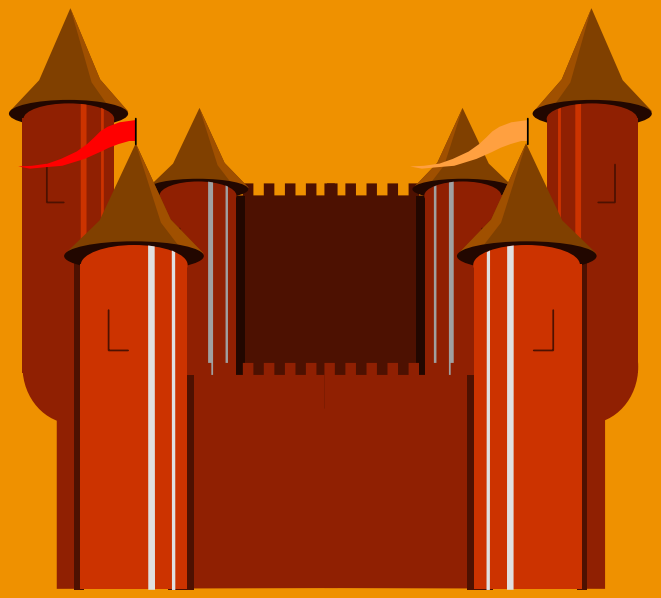
ORF 544

Stochastic Optimization and Learning

Spring, 2019



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

Week 11

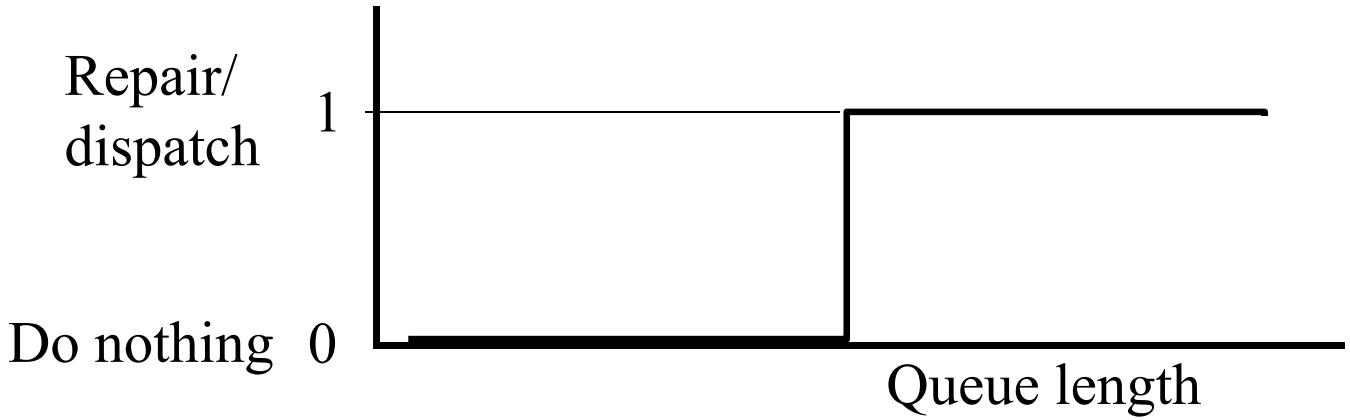
Forward approximate dynamic programming

Approximate dynamic programming

Exploiting monotonicity

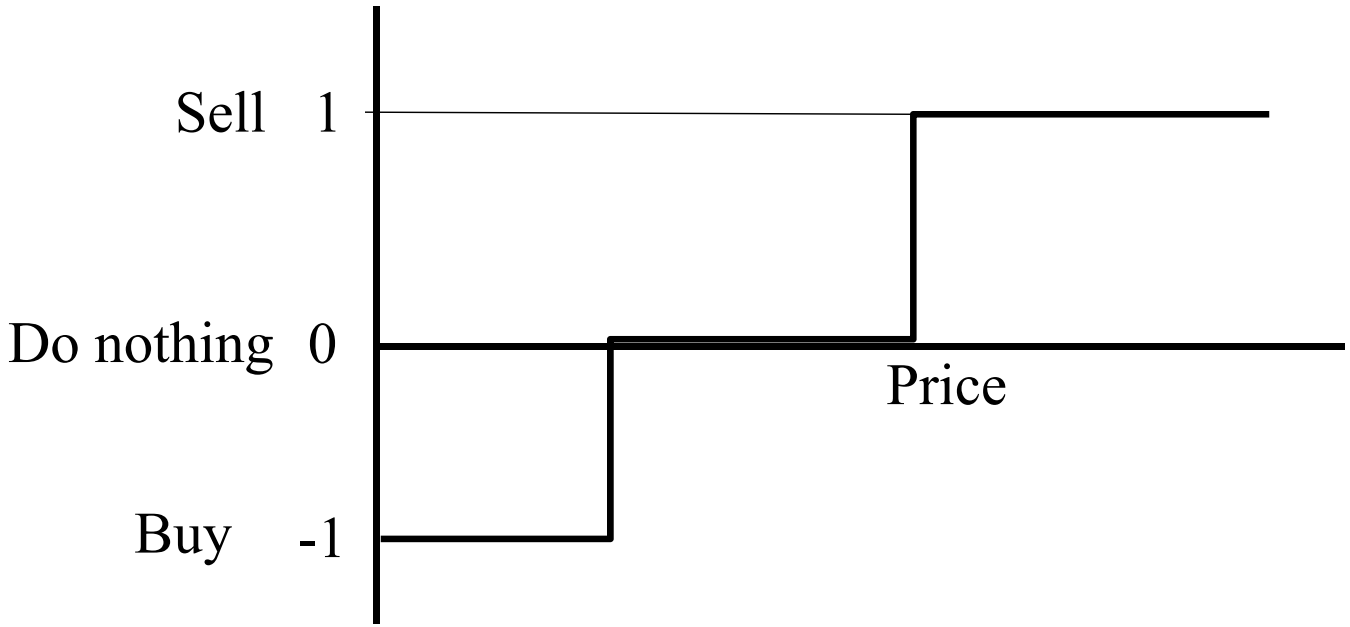
Exploiting monotonicity

- Monotonicity in dynamic programming
  - » There are many problems where the value function increases (or decreases) monotonically with all dimensions of a state variable:
  - » Operations research
    - Optimal equipment replacement – Replace when a measure of deterioration exceeds some point
    - Dispatching customers in a queue – Costs increase monotonically with the number of customers in the queue. Dispatch when the queue is over some number.



Exploiting monotonicity

- Monotonicity in dynamic programming (cont'd)
  - » Energy
    - Buy when electricity price is below one number, sell when above another number:



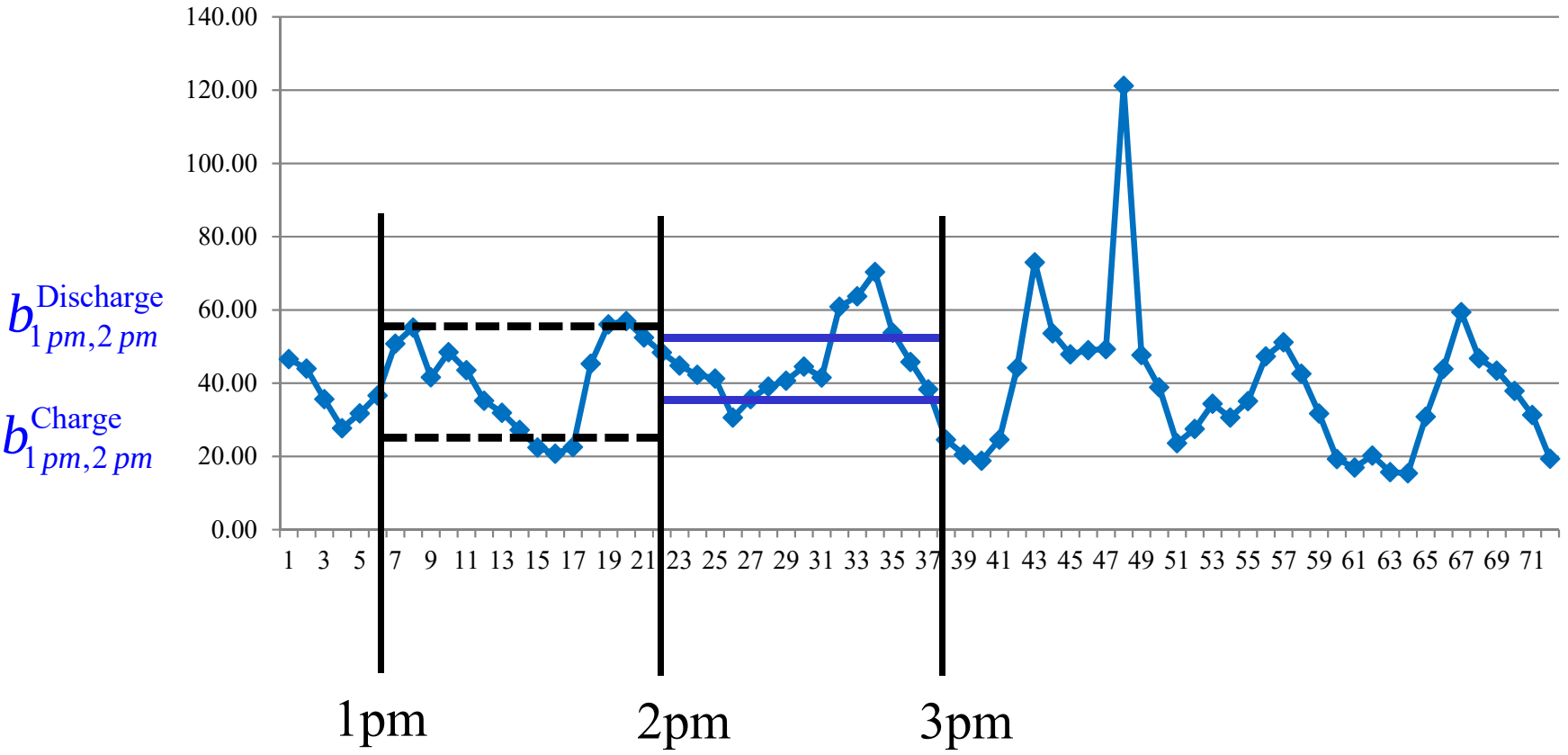
    - Value of energy in storage increases with amount being held (and perhaps price, speed of wind, demand, …)

Exploiting monotonicity

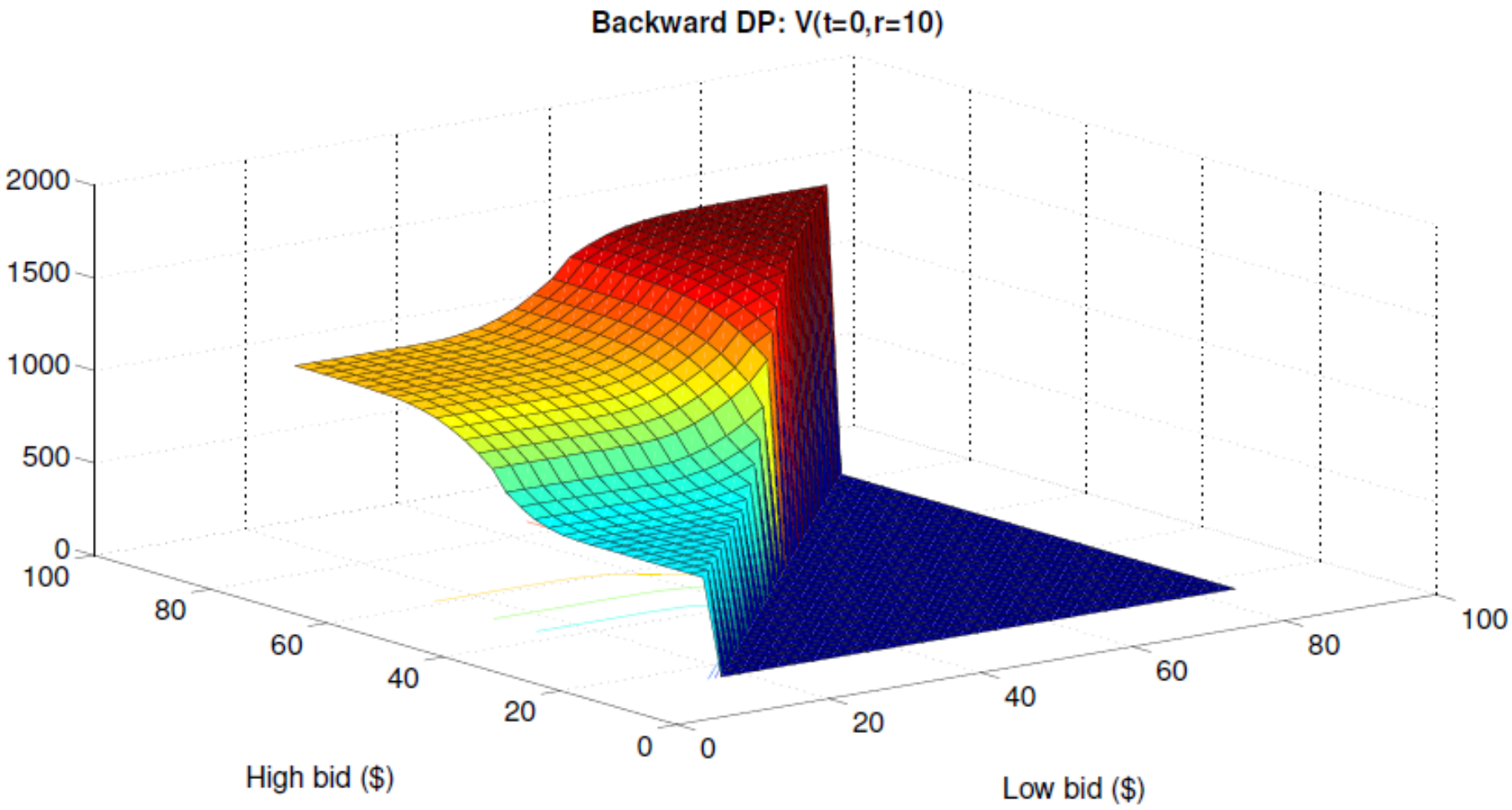
- Monotonicity in dynamic programming (cont'd)
  - » Health
    - Dosage of a diabetes drug increases with blood sugar.
    - Dosage of statins (for reducing cholesterol) increase as the cholesterol level increases (and also increases with age, weight).
  - » Finance
    - The value of holding cash in a mutual fund increases with the amount of redemptions, stock indices, and interest rates.
    - The value of holding an option increases with price and volatility.

Exploiting monotonicity

- Bid is placed at 1pm, consisting of charge and discharge prices between 2pm and 3pm.



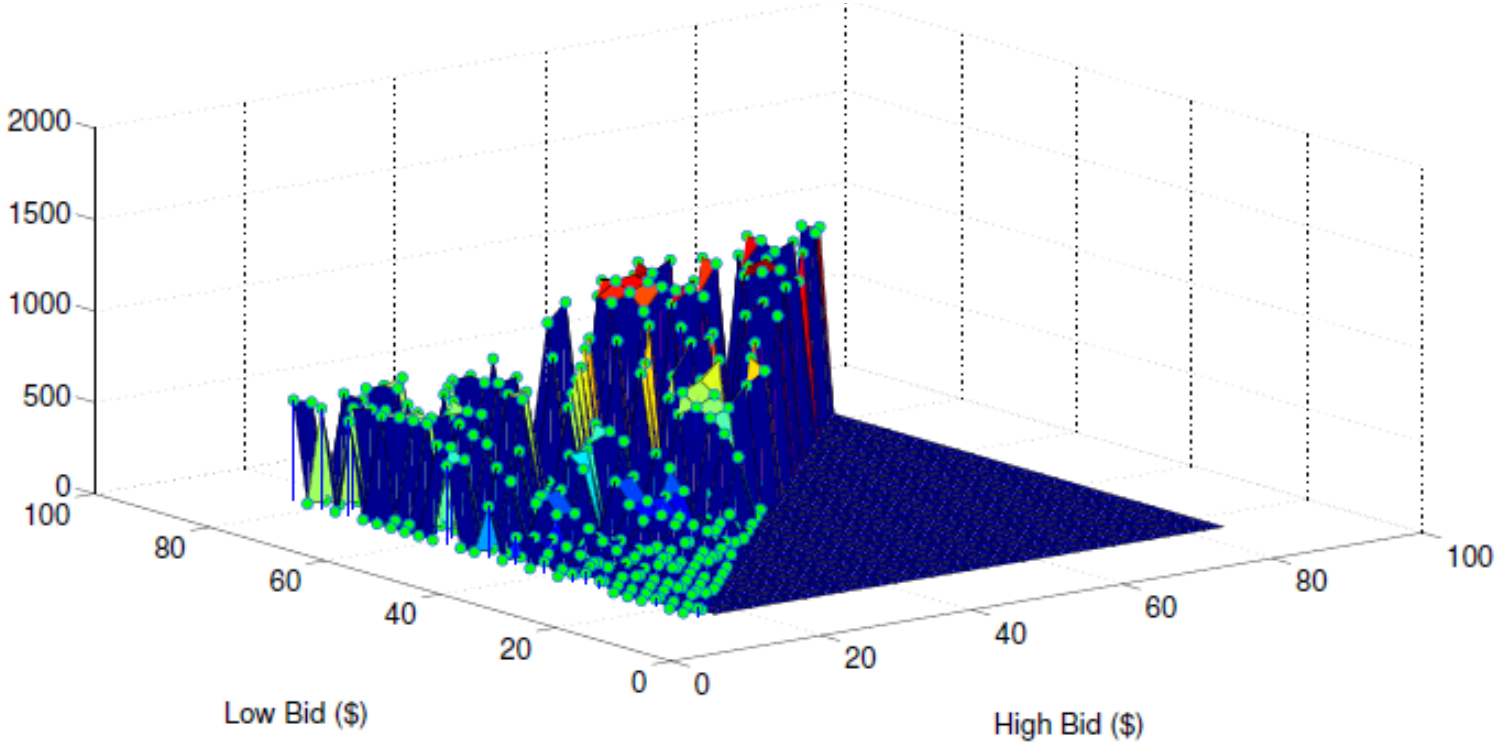
Exploiting monotonicity



The exact value function

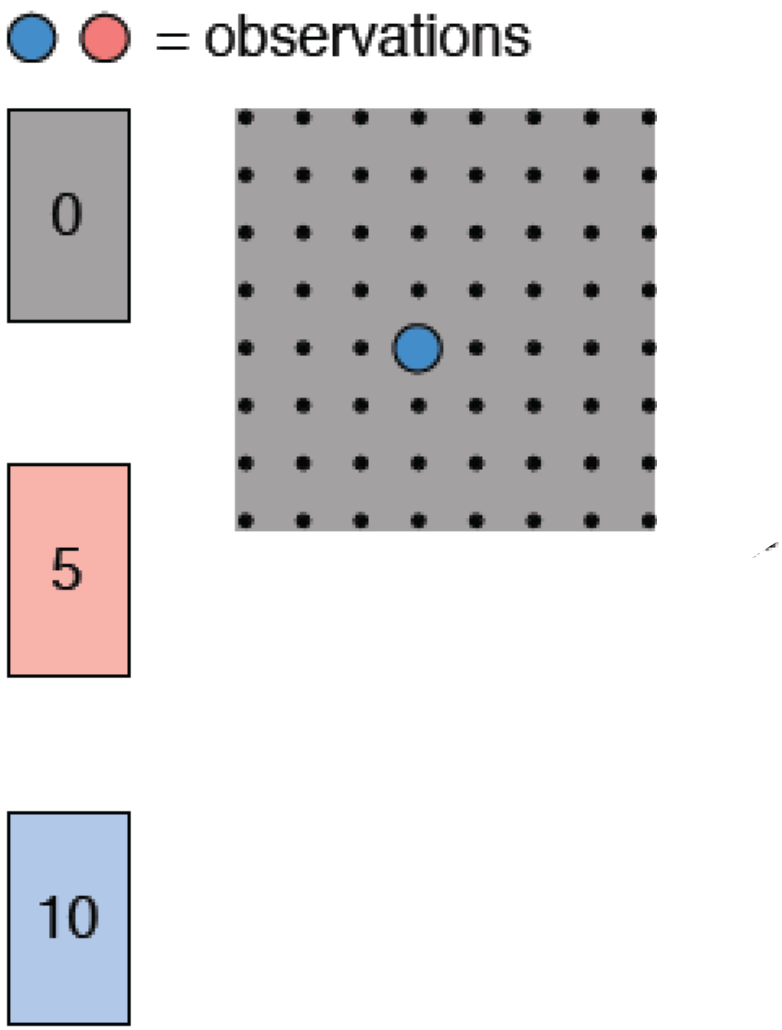
Exploiting monotonicity

Approximate value function without monotonicity



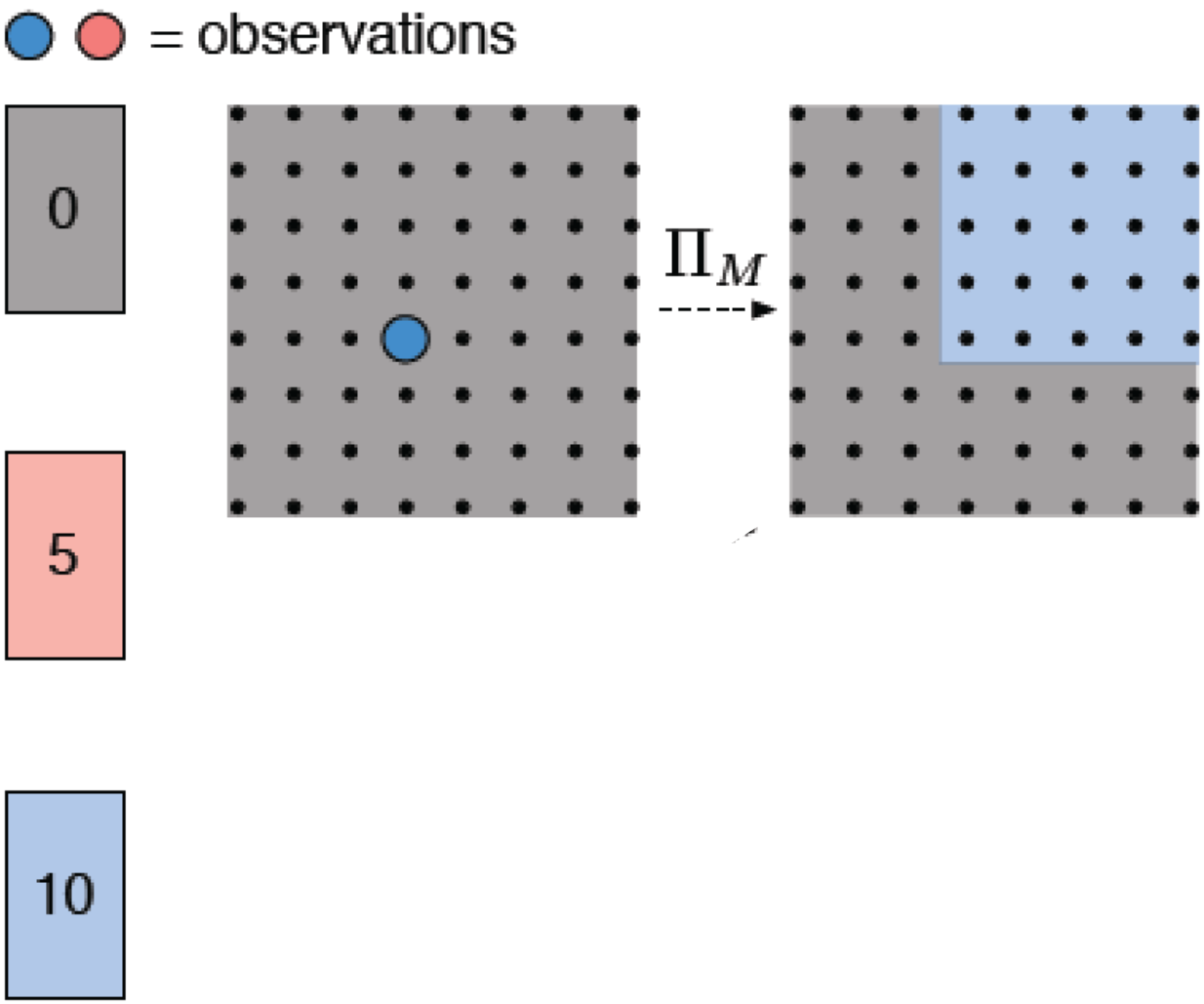
Exploiting monotonicity

- Maintaining monotonicity



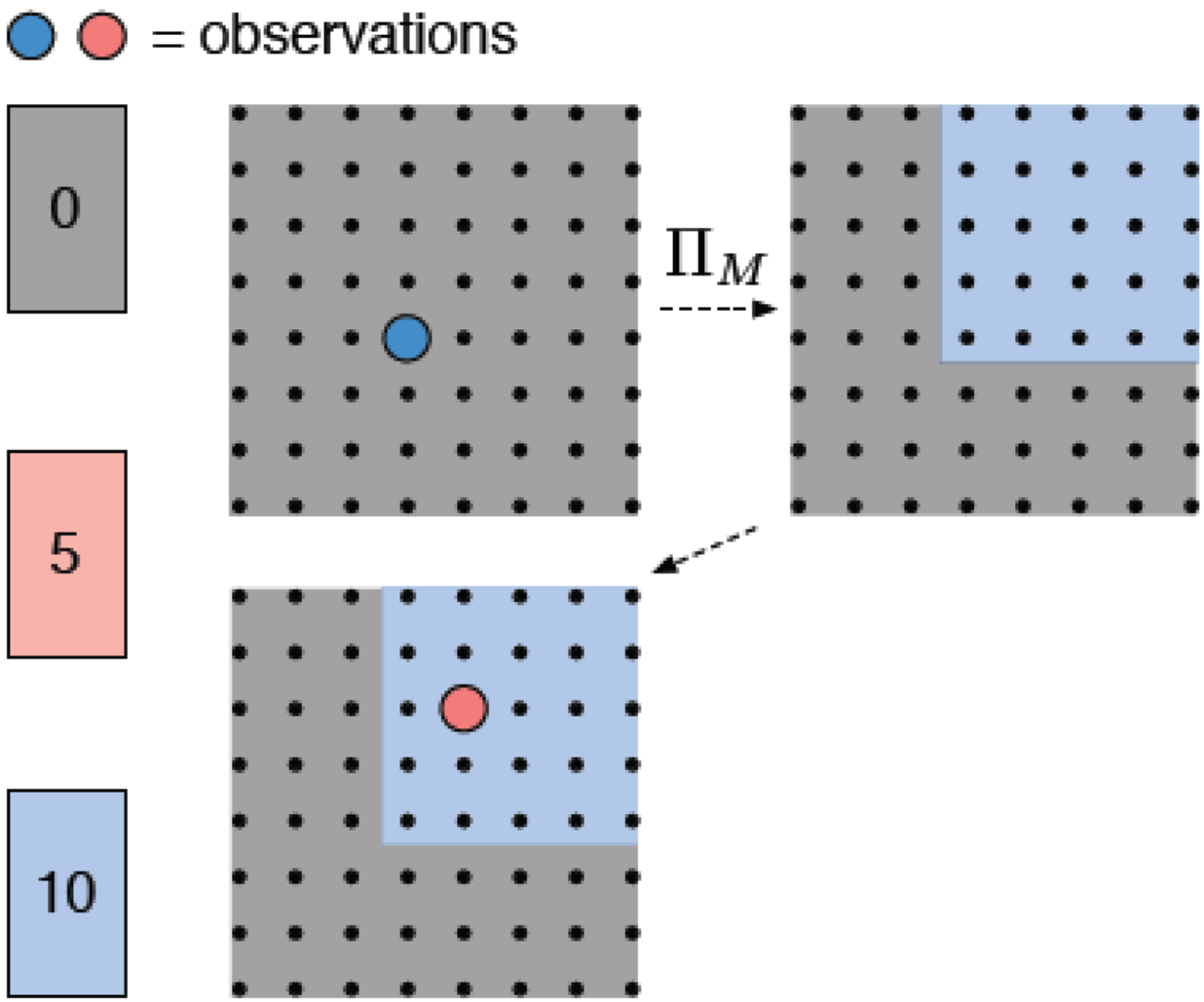
Exploiting monotonicity

- Maintaining monotonicity



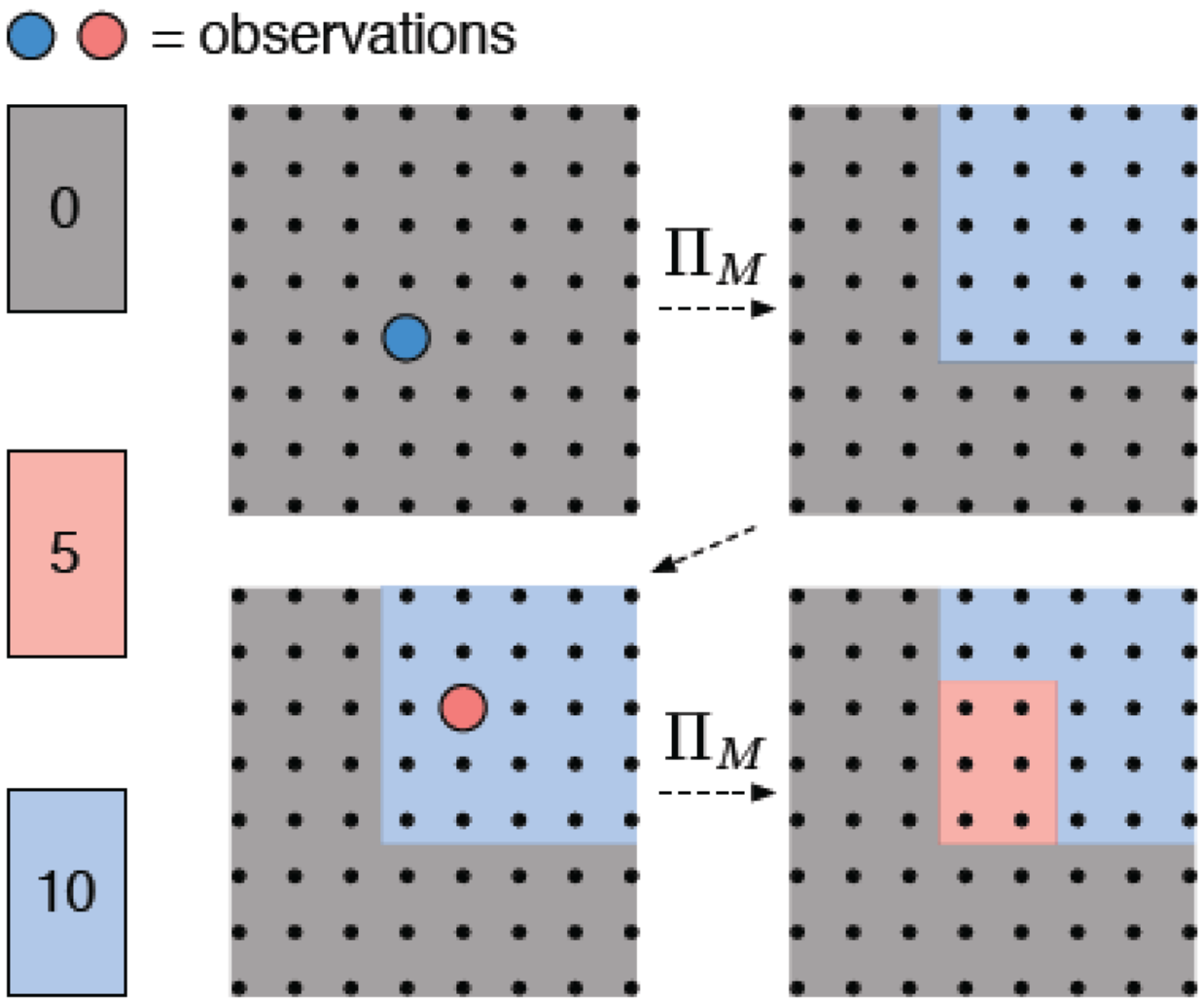
Exploiting monotonicity

- Maintaining monotonicity

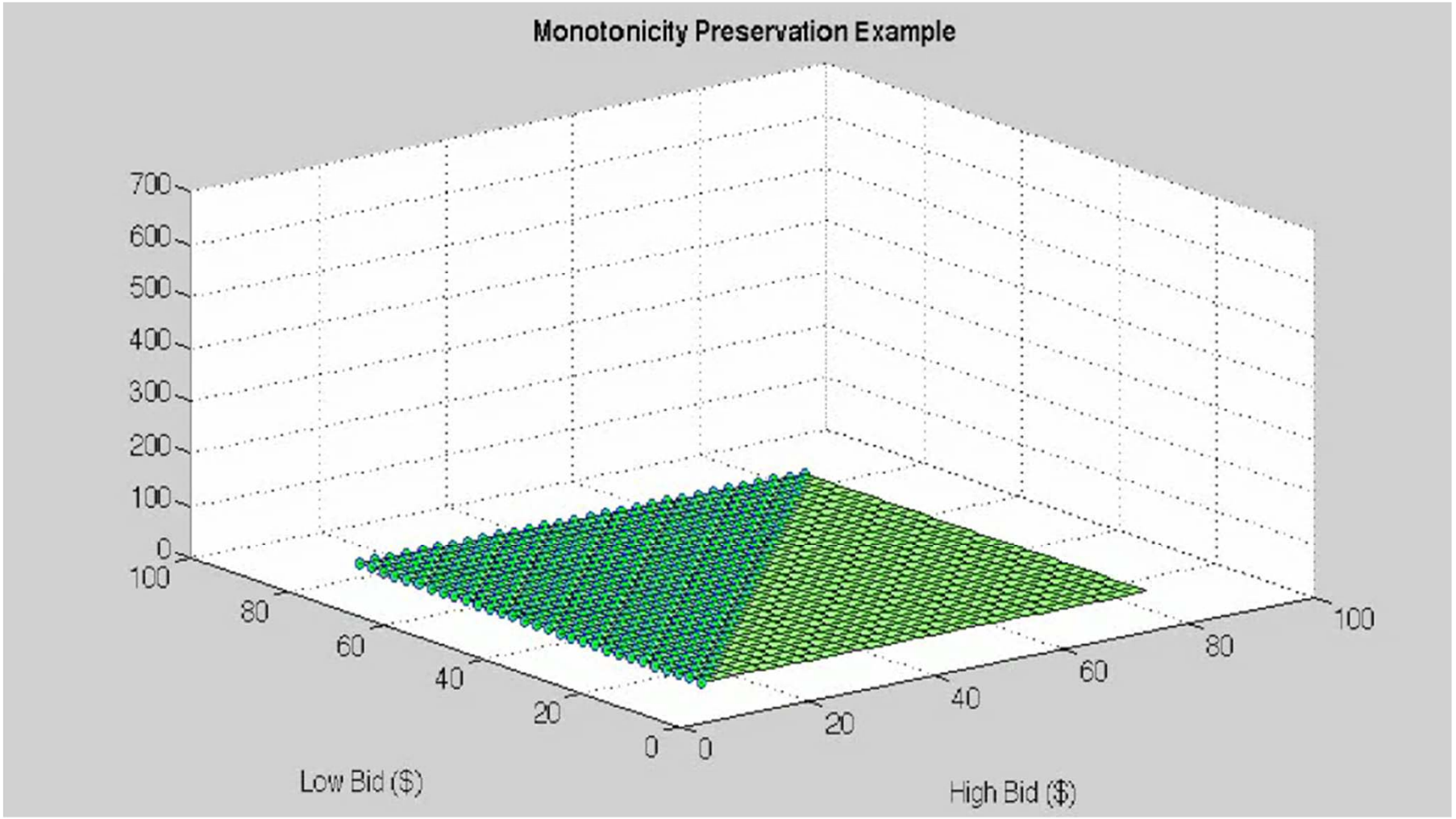


Exploiting monotonicity

- Maintaining monotonicity

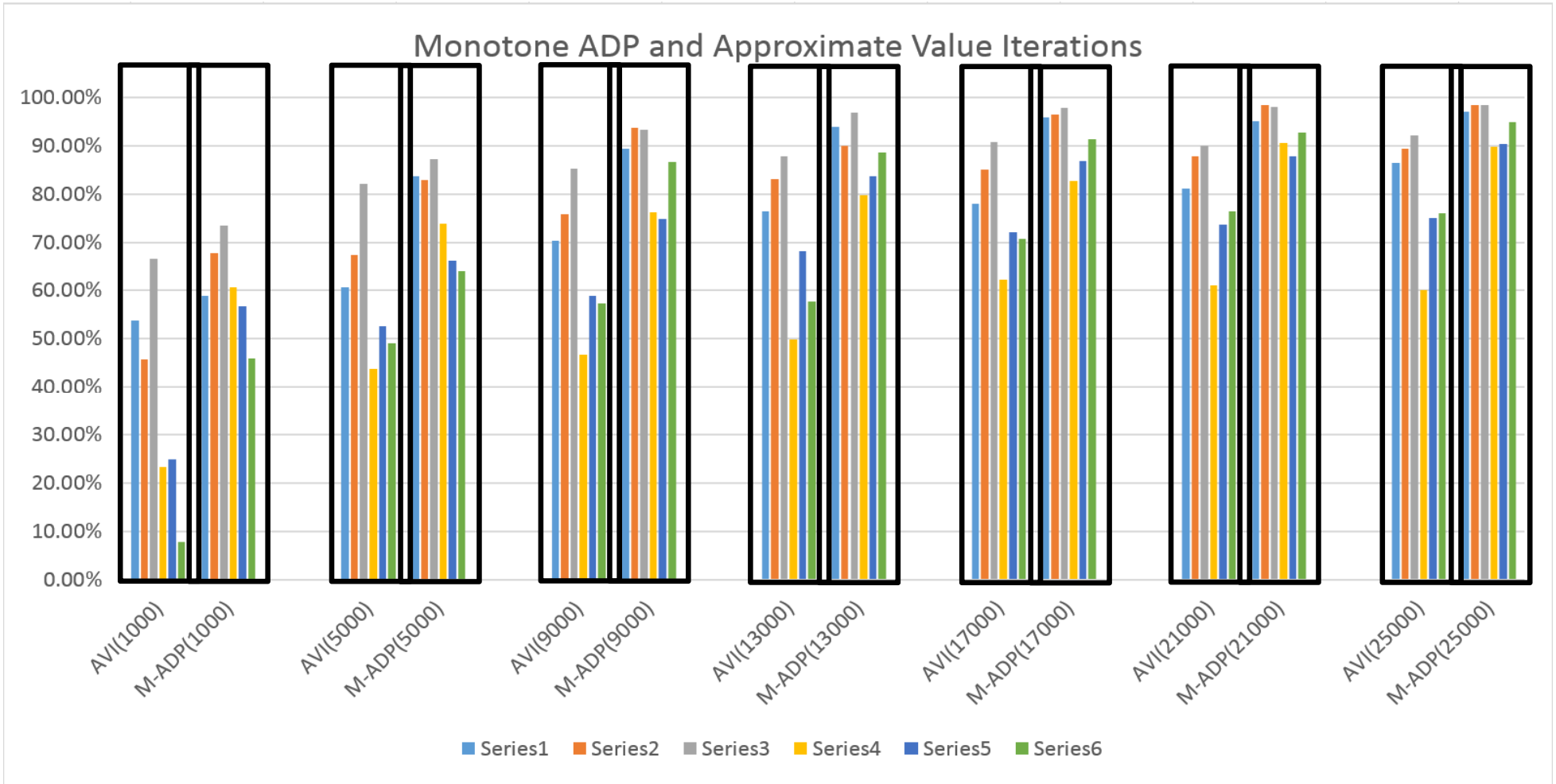


Exploiting monotonicity



Exploiting monotonicity

- Benchmarking
  - » M-ADP and AVI (no. of iterations)



Exploiting monotonicity

- Observations
  - » Initial experiments using various machine learning approximations produced poor results (60-80 percent of optimal)
  - » Vanilla lookup table works poorly (within computing budgets that spanned days)
  - » Lookup table with monotonicity worked quite well
  - » We have run this algorithm on problems with up to 7 dimensions (but that is our limit)

Backward ADP

- Backward ADP for a clinical trial problem
  - » Problem is to learn the value of a new drug within a budget of patients to be tested.
  - » Backward MDP required 268-485 hours.
  - » Forward ADP exploiting monotonicity (we will cover thiss later) required 18-30 hours.
  - » Backward ADP required 20 minutes, with a solution that was 1.2 percent within optimal.

Table 3 Computation Time Comparison between Backward MDP Algorithm and ADP Algorithm

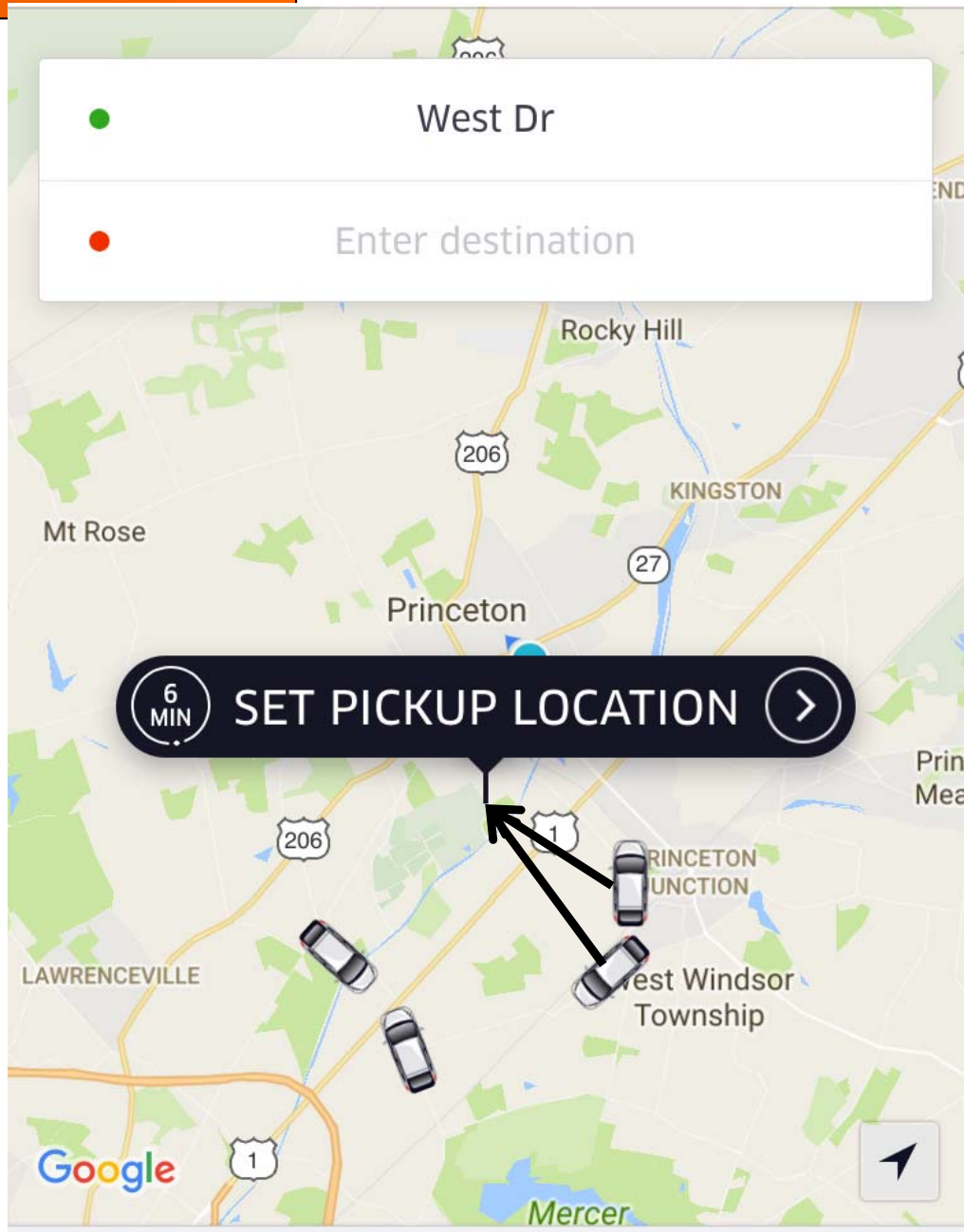
$\frac{n_1}{n_3}$	$\frac{n_2}{n_3}$	MDP CPU (hrs)	ADP CPU (hrs)	ADP Style	Gap (%)	Discretization Interval Length: backward MDP algorithm Enrollment	backward MDP algorithm Treatment	backward MDP algorithm Control	ADP algorithm Enrollment	ADP algorithm Treatment	ADP algorithm Control
0.35	0.65	360	26	forward	3.2	20	20	20	40	80	80
0.40	0.65	290	18	forward	0.1	20	20	20	40	80	80
0.40	0.75	440	30	forward	-3.2	20	20	20	40	80	80
0.55	0.85	485	18	forward	3.7	10	20	20	40	80	80
0.55	0.85	268	0.3	backward	1.2	10	736	72	10	1487	760

Approximate dynamic programming

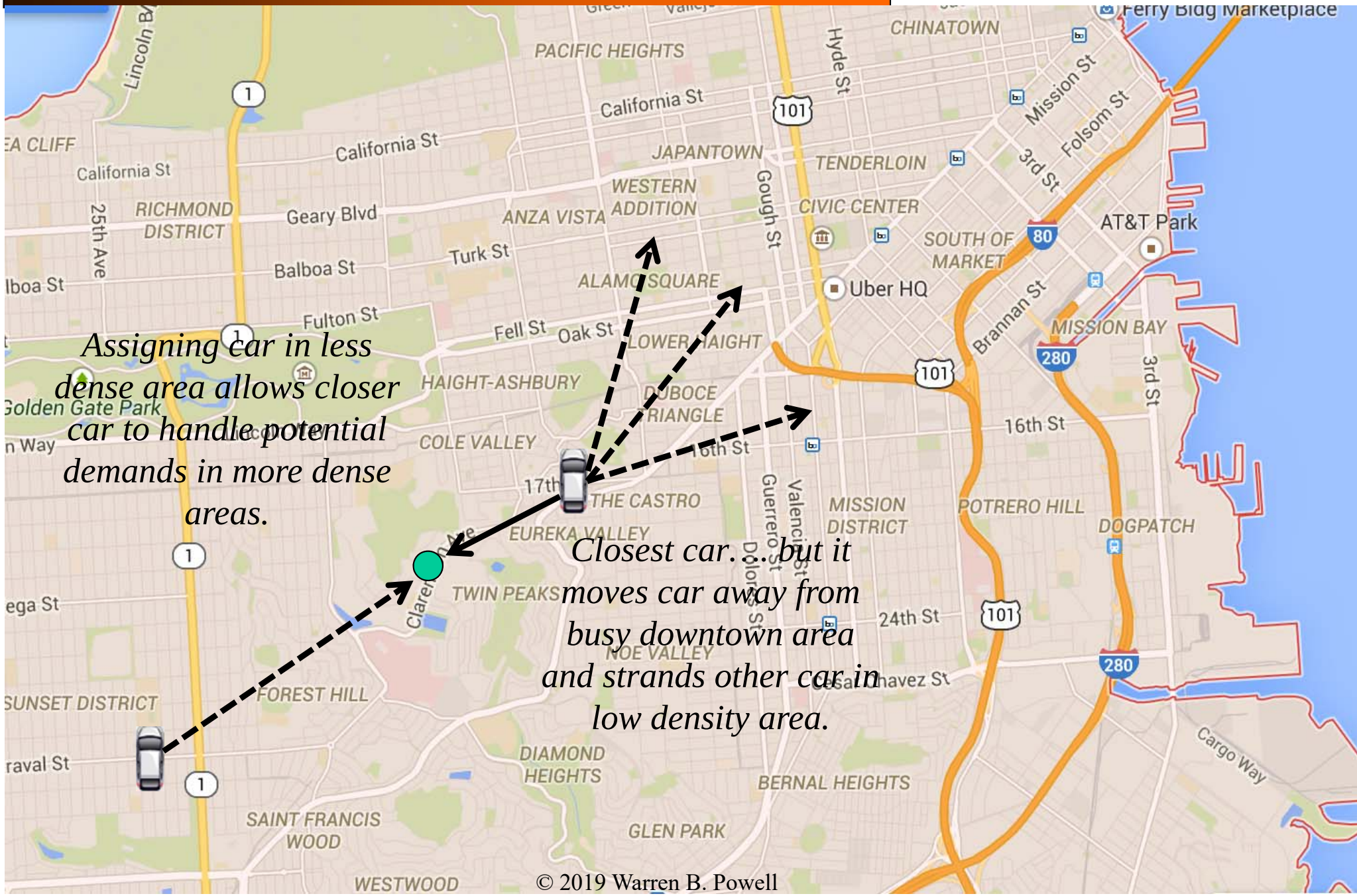
Driverless EV problem

Driverless EV optimization

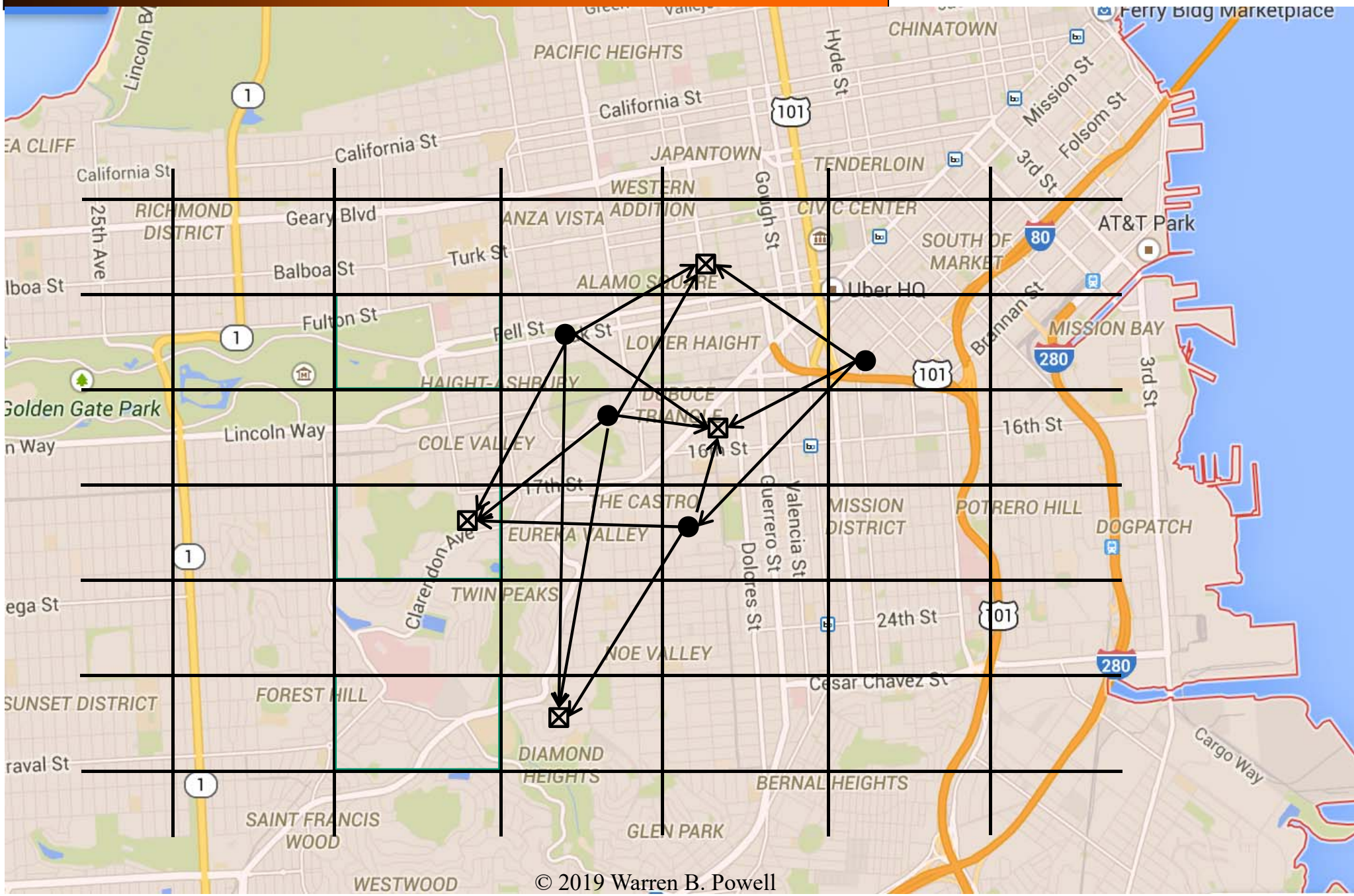
- Current Uber logic:
  - » Show nearest 8 drivers.
  - » Contact closest driver to confirm assignment.
  - » If driver does not confirm, contact second closest driver.
- Limitations:
  - » Ignores potential future opportunities for each driver.



Driverless EV optimization



Driverless EV optimization

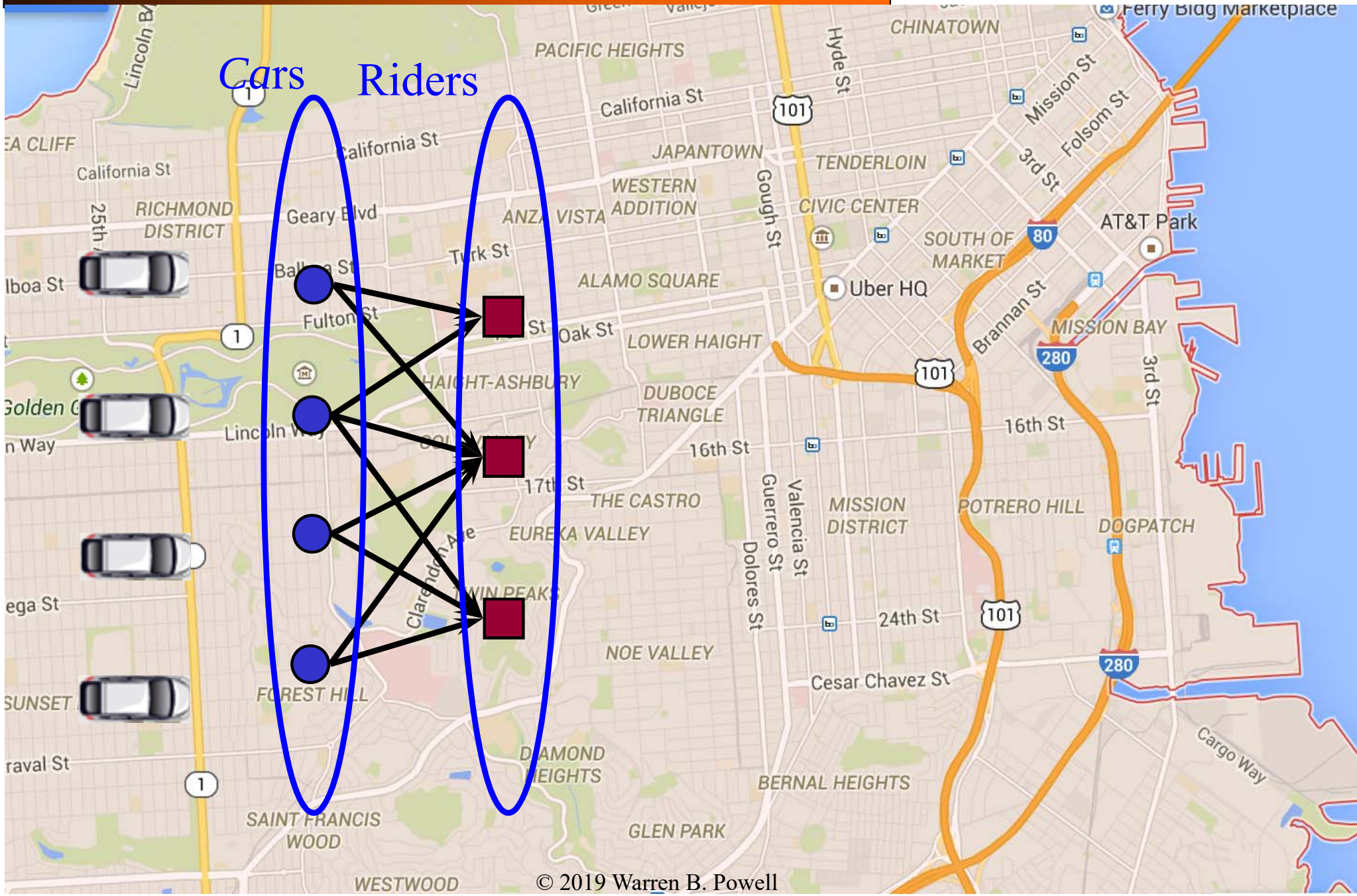


Driverless EV optimization

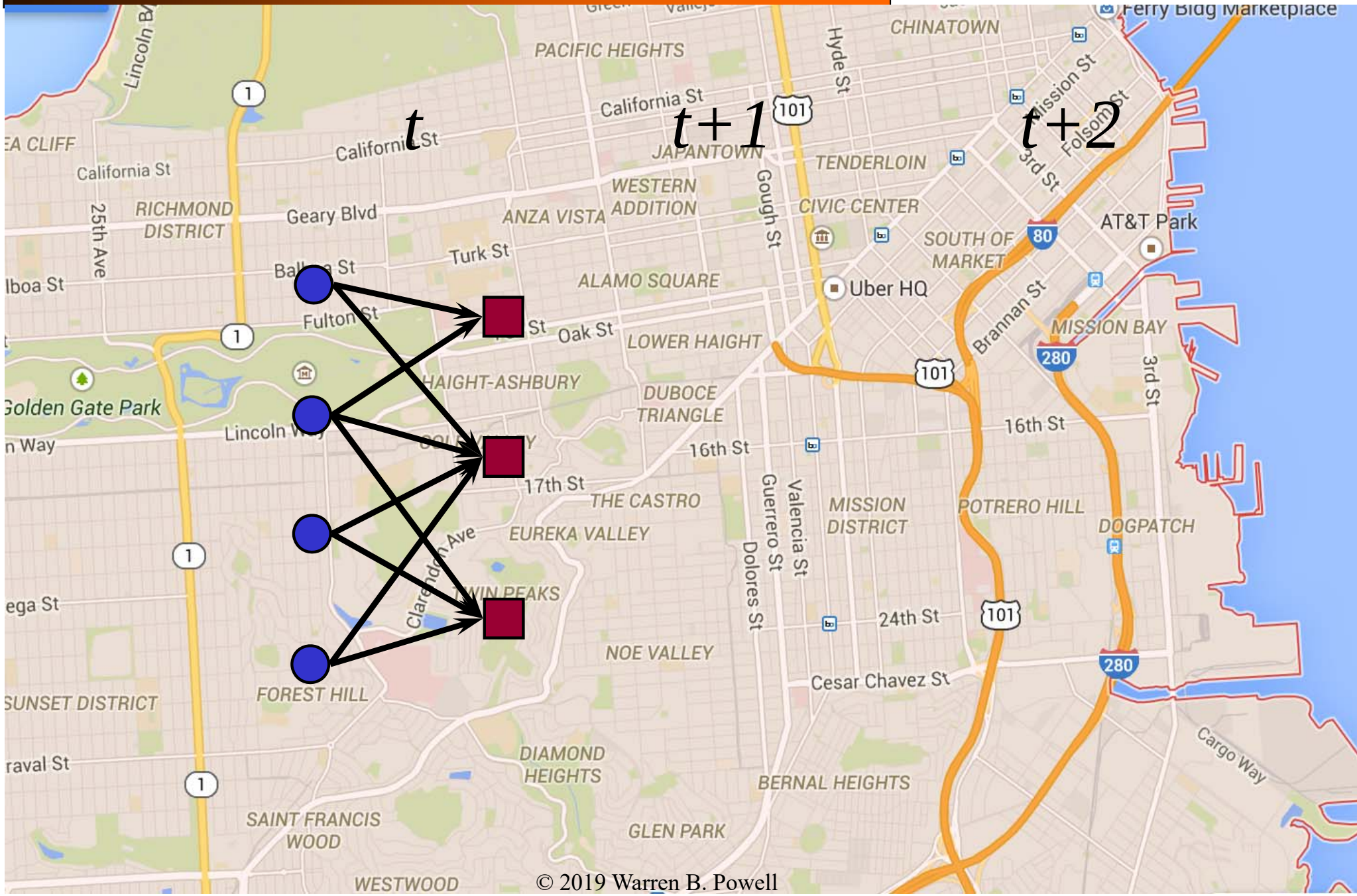
- The central operator should think about:
  - Should it accept this trip?
  - What is the best car to assign to the trip considering
    - The type of car
    - The charge level of the battery

» If it doesn't assign a car to a trip, should it:
  - Sit where it is?
  - Reposition to a better location?
  - Recharge the battery?
  - Move to a parking facility?

