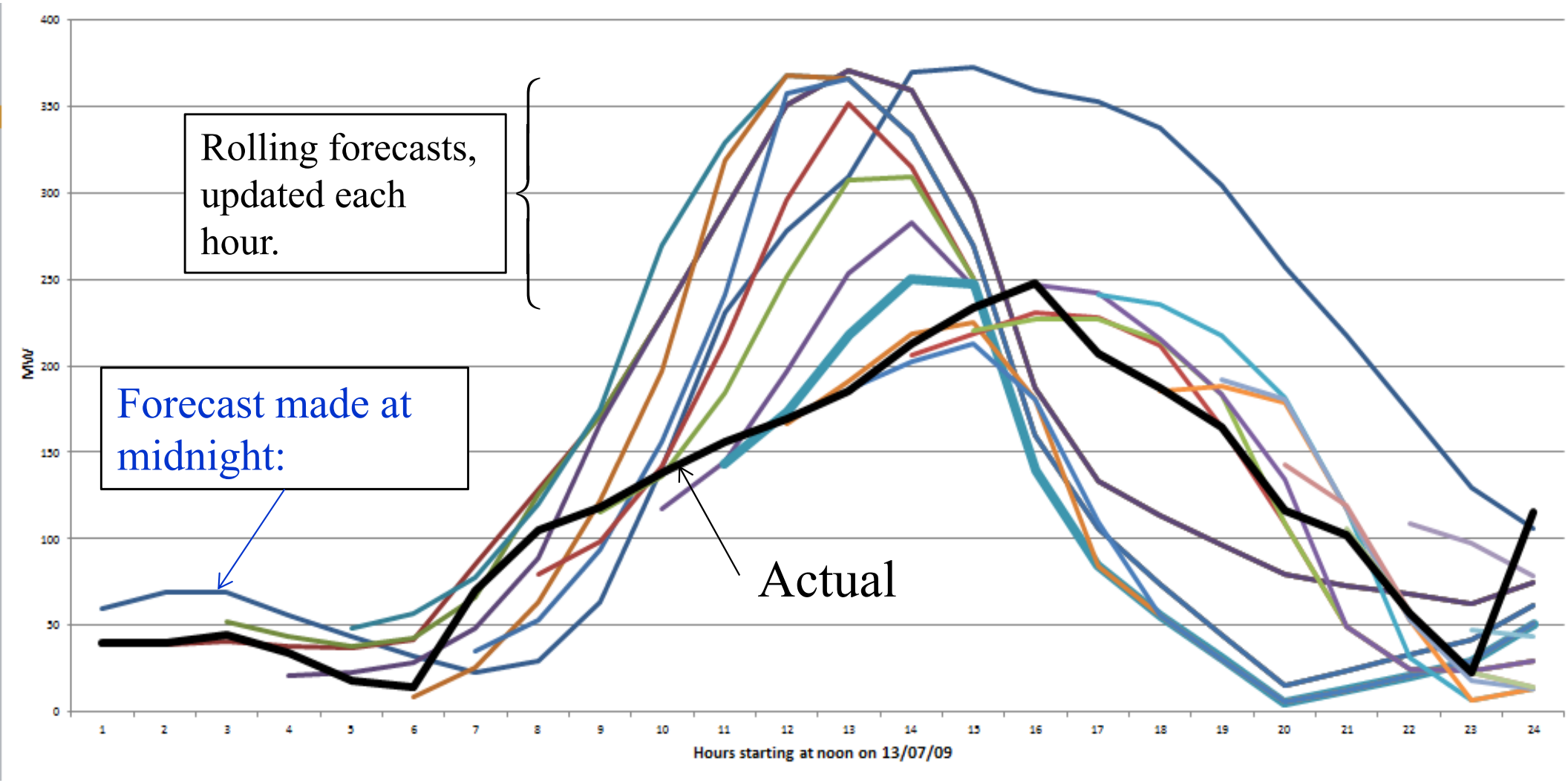
where

- $R_t \in [0, R_{\max}]$ is the level of energy in storage at time t
- E_t is the amount of energy available from wind
- P_t is the spot price of electricity
- D_t is the power demand
- G_t is the energy available from the grid



Energy storage optimization

- Forecasts evolve over time as new information arrives:



Energy storage optimization

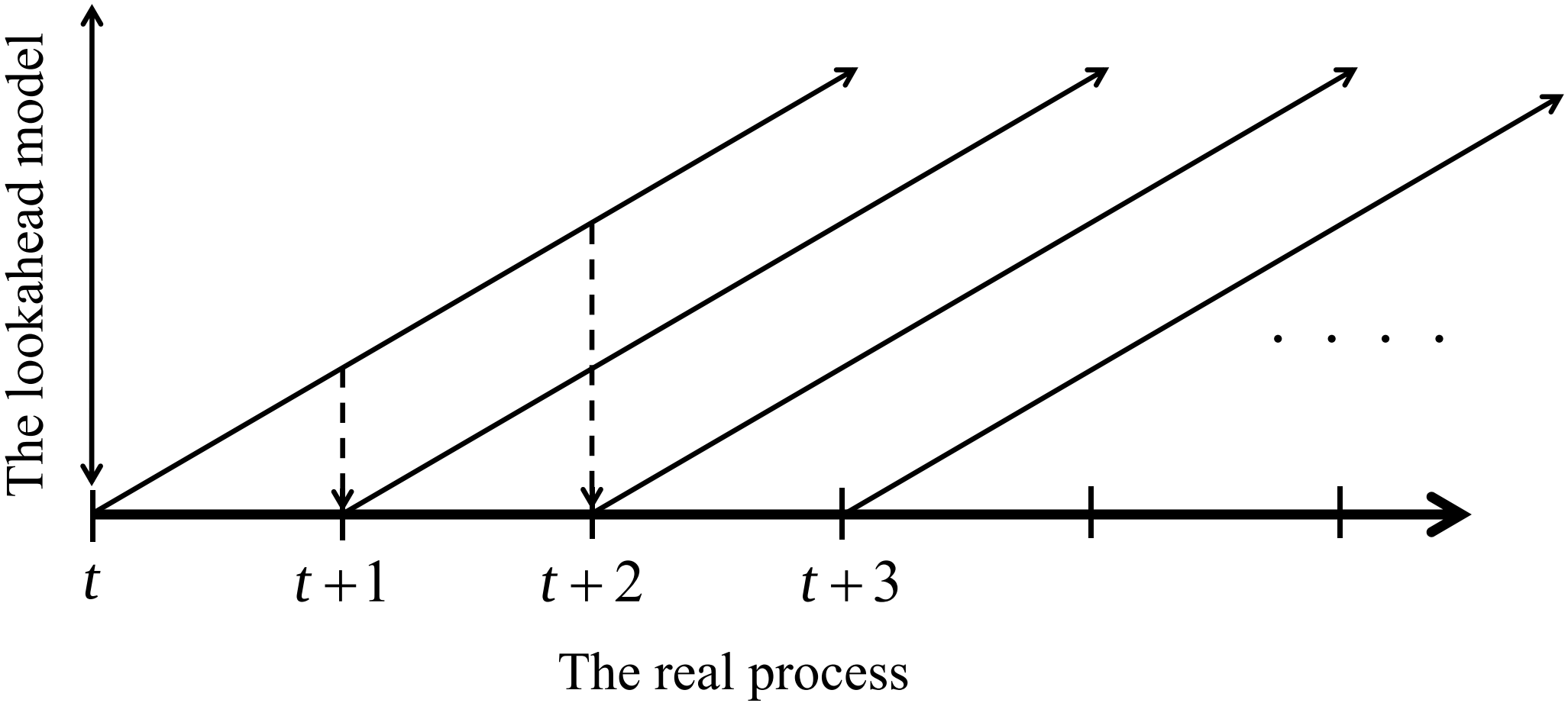
- Benchmark policy – Deterministic lookahead

$$X_t^{\text{D-LA}}(S_t) = \underset{x_t,(\tilde{x}_{tt'},t'=t+1,\ldots,t+H)}{\text{argmin}} \left(C(S_t,x_t) + \left[\sum_{t'=t+1}^{t+H} \tilde{c}_{tt'}\tilde{x}_{tt'} \right] \right)$$

$$\begin{aligned}
\tilde{x}_{tt'}^{wd} + \beta\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{gd} &\le f_{tt'}^{D} \\
\tilde{x}_{tt'}^{gd} + \tilde{x}_{tt'}^{gr} &\le f_{tt'}^{G} \\
\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{rg} &\le \tilde{R}_{tt'} \\
\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{gr} &\le R^{\max} - \tilde{R}_{tt'} \\
\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{wd} &\le f_{tt'}^{E} \\
\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{gr} &\le \gamma^{charge} \\
\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{rg} &\le \gamma^{discharge}
\end{aligned}$$

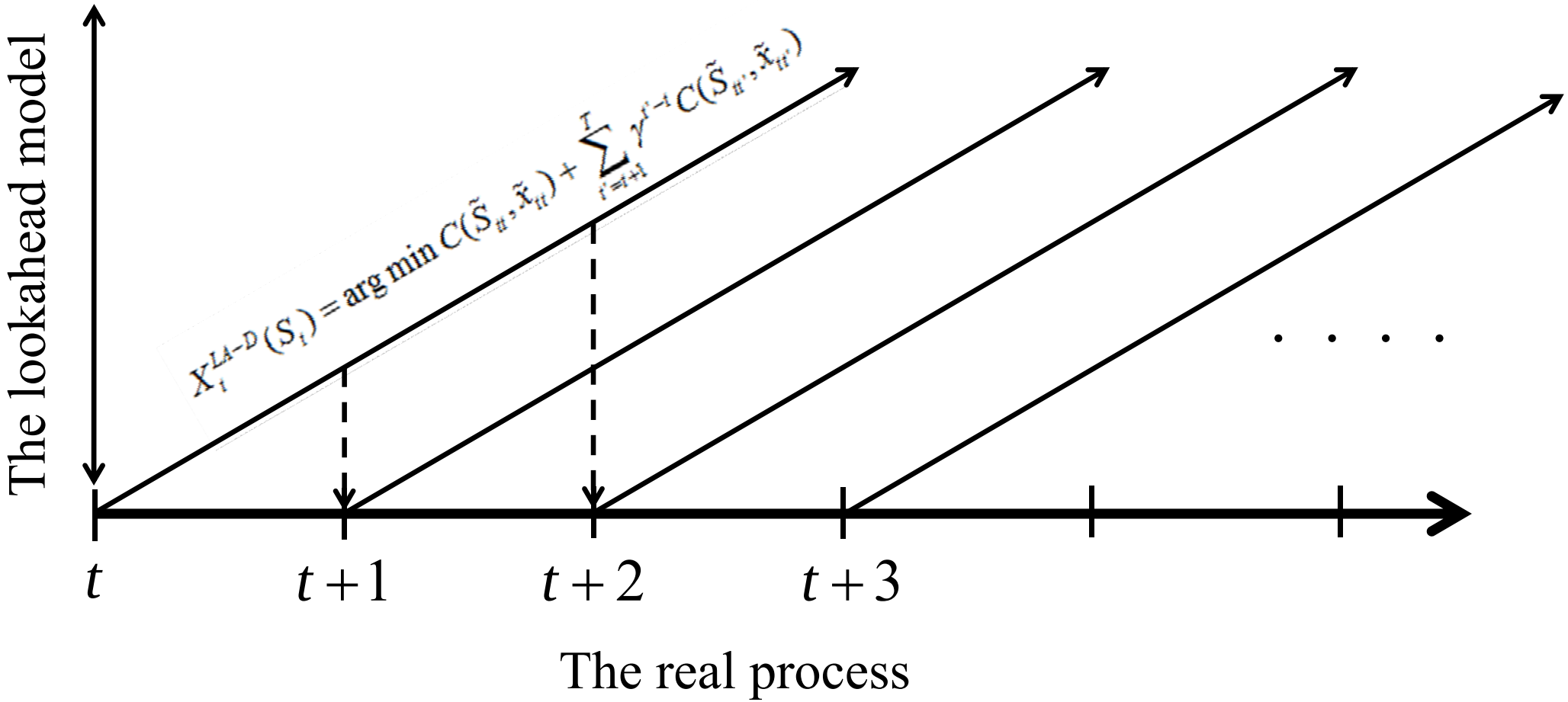
Energy storage optimization

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



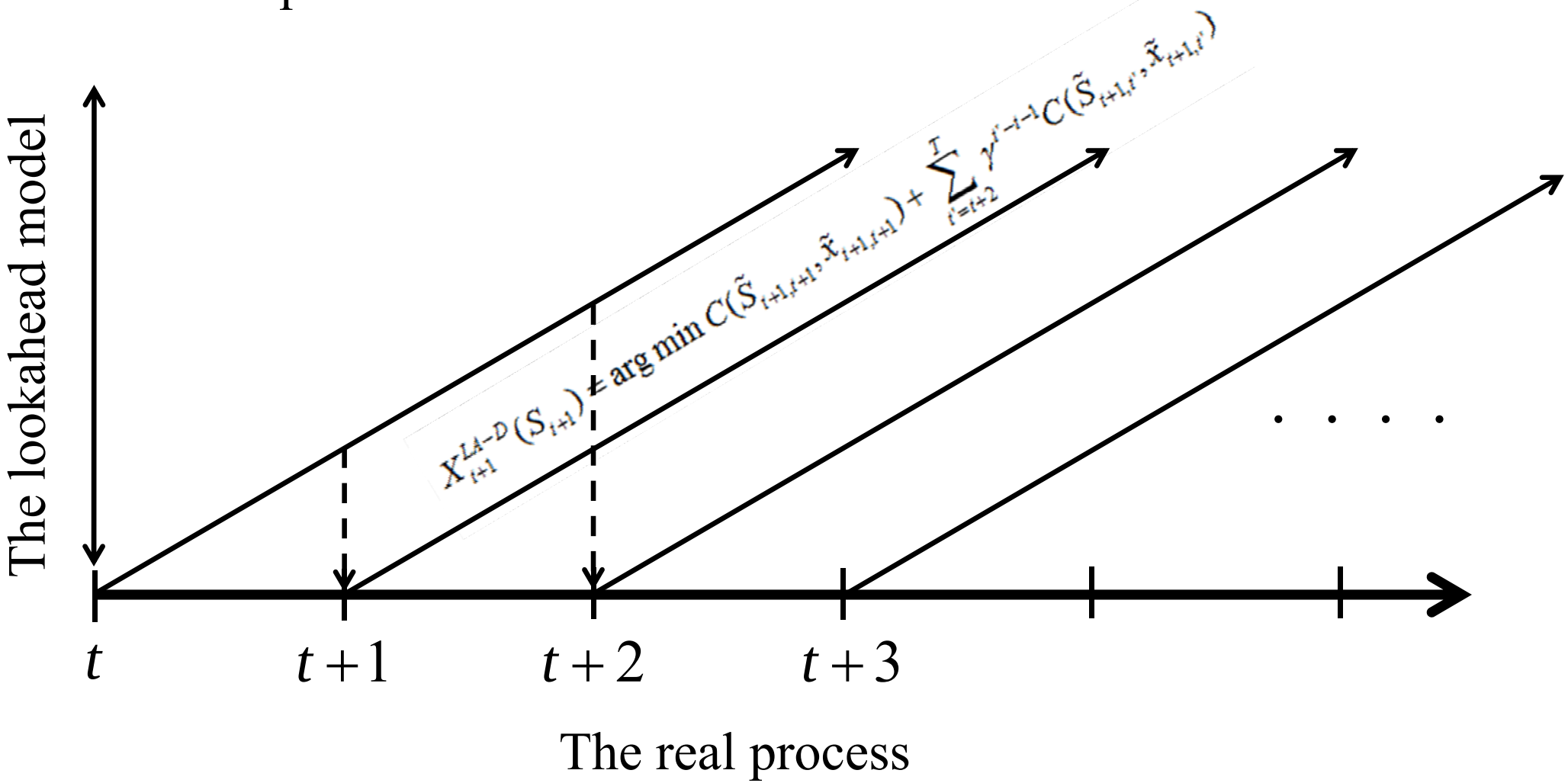
Energy storage optimization

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



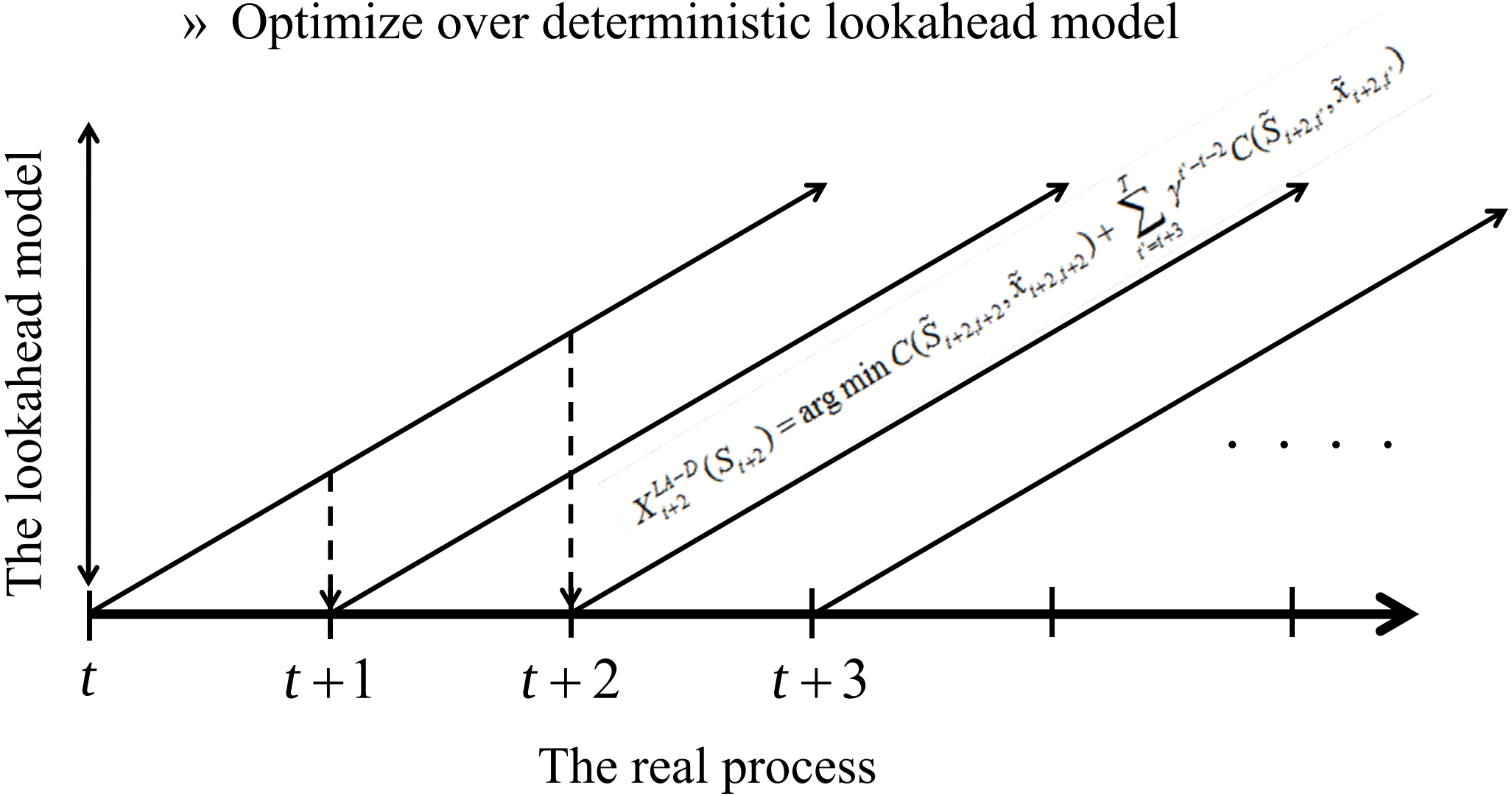
Energy storage optimization

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



Energy storage optimization

- Lookahead policies peek into the future
  - » Optimize over deterministic lookahead model



Energy storage optimization

- Benchmark policy – Deterministic lookahead

$$X_t^{\text{D-LA}}(S_t) = \underset{x_t,(\tilde{x}_{tt'},t'=t+1,\ldots,t+H)}{\text{argmin}} \left(C(S_t,x_t) + \left[\sum_{t'=t+1}^{t+H} \tilde{c}_{tt'}\tilde{x}_{tt'} \right] \right)$$

$$\tilde{x}_{tt'}^{wd} + \beta\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{gd} \le f_{tt'}^{D}$$

$$\tilde{x}_{tt'}^{gd} + \tilde{x}_{tt'}^{gr} \le f_{tt'}^{G}$$

$$\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{rg} \le \tilde{R}_{tt'}$$

$$\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{gr} \le R^{\max} - \tilde{R}_{tt'}$$

$$\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{wd} \le f_{tt'}^{E}$$

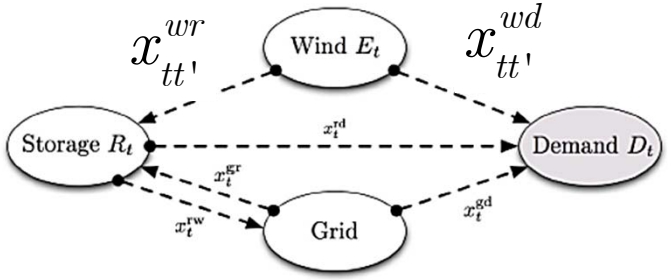
$$\tilde{x}_{tt'}^{wr} + \tilde{x}_{tt'}^{gr} \le \gamma^{charge}$$

$$\tilde{x}_{tt'}^{rd} + \tilde{x}_{tt'}^{rg} \le \gamma^{discharge}$$

Energy storage optimization

- 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):

$$x^{wr}_{tt'} + x^{wd}_{tt'} \leq \theta_{t'-t} f^{E}_{tt'}$$

  - » We can simulate the performance of a parameterized policy

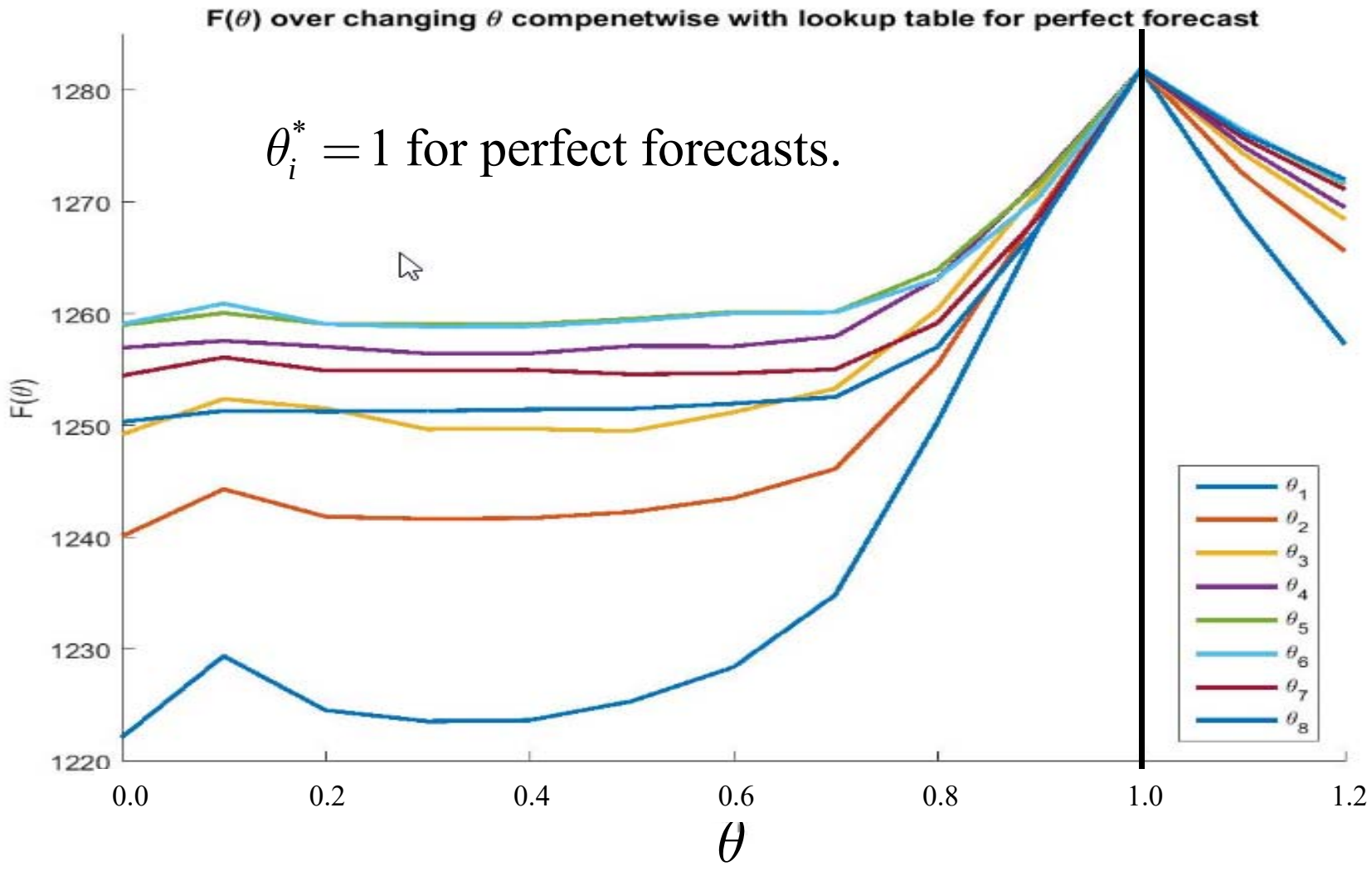
$$\bar{F}(\theta, \omega) = \sum_{t=0}^{T} C\left(S_t(\omega), X^{\pi}_t(S_t(\omega) \mid \theta)\right)$$

  - » The challenge is to optimize the parameters:

$$\min_{\theta} \mathbb{E} \sum_{t=0}^{T} C\left(S_t, X^{\pi}_t(S_t \mid \theta)\right)$$

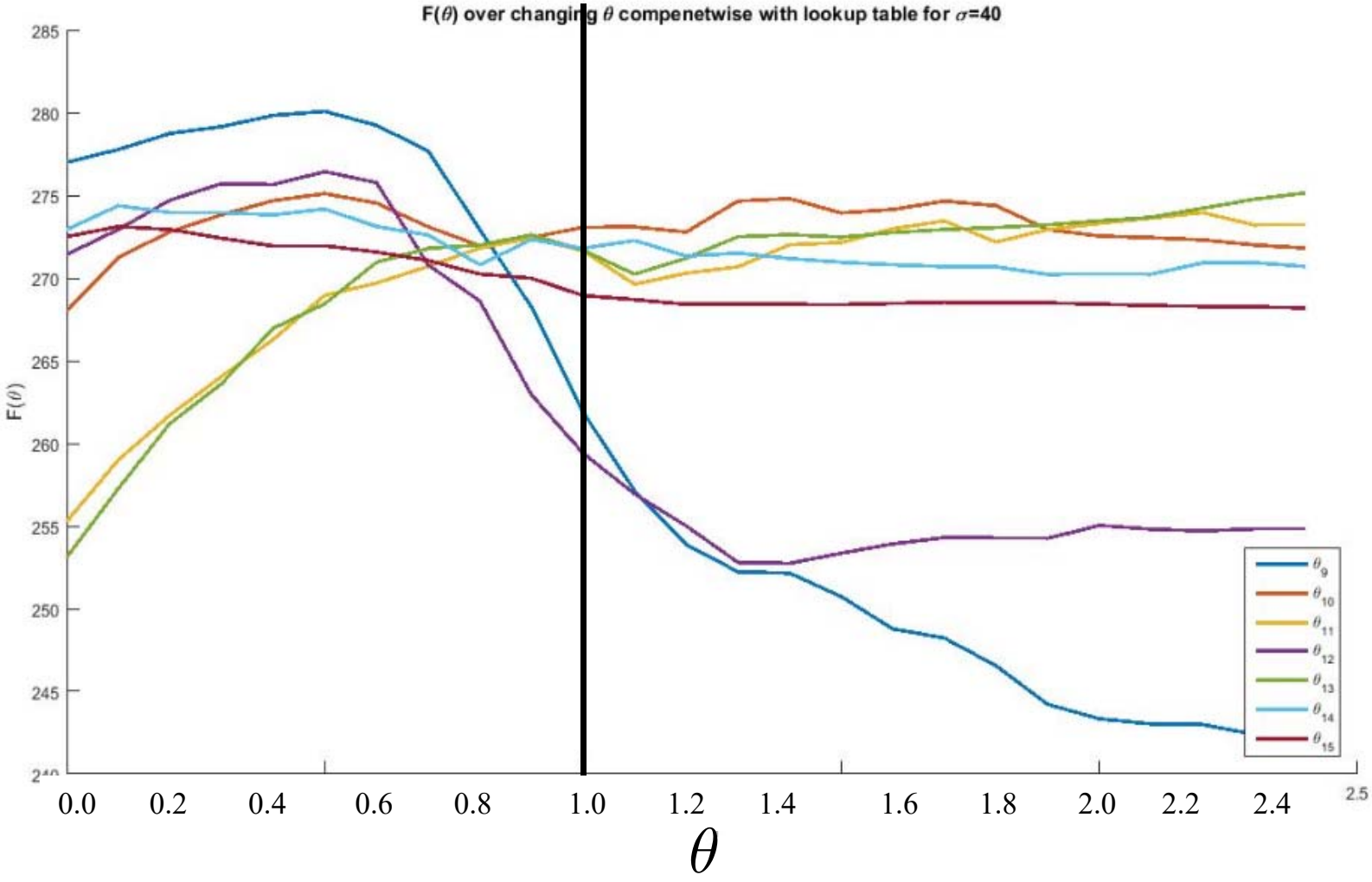
Energy storage optimization

- One-dimensional contour plots – perfect forecast
  - » θ_i for i=1,…, 8 hours into the future.