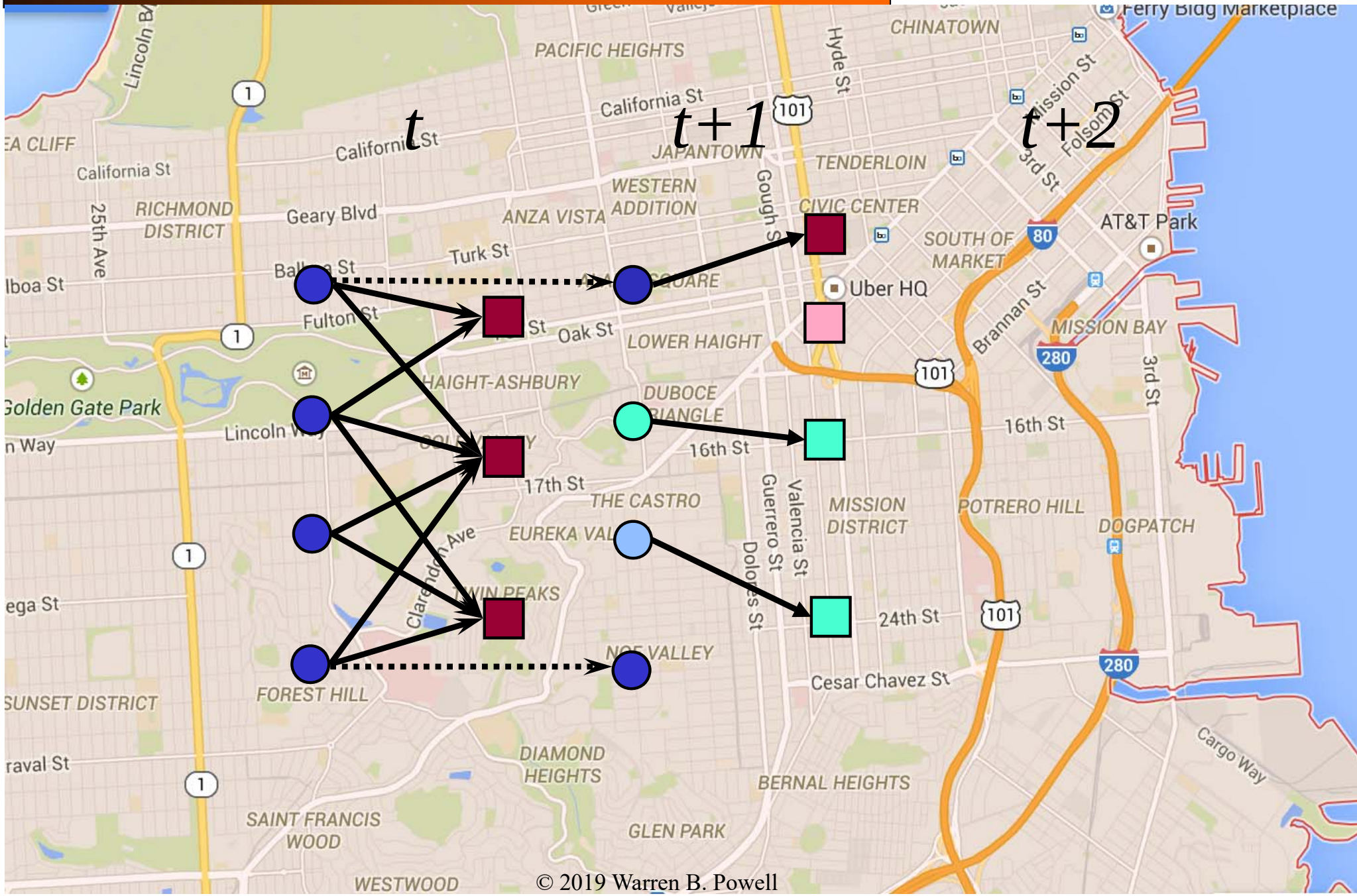
Driverless EV optimization



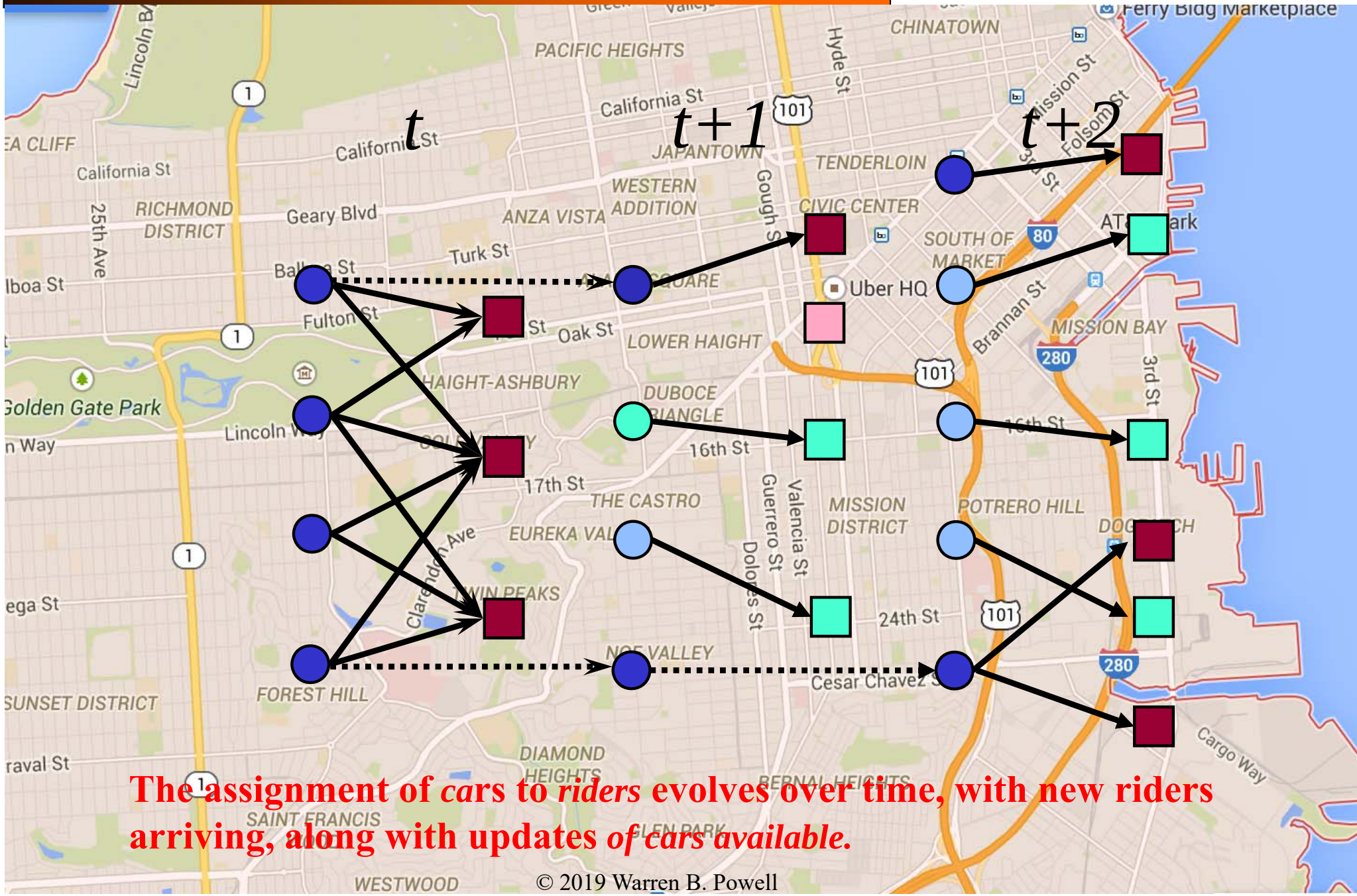
Driverless EV optimization



Driverless EV optimization



Driverless EV optimization



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

Driverless EV optimization

- Policies based on value function approximations

$$X_t^{VFA}(S_t^n) = \arg\max_{x_t} \left(C(S_t^n, x_t) + \bar{V}_t^{n-1}\left(S^{M,x}(S_t^n, x_t)\right)\right)$$

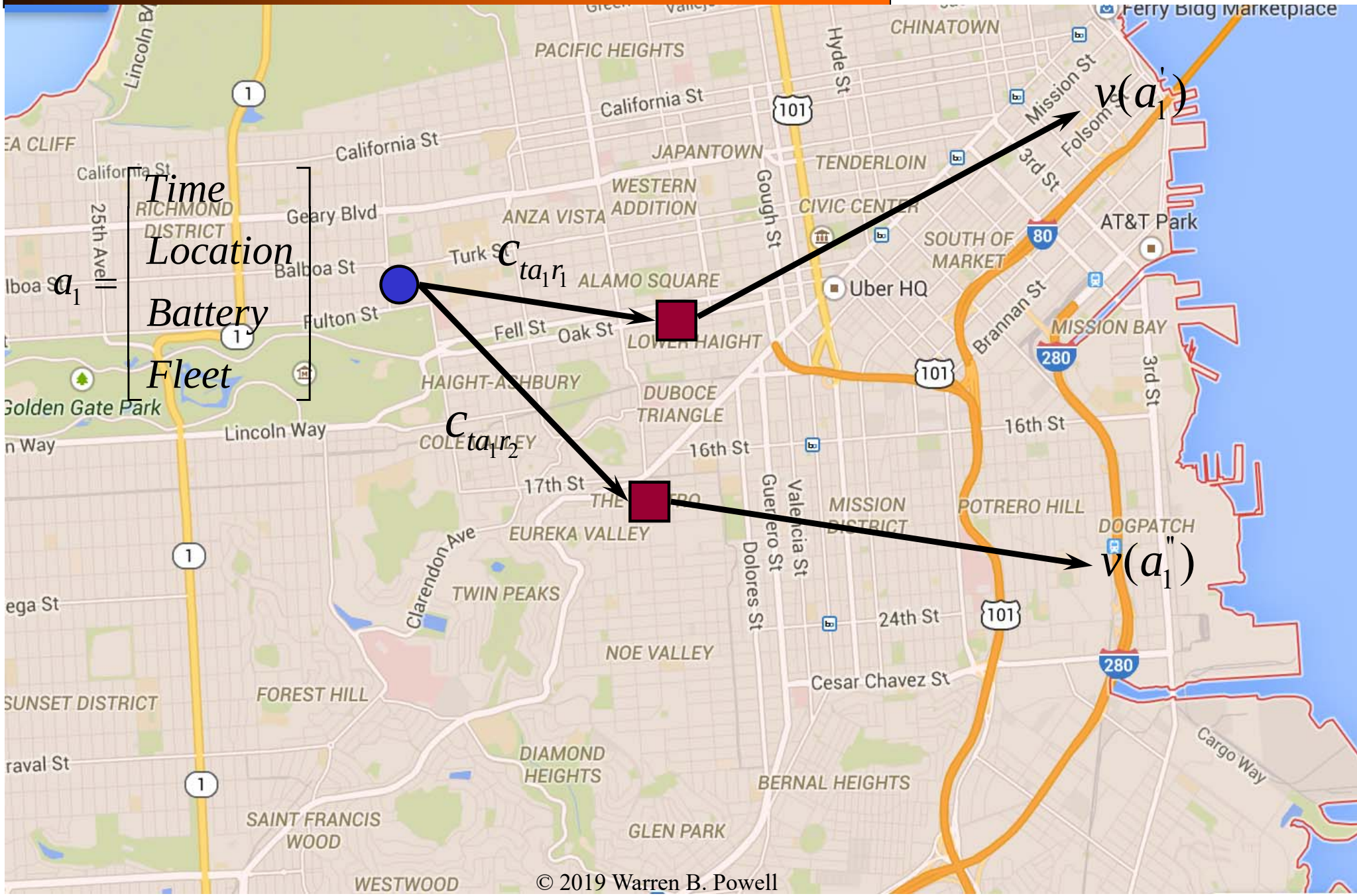
Contribution for taking decision x_t from state S_t^n

Expected Value of the Post-decision State

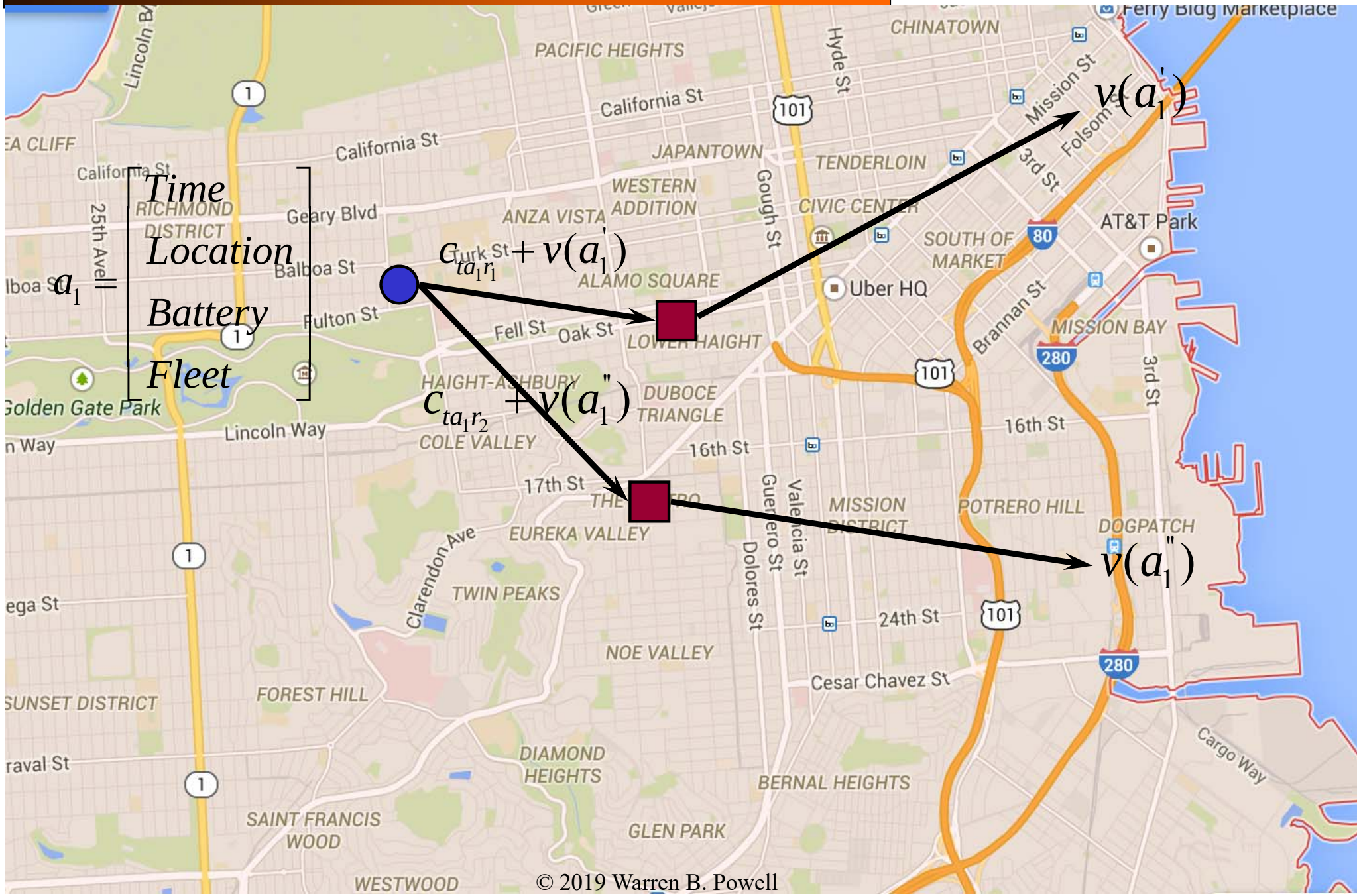
» Exact value functions are rare:
  - Discrete states and actions, with a computable one-step transition matrix.

» Approximate value functions are defined by:
  - Approximation architecture
    - Linear, nonlinear separable, nonlinear
  - Learning strategy
    - Pure forward pass, two-pass

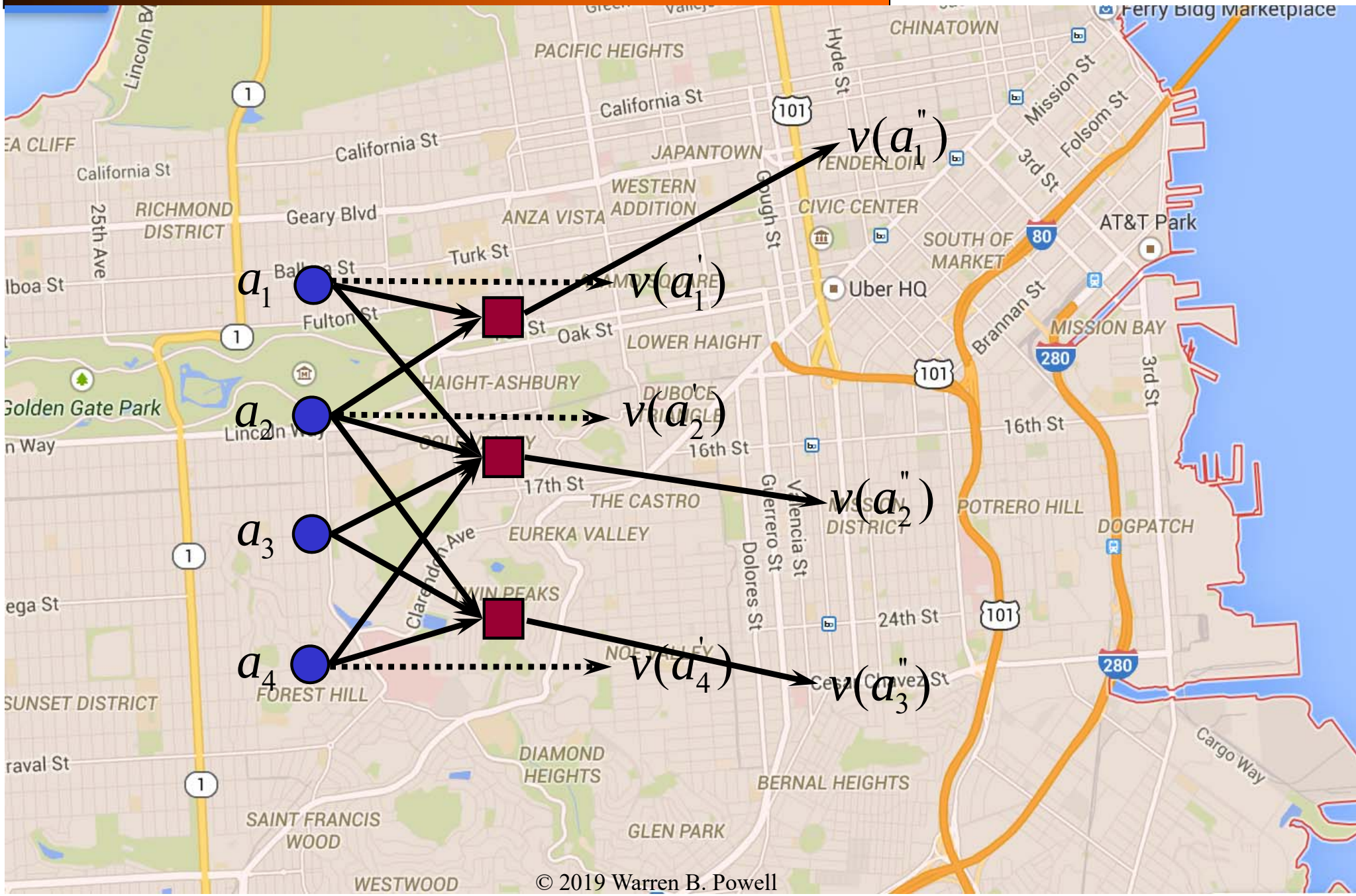
Driverless EV optimization



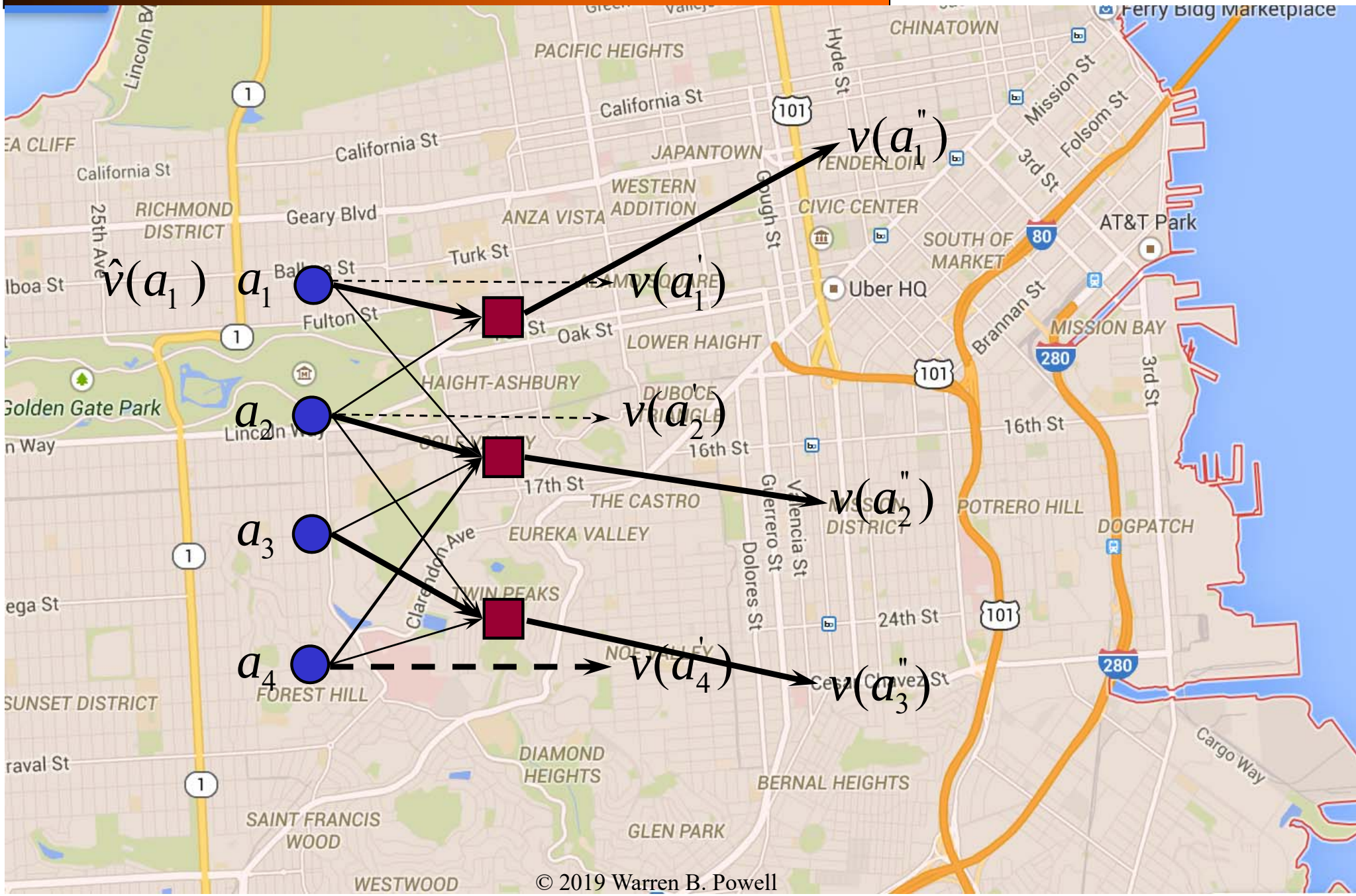
Driverless EV optimization



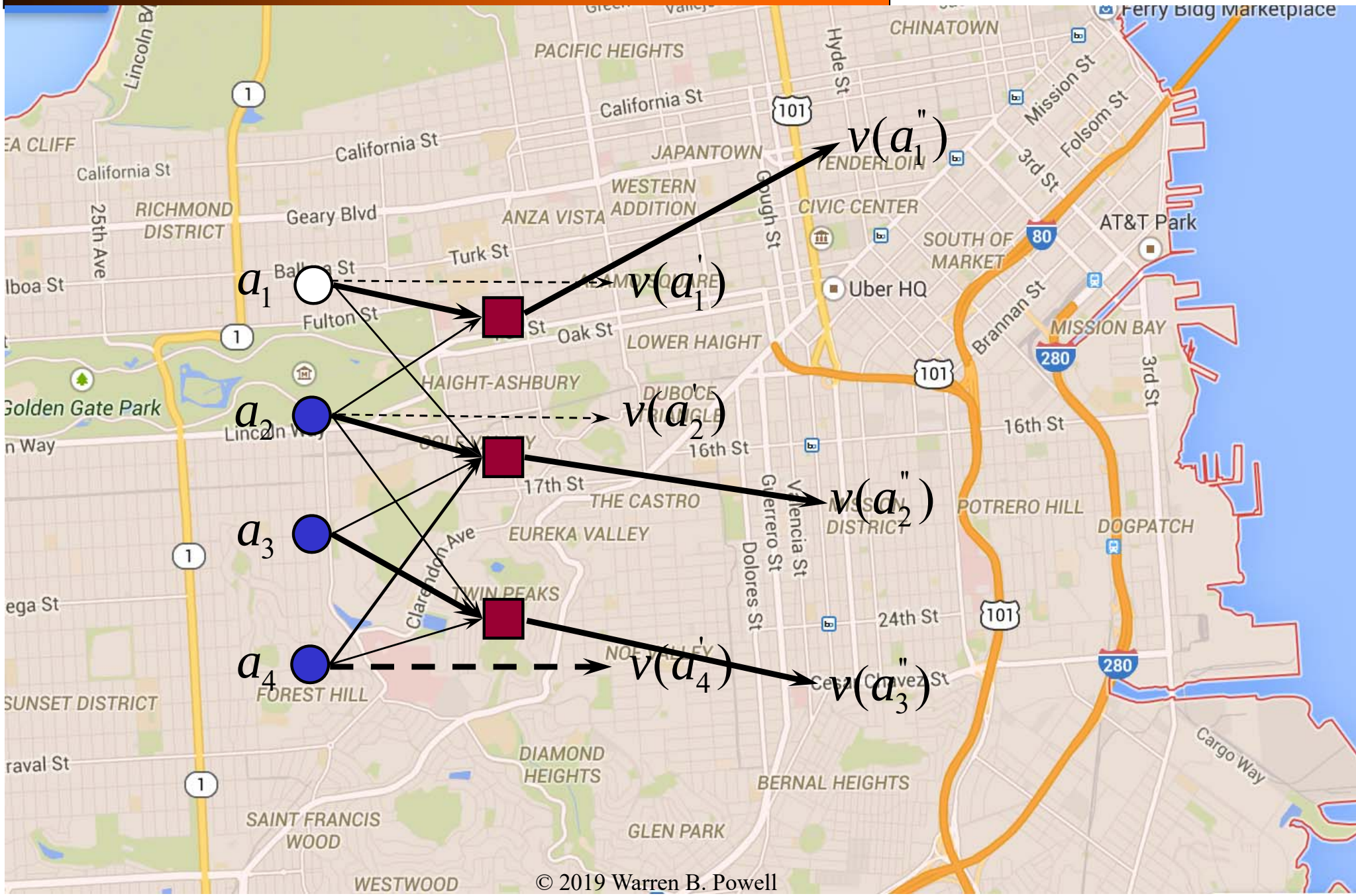
Driverless EV optimization



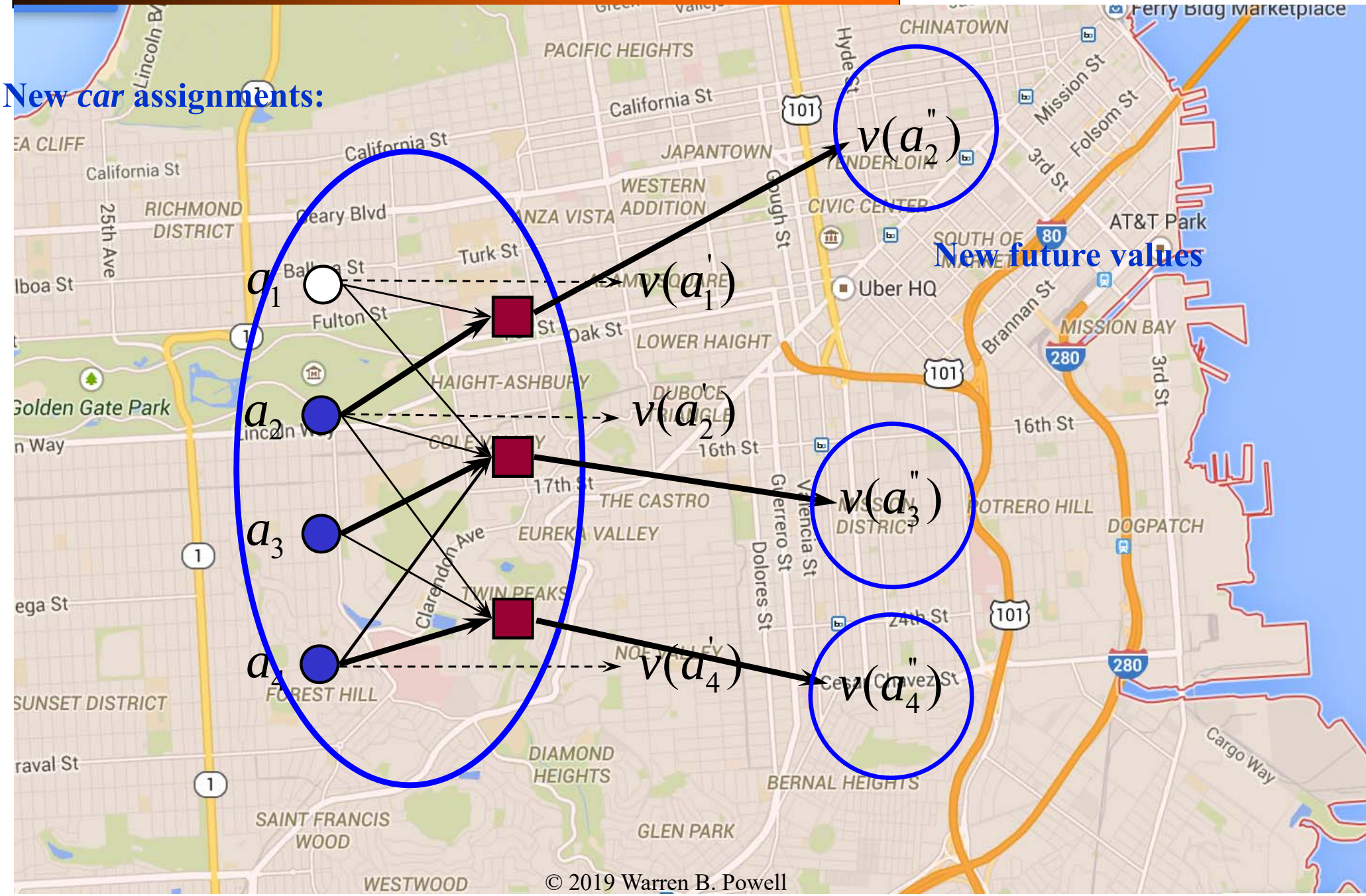
Driverless EV optimization



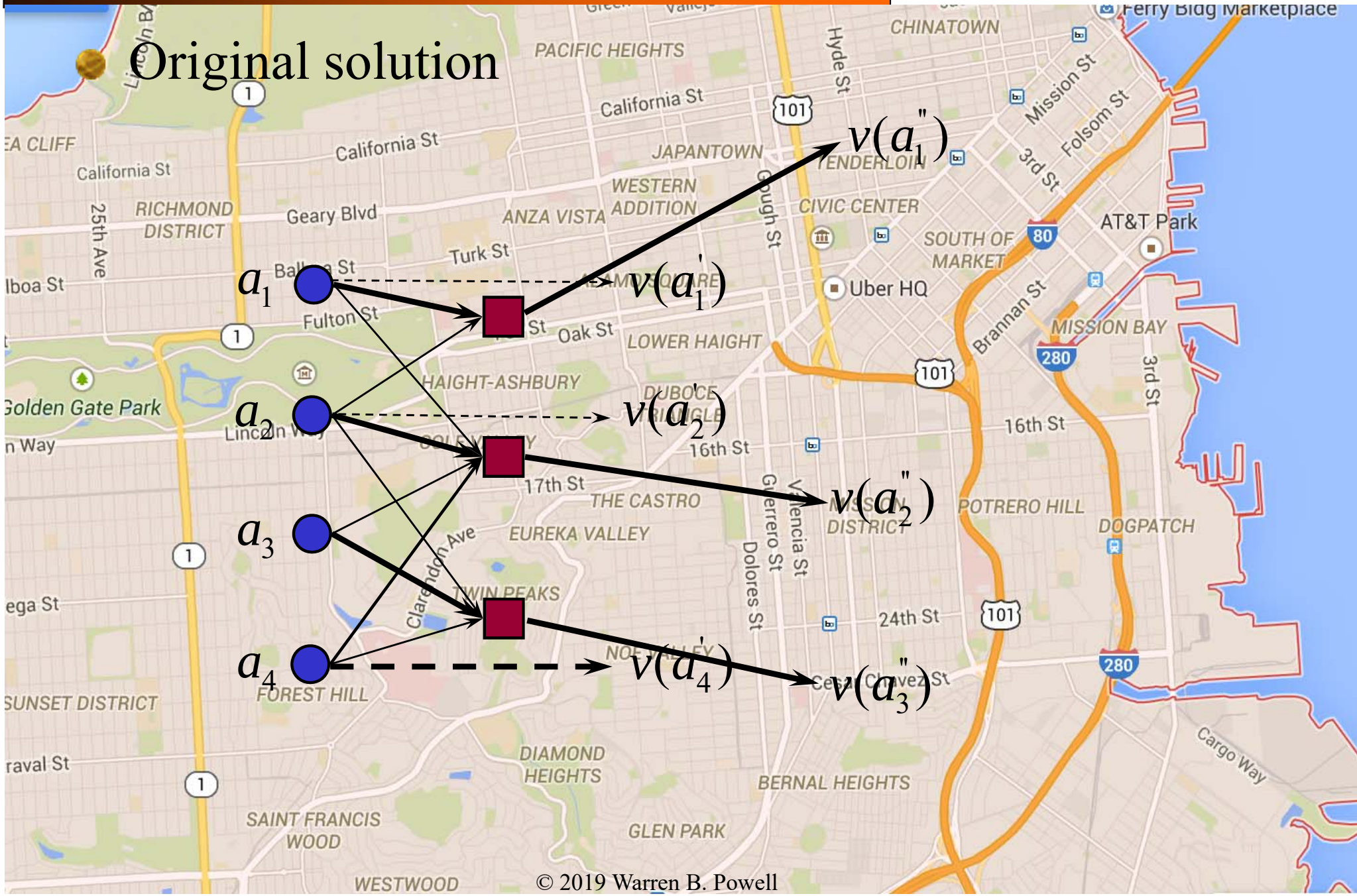
Driverless EV optimization



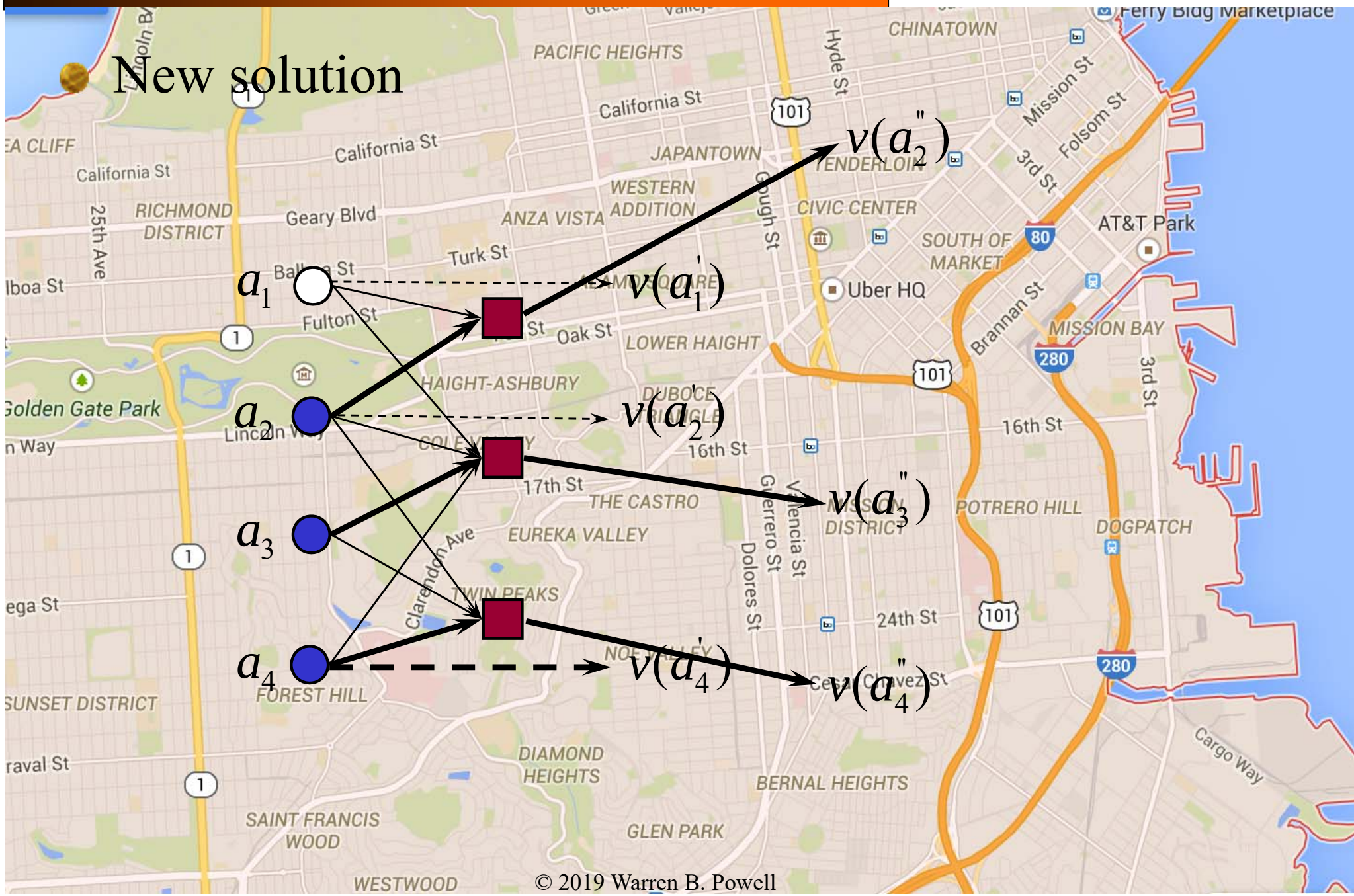
Driverless EV optimization



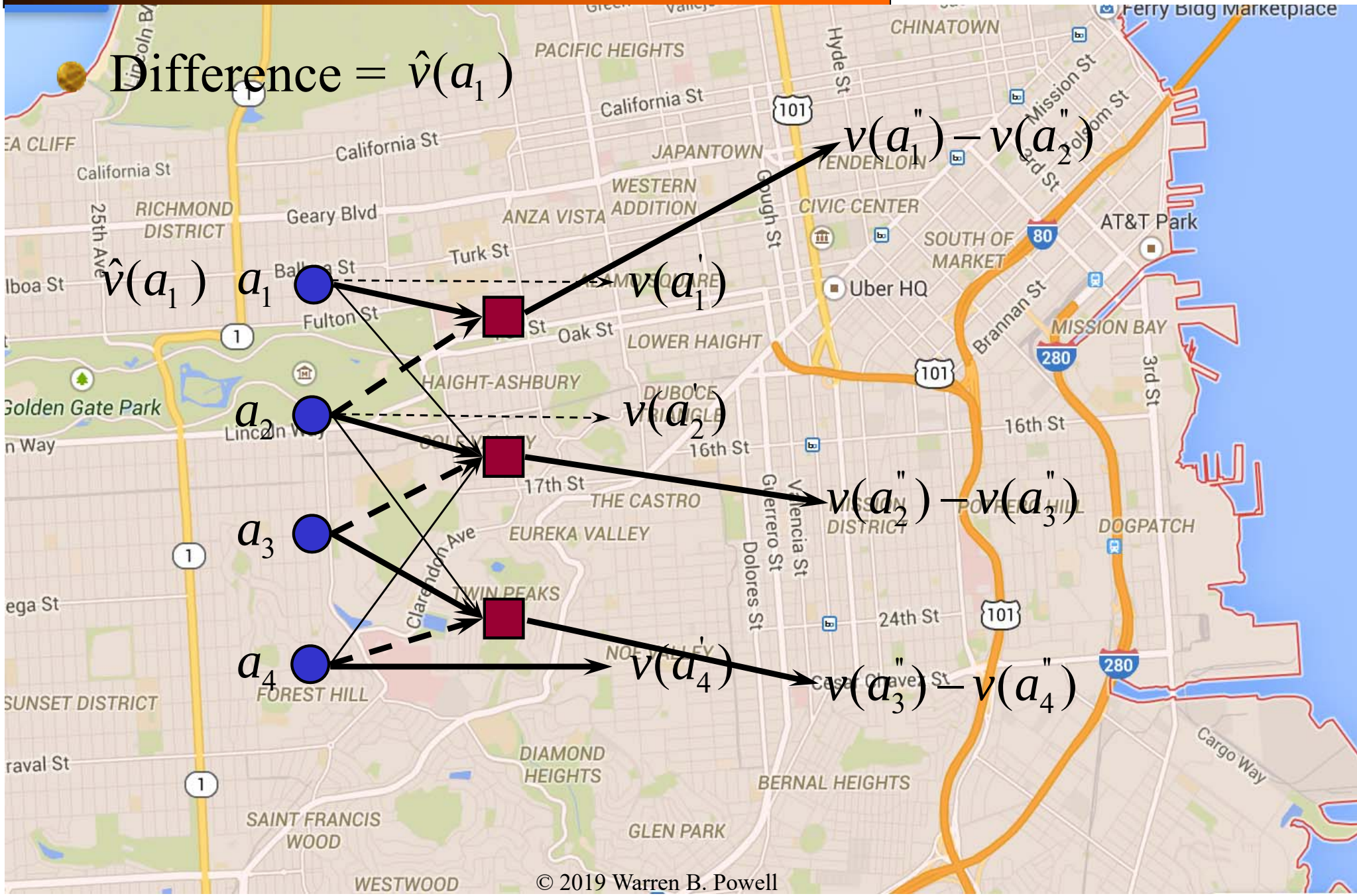
Driverless EV optimization



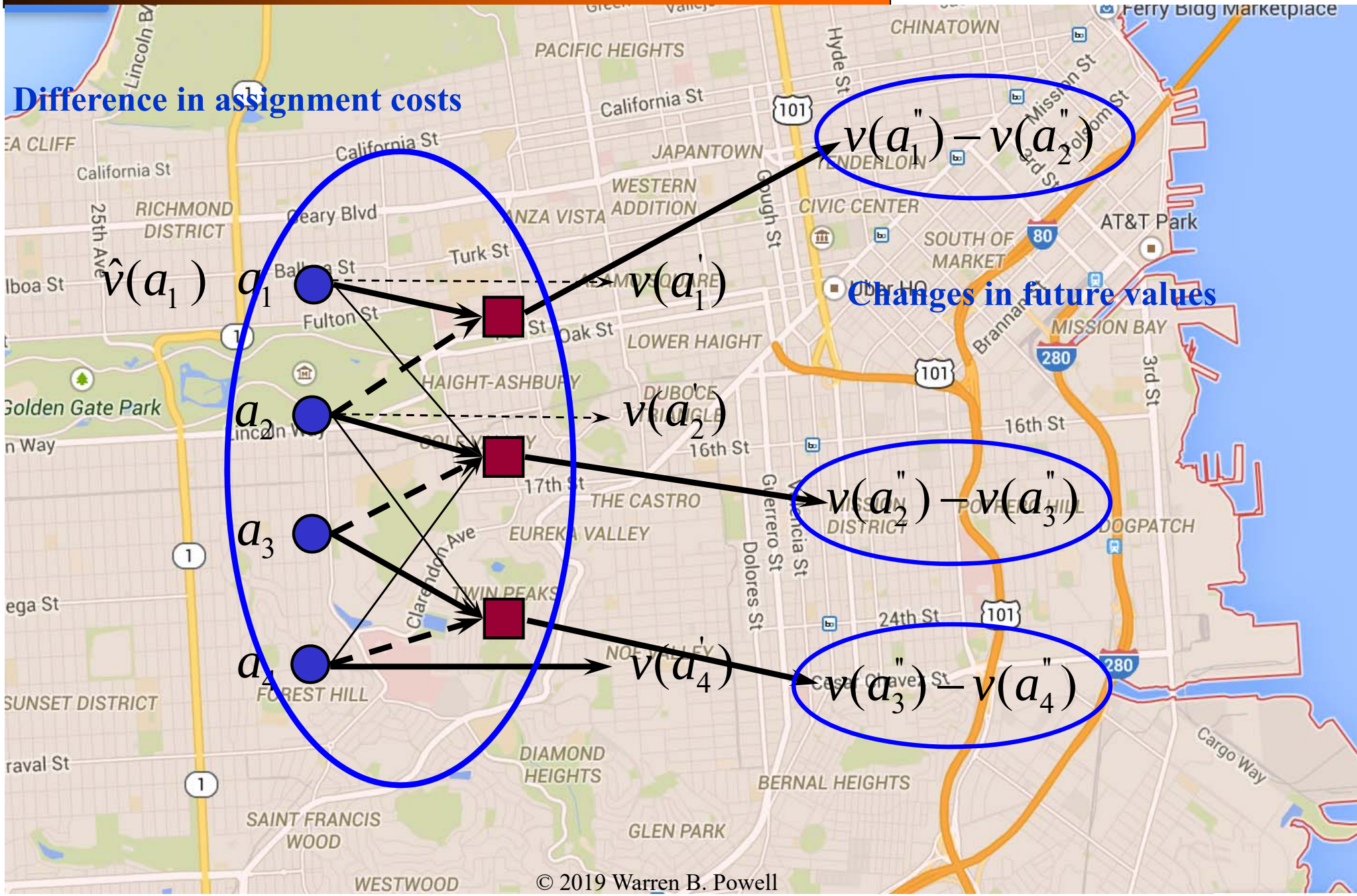
Driverless EV optimization



Driverless EV optimization

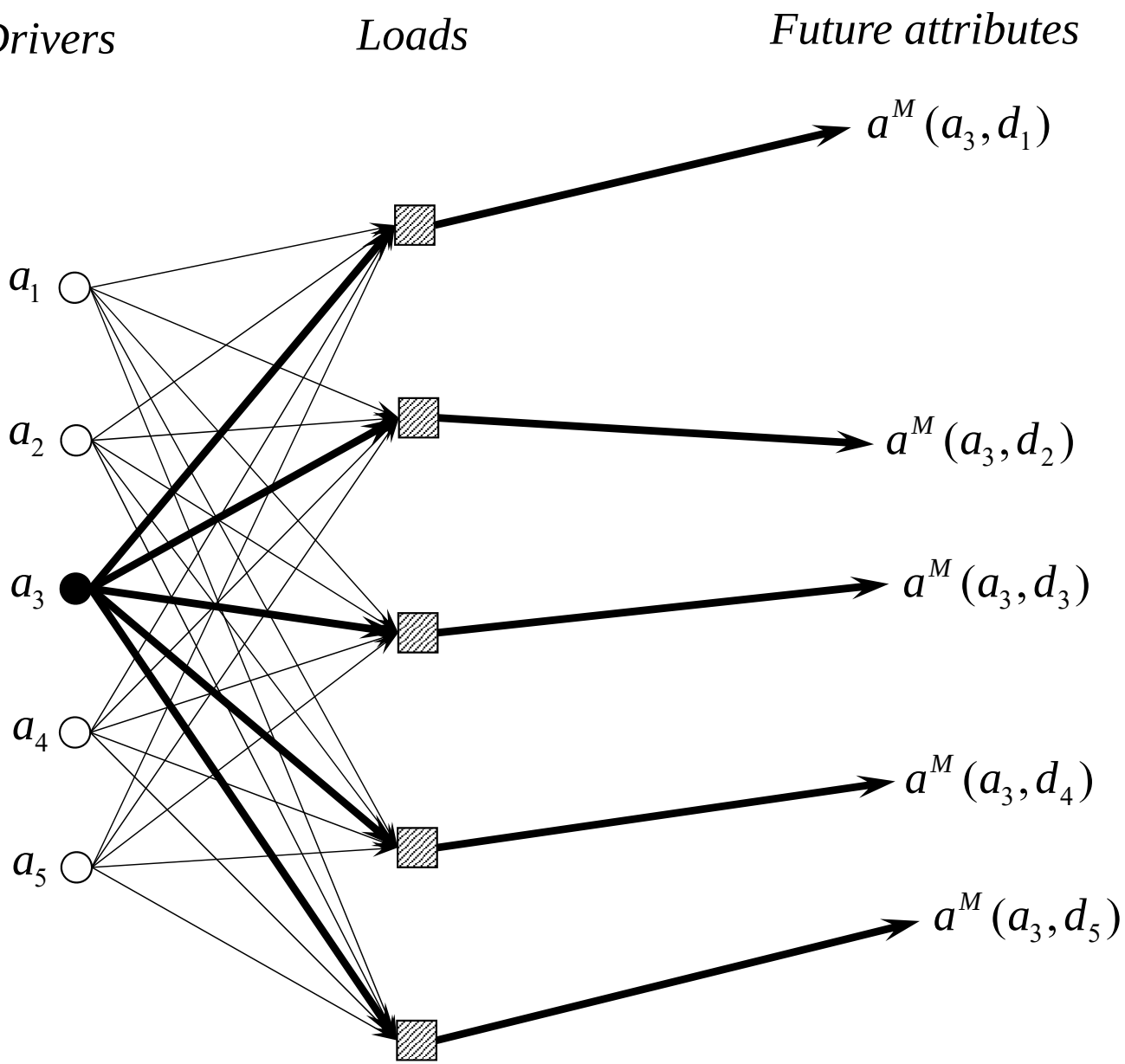


Driverless EV optimization



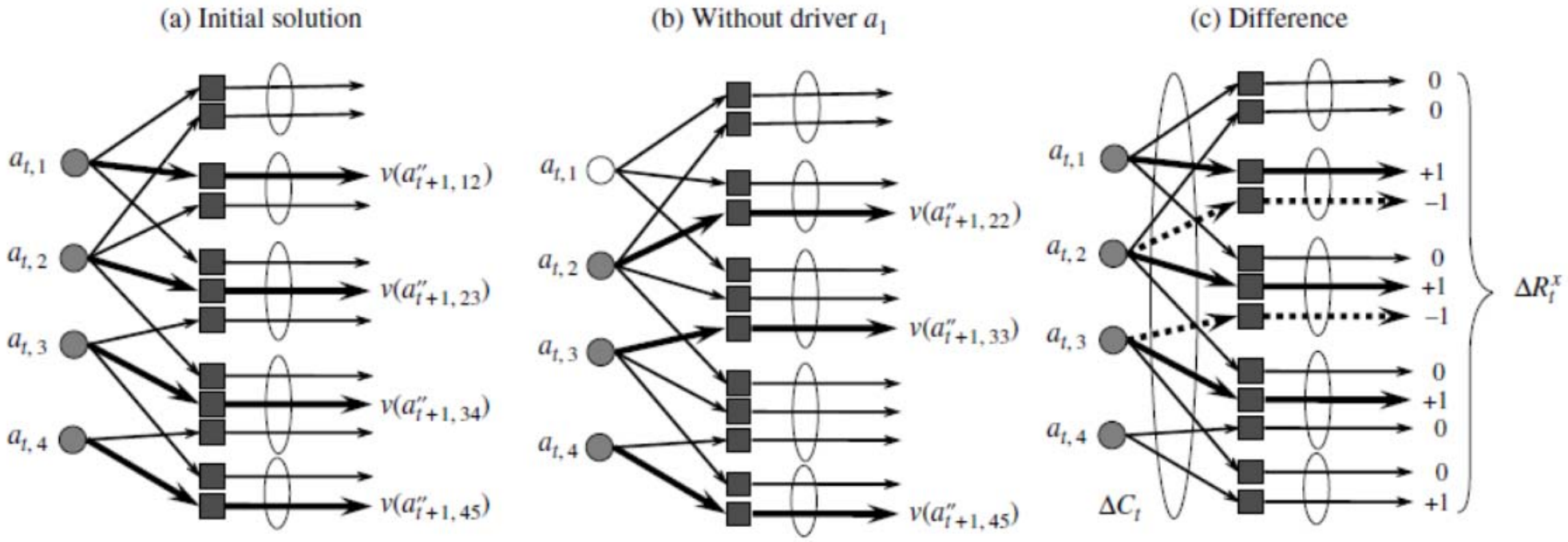
Driverless EV optimization

- Assignment network
  - » Capture the value of downstream driver.
  - » $a^M(a_3, d_1)$ = Attribute vector of driverin the future given a decision d.
  - » Add this value to the assignment arc.



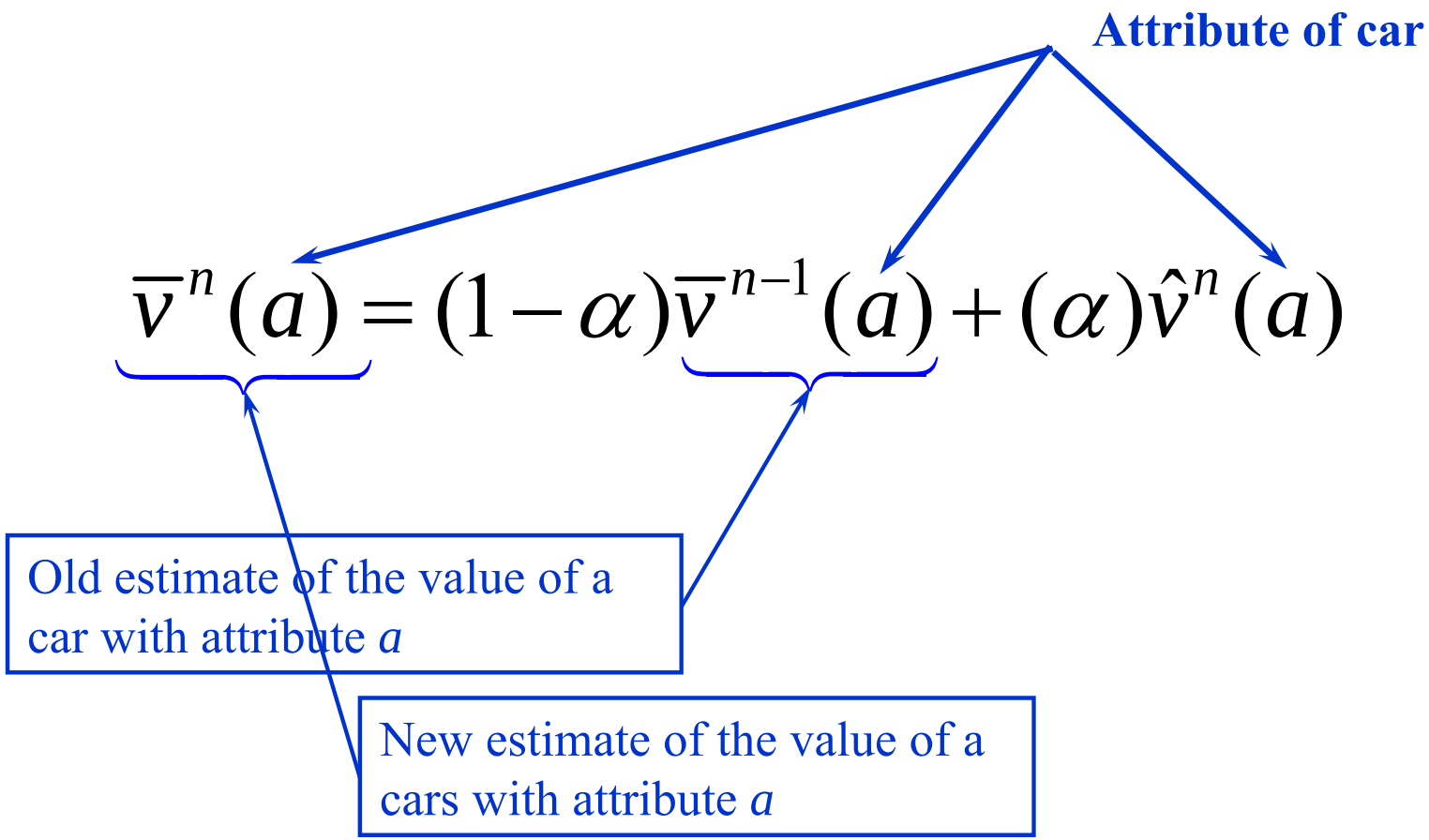
Driverless EV optimization

- Finding the marginal value of a driver:
  - » Dual variables
    - Can provide unreliable estimates.
    - Need to get the marginal value of drivers who are not actually there.
  - » Numerical derivatives:



Driverless EV optimization

- Estimating average values:

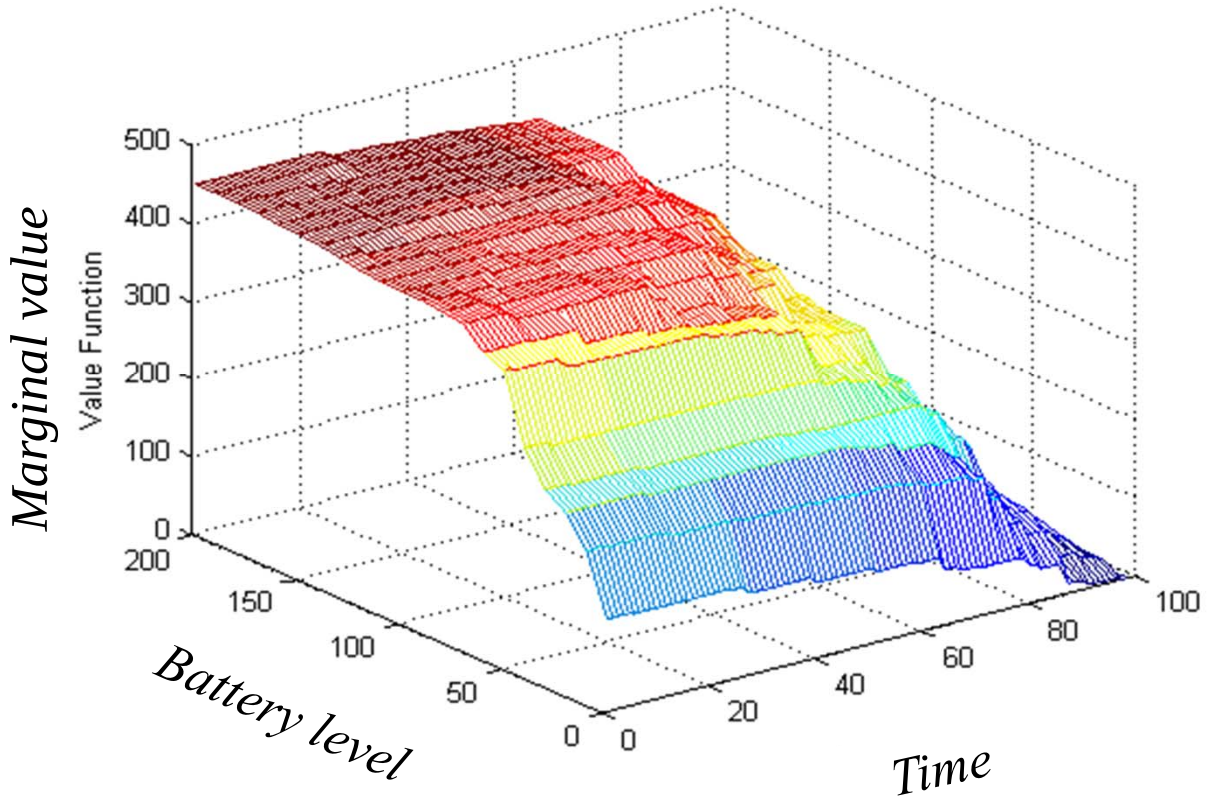


Driverless EV optimization

- We had to use two statistical techniques to accelerate learning:
  - » Monotonicity with respect to time and charge level.
  - » Hierarchical aggregation (just as we did with the trucking example).

Driverless fleets of EVs using ADP

- The value of a vehicle in the future
  - » Value function approximation captures charge level, as well as time and location.
  - » Hierarchical aggregation accelerated the learning process



Driverless EV optimization

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:

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

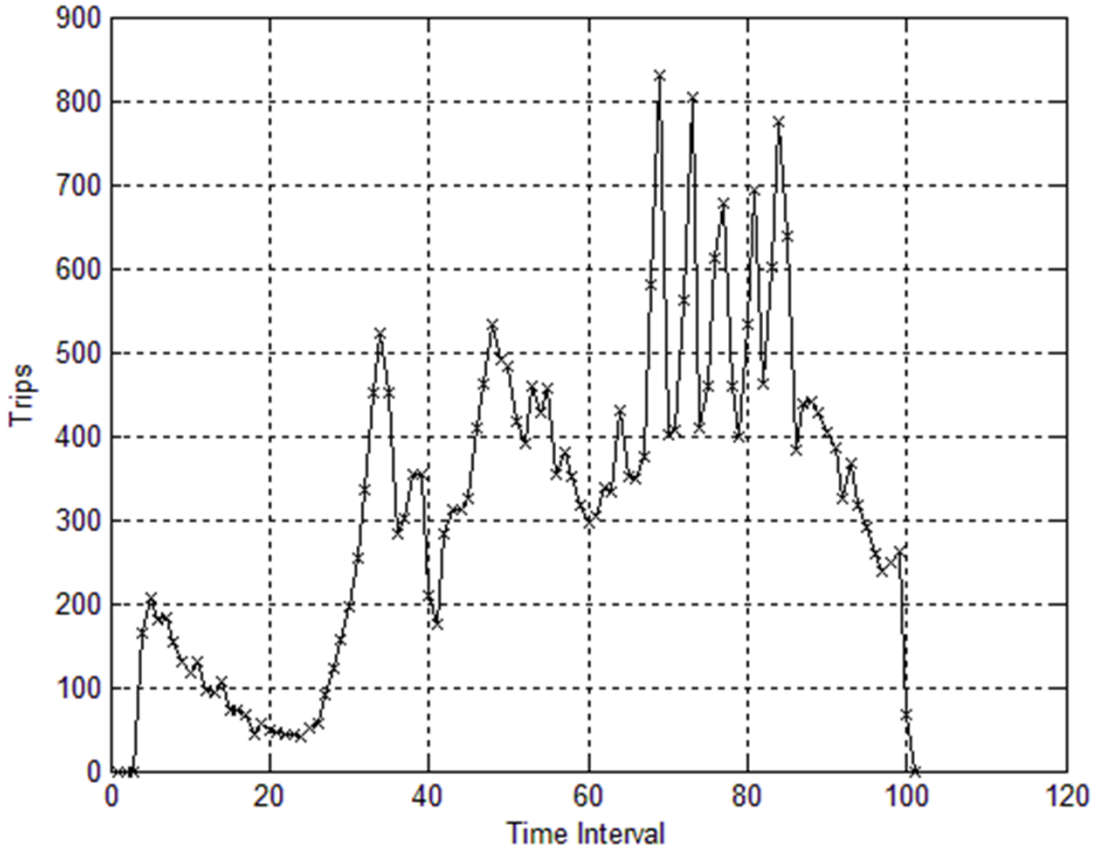
Simulation

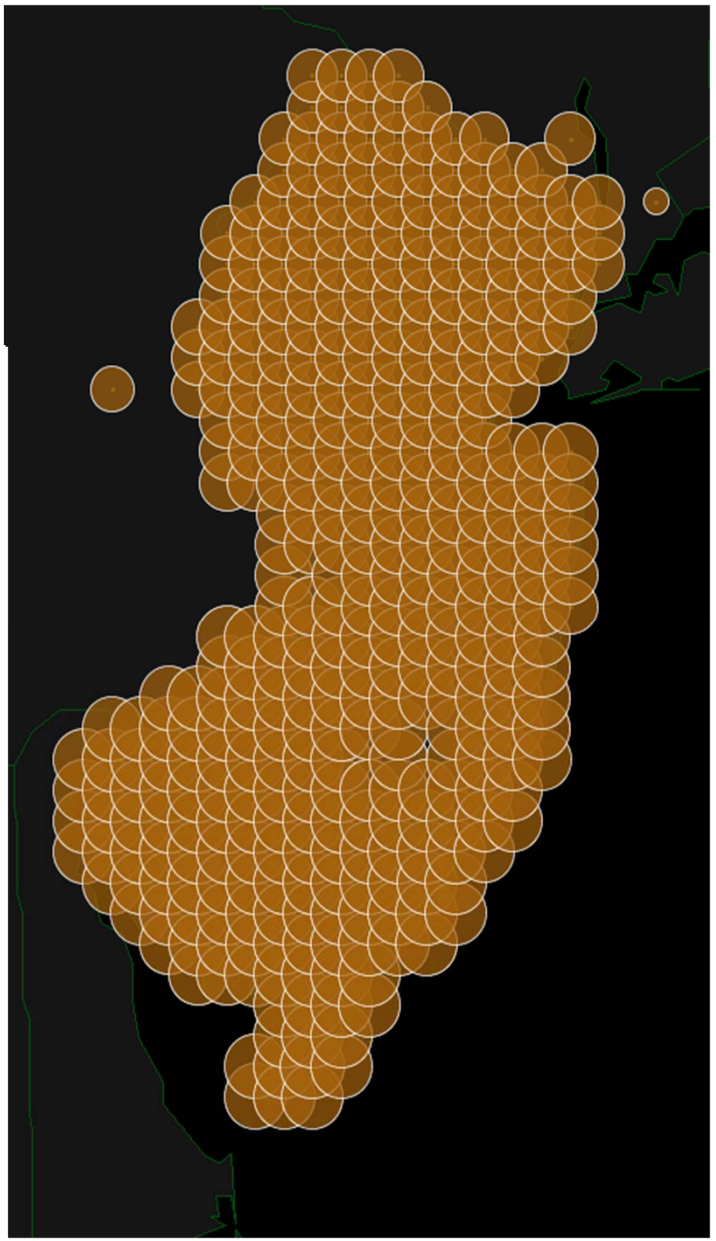
Step 5: Return to step 1.

"on policy learning"

Driverless EV optimization

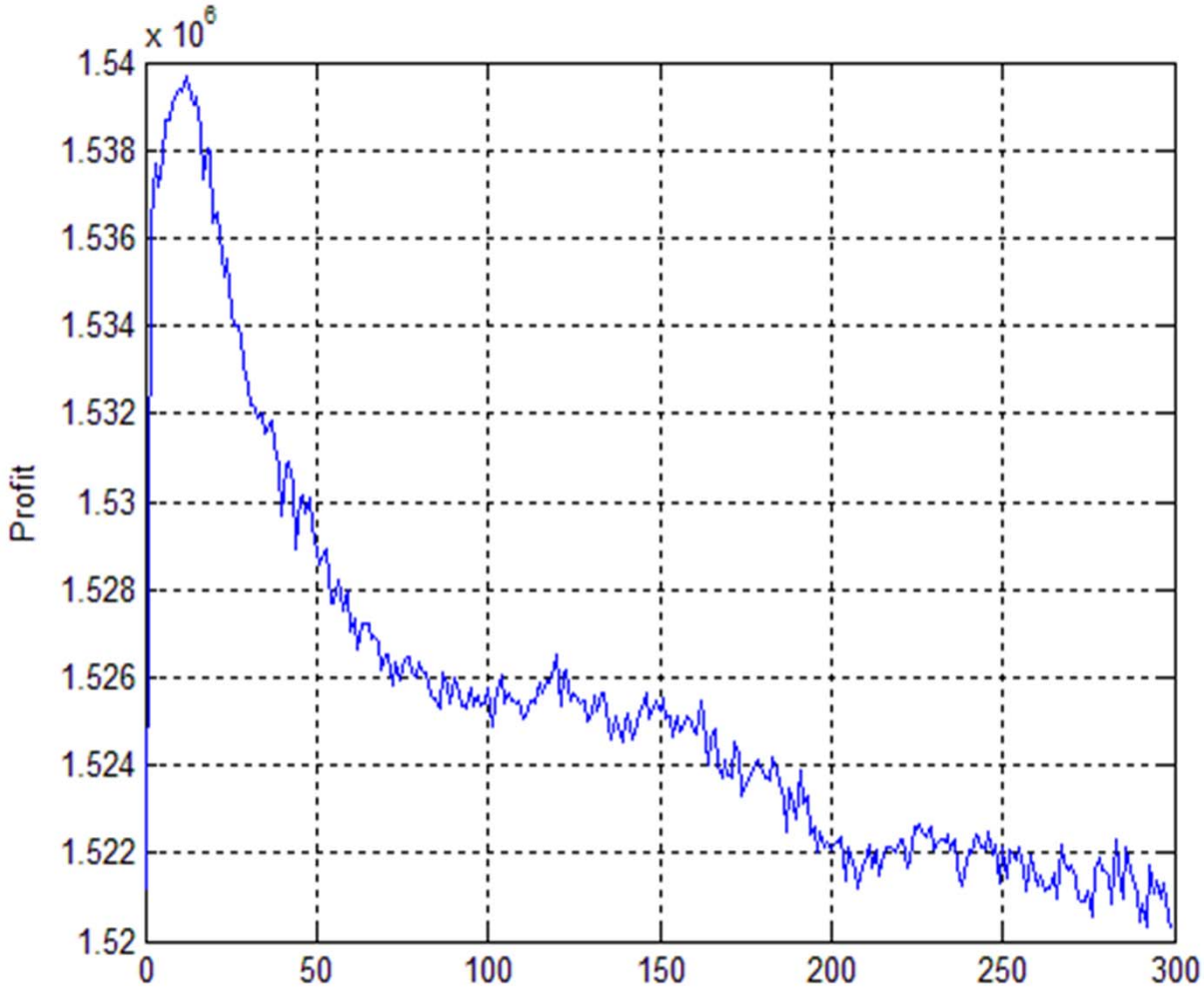
- *22000 zones*
- *2000 cars*
- *Battery capacity: 50KWh*
- *31560 Trips*





Driverless EV optimization

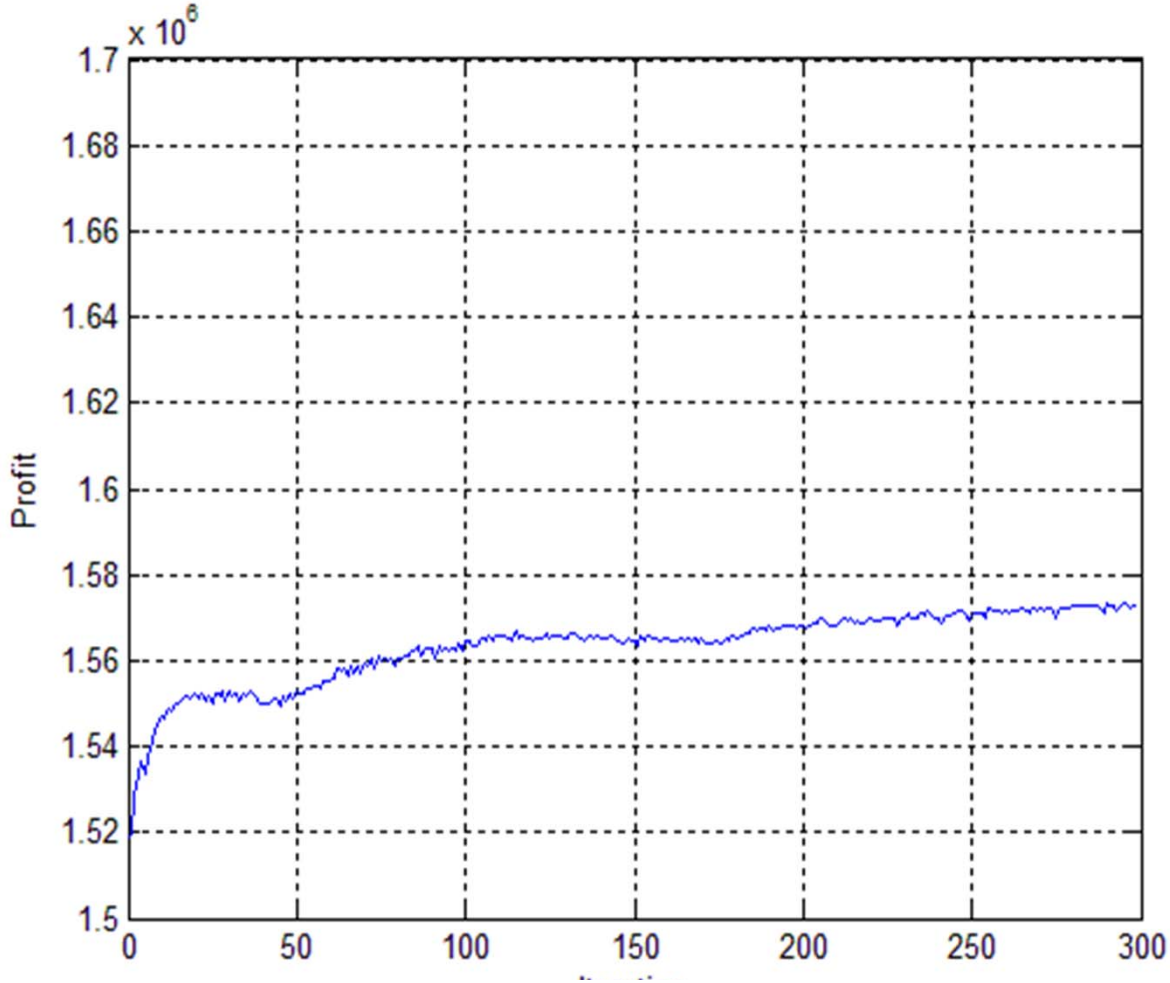
- No aggregation – (!!) Solution gets worse!



Profit versus number of iterations

Driverless EV optimization

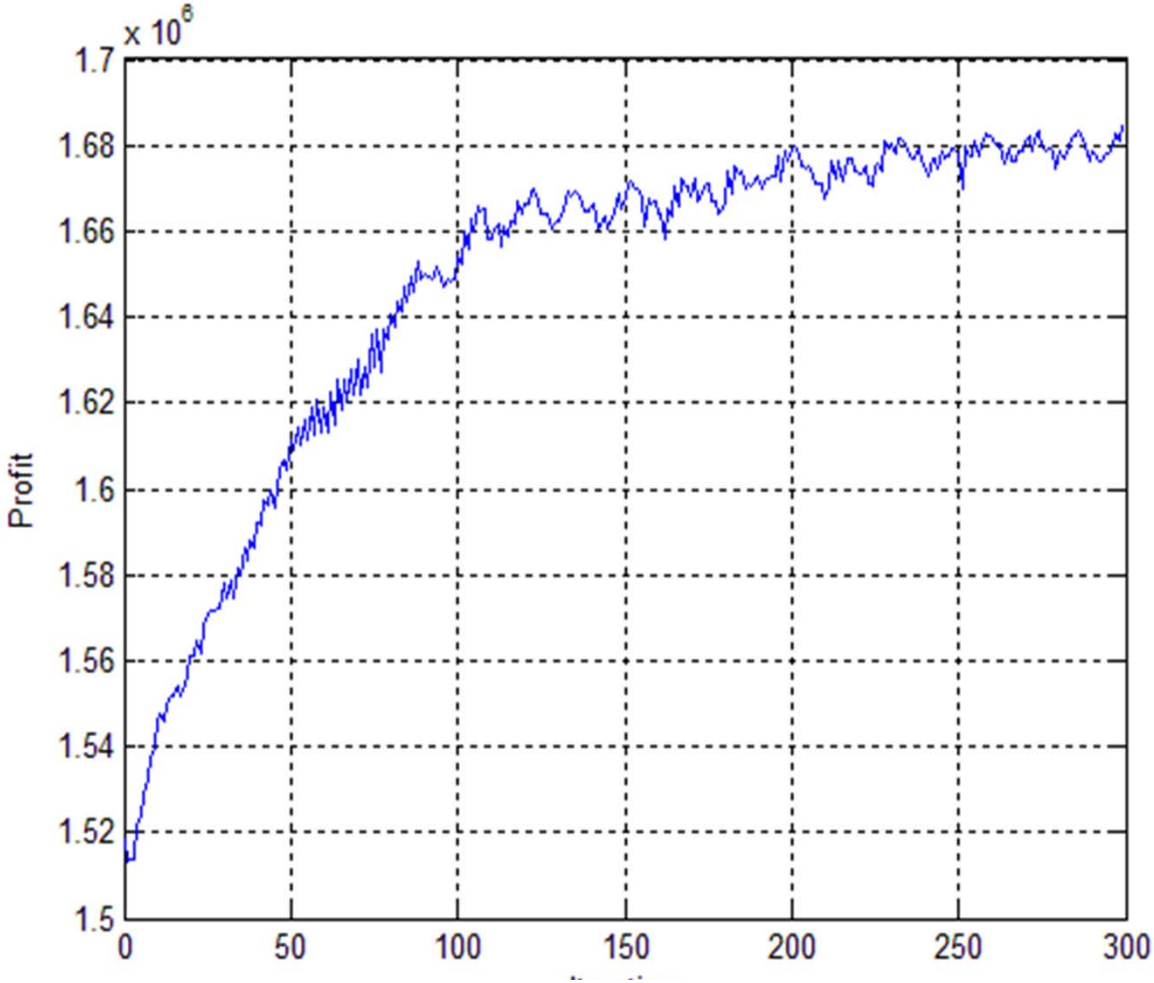
- Three levels of aggregation - Better



Profit versus number of iterations

Driverless EV optimization

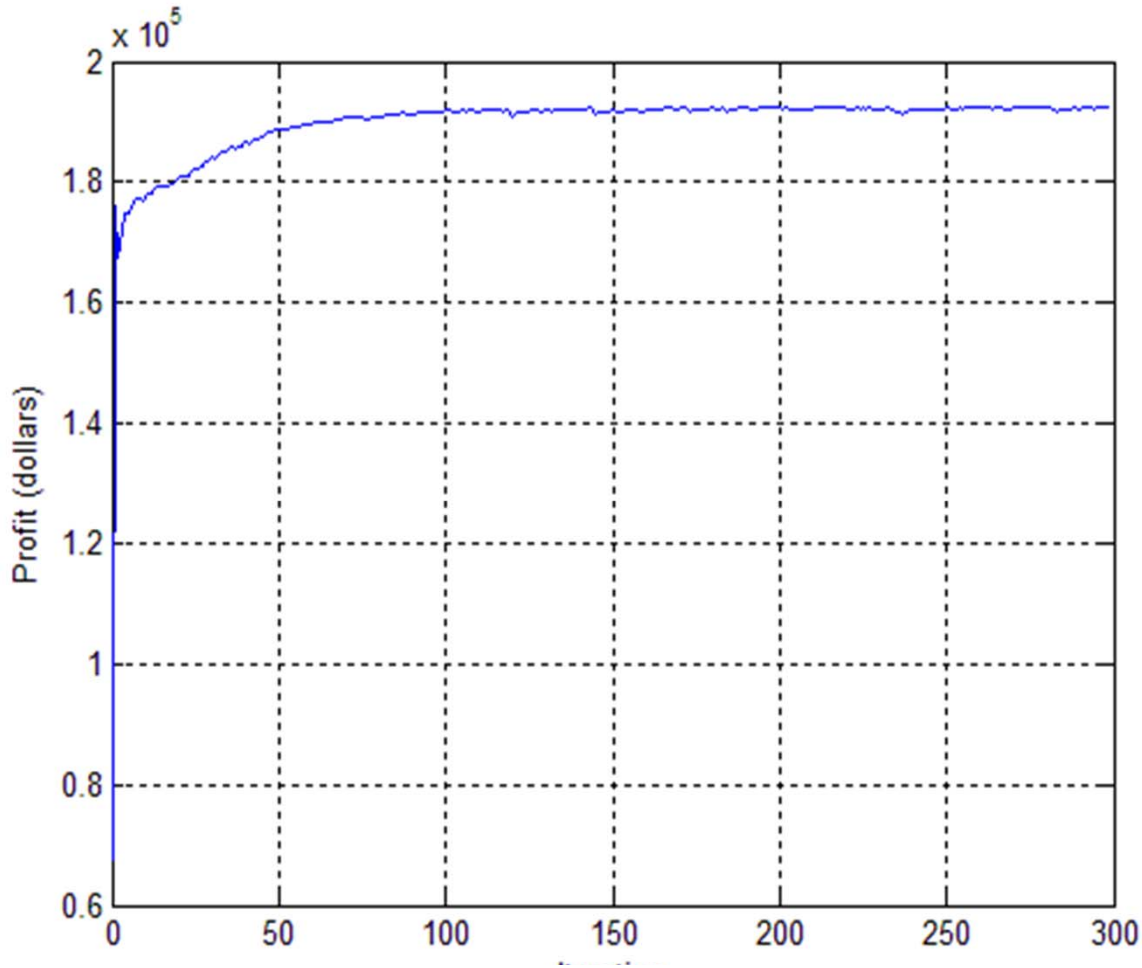
- Five levels of aggregation – old VFAs



Profit versus number of iterations

Driverless EV optimization

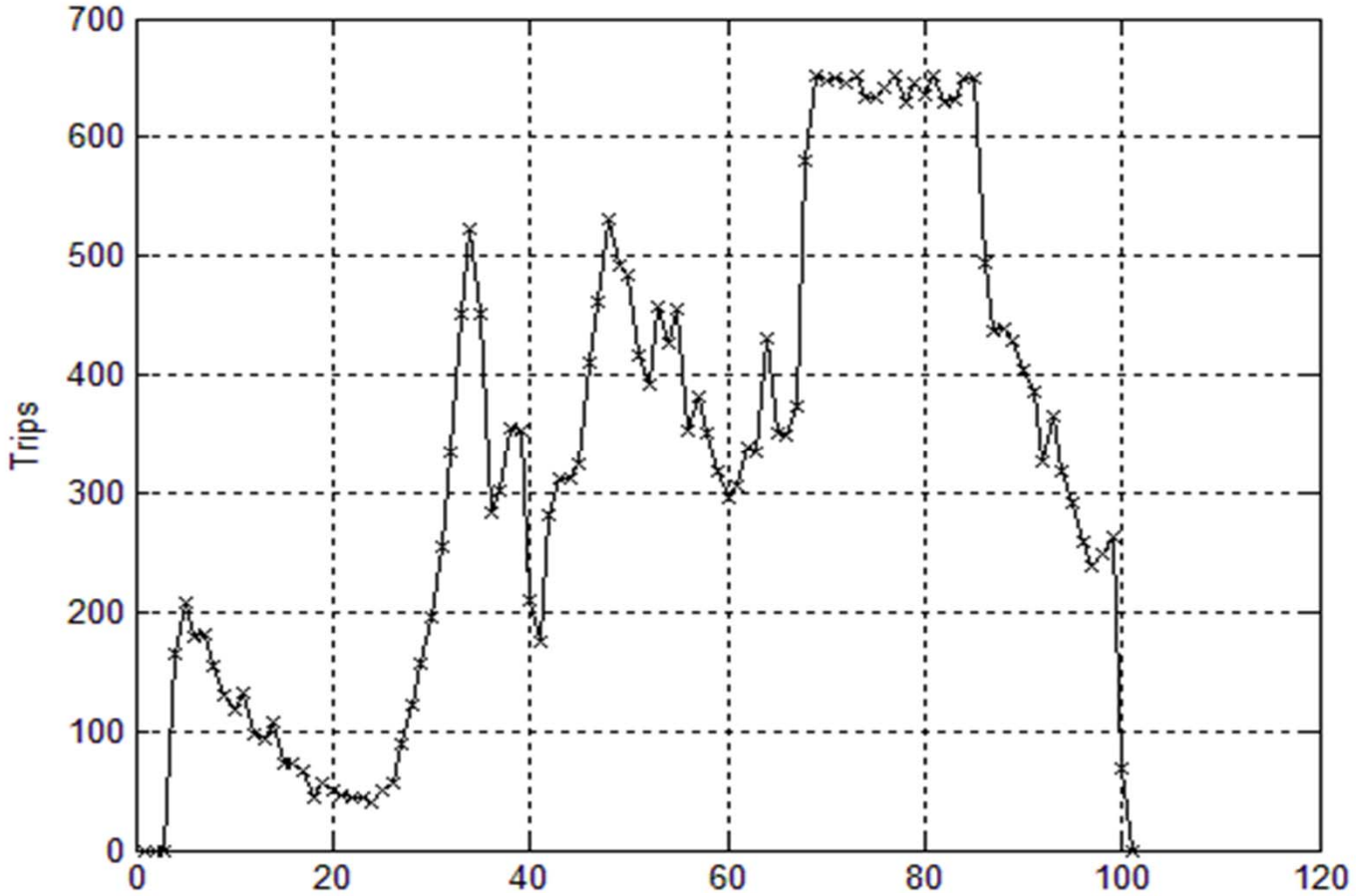
- Five levels of aggregation – new VFAs



Profit versus number of iterations

Driverless EV optimization

- Trip requests over time
  - » Challenge is to recharge during off-peak periods

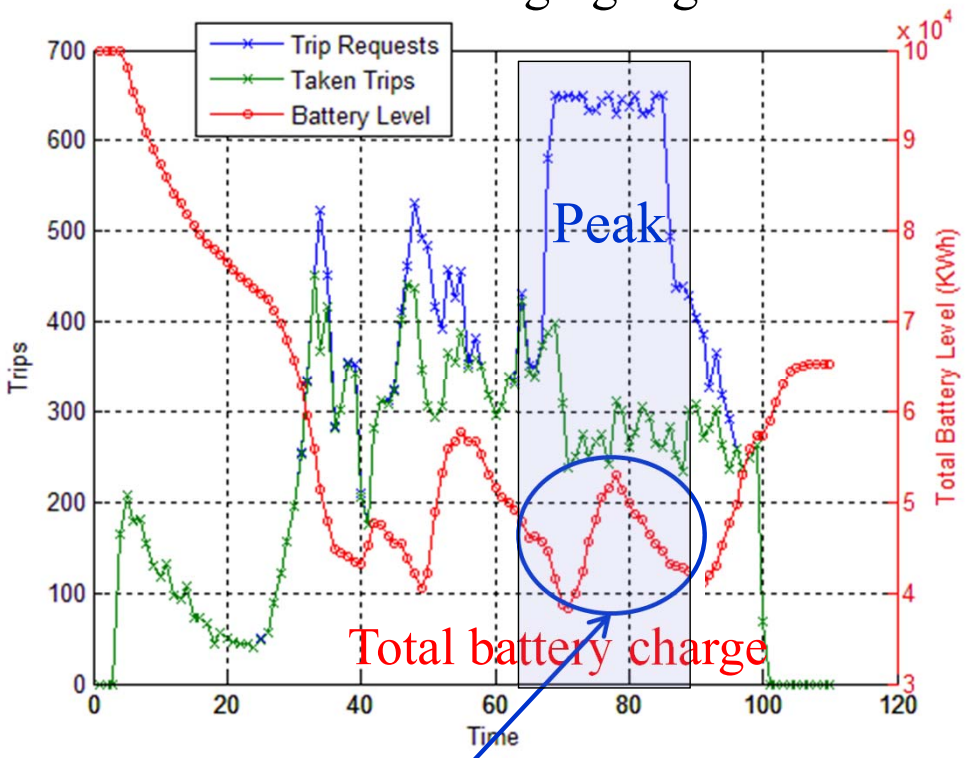


Trip Requests vs Time

Driverless fleets of EVs using ADP

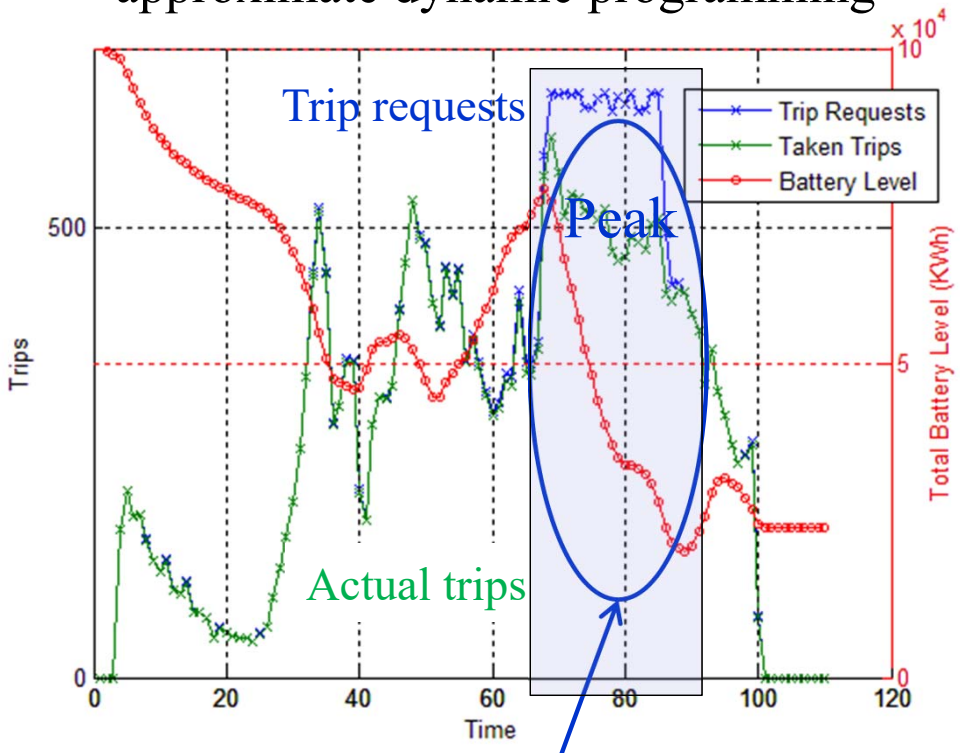
- Heuristic dispatch vs. ADP-based policies
  - » Effect of value function approximations on recharging

Heuristic recharging logic



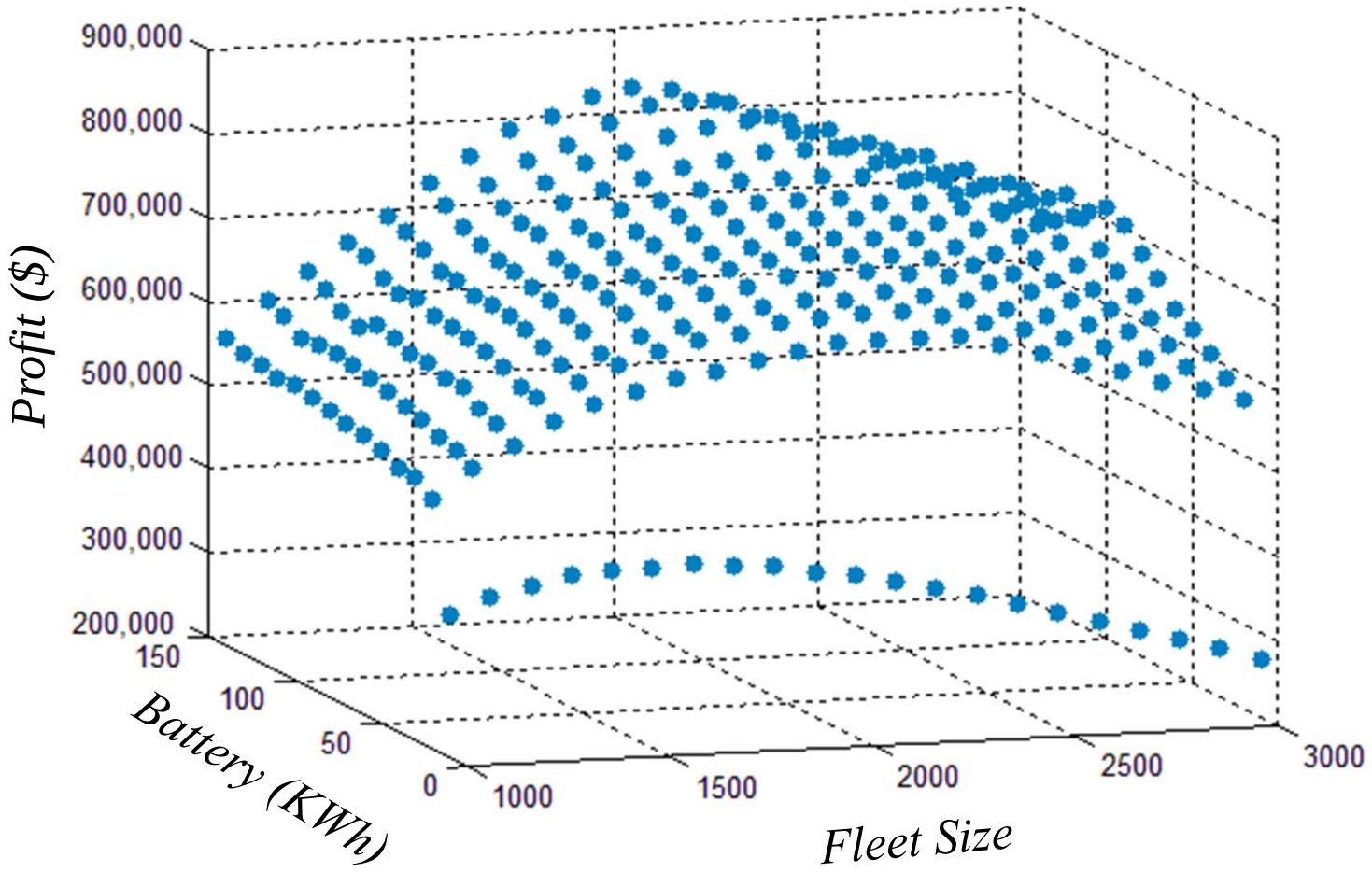
Recharging during peak period

Recharging controlled by approximate dynamic programming



No recharging during peak period

The economics of driverless fleets



We can simulate different fleet sizes and battery capacities, properly modeling recharging behaviors given battery capacity.

Approximate dynamic programming

Exploiting convexity in an inventory problem

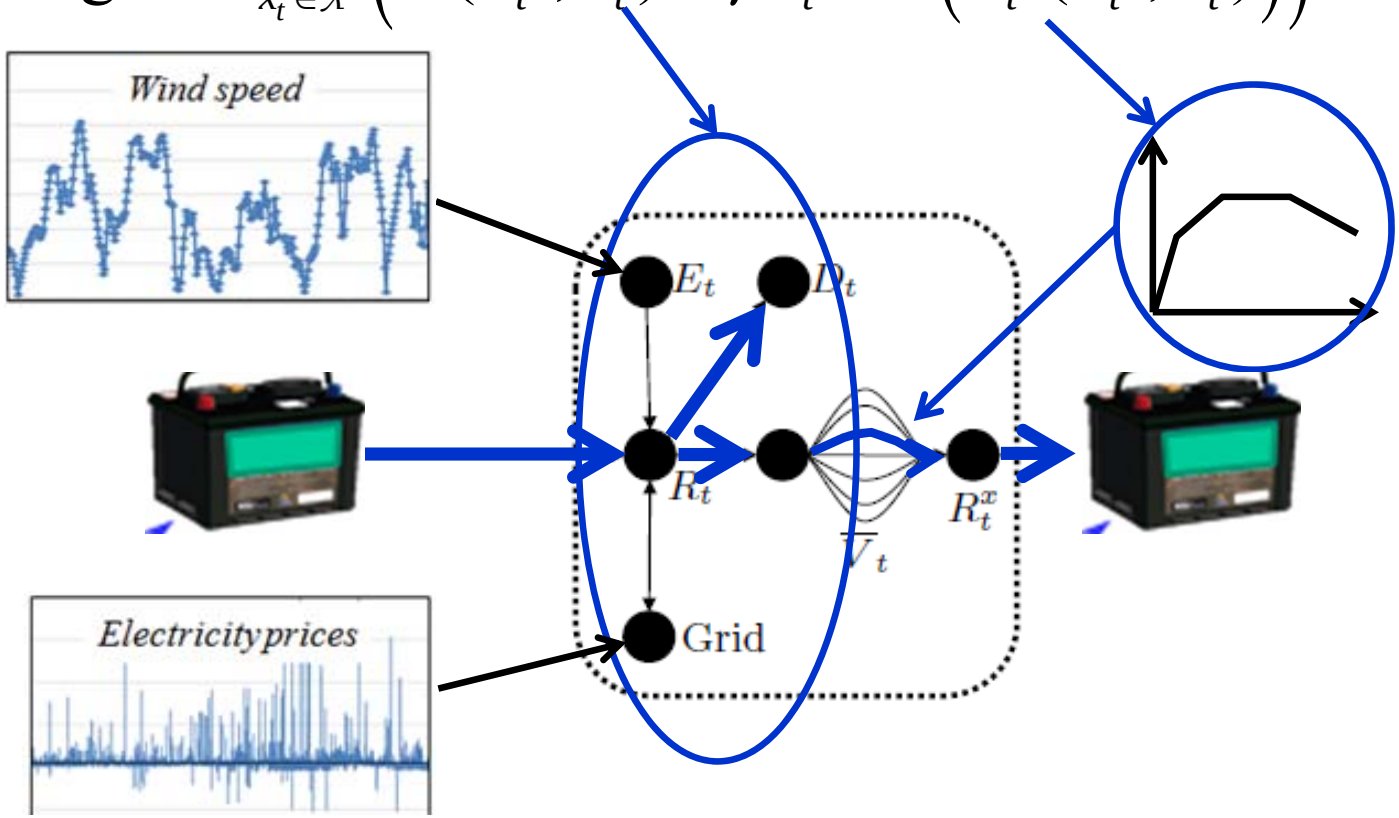
Exploiting convexity

- Single inventory problem
  - » With pre-decision state:

$$x_t^n = \arg\max_{x_t \in \mathcal{X}} \left(C(S_t^n, x_t) + \gamma \mathbb{E}\left\{ \bar{V}_{t+1}^{n-1}\left(S_{t+1}(S_t^n, x_t, W_{t+1}) \right) \mid S_t \right\} \right)$$

  - » With post-decision state:

$$x_t^n = \arg\max_{x_t \in \mathcal{X}} \left(C(S_t^n, x_t) + \gamma \bar{V}_t^{x,n-1}\left(S_t^x(S_t^n, x_t) \right) \right)$$



Exploiting convexity

- Updating strategies
  - » Forward pass (approximate value iteration) updating as we go (also known as TD(0)):

    - Compute the value of being in a state by "bootstrapping" the downstream value function approximation:

$$\hat{V}_t^n(S_t^n) = \max_{x_t \in \mathcal{X}} \left(C(S_t^n, x_t) + \gamma \bar{V}_t^{x,n-1}\left(S_t^x(S_t^n, x_t)\right)\right)$$

    - Now compute the marginal value using

$$\hat{v}_t^n = \frac{\hat{V}_t^n(S_t^n + \delta) - \hat{V}_t^n(S_t^n)}{\delta}$$

    - Use this to update the piecewise linear value functions.

Exploiting convexity

- Updating strategies
  - » Double pass:
    - Perform forward pass simulating the policy for $t = 0, \ldots, T$:

$$x_t^n = \arg\max_{x_t \in \mathcal{X}} \left(C(S_t^n, x_t) + \gamma \bar{V}_t^{x,n-1} \left(S_t^x (S_t^n, x_t) \right) \right)$$

    - Now perturb state by δ and solve again:

$$x_t^{+\delta,n} = \arg\max_{x_t \in \mathcal{X}} \left(C(S_t^n + \delta, x_t) + \gamma \bar{V}_t^{x,n-1} \left(S_t^x (S_t^n + \delta, x_t) \right) \right)$$

Update $S_{t+1}^n = S_t^n + x_t^n$ (remember x_t^n may be negative). Store S_t^n as you proceed. Also store incremental cost

$$\delta C_t^n = C(S_t^n + \delta, x_t^{+\delta,n}) - C(S_t^n, x_t^n)$$

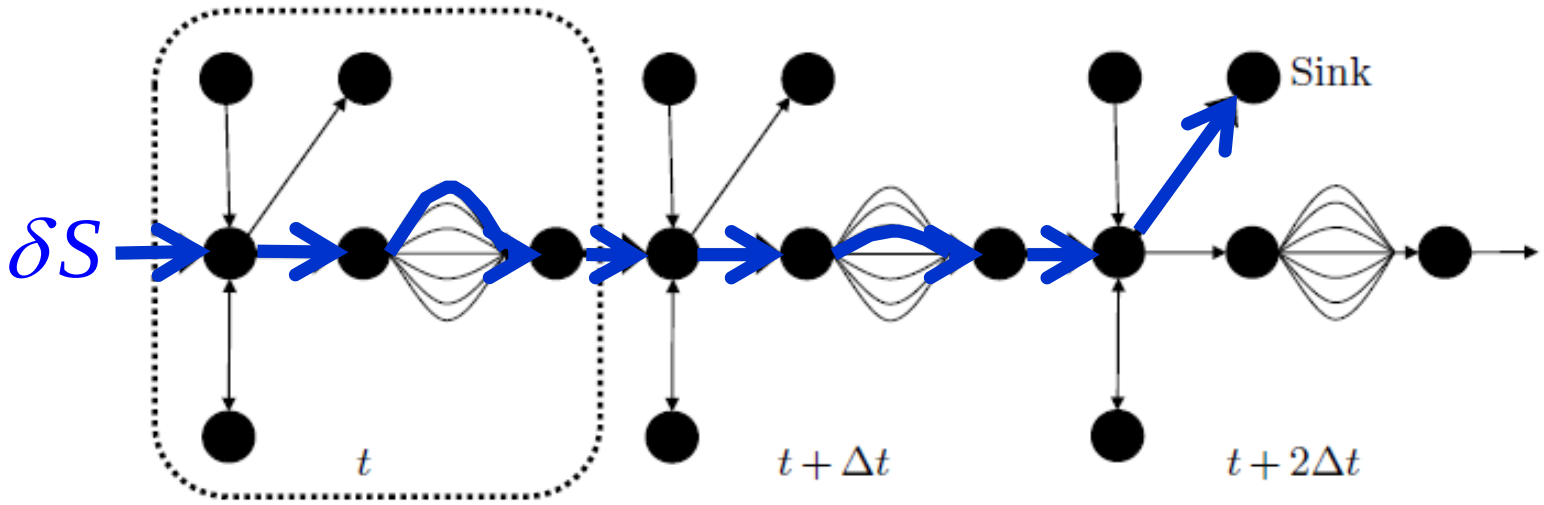
    - Perform backward pass, computing marginal values (and performing updates):

$$\hat{v}_t^n = \delta C(S_t^n, x_t^n) + \hat{v}_{t+1}^n$$

$$\bar{V}_{t-1}^n (S_{t-1}^n) \leftarrow U^V (\bar{V}_{t-1}^{n-1} (S_{t-1}^n), S_{t-1}^n, \hat{v}_t^n)$$

Exploiting convexity

- We update the piecewise linear value functions by computing estimates of slopes using a backward pass:



» The cost along the marginal path is the derivative of the simulation with respect to the flow perturbation.

Exploiting convexity

- From pre- to post-decision state

  » Get marginal value of inventory, $\hat{v}_t^n$, at time t for inventory level s_t^n during iteration n.

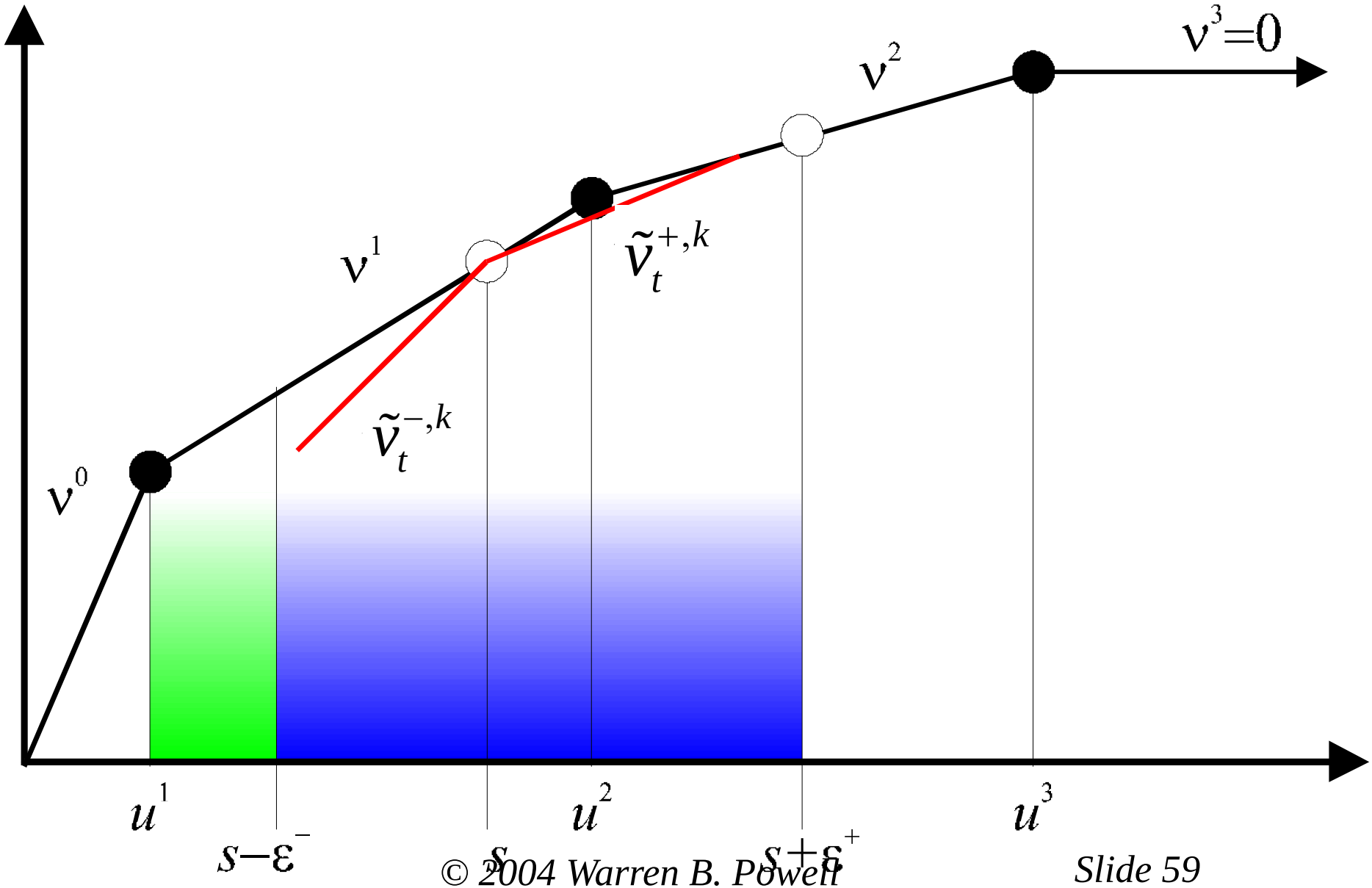
  » Use this to update the value function around the *previous post-decision state* $s_{t-1}^{x,n}$ to obtain $\bar{V}_{t-1}^{x,n}(s)$.

Exploiting convexity

- Updating strategies
  - » The choice between a pure forward pass (TD(0)) versus a double pass (TD(1)) is highly problem dependent. Every problem has a natural "horizon."
  - » We have found that in our fleet management problems, the pure forward pass works fine and is much easier to implement.
  - » For energy storage problems, it is essential that we use a double pass, since a decision at time t can have an impact hundreds of time periods into the future.

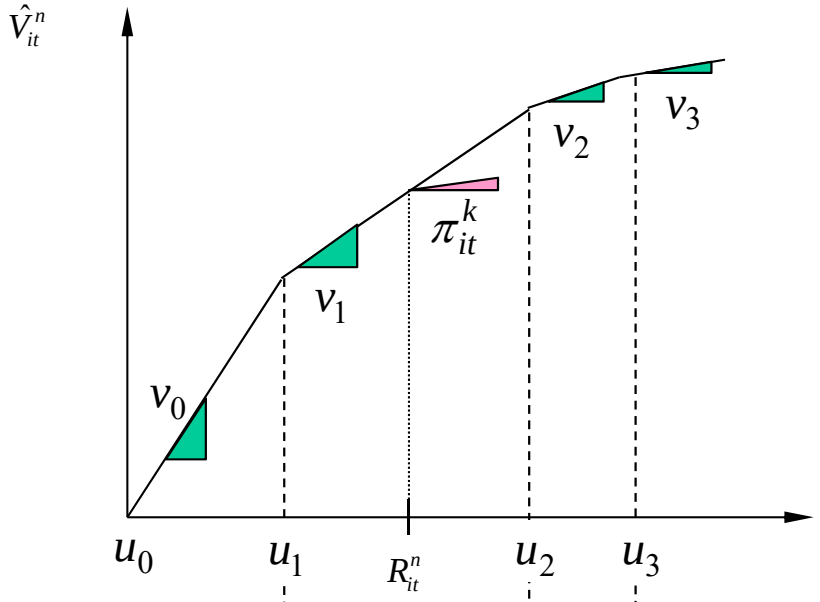
Exploiting convexity

- It is important to maintain concavity:



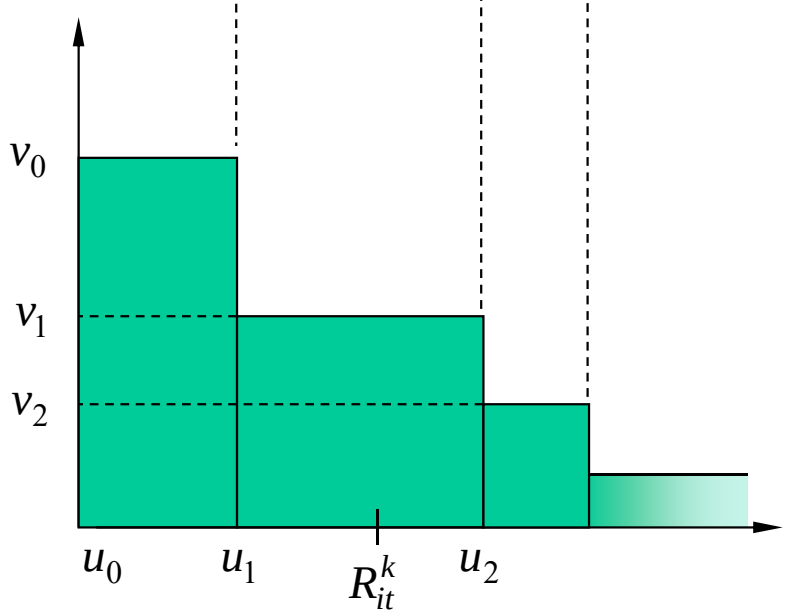
Exploiting convexity

Value function



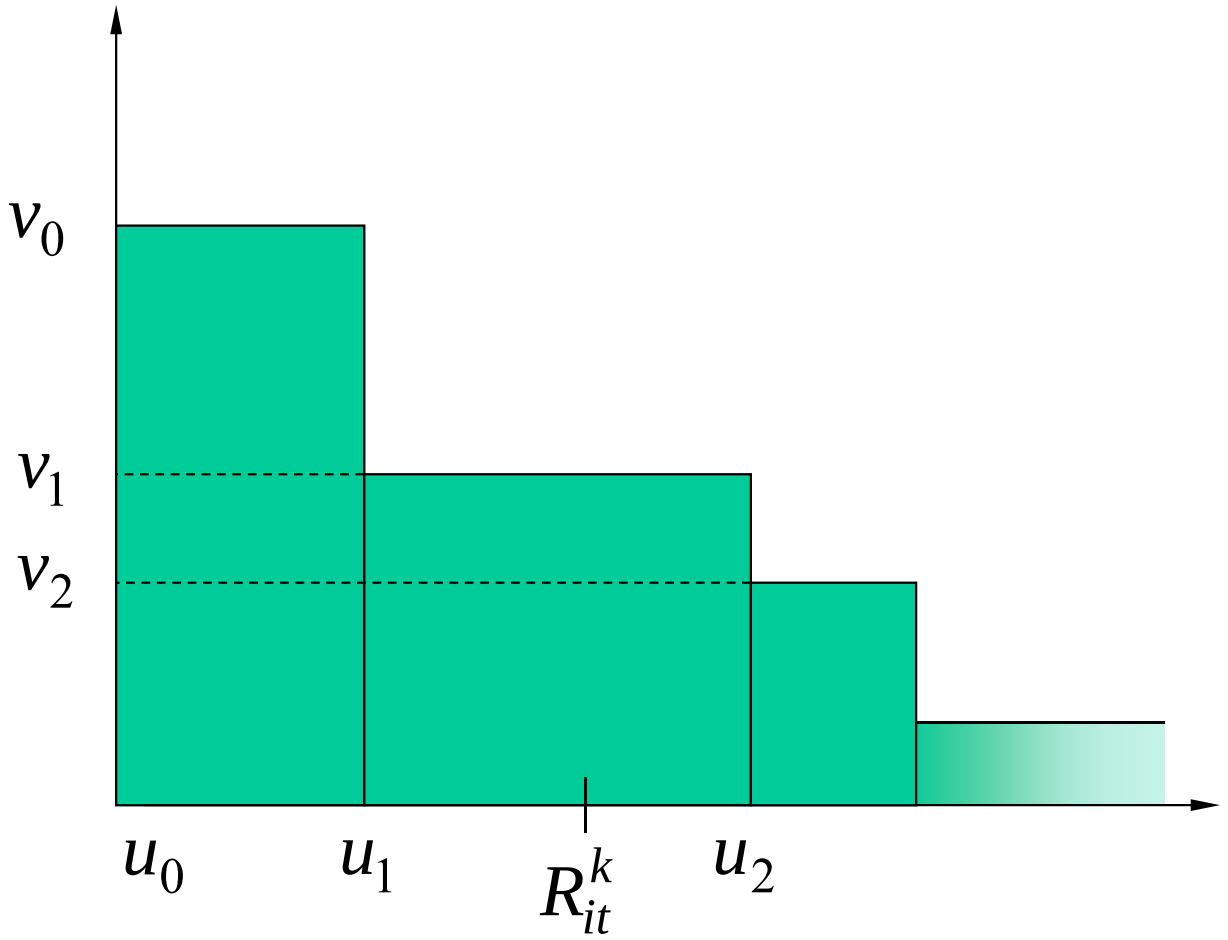
A concave function…

Slopes

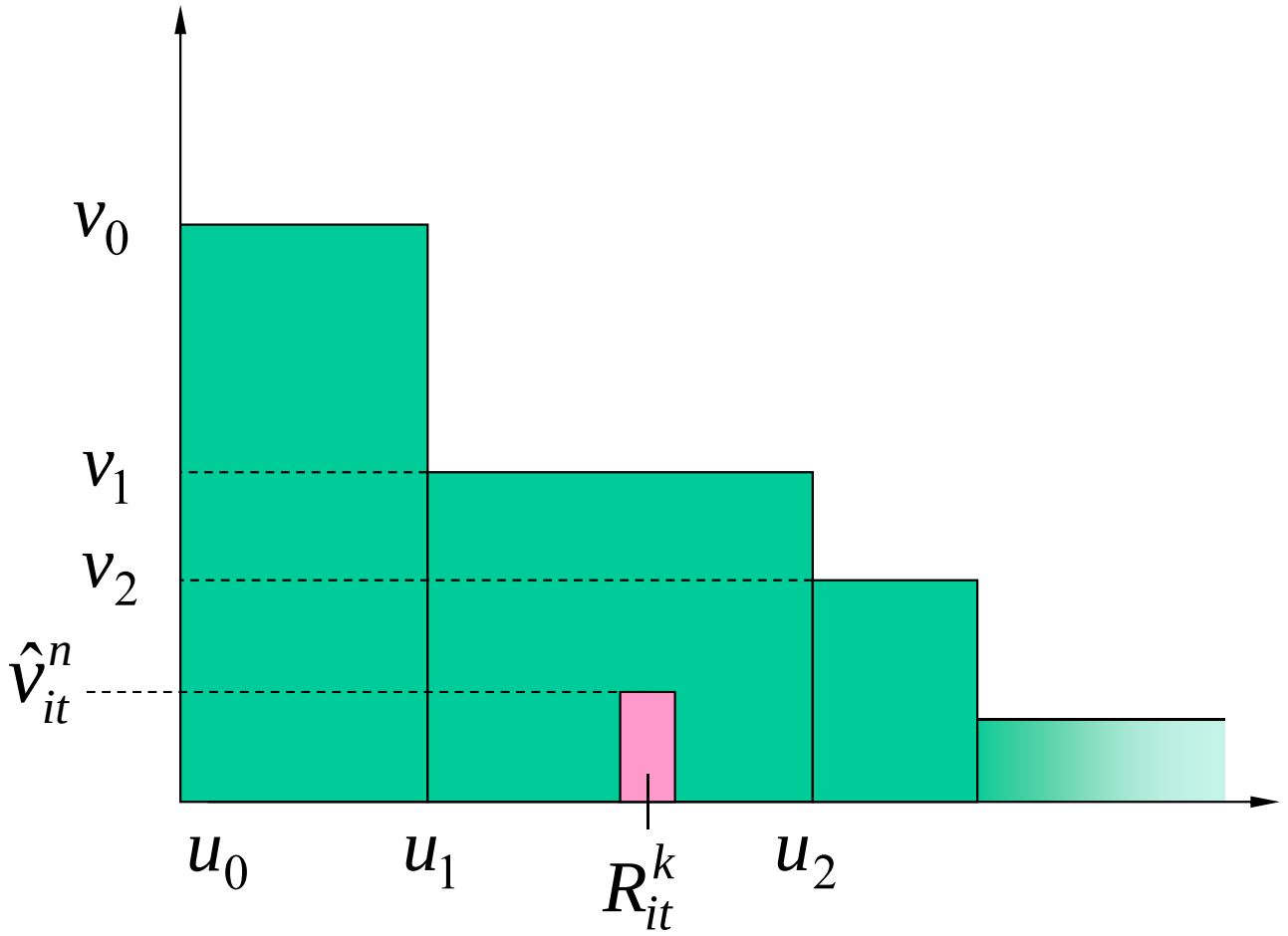


… has monotonically decreasing slopes. But updating the function with a stochastic gradient may violate this property.

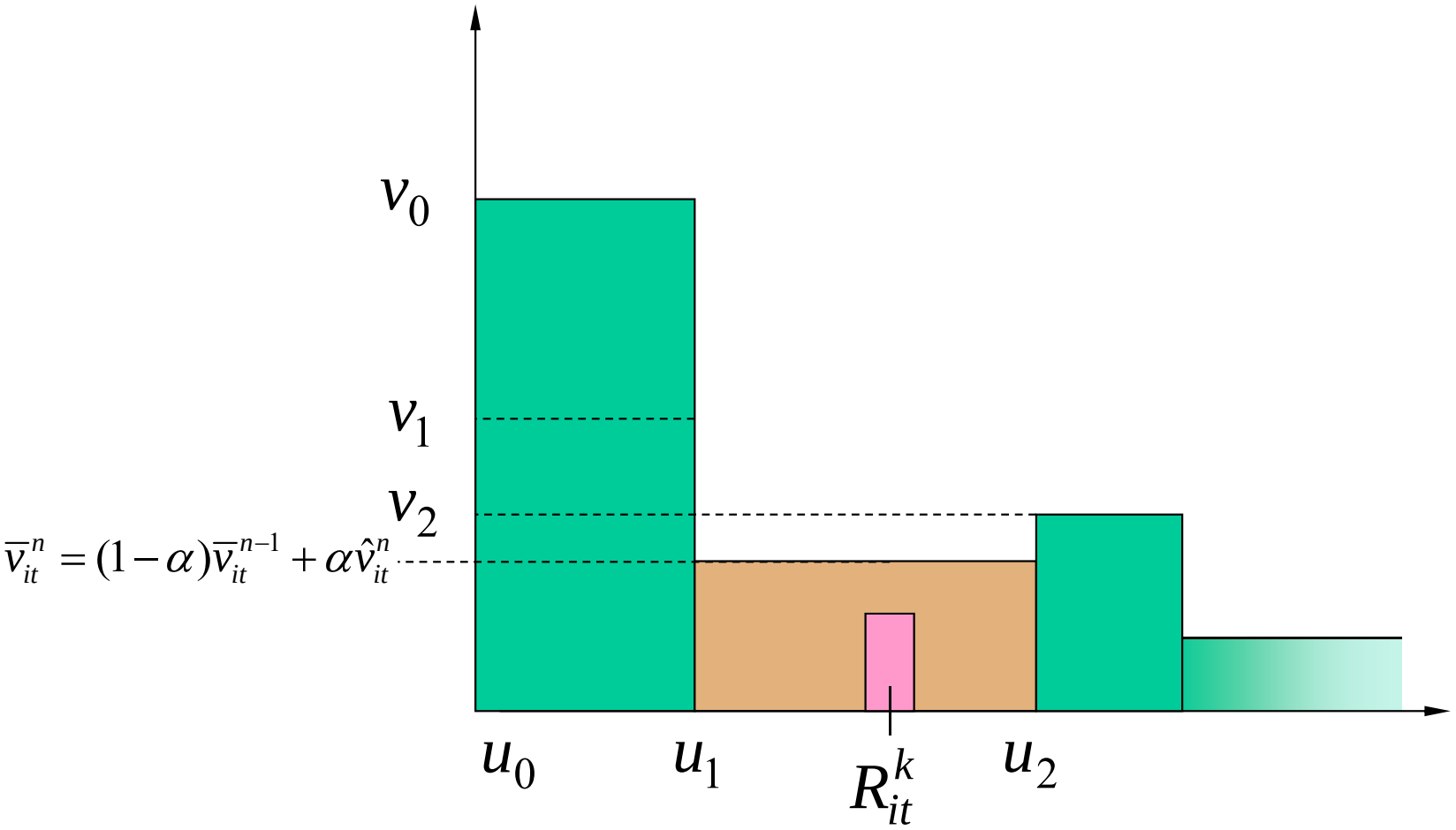
Exploiting convexity



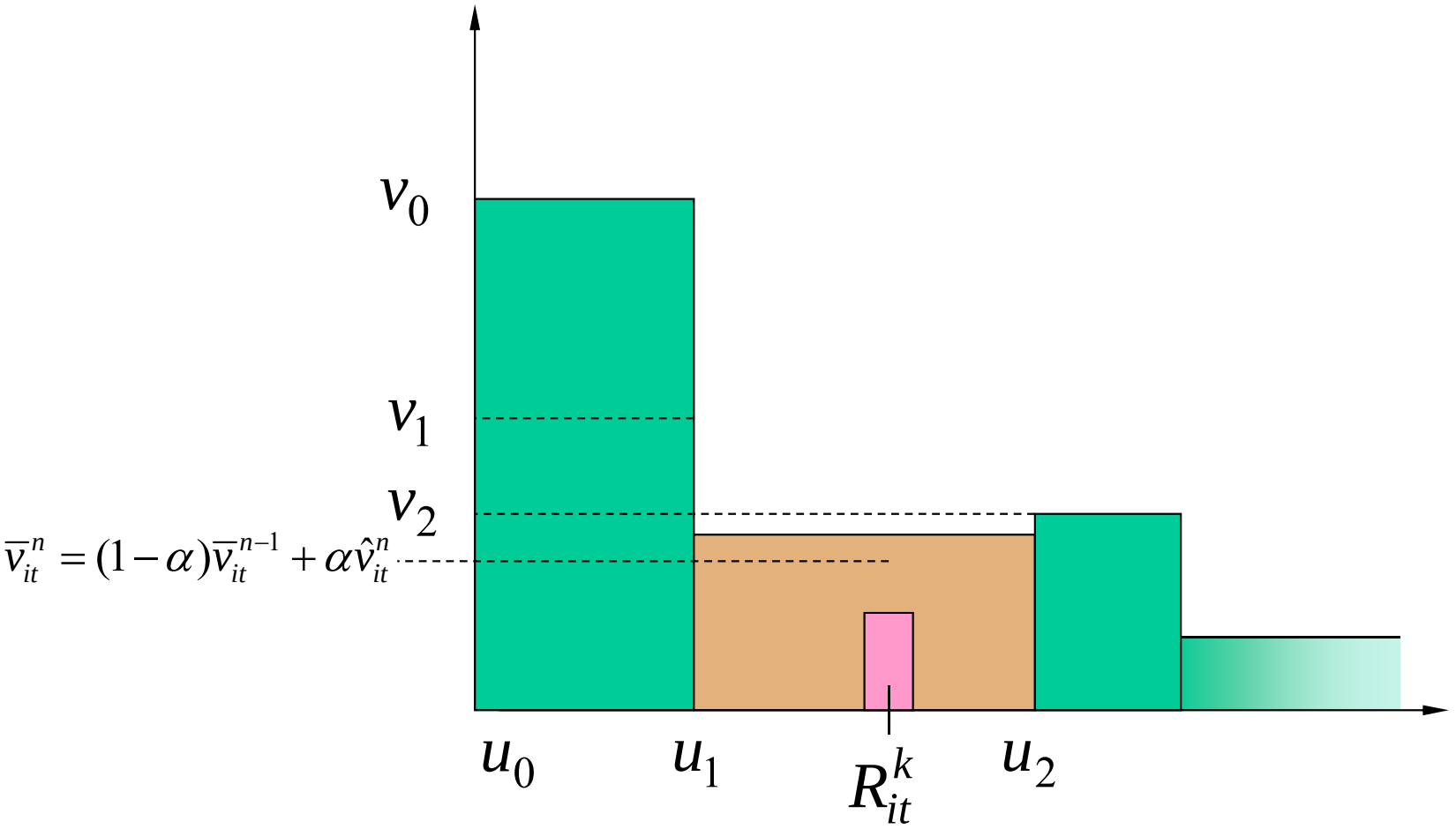
Exploiting convexity



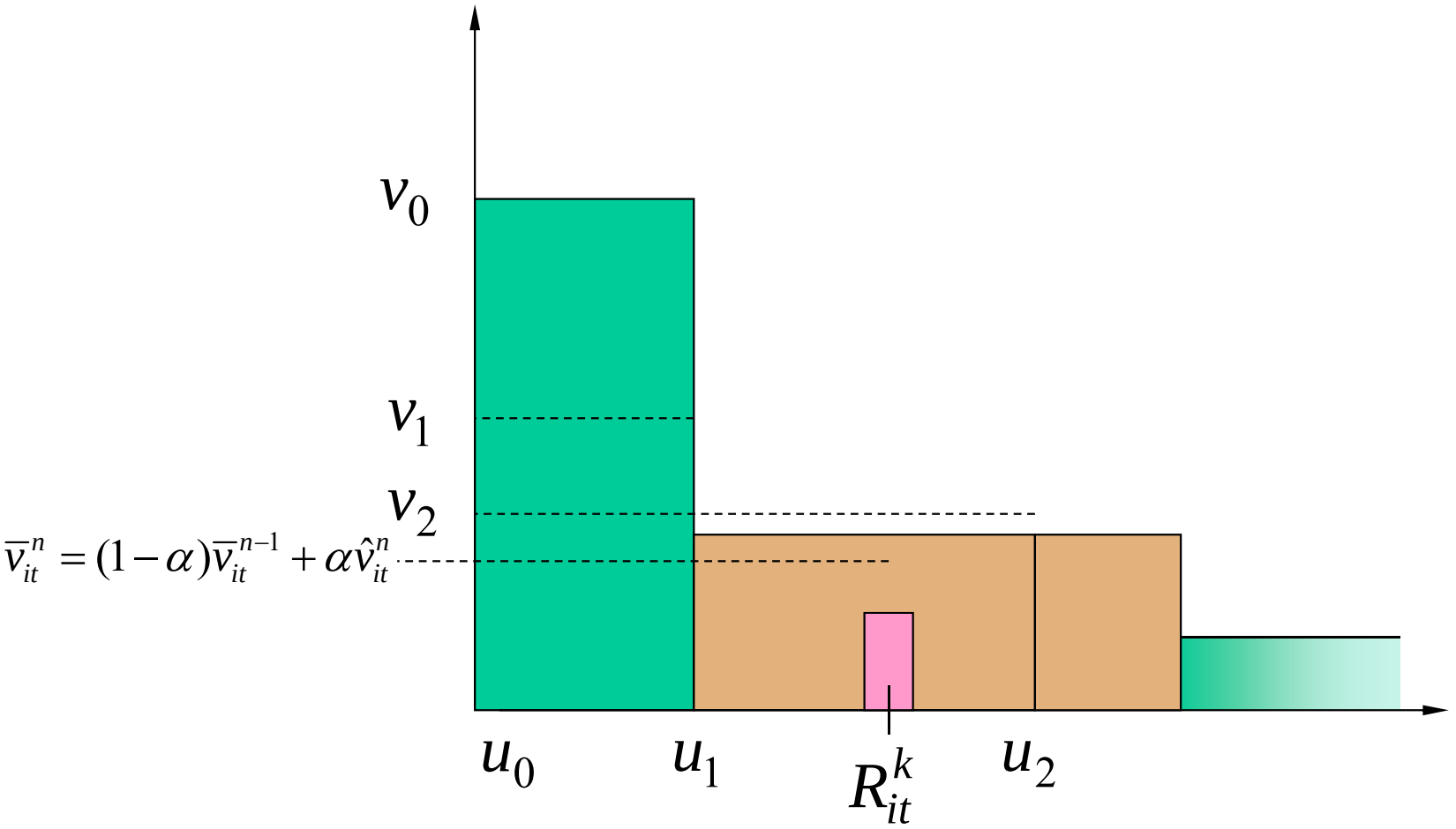
Exploiting convexity



Exploiting convexity



Exploiting convexity

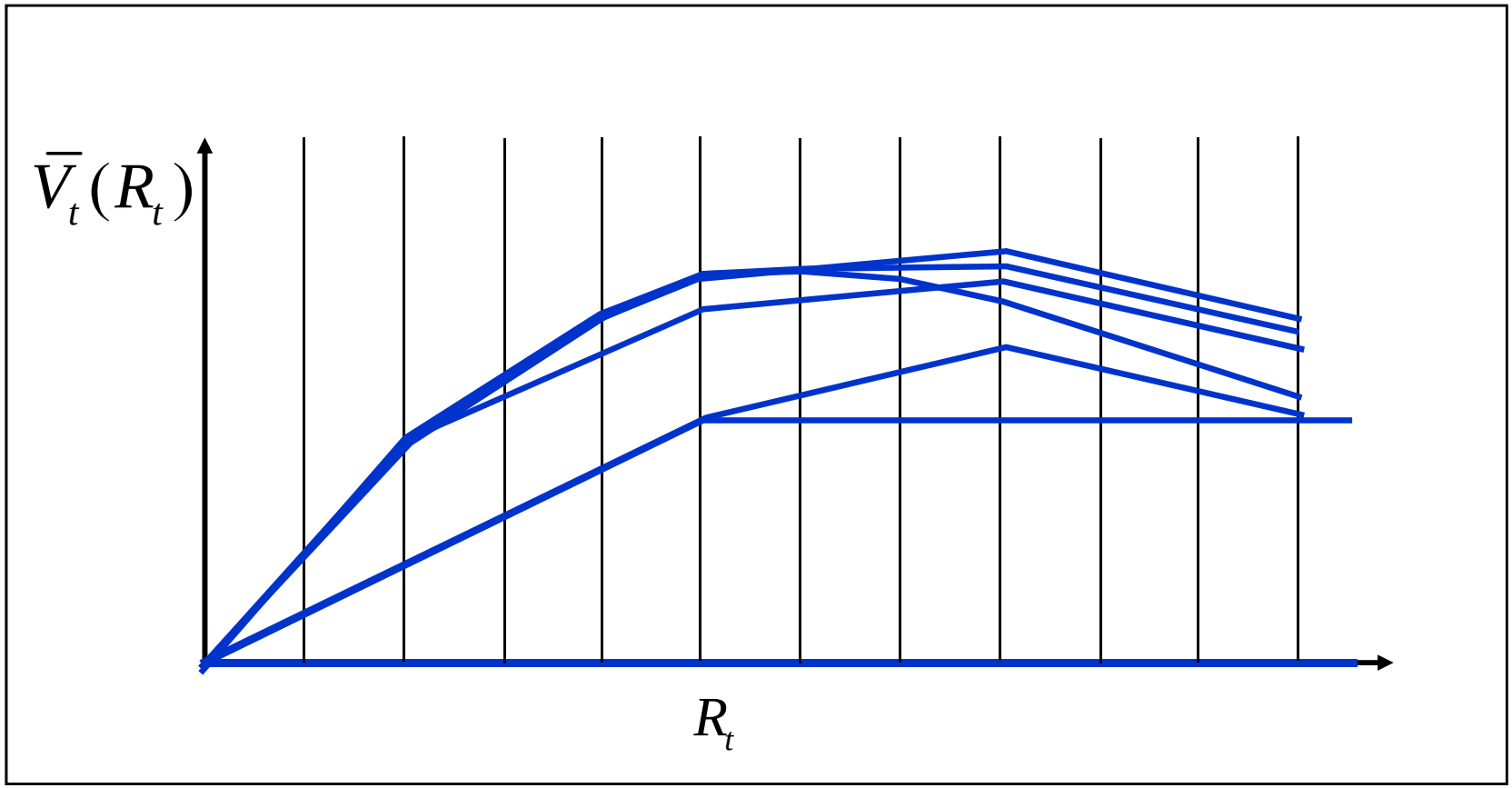


Exploiting convexity

- Ways for maintaining concavity (monotonicity in the slopes):
  - » CAVE algorithm – Use updated slope at one point to update over a range $+/-\delta^n$ which shrinks with iterations (shrinking factor is tunable parameter). Expand δ if needed to ensure concavity is maintained.
    - Works well! But we could never prove convergence.
  - » Leveling algorithm – Update at a point x^n.
    - Force monotonicity in slopes by increasing/decreasing slopes farther from x^n as needed.
    - Works fine, without tunable parameters
  - » SPAR algorithm – Perform nearest point projection onto space of monotone functions.
    - Nice theoretical convergence proof – could never get it to work.