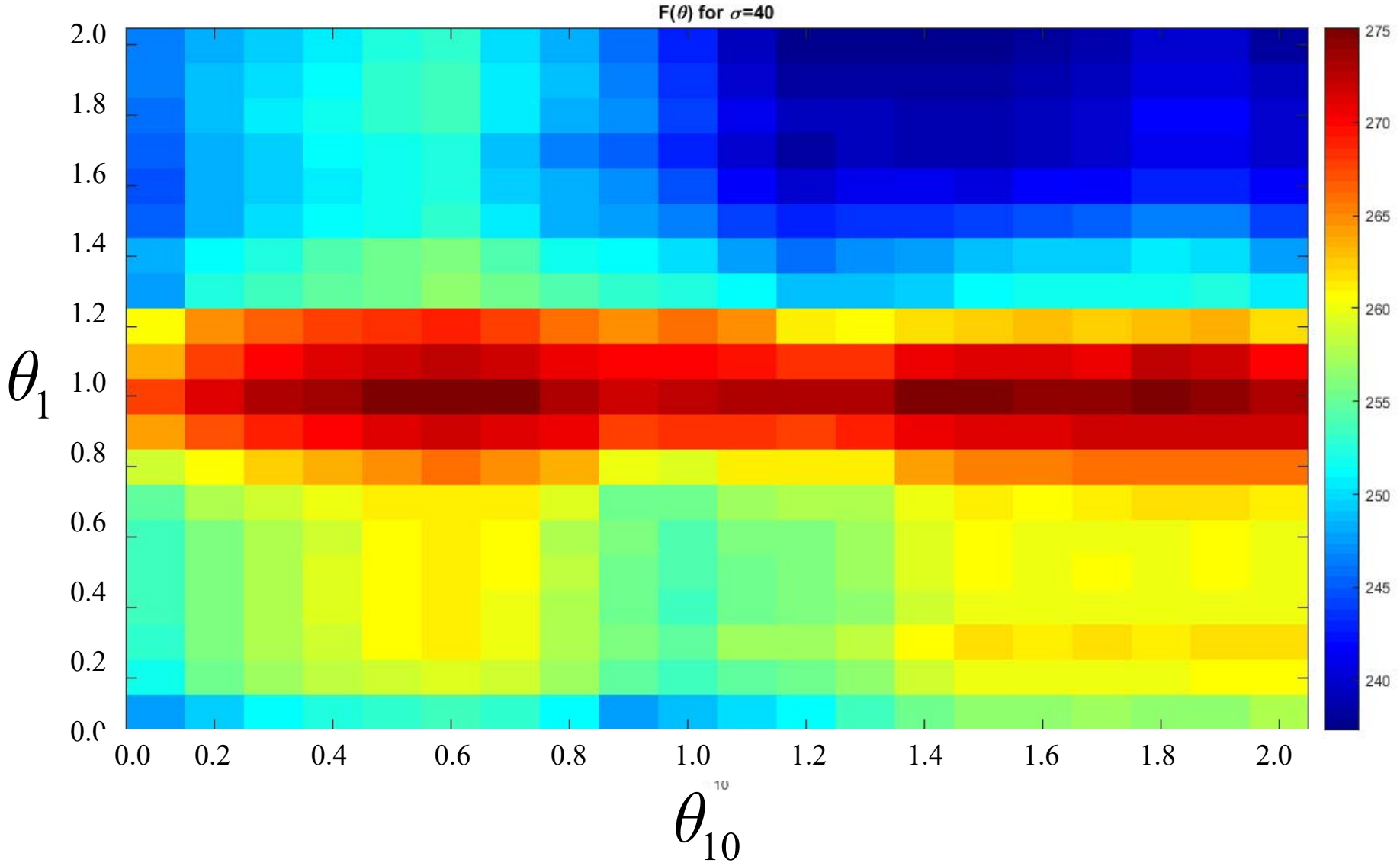


Energy storage optimization

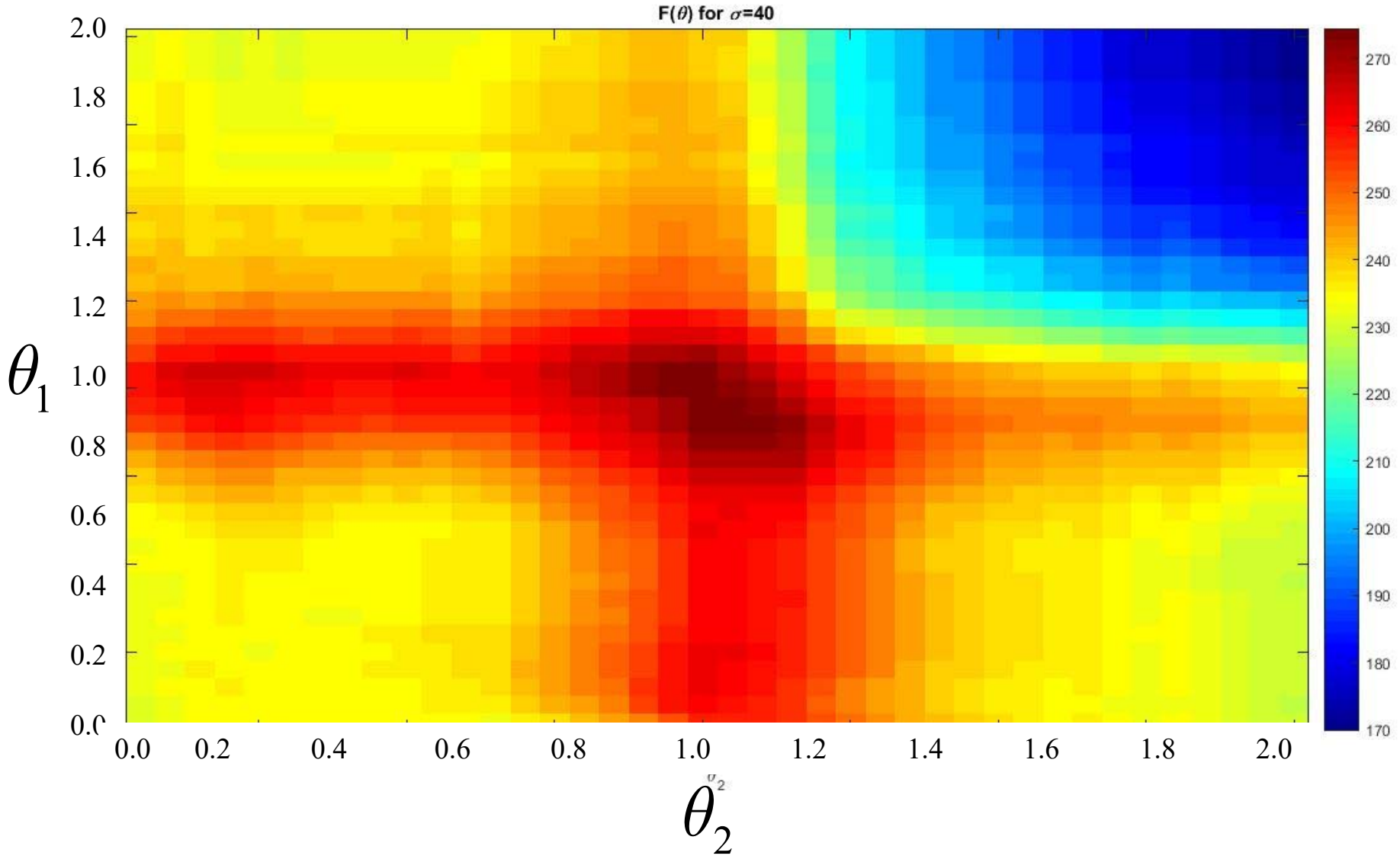
- One-dimensional contour plots-uncertain forecast
  - » θ_i for i=9,…, 15 hours into the future



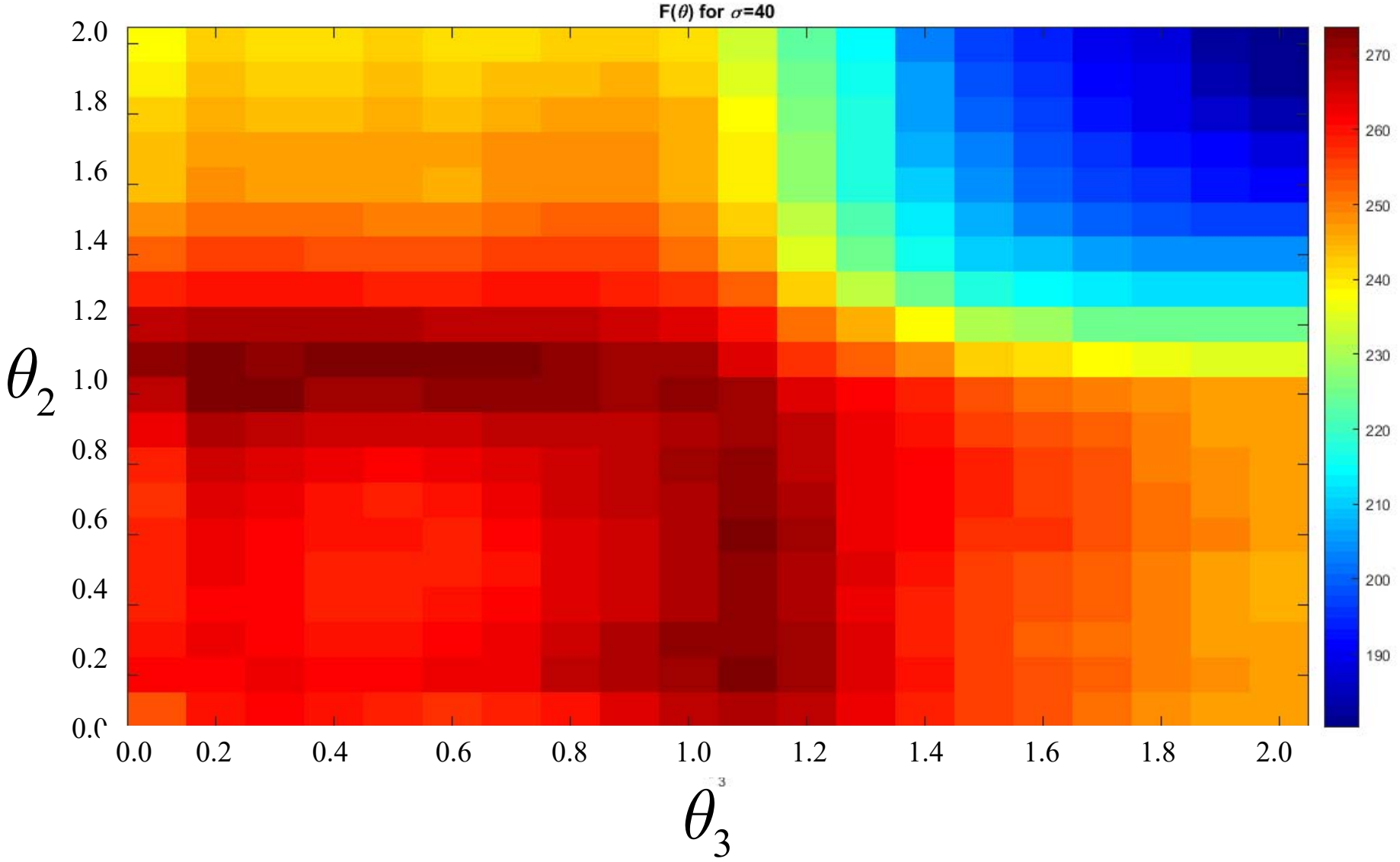
Energy storage optimization



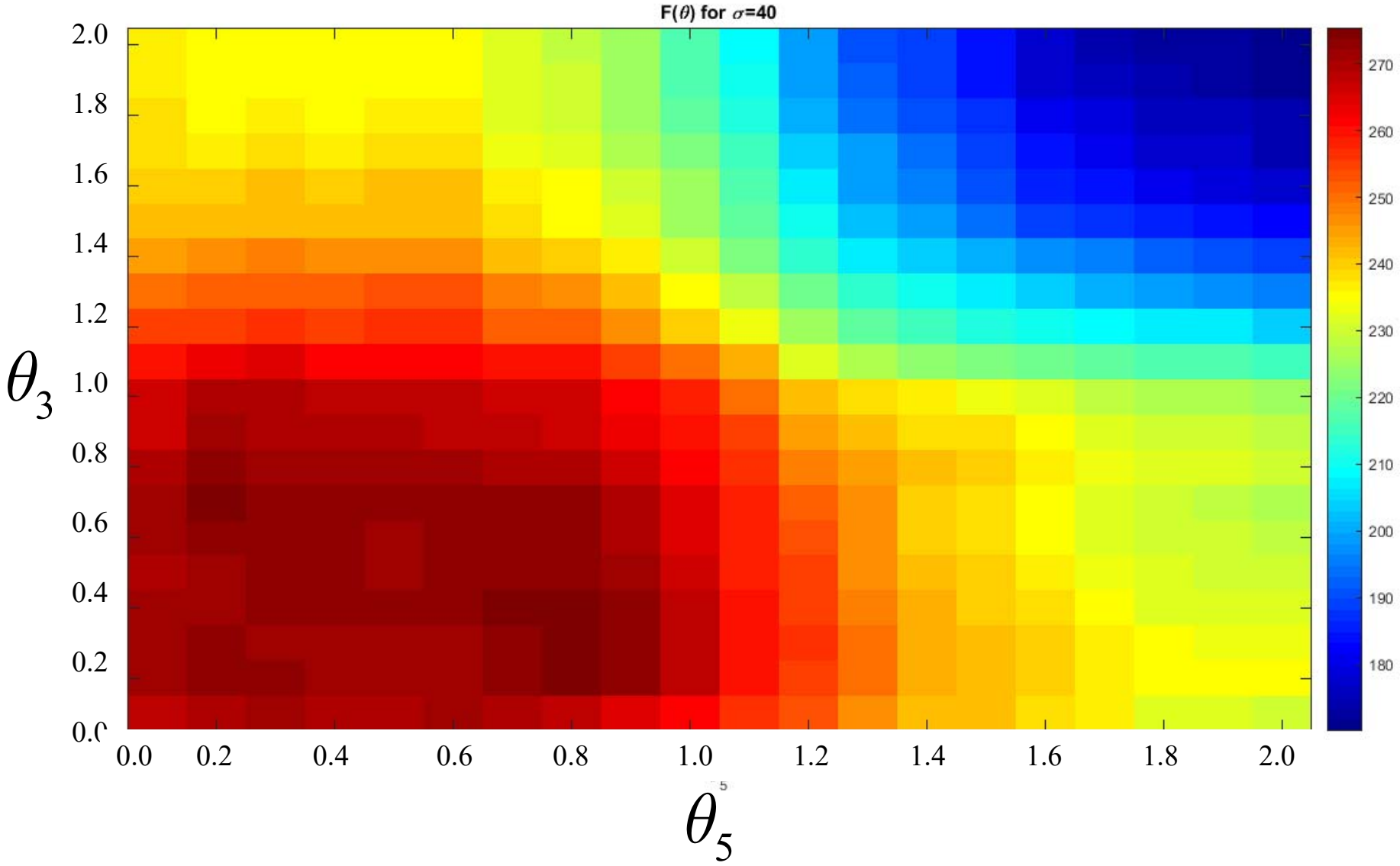
Energy storage optimization



Energy storage optimization



Energy storage optimization



Energy storage optimization

- Simultaneous perturbation stochastic approximation
  - » Let:
    - x^n be a p – dimensional vector.
    - δ^n be a scalar perturbation
    - Z^n be a p –dimensional vector, with each element drawn from a normal (0,1) distribution.
  - » We can obtain a sampled estimate of the gradient $\nabla_x F(x^n, W^{n+1})$ using two function evaluations: $F(x^n + \delta^n Z^n)$ and $F(x^n - \delta^n Z^n)$

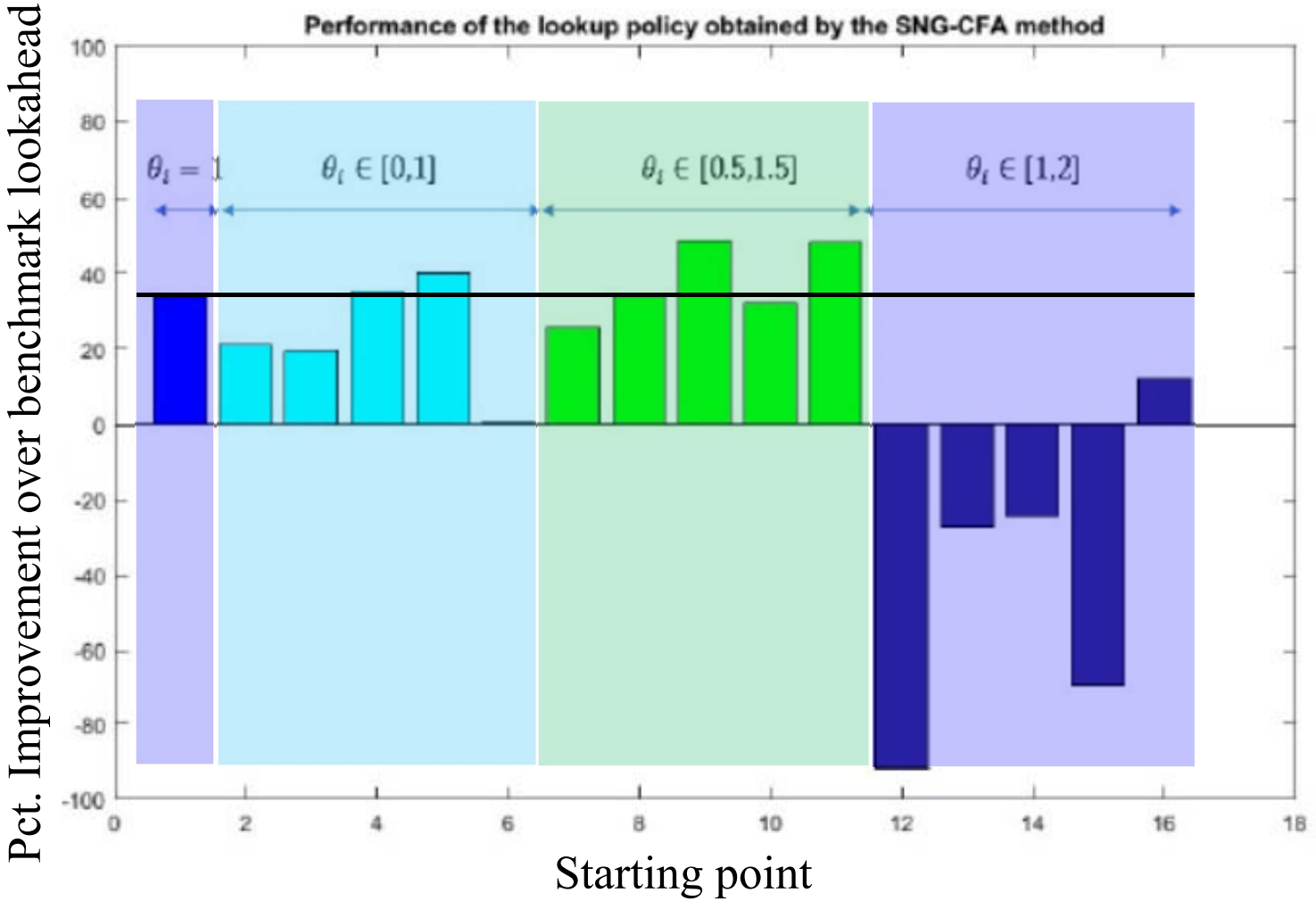
$$\nabla_x F(x^n, W^{n+1}) = \begin{bmatrix} \dfrac{F(x^n + \delta^n Z^n) - F(x^n - \delta^n Z^n)}{2\delta^n Z_1^n} \\ \dfrac{F(x^n + \delta^n Z^n) - F(x^n - \delta^n Z^n)}{2\delta^n Z_2^n} \\ \vdots \\ \dfrac{F(x^n + \delta^n Z^n) - F(x^n - \delta^n Z^n)}{2\delta^n Z_p^n} \end{bmatrix}$$

Energy storage optimization

- Notes:
  - » The next three slides represent applications of the SPSA algorithm with different stepsize rules.
  - » These illustrate the importance of tuning stepsizes (or more generally, stepsize policies).
  - » In fact, it is necessary to tune the stepsize depending on the starting point (or at least the starting region).
  - » These were done with RMSProp.

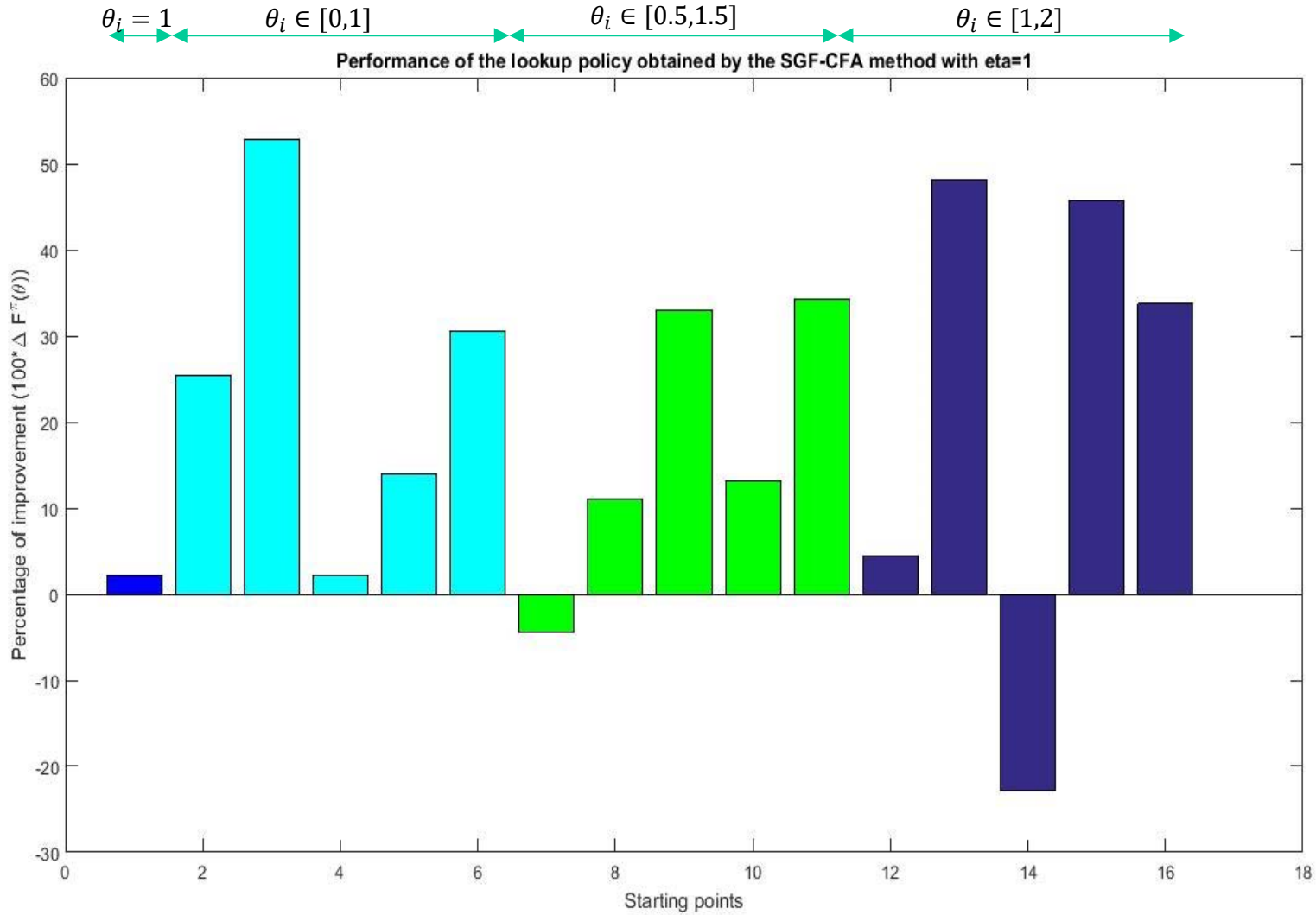
Energy storage optimization

- Effect of starting points
  - » Using a single starting point $\theta = 1$ is fairly reliable.