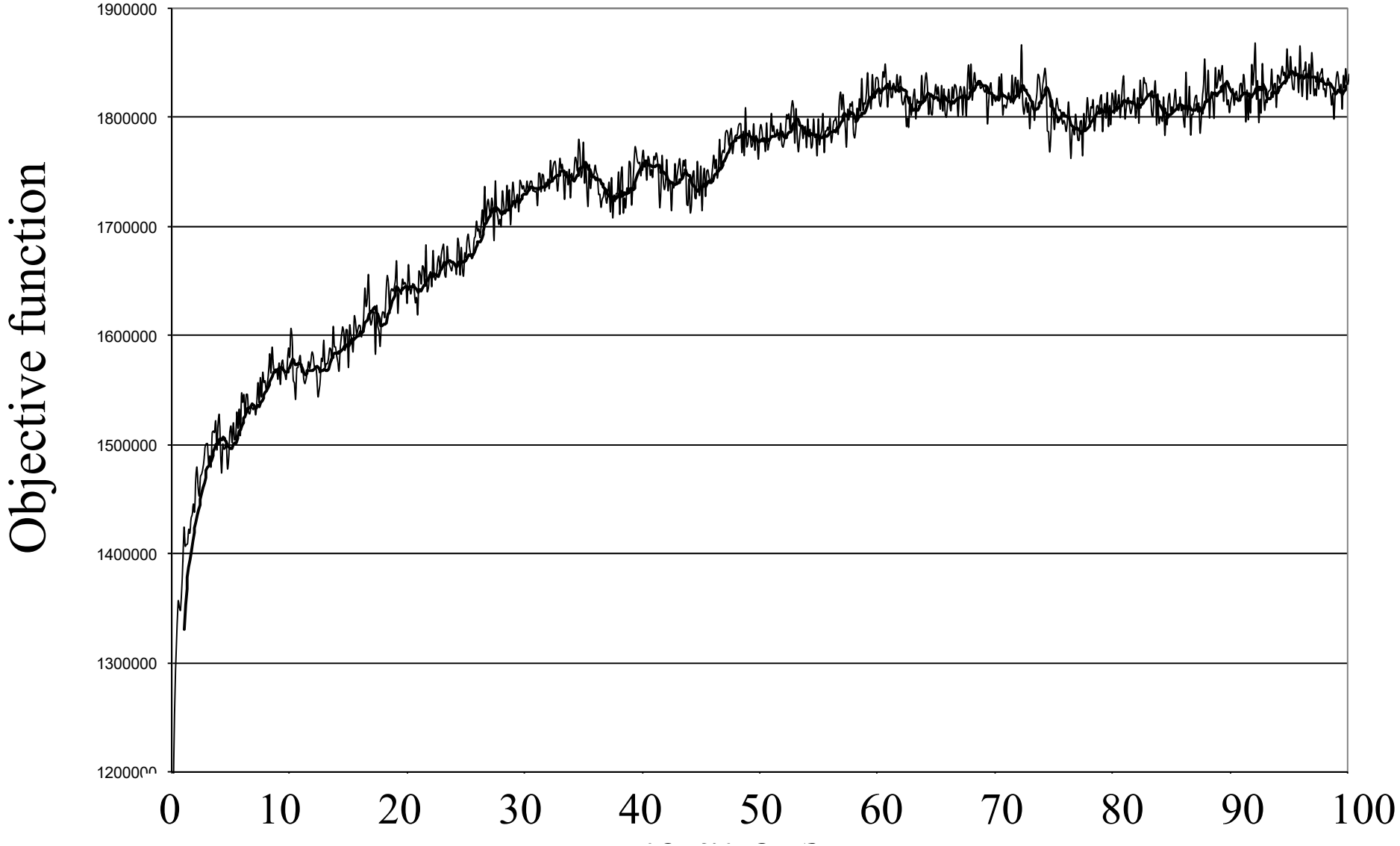
Exploiting convexity

- Derivatives are used to estimate a piecewise linear approximation



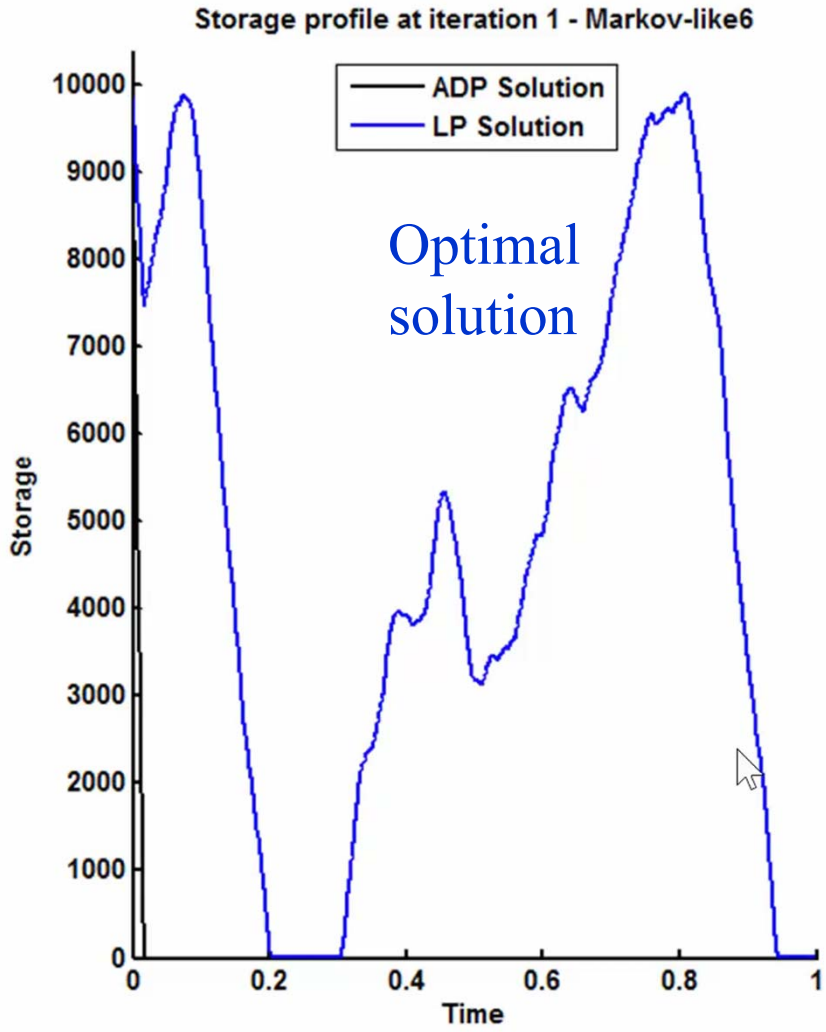
Exploiting convexity

- With luck, your objective function improves



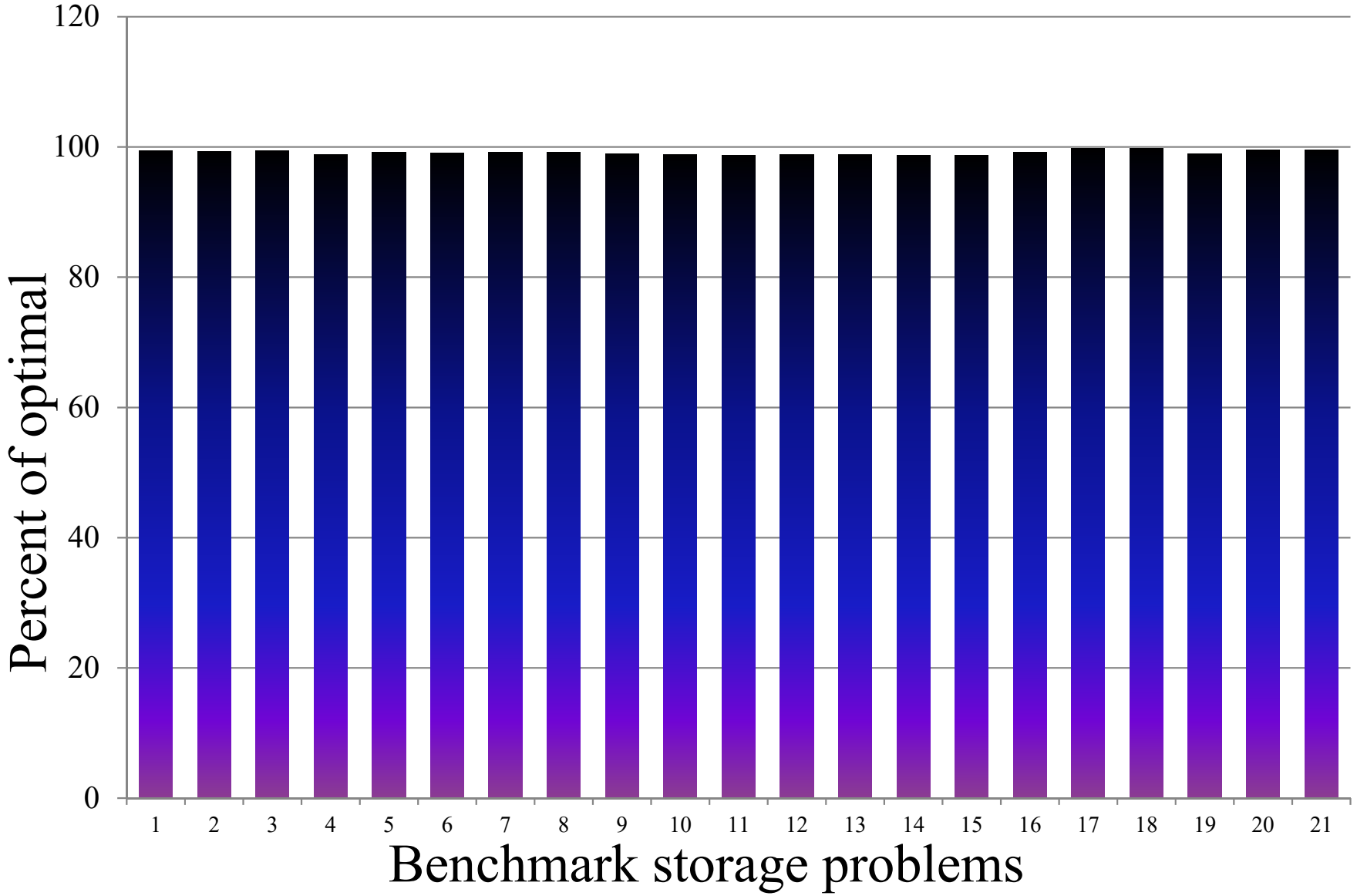
Exploiting convexity

- Testing on a time-dependent, deterministic problem
  - » Blue = optimal (found by solving a single linear program over entire horizon)
  - » Black = ADP solution



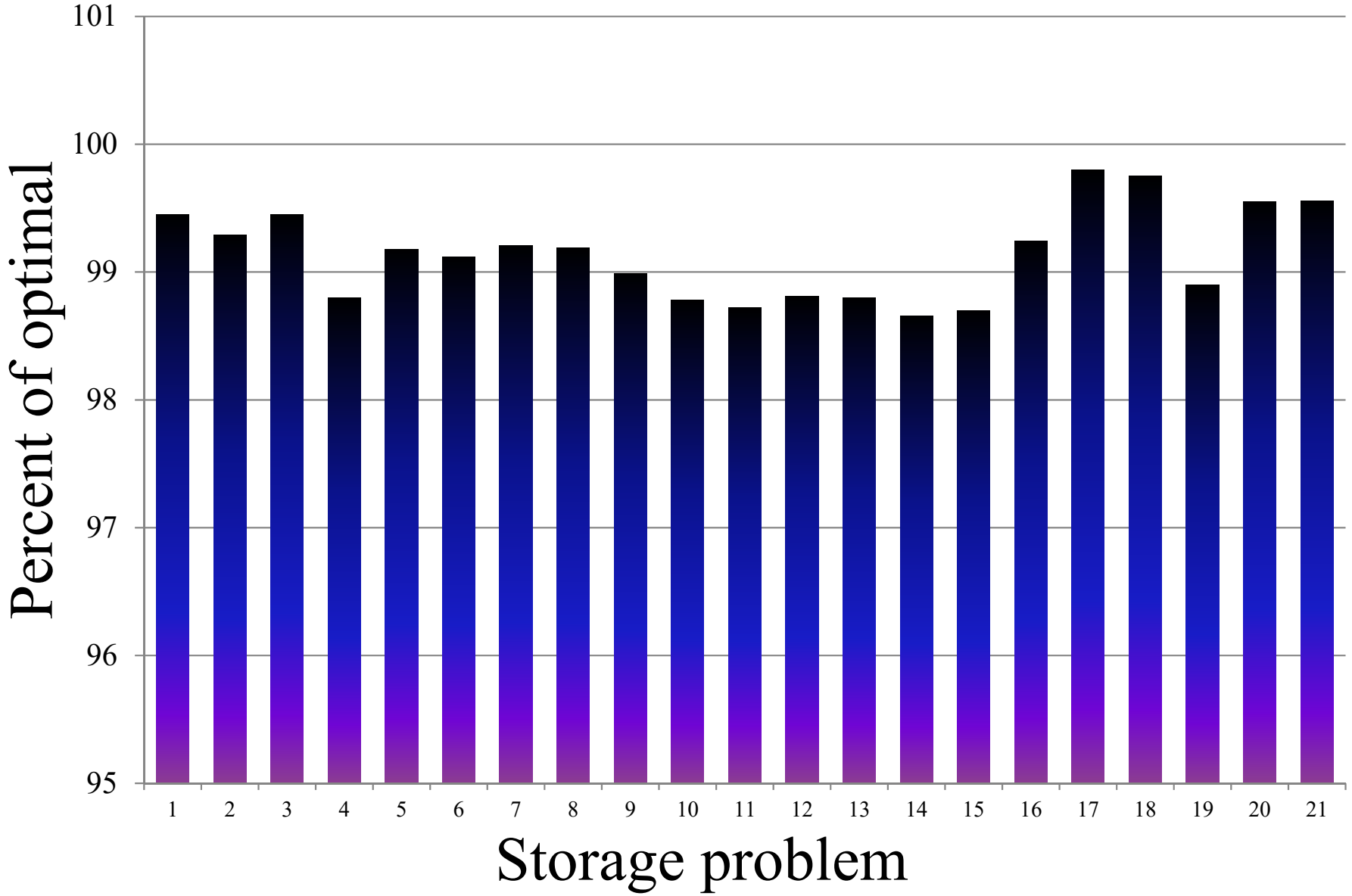
Exploiting convexity

- Stochastic, time-dependent problems



Exploiting convexity

- Stochastic, time-dependent problems

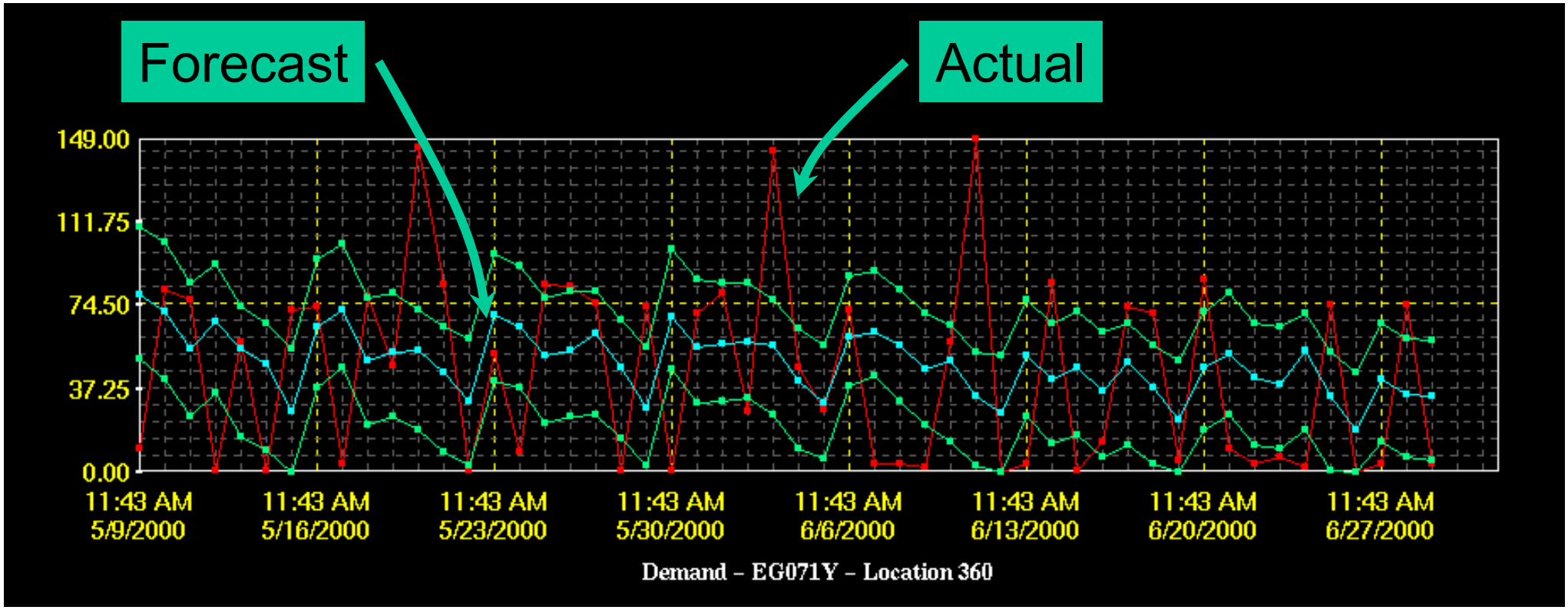


Approximate dynamic programming

Resource allocation (freight cars) for two-stage problem

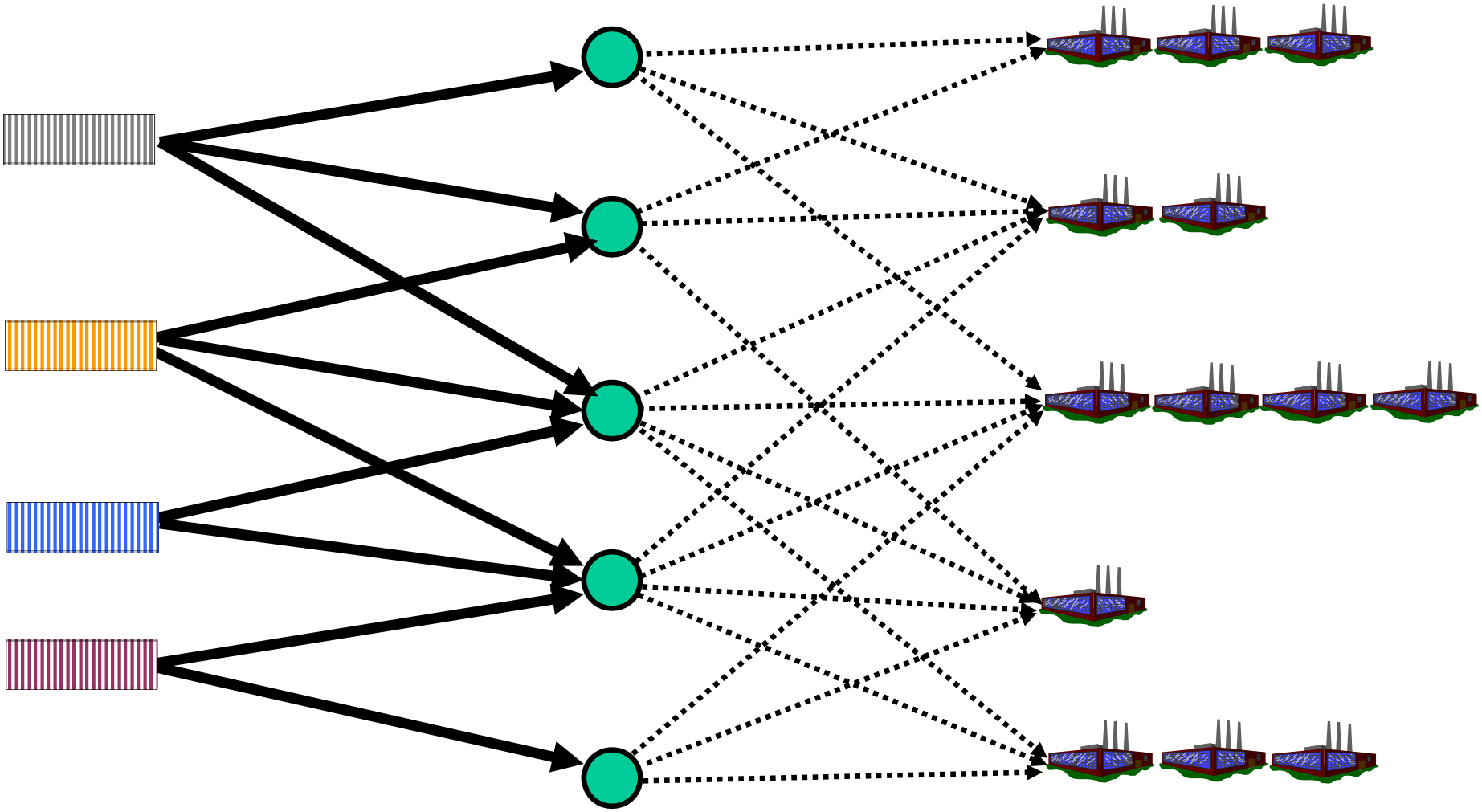

7044
7044

Forecasts of Car Demands



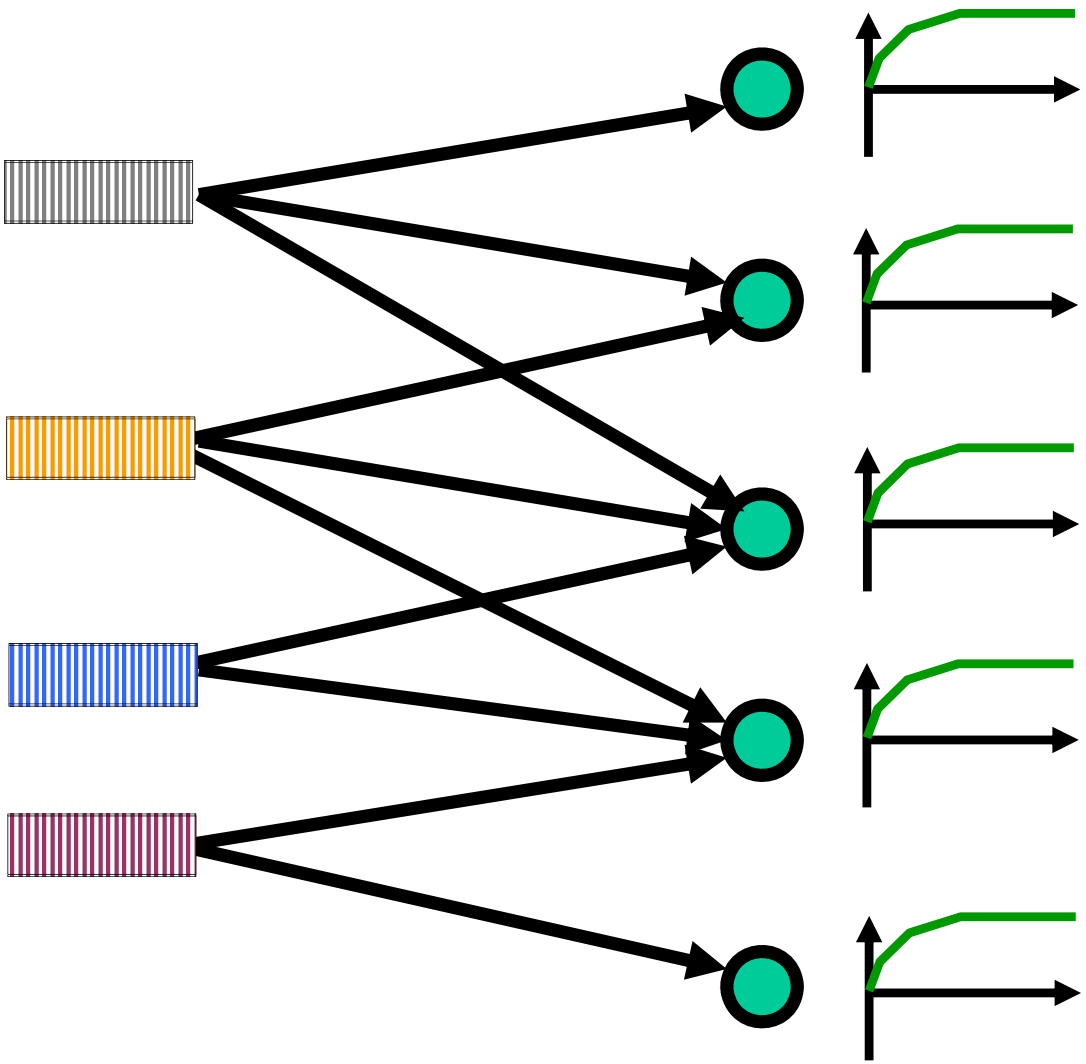
Two-stage problems

- Two-stage resource allocation under uncertainty



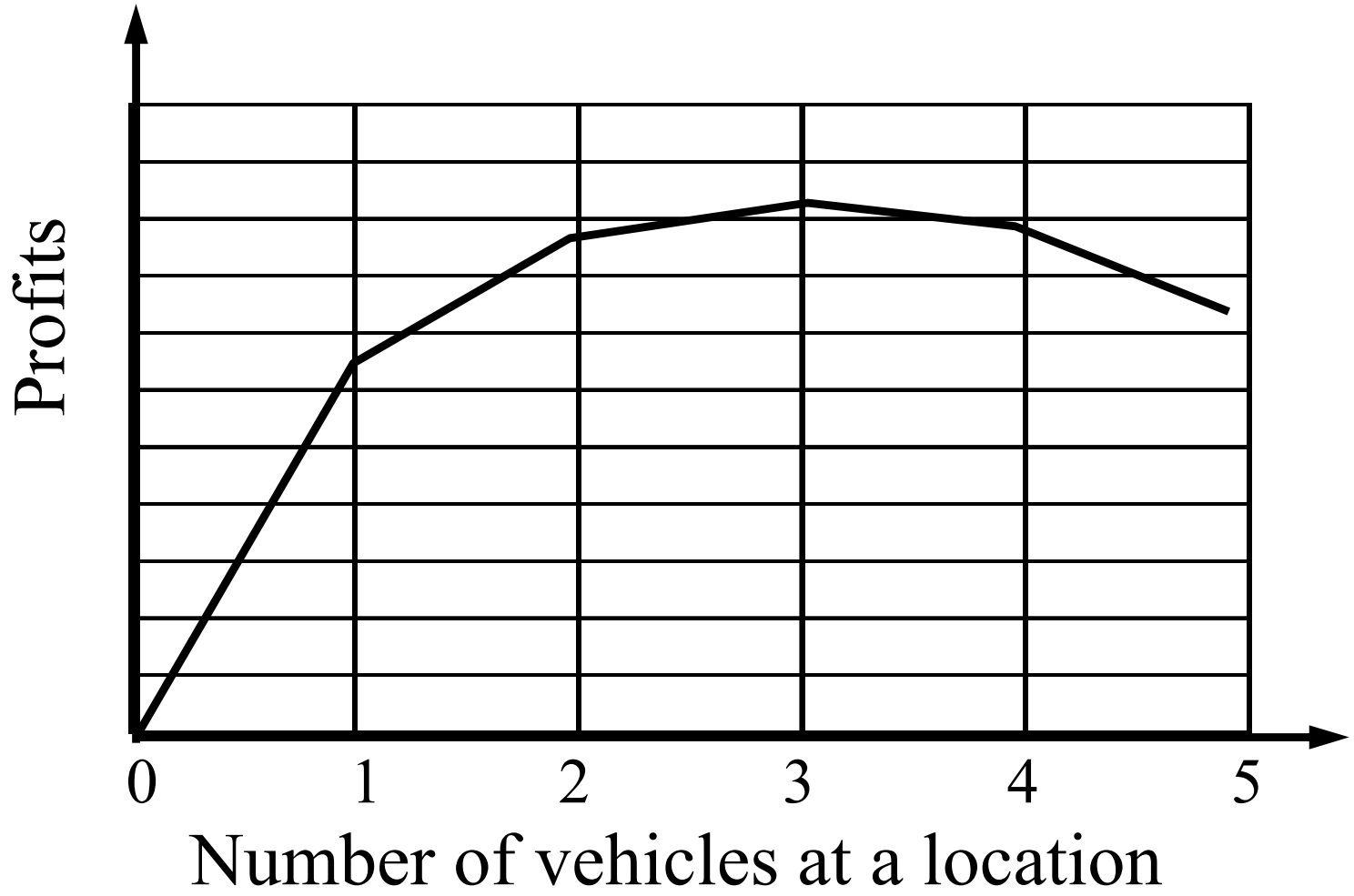
Optimization frameworks

- We obtain piecewise linear recourse functions for each regions.



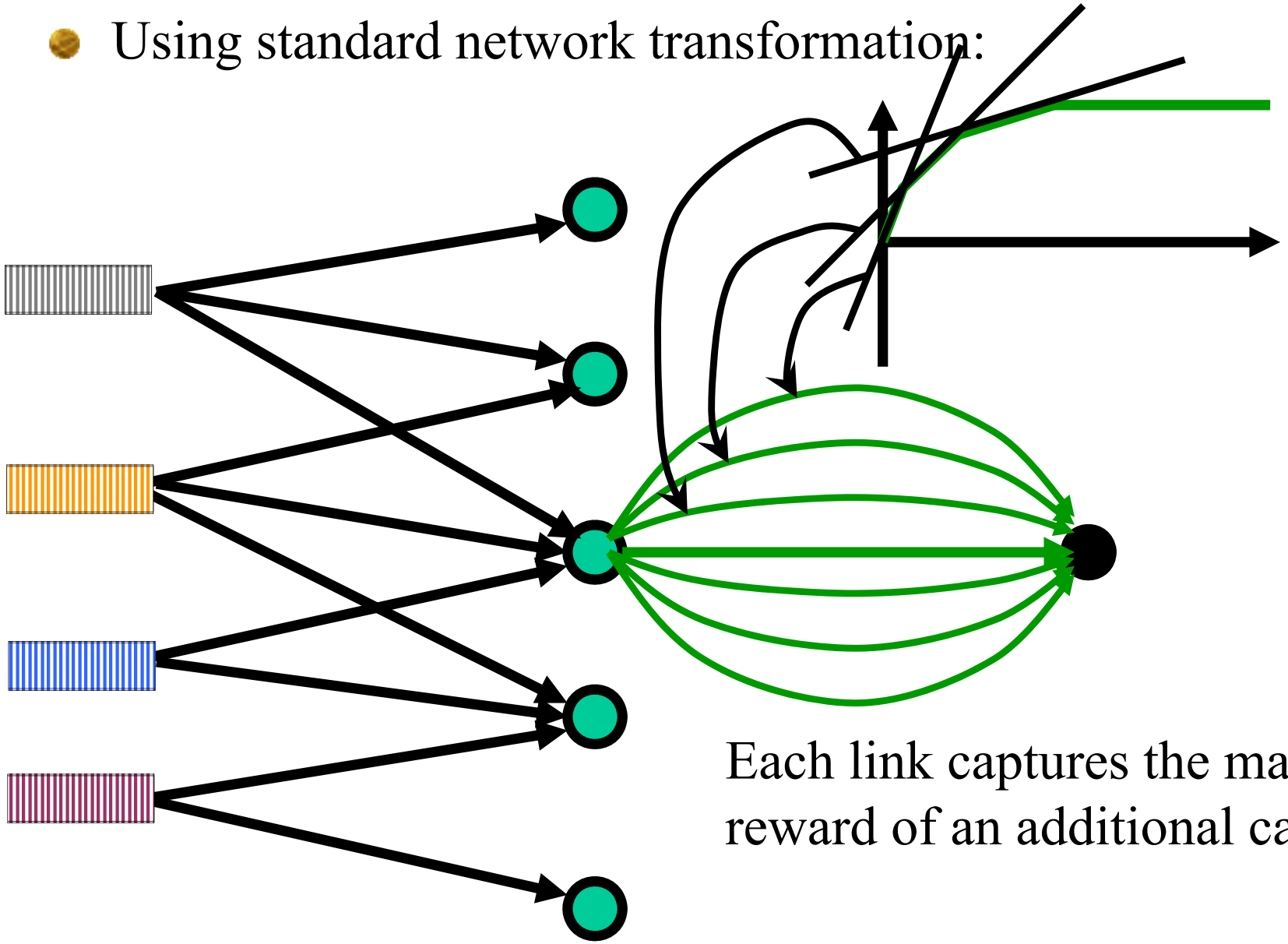
Optimization frameworks

- The function is piecewise linear on the integers.



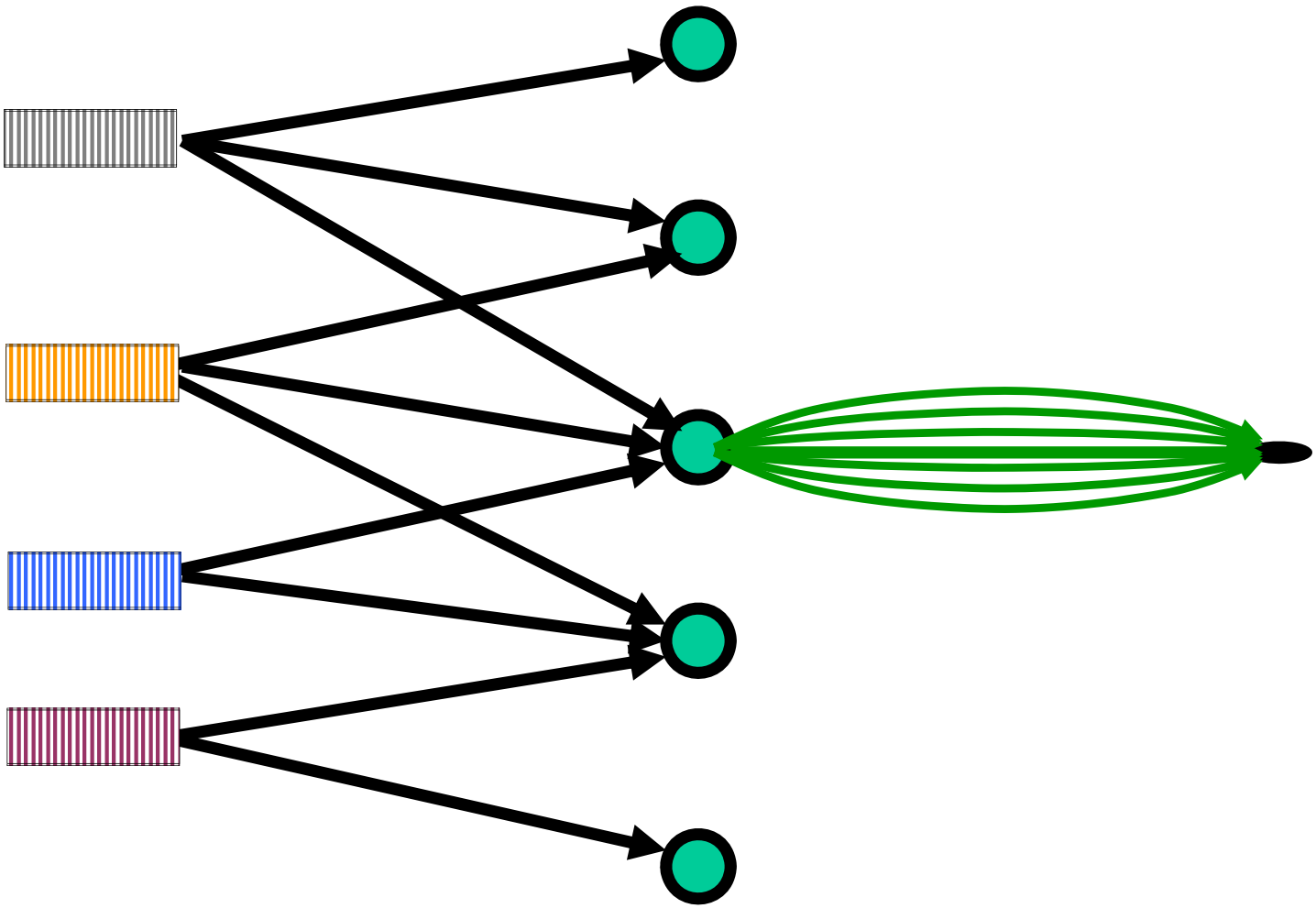
Optimization frameworks

- Using standard network transformation:

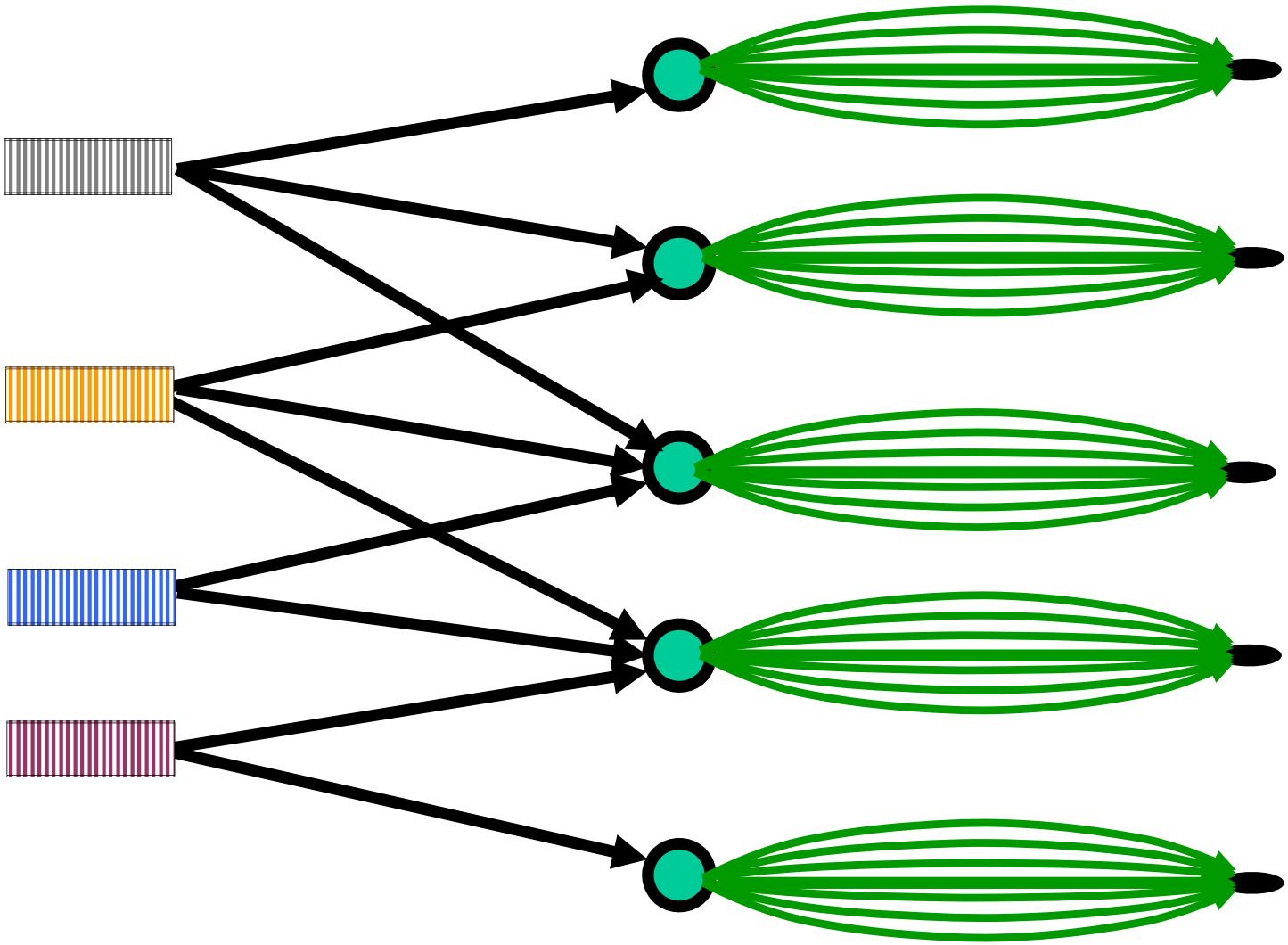


Each link captures the marginal reward of an additional car.

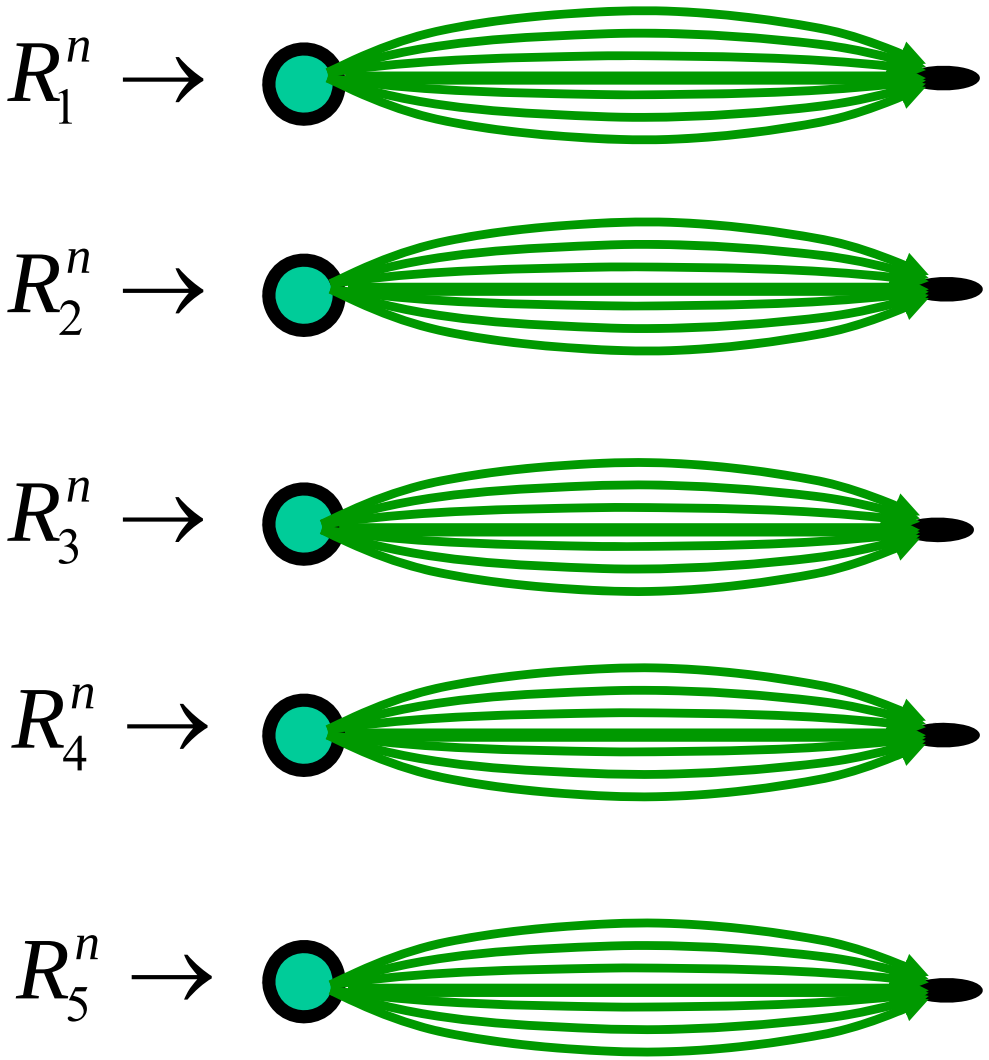
Two-stage problems



Two-stage problems



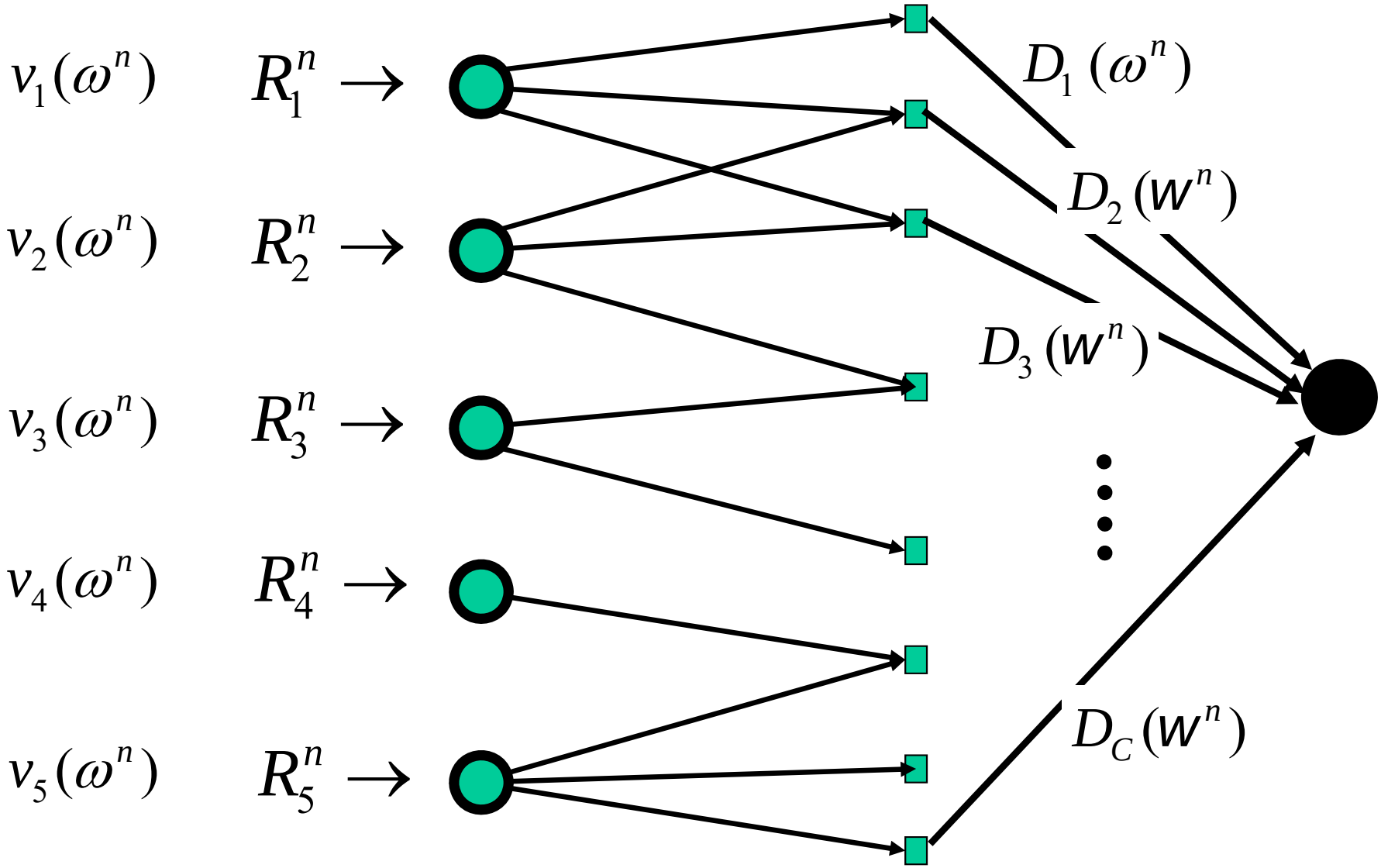
Two-stage problems



Two-stage problems

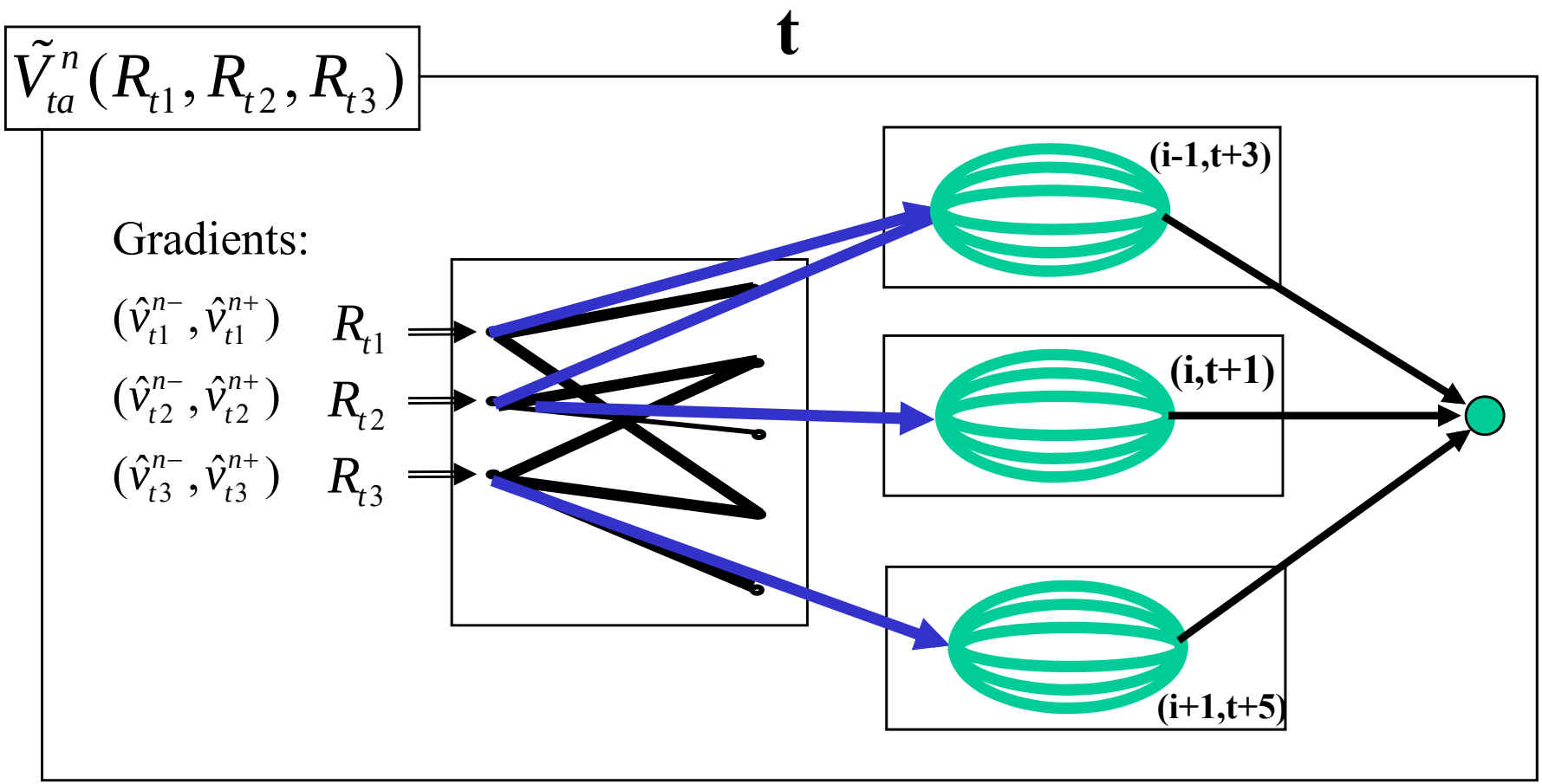
- We estimate the functions by sampling from our distributions.

Marginal value:



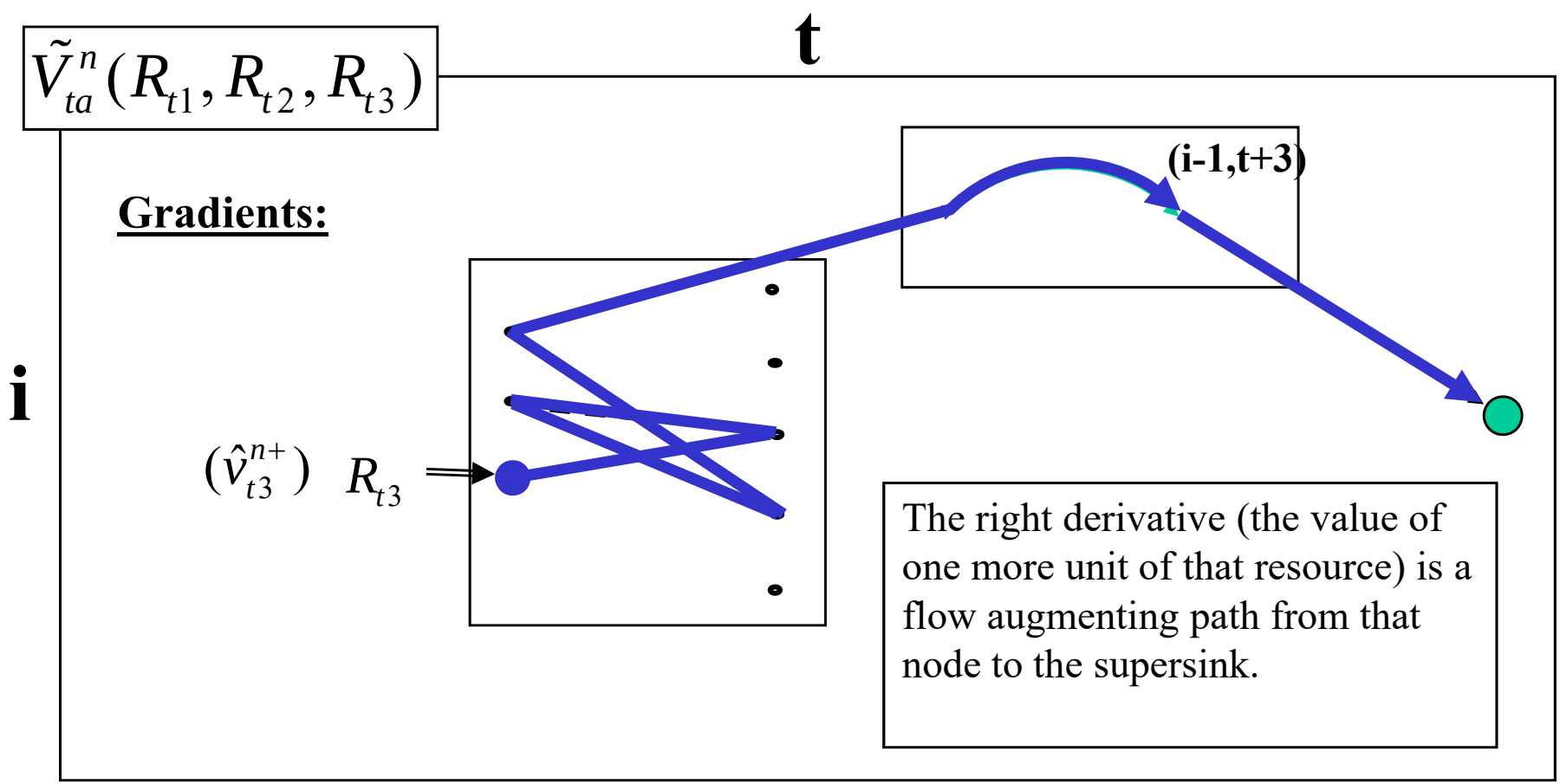
Two-stage problems

- The time t subproblem:



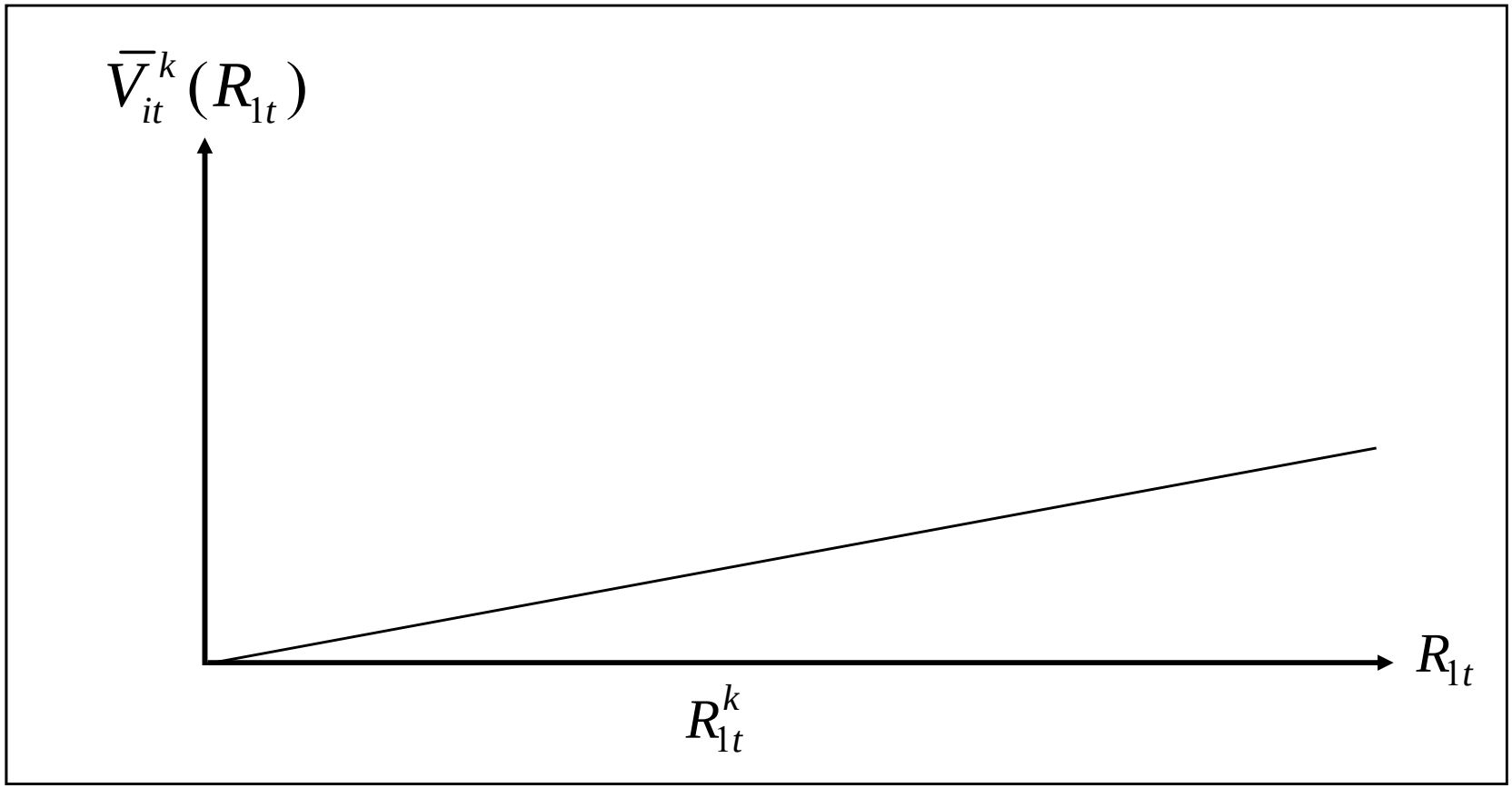
Two-stage problems

- Left and right gradients are found by solving flow augmenting path problems.



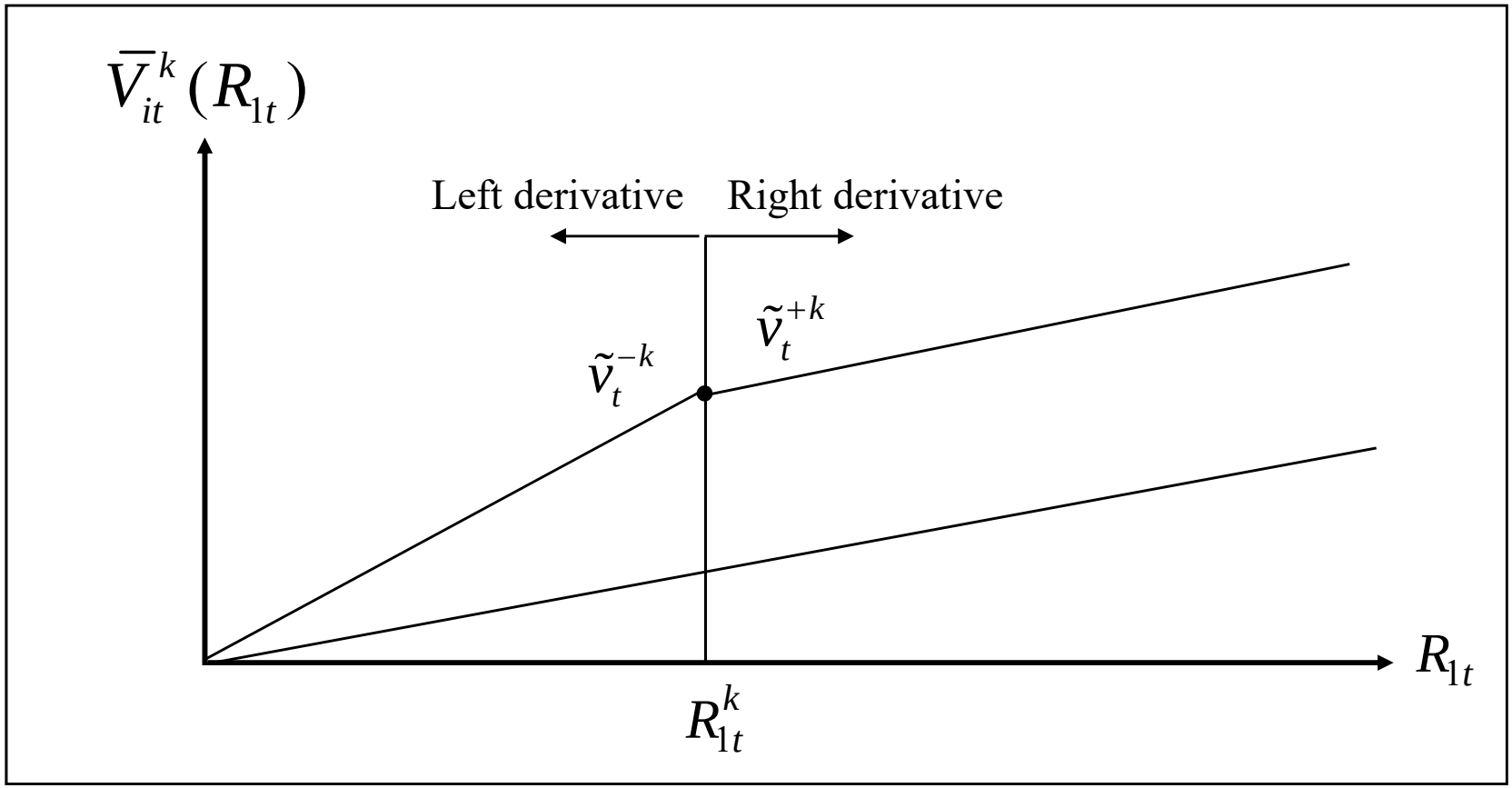
Two-stage problems

- Left and right derivatives are used to build up a nonlinear approximation of the subproblem.



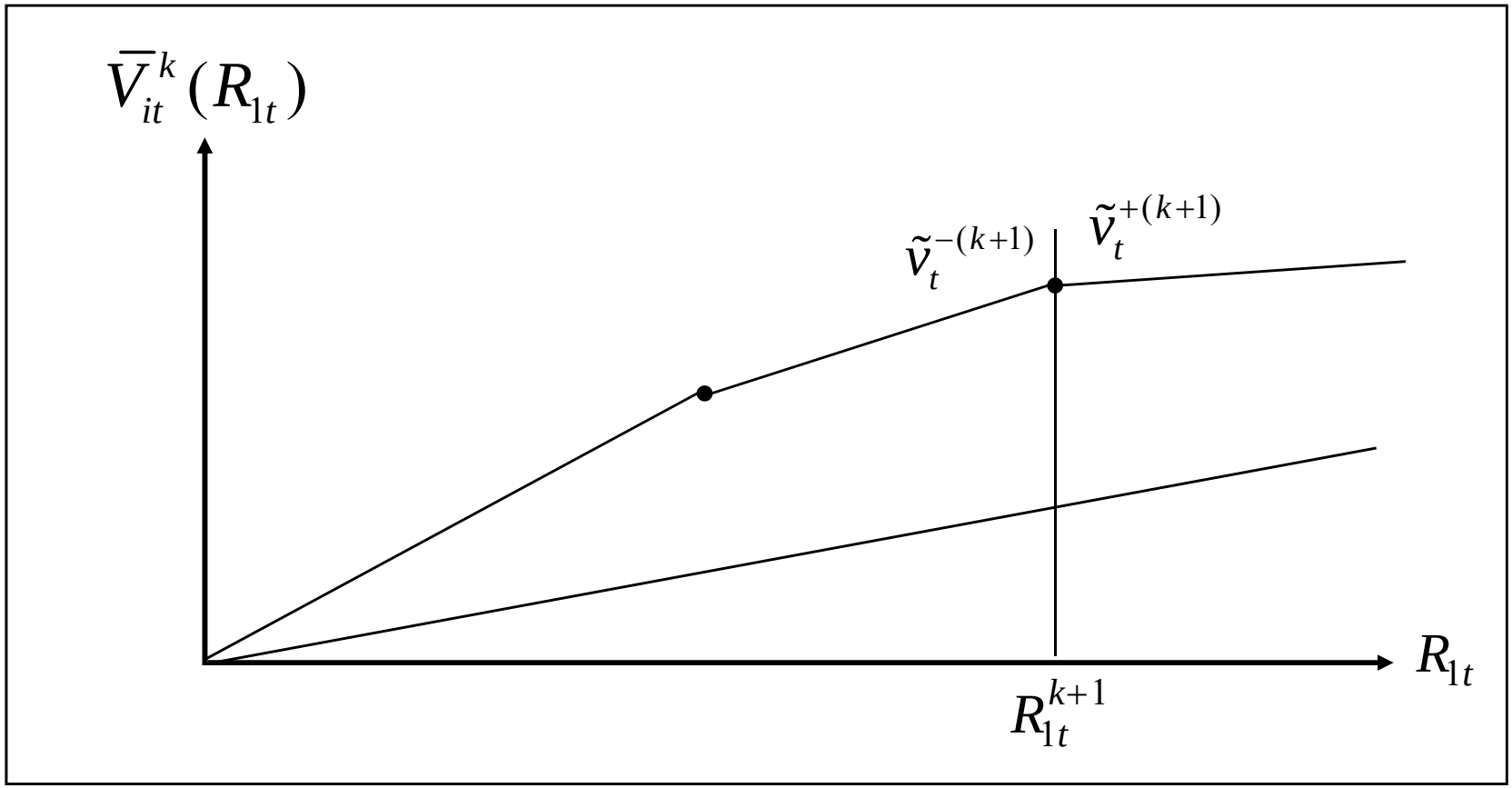
Two-stage problems

- Left and right derivatives are used to build up a nonlinear approximation of the subproblem.

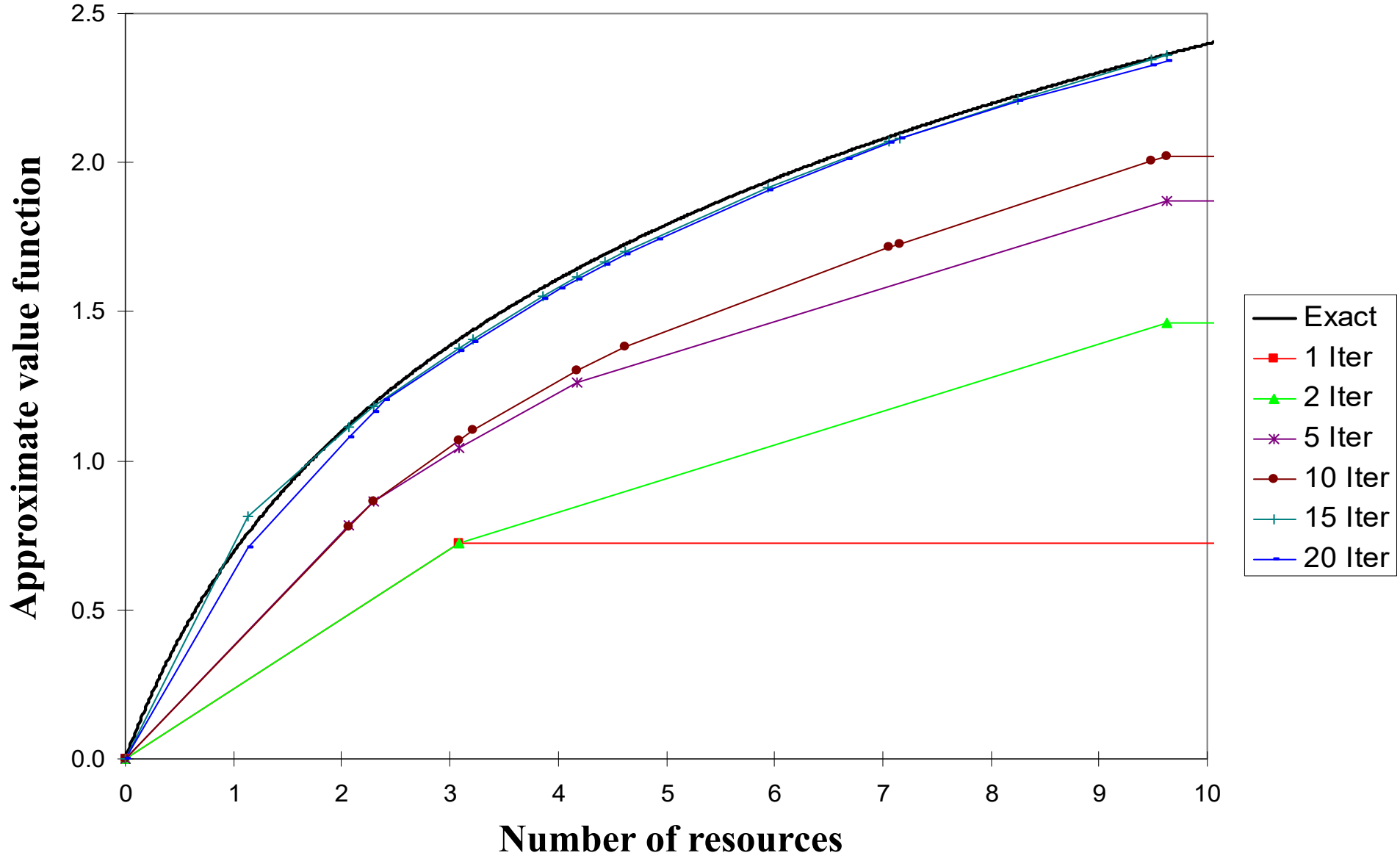


Two-stage problems

- Each iteration adds new segments, as well as refining old ones.