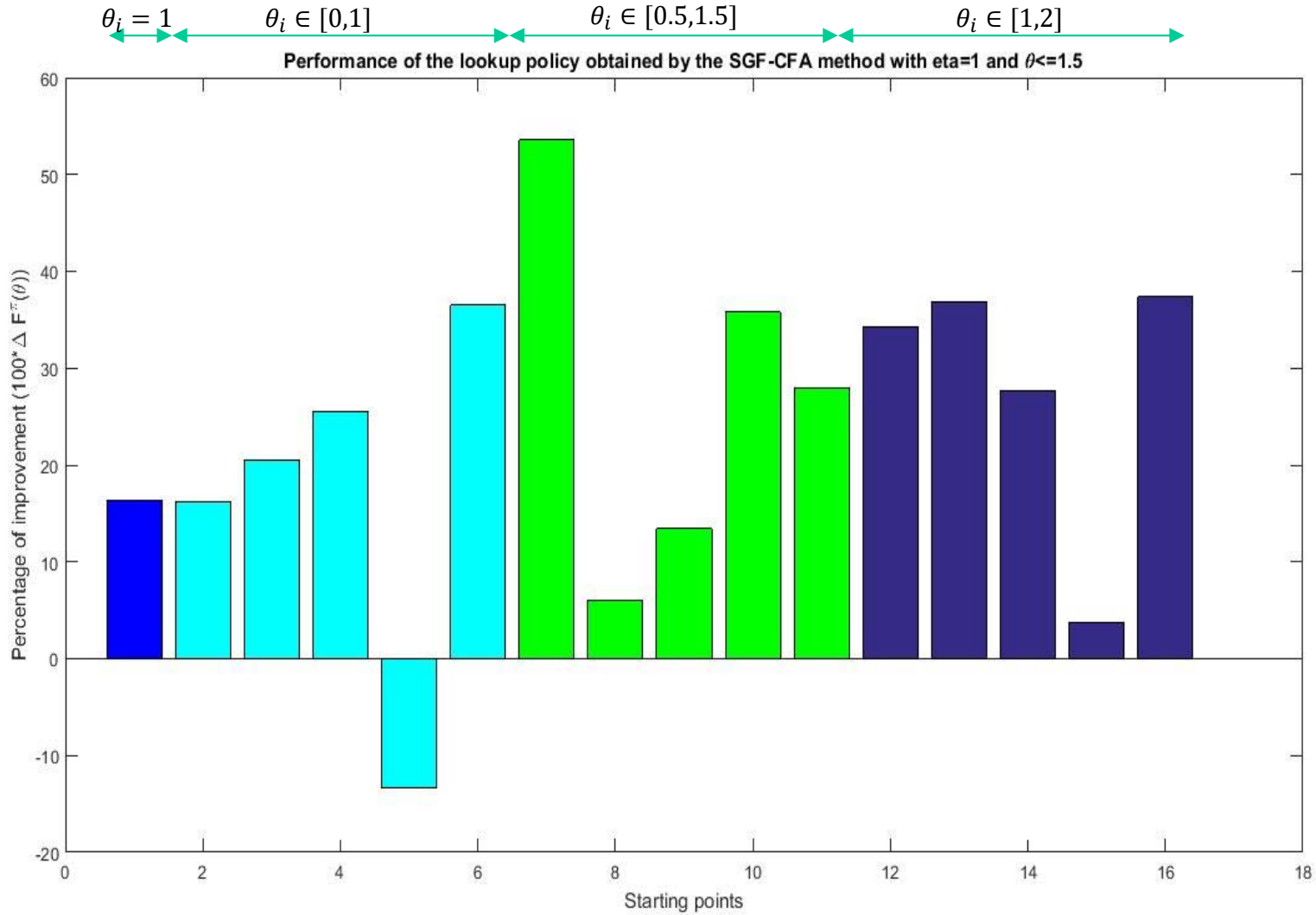
Energy storage optimization

- Tuning the parameters



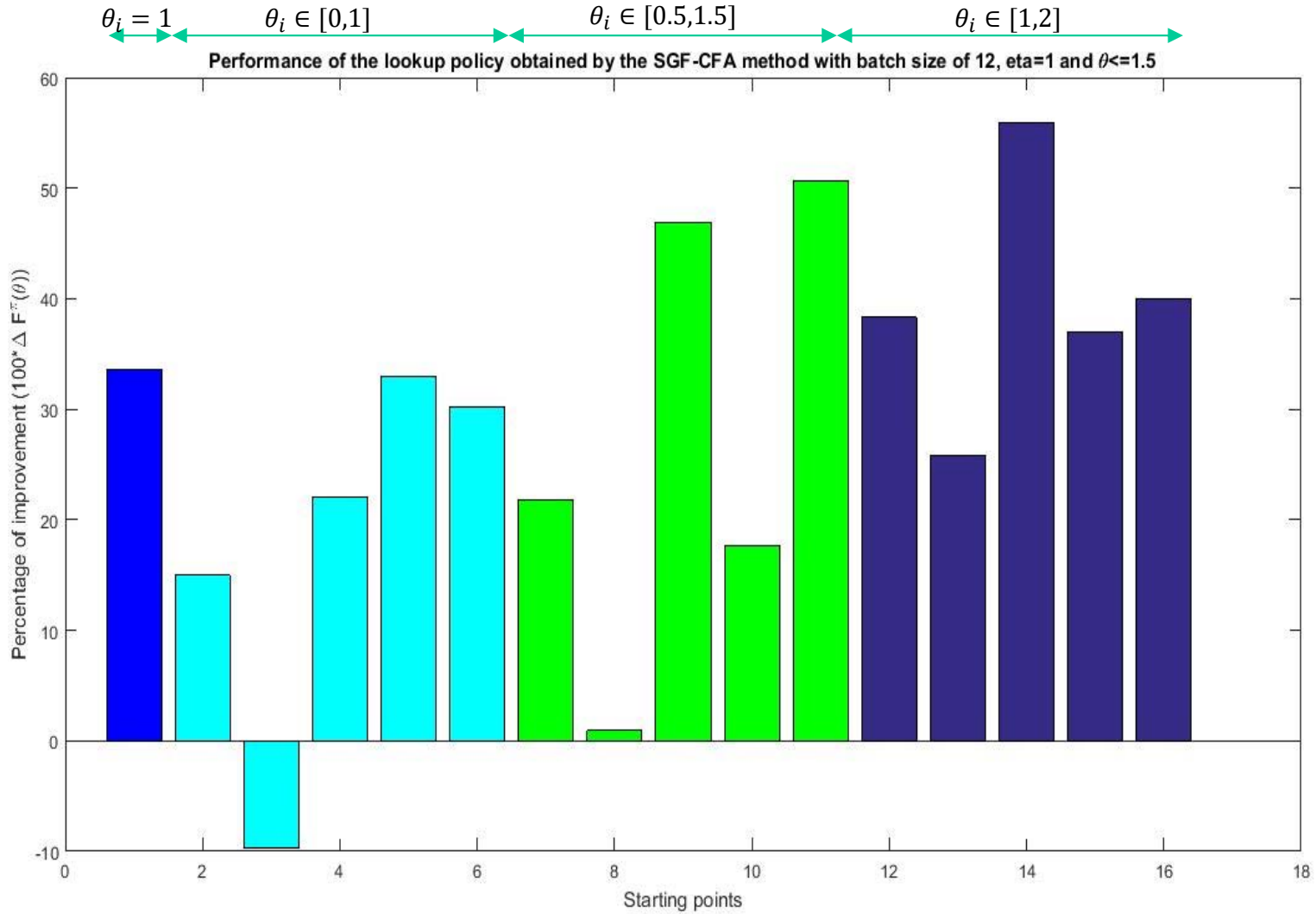
Energy storage optimization

- Tuning the parameters



Energy storage optimization

- Tuning the parameters

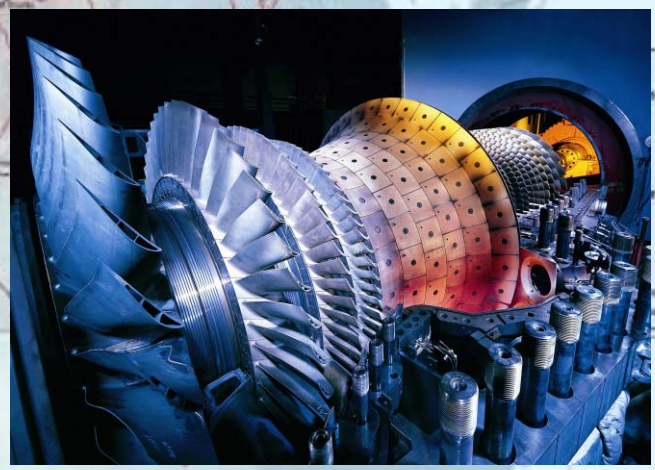


Energy storage optimization

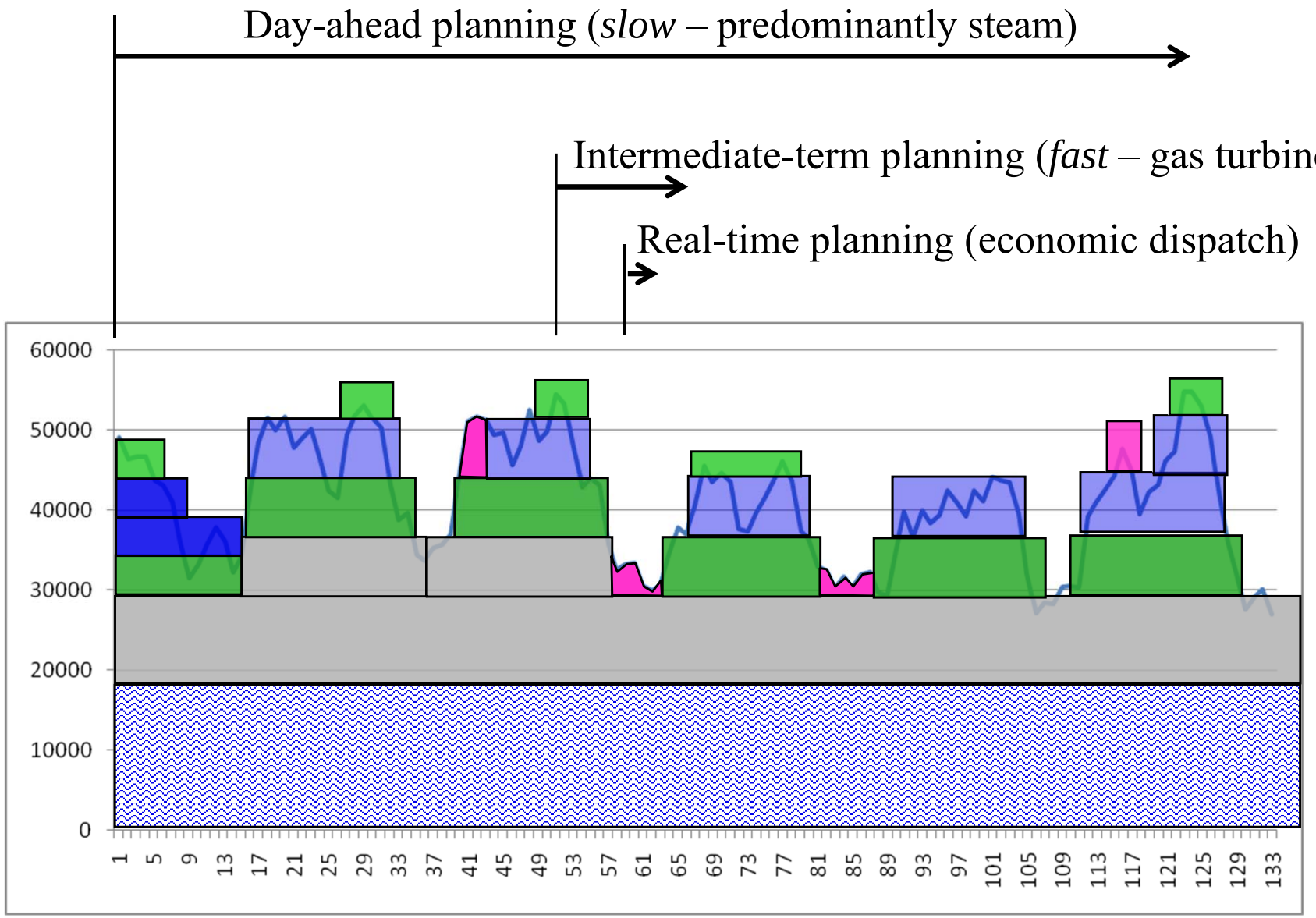
- Notes:
  - » Stepsize policies are extremely random.
  - » While there are rate of convergence results, convergence is very random.
- Research directions:
  - » Performing one-dimensional search using a "noisy Fibonacci search"
  - » Using a homotopy method:
    - Exploits the fact that $\theta = 1$ when the noise in the forecast error = zero.
    - Parameterize the noise by $\eta\sigma^2$ and vary η from 0 to 1.
    - Try to find $\theta^*(\eta)$ starting with $\theta^*(0) = 1$.

Case application – Stochastic unit commitment

The unit commitment problem

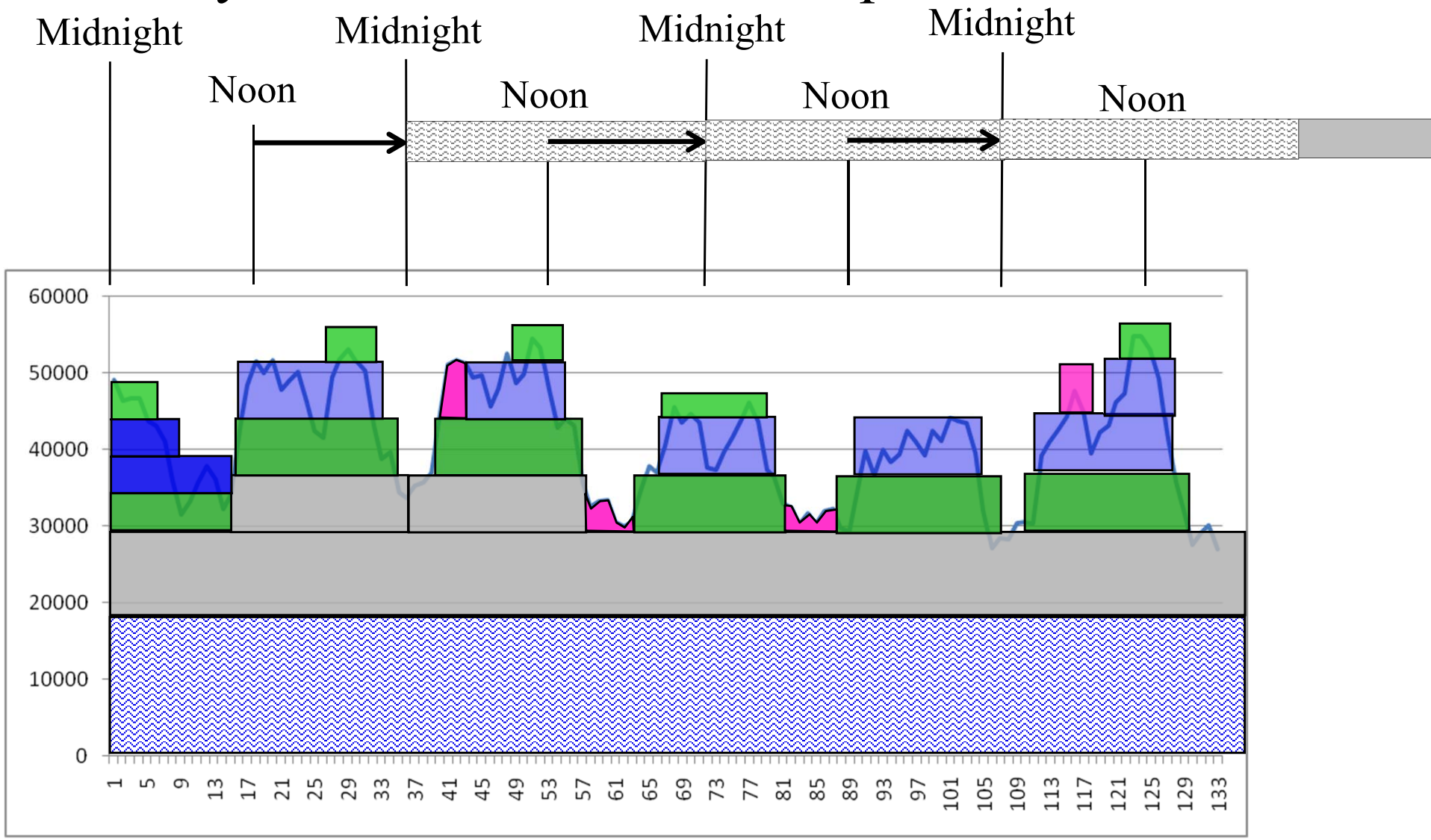


The timing of decisions



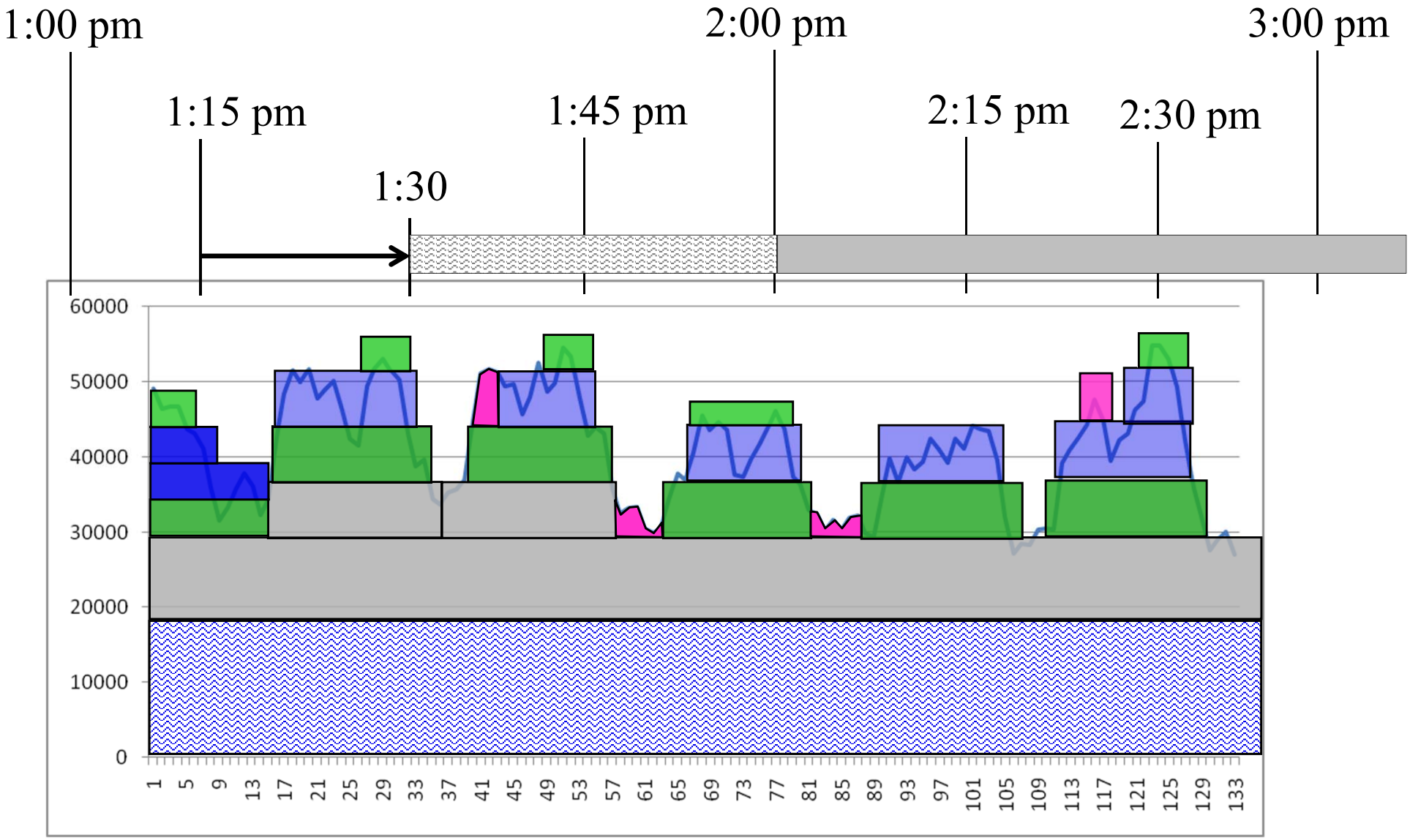
The timing of decisions

- The day-ahead unit commitment problem



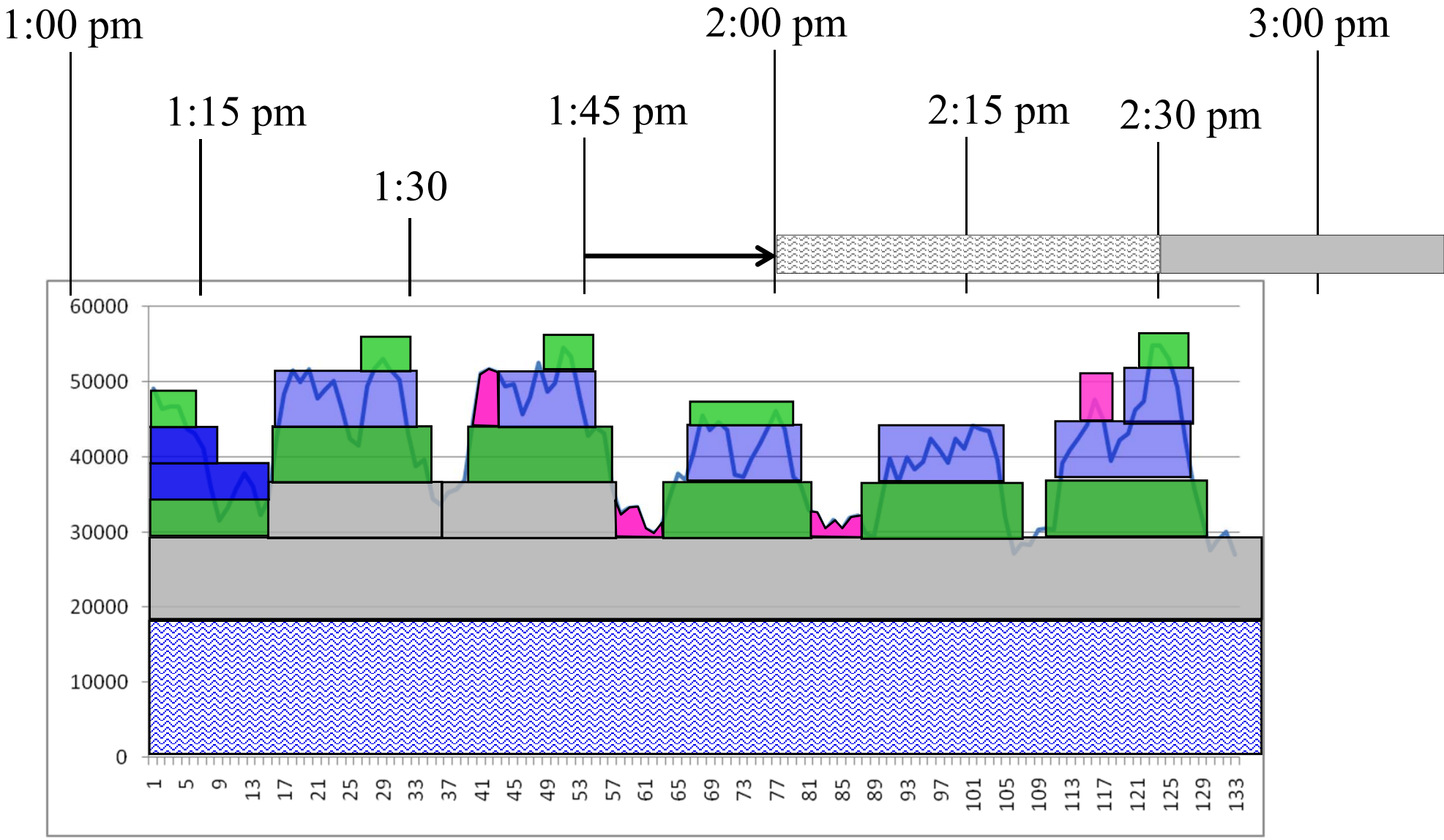
The timing of decisions

- Intermediate-term unit commitment problem



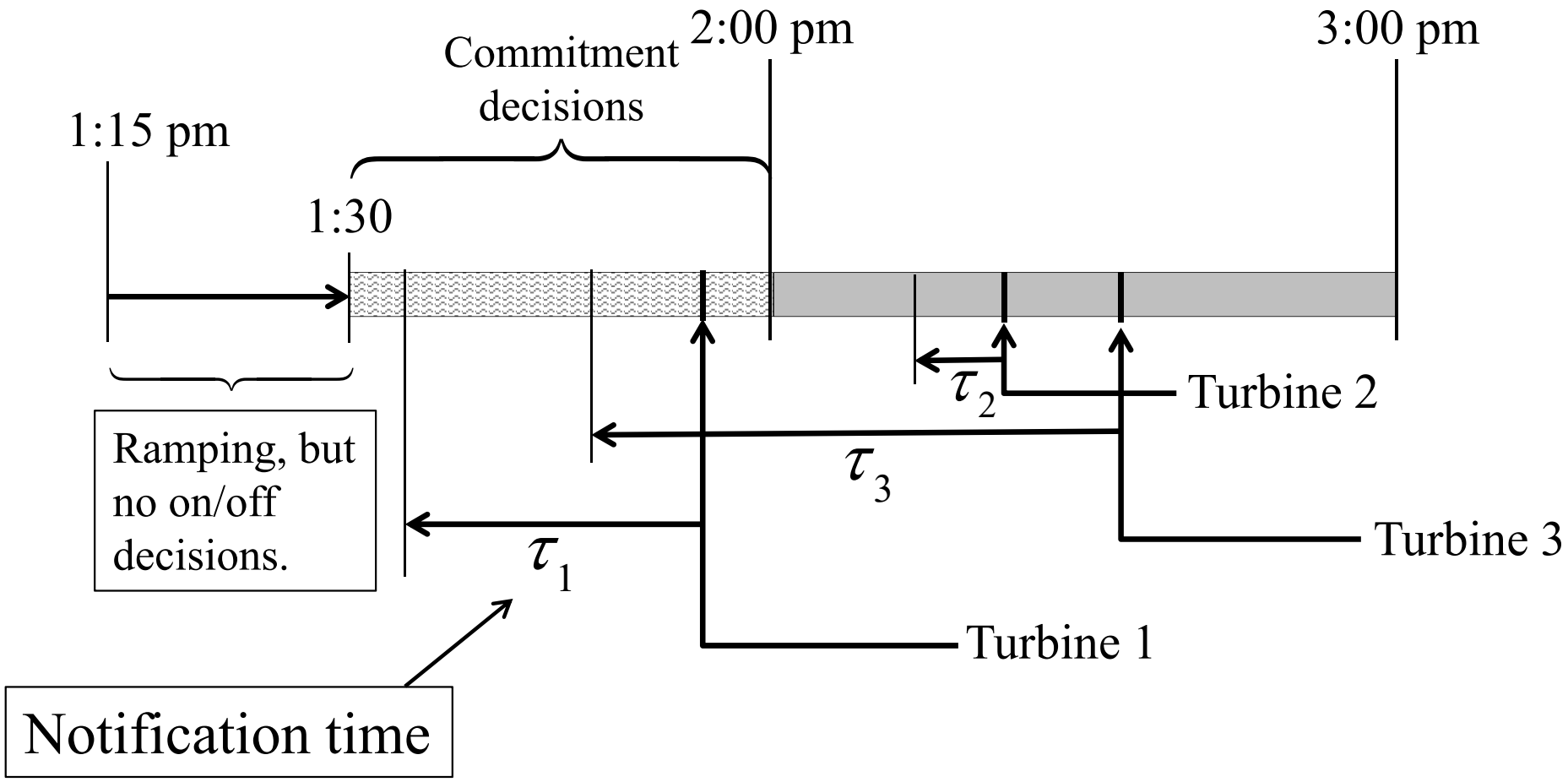
The timing of decisions

- Intermediate-term unit commitment problem



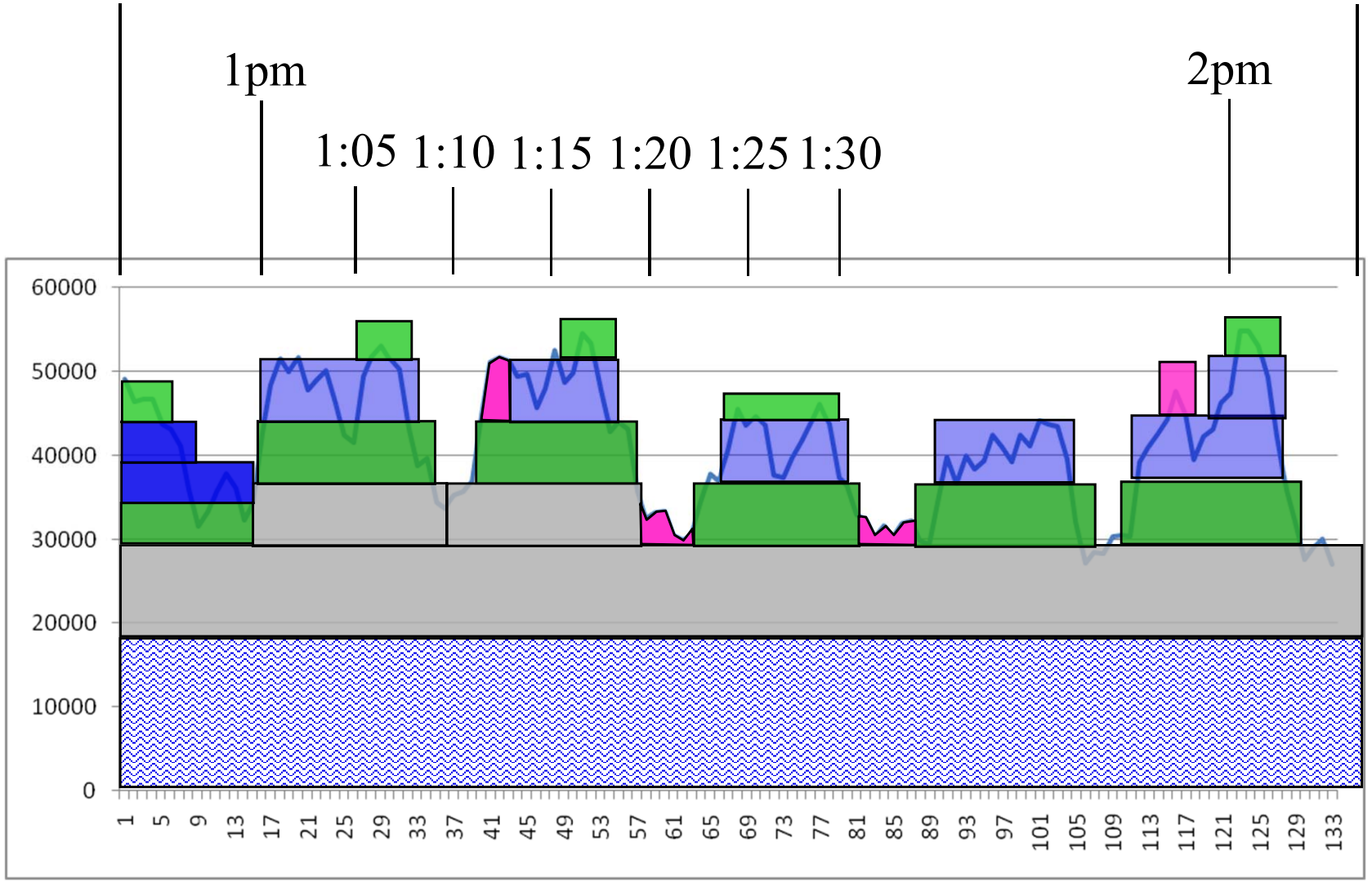
The timing of decisions

- Intermediate-term unit commitment problem



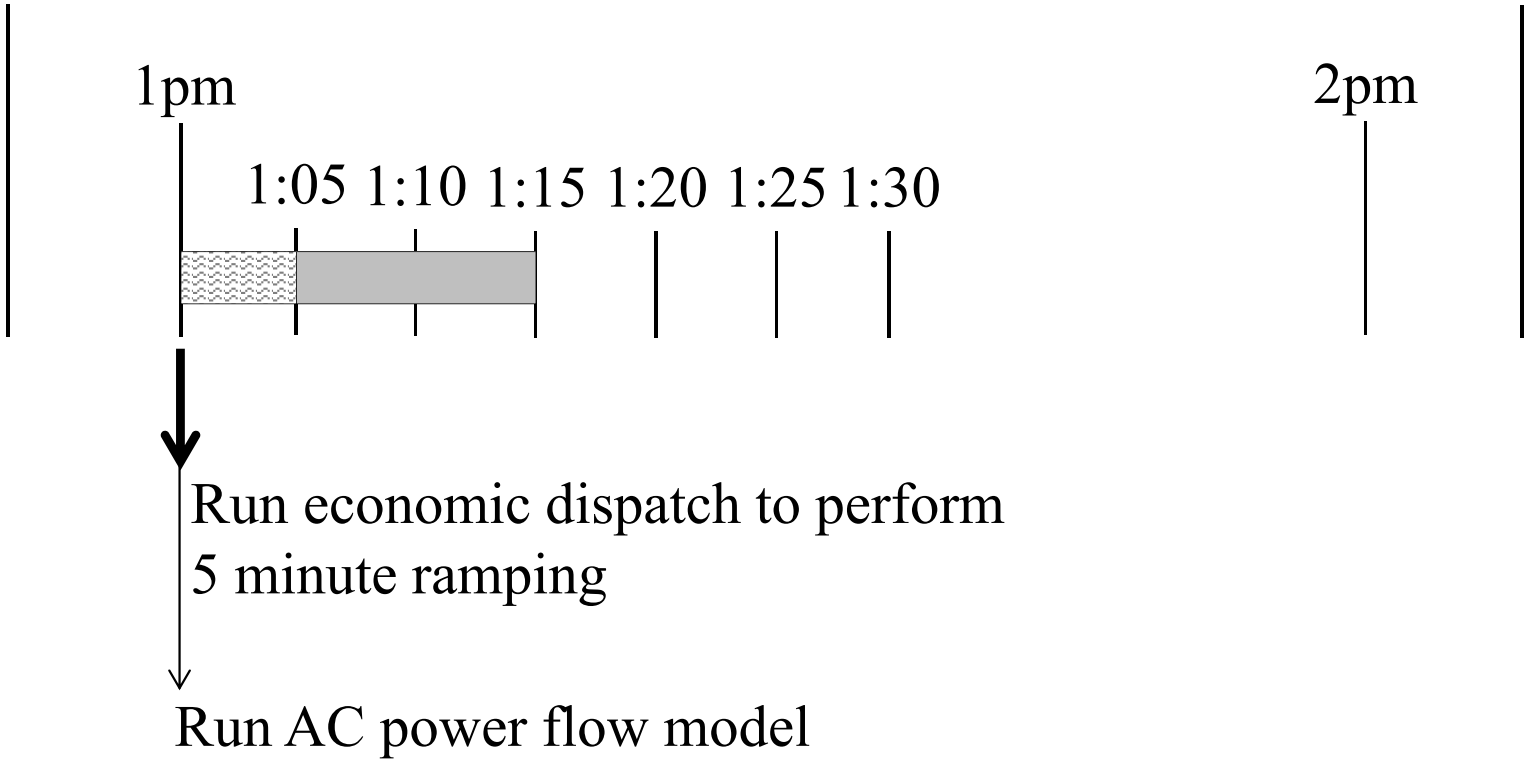
The timing of decisions

- Real-time economic dispatch problem



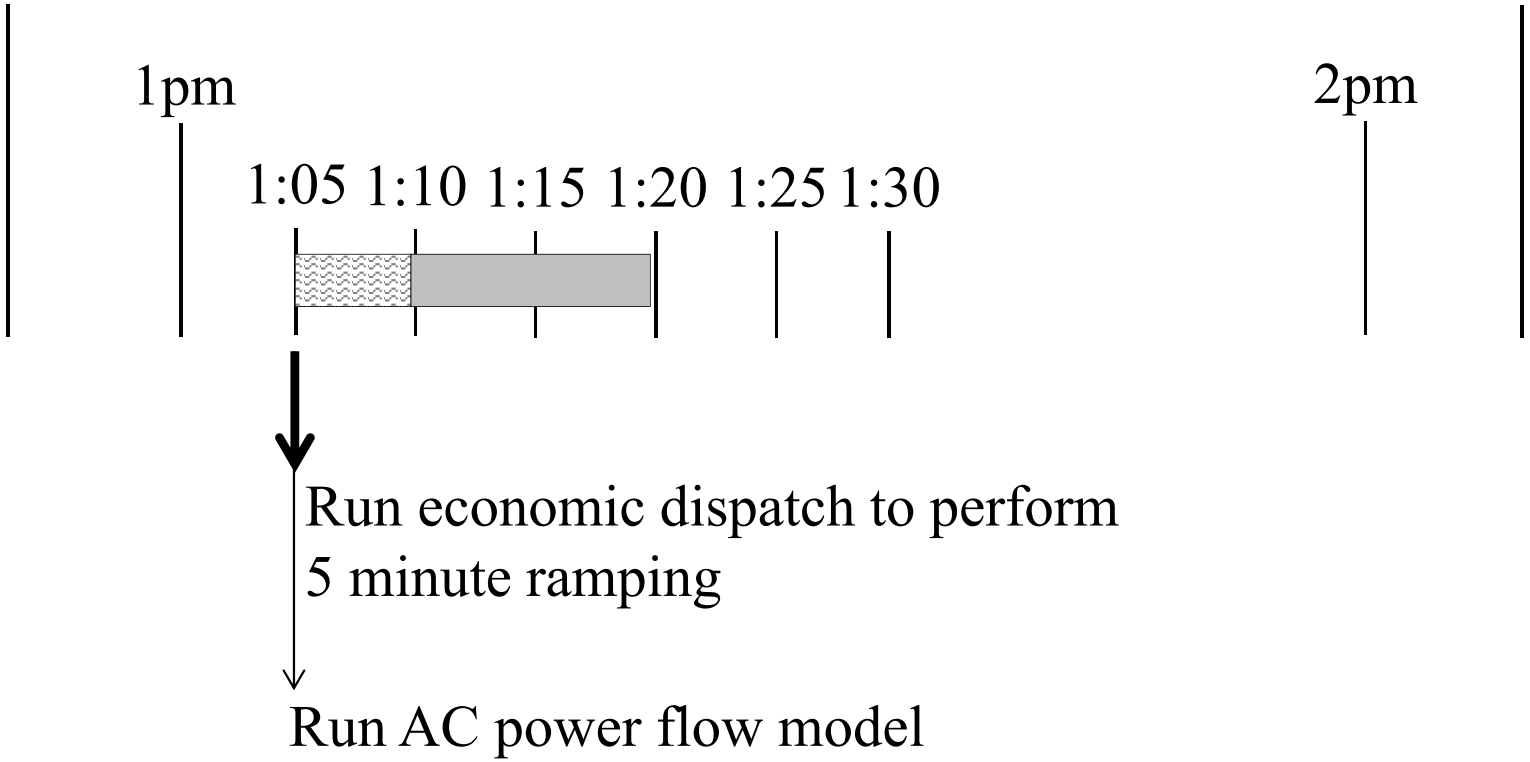
The timing of decisions

- Real-time economic dispatch problem



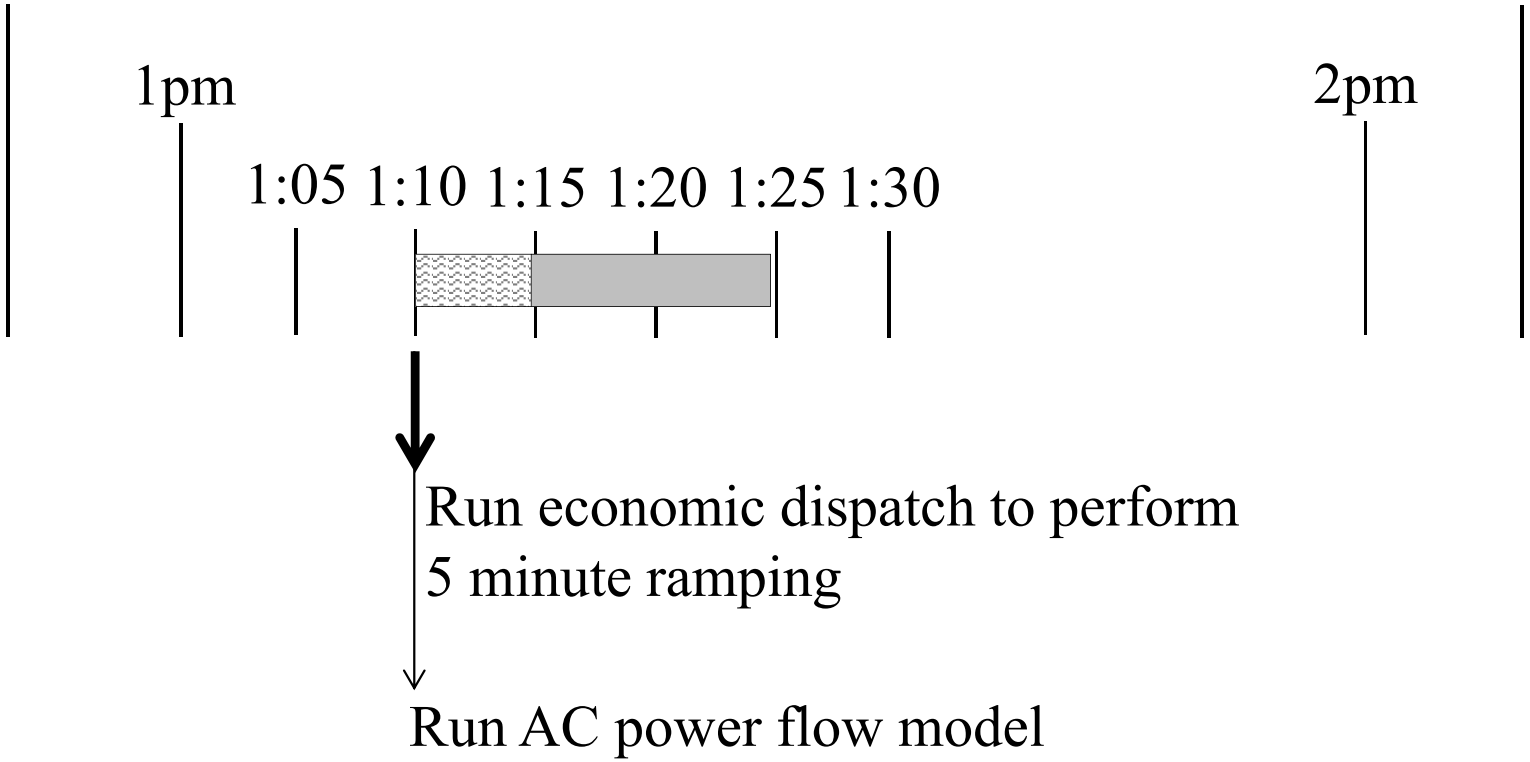
The timing of decisions

- Real-time economic dispatch problem



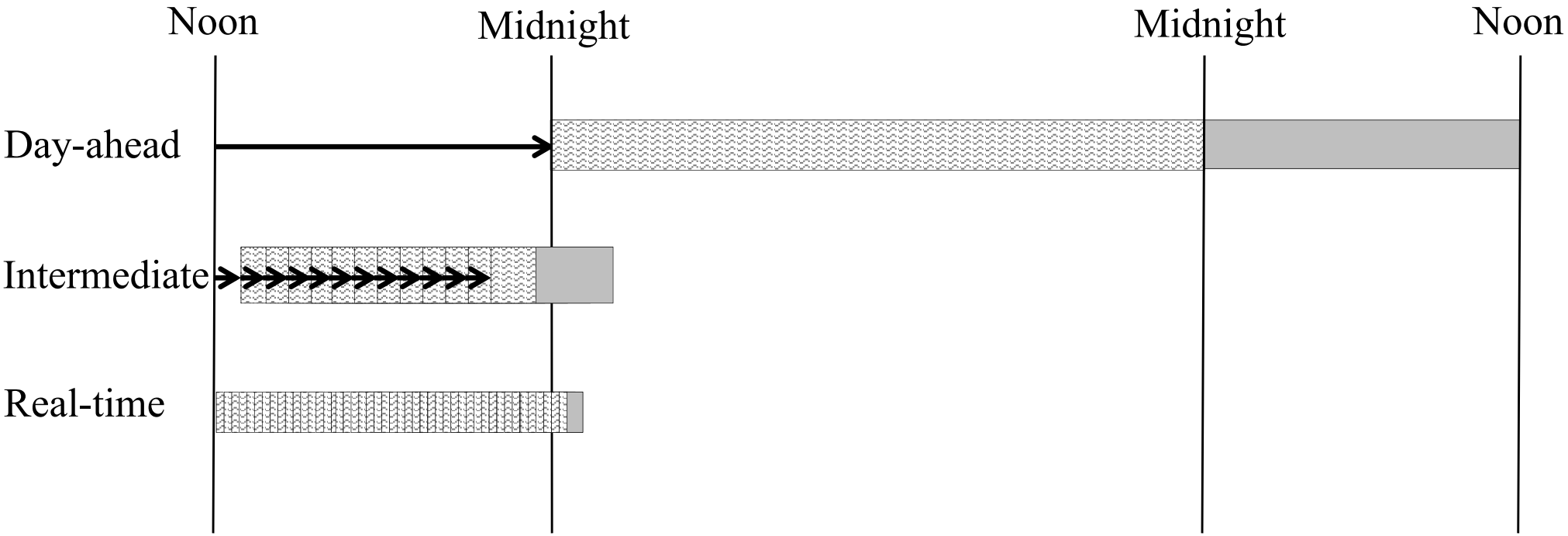
The timing of decisions

- Real-time economic dispatch problem



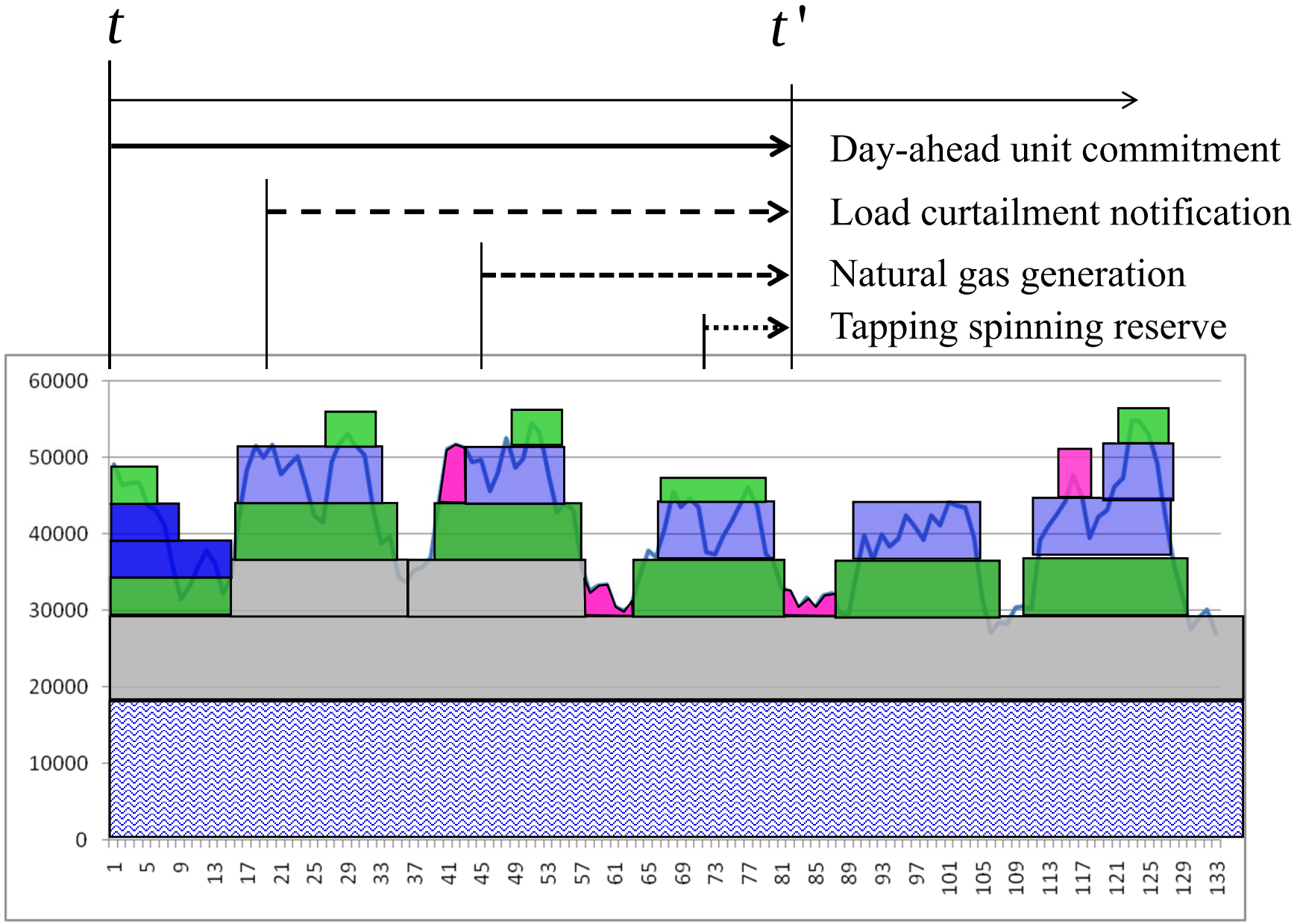
Parallel implementation

- Nested decisions are optimized in parallel

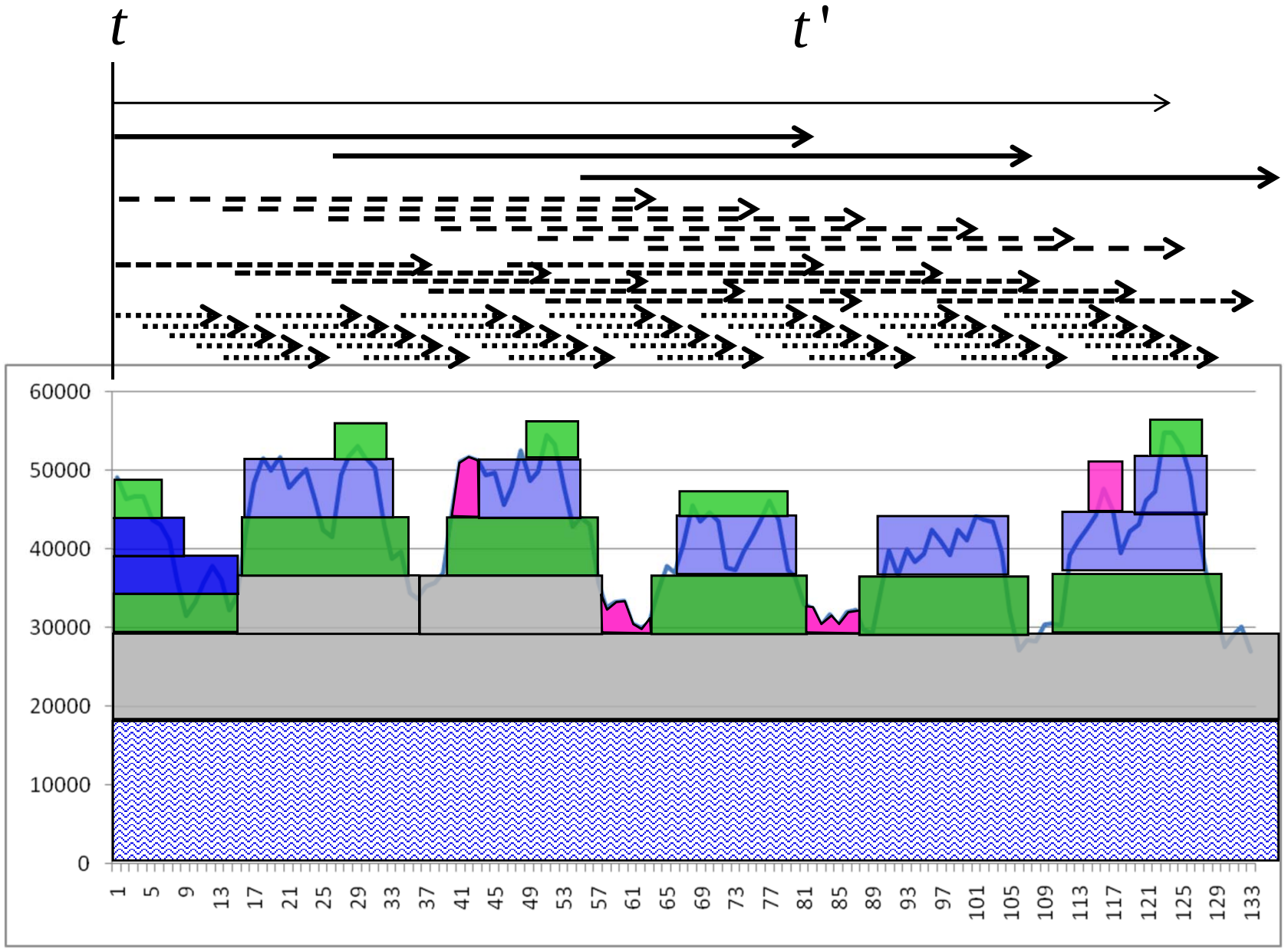


» We exploit the gap between when a model is run, and when unit commitment decisions are made, to continue optimizing nested decisions.

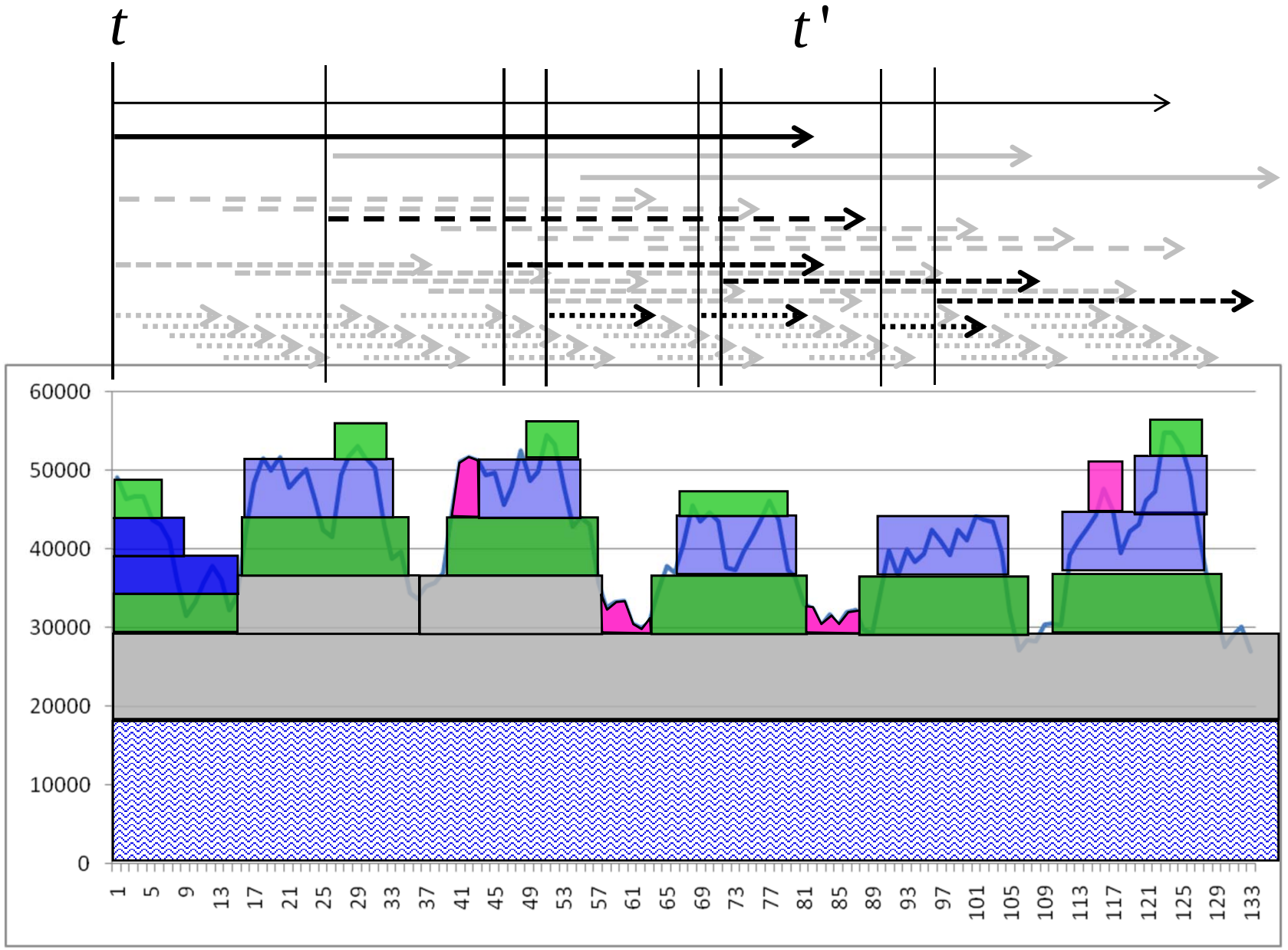
The time-staging of decisions



The nesting of decisions



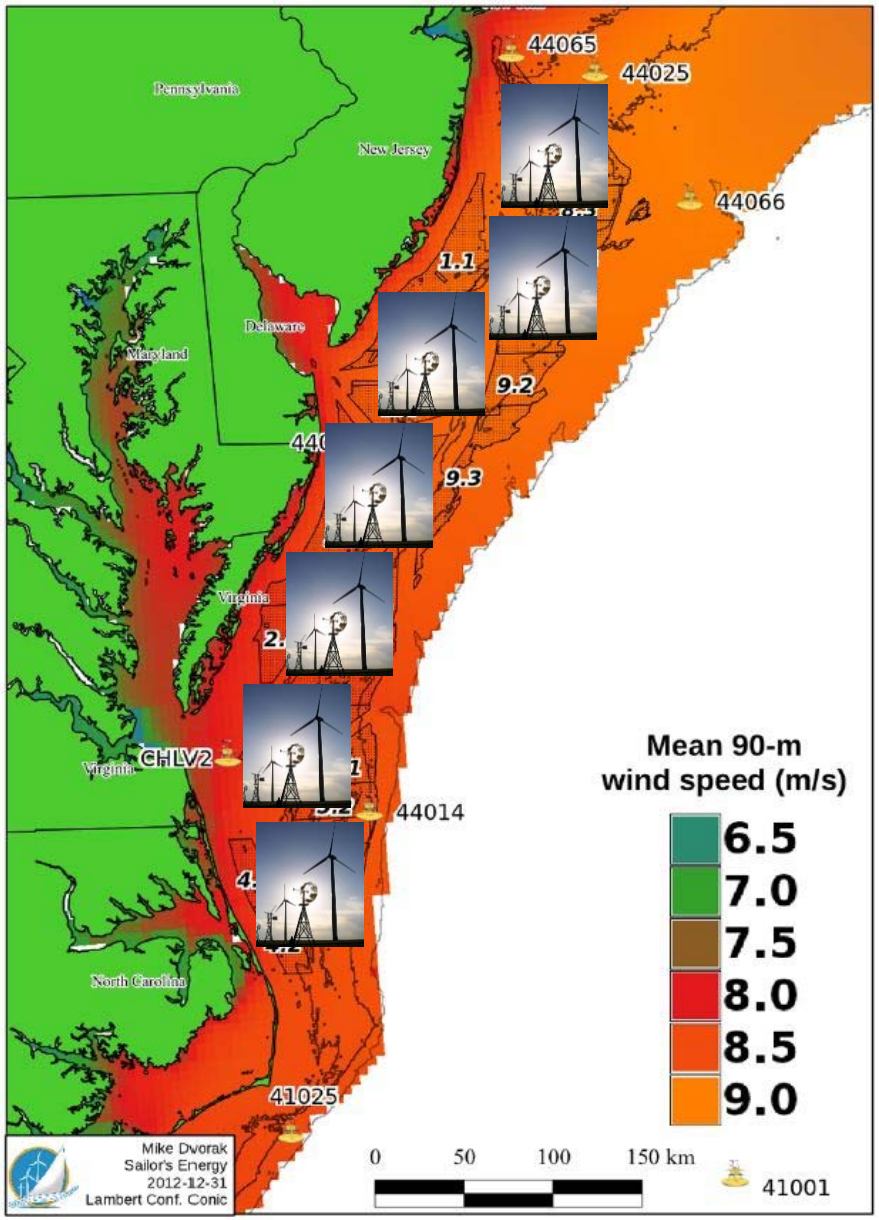
The nesting of decisions



SMART-ISO: Offshore wind study

- Mid-Atlantic Offshore Wind Integration and Transmission Study (U. Delaware & partners, funded by DOE)