Two-stage problems



What we can prove

- SPAR algorithm
  - » Powell, Ruszczynski and Topaloglu, *Mathematics of Operations Research (2004)*
- Leveling algorithm
  - » Topaloglu and Powell, *Operations Research Letters, 2003.*
- But so far, our fastest convergence is with the original version, called the CAVE algorithm, for which convergence has not been proven.
  - » Godfrey and Powell, *Management Science, 2001.*

Approximate dynamic programming

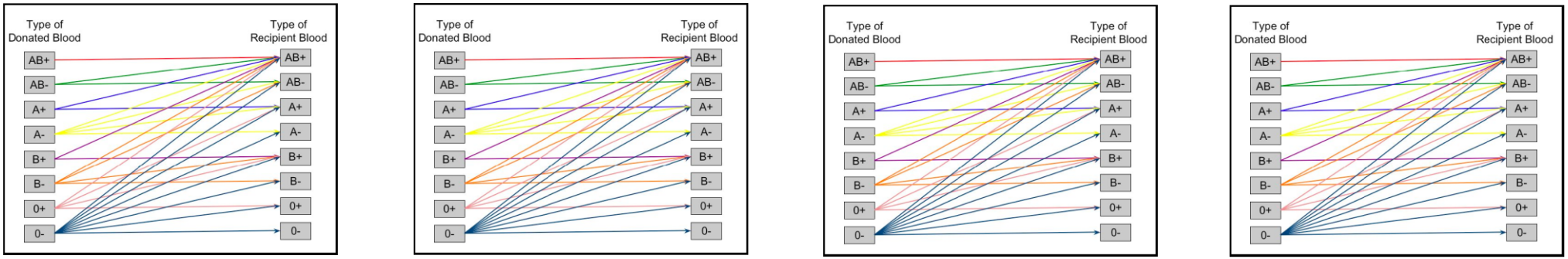
Blood management problem

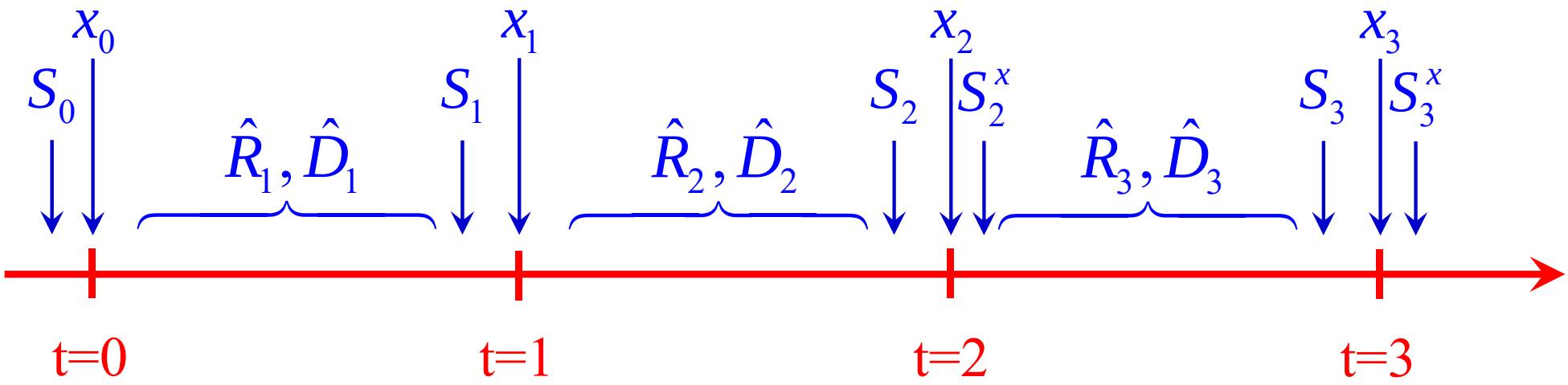
Management multiple resources

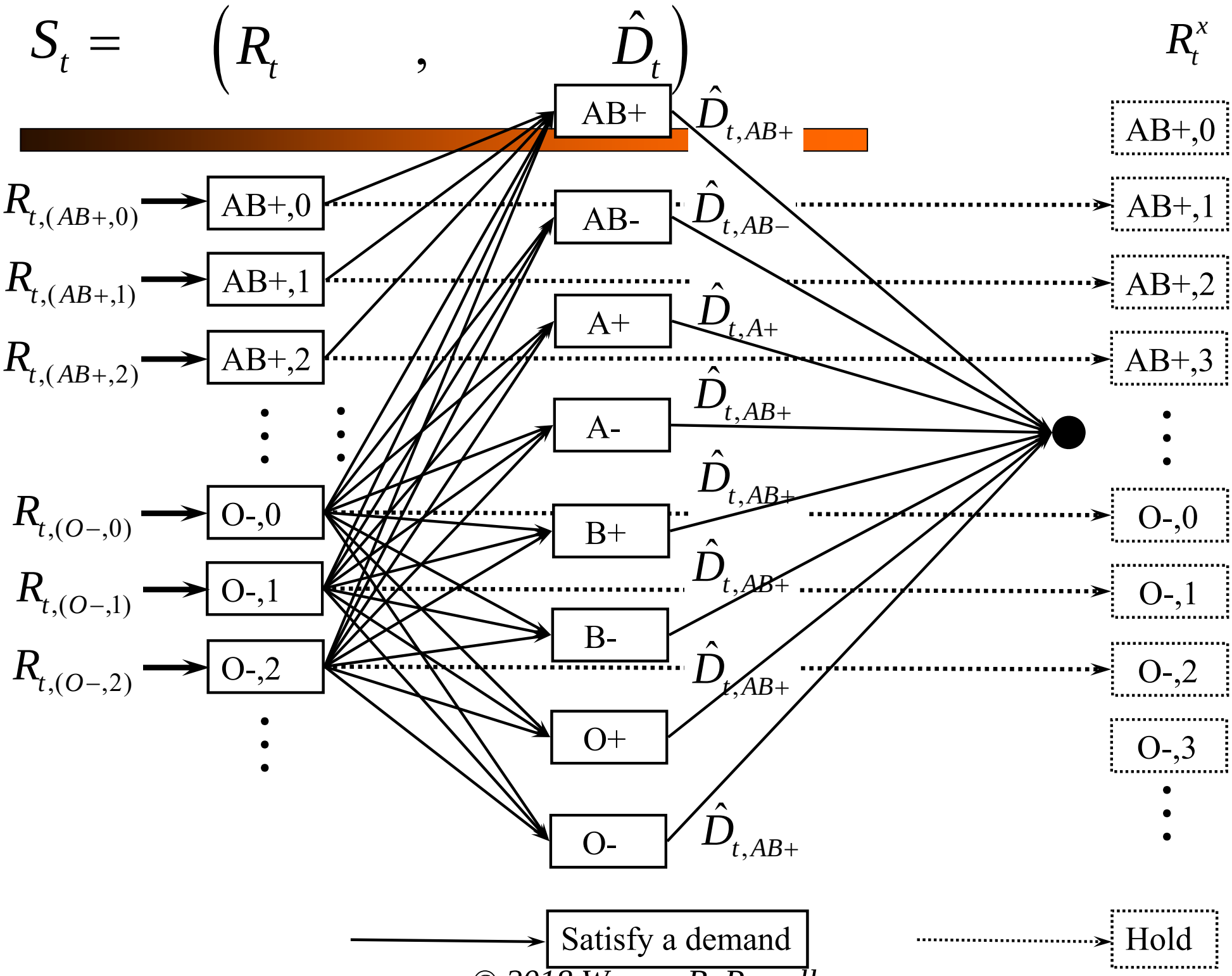
Blood management

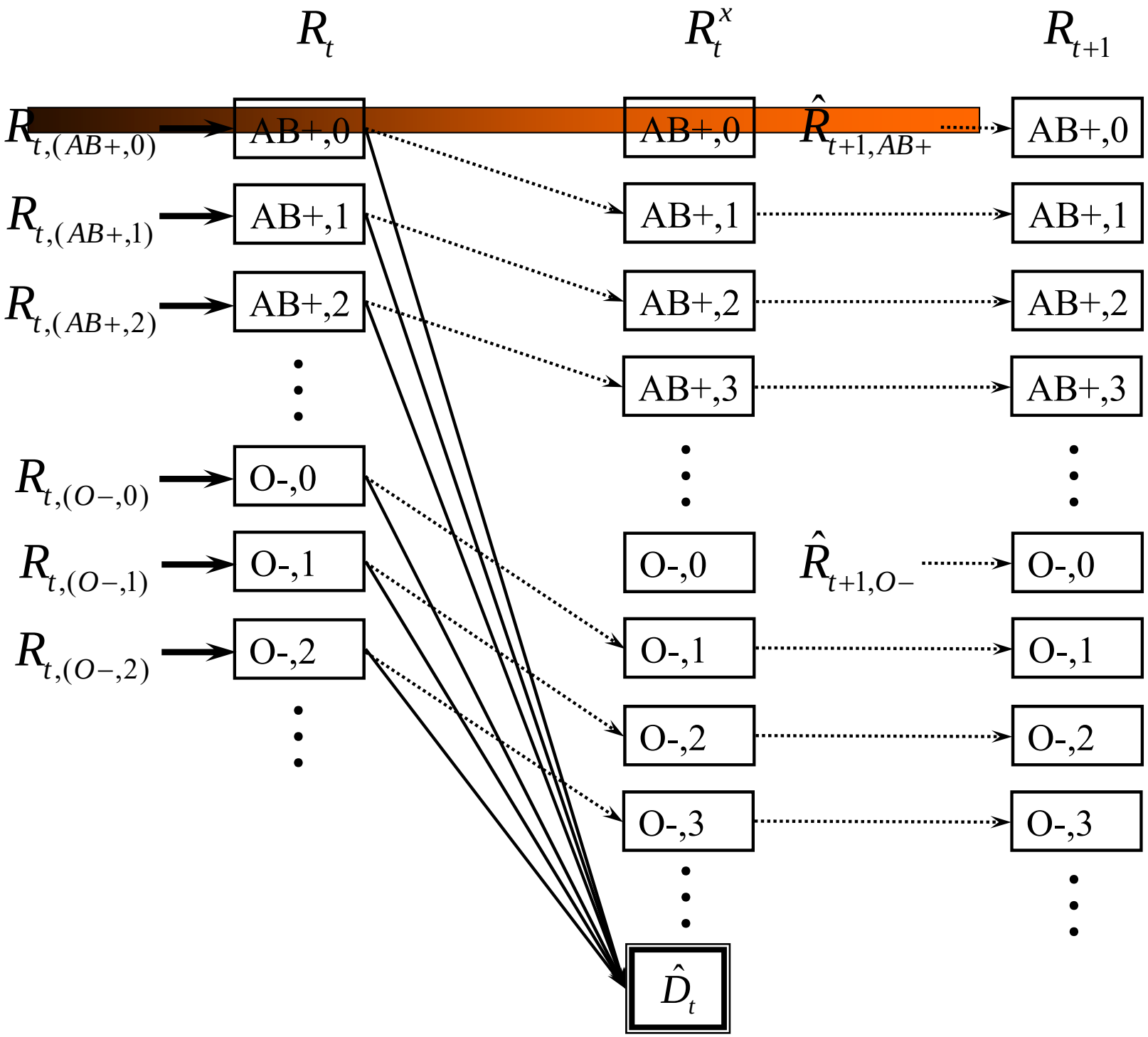
- Managing blood inventories over time

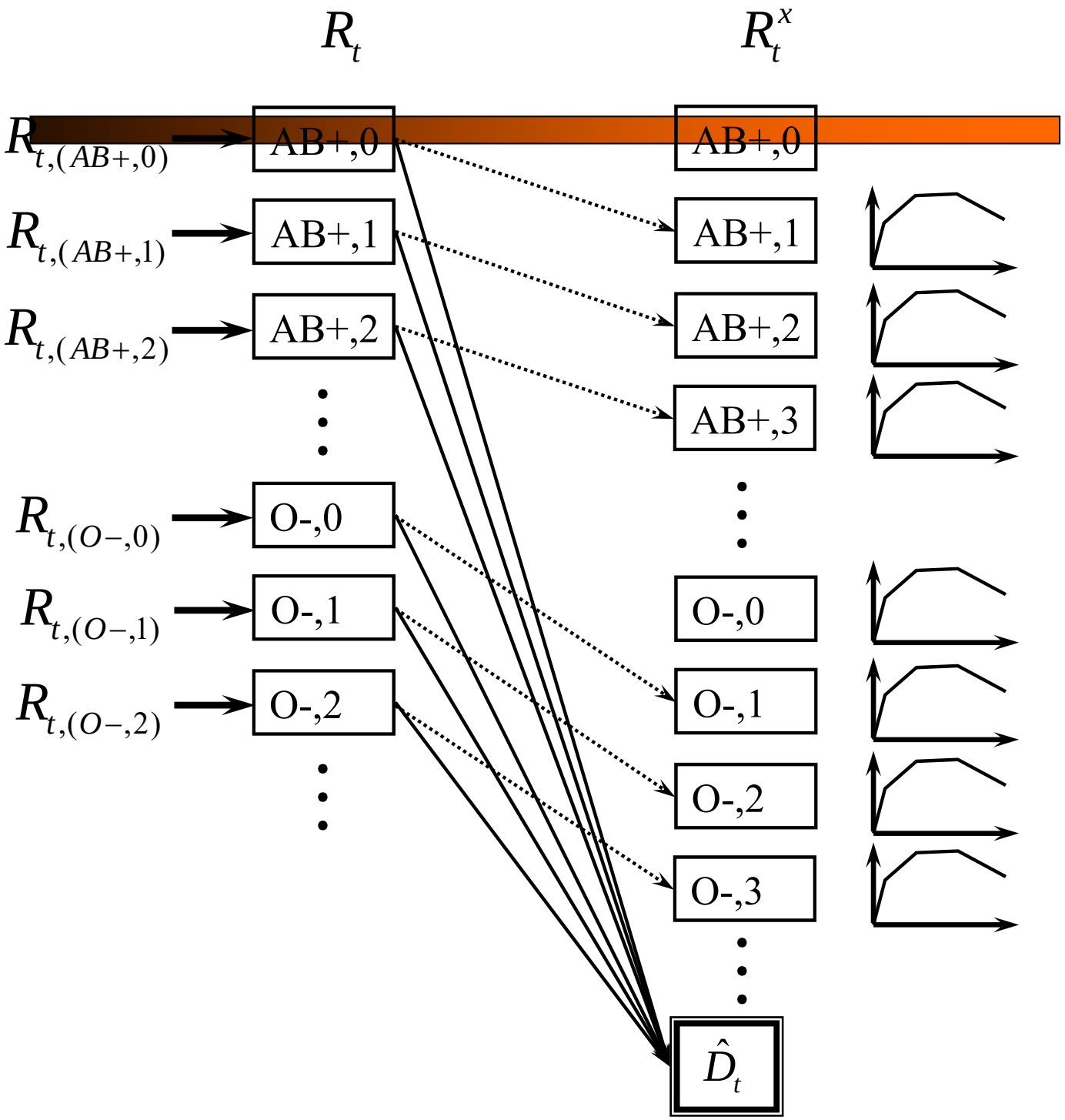
Week 0 Week 1 Week 2 Week 3

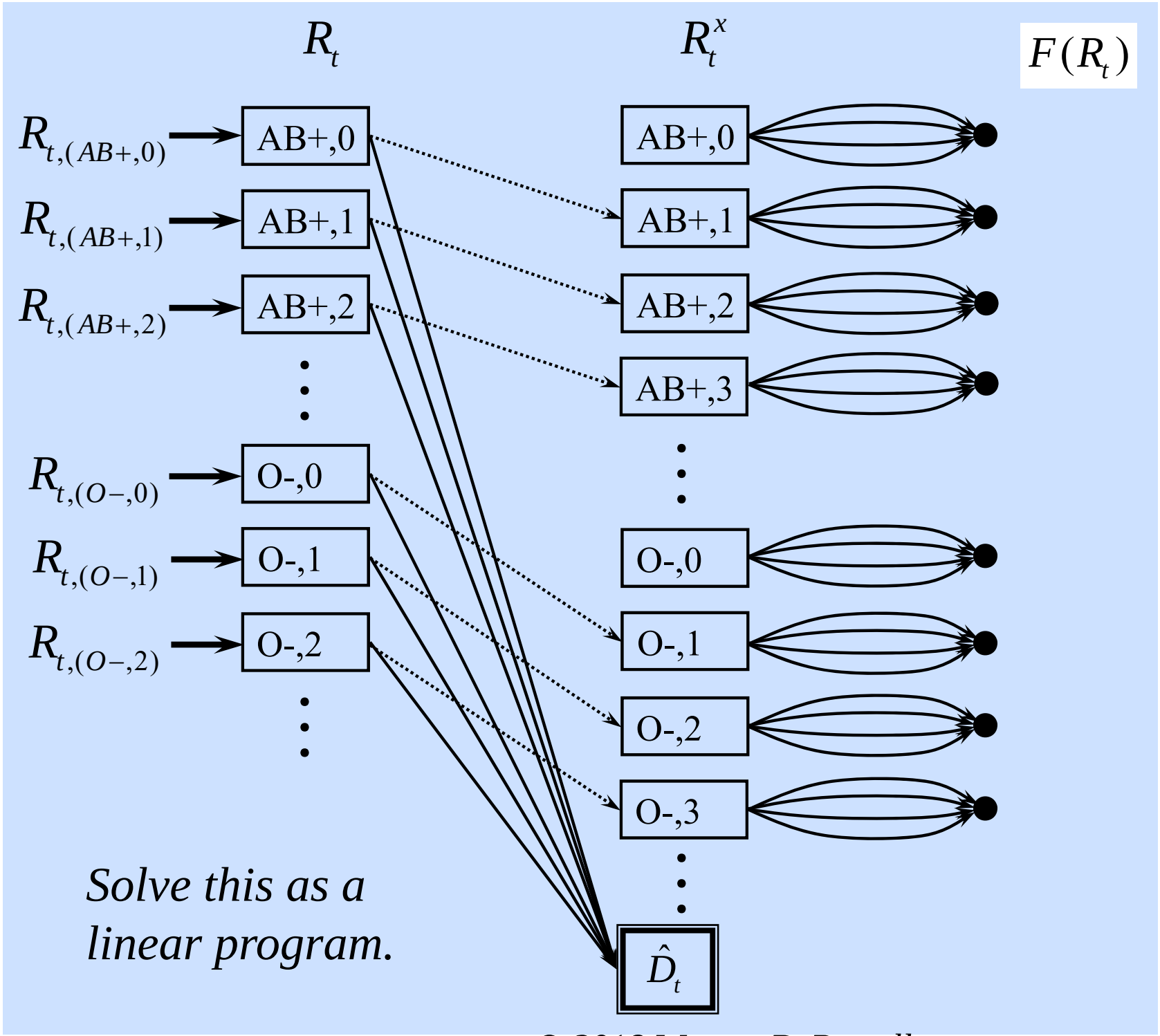


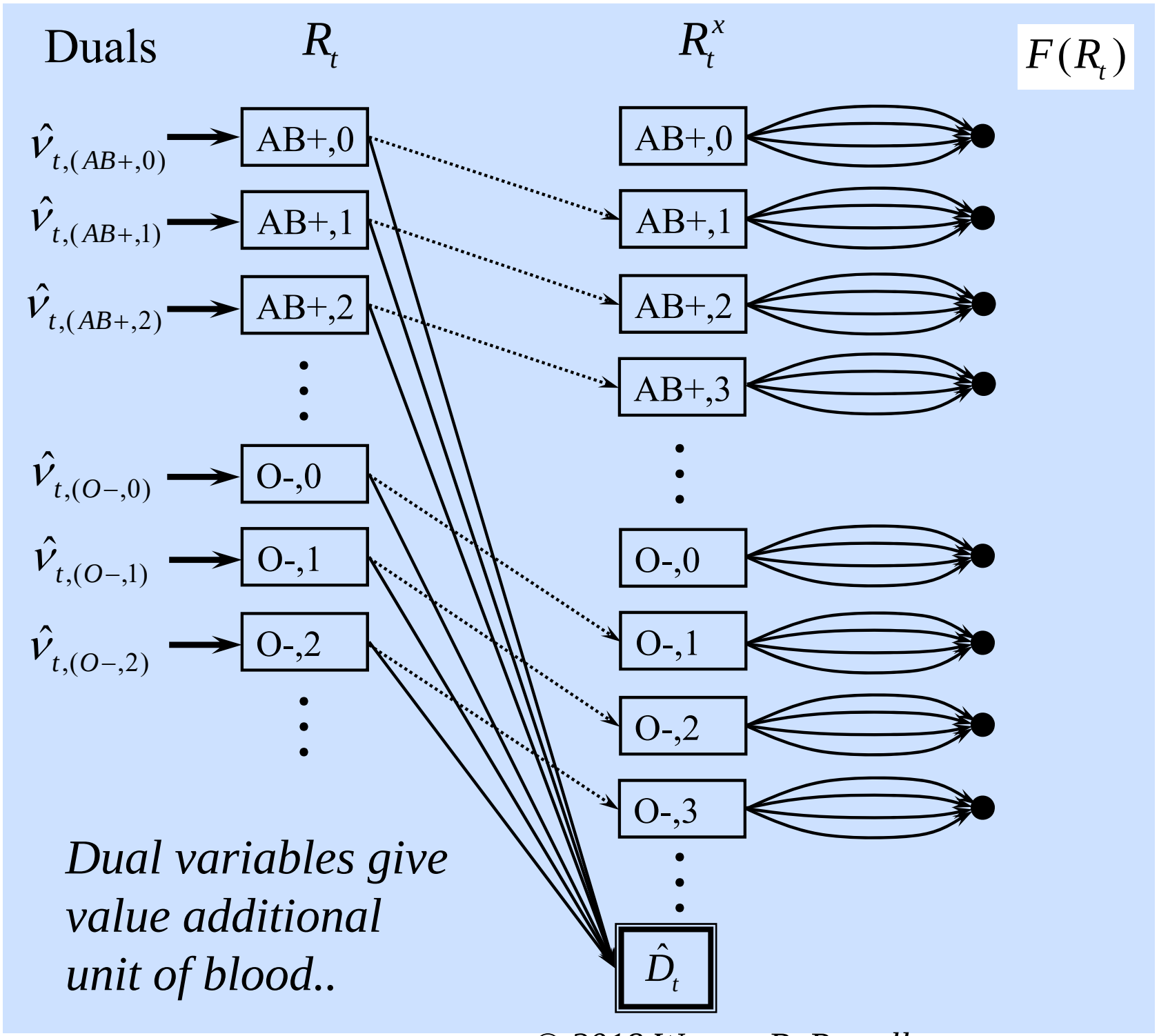






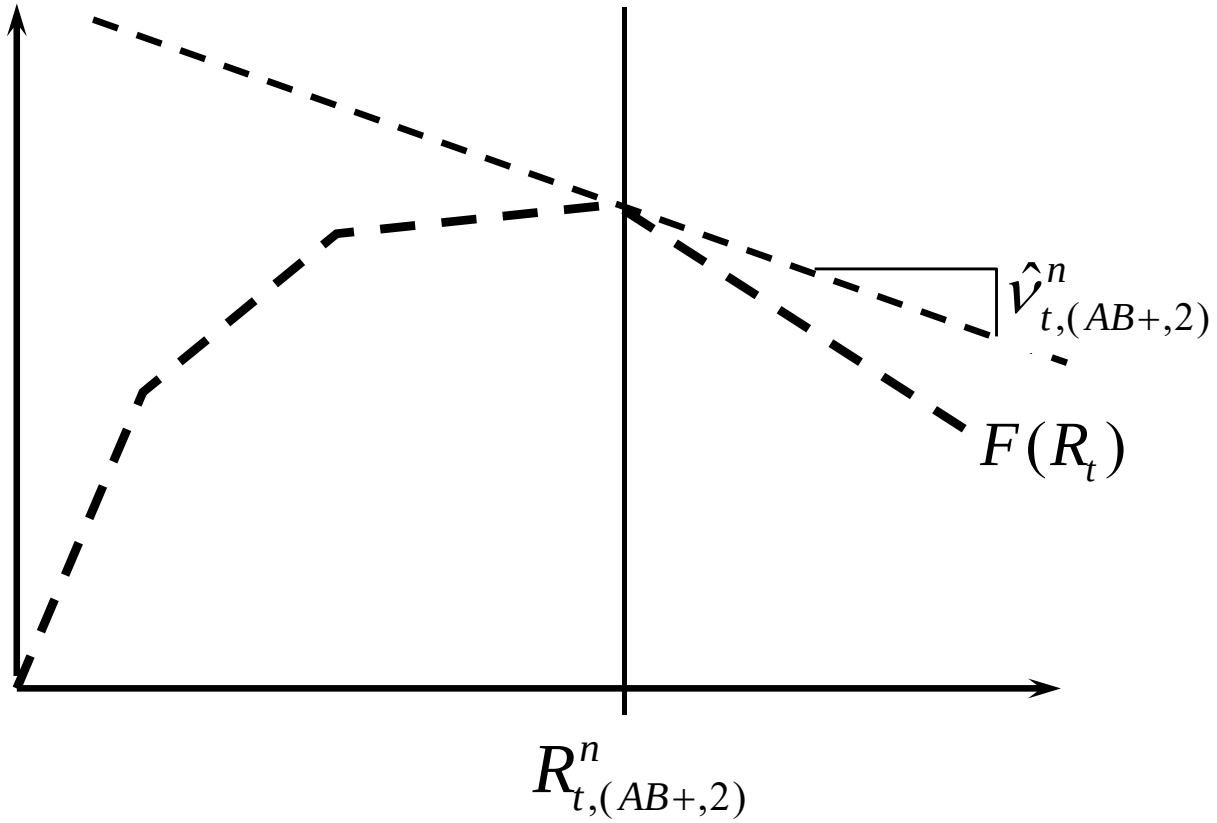






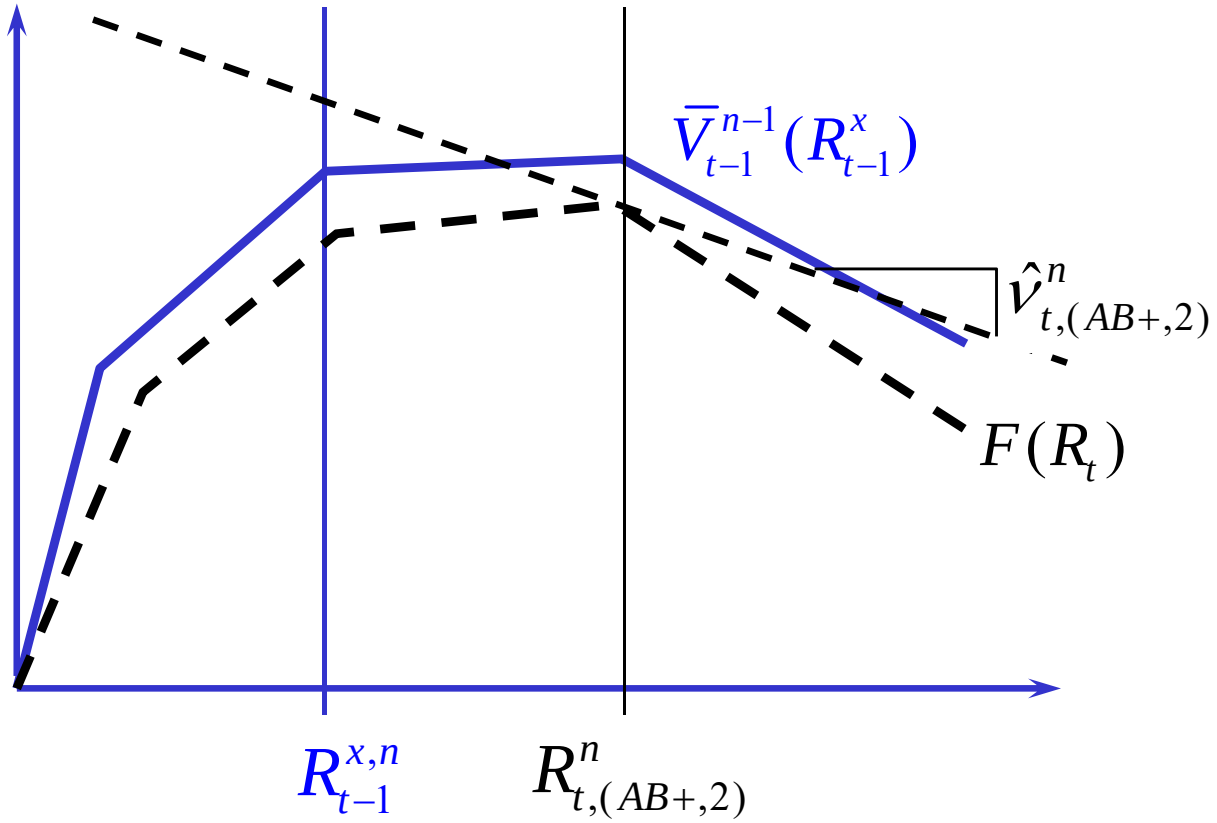
Updating the value function approximation

- Estimate the gradient at R_t^n



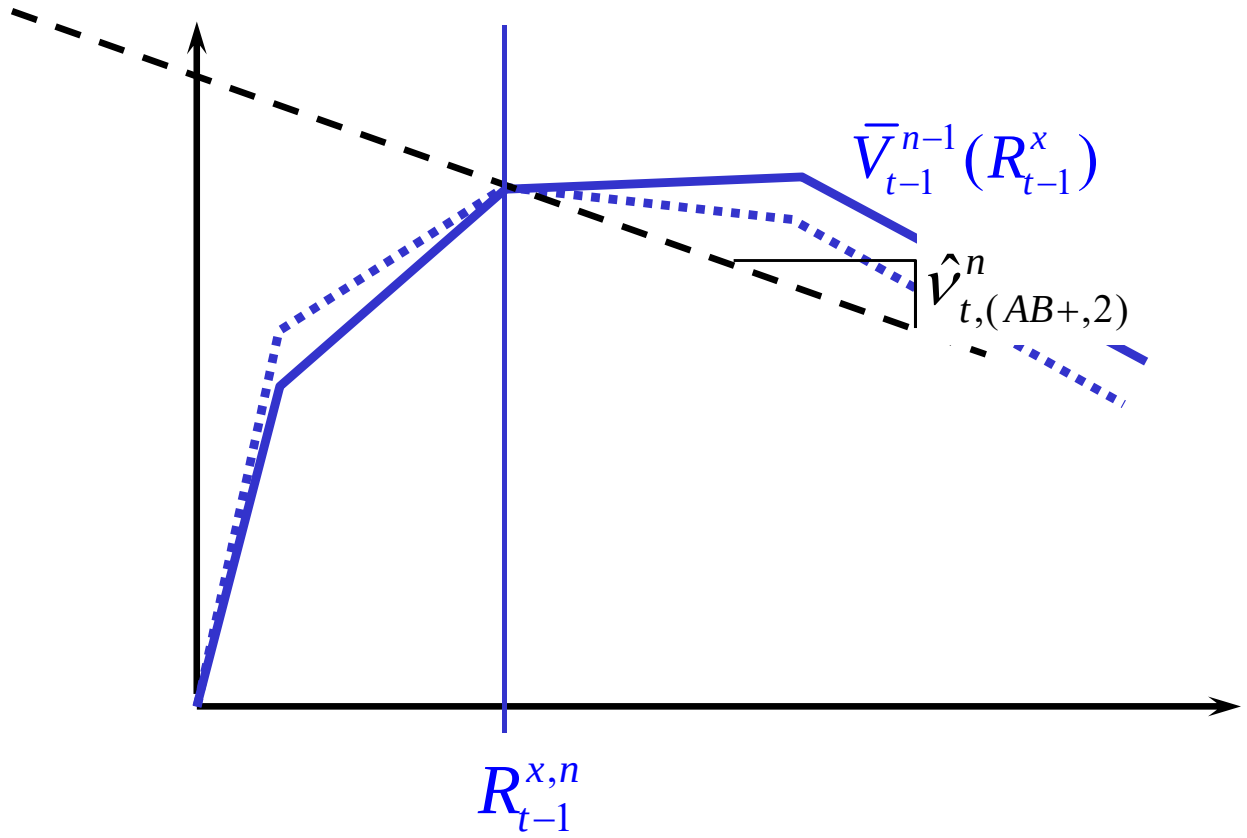
Updating the value function approximation

- Update the value function at $R_{t-1}^{x,n}$



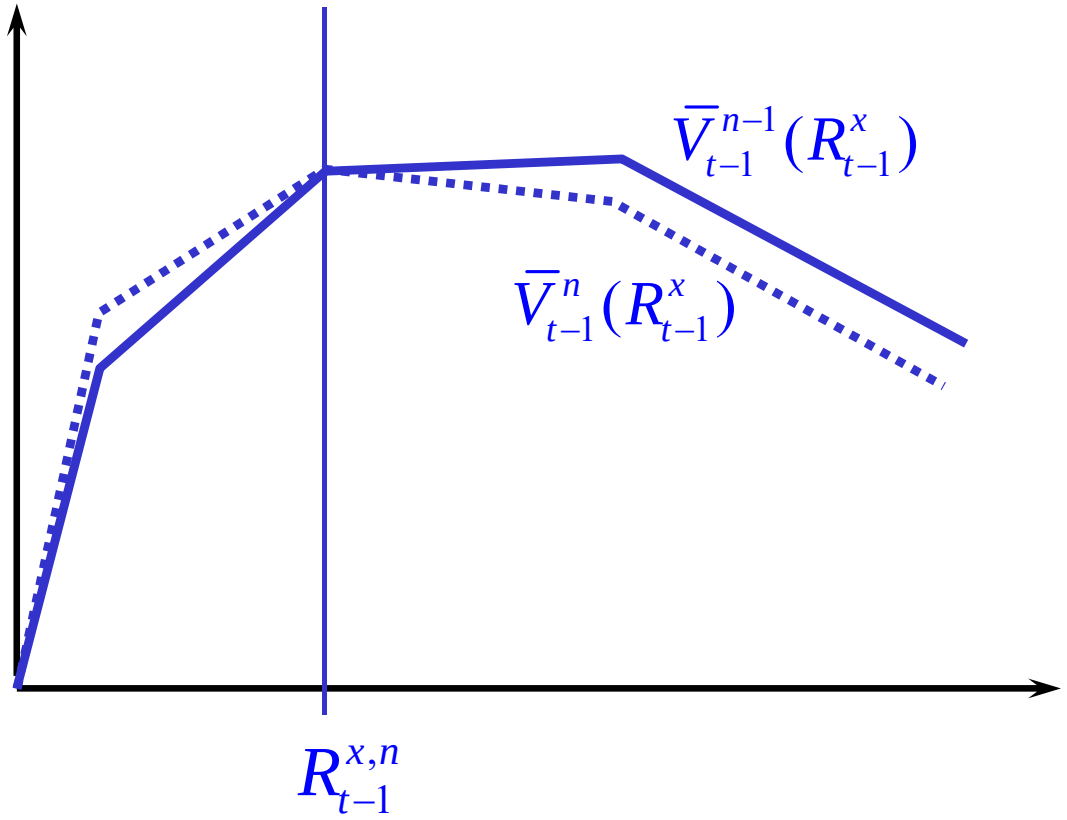
Updating the value function approximation

- Update the value function at $R_{t-1}^{x,n}$



Updating the value function approximation

- Update the value function at $R_{t-1}^{x,n}$

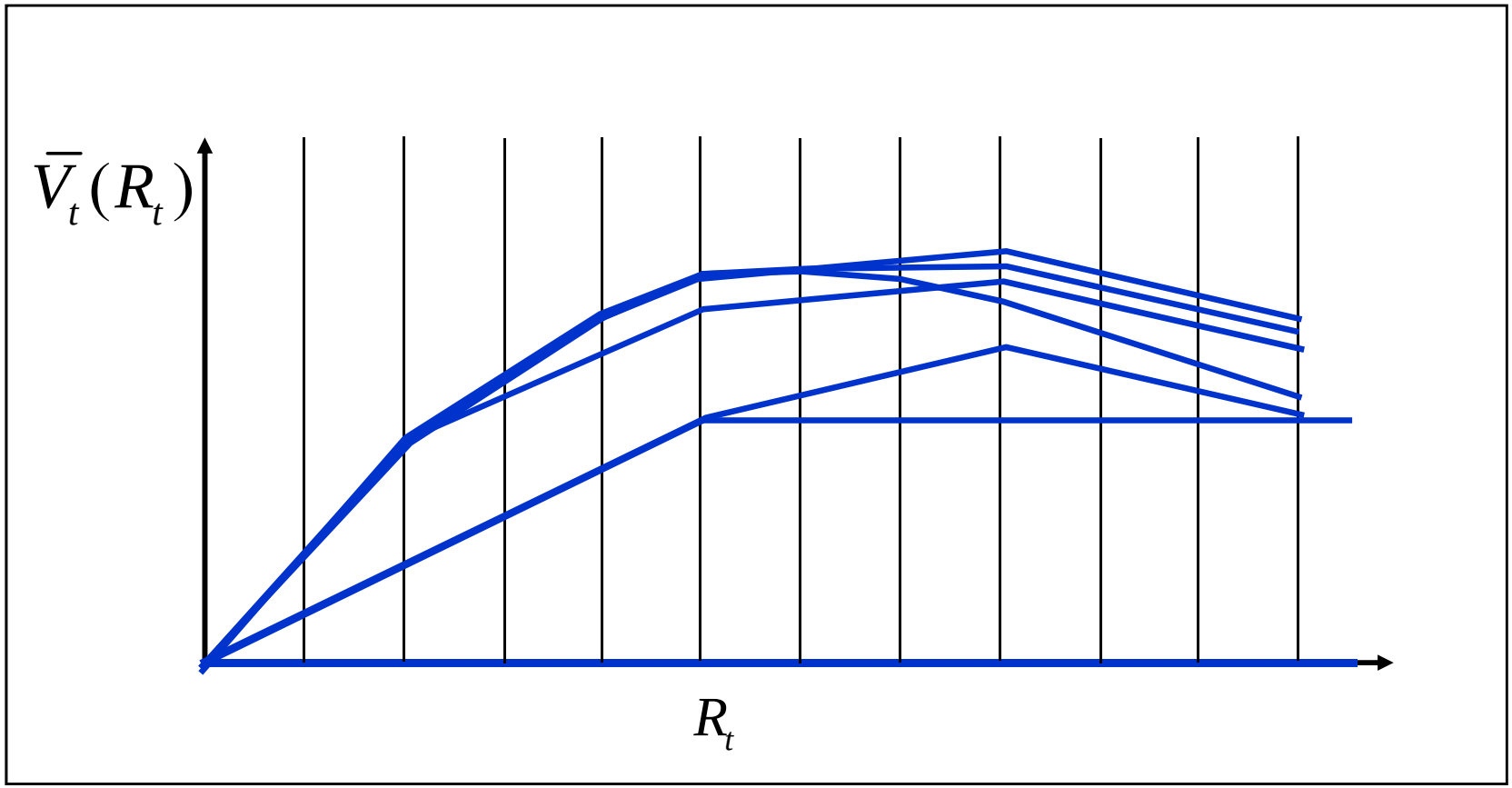


Updating the value function approximation

- Notes:
  - » We get a marginal value for *each* supply node.
  - » These are provided automatically by linear programming solvers.
  - » Be careful when the supply at the node = 0. While we would like to have the marginal value of one more, if we use the marginal values produced by the linear programming code, it might be the slope to the left (where the "supply" would be negative).

Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation



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:

$$\max_x \left(C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right)$$

to obtain x_t^n and dual variables $\left(\hat{v}_{ti}^n\right)$.

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 value iteration

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

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

$$\max_x \left(C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right)$$

to obtain x_t^n and dual variables $\left(\hat{v}_{ti}^n\right)$.

Deterministic optimization

Approximate value iteration

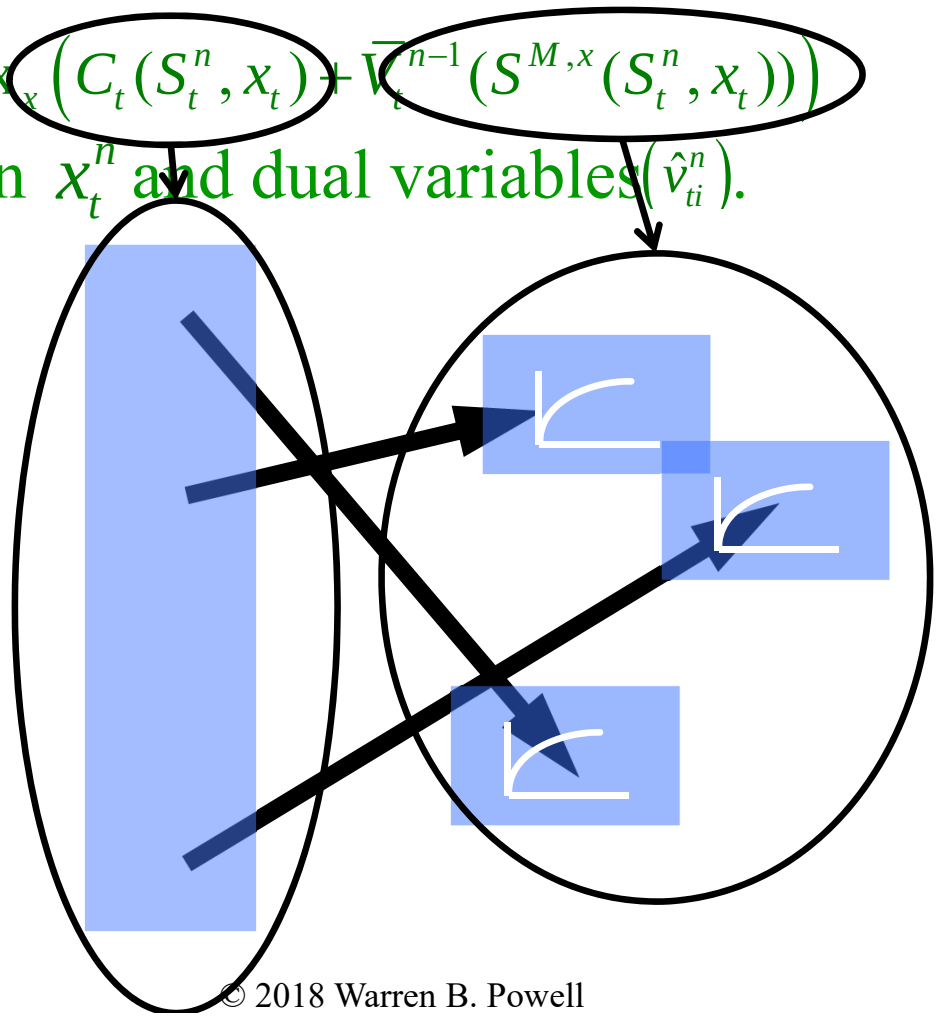
Step 1: Start with a pre-decision state S_t^n

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

$$\max_x \left(C_t(S_t^n, x_t) + \overline{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right)$$

to obtain x_t^n and dual variables $\left(\hat{v}_{ti}^n\right)$.

Deterministic optimization



Approximate value iteration

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

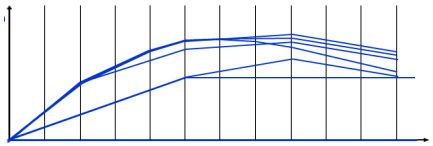
Step 2: Solve the deterministic optimization using an approximate value function:

$$\max_x \left(C_t(S_t^n, x_t) + \bar{V}_t^{n-1}(S^{M,x}(S_t^n, x_t)) \right)$$

to obtain x_t^n and dual variables $\left(\hat{v}_{ti}^n\right)$.

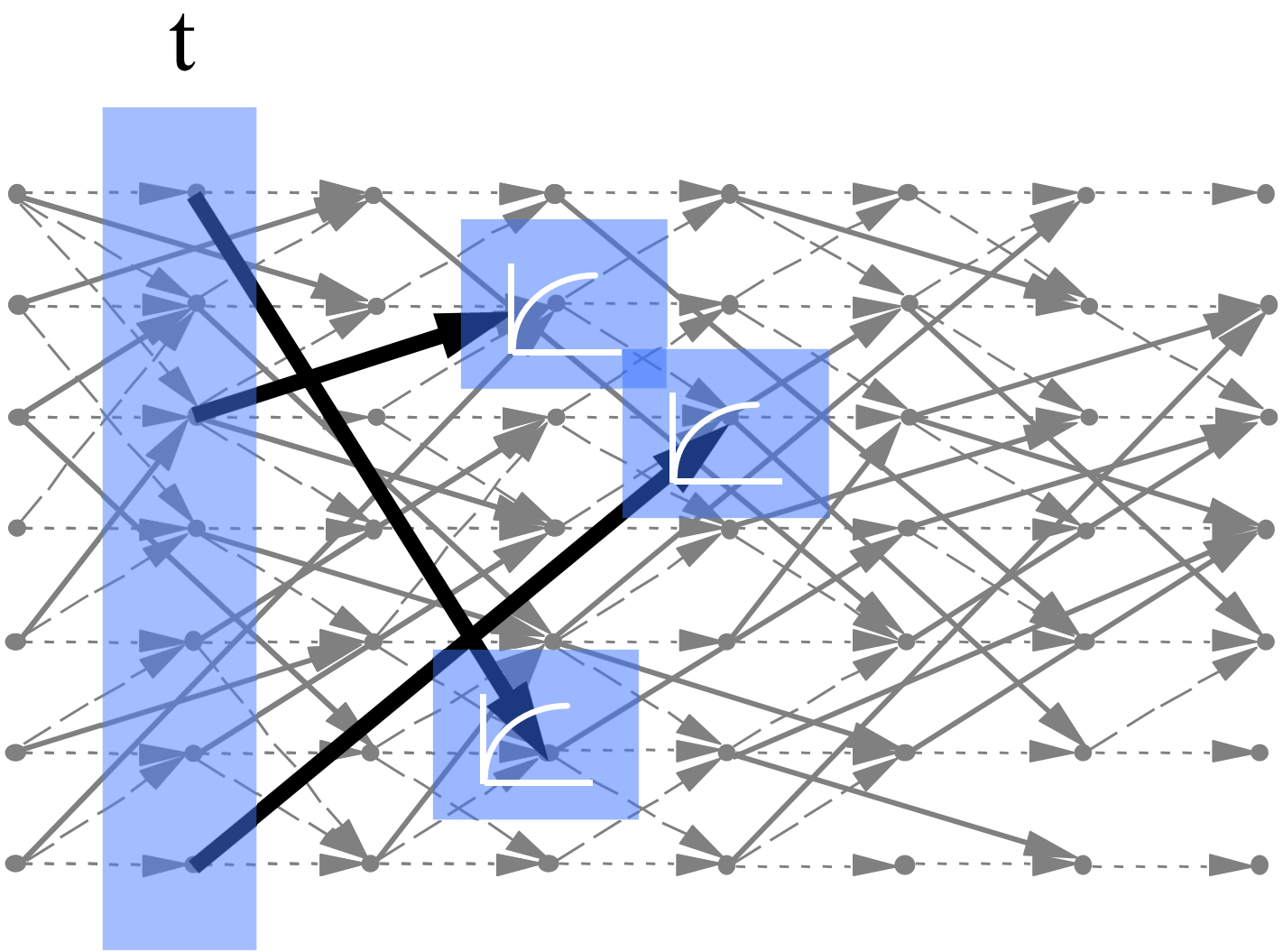
Deterministic optimization

Step 3: Update the value function approximation

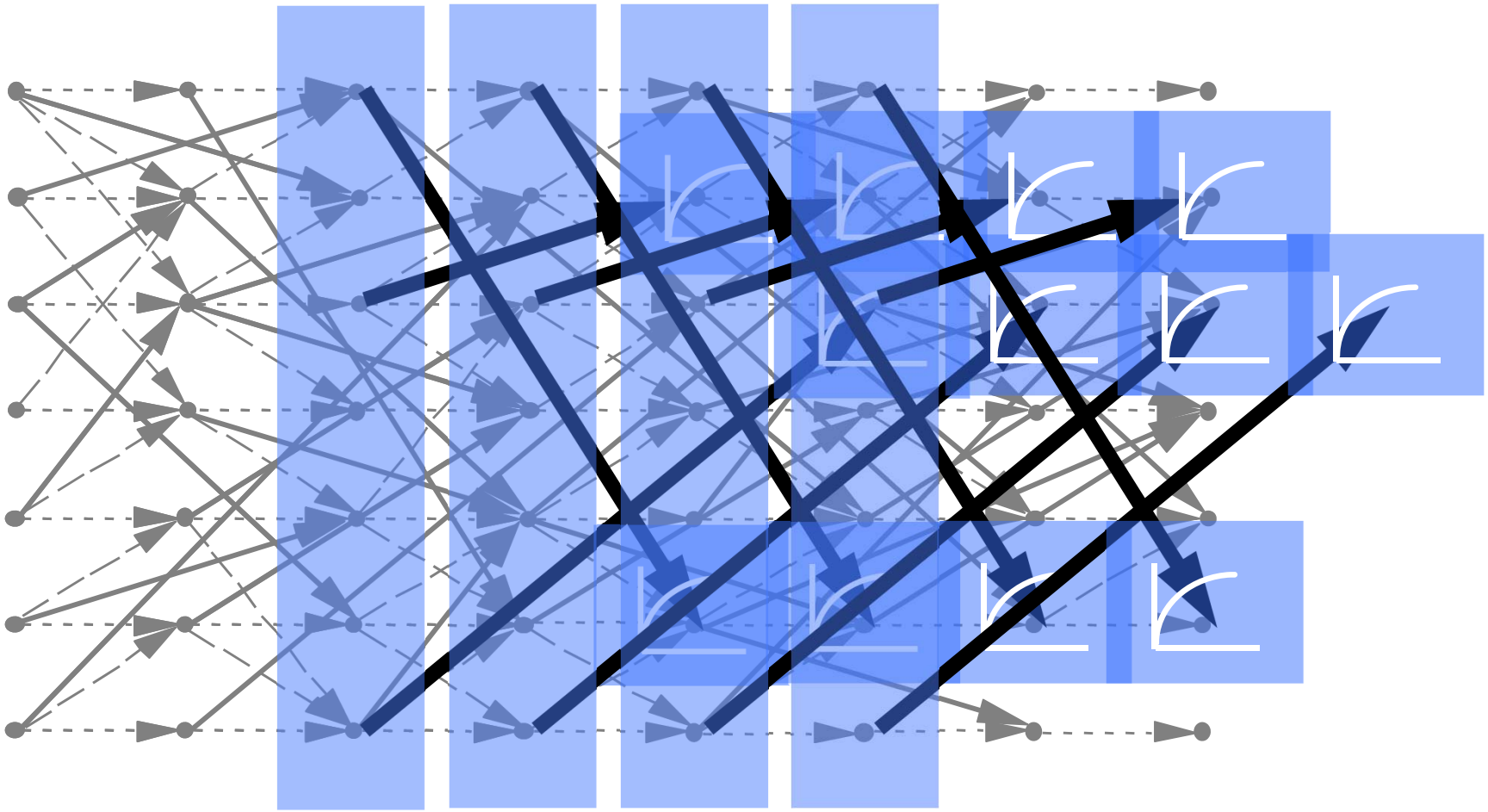


Recursive statistics

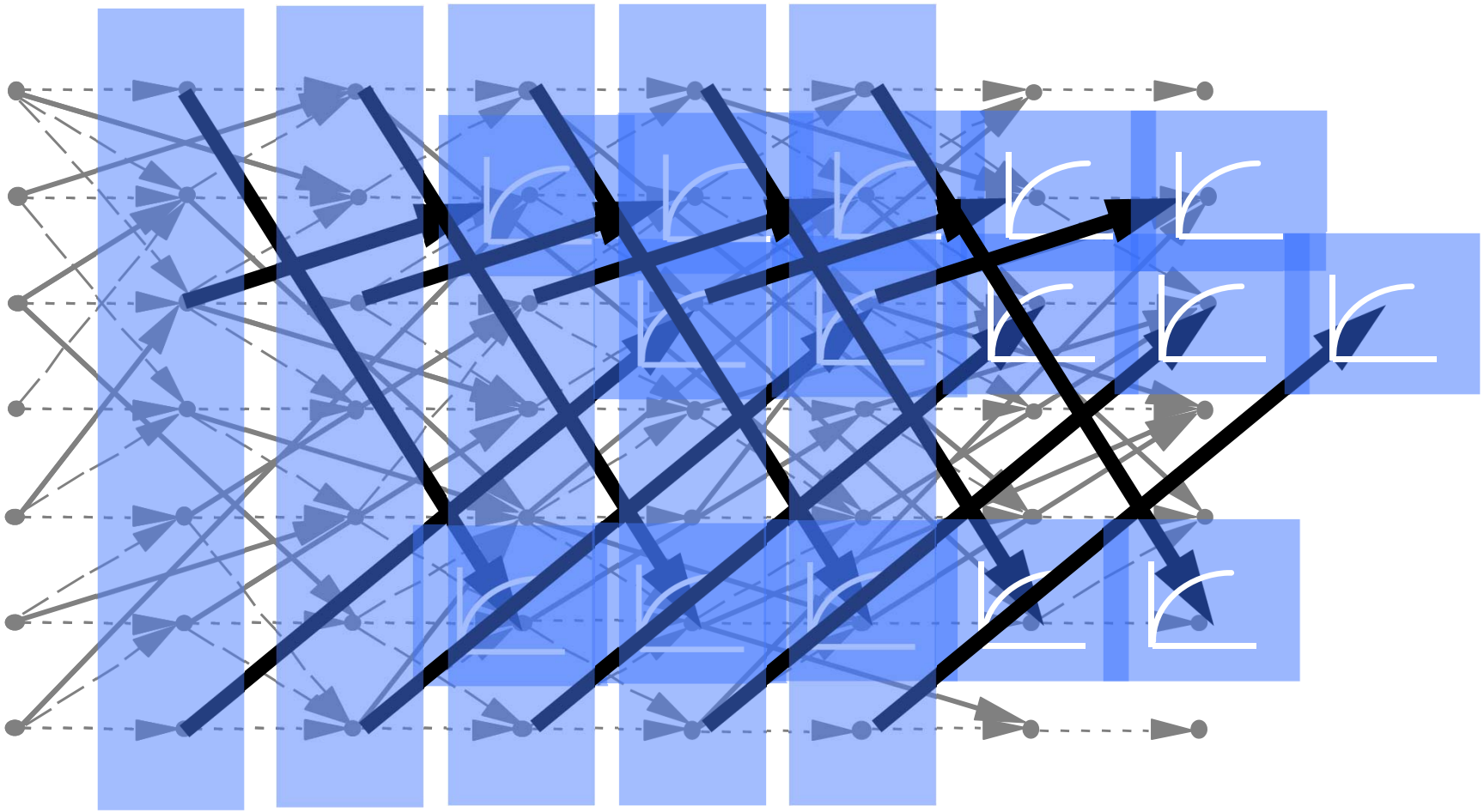
Iterative learning



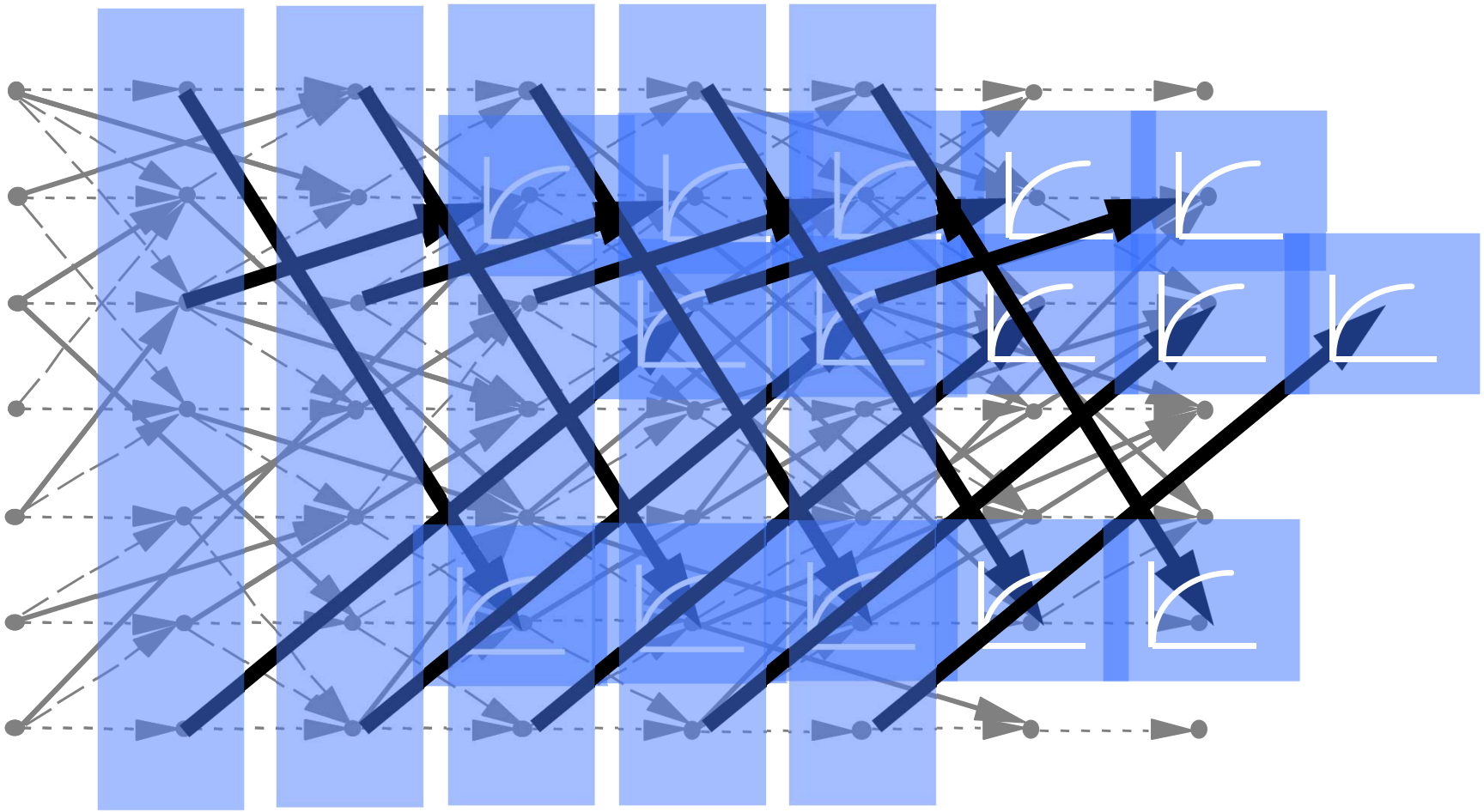
Iterative learning



Iterative learning



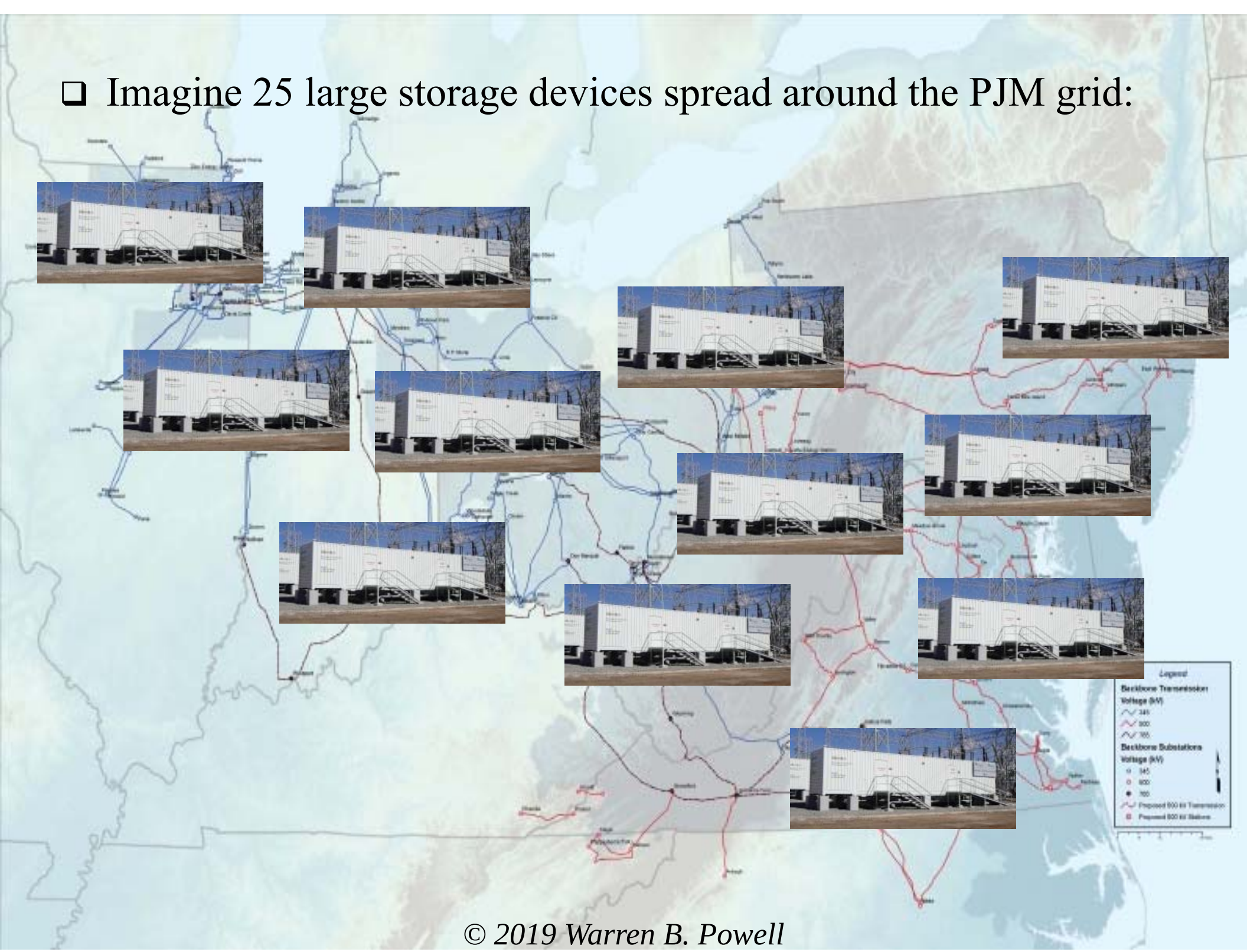
Iterative learning



Approximate dynamic programming

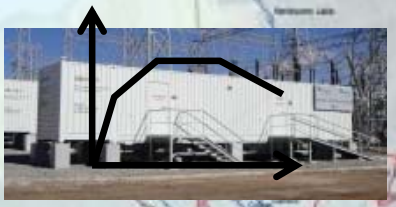
Grid level storage

- Imagine 25 large storage devices spread around the PJM grid:



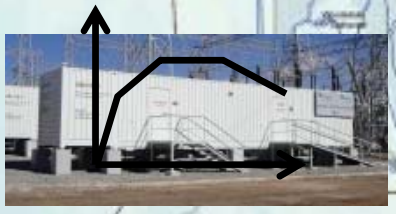
- Optimizing battery storage



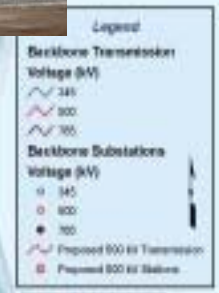




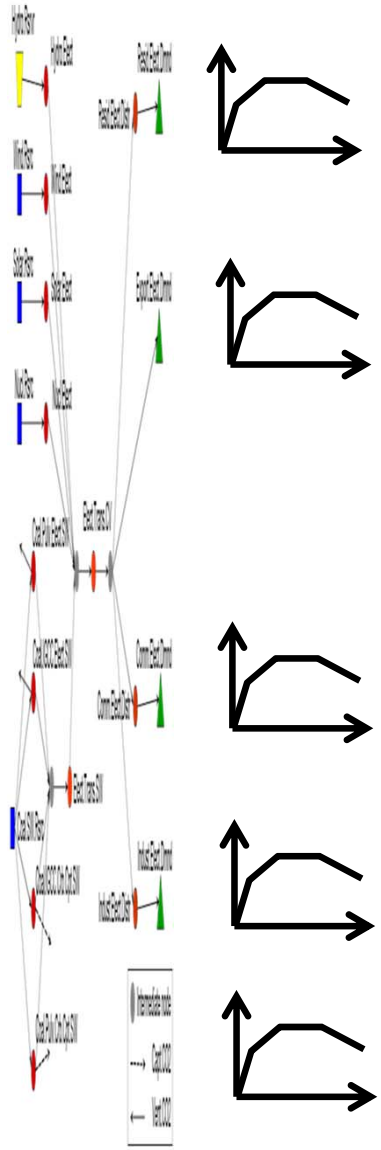




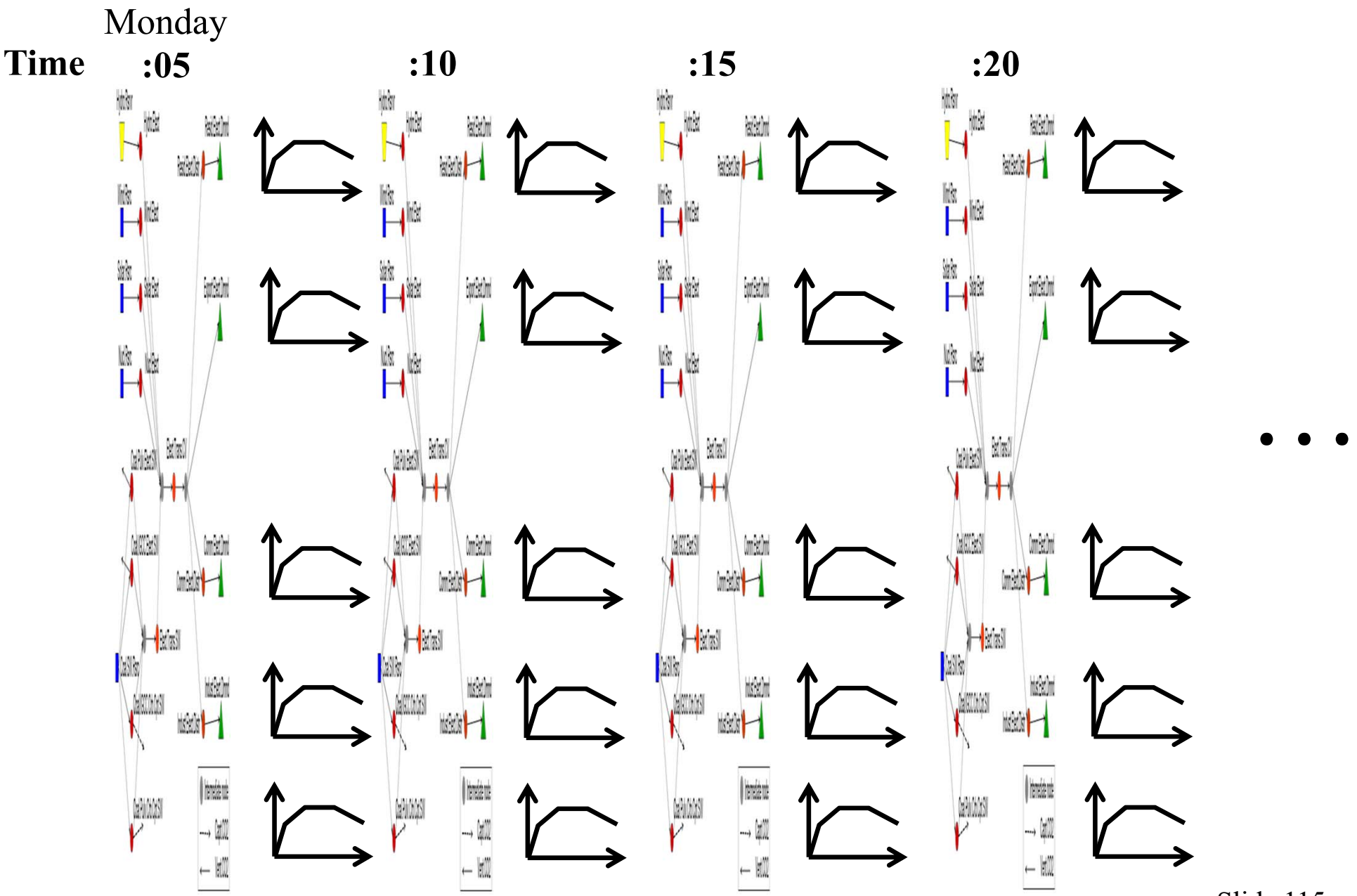




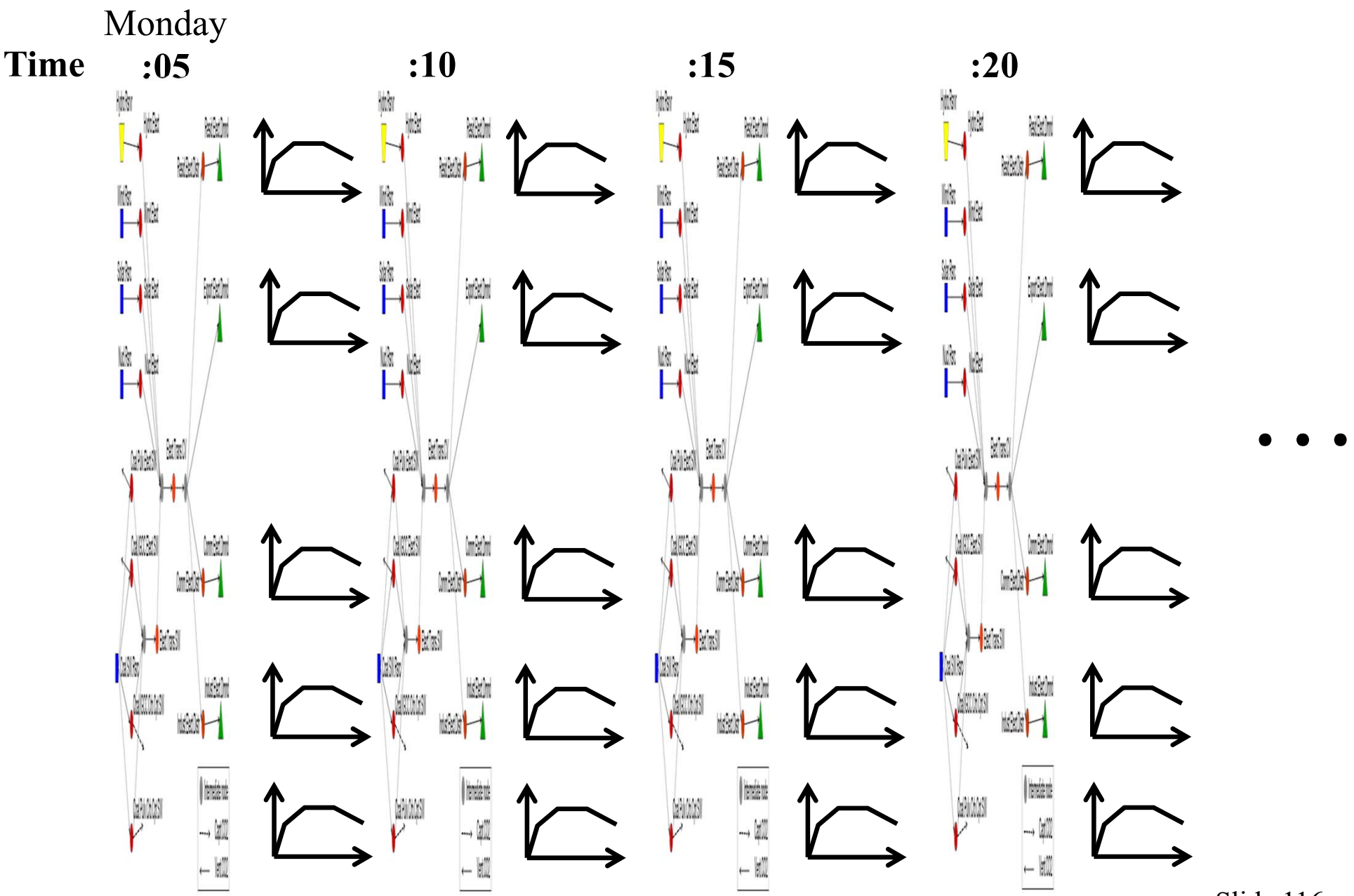
Value function approximations



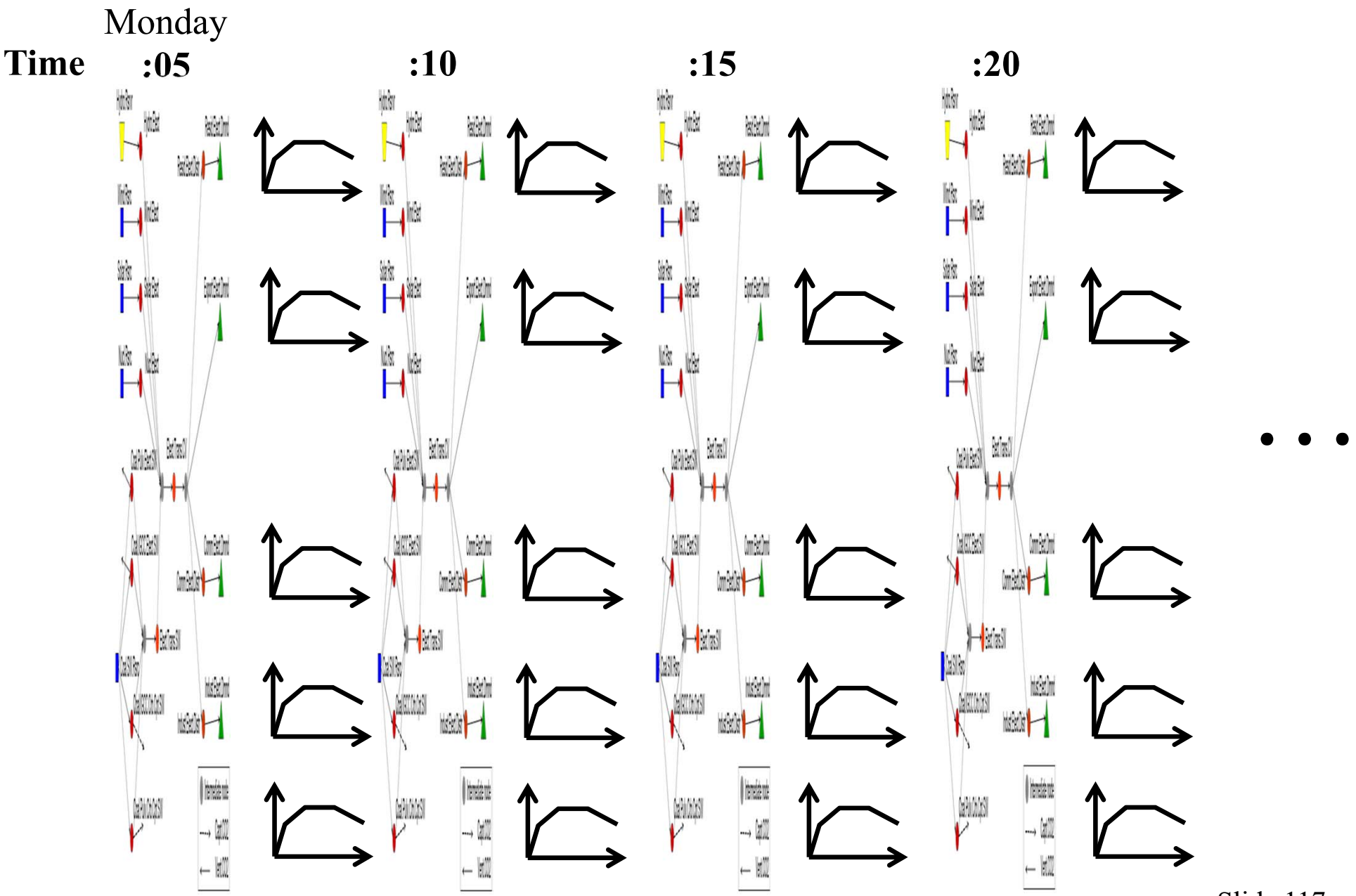
Value function approximations



Value function approximations

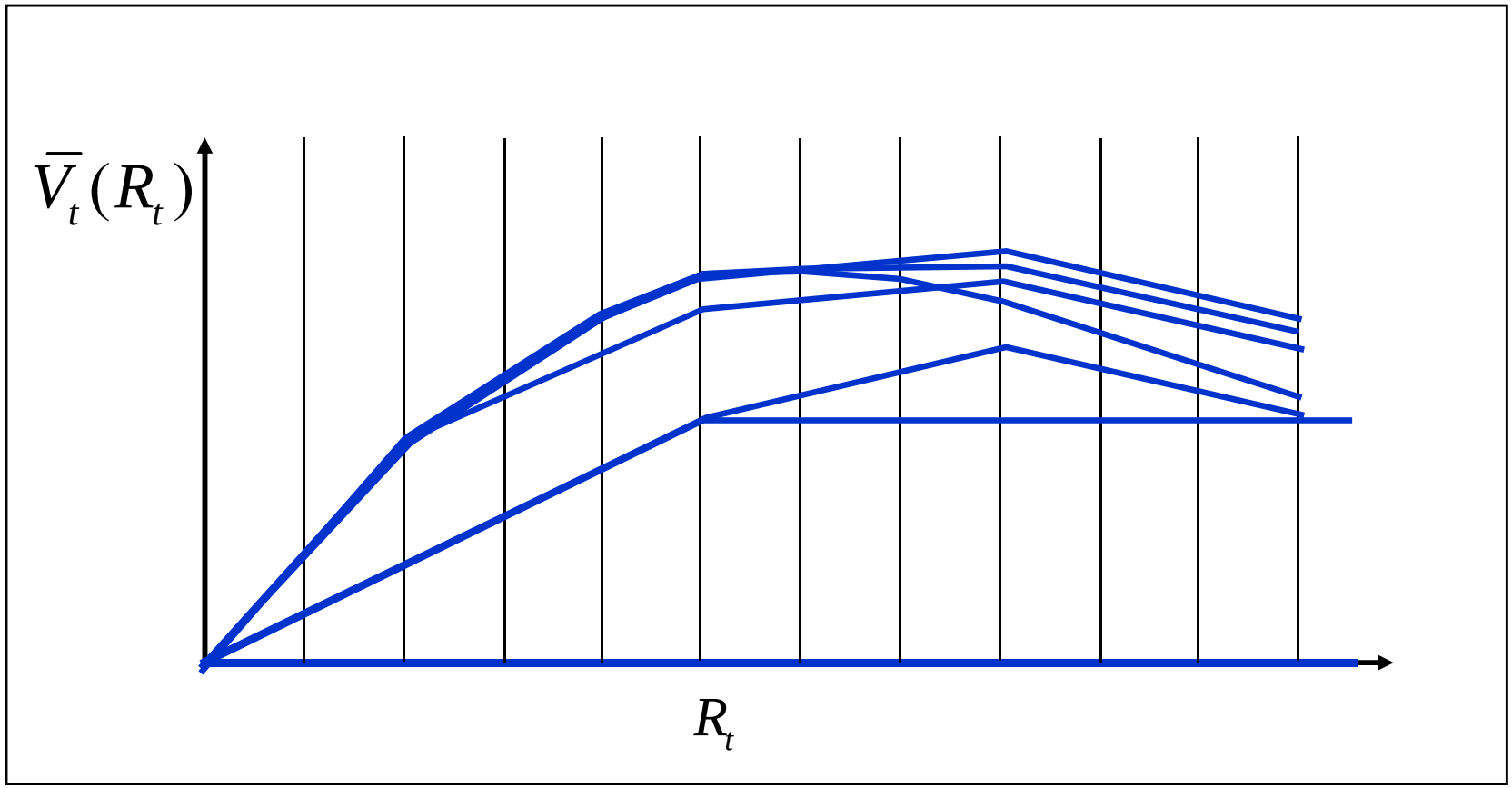


Value function approximations



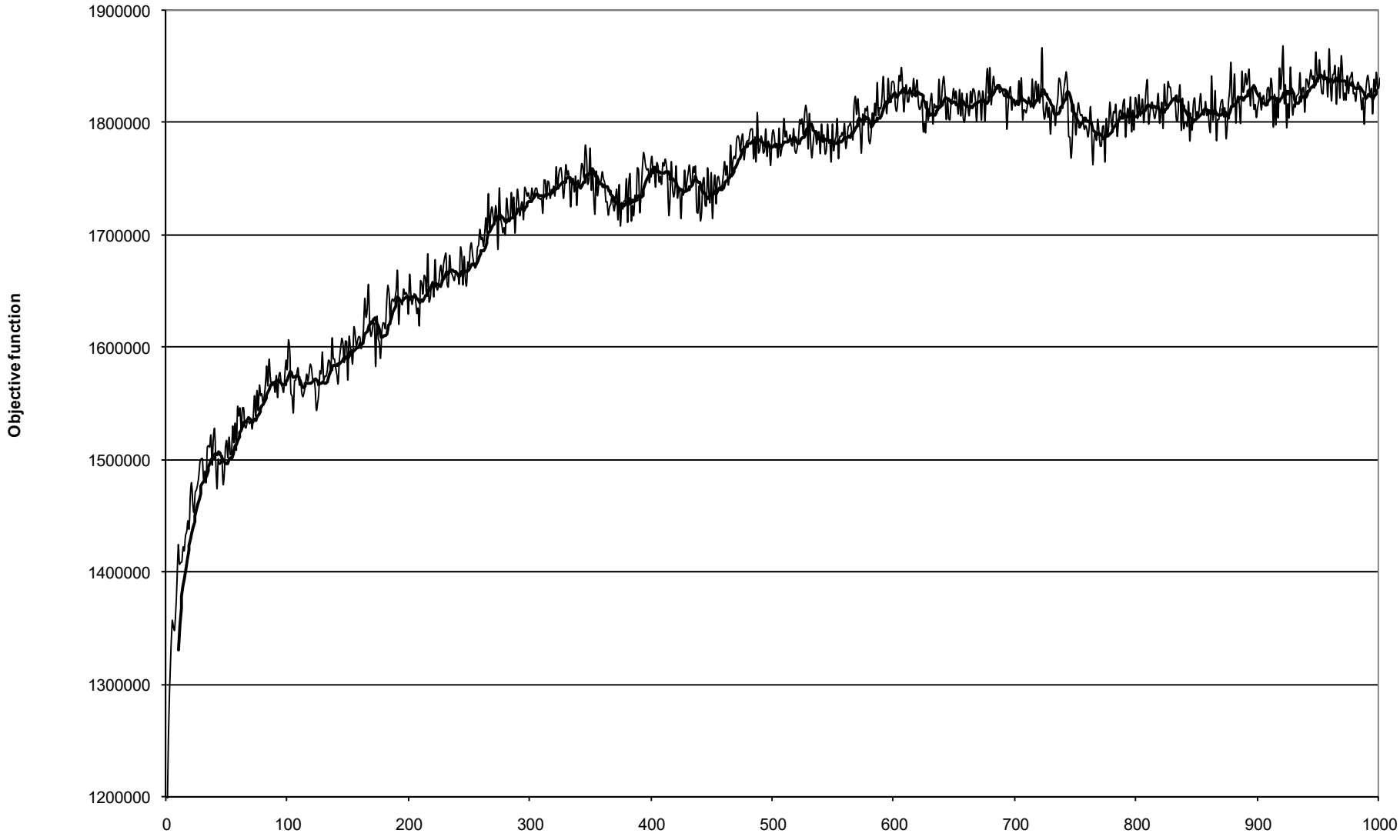
Exploiting concavity

- Derivatives are used to estimate a piecewise linear approximation



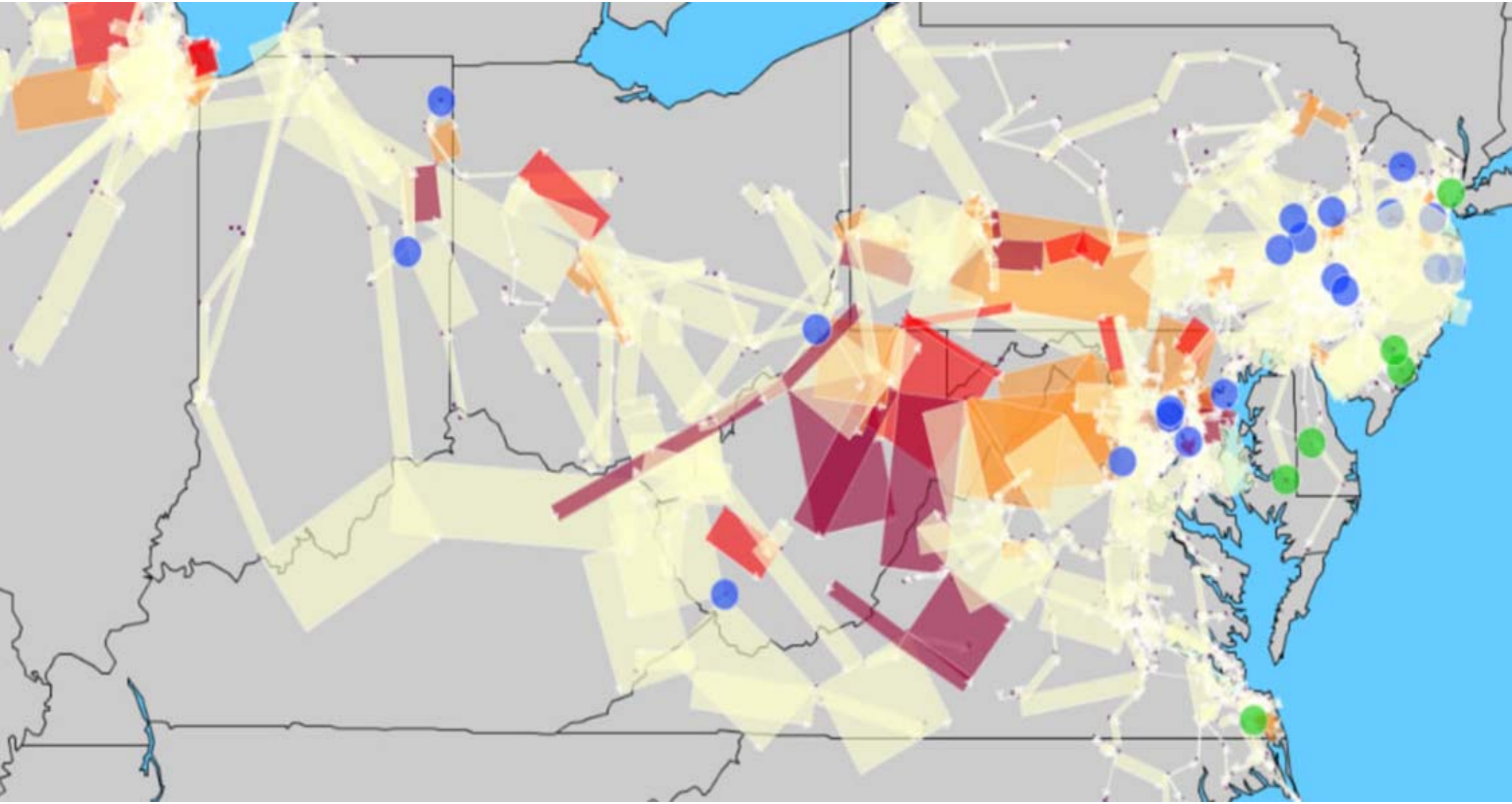
Approximate dynamic programming

- … a typical performance graph.



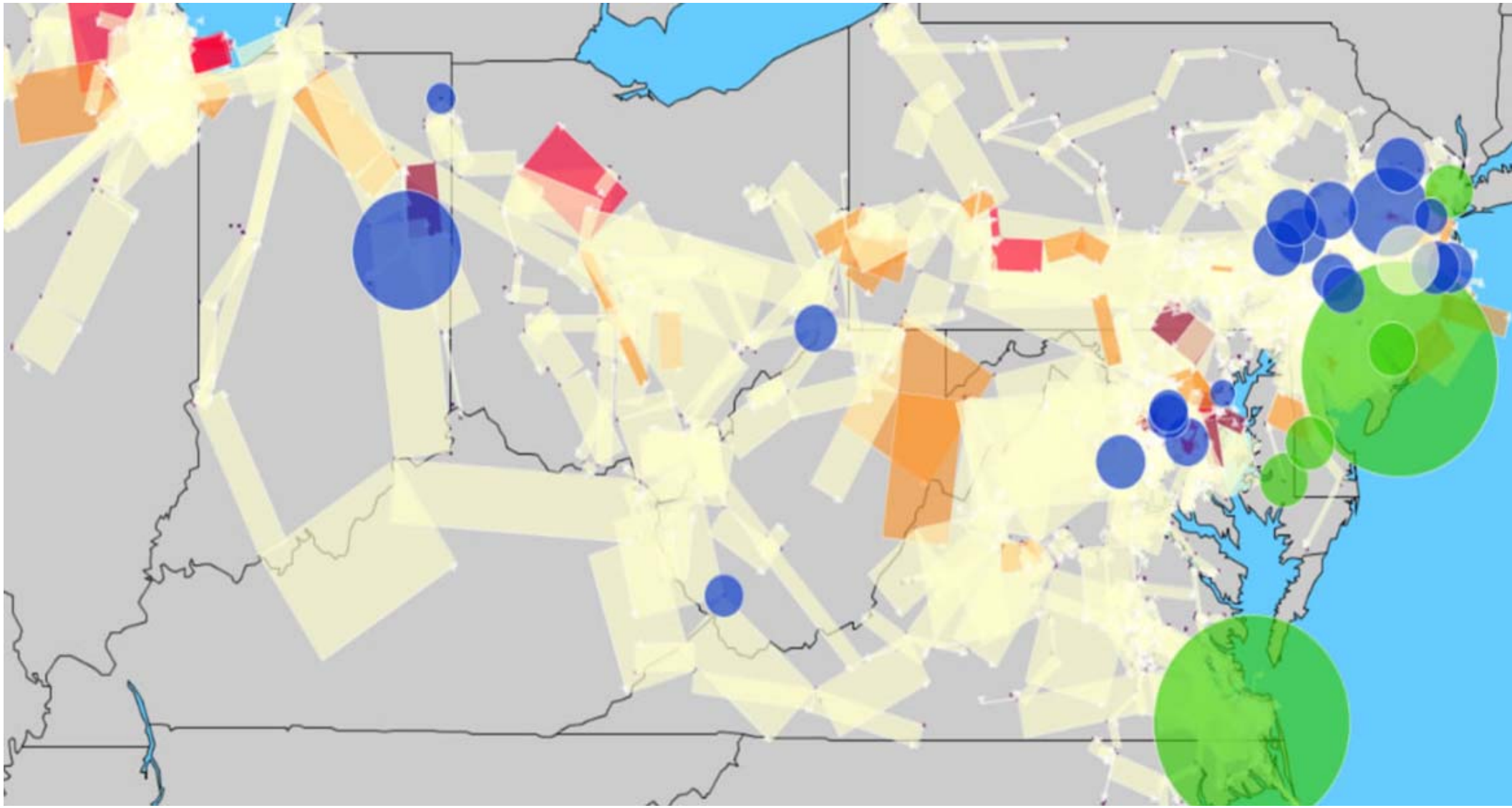
Grid level storage control

- Without storage



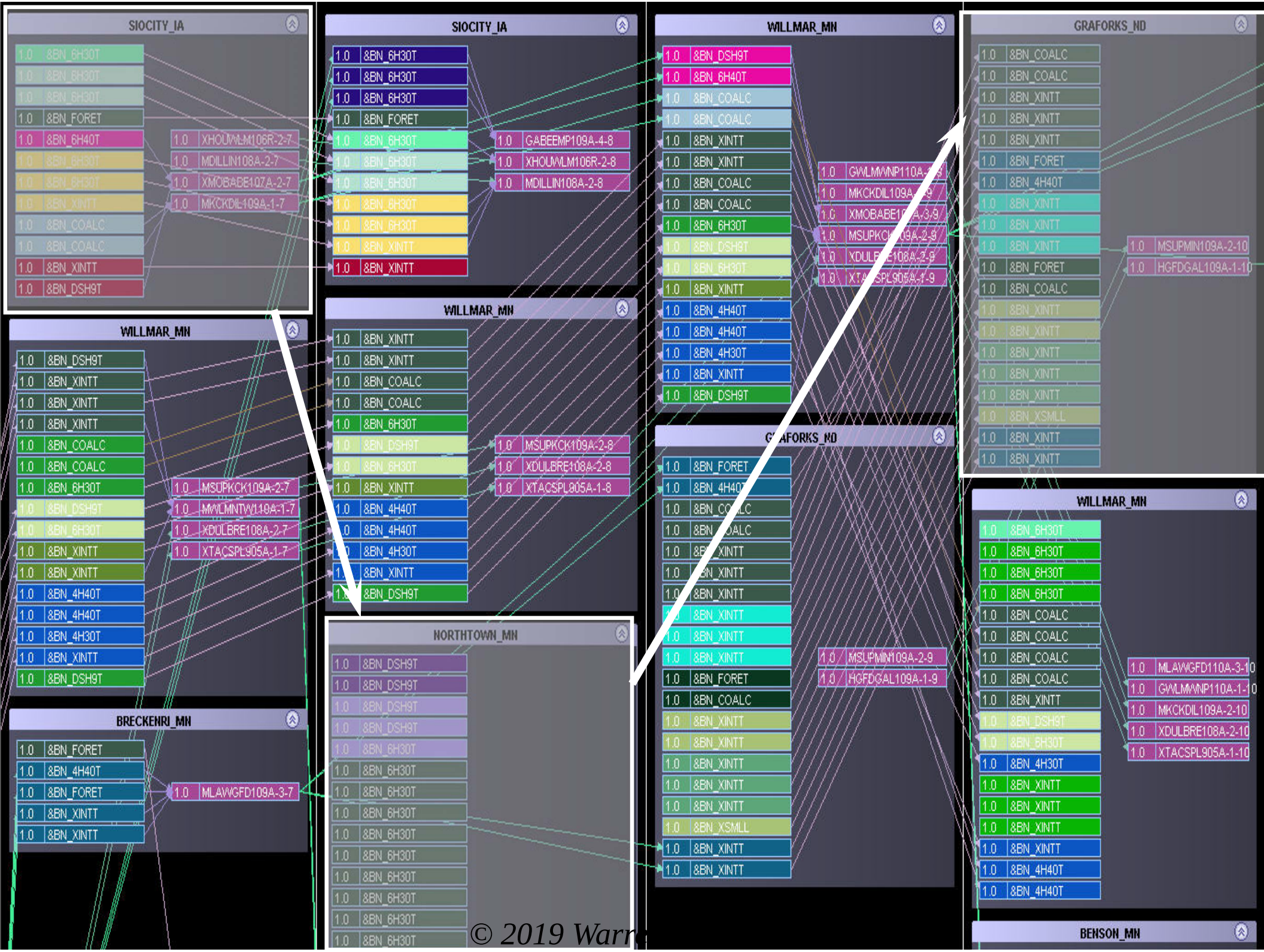
Grid level storage control

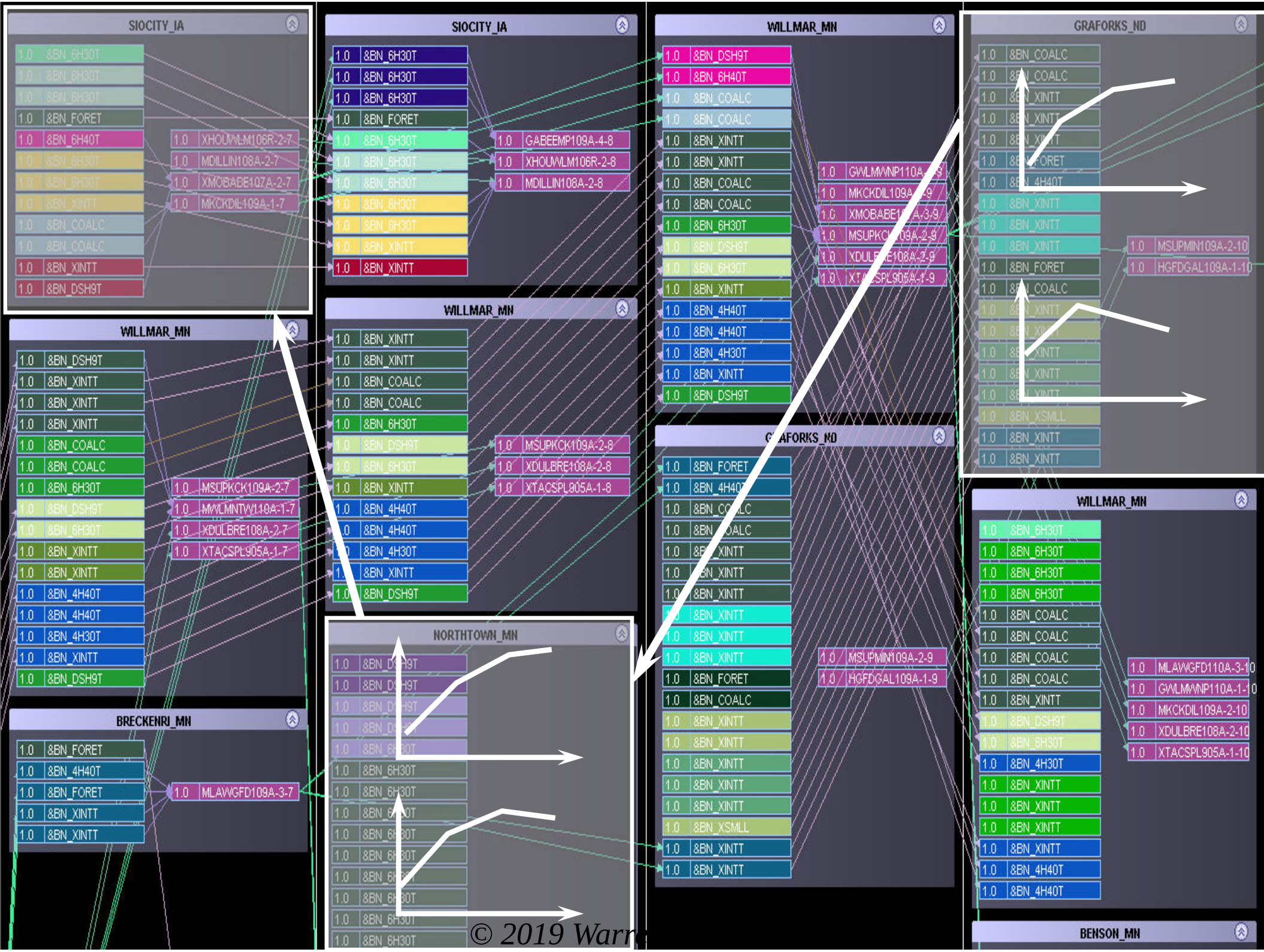
- With storage

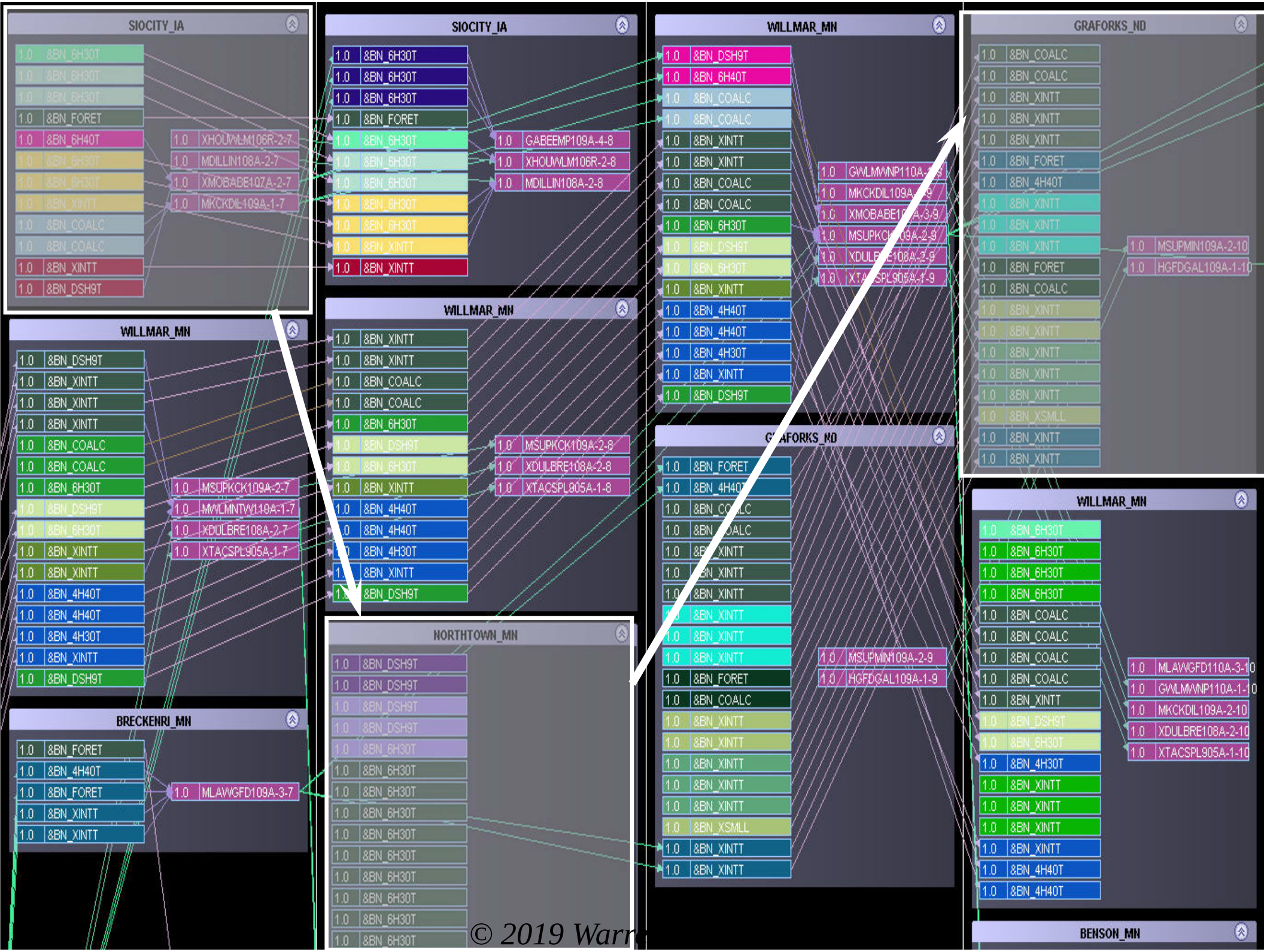


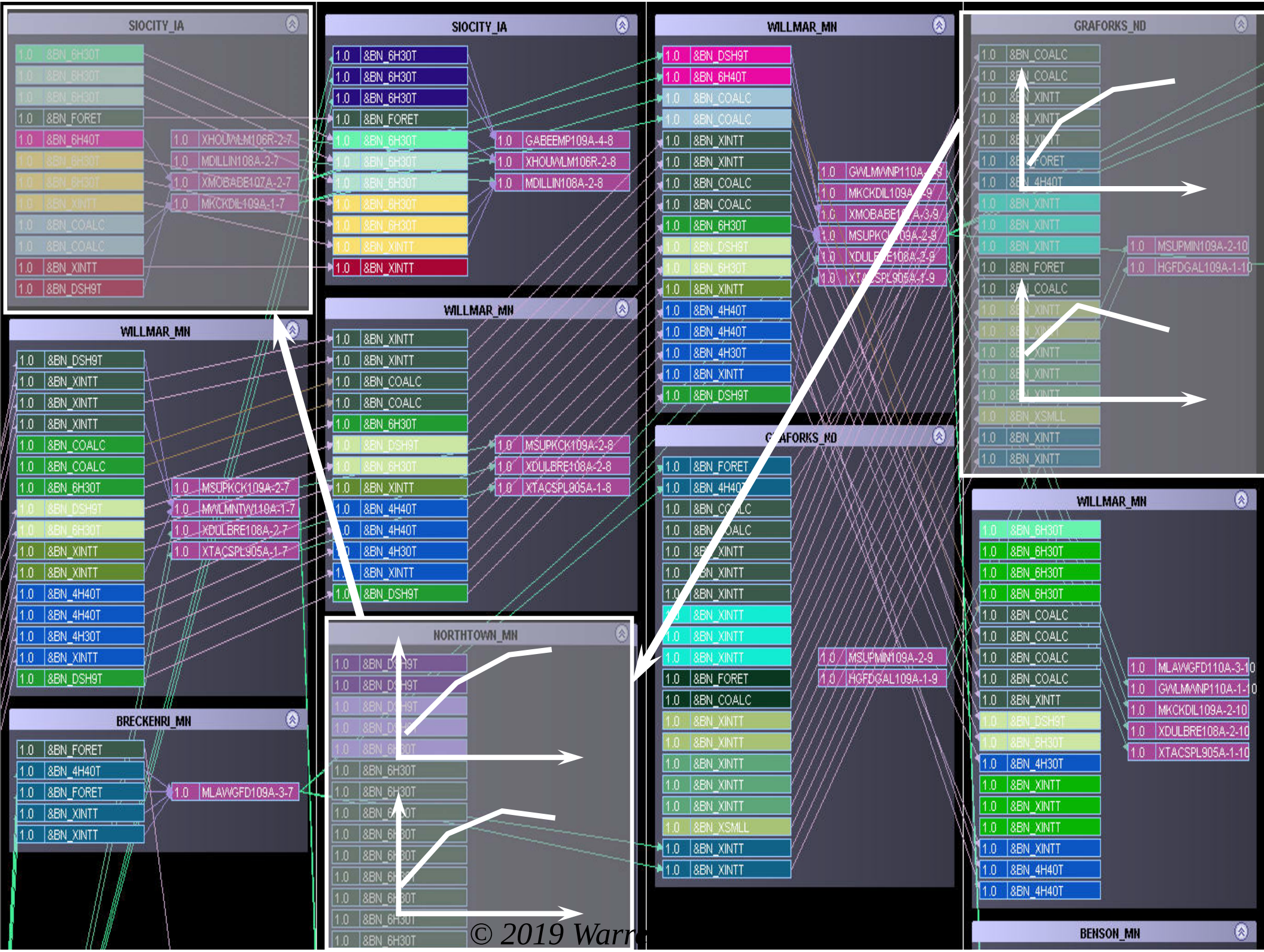
Approximate dynamic programming

Locomotive application

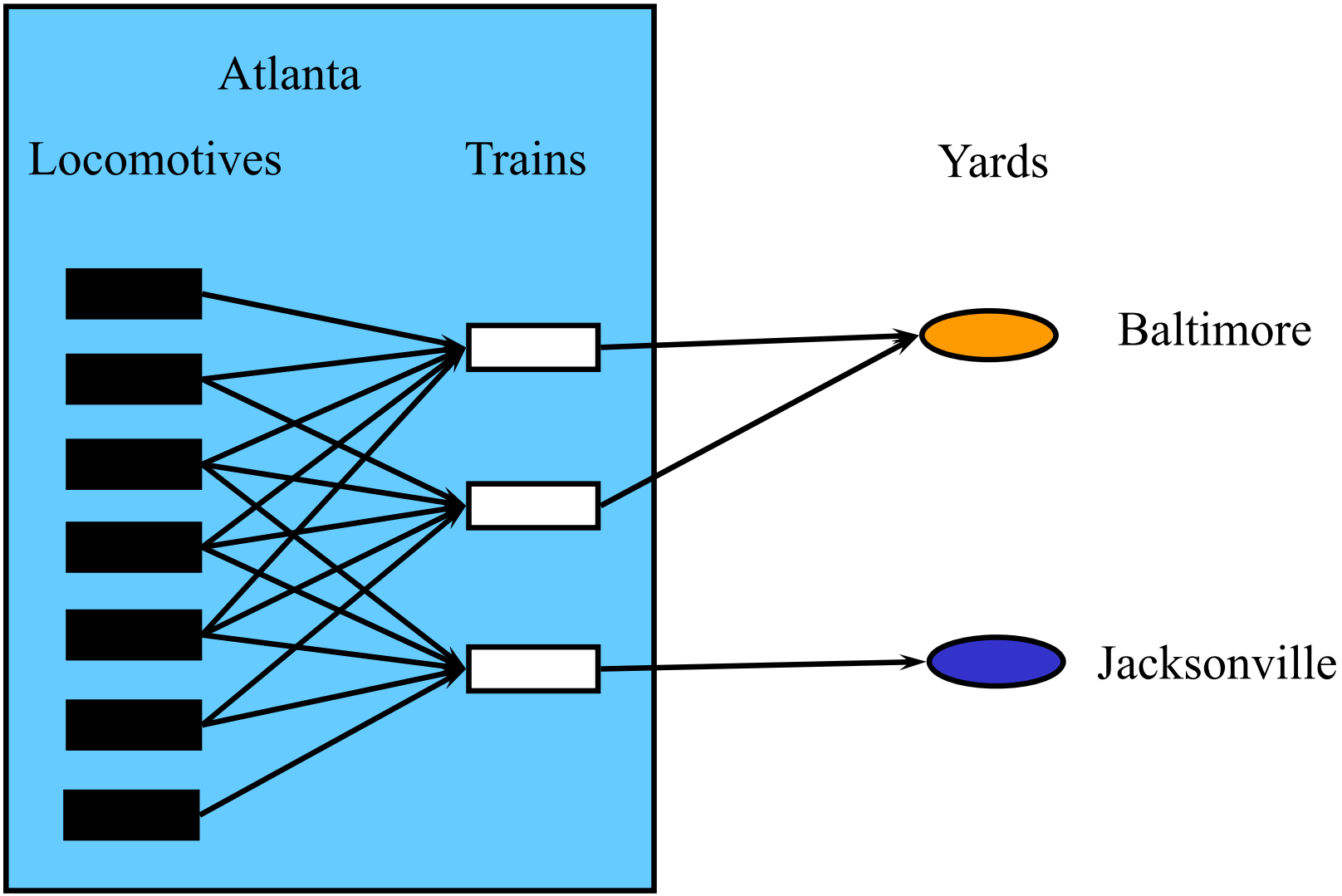

SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H40T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 XHOUWLM106R-2-7
1.0 MDILLIN108A-2-7
1.0 XMOBABE107A-2-7
1.0 MKCKDIL109A-1-7
SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 GABEEMP109A-4-8
1.0 XHOUWLM106R-2-8
1.0 MDILLIN108A-2-8
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_6H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 GWLMWNP110A-3-9
1.0 MKCKDIL109A-2-9
1.0 XMOBABE107A-3-9
1.0 MSUPKCK109A-2-9
1.0 XDULBRE108A-2-9
1.0 XTACSPL905A-1-9
GRAFORKS_ND
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-10
1.0 HGFDGAL109A-1-10
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-7
1.0 MWLMNTW110A-1-7
1.0 XDULBRE108A-2-7
1.0 XTACSPL905A-1-7
WILLMAR_MN
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-8
1.0 XDULBRE108A-2-8
1.0 XTACSPL905A-1-8
GRAFORKS_ND
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-9
1.0 HGFDGAL109A-1-9
BRECKENRI_MN
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_FORET
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MLAWGFD109A-3-7
NORTHTOWN_MN
1.0 &BN_DSH9T
1.0 &BN_DSH9T
1.0 &BN_DSH9T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
WILLMAR_MN
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 MLAWGFD110A-3-10
1.0 GWLMWNP110A-1-10
1.0 MKCKDIL109A-2-10
1.0 XDULBRE108A-2-10
1.0 XTACSPL905A-1-10
BENSON_MN


SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H40T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 XHOUWLM106R-2-7
1.0 MDILLIN108A-2-7
1.0 XMOBABE107A-2-7
1.0 MKCKDIL109A-1-7
SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 GABEEMP109A-4-8
1.0 XHOUWLM106R-2-8
1.0 MDILLIN108A-2-8
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_6H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 GWLMWNP110A-3-9
1.0 MKCKDIL109A-3-9
1.0 XMOBABE107A-3-9
1.0 MSUPKCK109A-2-9
1.0 XDULBRE108A-2-9
1.0 XTACSPL905A-1-9
GRAFORKS_ND
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-10
1.0 HGFDGAL109A-1-10
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-7
1.0 MWLMNTW110A-1-7
1.0 XDULBRE108A-2-7
1.0 XTACSPL905A-1-7
WILLMAR_MN
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-8
1.0 XDULBRE108A-2-8
1.0 XTACSPL905A-1-8
GRAFORKS_ND
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-9
1.0 HGFDGAL109A-1-9
WILLMAR_MN
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 MLAWGFD110A-3-10
1.0 GWLMWNP110A-1-10
1.0 MKCKDIL109A-2-10
1.0 XDULBRE108A-2-10
1.0 XTACSPL905A-1-10
BRECKENRI_MN
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_FORET
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MLAWGFD109A-3-7
NORTHTOWN_MN
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
BENSON_MN
© 2019 Warr

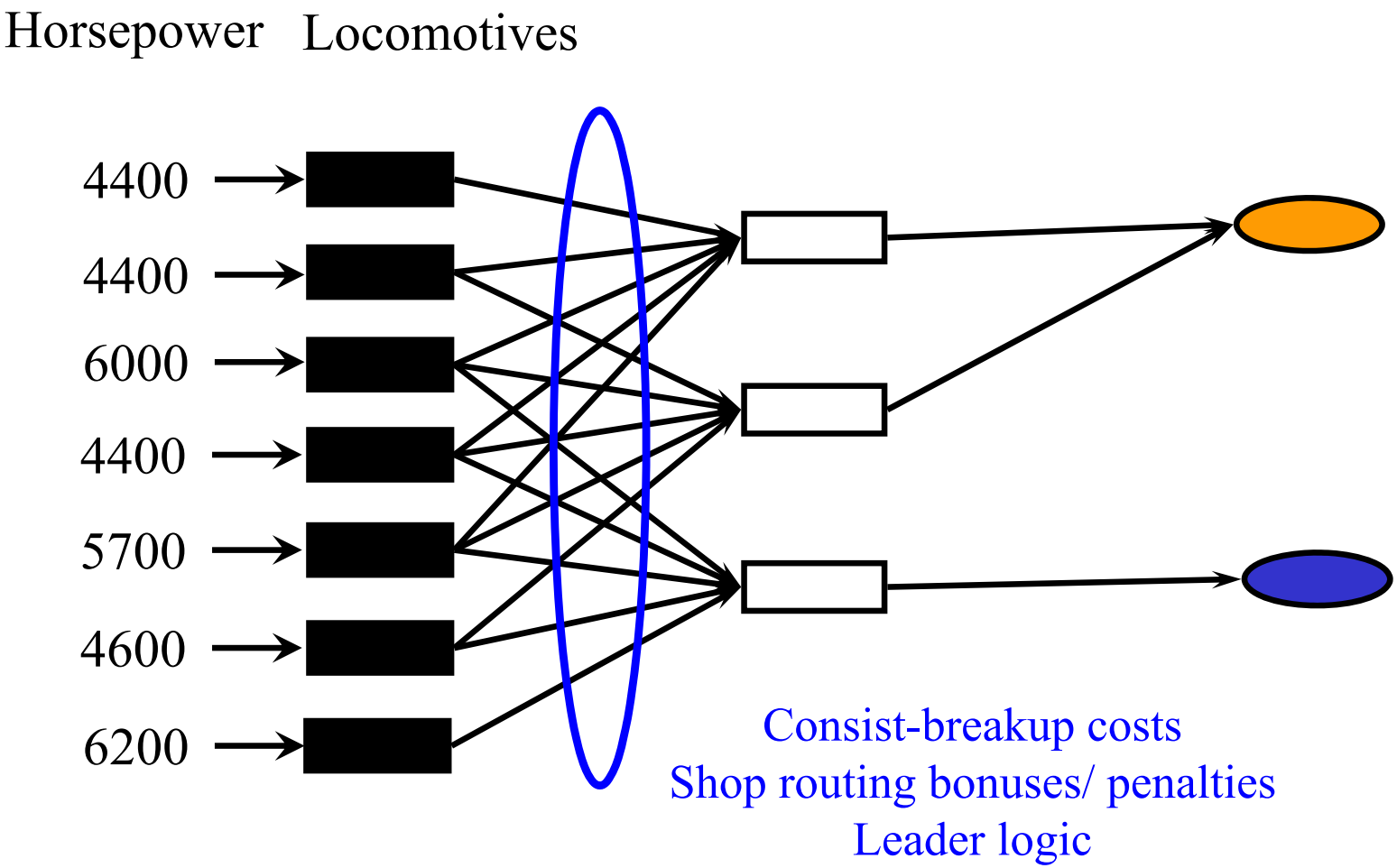

SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H40T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 XHOUWLM106R-2-7
1.0 MDILLIN108A-2-7
1.0 XMOBABE107A-2-7
1.0 MKCKDIL109A-1-7
SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 GABEEMP109A-4-8
1.0 XHOUWLM106R-2-8
1.0 MDILLIN108A-2-8
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_6H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 GWLMWNP110A-3-9
1.0 MKCKDIL109A-2-9
1.0 XMOBABE107A-3-9
1.0 MSUPKCK109A-2-9
1.0 XDULBRE108A-2-9
1.0 XTACSPL905A-1-9
GRAFORKS_ND
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-10
1.0 HGFDGAL109A-1-10
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-7
1.0 MWLMNTW110A-1-7
1.0 XDULBRE108A-2-7
1.0 XTACSPL905A-1-7
WILLMAR_MN
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-8
1.0 XDULBRE108A-2-8
1.0 XTACSPL905A-1-8
GRAFORKS_ND
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-9
1.0 HGFDGAL109A-1-9
BRECKENRI_MN
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_FORET
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MLAWGFD109A-3-7
NORTHTOWN_MN
1.0 &BN_DSH9T
1.0 &BN_DSH9T
1.0 &BN_DSH9T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
WILLMAR_MN
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 MLAWGFD110A-3-10
1.0 GWLMWNP110A-1-10
1.0 MKCKDIL109A-2-10
1.0 XDULBRE108A-2-10
1.0 XTACSPL905A-1-10
BENSON_MN


SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H40T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 XHOUWLM106R-2-7
1.0 MDILLIN108A-2-7
1.0 XMOBABE107A-2-7
1.0 MKCKDIL109A-1-7
SIOCITY_IA
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_FORET
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 GABEEMP109A-4-8
1.0 XHOUWLM106R-2-8
1.0 MDILLIN108A-2-8
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_6H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 GWLMWNP110A-3-9
1.0 MKCKDIL109A-3-9
1.0 XMOBABE107A-3-9
1.0 MSUPKCK109A-2-9
1.0 XDULBRE108A-2-9
1.0 XTACSPL905A-1-9
GRAFORKS_ND
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-10
1.0 HGFDGAL109A-1-10
WILLMAR_MN
1.0 &BN_DSH9T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-7
1.0 MWLMNTW110A-1-7
1.0 XDULBRE108A-2-7
1.0 XTACSPL905A-1-7
WILLMAR_MN
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_6H30T
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 MSUPKCK109A-2-8
1.0 XDULBRE108A-2-8
1.0 XTACSPL905A-1-8
GRAFORKS_ND
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_FORET
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XSMLL
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MSUPMIN109A-2-9
1.0 HGFDGAL109A-1-9
WILLMAR_MN
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_COALC
1.0 &BN_XINTT
1.0 &BN_DSH9T
1.0 &BN_6H30T
1.0 &BN_4H30T
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 &BN_4H40T
1.0 &BN_4H40T
1.0 MLAWGFD110A-3-10
1.0 GWLMWNP110A-1-10
1.0 MKCKDIL109A-2-10
1.0 XDULBRE108A-2-10
1.0 XTACSPL905A-1-10
BRECKENRI_MN
1.0 &BN_FORET
1.0 &BN_4H40T
1.0 &BN_FORET
1.0 &BN_XINTT
1.0 &BN_XINTT
1.0 MLAWGFD109A-3-7
NORTHTOWN_MN
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
1.0 &BN_6H30T
BENSON_MN
© 2019 Warr

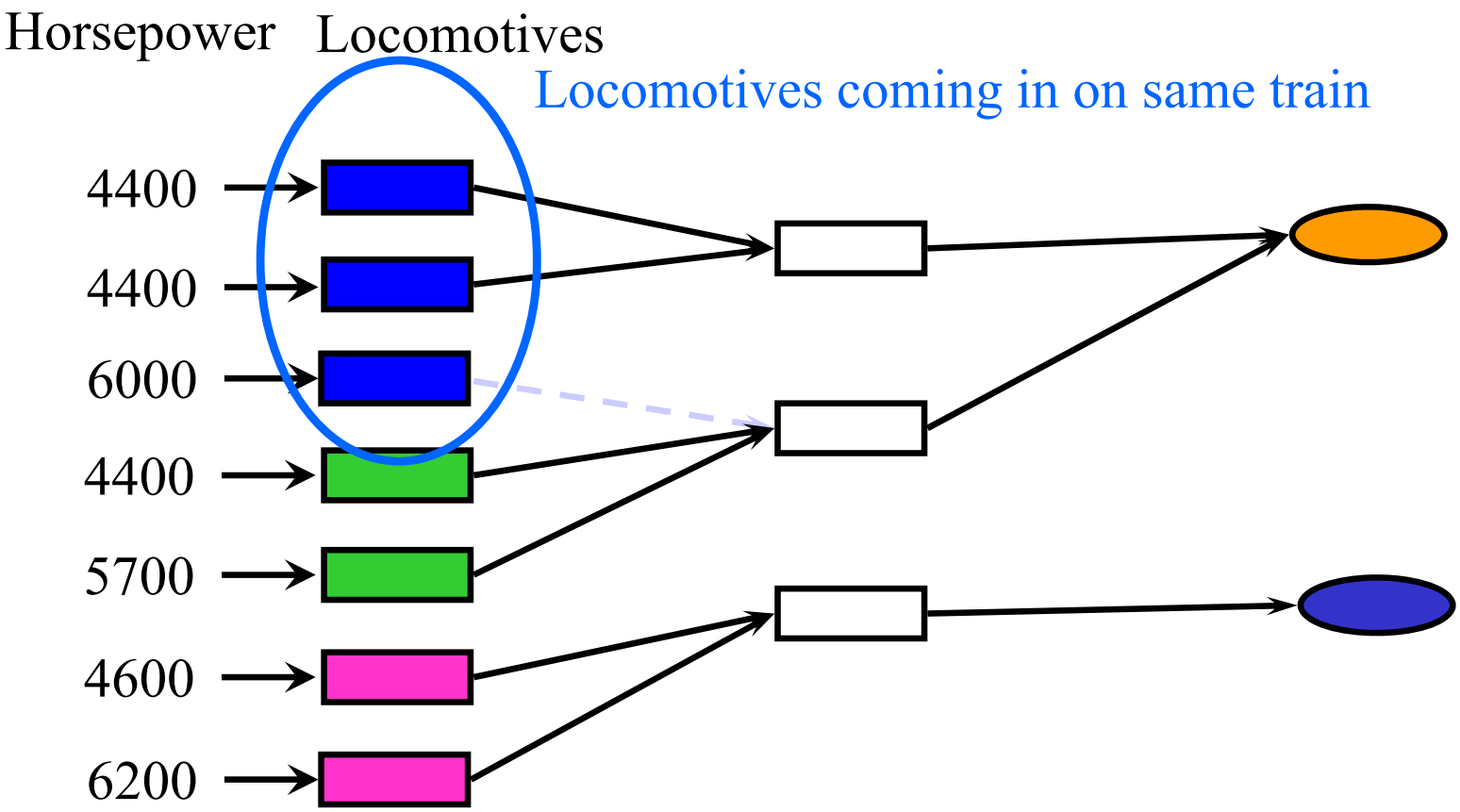
The locomotive assignment problem



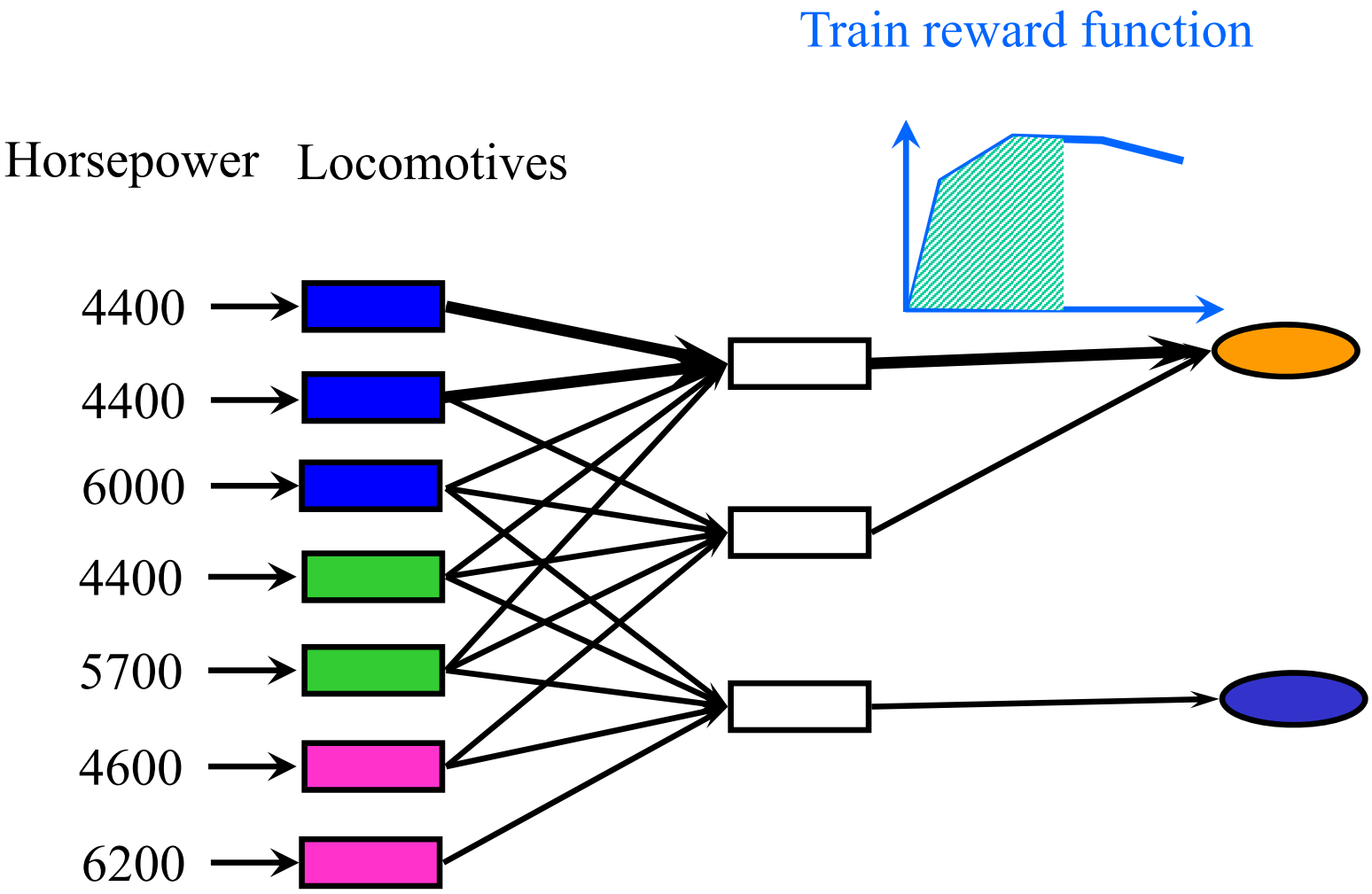
The locomotive assignment problem



The locomotive assignment problem

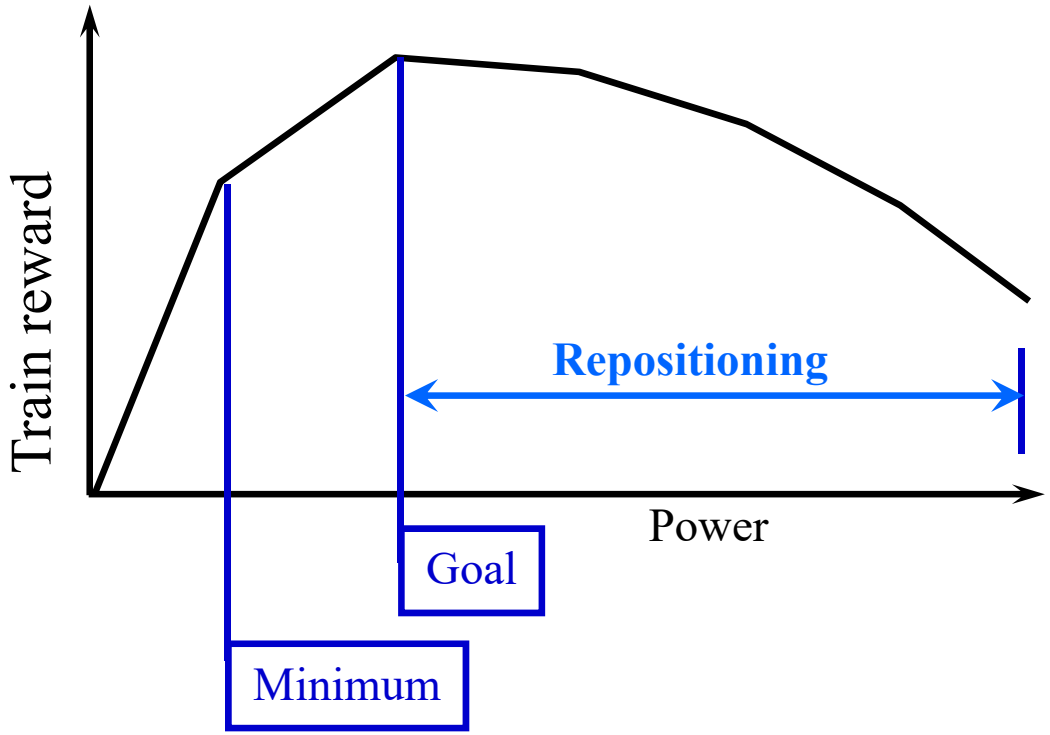


The locomotive assignment problem

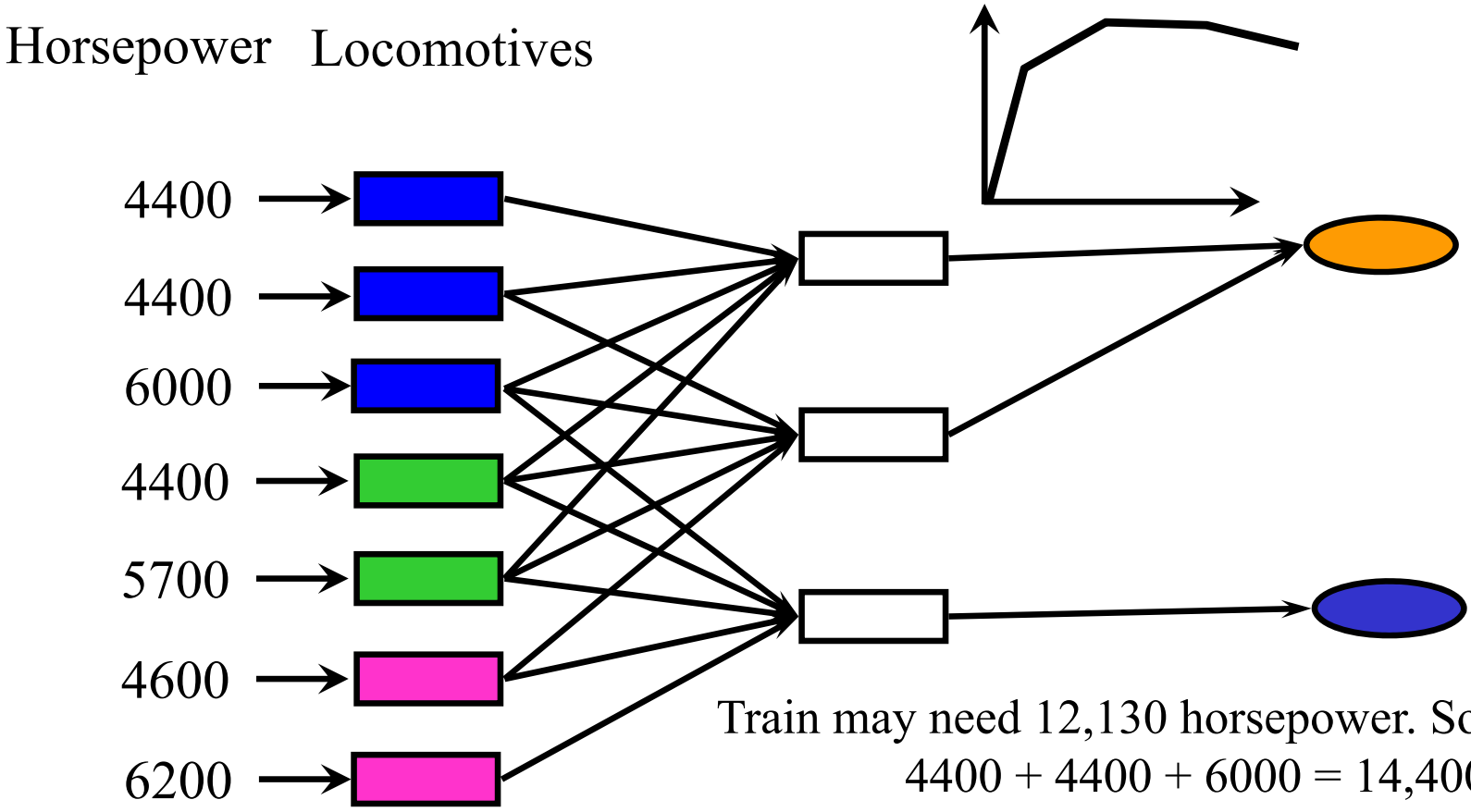


The locomotive assignment problem

- The train reward function



The locomotive assignment problem



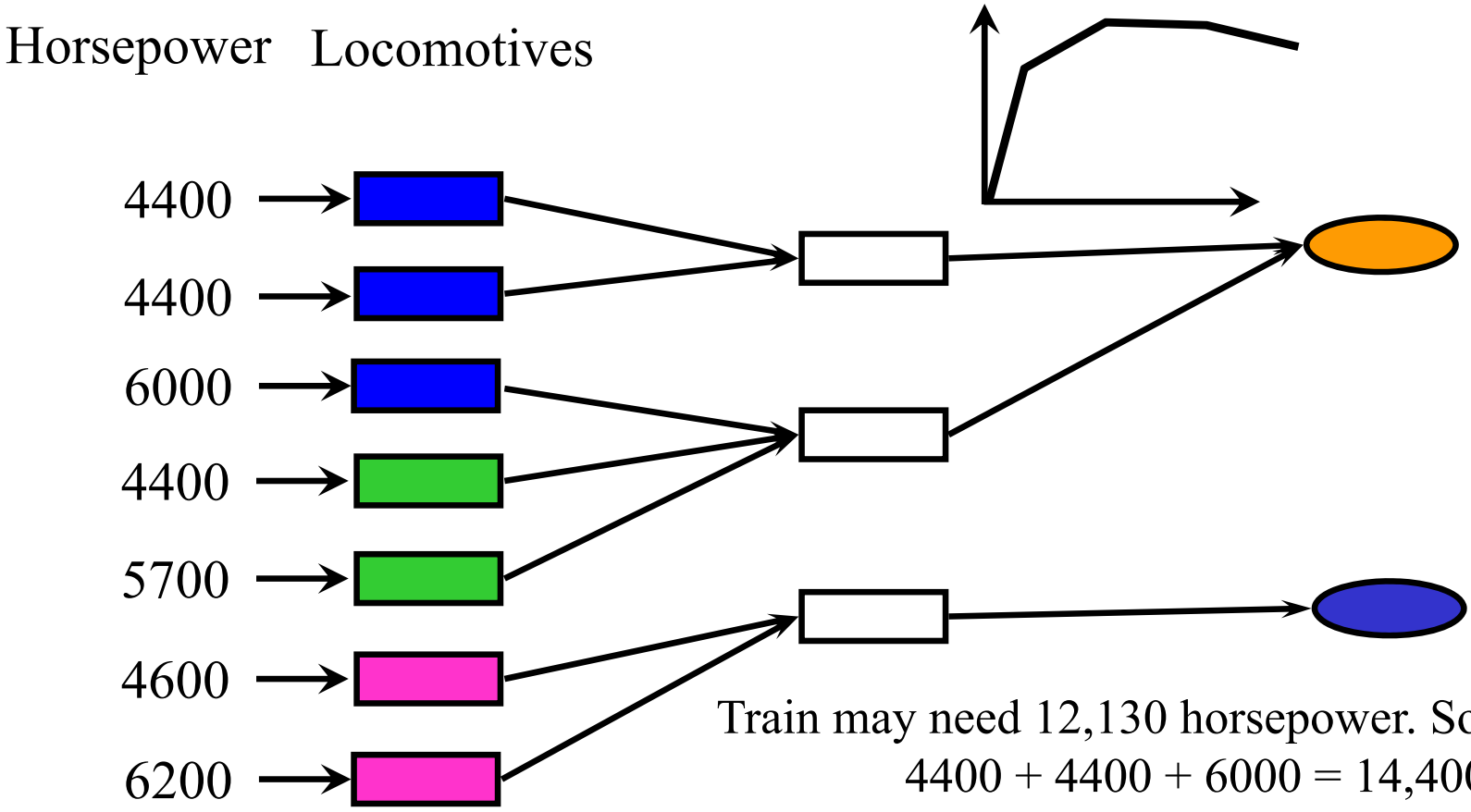
Train may need 12,130 horsepower. Solutions:

4400 + 4400 + 6000 = 14,400

4400+4400+4400 = 13,200

6000+6200 = 12,200

The locomotive assignment problem



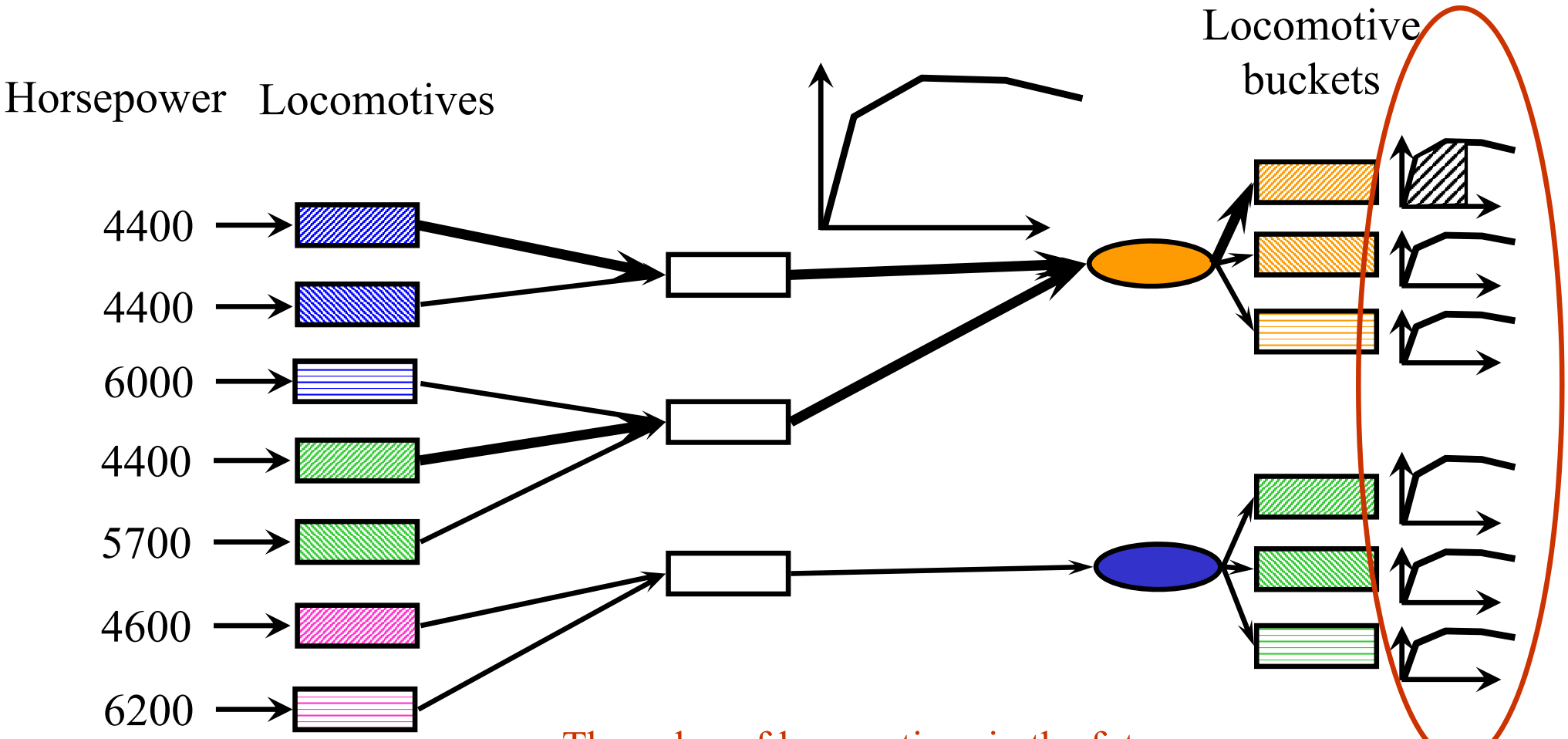
Train may need 12,130 horsepower. Solutions:

4400 + 4400 + 6000 = 14,400

4400+4400+4400 = 13,200

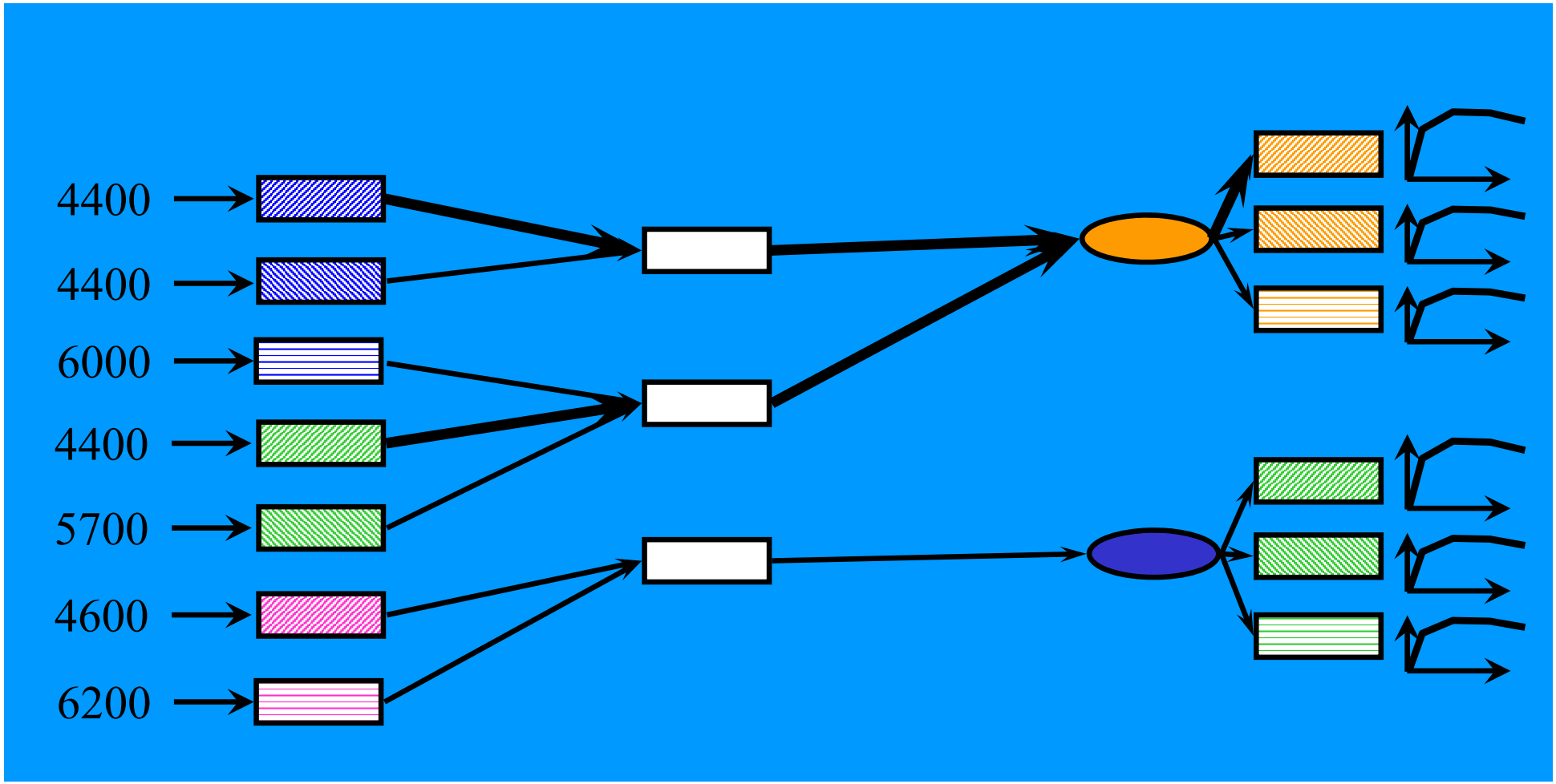
6000+6200 = 12,200

The locomotive assignment problem



The value of locomotives in the future is captured through nonlinear approximations.