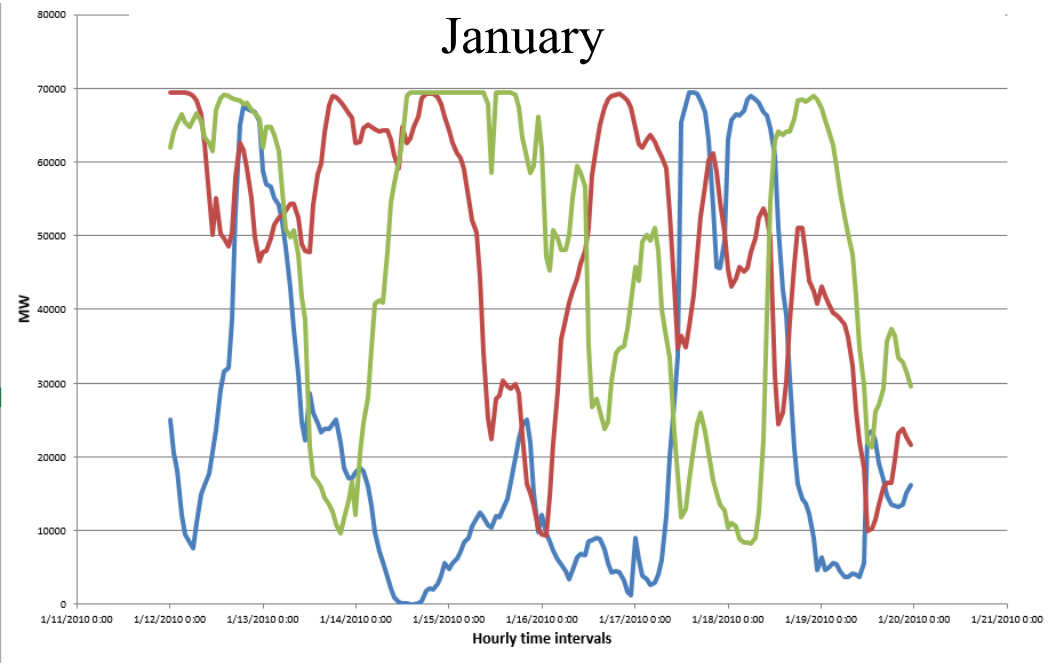
- 29 offshore sub-blocks in 5 build-out scenarios:
  - » 1: 8 GW
  - » 2: 28 GW
  - » 3: 40 GW
  - » 4: 55 GW
  - » 5: 78 GW

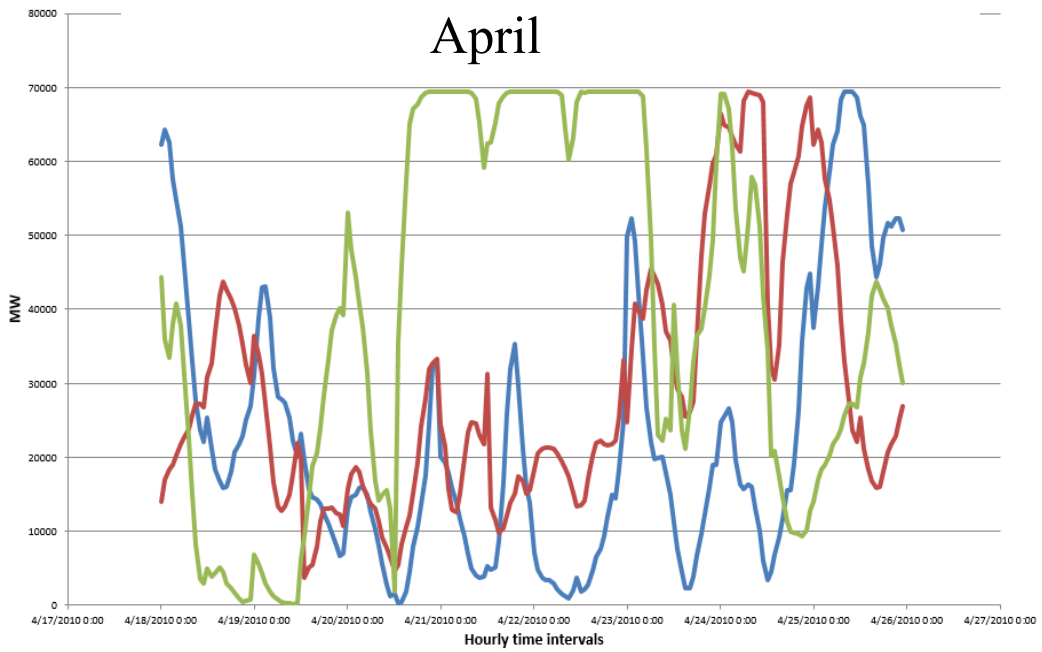


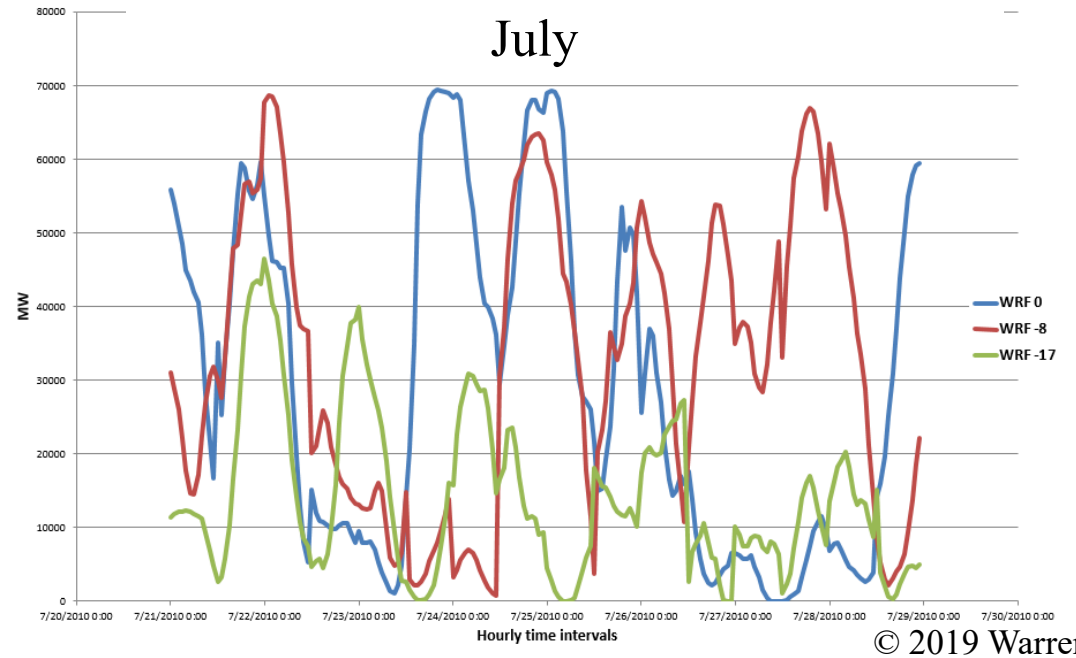
Generating wind sample paths

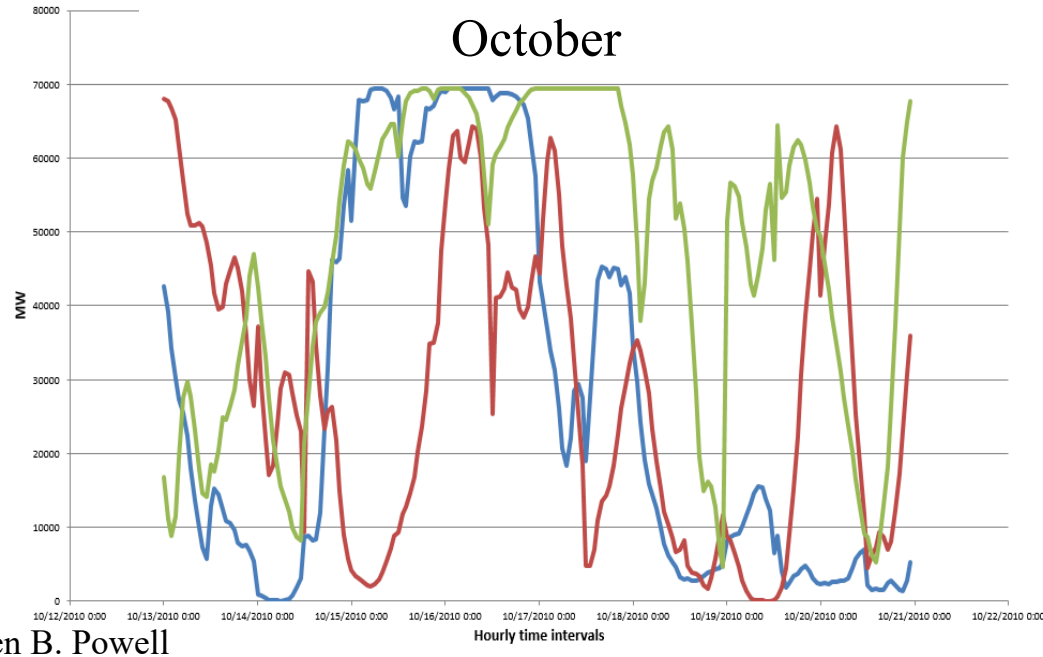
- Methodology for creating off-shore wind samples
  - » Use actual forecasts from on-shore wind to develop a stochastic error model
  - » Generate sample paths of wind by sampling errors from on-shore stochastic error model
  - » Repeat this for base case and five buildout levels
- Our sample paths are based on:
  - » Four months: January, April, July and October
  - » WRF forecasts from three weeks each month
  - » Seven sample paths generated around each forecast, creating a total of 84 sample paths.
  - » The next four screens show the WRF forecasts for each month, followed by a screen with one forecast, and the seven sample paths generated around that forecast.

WRF simulations for offshore wind



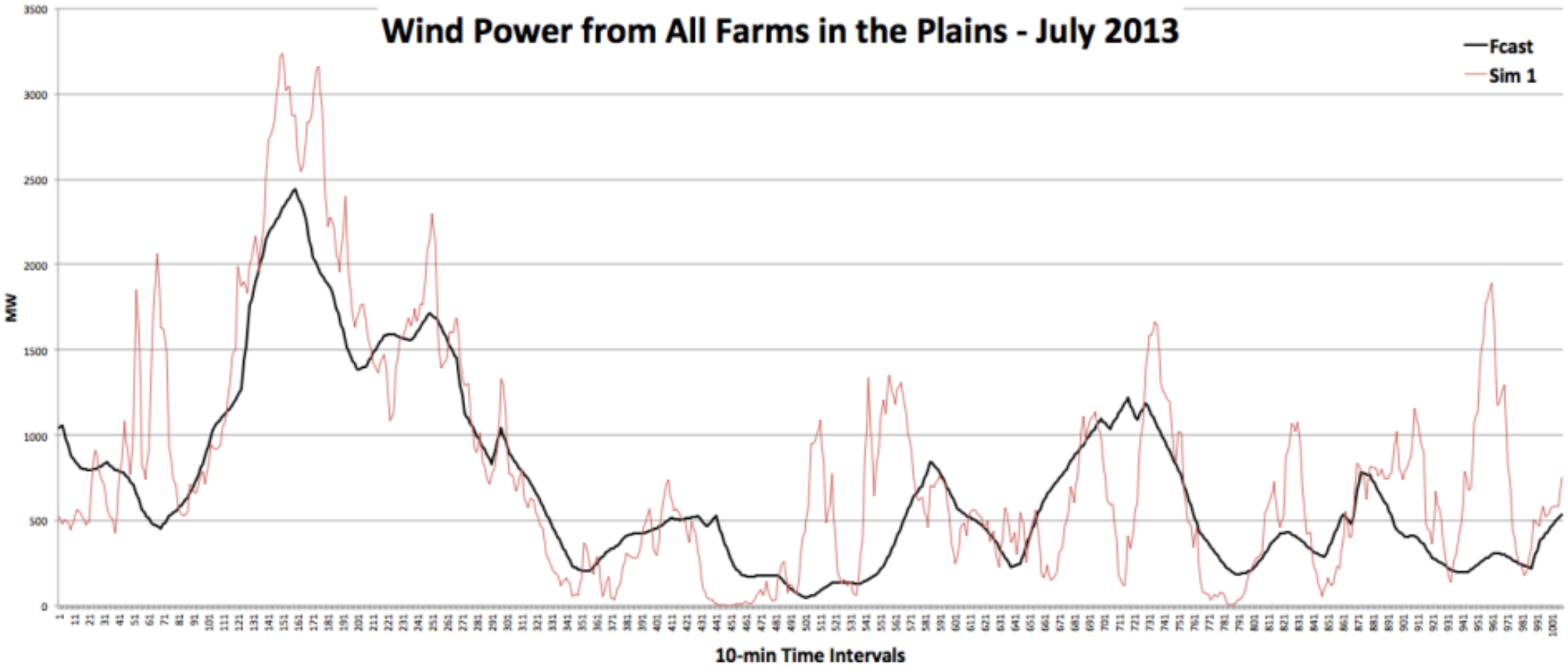






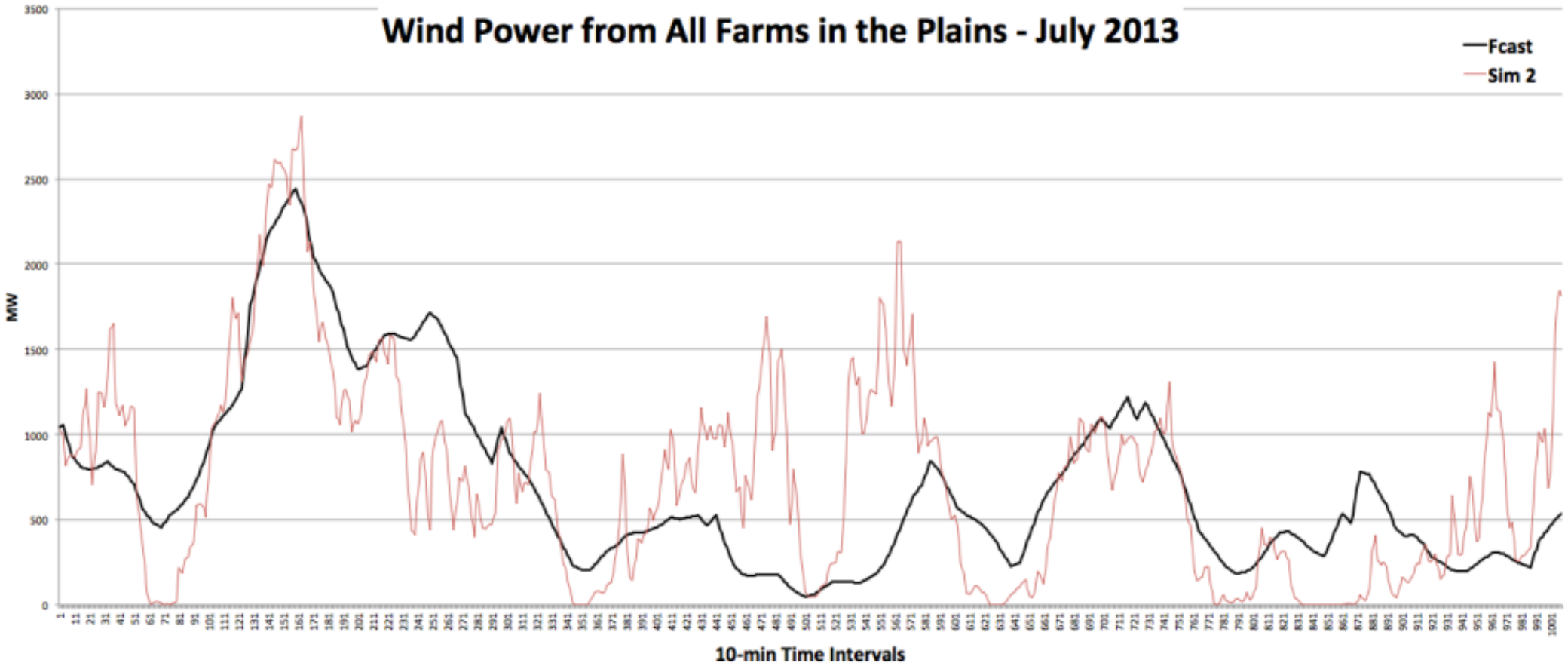
Stochastic lookahead policies

- Creating wind scenarios (Scenario #1)



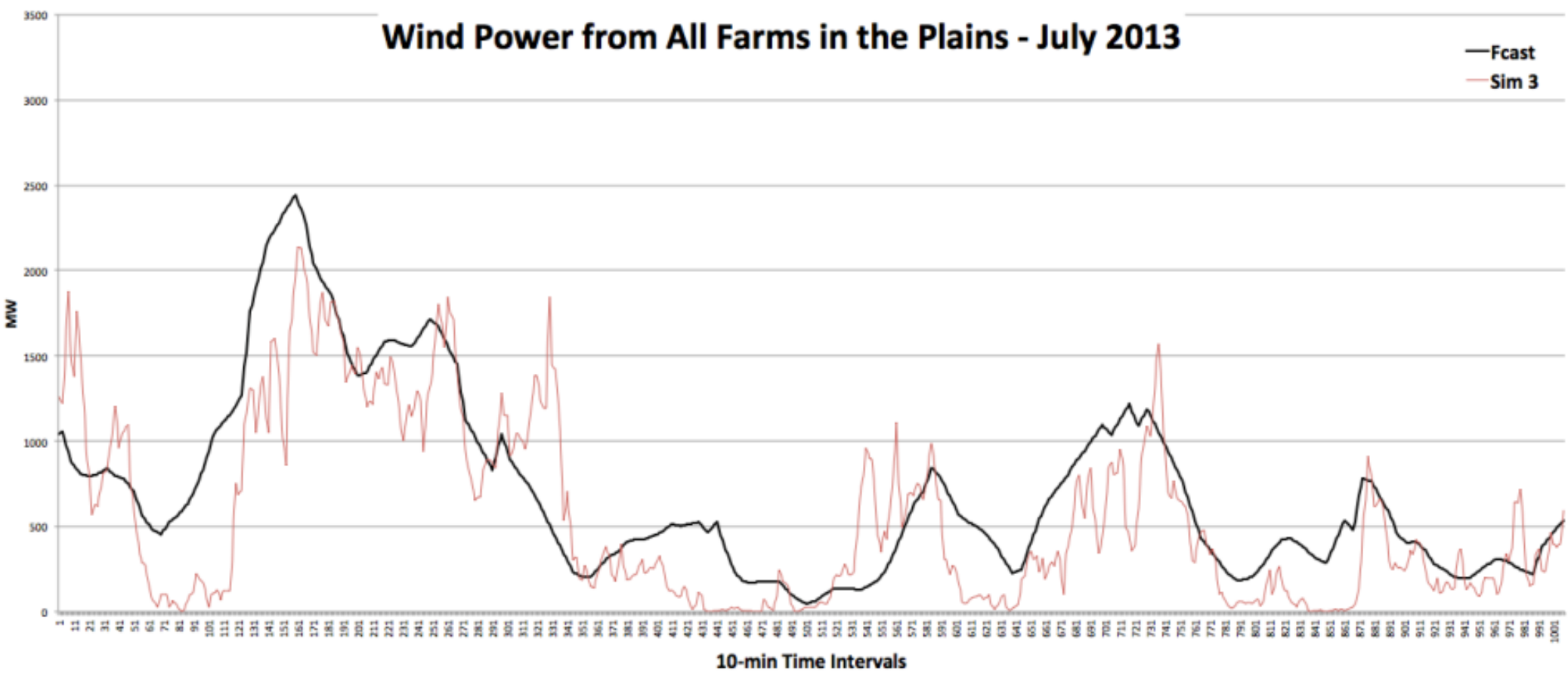
Stochastic lookahead policies

- Creating wind scenarios (Scenario #2)



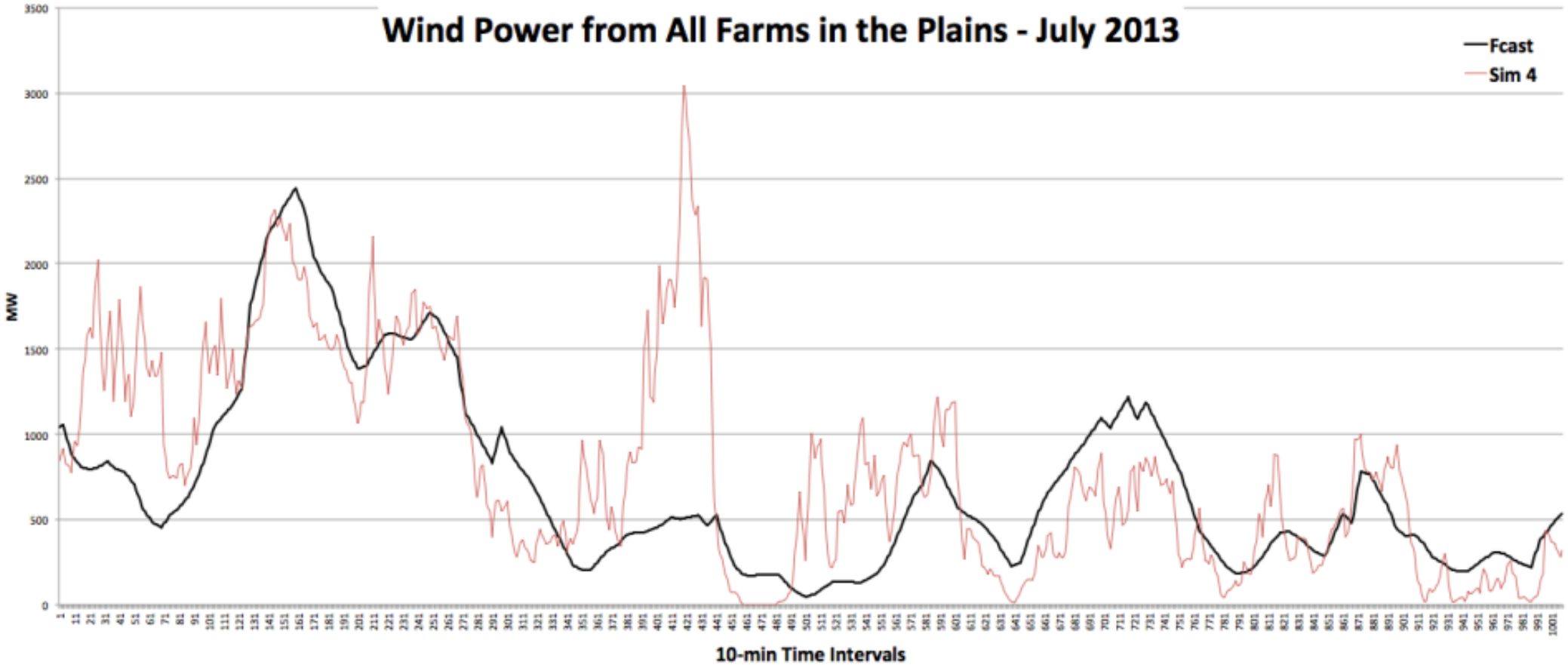
Stochastic lookahead policies

- Creating wind scenarios (Scenario #3)



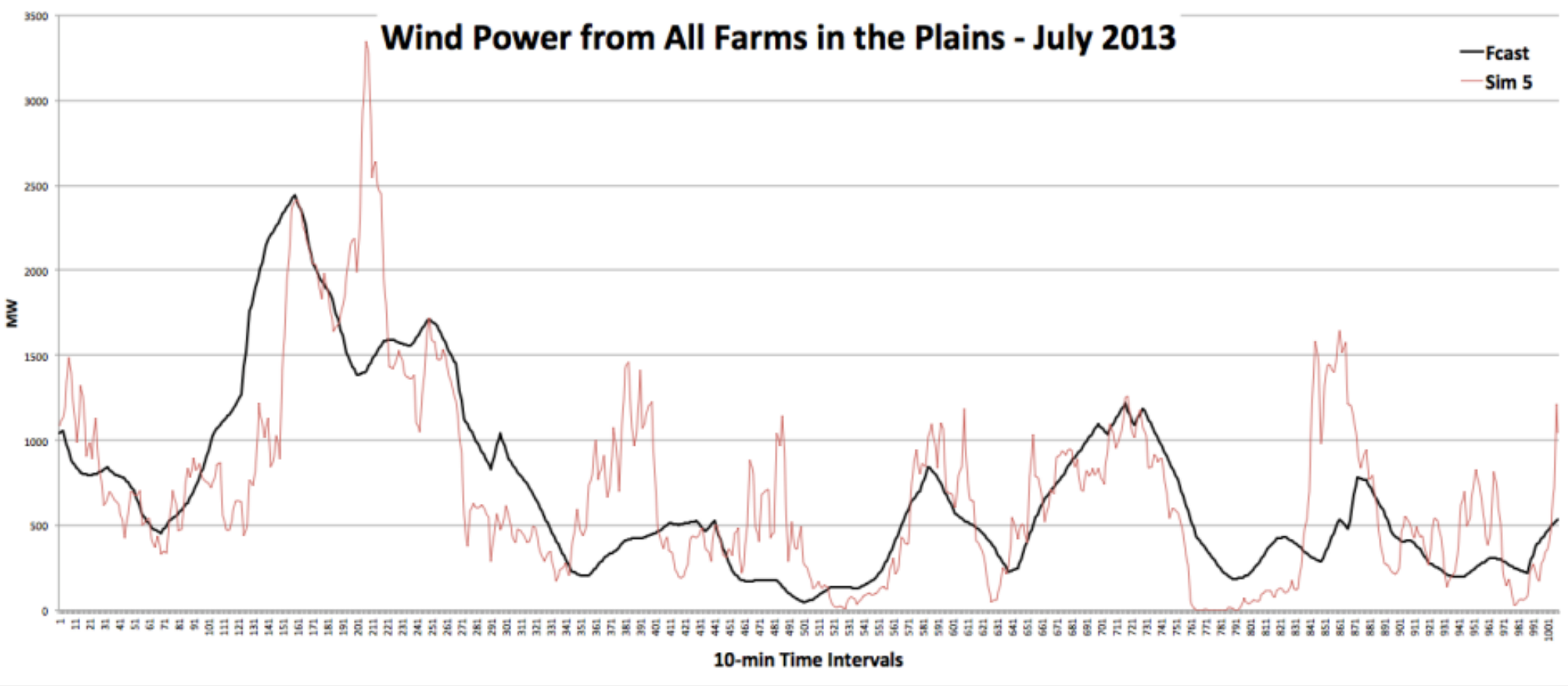
Stochastic lookahead policies

- Creating wind scenarios (Scenario #4)



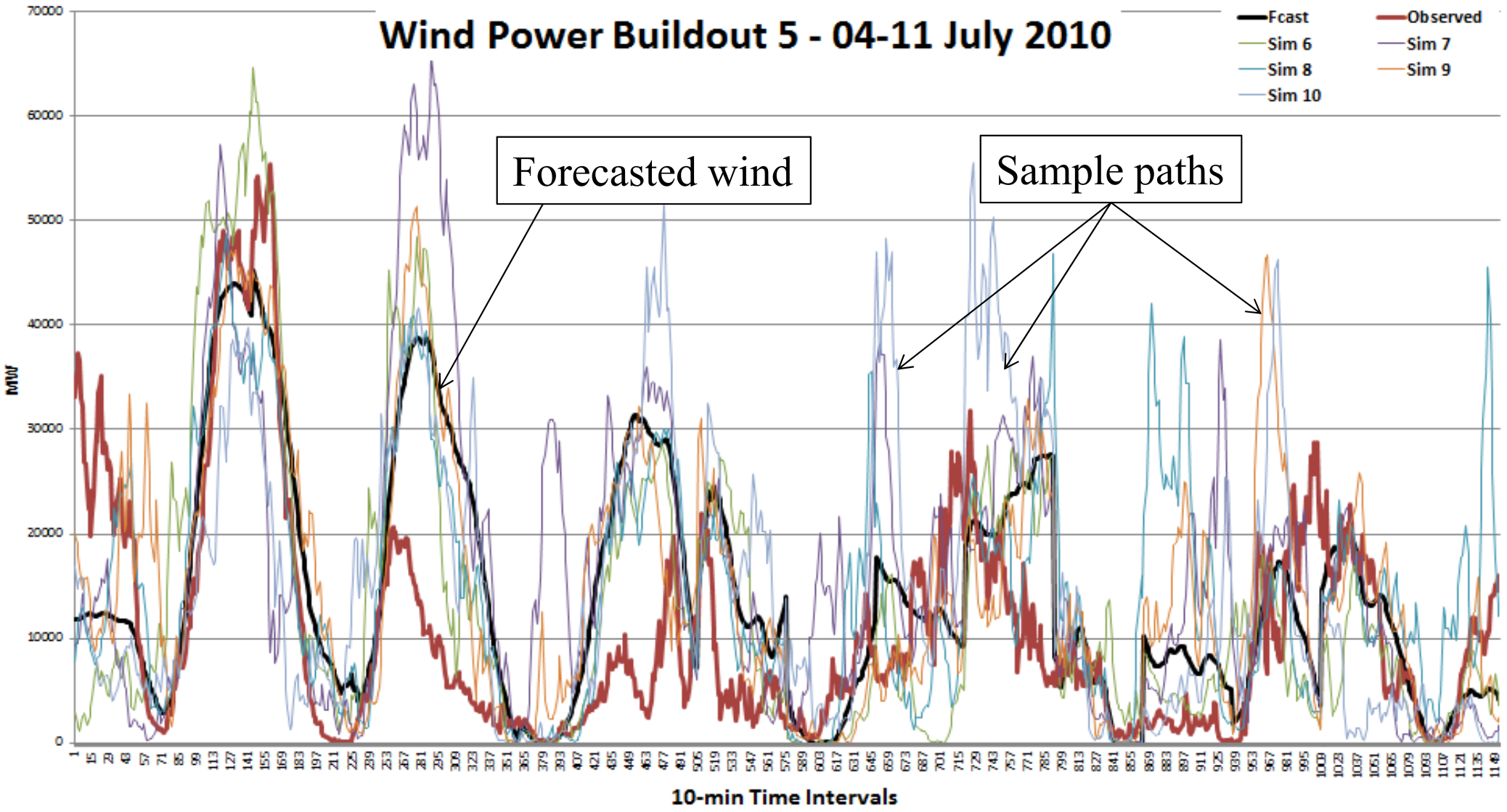
Stochastic lookahead policies

- Creating wind scenarios (Scenario #5)



Modeling forecast errors

- Offshore wind – Buildout level 5 – ARMA error model



The robust CFA for unit commitment

- The robust cost function approximation
  - » The ISOs today use simple rules to obtain robust solutions, such as scheduling reserves.
  - » These strategies are fairly sophisticated:
    - Spinning
    - Non-spinning
    - Distributed spatially across the regions.
  - » This reserve strategy is a form of *robust cost function approximation*. They represent a parametric modification of a base cost policy, and have been tuned over the years in an online setting (called the real world).
  - » The ISOs use what would be a hybrid: the robust CFA concept applied to a deterministic lookahead base model.

Four classes of policies

1) Policy function approximations (PFAs)
   - » Lookup tables, rules, parametric functions

2) Cost function approximation (CFAs)
   - » $X^{CFA}(S_t \mid \theta) = \arg\min_{x_t \in \bar{X}_t(\theta)} \bar{C}^{\pi}(S_t, x_t \mid \theta)$

3) Policies based on value function approximations (VFAs)
   - » $X_t^{VFA}(S_t) = \arg\min_{x_t} \left(C(S_t, x_t) + \gamma \bar{V}_t^x \left(S_t^x(S_t, x_t) \right) \right)$

4) Lookahead policies
   - » ***Deterministic lookahead:***

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

   - » ***Stochastic lookahead (e.g. stochastic trees)***

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

A hybrid lookahead-CFA policy

- A deterministic lookahead model
  - » Optimize over all decisions at the same time

$$\min_{\substack{(x_{tt'})_{t'=1,\ldots,24} \\ (yt_{t'})_{t'=1,\ldots,24}}} \sum_{t'=t}^{t+H} C(x_{tt'}, y_{tt'})$$

Steam generation

Gas turbines

  - » In a deterministic model, we mix generators with different notification times:
    - Steam generation is made day-ahead
    - Gas turbines can be planned an hour ahead or less

A hybrid lookahead-CFA policy

- A deterministic lookahead policy
  - » This is the policy produced by solving a deterministic lookahead model

$$X_t^{\pi}(S_t) = \min_{\substack{(x_{tt'})_{t'=1,\ldots,24} \\ (yt_{t'})_{t'=1,\ldots,24}}} \sum_{t'=t}^{t+H} C(x_{tt'}, y_{tt'})$$

Steam generation

Gas turbines

  - » *No ISO uses a deterministic lookahead model. It would never work, and for this reason they have never used it. They always modify the model to produce a robust solution.*

A hybrid lookahead-CFA policy

- A robust CFA policy
  - » The ISOs introduce reserves:

$$X_t^\pi(S_t|\theta) = \min_{\substack{(x_{tt'})_{t'=1,\ldots,24} \\ (y_{tt'})_{t'=1,\ldots,24}}} \sum_{t'=t}^{t+H} C(x_{tt'}, y_{tt'})$$

$$x_{t,t'}^{\max} - x_{t,t'} \geq \theta^{up} L_{tt'} \qquad \text{Up-ramping reserve}$$

$$x_{t,t'} - x_{t,t'}^{\max} \geq \theta^{down} L_{tt'} \qquad \text{Down-ramping reserve}$$

  - » This modification is a form of parametric function (a parametric cost function approximation). It has to be tuned to produce a robust policy.

A hybrid lookahead-CFA policy

- A robust CFA policy
  - » The ISOs introduce reserves:

$$X_t^\pi(S_t|\theta) = \min_{\substack{(x_{tt'})_{t'=1,\ldots,24} \\ (y_{tt'})_{t'=1,\ldots,24}}} \sum_{t'=t}^{t+H} C(x_{tt'}, y_{tt'})$$

$$x_{t,t'}^{\max} - x_{t,t'} \geq \theta^{up} L_{tt'} \qquad \text{Up-ramping reserve}$$

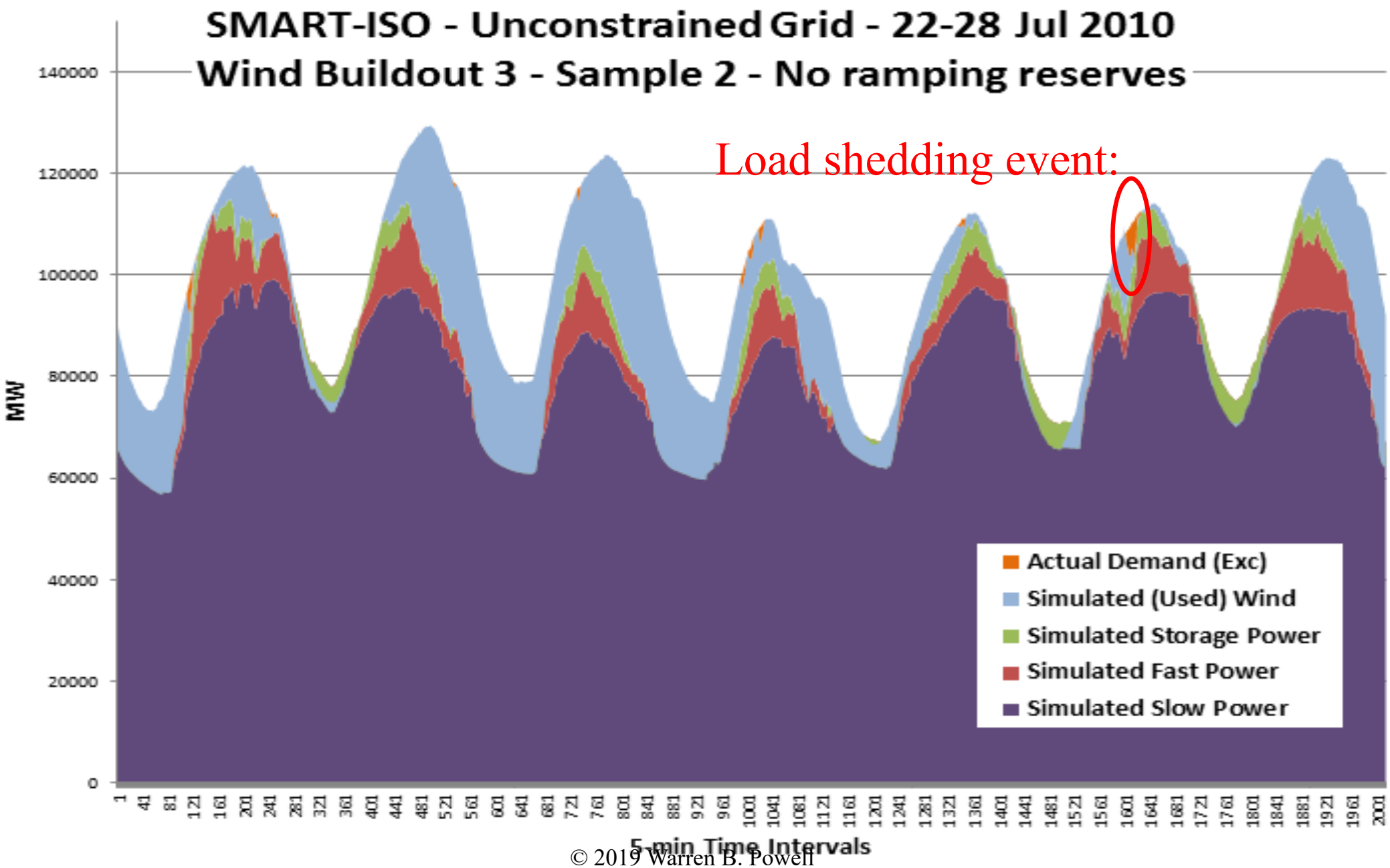
$$x_{t,t'} - x_{t,t'}^{\max} \geq \theta^{down} L_{tt'} \qquad \text{Down-ramping reserve}$$

  - » It is easy to tune this policy when there are only two parameters $\theta = \left(\theta^{up}, \theta^{down}\right)$
  - » But we might want the parameters to depend on information such as forecasts.

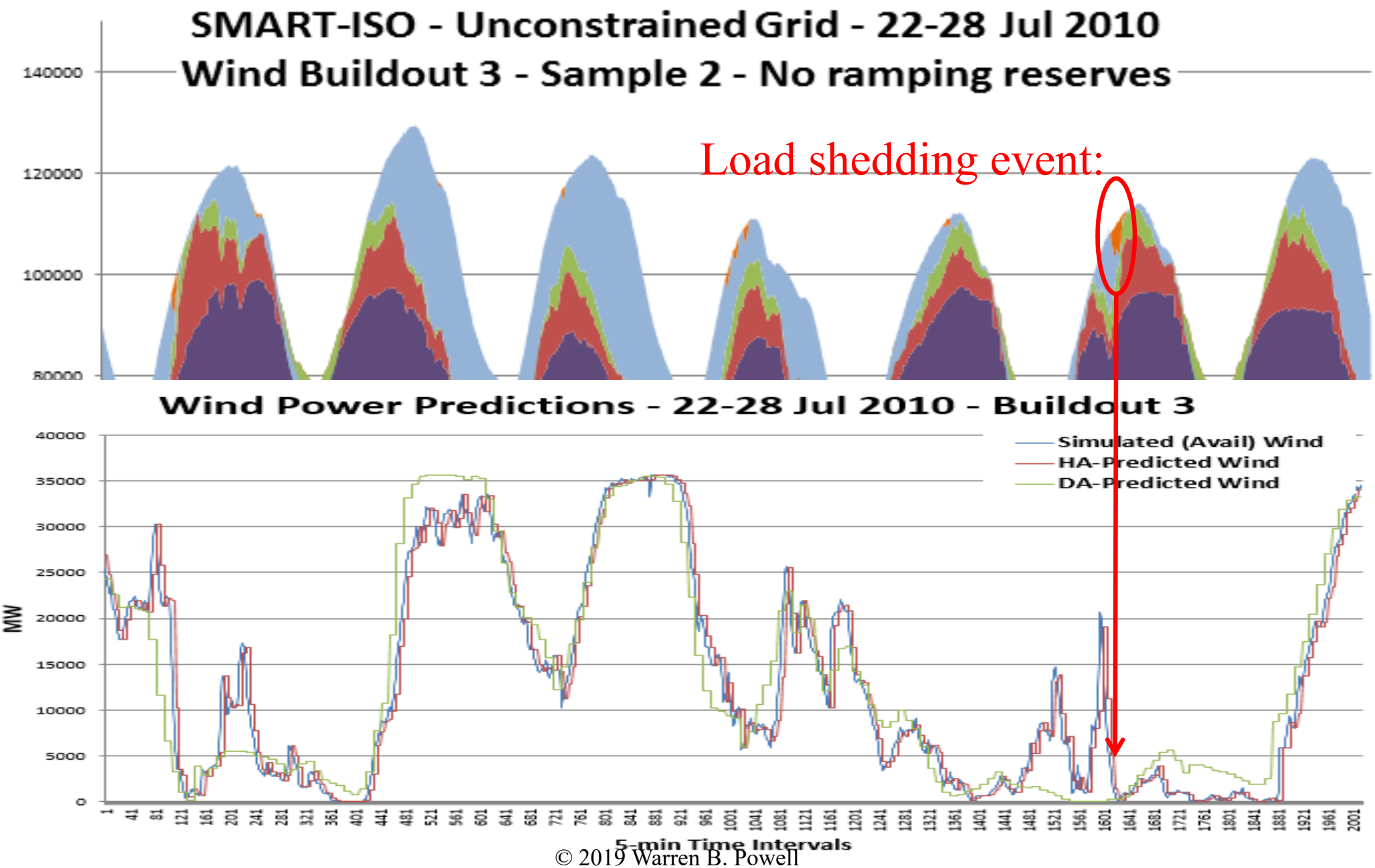
Stochastic unit commitment

- Properties of a parametric cost function approximation
  - » It allows us to use domain expertise to control the structure of the solution
  - » We can force the model to schedule extra reserve:
    - At all points in time (or just during the day)
    - Allocated across regions
  - » This is all accomplished without using a multiscenario model.
  - » The next slides illustrate the solution before the reserve policy is imposed, and after. Pay attention to the change pattern of gas turbines (in maroon).

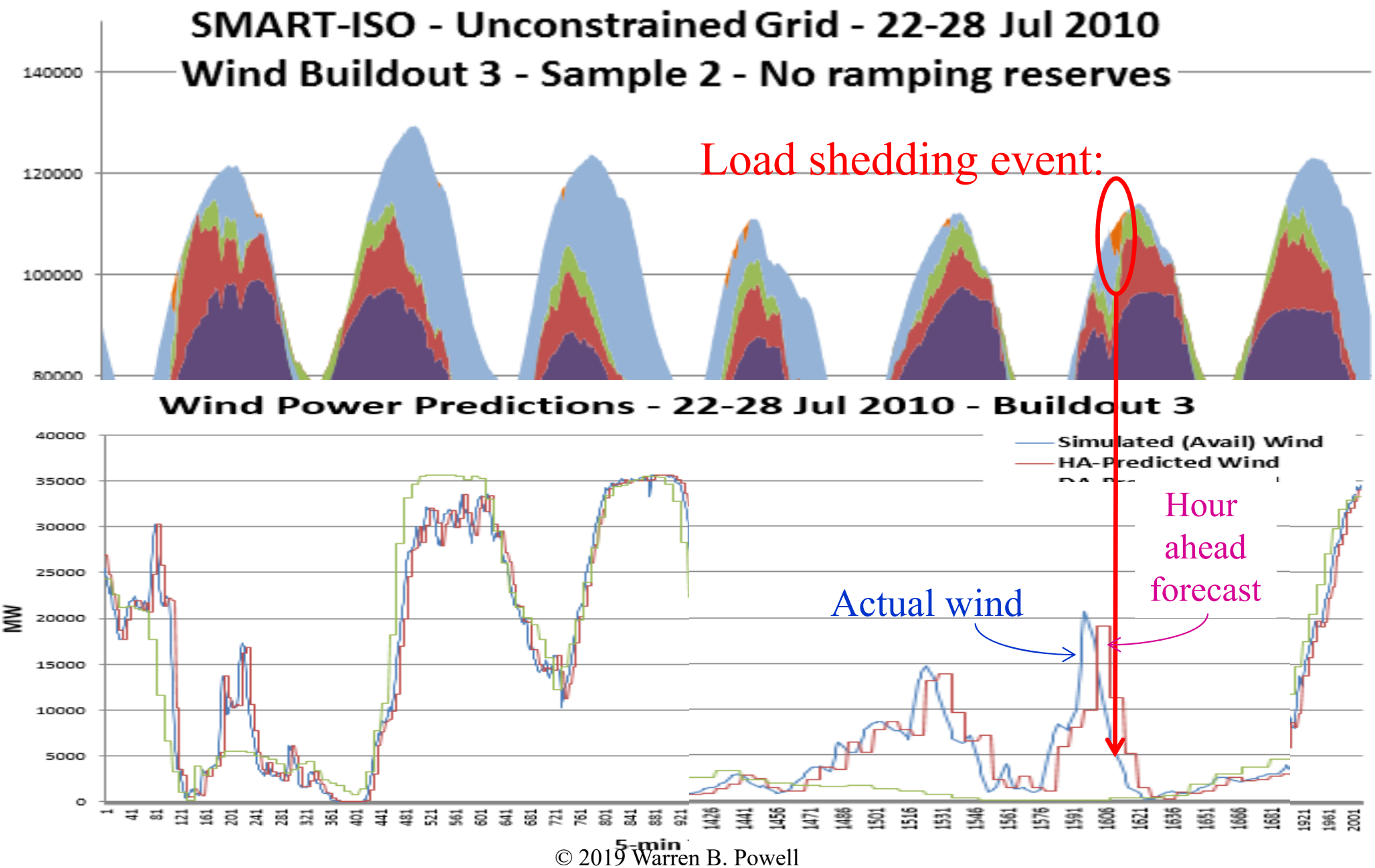
Stochastic lookahead policies



Stochastic lookahead policies



Stochastic lookahead policies



Stochastic unit commitment

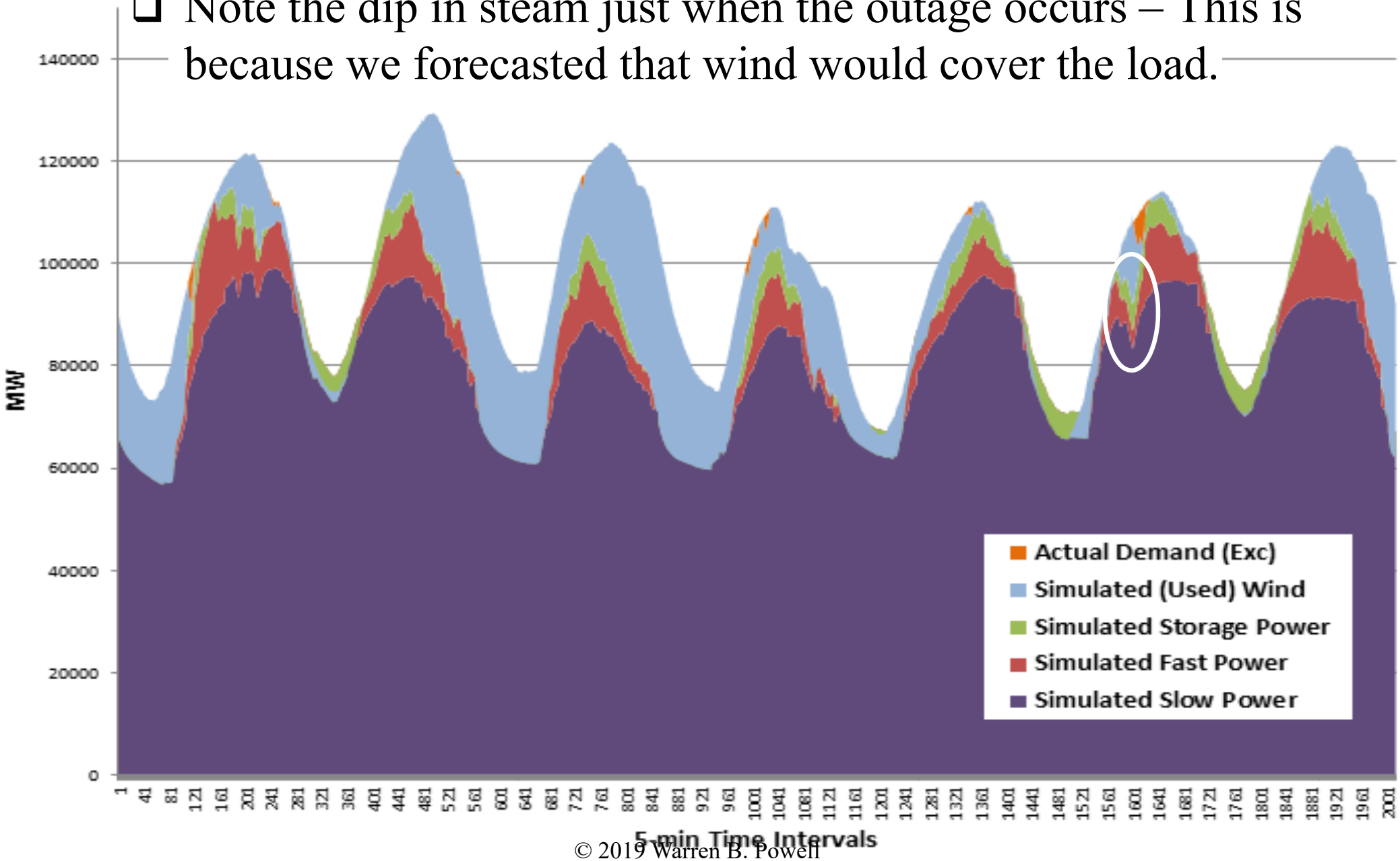
- The outage appears to happen under the following conditions:
  - » It is July (we are maxed out on gas turbines – we were unable to schedule more gas reserves than we did).
  - » The day-ahead forecast of energy from wind was low (it is hard to see, but the day-ahead forecast of wind energy was virtually zero).
  - » The energy from wind was above the forecast, but then dropped suddenly.
  - » It was the sudden drop, and the lack of sufficient turbine reserves, that created the load shedding event.
  - » Note that it is very important to capture the actual behavior of wind.

Stochastic unit commitment

- What is next:
  - » We are going to try to fix the outage by allowing the day-ahead model to see the future.
  - » This will fix the problem, by scheduling additional steam, but it does so by scheduling extra steam at *exactly the time that it is needed*.
  - » This is the power of the sophisticated tools that are used by the ISOs (which we are using). If you allow a code such as Cplex (or Gurobi) to optimize across hundreds of generators, you will get almost exactly what you need.

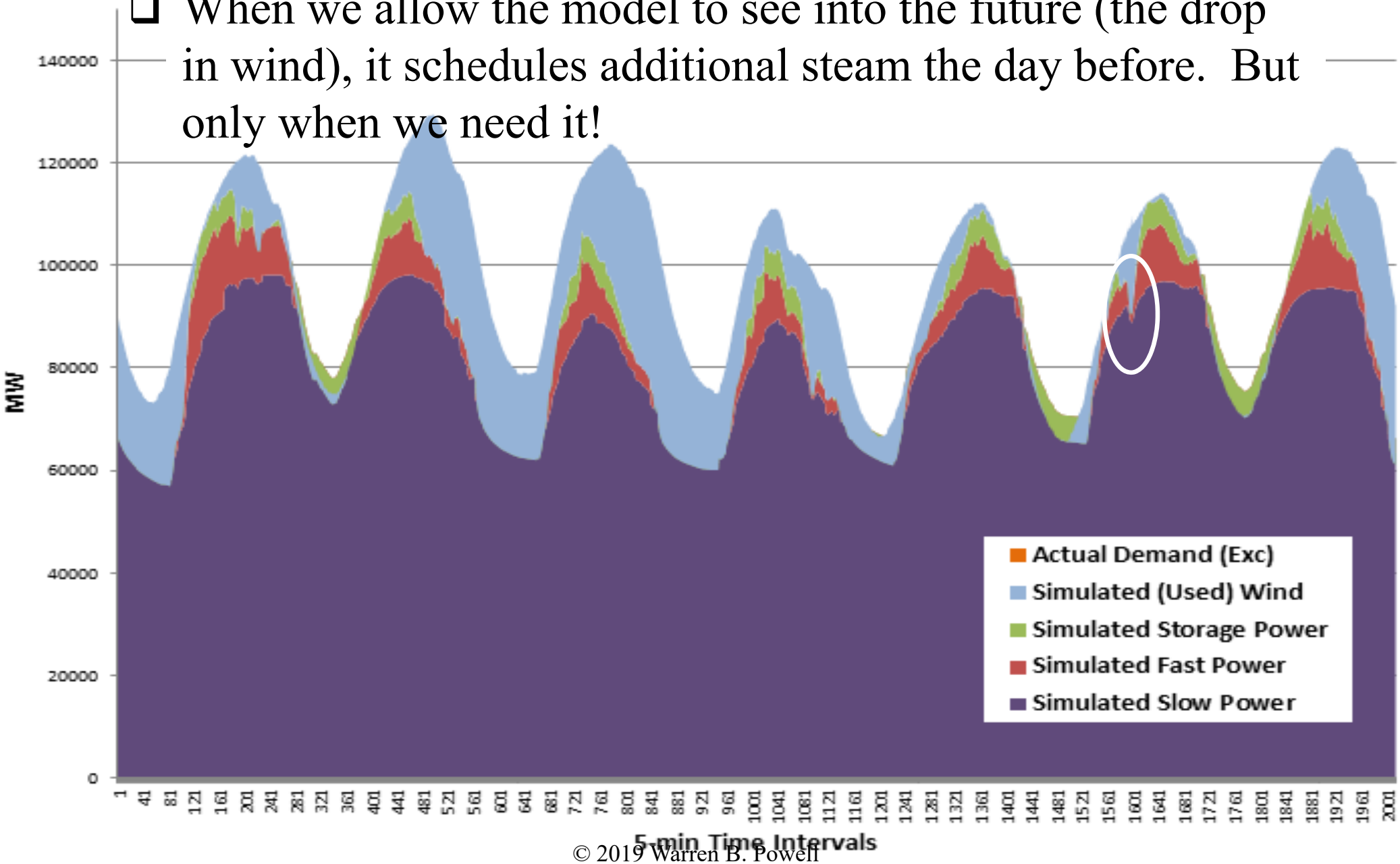
Stochastic lookahead policies

- Note the dip in steam just when the outage occurs – This is because we forecasted that wind would cover the load.