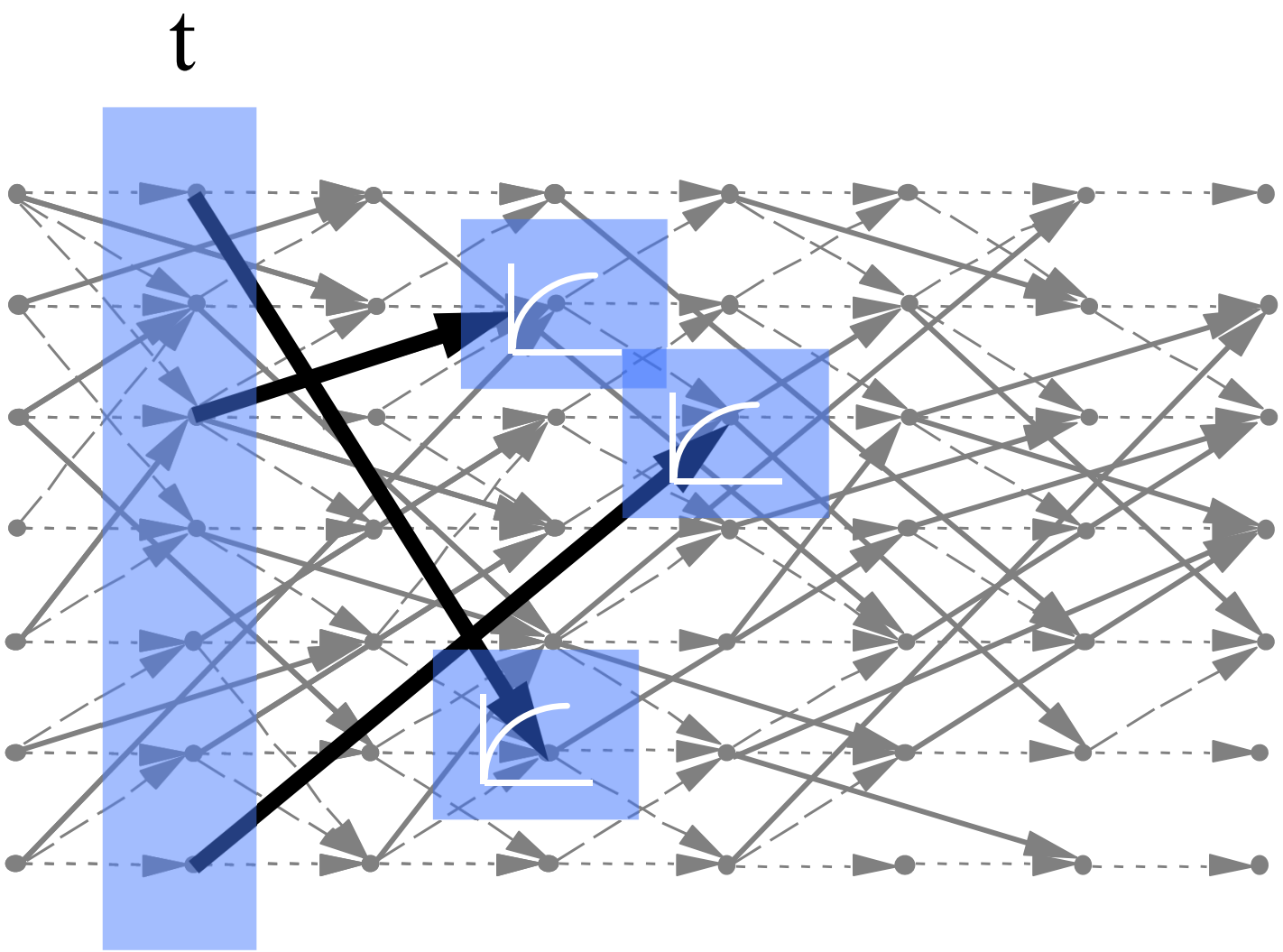
The locomotive assignment problem



Locomotive subproblem can be solved quickly using Cplex

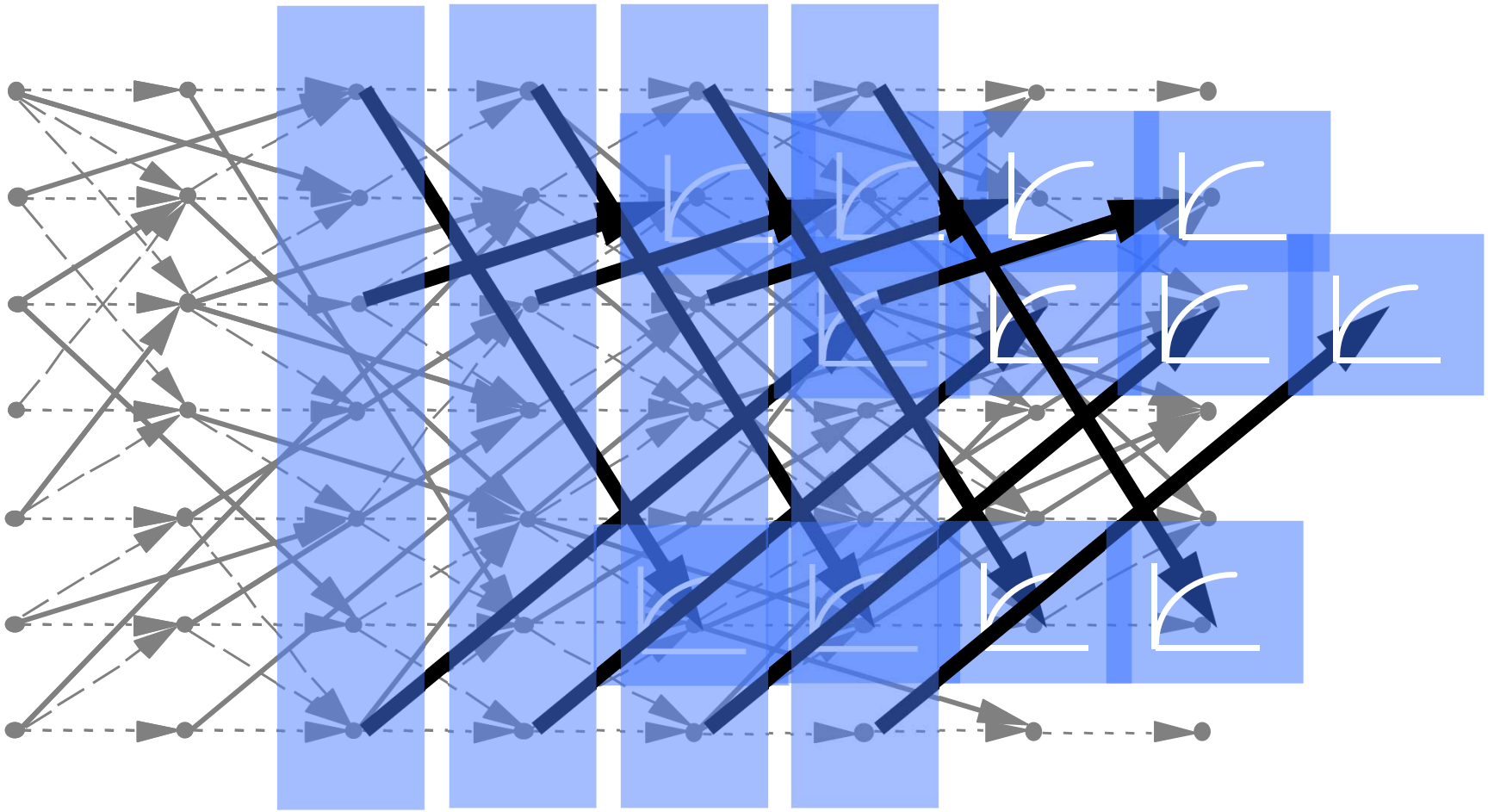
Iterative learning

- Approximate dynamic programming



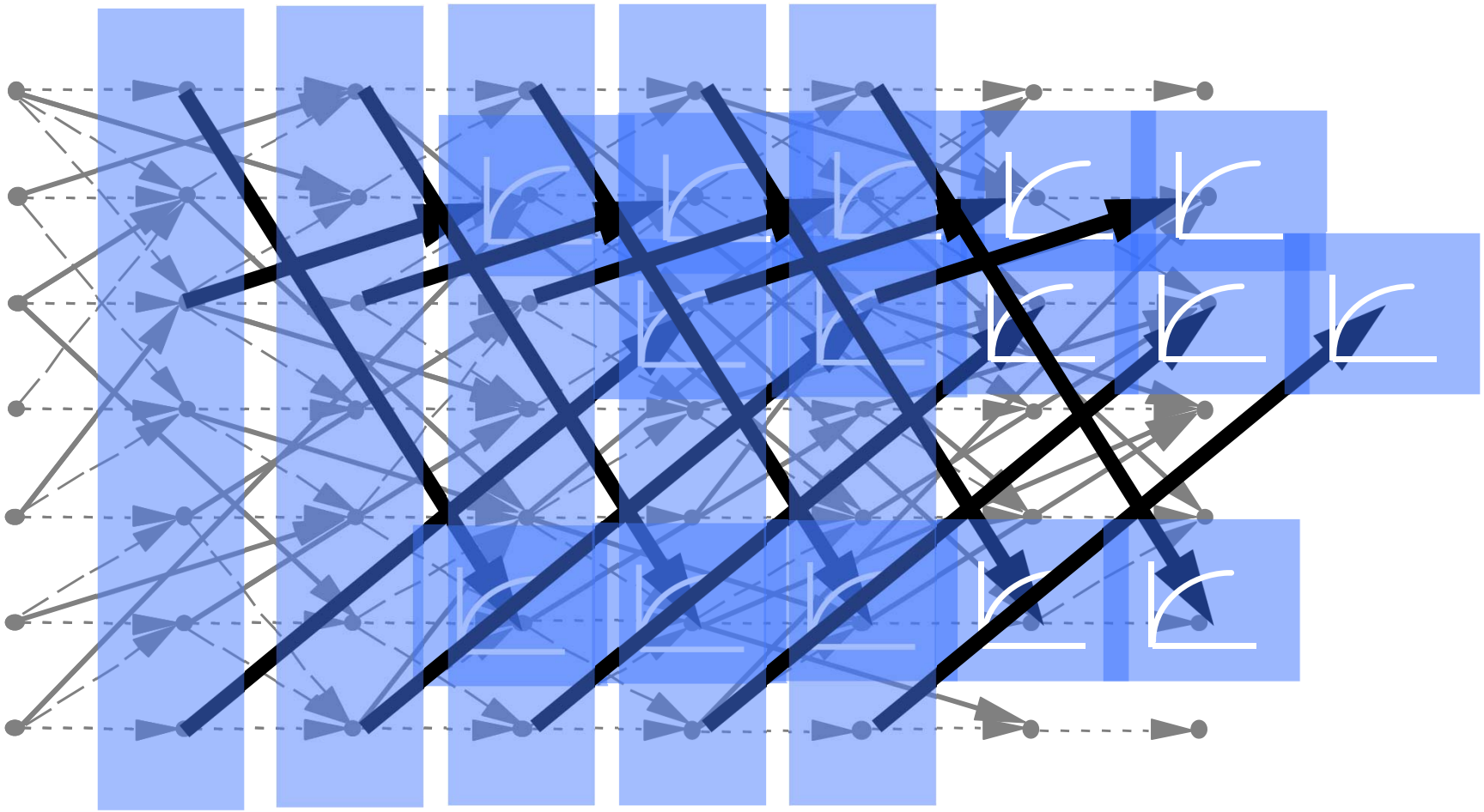
Iterative learning

- Approximate dynamic programming



Iterative learning

- Approximate dynamic programming



Approximate dynamic programming

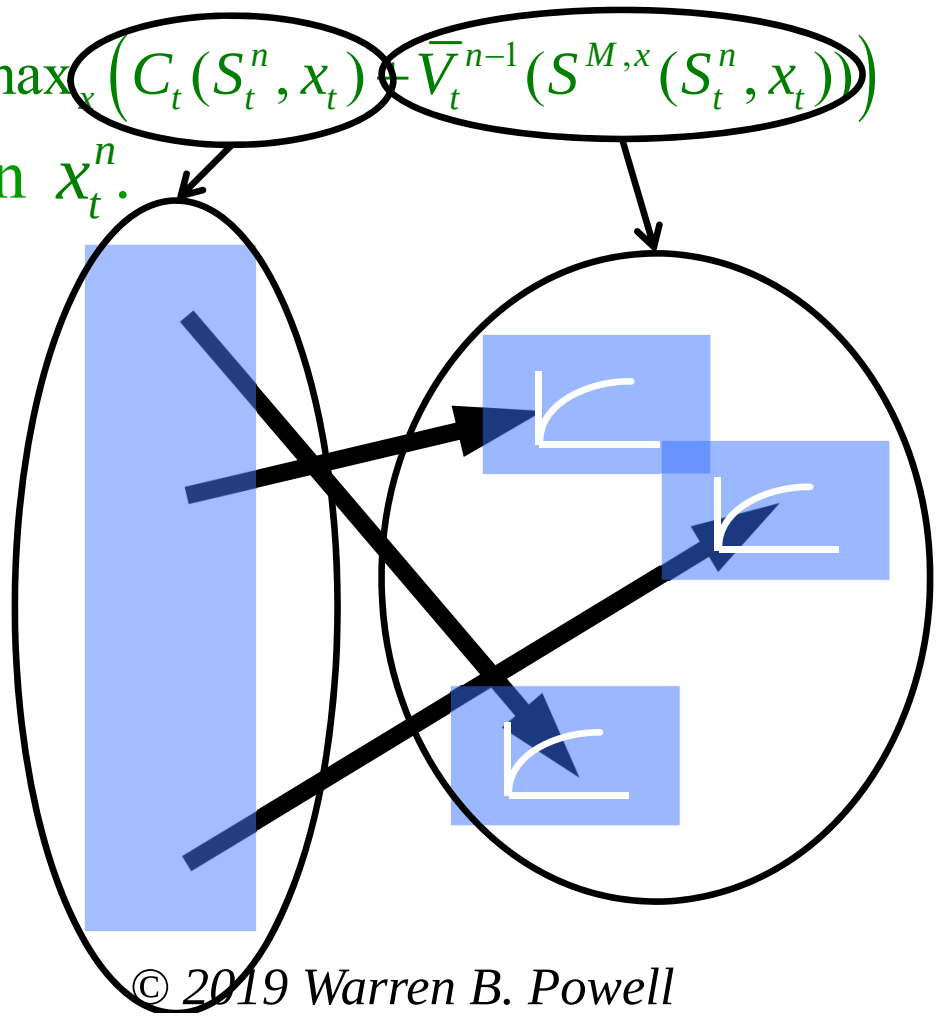
Step 1: Start with a pre-decision state S_t^n

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

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

to obtain x_t^n.

Deterministic optimization



Approximate dynamic programming

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

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

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

to obtain x_t^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:

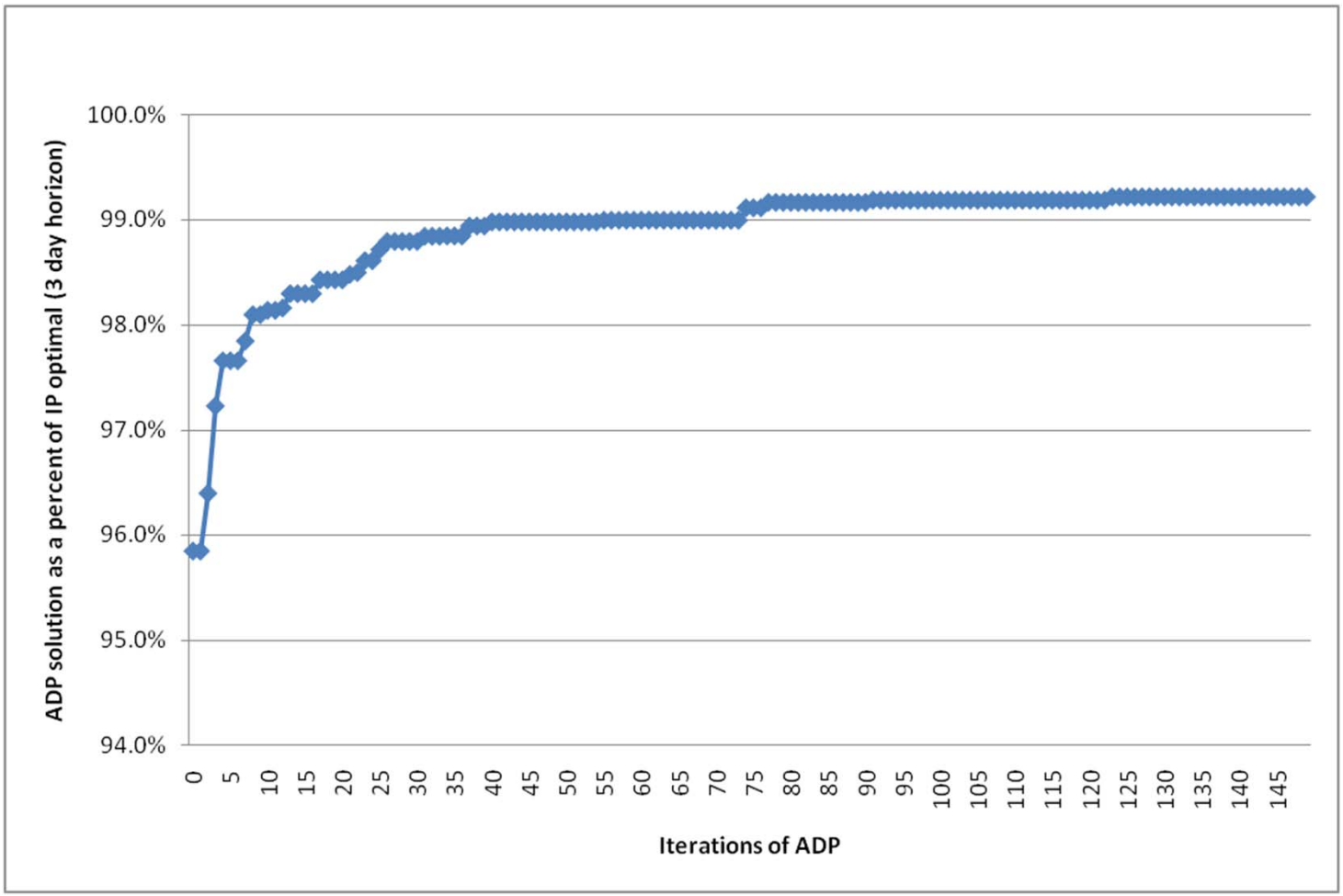
$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Simulation

Step 5: Return to step 1.

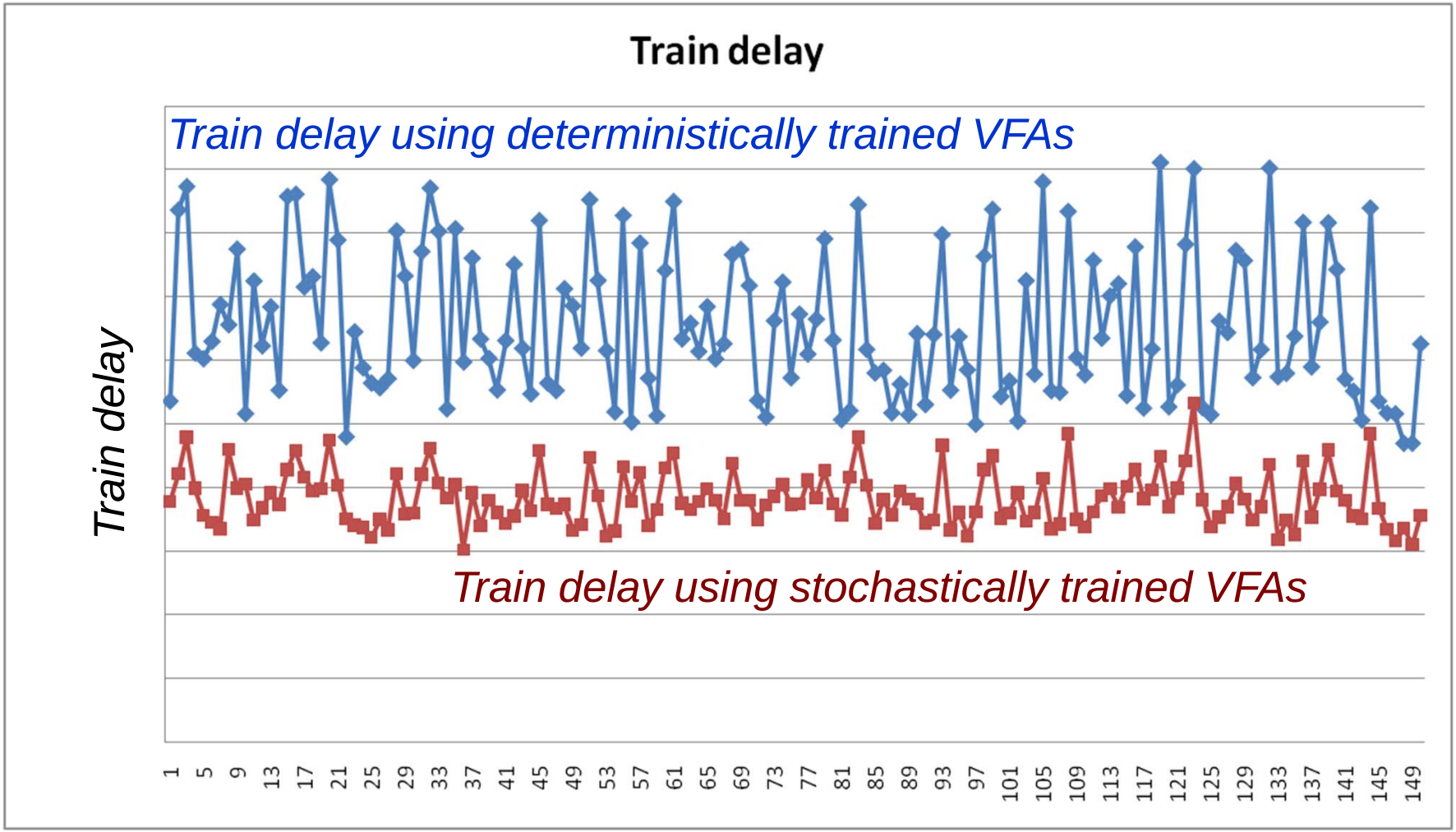
Laboratory testing

- ADP as a percent of IP optimal
  - » 3 day horizon
  - » Five locomotive types



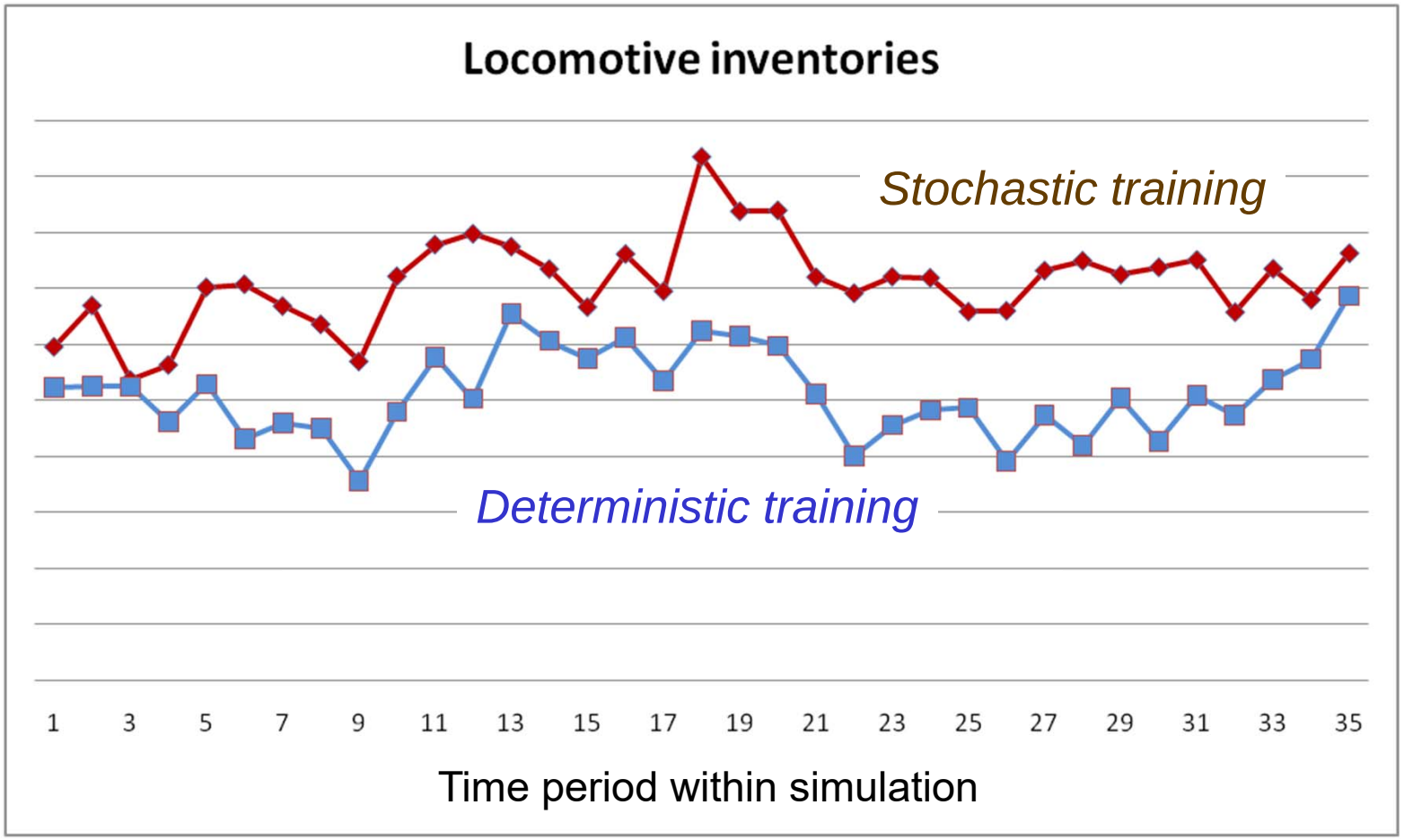
Laboratory testing

- Train delay with uncertain transit times and yard delays



Laboratory testing

- How do we do it?



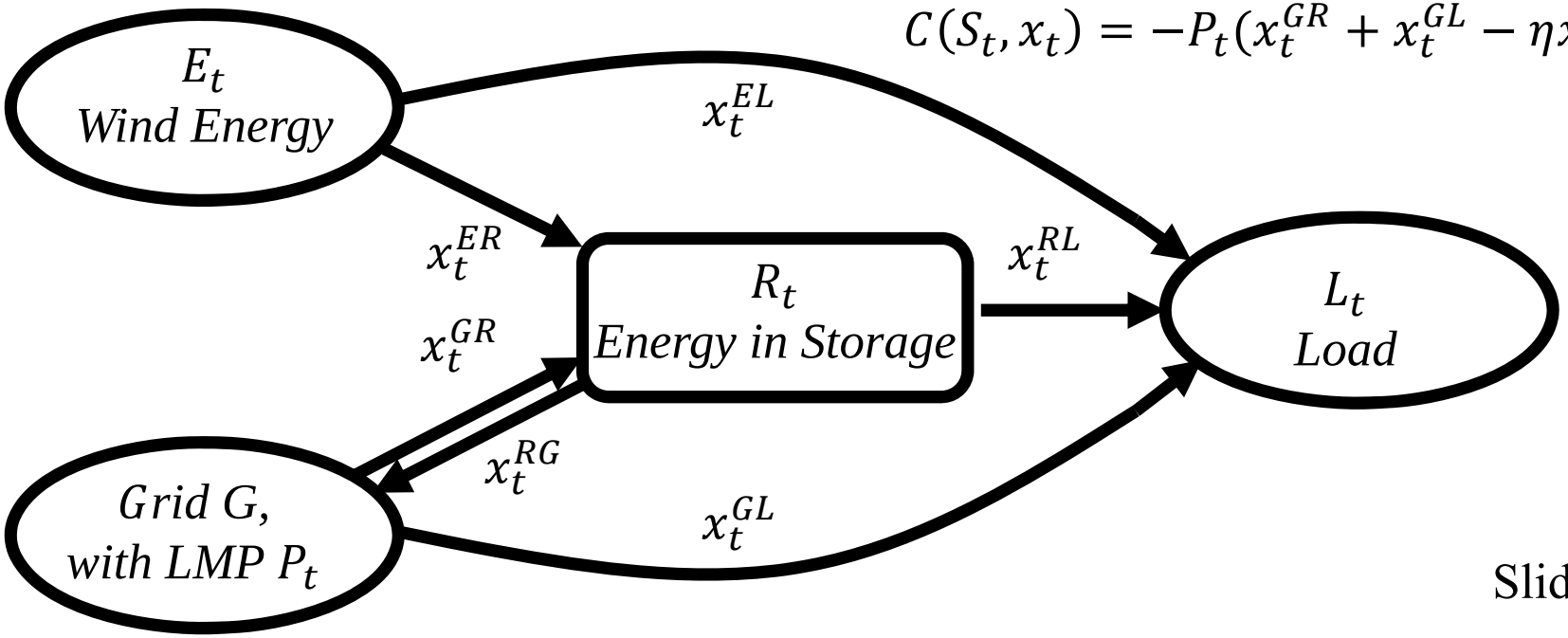
» The stochastic model keeps more power in inventory. The challenge is knowing when and where.

Experiments with ADP for energy storage

"Does anything work"

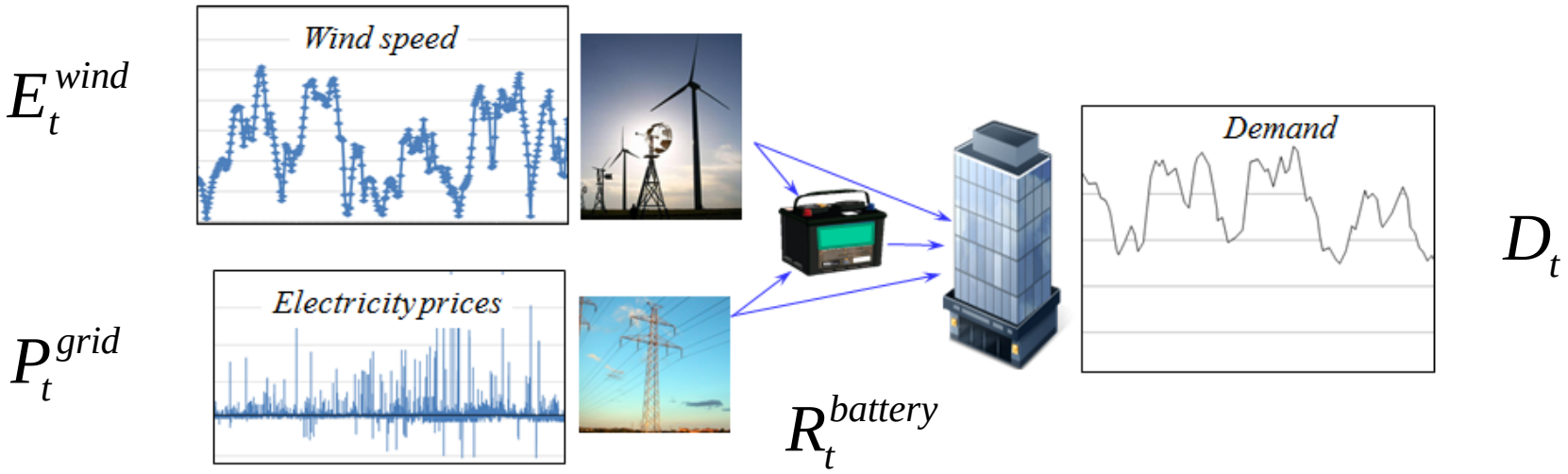
Energy storage model

$$C(S_t, x_t) = -P_t(x_t^{GR} + x_t^{GL} - \eta x_t^{RG})$$



A storage problem

- 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

A storage problem

- Bellman's optimality equation

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

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

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

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

These are the "three curses of dimensionality."

Approximate policy iteration

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

Step 2: Inner loop: Do for m=1,…,M:

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

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

to obtain x^m.

Step 2b: Update the value function approximation

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

Step 2c: Obtain Monte Carlo sample of $W(\omega^m)$ and compute the next pre-decision state:

$$S^{m+1} = S^M(S^m, x^m, W(\omega^m))$$

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

Approximate policy iteration

- Machine learning methods (coded in R)
  - » **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.

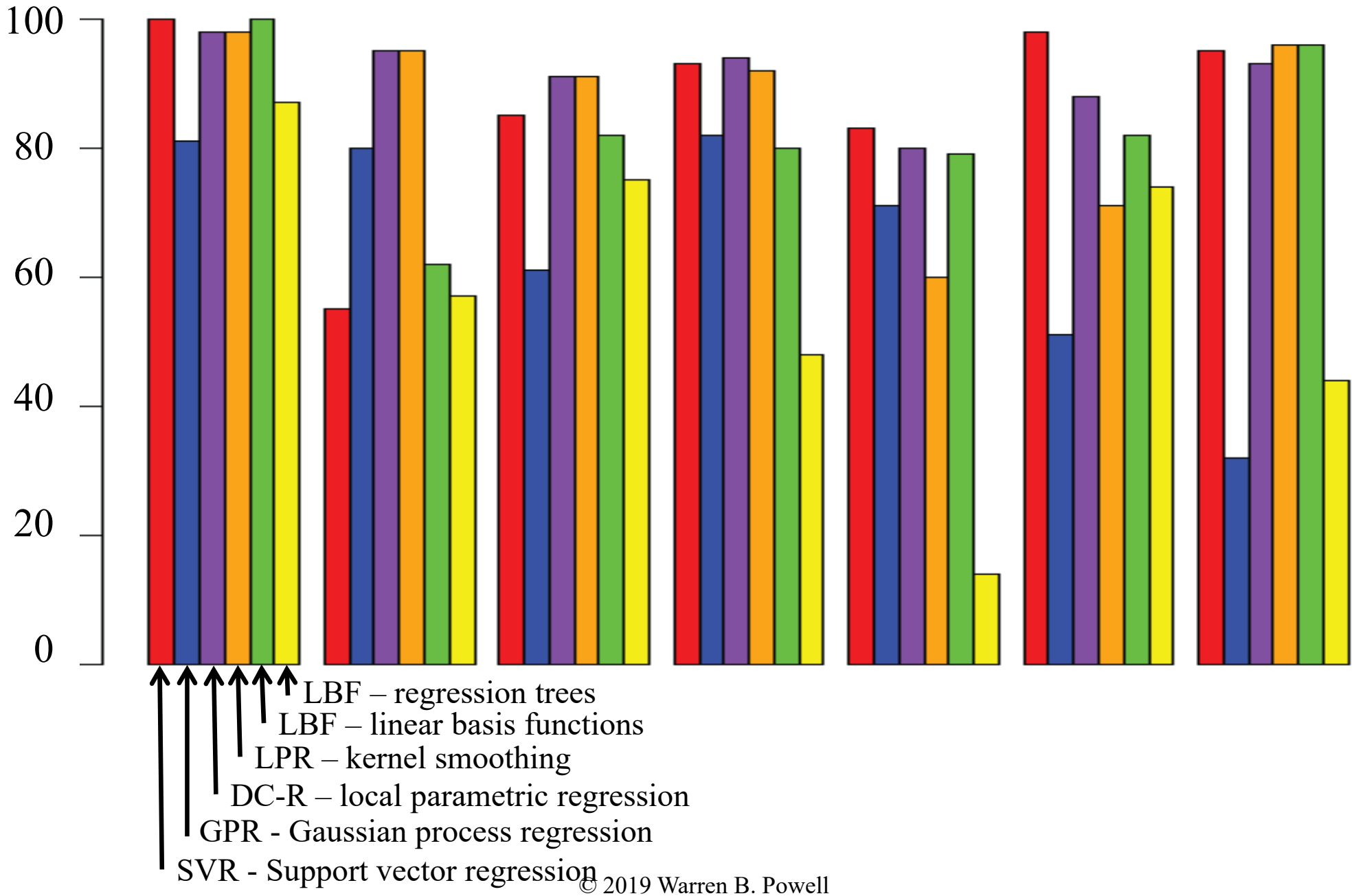
Approximate policy iteration

- Test problem sets

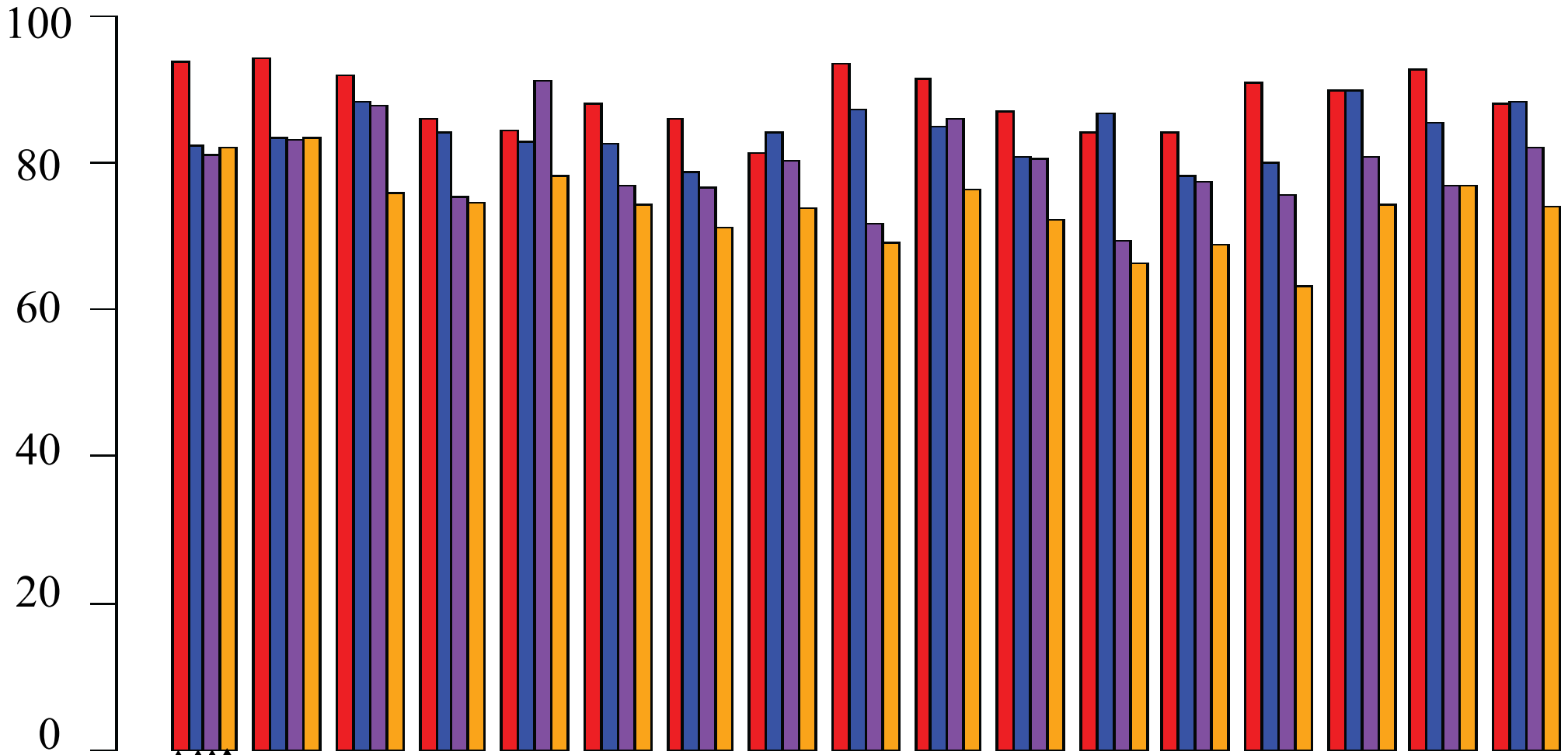
  » Linear Gaussian control
    - P1 = Linear-quadratic regulation
    - Remaining problems (P2-P7) are nonquadratic

  » Finite horizon energy storage problems (Salas benchmark problems)
    - 100 time-period problems
    - Value functions are fitted for each time period

Linear Gaussian control

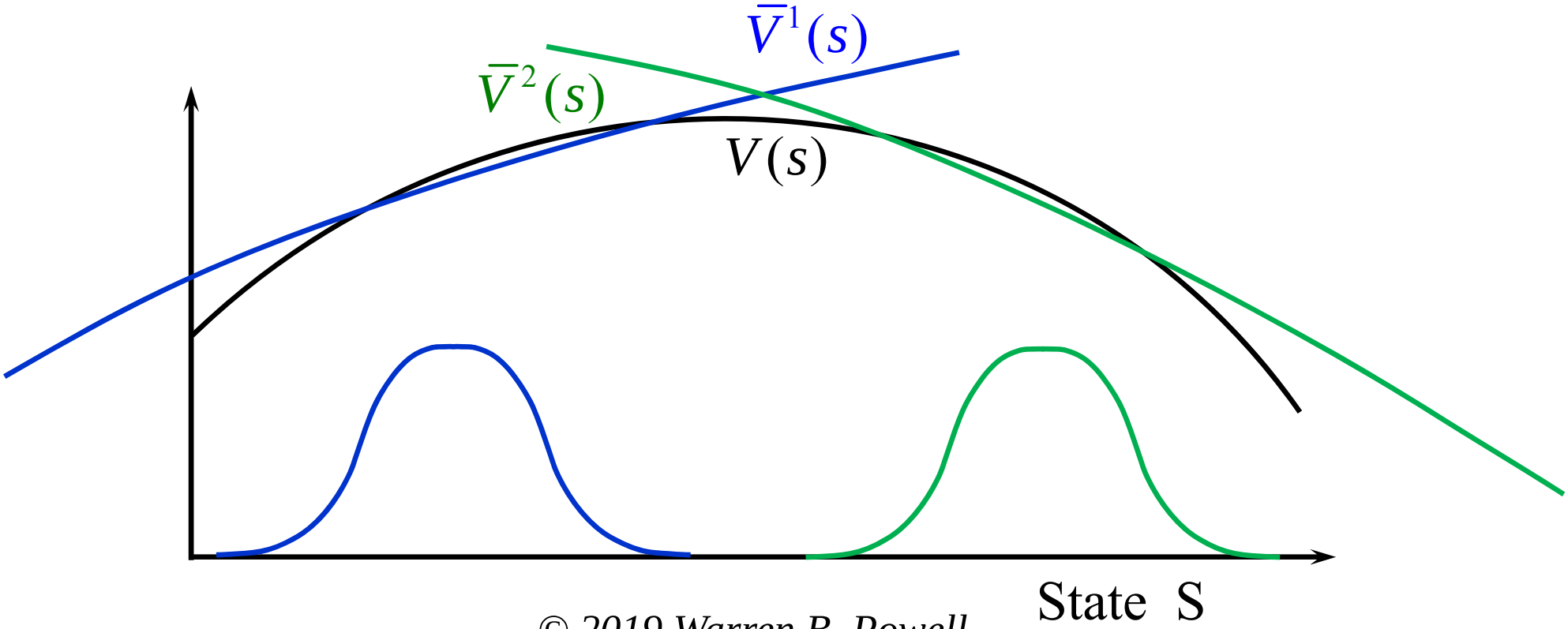


Energy storage applications



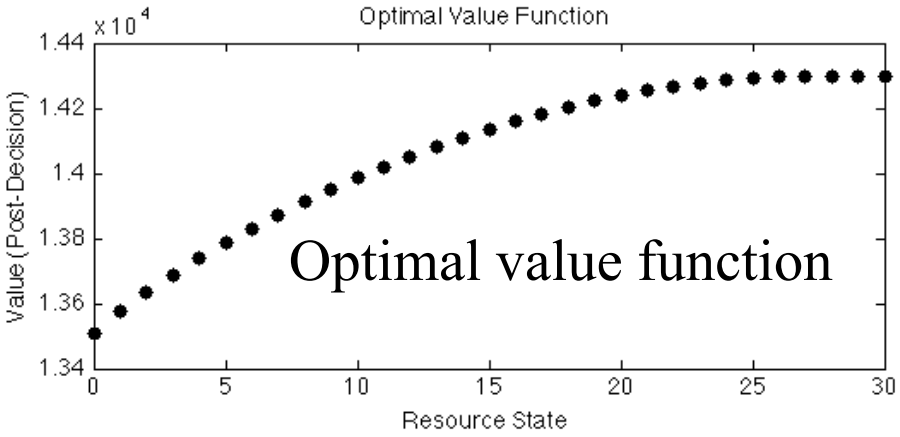
Approximate policy iteration

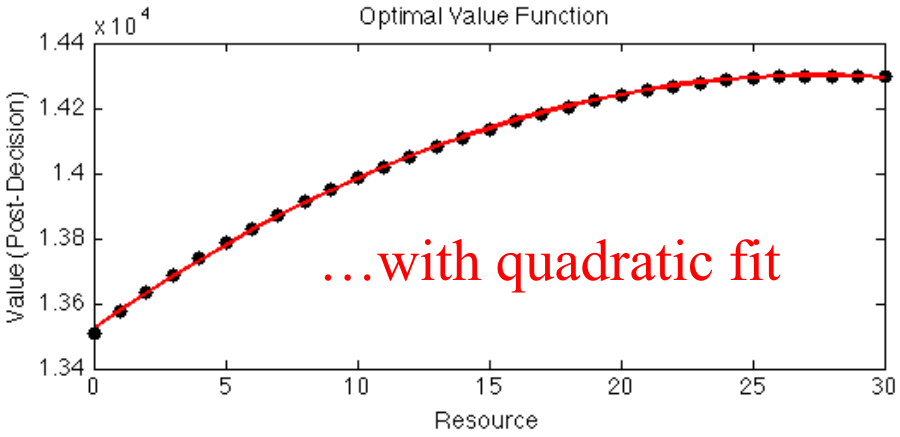
- A tale of two distributions
  - » The *sampling distribution*, which governs the likelihood that we sample a state.
  - » The *learning distribution*, which is the distribution of states we would visit given the current policy

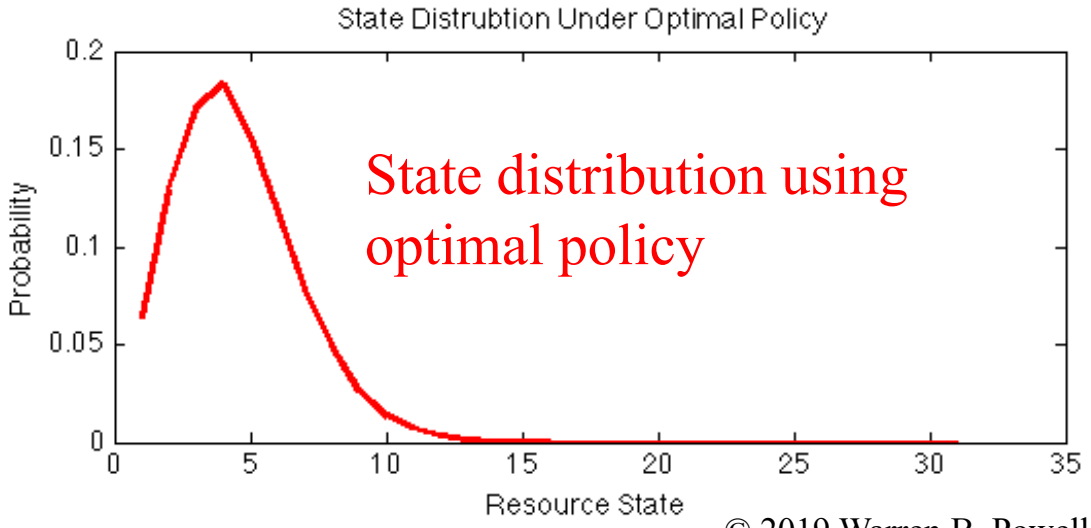


Approximate policy iteration

- Using the optimal value function

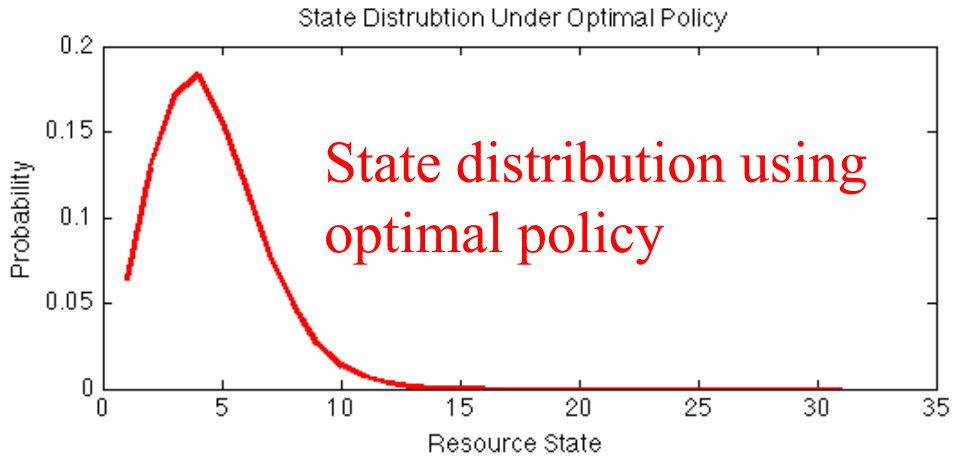




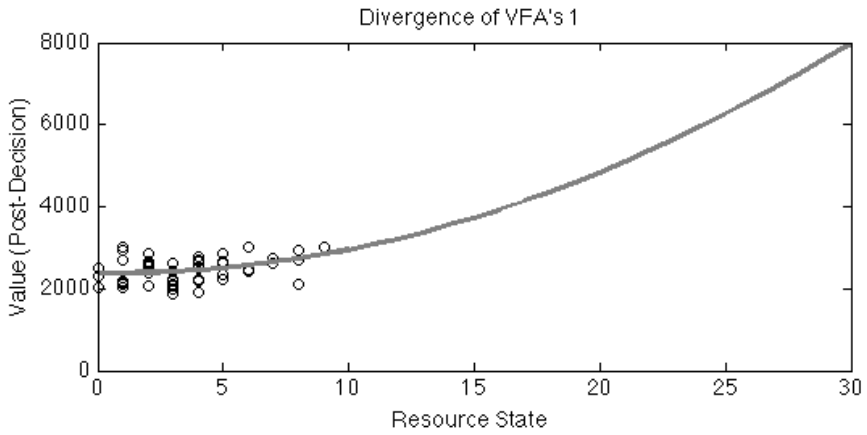


Now we are going to use the optimal policy to fit approximate value functions and watch the stability.

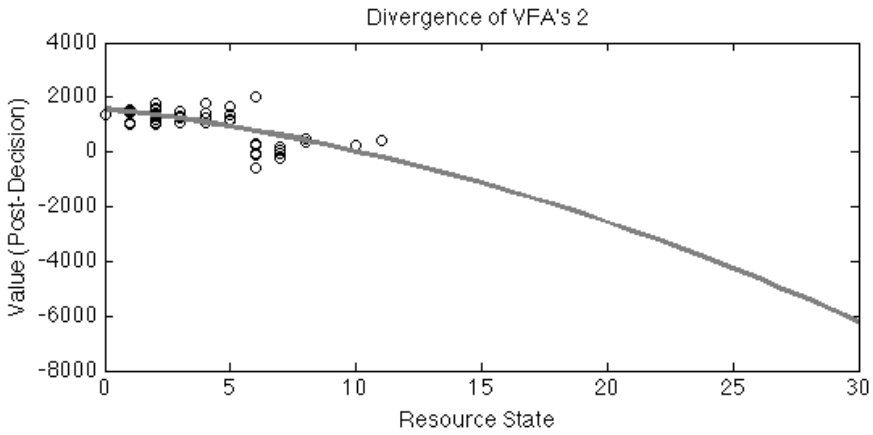
Approximate policy iteration



- Policy evaluation: 500 samples (problem only has 31 states!)
- After 50 policy improvements with **optimal distribution**: divergence in sequence of VFA's, 40%-70% optimality.
- After 50 policy improvements with **uniform distribution**: stable VFAs, 90% optimality.



VFA estimated after 50 policy iterations



VFA estimated after 51 policy iterations

Policy iteration with parametric VFA

- Observations
  - » Experiments using a number of machine learning approximations produced poor results (60-80 percent of optimal)
  - » Policy search using a simple error correction term works surprisingly well (for this problem)
  - » Using low-order polynomial approximations fit using state distributions simulated using the *optimal policy* works poorly!

A Comparison of Approximate Dynamic Programming Techniques on Benchmark Energy Storage Problems: Does Anything Work?

Daniel R. Jiang, Thuy V. Pham, Warren B. Powell, Daniel F. Salas, and Warren R. Scott

Approximate value functions can work very well, but you need structure to guide the learning process. ADP needs benchmarks and careful tuning. WBP

***Abstract*—As** **like solar and** **problem of find** **is becoming in** **problems are often modeled as stochastic dynamic programs, but when the state space becomes large, traditional (exact) techniques such as backward induction, policy iteration, or value iteration quickly become computationally intractable. Approximate dynamic programming (ADP) thus becomes a natural solution technique for solving these problems to near-optimality using significantly fewer computational resources. In this paper, we compare the performance of the following: various approximation architectures used approximate policy iteration (API), approximate value iteration (AVI) with structured lookup table, and direct policy search on an energy storage problem, for which optimal benchmarks exist.**

gh lookup table **at the additional** **extremely effec-** **more advanced**

statistical estimation methods.

This paper reports on the performance of a variety of approximation methods that have been developed in the approximate dynamic programming community, tested using a series of optimal benchmark problems drawn from a relatively simple energy storage application. These suggest that methods based on Bellman error minimization, using both approximate value iteration and approximate policy iteration, work surprisingly poorly if we use approximation methods drawn from machine learning. Pure table lookup also works poorly. By contrast,

"I think you give a too rosy a picture of ADP…."
Andy Barto, in comments on a paper (2009)

"Is the RL glass half full, or half empty?"
Rich Sutton, NIPS workshop, (2014)

Least squares approximate policy iteration

…and comparison to policy search

Least squares API vs. policy search

- Assume we approximate our value function using a linear model:

$$\bar{V}(S) = \sum_{f \in F} \theta_f \phi_f(S) = \phi(S)^T \theta$$

- Using the post-decision state

$$\bar{V}^x(S^x) = \sum_{f \in F} \theta_f \phi_f(S^x) = \phi(S)^T \theta$$

- In steady state, we might write

$$\bar{V}^x(S^x_{t-1}) = \mathbb{E}\left\{\max_x C(S^x, x) + \gamma \bar{V}^x(S^x_t) \mid S^x_t\right\}$$

Least squares API vs. policy search

- Least squares policy iteration (Lagoudakis and Parr)
  - » Bellman's equation (infinite horizon)

$$V^x(S^x_{t-1}) = \mathbb{E}[\max_x\{C(S_t,x) + \gamma V^x(S^x_t)\}|S^x_{t-1}]$$

  - » … is equivalent to (for a fixed policy)

$$\phi(S^x_{t-1})^T\theta = \mathbb{E}[C(S_t, X^\pi(S_t|\theta)) + \gamma\phi(S^x_t)^T\theta|S^x_{t-1}]$$

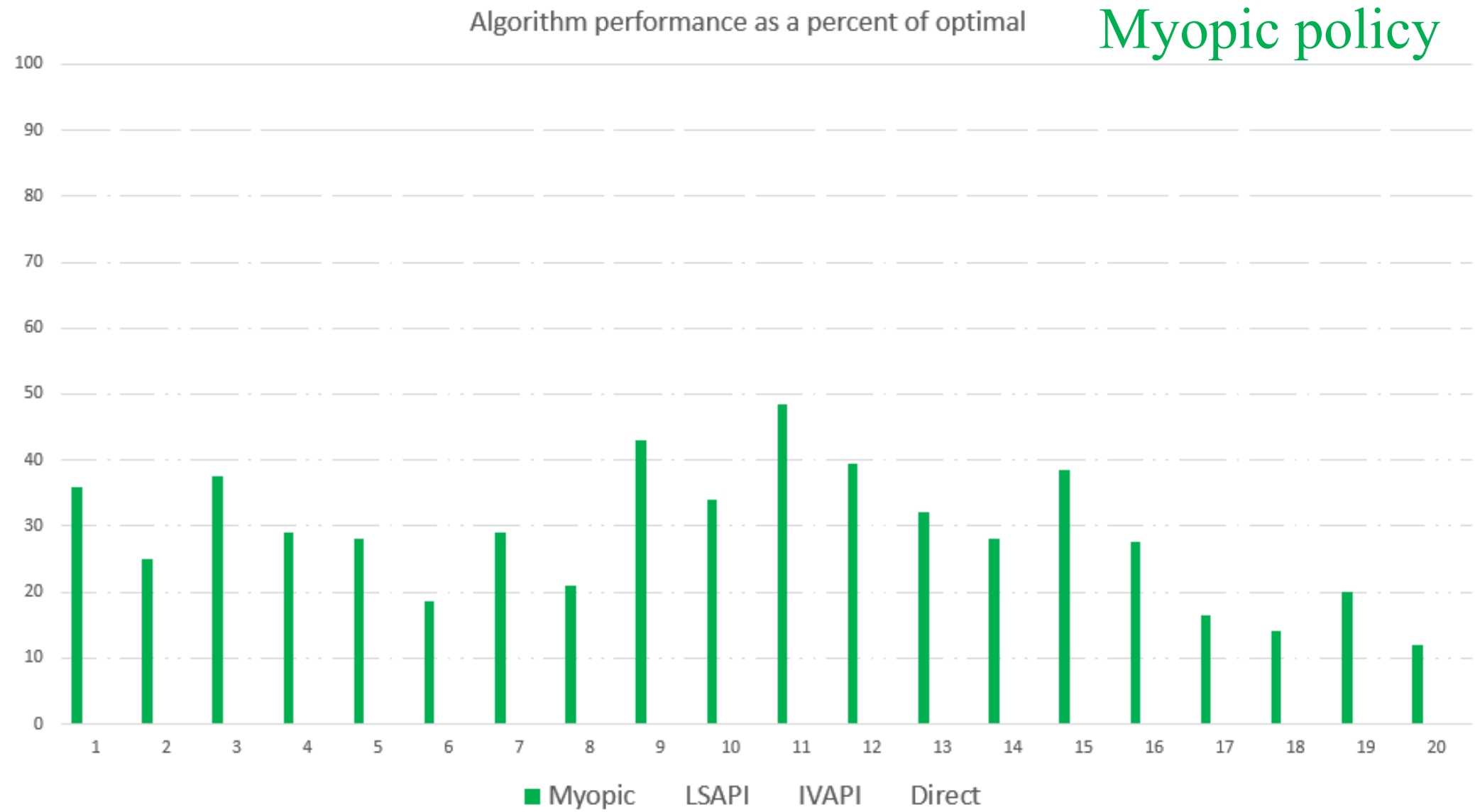
  - » Rearranging gives:

$$\underbrace{C(S_t,X^\pi(S_t|\theta))}_{C_{t,i}} = \underbrace{(\phi(S^x_{t-1}) - \gamma\mathbb{E}[\phi(S^x_t)|S^x_{t-1}])^T}_{X_{t,i}}\theta + \underbrace{C(S_t,X^\pi(S_t|\theta)) - \mathbb{E}[C(S_t,X^\pi(S_t|\theta))|S^x_{t-1}]}_{C_{t,i}-\bar{C}_{t,i}}$$

  - » … where "X" is our explanatory variable. But we cannot compute this exactly (due to the expectation) so we can sample it.