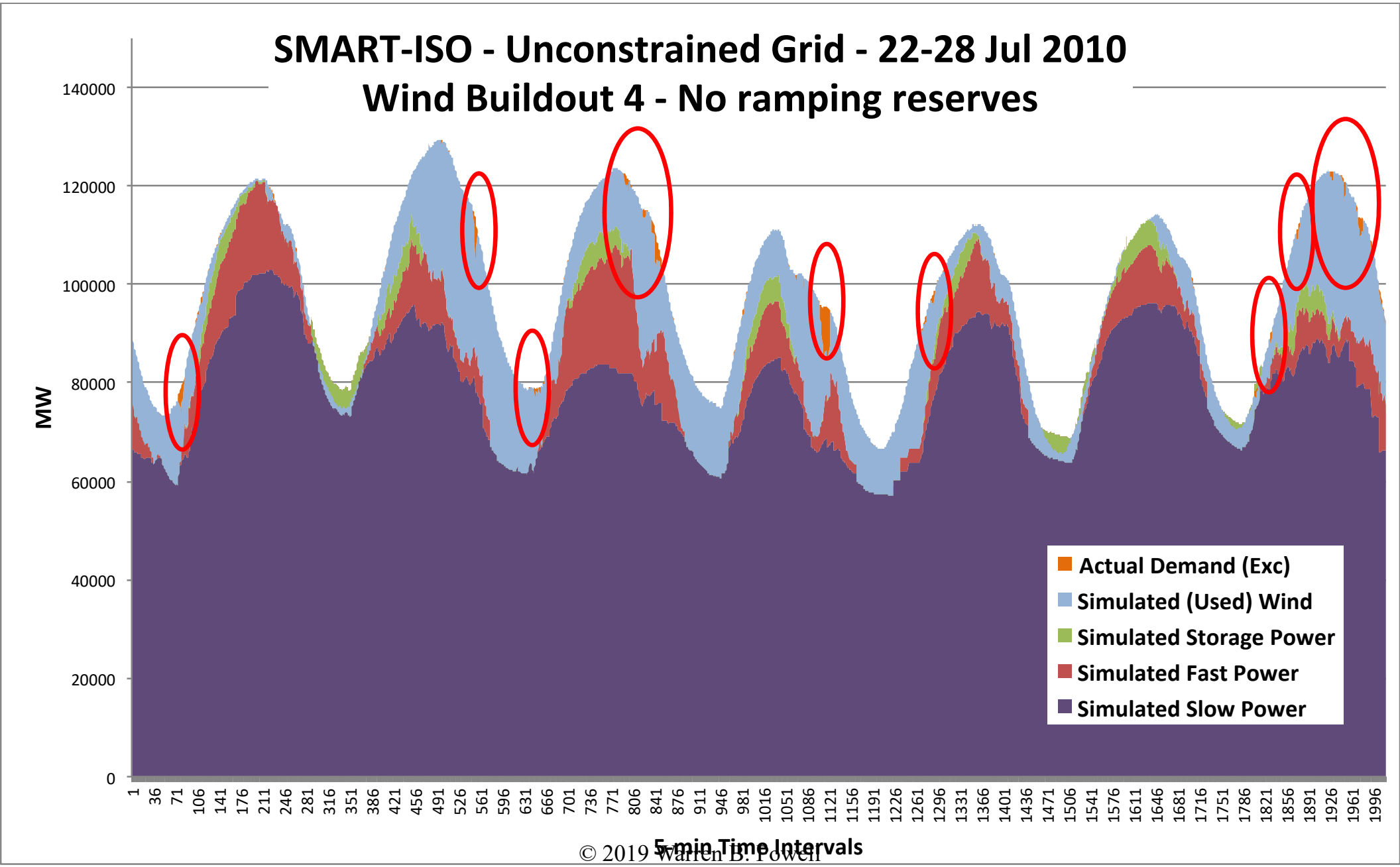


Stochastic lookahead policies

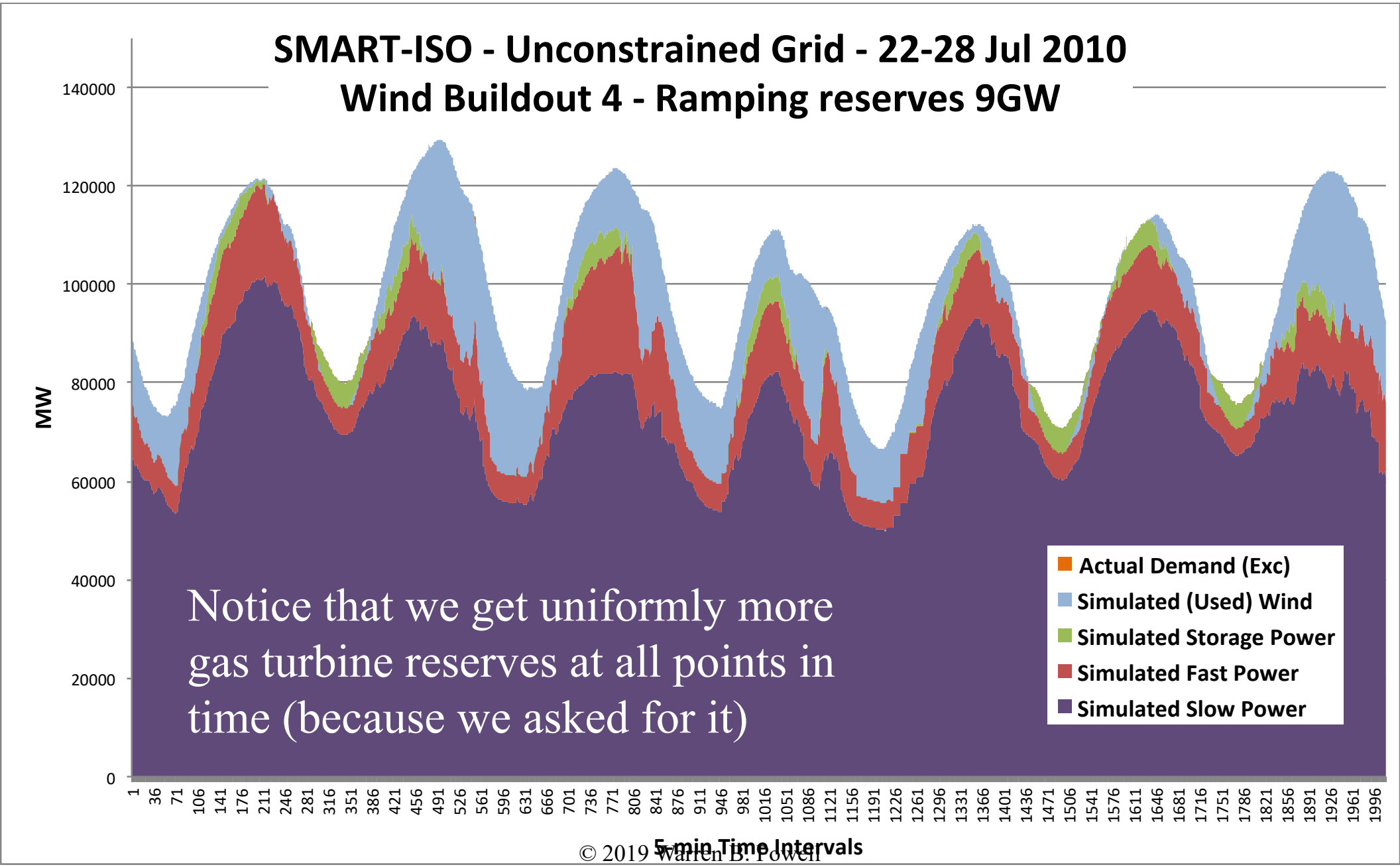
- When we allow the model to see into the future (the drop in wind), it schedules additional steam the day before. But only when we need it!



SMART-ISO: Offshore wind study



SMART-ISO: Offshore wind study



Discrete Markov decision processes

Modeling

Discrete Markov decision processes

- Dynamic programming
  - » A "dynamic program" is *any* sequential decision process which can be written

$$(S_0, x_0 = X^\pi(S_0), W_1, S_1, x_1 = X^\pi(S_1), W_2, S_2, \ldots)$$

  - » The problem is to find an effective policy $X^\pi(S_t)$, in the presence of the uncertainties $W_1, W_2, \ldots$

Discrete Markov decision processes

- Our modeling framework consists of:
  - » States
    - Initial state S_0
    - Dynamic states S_t
  - » Decisions/actions/controls x_t, a_t, u_t
  - » Exogenous information W_t
  - » Transition function:
    - $S_{t+1} = S^M(S_t, x_t, W_{t+1})$
  - » Objective function

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

Discrete Markov decision processes

- "Canonical model" from Puterman (Ch. 3)

Decision epochs:

$$T = \{1, 2, \ldots, N\}, \quad N \leq \infty.$$

States:

$$S = S^1 \times S^2 \times \cdots \times S^K.$$

Actions (process to be selected):

$$A_s = \{1, 2, \ldots, K\}, \quad s \in S.$$

Rewards:

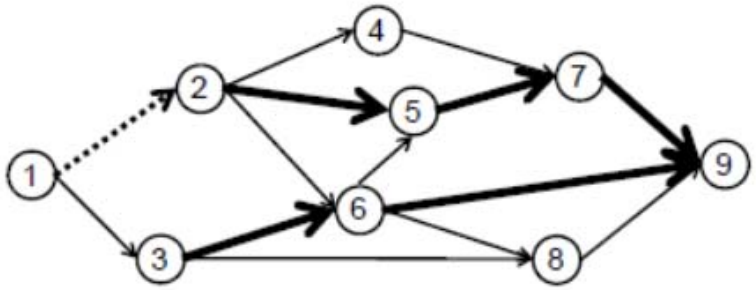
$$r_t((s^1, s^2, \ldots, s^K), a) = r_t^i(s^i) \quad \text{if } a = i.$$

Transition probabilities:

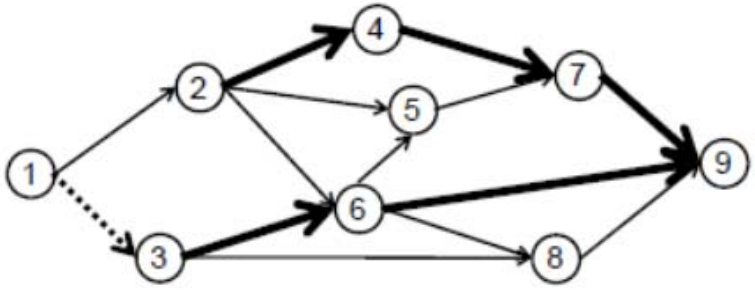
$$p_t((u^1, \ldots, u^k)|(s^1, \ldots, s^k), i) = \begin{cases} p(j^i|s^i) & u^i = j^i \text{ and } u^m = s^m \quad \text{for } m \neq i \\ 0 & u^m \neq s^m \quad \text{for some } m \neq i \end{cases}$$

Discrete Markov decision processes

- The optimality equation – deterministic problem
  - » The decision out of node 1 depends on what we are planning on doing out of node 2.



(a)



(b)

Figure 11.2 (a) Decision to go (1,2) given the path 2-5-7-9. (b) Decision to go (1,3) when path out of node 2 changes.

Discrete Markov decision processes

- Bellman for deterministic problems
  - » The best decision out of node S_t is given by

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

  - » where

$$V_t(S_t) = \arg\max_{x_t} \left(C(S_t, x_t) + V_{t+1}(S_{t+1}) \right)$$

  - » The essential idea is that you can make the best decision now (out of state S_t) given the contribution of decision x_t, given by $C(S_t, x_t)$, plus the optimal value of the state S_{t+1} that the decision x_t takes us to.

Discrete Markov decision processes

- The optimality equations for stochastic problems:
  - » We start with our objective:

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

  - » An optimal policy finds the best decision given the optimal value of the state that the decision takes us to:

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

  - » We then replace the downstream value of starting in state S_{t+1} with a value function:

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

  - » The "optimality equations" are then

Discrete Markov decision processes

- Bellman for stochastic problems

When we are solving stochastic problems, we have to model the fact that new information becomes available after we make the decision a_t. The result can be uncertainty in both the contribution earned, and in the determination of the next state we visit, S_{t+1}. For example, consider the problem of managing oil inventories for a refinery. Let the state S_t be the inventory in thousands of barrels of oil at time t (we require S_t to be integer). Let a_t be the amount of oil ordered at time t that will be available for use between t and $t+1$, and let $\hat{D}_{t+1}$ be the demand for oil between t and $t+1$. The state variable is governed by the simple inventory equation

$$S_{t+1}(S_t, a_t, \hat{D}_{t+1}) = \max\{0, S_t + a_t - \hat{D}_{t+1}\}.$$

We have written the state S_{t+1} using $S_{t+1}(S_t, a_t)$ to express the dependence on S_t and a_t, but it is common to simply write S_{t+1} and let the dependence on S_t and a_t be implicit. Since $\hat{D}_{t+1}$ is random at time t when we have to choose a_t, we do not know S_{t+1}. But if we know the probability distribution of the demand $\hat{D}$, we can work out the probability that S_{t+1} will take on a particular value. If $\mathbb{P}^D(d) = \mathbb{P}[\hat{D} = d]$ is our probability distribution, then we can find the probability distribution for S_{t+1} using

$$Prob(S_{t+1} = s') = \begin{cases} 0 & \text{if } s' > S_t + a_t, \\ \mathbb{P}^D(S_t + a_t - s') & \text{if } 0 < s' \leq S_t + a_t, \\ \sum_{d=S_t+a_t}^{\infty} \mathbb{P}^D(d) & \text{if } s' = 0. \end{cases}$$

These probabilities depend on S_t and a_t, so we write the probability distribution as

$$\mathbb{P}(S_{t+1}|S_t, a_t) = \text{The probability of } S_{t+1} \text{ given } S_t \text{ and } a_t.$$

Discrete Markov decision processes

We can then modify the deterministic optimality equation in (14.2) by simply adding an expectation, giving us

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \left(C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(S_{t+1} = s' | S_t, a_t) V_{t+1}(s')\right). \quad (14.3)$$

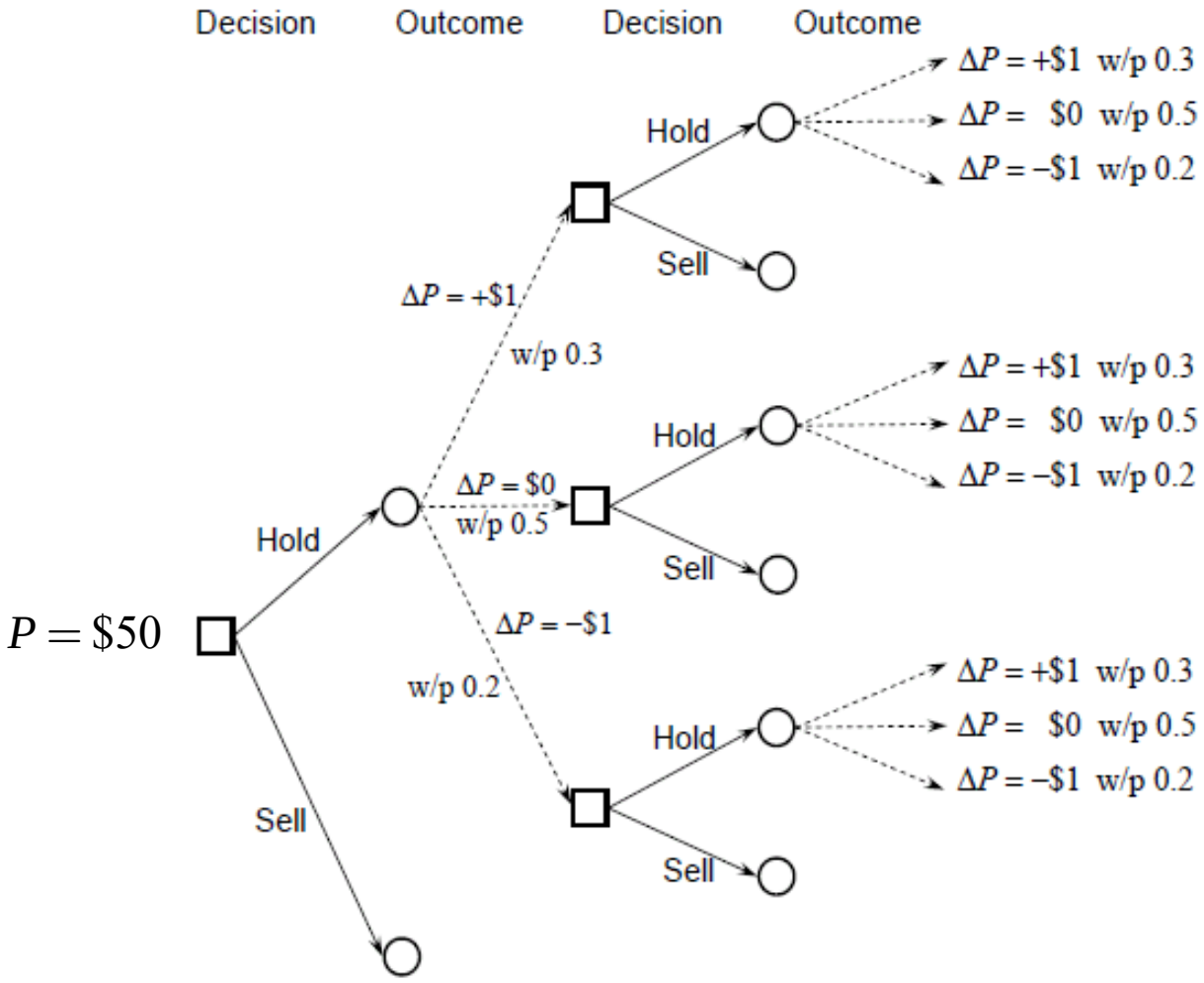
We refer to this as the *standard form* of Bellman's equations, since this is the version that is used by virtually every textbook on stochastic, dynamic programming. An equivalent form that is more natural for approximate dynamic programming is to write

$$V_t(S_t) = \max_{a_t \in \mathcal{A}_t} \left(C_t(S_t, a_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}(S_t, a_t, W_{t+1})) | S_t\}\right), \quad (14.4)$$

where we simply use an expectation instead of summing over probabilities. We refer to this equation as the *expectation form* of Bellman's equation. This version forms the basis for our algorithmic work in later chapters.

Discrete Markov decision processes

- Decision tree without learning

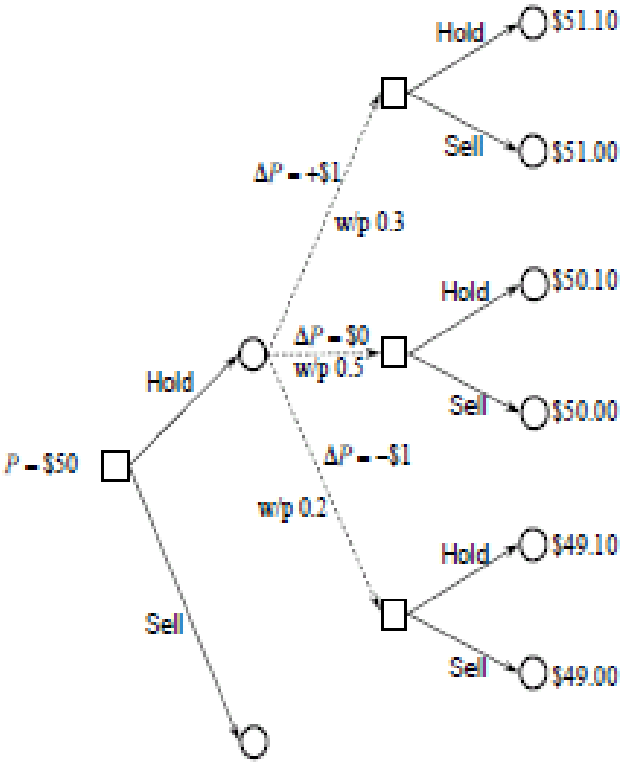


Discrete Markov decision processes

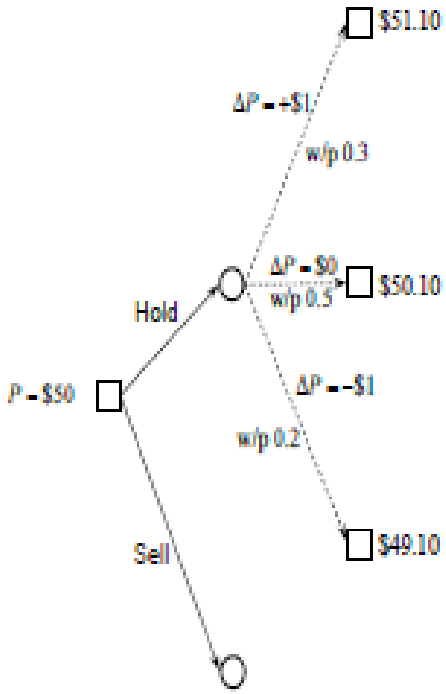
- Some vocabulary:
  - » Square nodes are:
    - Decision nodes – Links emanating from the squares are decisions
    - Also represents the state of our system (it implicitly captures all the information we need to make a decision).
    - Also known as the "pre-decision state."
  - » Circle nodes are:
    - 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.

Discrete Markov decision processes

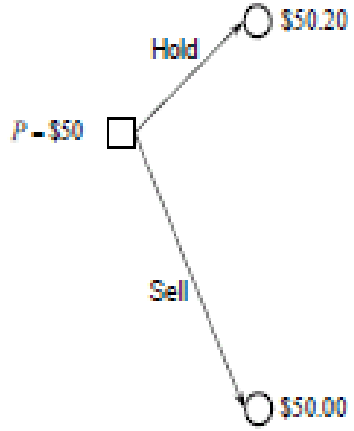
- Rolling back the tree



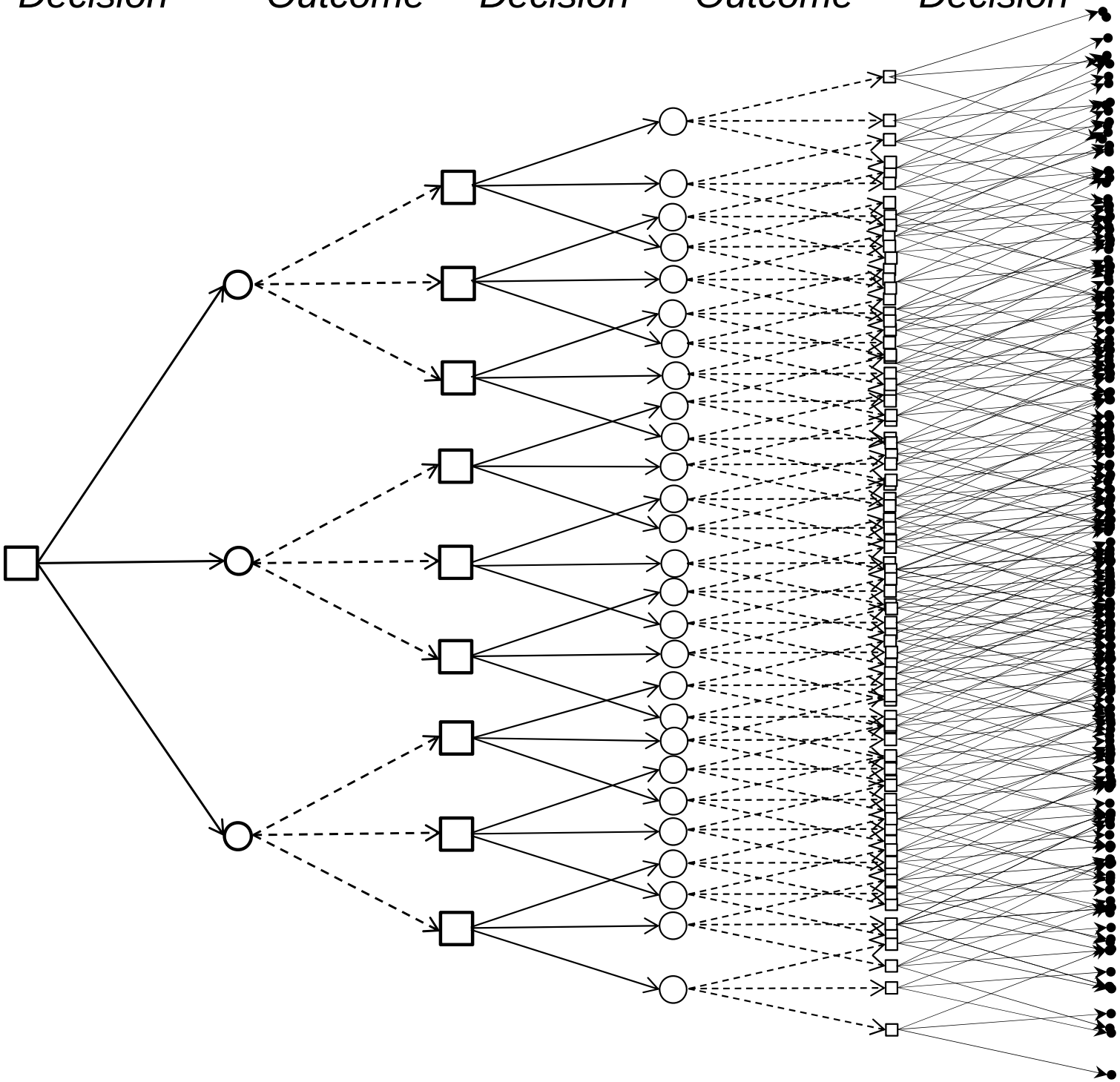
1.7(a)



1.7(b)

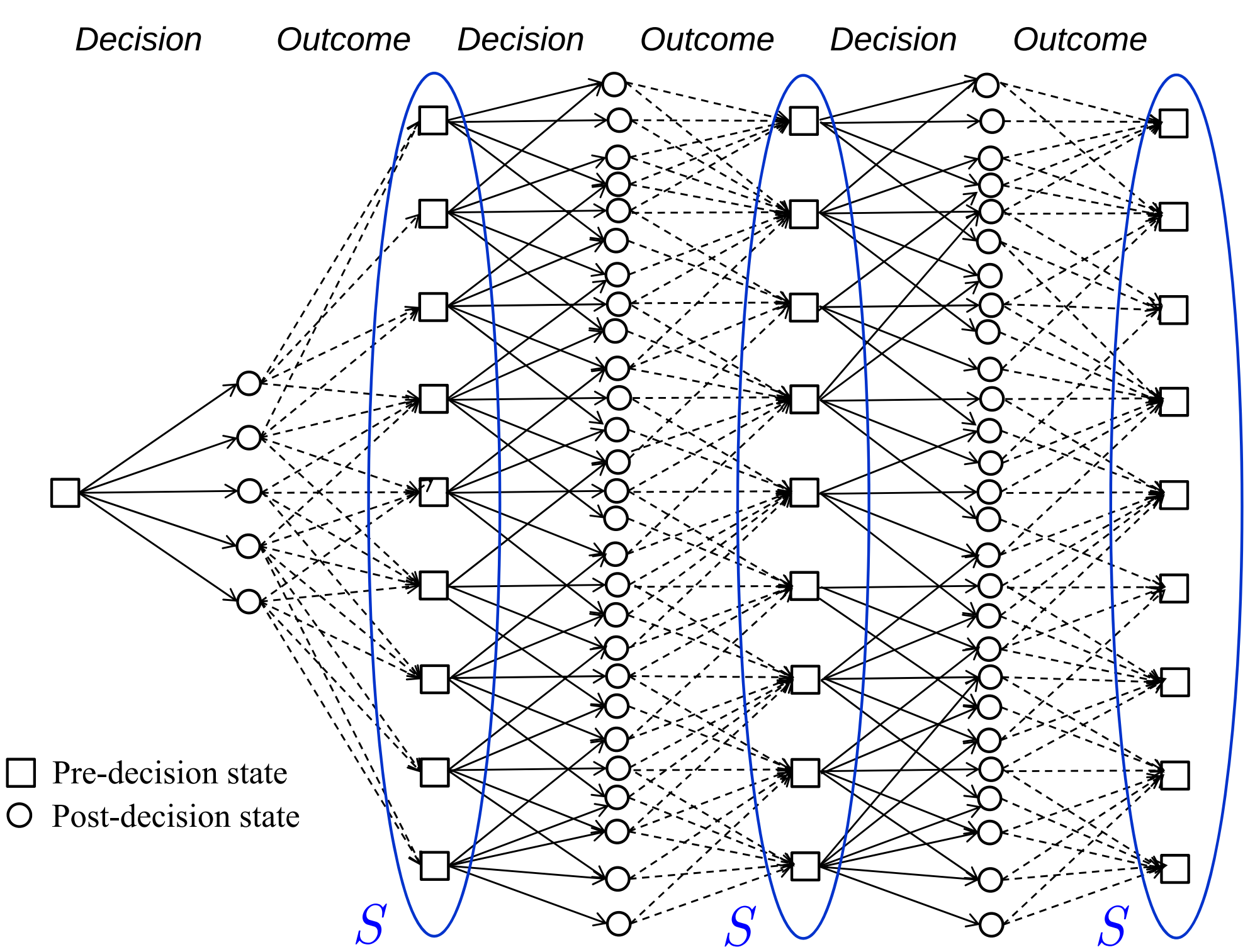


1.7(c)



Discrete Markov decision processes

- 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.
  - » This is how sequential decisions were approached before Richard Bellman.



Discrete Markov decision processes

- Notes:
  - » "Dynamic programming" (or more precisely, discrete Markov decision processes) exploits the property that there is (or may be) a compact set of states that all paths return to.
  - » Capturing this avoids the explosion that happens with decision trees.

Discrete Markov decision processes

- Optimality equations in matrix form

We would say that "$p_{ss'}(a)$ is the probability that we end up in state s' if we start in state s at time t when we are taking action a." Now assume that we have a function $A_t^\pi(s)$ that determines the action a we should take when in state s. It is common to write the transition probability $p_{ss'}(a)$ in the form

$$p_{ss'}^\pi = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t^\pi(s) = a).$$

We can now write this in matrix form

$$P_t^\pi = \text{The one-step transition matrix under policy } \pi,$$

where $p_{ss'}^\pi$ is the element in row s and column s'. There is a different matrix P^π for each policy (decision rule) π.

Now let c_t^π be a column vector with element $c_t^\pi(s) = C_t(s, A_t^\pi(s))$, and let v_{t+1} be a column vector with element $V_{t+1}(s)$. Then (14.3) is equivalent to

$$\begin{bmatrix} \vdots \\ v_t(s) \\ \vdots \end{bmatrix} = \max_\pi \left(\begin{bmatrix} \vdots \\ c_t^\pi(s) \\ \vdots \end{bmatrix} + \gamma \begin{bmatrix} \ddots & & \\ & p_{ss'}^\pi & \\ & & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ v_{t+1}(s') \\ \vdots \end{bmatrix} \right). \quad (14.7)$$

where the maximization is performed for each element (state) in the vector. In matrix/vector form, equation (14.7) can be written

$$v_t = \max_\pi \left(c_t^\pi + \gamma P_t^\pi v_{t+1} \right). \quad (14.8)$$

Discrete Markov decision processes

dynamic programming. Instead, it seems much more natural to calculate V_t^π recursively using

$$V_t^\pi(S_t) = C_t(S_t, A_t^\pi(S_t)) + \mathbb{E}\left\{V_{t+1}^\pi(S_{t+1})|S_t\right\}.$$

It is not hard to show (by stepping backward in time) that

$$F_t^\pi(S_t) = V_t^\pi(S_t).$$

The proof, given in section 14.10.1, uses a proof by induction: assume it is true for V_{t+1}^π, and then show that it is true for V_t^π (not surprisingly, inductive proofs are very popular in dynamic programming).

With this result in hand, we can then establish the following key result. Let $V_t(S_t)$ be a solution to equation (14.4) (or (14.3)). Then

$$\begin{aligned} F_t^* &= \max_{\pi\in\Pi} F_t^\pi(S_t) \\ &= V_t(S_t). \end{aligned} \tag{14.9}$$

Equation (14.9) establishes the equivalence between (a) the value of being in state S_t and following the optimal policy and (b) the optimal value function at state S_t. While these are indeed equivalent, the equivalence is the result of a theorem (established in section 14.10.1). However, it is not unusual to find people who lose sight of the original objective function. Later, we have to solve these equations approximately, and we will need to use the original objective function to evaluate the quality of a solution.

Discrete Markov decision processes

- Computing the one-step transition matrix
  - » The MDP community treats the one-step transition matrix as data, but it is usually computationally intractable.

Assume that the random information W_{t+1} that arrives between t and $t+1$ is independent of all prior information. Let Ω_{t+1} be the set of possible outcomes of W_{t+1} (for simplicity, we assume that Ω_{t+1} is discrete), where $\mathbb{P}(W_{t+1} = \omega_{t+1})$ is the probability of outcome $\omega_{t+1} \in \Omega_{t+1}$. Also define the indicator function

$$1_{\{X\}} = \begin{cases} 1 & \text{if the statement "}X\text{" is true.} \\ 0 & \text{otherwise.} \end{cases}$$

Here, "X" represents a logical condition (such as, "is $S_t = 6$?"). We now observe that the one-step transition probability $\mathbb{P}_t(S_{t+1}|S_t, a_t)$ can be written

$$\begin{aligned} \mathbb{P}_t(S_{t+1}|S_t, a_t) &= \mathbb{E}1_{\{s'=S^M(S_t,a_t,W_{t+1})\}} \\ &= \sum_{\omega_{t+1}\in\Omega_{t+1}} \mathbb{P}(\omega_{t+1})1_{\{s'=S^M(S_t,a_t,\omega_{t+1})\}} \end{aligned}$$

So, finding the one-step transition matrix means that all we have to do is to sum over all possible outcomes of the information W_{t+1} and add up the probabilities that take us from a particular state-action pair (S_t, a_t) to a particular state $S_{t+1} = s'$. Sounds easy.

Discrete Markov decision processes

- Bellman's equations using operator notation

The vector form of Bellman's equation in (14.8) can be written even more compactly using operator notation. Let $\mathcal{M}$ be the "max" (or "min") operator in (14.8) that can be viewed as acting on the vector v_{t+1} to produce the vector v_t. If we have a given policy π, we can write

$$\mathcal{M}^{\pi}v(s) = C_t(s, A^{\pi}(s)) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}_t(s'|s, A^{\pi}(s))v_{t+1}(s').$$

Alternatively, we can find the best action, which we represent using

$$\mathcal{M}v(s) = \max_a \left(C_t(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}_t(s'|s, a)v_{t+1}(s')\right).$$

Here, $\mathcal{M}v$ produces a vector, and $\mathcal{M}v(s)$ refers to element s of this vector. In vector form, we would write

$$\mathcal{M}v = \max_{\pi} \left(c_t^{\pi} + \gamma P_t^{\pi} v_{t+1}\right).$$

Discrete Markov decision processes

- Bellman's equations using operator notation

Now let $\mathcal{V}$ be the space of value functions. Then, $\mathcal{M}$ is a mapping

$$\mathcal{M} : \mathcal{V} \rightarrow \mathcal{V}.$$

We may also define the operator $\mathcal{M}^\pi$ for a particular policy π using

$$\mathcal{M}^\pi(v) = c_t^\pi + \gamma P^\pi v \tag{14.11}$$

for some vector $v \in \mathcal{V}$. $\mathcal{M}^\pi$ is known as a *linear operator* since the operations that it performs on v are additive and multiplicative. In mathematics, the function $c_t^\pi + \gamma P^\pi v$ is known as an *affine function*. This notation is particularly useful in mathematical proofs (see in particular some of the proofs in section 14.10), but we will not use this notation when we describe models and algorithms.

Discrete Markov decision processes

Backward dynamic programming

Value iteration

Policy iteration

Average reward

Backward dynamic programming

- Finite horizon
  - » Backward MDP

Step 0. Initialization:

Initialize the terminal contribution $V_T(S_T)$.

Set $t = T - 1$.

Step 1. Calculate:

$$V_t(S_t) = \max_{a_t} \left\{ C_t(S_t, a_t) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|S_t, a_t) V_{t+1}(s') \right\}$$

for all $S_t \in \mathcal{S}$.

Step 2. If $t > 0$, decrement t and return to step 1. Else, stop.

Figure 14.3 A backward dynamic programming algorithm.

  - » So, if we can compute the one-step transition matrix, this gives us the optimal solution.

Backward dynamic programming

- Backward dynamic programming in one dimension

Step 0: Initialize $V_{T+1}(S_{T+1}) = 0$ for $S_{T+1} \in S$

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

Step 2: Loop over $S_t \in S$

Step 3: Loop over all actions $a \in A_s$

Step 4: Take the expectation over exogenous information:

$$\text{Compute } Q(S_t, a_t) = C(S_t, a_t) + \sum_{w=0}^{100} V_{t+1}(S_{t+1} = S^M(S_t, a_t, w) P^W(w)$$

End step 4;

End Step 3;

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

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

End Step 2;

End Step 1;

Backward dynamic programming

- Notes:
  - » If you can compute the one-step transition matrix $p(s'|s,a)$, then backward dynamic programming for a finite horizon is trivial.
  - » The problem is that the matrix $p(s'|s,a)$ is rarely computable:
    - State variables may be vector valued and/or continuous
    - Actions may be vector valued and/or continuous
    - The exogenous information W (implicit in the one-step transition matrix) may be vector valued and/or continuous.
  - » The MDP community transitioned to infinite horizon problems that dominate the field.

Backward dynamic programming

- Infinite horizon MDP

We begin with the optimality equations

$$V_t(S_t) = \max_{a_t \in \mathcal{A}} \mathbb{E}\left\{C_t(S_t, a_t) + \gamma V_{t+1}(S_{t+1}) | S_t\right\}.$$

We can think of a steady-state problem as one without the time dimension. Letting $V(s) = \lim_{t\to\infty} V_t(S_t)$ (and assuming the limit exists), we obtain the steady-state optimality equations

$$V(s) = \max_{a\in\mathcal{A}} \left\{C(s,a) + \gamma \sum_{s'\in\mathcal{S}} \mathbb{P}(s'|s,a)V(s')\right\}. \tag{14.12}$$

The functions $V(s)$ can be shown (as we do later) to be equivalent to solving the infinite horizon problem

$$\max_{\pi\in\Pi} \mathbb{E}\left\{\sum_{t=0}^{\infty} \gamma^t C_t(S_t, A_t^\pi(S_t))\right\}. \tag{14.13}$$

Now define

$$\begin{aligned} P^{\pi,t} &= t\text{-step transition matrix, over periods } 0, 1, \ldots, t-1, \text{ given policy } \pi \\ &= \Pi_{t'=0}^{t-1} P_{t'}^{\pi}. \end{aligned} \tag{14.14}$$

Backward dynamic programming

We further define $P^{\pi,0}$ to be the identity matrix. As before, let c_t^π be the column vector of the expected cost of being in each state given that we choose the action a_t described by policy π, where the element for state s is $c_t^\pi(s) = C_t(s, A^\pi(s))$. The infinite horizon, discounted value of a policy π starting at time t is given by

$$v_t^\pi = \sum_{t'=t}^{\infty} \gamma^{t'-t} P^{\pi,t'-t} c_{t'}^\pi. \tag{14.15}$$

Assume that after following policy π_0 we follow policy $\pi_1 = \pi_2 = \ldots = \pi$. In this case, equation (14.15) can now be written as (starting at $t = 0$)

$$\begin{aligned}
v^{\pi_0} &= c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} P^{\pi,t'} c_{t'}^\pi && (14.16)\\
&= c^{\pi_0} + \sum_{t'=1}^{\infty} \gamma^{t'} \left(\Pi_{t''=0}^{t'-1} P_{t''}^\pi\right) c_{t'}^\pi && (14.17)\\
&= c^{\pi_0} + \gamma P^{\pi_0} \sum_{t'=1}^{\infty} \gamma^{t'-1} \left(\Pi_{t''=1}^{t'-1} P_{t''}^\pi\right) c_{t'}^\pi && (14.18)\\
&= c^{\pi_0} + \gamma P^{\pi_0} v^\pi. && (14.19)
\end{aligned}$$

Backward dynamic programming

Equation (14.19) shows us that the value of a policy is the single period reward plus a discounted terminal reward that is the same as the value of a policy starting at time 1. If our decision rule is stationary, then $\pi_0 = \pi_1 = \ldots = \pi_t = \pi$, which allows us to rewrite (14.19) as

$$v^\pi = c^\pi + \gamma P^\pi v^\pi. \tag{14.20}$$

This allows us to solve for the stationary reward explicitly (as long as $0 \leq \gamma < 1$), giving us

$$v^\pi = (I - \gamma P^\pi)^{-1} c^\pi.$$

We can also write an infinite horizon version of the optimality equations using our operator notation. Letting $\mathcal{M}$ be the "max" (or "min") operator (also known as the Bellman operator), the infinite horizon version of equation (14.11) would be written

$$\mathcal{M}^\pi(v) = c^\pi + \gamma P^\pi v. \tag{14.21}$$

Backward dynamic programming

14.4 VALUE ITERATION

Value iteration is perhaps the most widely used algorithm in dynamic programming because it is the simplest to implement and, as a result, often tends to be the most natural way of solving many problems. It is virtually identical to backward dynamic programming for finite horizon problems. In addition, most of our work in approximate dynamic programming is based on value iteration.

Value iteration comes in several flavors. The basic version of the value iteration algorithm is given in figure 14.4. The proof of convergence (see section 14.10.2) is quite elegant for students who enjoy mathematics. The algorithm also has several nice properties that we explore below.

It is easy to see that the value iteration algorithm is similar to the backward dynamic programming algorithm. Rather than using a subscript t, which we decrement from T back to 0, we use an iteration counter n that starts at 0 and increases until we satisfy a convergence criterion. Here, we stop the algorithm when

$$\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma,$$

where $\|v\|$ is the max-norm defined by

$$\|v\| = \max_s |v(s)|.$$

Backward dynamic programming

Step 0. Initialization:

Set $v^0(s) = 0 \ \forall s \in \mathcal{S}$.

Fix a tolerance parameter $\epsilon > 0$.

Set $n = 1$.

Step 1. For each $s \in \mathcal{S}$ compute:

$$v^n(s) = \max_{a \in \mathcal{A}} \left(C(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s,a) v^{n-1}(s') \right). \tag{14.22}$$

Step 2. If $\|v^n - v^{n-1}\| < \epsilon(1-\gamma)/2\gamma$, let π^ϵ be the resulting policy that solves (14.22), and let $v^\epsilon = v^n$ and stop; else set $n = n + 1$ and go to step 1.

Figure 14.4 The value iteration algorithm for infinite horizon optimization

Thus, $\|v\|$ is the largest absolute value of a vector of elements. Thus, we stop if the largest change in the value of being in any state is less than $\epsilon(1-\gamma)/2\gamma$ where ϵ is a specified error tolerance.

Below, we describe a Gauss-Seidel variant which is a useful method for accelerating value iteration, and a version known as relative value iteration.

Backward dynamic programming

14.4.3 Bounds and rates of convergence

One important property of value iteration algorithms is that if our initial estimate is too low, the algorithm will rise to the correct value from below. Similarly, if our initial estimate is too high, the algorithm will approach the correct value from above. This property is formalized in the following theorem:

Theorem 14.4.1 *For a vector $v \in \mathcal{V}$:*

(a) If v satisfies $v \geq \mathcal{M}v$, then $v \geq v^*$.

(b) If v satisfies $v \leq \mathcal{M}v$, then $v \leq v^*$.

(c) If v satisfies $v = \mathcal{M}v$, then v is the unique solution to this system of equations and $v = v^*$.

Theorem 14.4.2 *If we apply the value iteration algorithm with stopping parameter ϵ and the algorithm terminates at iteration n with value function v^{n+1}, then*

$$\|v^{n+1} - v^*\| \leq \epsilon/2. \tag{14.24}$$

Let π^ϵ be the policy that we terminate with, and let v^{π^ϵ} be the value of this policy. Then

$$\|v^{\pi^\epsilon} - v^*\| \leq \epsilon.$$

We can provide some additional insights into the bound, as well as the rate of convergence, by considering a trivial dynamic program. In this problem, we receive a constant reward c at every iteration. There are no decisions, and there is no randomness. The value of this "game" is quickly seen to be

$$\begin{aligned} v^* &= \sum_{n=0}^{\infty} \gamma^n c \\ &= \frac{1}{1-\gamma} c. \end{aligned} \tag{14.25}$$

Consider what happens when we solve this problem using value iteration. Starting with $v^0 = 0$, we would use the iteration

$$v^n = c + \gamma v^{n-1}.$$

After we have repeated this n times, we have

$$\begin{aligned} v^n &= \sum_{m=0}^{n-1} \gamma^n c \\ &= \frac{1-\gamma^n}{1-\gamma} c. \end{aligned} \tag{14.26}$$

Comparing equations (14.25) and (14.26), we see that

$$v^n - v^* = -\frac{\gamma^n}{1-\gamma} c. \tag{14.27}$$

Backward dynamic programming

Similarly, the change in the value from one iteration to the next is given by

$$\begin{aligned}\|v^{n+1} - v^n\| &= \left|\frac{\gamma^{n+1}}{1-\gamma} - \frac{\gamma^n}{1-\gamma}\right| c \\ &= \gamma^n \left|\frac{\gamma}{1-\gamma} - \frac{1}{1-\gamma}\right| c \\ &= \gamma^n \left|\frac{\gamma - 1}{1-\gamma}\right| c \\ &= \gamma^n c.\end{aligned}$$

If we stop at iteration $n+1$, then it means that

$$\gamma^n c \leq \epsilon/2 \left(\frac{1-\gamma}{\gamma}\right). \tag{14.28}$$

If we choose ϵ so that (14.28) holds with equality, then our error bound (from 14.24) is

$$\begin{aligned}\|v^{n+1} - v^*\| &\leq \epsilon/2 \\ &= \frac{\gamma^{n+1}}{1-\gamma} c.\end{aligned}$$

From (14.27), we know that the distance to the optimal solution is

$$|v^{n+1} - v^*| = \frac{\gamma^{n+1}}{1-\gamma} c,$$

Backward dynamic programming

- Policy iteration

Step 0. Initialization:

Step 0a. Select a policy π^0.

Step 0b. Set $n = 1$.

Step 1. Given a policy π^{n-1}:

Step 1a. Compute the one-step transition matrix $P^{\pi^{n-1}}$.

Step 1b Compute the contribution vector $c^{\pi^{n-1}}$ where the element for state s is given by $c^{\pi^{n-1}}(s) = C(s, A^{\pi^{n-1}})$.

Step 2. Let $v^{\pi,n}$ be the solution to

$$(I - \gamma P^{\pi^{n-1}})v = c^{\pi^{n-1}}.$$

Step 3. Find a policy π^n defined by

$$a^n(s) = \arg\max_{a \in \mathcal{A}} \left(C(a) + \gamma P^{\pi} v^n\right).$$

This requires that we compute an action for each state s.

Step 4. If $a^n(s) = a^{n-1}(s)$ for all states s, then set $a^* = a^n$; otherwise, set $n = n + 1$ and go to step 1.

Figure 14.7 Policy iteration

Backward dynamic programming

14.7 AVERAGE REWARD DYNAMIC PROGRAMMING

There are settings where the natural objective function is to maximize the *average* contribution per unit time. Assume we start in state s. Then, the average reward from starting in state s and following policy π is given by

$$\max_{\pi} F^{\pi}(s) = \lim_{T\rightarrow\infty} \frac{1}{T}\mathbb{E}\sum_{t=0}^{T} C(S_t, A^{\pi}(S_t)). \tag{14.32}$$

Here, $F^{\pi}(s)$ is the expected reward *per time period*. In matrix form, the total value of following a policy π over a horizon T can be written as

$$V_T^{\pi} = \sum_{t=0}^{T} (P^{\pi})^t c^{\pi},$$

where V_T^{π} is a column vector with element $V_T^{\pi}(s)$ giving the expected contribution over T time periods when starting in state s. We can get a sense of how $V_T^{\pi}(s)$ behaves by watching what happens as T becomes large. Assuming that our underlying Markov chain is ergodic (all the states communicate with each other with positive probability), we know that $(P^{\pi})^T \rightarrow P^*$ where the rows of P^* are all the same.

Now define a column vector g given by

$$g^{\pi} = P^* c^{\pi}.$$

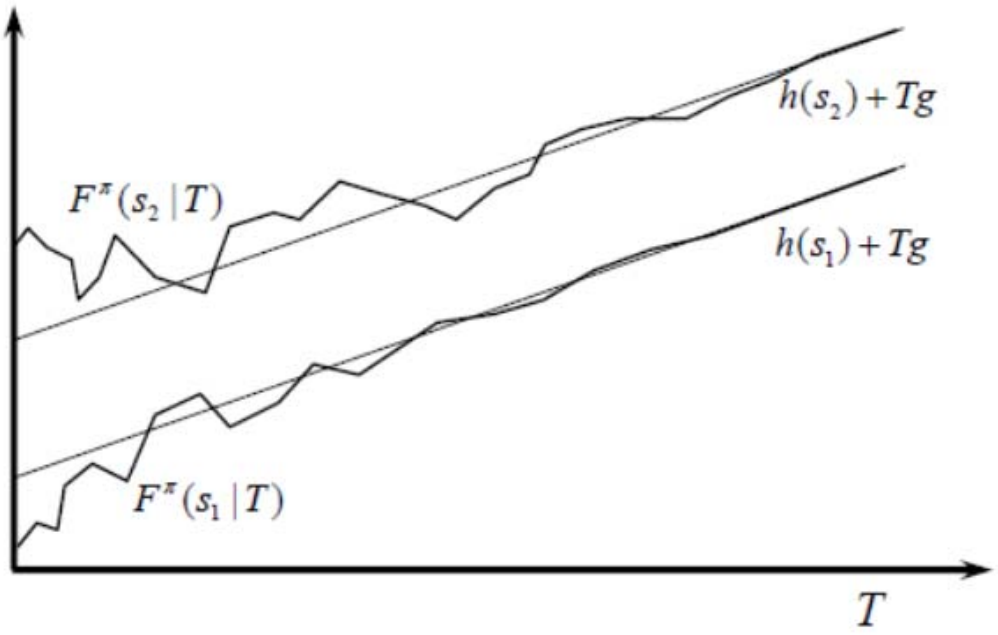


Figure 14.9 Cumulative contribution over a horizon T when starting in states s_1 and s_2, showing growth approaching a rate that is independent of the starting state.

Since the rows of P^* are all the same, all the elements of g^π are the same, and each element gives the average contribution per time period using the steady state probability of being in each state. For finite T, each element of the column vector V_T^π is not the same, since the contributions we earn in the first few time periods depends on our starting state. But it is not hard to see that as T grows large, we can write

$$V_T^\pi \rightarrow h^\pi + Tg^\pi,$$

where h^π captures the state-dependent differences in the total contribution, while g^π is the state-independent average contribution in the limit. Figure 14.9 illustrates the growth in V_T^π toward a linear function.

Backward MDP and the curse of dimensionality

Curse of dimensionality

- The ultimate policy is to optimize from now on:

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

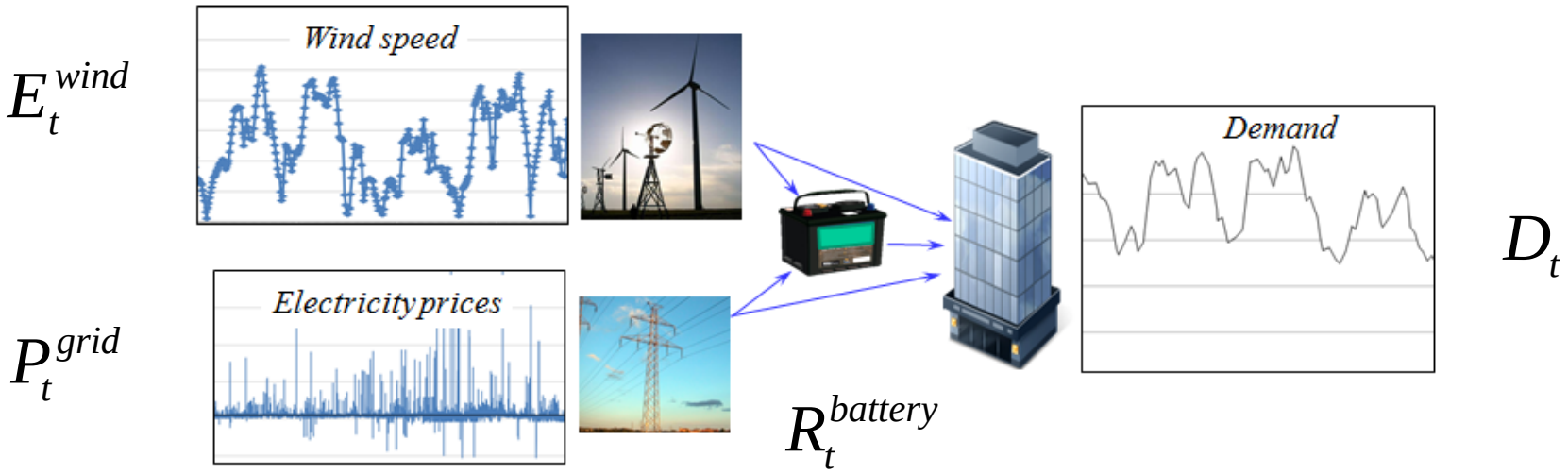
- Ideally, we would like to replace the future contributions with a single value function:

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

- *Sometimes* we can compute this function exactly!

Curse of dimensionality

- Energy storage with stochastic prices, supplies and demands.



$$E_{t+1}^{wind} = E_t^{wind} + \hat{E}_{t+1}^{wind}$$
$$P_{t+1}^{grid} = P_t^{grid} + \hat{P}_{t+1}^{grid}$$
$$D_{t+1}^{load} = D_t^{load} + \hat{D}_{t+1}^{load}$$
$$R_{t+1}^{battery} = R_t^{battery} + Ax_t$$

W_{t+1} = Exogenous inputs

S_t = State variable

x_t = Controllable inputs

Curse of dimensionality

- Bellman's optimality equation

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

$$\begin{bmatrix} E_t^{wind} \\ P_t^{grid} \\ D_t^{load} \\ R_t^{battery} \end{bmatrix}$$

$$\begin{bmatrix} x_t^{wind-battery} \\ x_t^{wind-load} \\ x_t^{grid-battery} \\ x_t^{grid-load} \\ x_t^{battery-load} \end{bmatrix}$$

$$\begin{bmatrix} \hat{E}_{t+1}^{wind} \\ \hat{P}_{t+1}^{grid} \\ \hat{D}_{t+1}^{load} \end{bmatrix}$$

These are the "three curses of dimensionality."

Curse of dimensionality

- 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;

Curse of dimensionality

- 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;

Curse of dimensionality

- 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.

Curse of dimensionality

- 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
    - To be covered next week