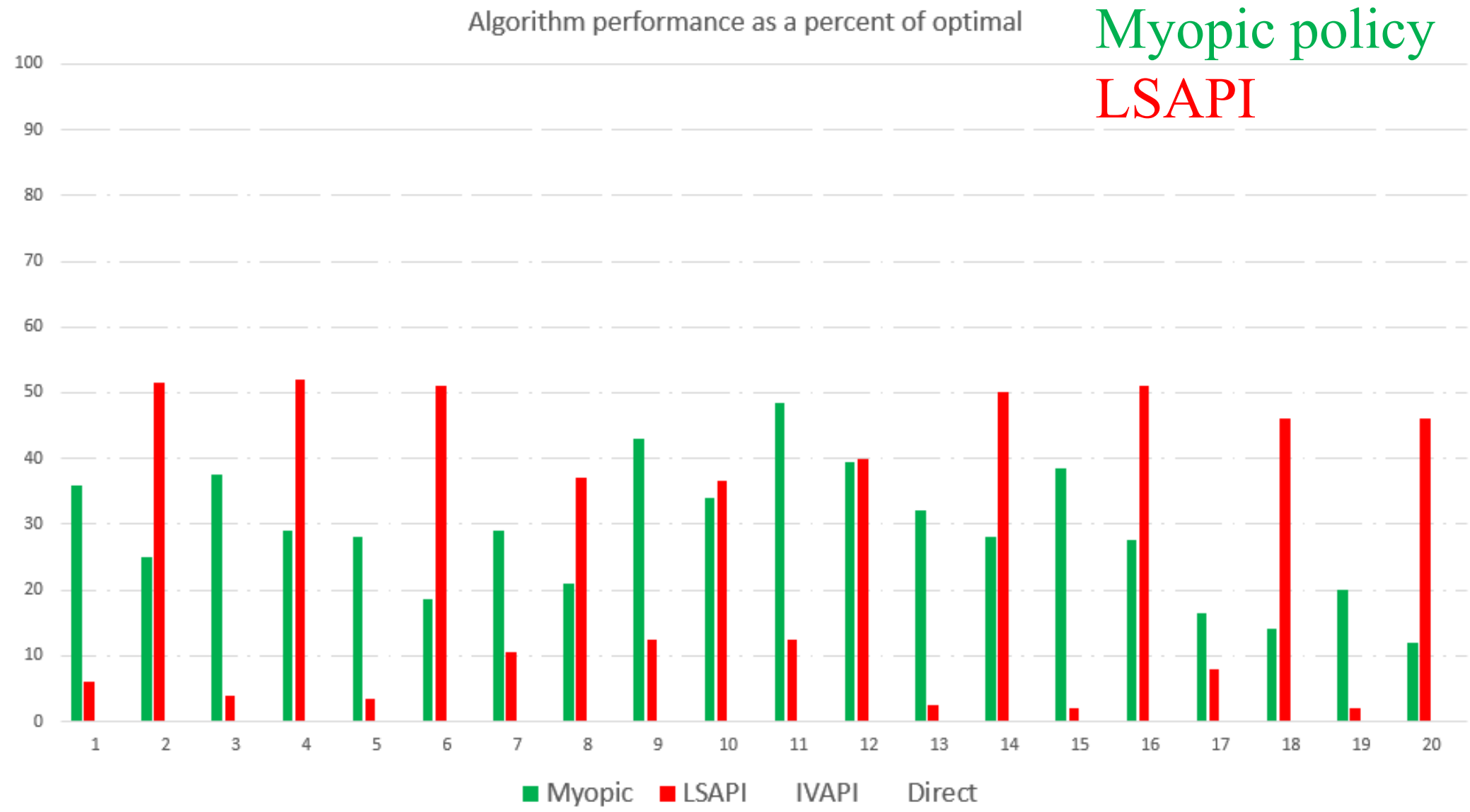
Need to sample introduces "errors in variable" formulation

Least squares API vs. policy search



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Least squares API vs. policy search



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Least squares API vs. policy search

- Variations of LSAPI:

Instrumental Variables Bellman Error Minimization (LSTD)

$$\hat{\theta} = [(\Phi_{t-1})^T(\Phi_{t-1} - \gamma\Phi_t)]^{-1}(\Phi_{t-1})^T C_t,$$

Least-Squares Projected Bellman Error Minimization

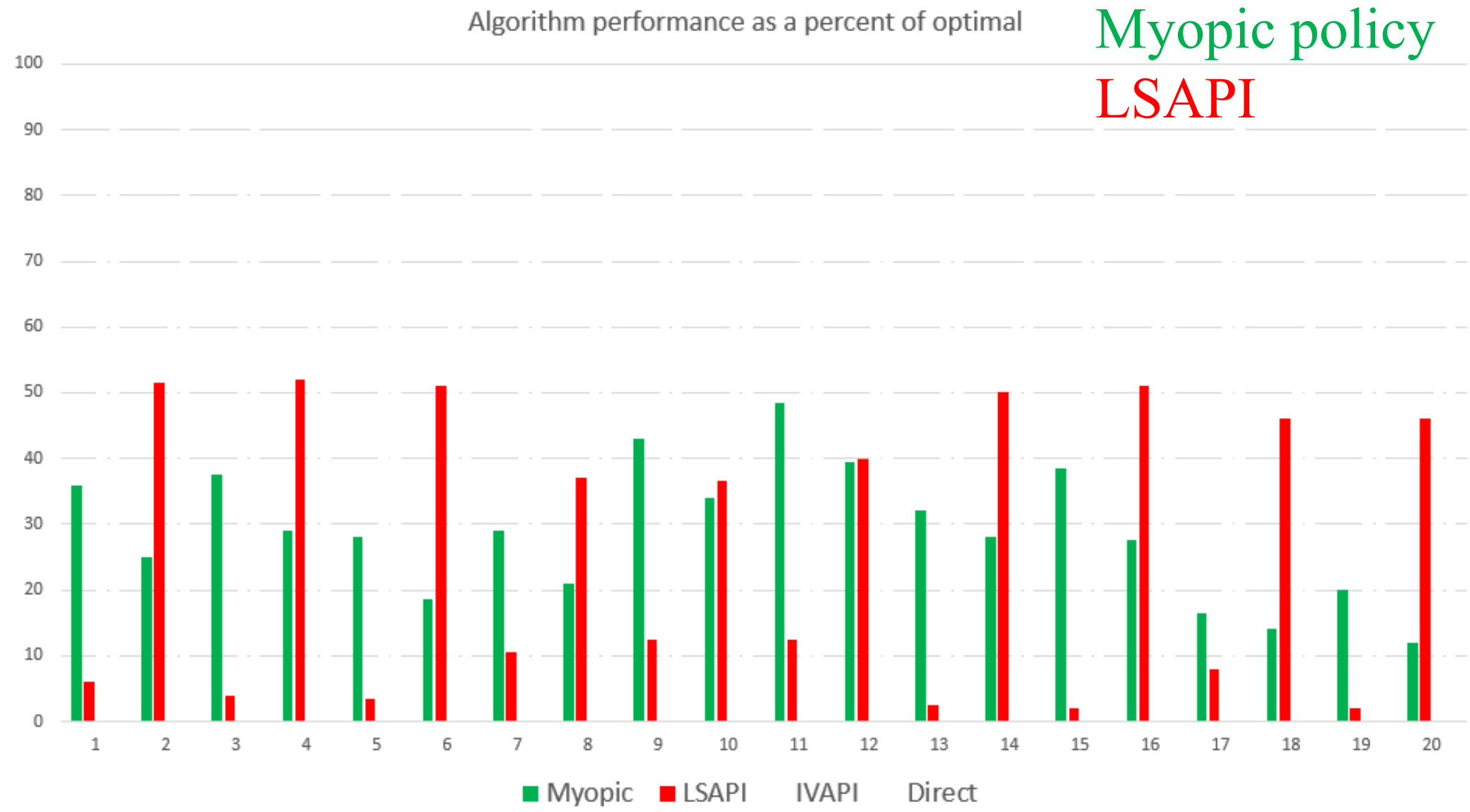
$$\hat{\theta} = \left[(\Pi_{t-1}(\Phi_{t-1} - \gamma\Phi_t))^T(\Pi_{t-1}(\Phi_{t-1} - \gamma\Phi_t))\right]^{-1}(\Pi_{t-1}(\Phi_{t-1} - \gamma\Phi_t))^T\Pi_{t-1}C_t,$$

Instrumental Variables Projected Bellman Error Minimization

$$\hat{\theta} = \left[(\Phi_{t-1})^T\Pi_{t-1}(\Phi_{t-1} - \gamma\Phi_t)\right]^{-1}(\Phi_{t-1})^T\Pi_{t-1}C_t.$$

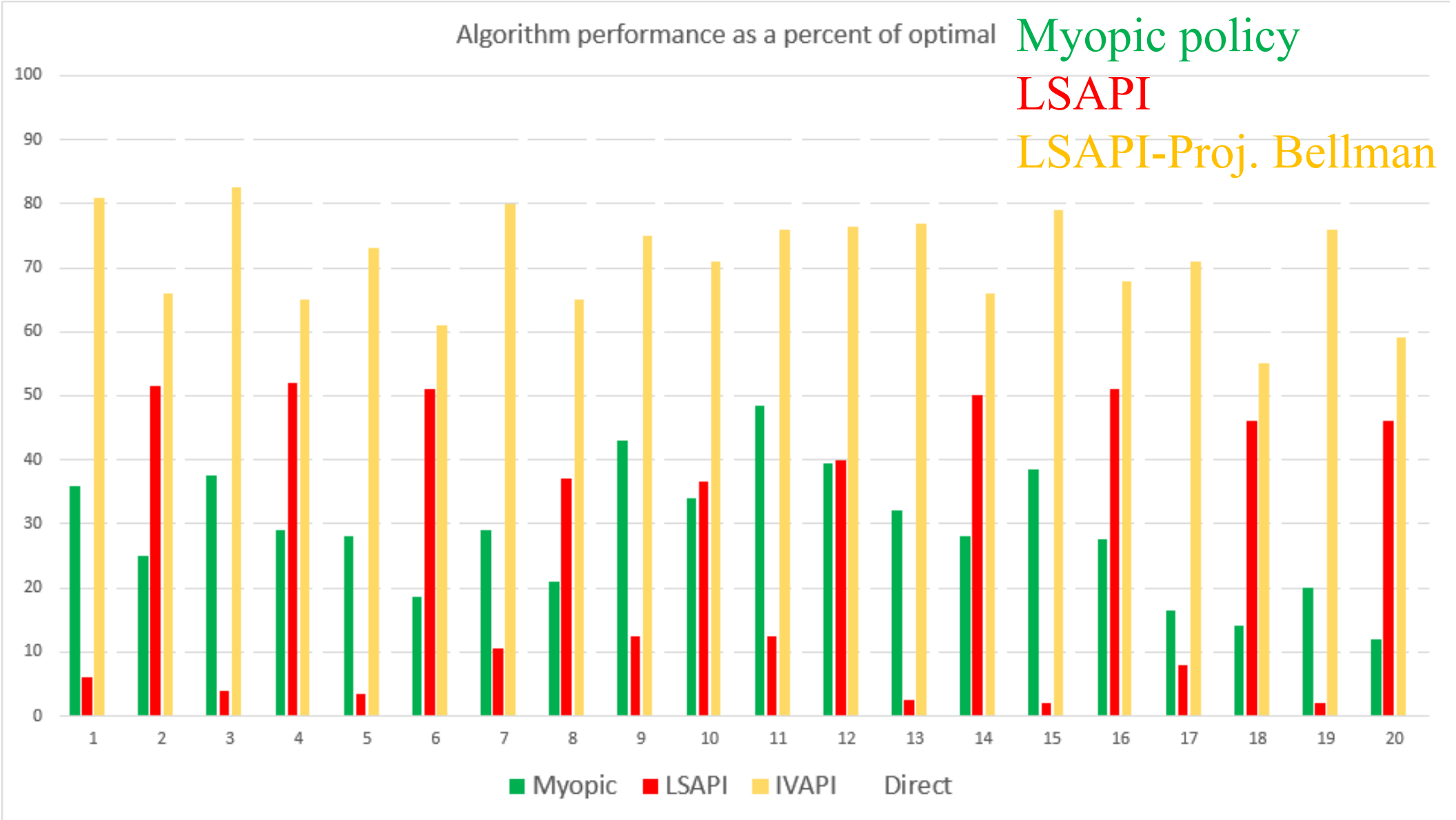
» *Theorem (W. Scott and W.B.P.)* Bellman error using instrumental variables and projected Bellman error minimization *are the same!*

Least squares API vs. policy search



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Least squares API vs. policy search



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

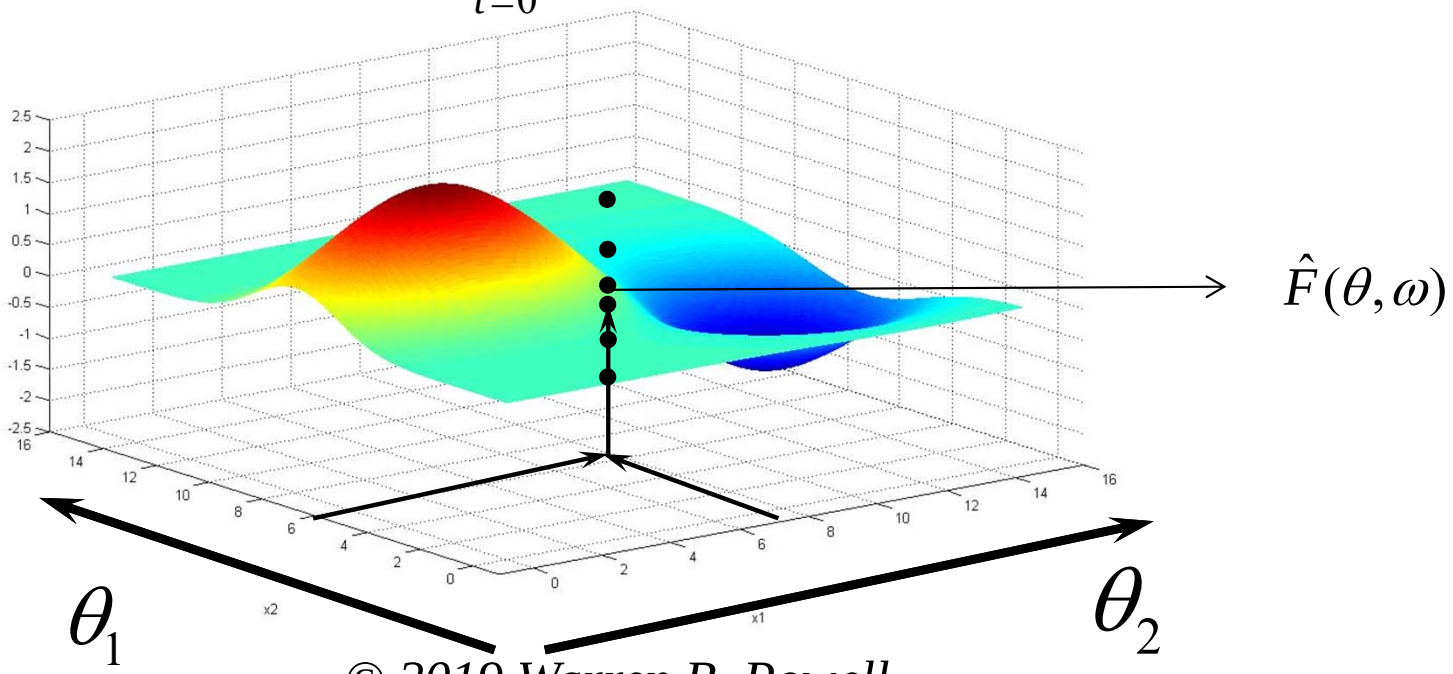
Least squares API vs. policy search

- Finding the best policy ("policy search")
  - » Assume our policy is given by

$$X^{\pi}(S_t \mid \theta) = \arg\max_x \left(C(S_t^n, x_t) + \sum_f \theta_f \phi_f (S_t^x (S_t^n, x_t)) \right)$$

  - » We wish to minimize the function

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



Least squares API vs. policy search

- Two solution methods
  - » Forward approximate dynamic programming
    - Use least squares approximate policy iteration (LSAPI) to solve

$$V^x(S^x_{t-1}) = \mathbb{E}\left\{\min_x \left(C(S_t, x_t) + V^x(S^x_t(S_t, x_t))\right) | S^x_{t-1}\right\}$$

$$\sum_f \theta_f \phi_f(S^x_{t-1}) \approx \mathbb{E}\left\{\min_x \left(C(S_t, x_t) + \sum_f \theta_f \phi_f(S^x_t(S^n_t, x_t))\right) | S^x_{t-1}\right\}$$

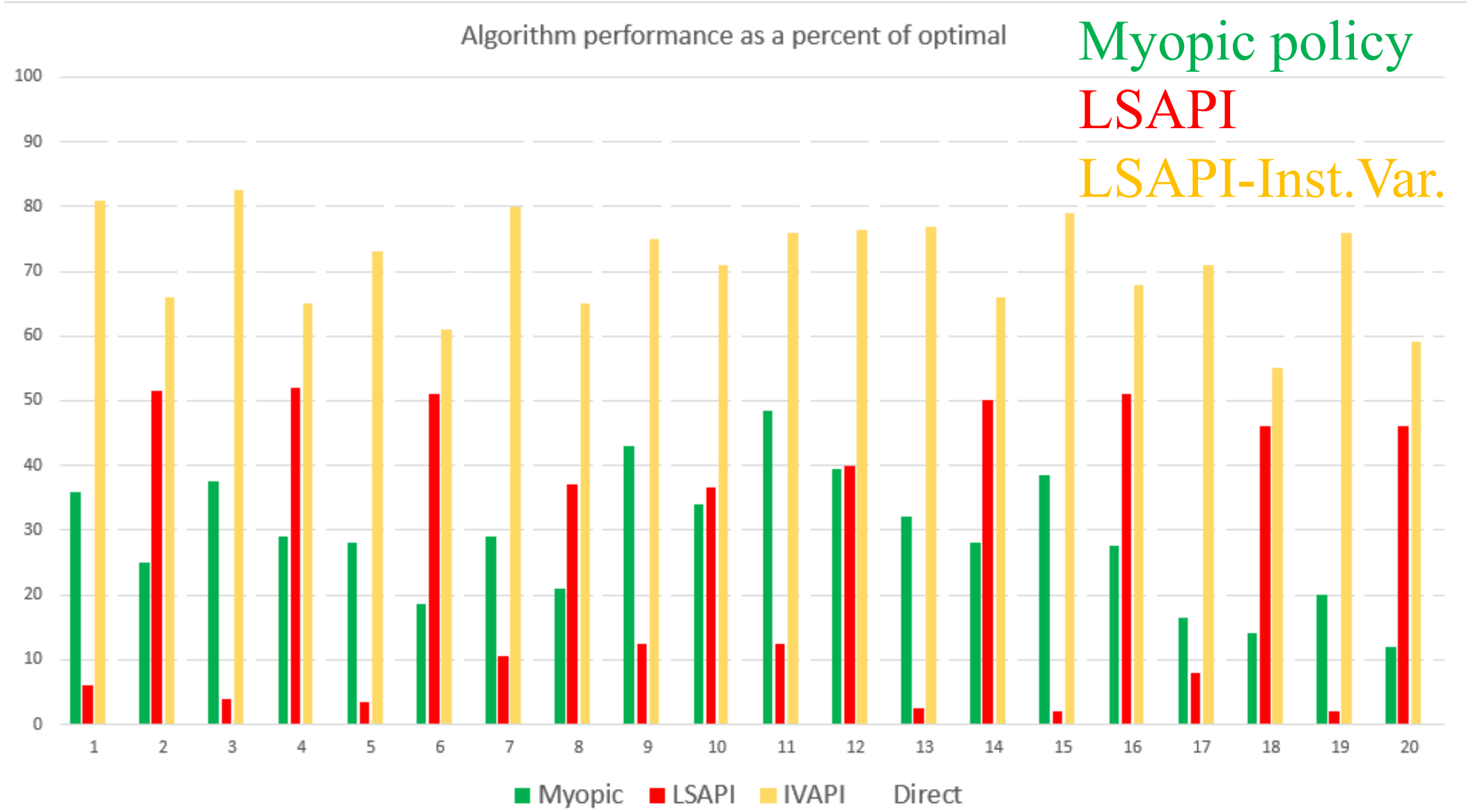
  - » Policy search
    - Find θ to optimize

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

where

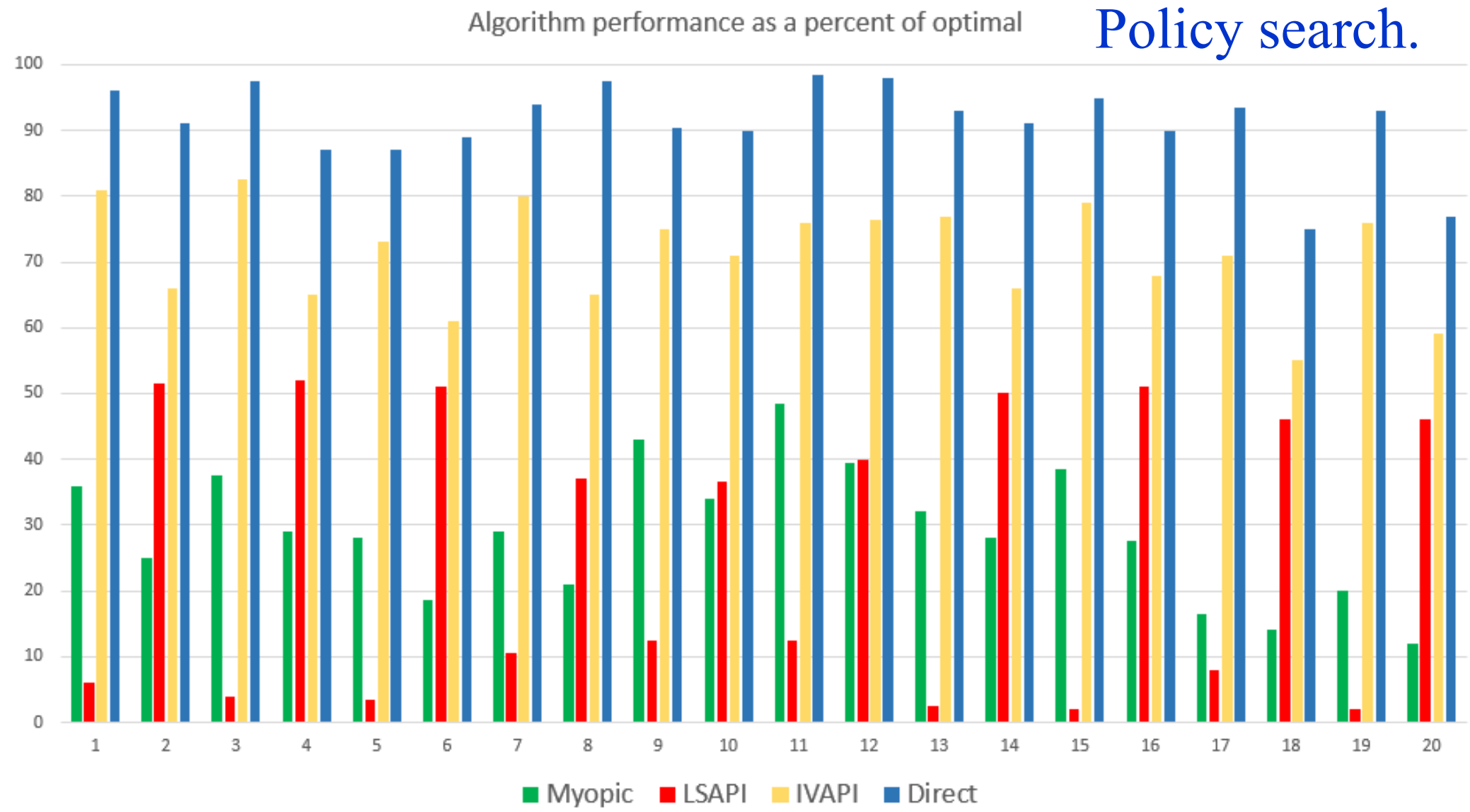
$$X^\pi(S_t | \theta) = \arg\min_x \left(C(S_t, x_t) + \sum_f \theta_f \phi_f(S^x_t(S_t, x_t))\right)$$

Least squares API vs. policy search



For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Least squares API vs. policy search



Policy search.

For benchmark datasets, see: http:www.castlelab.princeton.edu/datasets.htm

Least squares API vs. policy search

- Notes:
  - » Using various parametric approximations for the value function did not work well (but it did work well with the trucking application!)
  - » Using the *same* functions, but fitted using policy search, worked quite well.
  - » We do not know how general this conclusion is. For example, when we can exploit problem structure, value function approximations, fitted using Bellman's equation, can work quite well.
  - » It appears that the value function approximations need to be quite accurate, and not just locally.

Least squares API vs. policy search

- Notes
  - » So why don't we always do policy search?
    - It simply doesn't work all the time.
    - Unable to handle time-dependent policies.
    - The vector θ needs to be fairly low dimensional if we do not have access to derivatives (and derivatives where the CFA term is in the objective can be tricky).
  - » One strategy is to use both:
    - Start by trying to fit θ to approximate the value function. Use this to get an initial value of θ.
    - Then use policy search to tune it to something even better.

Approximate dynamic programming

Stochastic dual dynamic programming

ADP with Benders cuts (SDDP)

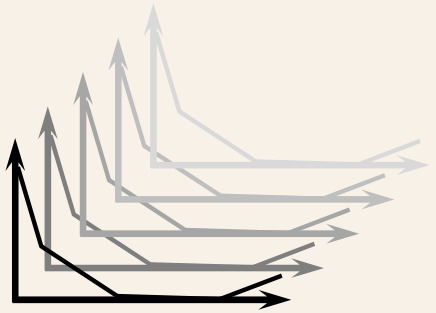
- ADP with Benders decomposition
  - » Mario Pereira (1991) was the first to notice that we could approximate stochastic linear programs (which are convex) using Benders cuts.
  - » SDDP requires two approximations:
    - We are limited to solving a *sampled* model. That is, we are limited to a small set of sample paths.
    - We have to assume that new information arriving at time t+1 is independent of the information at time t.
  - » SDDP is approximate dynamic programming using a two-pass procedure:
    - We simulate forward using the current value function approximation (represented using a set of cuts).
    - We then perform a backward pass to create a new cut for each time period, averaging over the set of samples.

ADP with Benders cuts (SDDP)

ADP with Piecewise Linear Separable VFA

- Assumes separability accross resource dimensions

$$\bar{V}_t(R) := \sum_{m=1}^{|R|} \bar{V}_{t,m}(R_m)$$

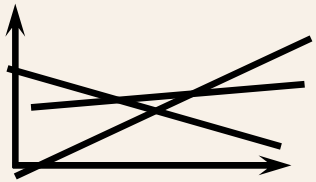


- Lack of optimality bounds.

Stochastic Dual Dynamic Programming - SAA

- Cost-to-go functions are approximated using lower-bounding outer approximations as the maximum over a collection of affine functions.

$$\bar{V}_t^k(R) := \max_{j \le k} \left\{ \alpha_t^j + \langle \beta_t^j, R \rangle \right\}$$



- Suitable for problems with small $|\Omega_t|, t = 1, \ldots, T$.

Financial Engineering &

ADP with Benders cuts (SDDP)

- The forward pass

Forward Pass at Iteration k

1: Sample $\omega \in \Omega$.
2: **for** $t = 0, \ldots, T$ **do**
3: **if** $(k = 0)$ **then**
4:

$$\text{Select } x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \{C(S_t(\omega), x_t)\}$$

5: **else**

Post-decision state variable

6:

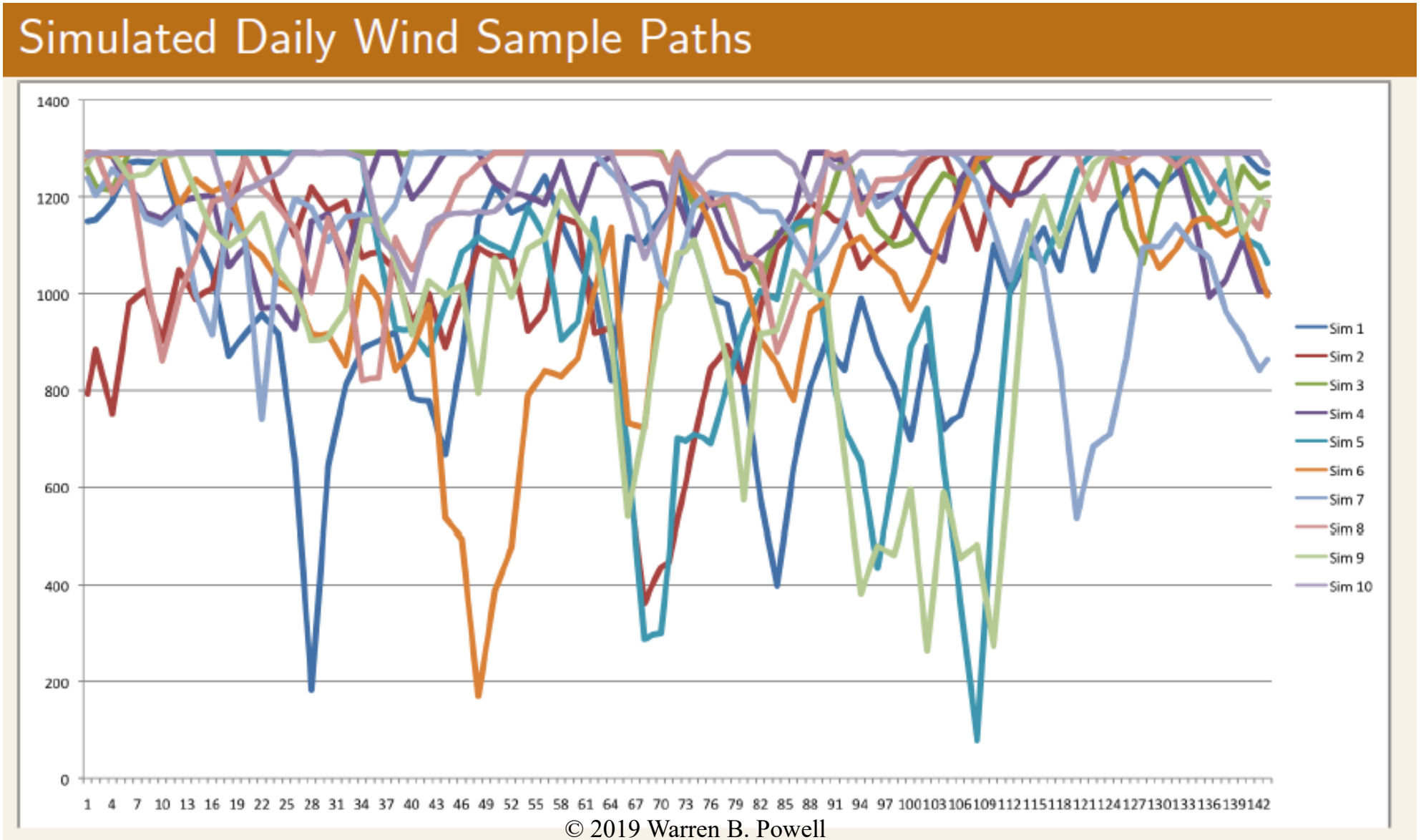
$$\text{Select } x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \left\{C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x)\right\}$$

Post-decision state variable

7: **end if**
8: Set $R_t^{x,k} \leftarrow B_t^k x_t^k$; $S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), I_{t+1}(\omega))$
9: **end for**

ADP with Benders cuts (SDDP)

- The set of sample paths (of wind)



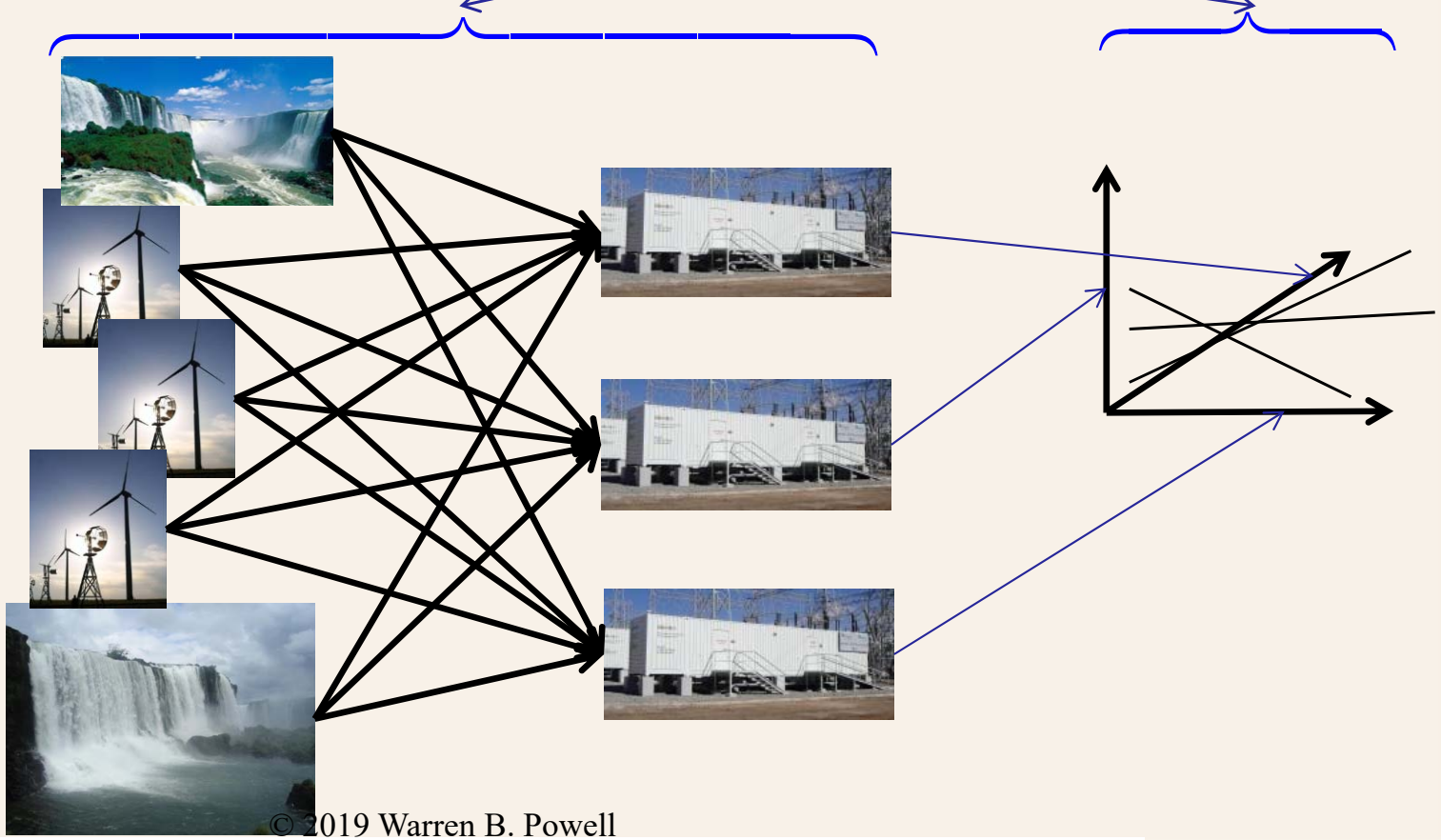
ADP with Benders cuts (SDDP)

Backward Pass at Iteration k

1: **for** $t = T, \ldots, 1$ **do**

2:

Define $\underline{V}_t^k(R_{t-1}^x, \omega_t) := \min_{x_t \in \mathcal{X}_t(R_{t-1}^x, I_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + \overline{V}_t^k(R_t^x) \right\}$



ADP with Benders cuts (SDDP)

Backward Pass at Iteration k

for $t = T, \ldots, 1$ **do**

Define $\underline{V}_t^k(R_{t-1}^x, \omega_t) := \min_{x_t \in \mathcal{X}_t(R_{t-1}^x, I_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + \overline{V}_t^k(R_t^x) \right\}$

  for all $\omega_t \in \Omega_t$ **do**

    Select $\beta_t^k(\omega_t) \in \partial_{R_{t-1}^x} \underline{V}_t^k(R_{t-1}^{x,k}, \omega_t)$

  end for

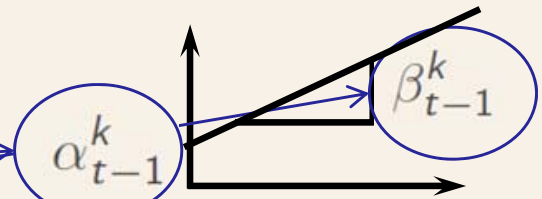
  $\alpha_{t-1}^k \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t) \underline{V}_t^k(R_t^{x,k}, \omega_t)$ $\quad \beta_{t-1}^k \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t) \beta_t^k(\omega_t)$

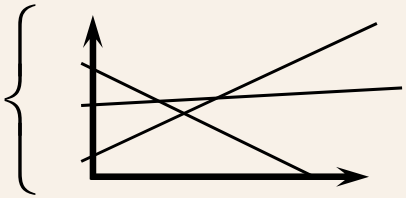
  $h_{t-1}^k(R_{t-1}^x) := \alpha_{t-1}^k + \langle \beta_{t-1}^k, R_{t-1}^x - R_{t-1}^{x,k} \rangle$

  $\overline{V}_{t-1}^k(R_{t-1}^x) := \max \left\{ \overline{V}_{t-1}^{k-1}(R_{t-1}^x),\ h_{t-1}^k(R_{t-1}^x) \right\}$

end for

$\underline{V}_0^k \leftarrow \left\{ \min_{x_0 \in \mathcal{X}_0(S_0)} C(S_0, x_0) + \overline{V}_0^k(R_0^x) \right\}$





ADP with Benders cuts (SDDP)

Backward Pass at Iteration k

for $t = T, \ldots, 1$ **do**

Define $\underline{V}_t^k(R_{t-1}^x, \omega_t) := \min_{x_t \in \mathcal{X}_t(R_{t-1}^x, I_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + \overline{V}_t^k(R_t^x) \right\}$

We need to solve a linear program for each sample path ("scenario") which is why this cannot be very large.

for all $\omega_t \in \Omega_t$ **do**

Select $\underline{\beta}_t^k(\omega_t) \in \partial_{R_{t-1}^x} \underline{V}_t^k(R_{t-1}^{x,k}, \omega_t)$

end for

$\alpha_{t-1}^k \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t) \underline{V}_t^k(R_t^{x,k}, \omega_t)$; $\beta_{t-1}^k \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t) \underline{\beta}_t^k(\omega_t)$

$h_{t-1}^k(R_{t-1}^x) := \alpha_{t-1}^k + \langle \beta_{t-1}^k, R_{t-1}^x - R_{t-1}^{x,k} \rangle$

$\overline{V}_{t-1}^k(R_{t-1}^x) := \max \left\{ \overline{V}_{t-1}^{k-1}(R_{t-1}^x),\ h_{t-1}^k(R_{t-1}^x) \right\}$

We then need to average over all of these linear programs.

end for

$\underline{V}_0^k \leftarrow \left\{ \min_{x_0 \in \mathcal{X}_0(S_0)} C(S_0, x_0) + \overline{V}_0^k(R_0^x) \right\}$

ADP with Benders cuts (SDDP)

- Regularization
  - » In the 1990's, Ruszczynski showed that Benders works much better if you introduce a regularization term, which penalizes deviations from the current solution.
  - » This powerful idea (now widely used in statistics) has not been generalized in a practical way to multiperiod ("multistage") problems.
  - » In recent work, Tsvetan Asamov introduced a practical way to do regularization for multiperiod problems (with hundreds of time periods). A key feature is that the regularization term does not depend on history.
  - » The method is *very* easy to implement.

ADP with Benders cuts (SDDP)

Forward Pass with Regularization

Sample $\omega \in \Omega$.
for $t = 0, \ldots, T$ **do**
  if $(k = 0)$ **then**
    Select $x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \{C(S_t(\omega), x_t)\}$
  else
    if $t < T$ **then**
      Select $x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \left\{ C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x) + \frac{\varrho^k}{2} \left\langle R_t^x - \overline{R}_t^{x,k-1}, Q_t(R_t^x - \overline{R}_t^{x,k-1}) \right\rangle \right\}$
    else
      Select $x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \left\{ C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x) \right\}$
    end if
  end if
  Set $R_t^{x,k} \leftarrow B_t^k x_t^k$; $S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), I_{t+1}(\omega))$
end for

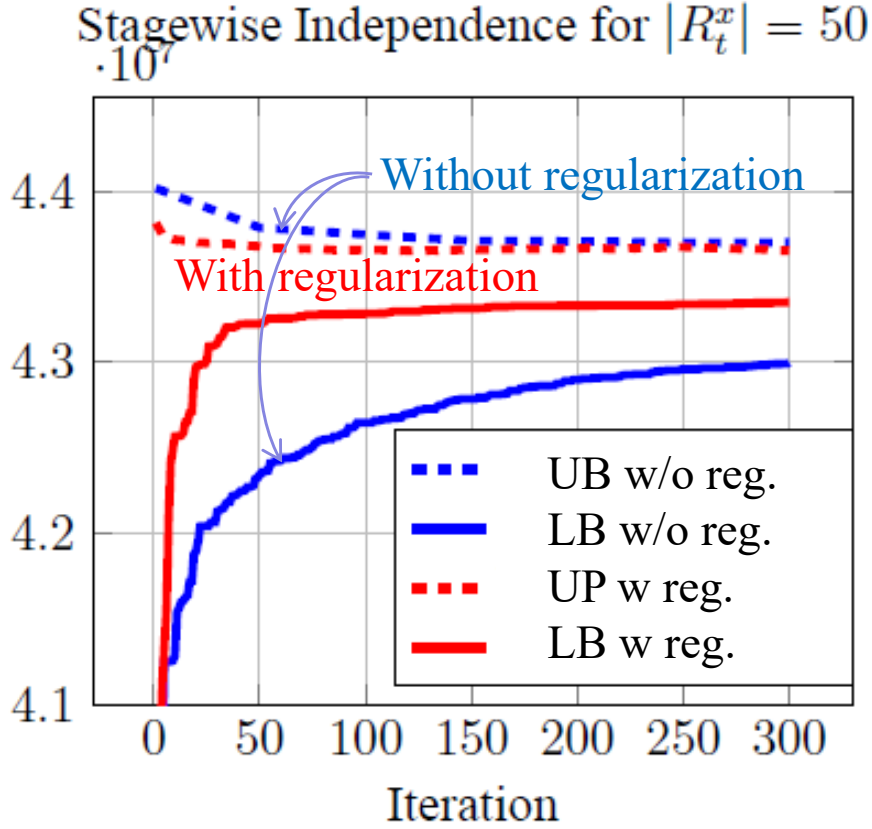
We penalize deviations from the previous resource state. This is not indexed by the sample path.

We simply retain the previous resource state.

ADP with Benders cuts (SDDP)

- SDDP with regularization

  - » Regularization slightly improves the upper bound, but dramatically improves the lower bound.
  - » There is minimal additional computational cost to introduce regularization for multiperiod problems.



ADP with Benders cuts (SDDP)

- Comparisons on a deterministic grid problem
  - » SPWL – Separable, piecewise linear approximations
  - » SDDP – Benders cuts
  - » 25 batteries

Slightly faster convergence using SPWL VFAs.

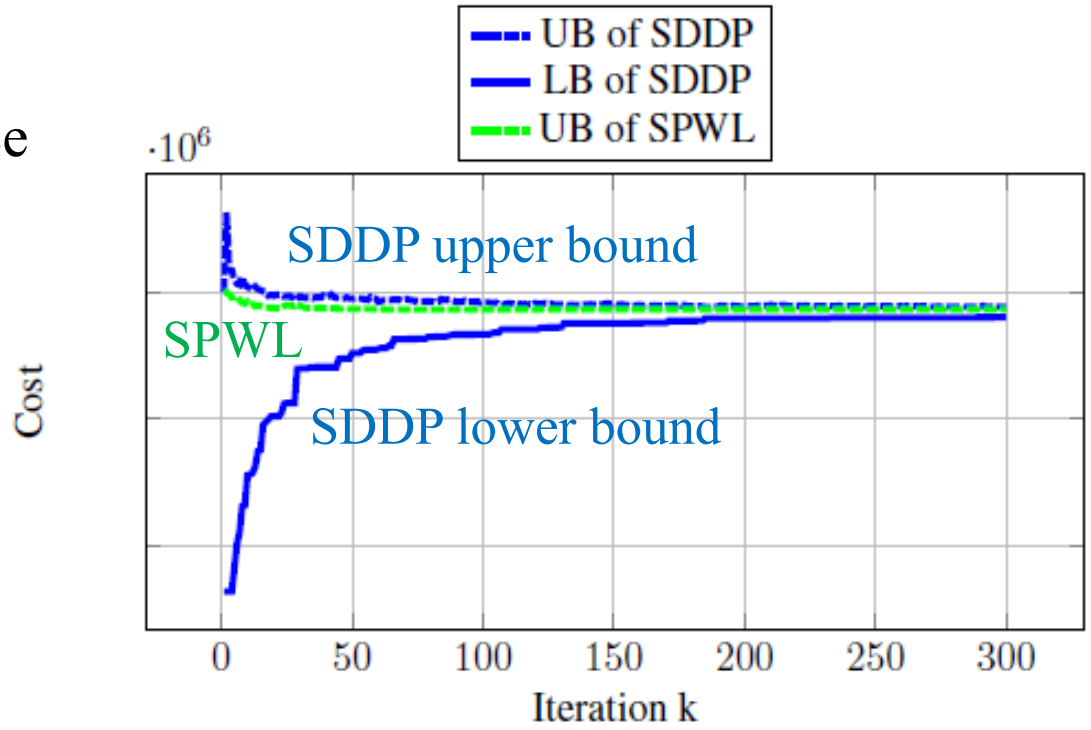


Figure 5: Comparison of multistage stochastic optimization methods for $|R| = 25$

ADP with Benders cuts (SDDP)

- Comparisons on a deterministic grid problem
  - » SPWL – Separable, piecewise linear approximations
  - » SDDP – Benders cuts
  - » 50 batteries

Much faster convergence using SPWL VFAs.

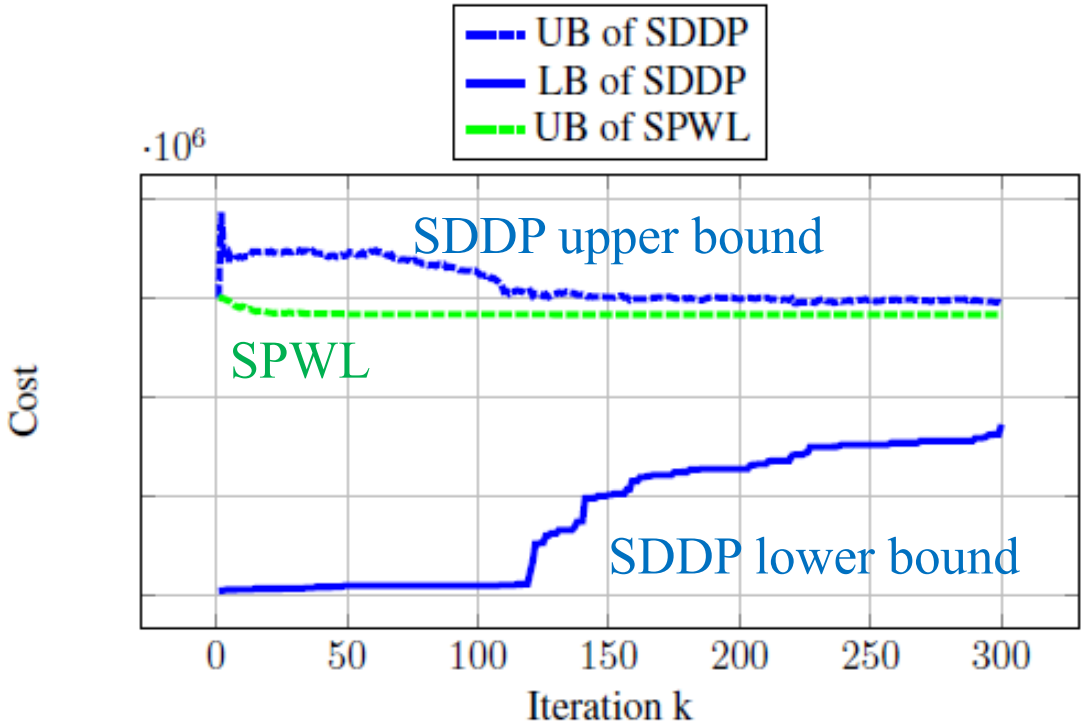


Figure 8: Comparison of multistage stochastic optimization methods for $|R| = 50$

ADP with Benders cuts (SDDP)

- Comparisons on a stochastic grid problem
  - » SPWL – Separable, piecewise linear approximations
  - » SDDP – Benders cuts
  - » 25 batteries

SPWL VFAs still exhibit slightly faster convergence.

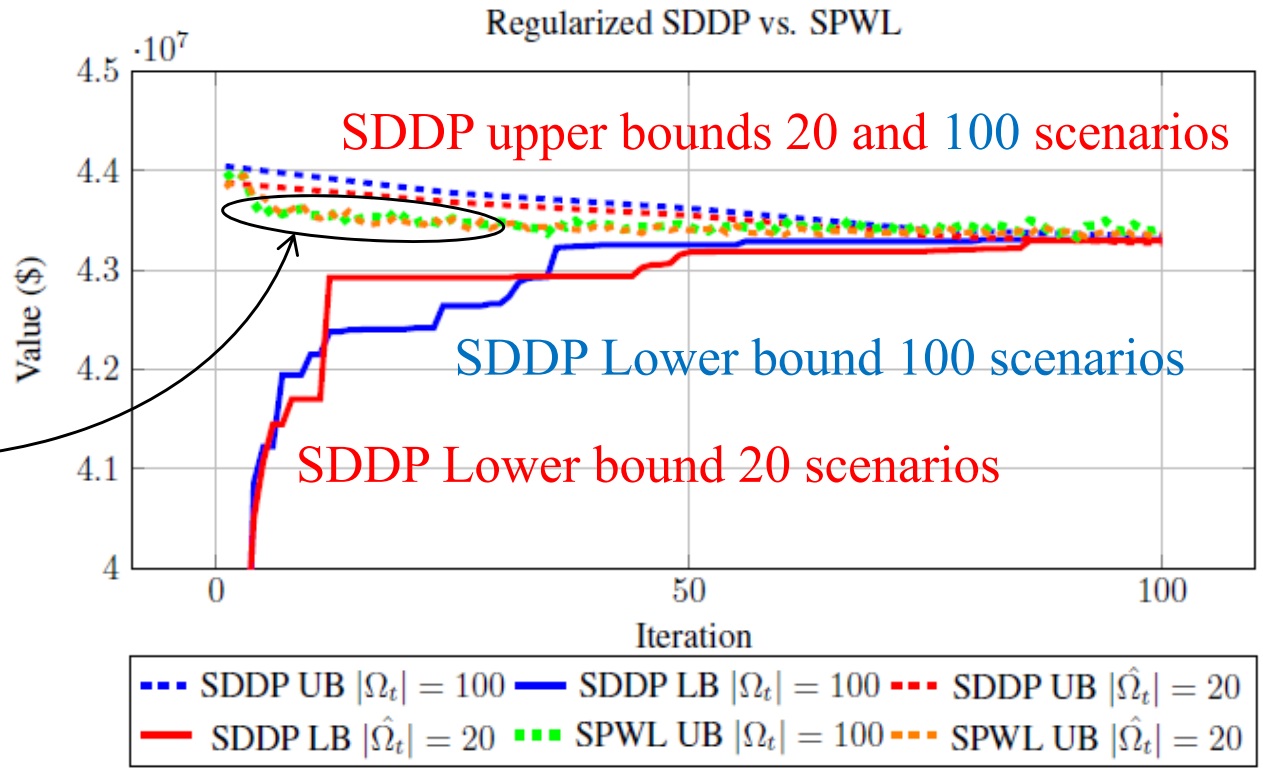


Figure 11: Numerical comparison of multistage stochastic optimization methods for $|R_t^x| = 25$

ADP with Benders cuts (SDDP)

- Comparisons on a stochastic grid problem
  - » SPWL – Separable, piecewise linear approximations
  - » SDDP – Benders cuts
  - » 50 batteries

SPWL VFAs still exhibit slightly faster convergence.

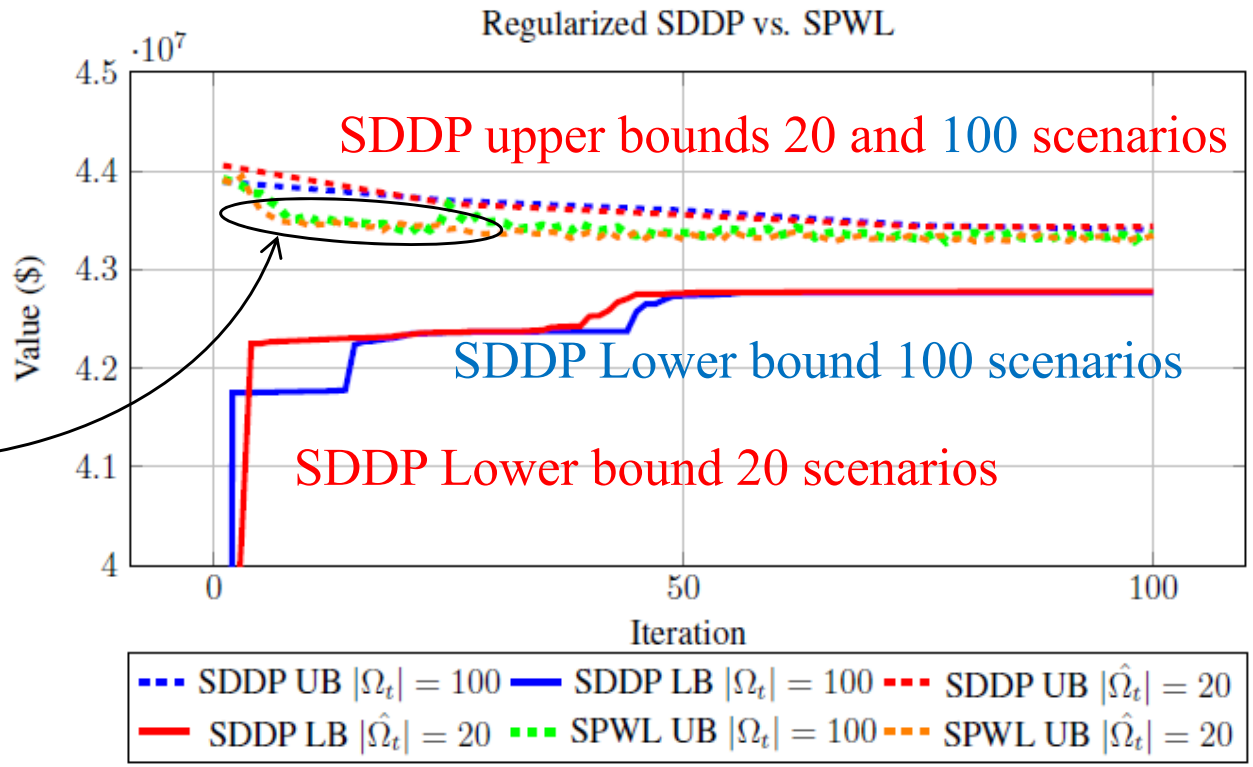


Figure 12: Numerical comparison of multistage stochastic optimization methods for $|R_t^x| = 50$

ADP with Benders cuts (SDDP)

- Comparisons on a stochastic grid problem
  - » SPWL – Separable, piecewise linear approximations
  - » SDDP – Benders cuts
  - » 100 batteries

Performance is very similar, despite high dimensionality (100 batteries)

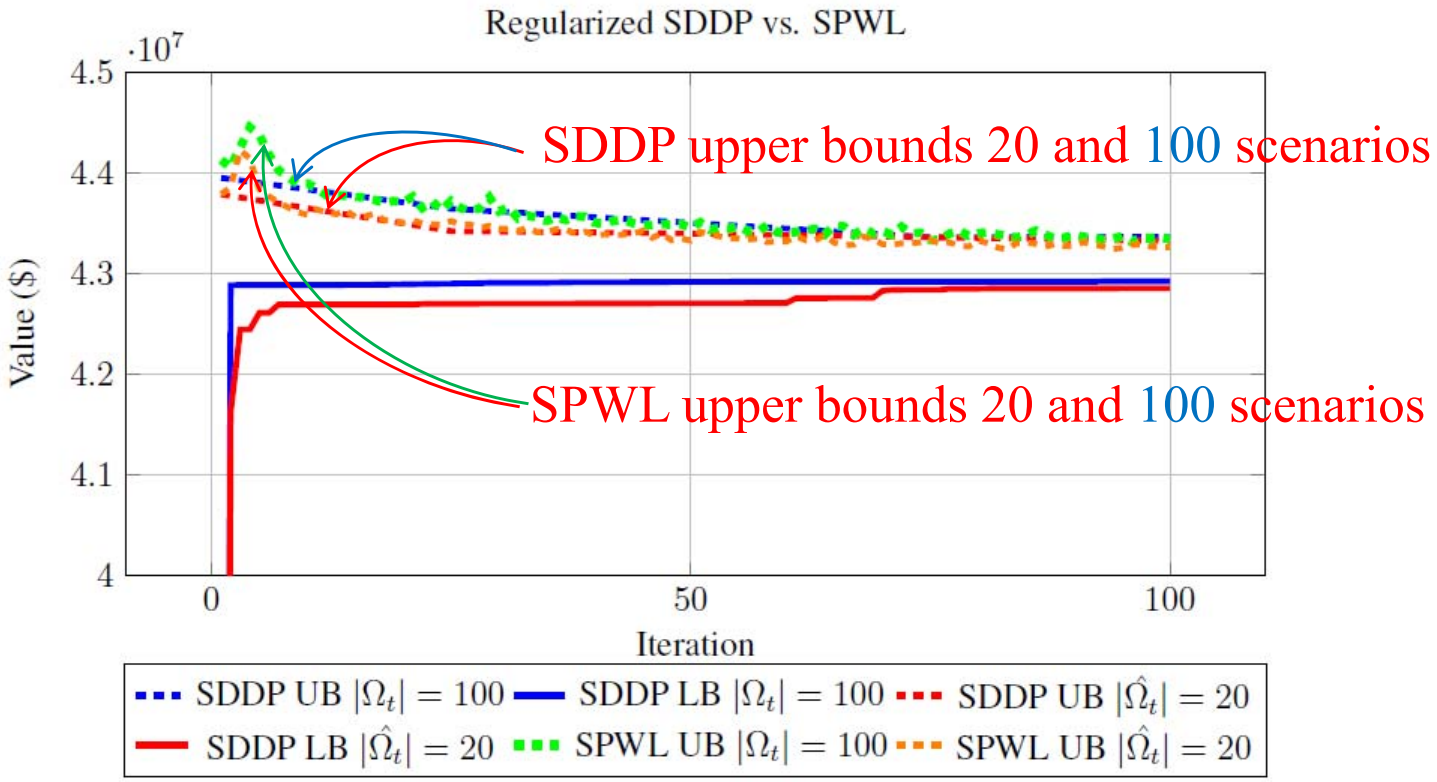


Figure 13: Numerical comparison of multistage stochastic optimization methods for $|R_t^x| = 100$

Problem classes

	Offline Terminal reward	Online Cumulative reward
State independent problems	$\max_\pi \mathbb{E}\{F(x^{\pi,N}, W)\|S_0\}$ Stochastic search (1)	$\max_\pi \mathbb{E}\{\sum_{n=0}^{N-1} F(X^\pi(S^n), W^{n+1})\|S_0\}$ Multiarmed bandit problem (2)
State dependent problems	$\max_{\pi^{lrn}} \mathbb{E}\{C(S, X^{\pi^{impl}}(S\|\theta^{imp}), W)\|S_0\}$ Offline dynamic programming (4)	$\max_\pi \mathbb{E}\{\sum_{t=0}^{T} C(S_t, X^\pi(S_t), W_{t+1})\|S_0\}$ Online dynamic programming (3)

Learning policies ("algorithms"):

- Approximate dynamic programming
- Q-learning
- SDDP (see next slide)

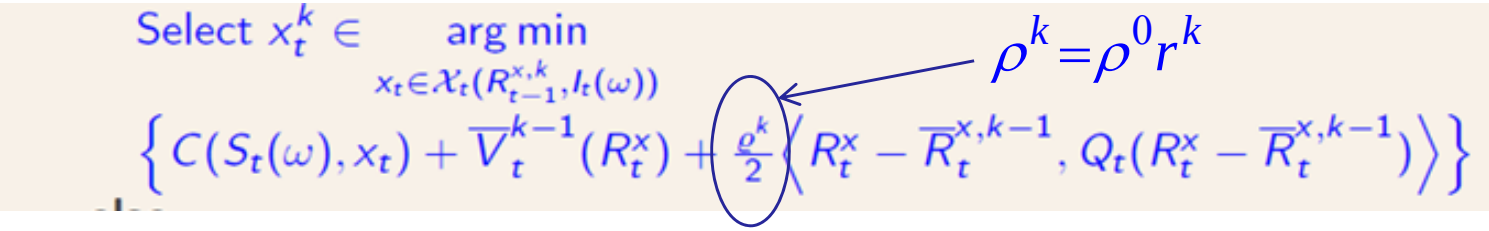
ADP with Benders cuts (SDDP)

Forward Pass with Regularization

Sample $\omega \in \Omega$.
for $t = 0, \ldots, T$ **do**
  if $(k = 0)$ **then**
    Select $x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \{C(S_t(\omega), x_t)\}$
  else
    if $t < T$ **then**
      Select $x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min}$
      $$\left\{ C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x) + \frac{\varrho^k}{2} \left\langle R_t^x - \overline{R}_t^{x,k-1}, Q_t(R_t^x - \overline{R}_t^{x,k-1}) \right\rangle \right\}$$
    else
      $$\text{Select } x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \left\{ C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x) \right\}$$
    end if
  end if
  Set $R_t^{x,k} \leftarrow B_t^k x_t^k$; $S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), I_{t+1}(\omega))$
end for

ADP with Benders cuts (SDDP)

- Notes:
  - » SDDP is a form of *learning policy*. This is a method for learning the cuts of the value function.
  - » It is parameterized by the regularization term



  - » So our learning policy is determined by:
    - The SDDP structure.
    - The parameters of the regularization term $\theta = (\rho^0, r)$
    - The graphs above represent optimization of the learning policy.
  - » Then we have to test the cuts as an implementation policy, and compare to other VFA approximations.

ADP with Benders cuts (SDDP)

- The forward pass

Forward Pass at Iteration k

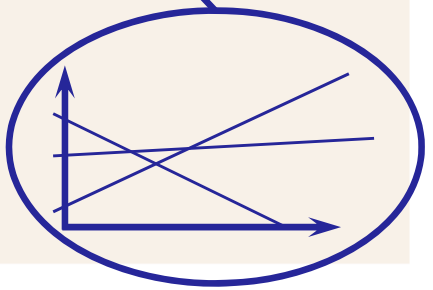
1: Sample $\omega \in \Omega$.
2: **for** $t = 0, \ldots, T$ **do**
3: **if** $(k = 0)$ **then**
4:

$$\text{Select } x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \{C(S_t(\omega), x_t)\}$$

5: **else** The value function approximation defines the implementation policy!
6:

$$\text{Select } x_t^k \in \underset{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, I_t(\omega))}{\arg\min} \left\{C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x)\right\}$$

7: **end if**
8: Set $R_t^{x,k} \leftarrow B_t^k x_t^k$; $S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), I_{t+1}(\omega))$
9: **end for**



ADP with Benders cuts (SDDP)

- More complex information processes
  - » We can easily approximate convex functions (using Benders or separable, piecewise linear value functions).
  - » But what if we have an external information process that affects the system?
    - Weather
    - Economy
  - » In the next slide, we show how the algorithm changes when we no longer enjoy "intertemporal independence."
  - » Assume that the information state is "weather" that may be sunny, cloudy or stormy.

ADP with Benders cuts (SDDP)

for $t = T, \ldots, 1$ **do**

Define $\underline{V}_t^k(R_{t-1}^x, \omega_t) := \min_{x_t \in \mathcal{X}_t(R_{t-1}^x, I_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + \overline{V}_t^k(R_t^x, I_t^x(\omega_t)) \right\}$

  for all $\omega_t \in \Omega_t$ **do**

    Select $\underline{\beta}_t^k(\omega_t) \in \partial_{R_{t-1}^x} \underline{V}_t^k(R_{t-1}^{x,k}, \omega_t)$

  end for

  for all $I_{t-1}^x \in \mathcal{I}_{t-1}^x(\Omega_{t-1})$ **do**

    $\alpha_{t-1}^k(I_{t-1}^x) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | I_{t-1}^x) \underline{V}_t^k(R_t^{x,k}, \omega_t);$

    $\beta_{t-1}^k(I_{t-1}^x) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | I_{t-1}^x) \underline{\beta}_t^k(\omega_t)$

    $h_{t-1}^k(R_{t-1}^x, I_{t-1}^x) := \alpha_{t-1}^k(I_{t-1}^x) + \langle \beta_{t-1}^k(I_{t-1}^x), R_{t-1}^x - R_{t-1}^{x,k} \rangle$

    $\overline{V}_{t-1}^k(R_{t-1}^x, I_{t-1}^x) := \max \left\{ \overline{V}_{t-1}^{k-1}(R_{t-1}^x, I_{t-1}^x),\ h_{t-1}^k(R_{t-1}^x, I_{t-1}^x) \right\}$

  end for

end for

$\overline{R}_t^{x,k} \leftarrow R_t^{x,k},\ t = 0, \ldots, T-1;\ \underline{V}_0^k \leftarrow \left\{ \min_{x_0 \in \mathcal{X}_0(S_0)} C(S_0, x_0) + \overline{V}_0^k(R_0^x) \right\}$

Weather states

ADP with Benders cuts (SDDP)

- More complex information processes (cont'd)
  - » In this algorithm, we are using a lookup table function for weather:

$$\alpha^k_{t-1}(I^x_{t-1}) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | I^x_{t-1}) \underline{V}^k_t(R^{x,k}_t, \omega_t);$$

$$\beta^k_{t-1}(I^x_{t-1}) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | I^x_{t-1}) \underline{\beta}^k_t(\omega_t)$$

  - » We generate a new cut for each information state. This can work quite well if the information state is simple, but not if it has multiple dimensions.