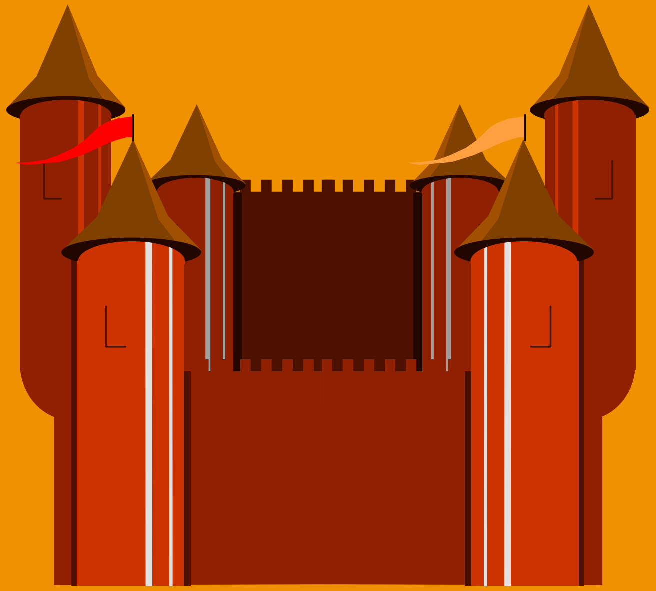
ORF 544

Stochastic Optimization and Learning

Spring, 2019



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

Week 10

Backward approximate dynamic programming

Backward MDP and the curse of dimensionality

Curse of dimensionality

- The ultimate policy is to optimize from now on:

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

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

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

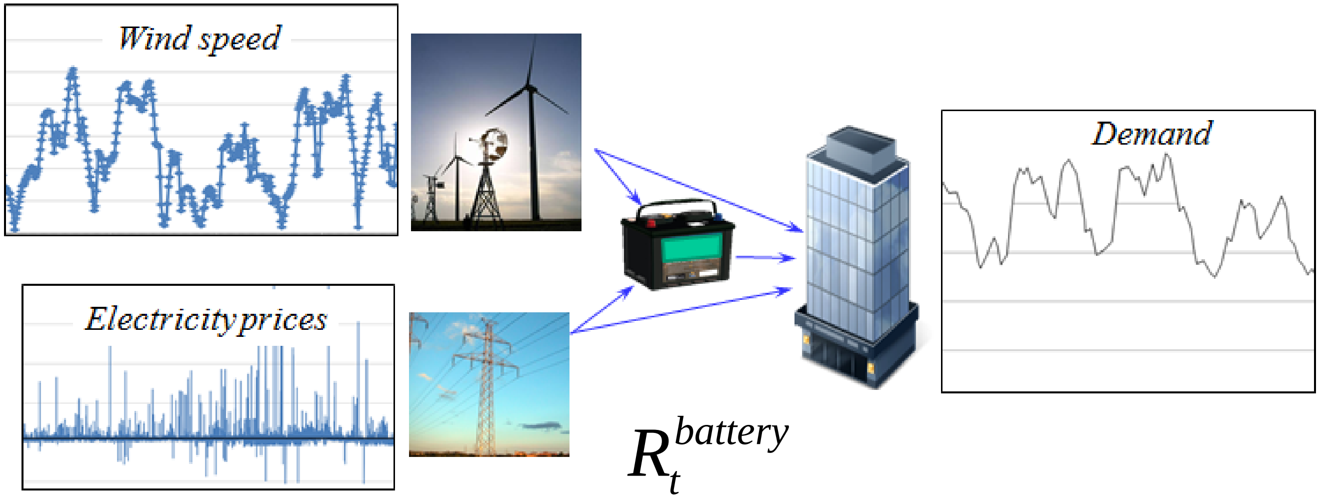
- *Sometimes* we can compute this function exactly!

Curse of dimensionality

- Energy storage with stochastic prices, supplies and demands.

E_t^{wind}

P_t^{grid}



$R_t^{battery}$

D_t

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

W_{t+1} = Exogenous inputs

S_t = State variable

x_t = Controllable inputs

© 2019 Warren B. Powell

Curse of dimensionality

- Bellman's optimality equation

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

$$\begin{bmatrix} E_t^{wind} \\ P_t^{grid} \\ D_t^{load} \\ R_t^{battery} \end{bmatrix} \qquad \begin{bmatrix} x_t^{wind-battery} \\ x_t^{wind-load} \\ x_t^{grid-battery} \\ x_t^{grid-load} \\ x_t^{battery-load} \end{bmatrix} \qquad \begin{bmatrix} \hat{E}_{t+1}^{wind} \\ \hat{P}_{t+1}^{grid} \\ \hat{D}_{t+1}^{load} \end{bmatrix}$$

These are the "three curses of dimensionality."

Curse of dimensionality

- Backward dynamic programming in one dimension

```
Step 0: Initialize $V_{T+1}(R_{T+1}) = 0$ for $R_{T+1} = 0,1,...,100$
Step 1: Step backward $t = T, T-1, T-2, ...$
  Step 2: Loop over $R_t = 0,1,...,100$
    Step 3: Loop over all decisions $-(R^{\max} - R_t) \le x_t \le R_t$
      Step 4: Take the expectation over exogenous information:
        Compute $Q(R_t, x_t) = C(R_t, x_t) + \boxed{\sum_{w=0}^{100}} V_{t+1}(\min\{R^{\max}, R_t - x + w\}) P^W(w)$
      End step 4;
    End Step 3;
    Find $V_t^*(R_t) = \max_{x_t} Q(R_t, x_t)$
    Store $X_t^{\pi^*}(R_t) = \arg\max_{x_t} Q(R_t, x_t)$.  (This is our policy)
  End Step 2;
End Step 1;
```

Curse of dimensionality

- Dynamic programming in multiple dimensions

```
Step 0: Initialize $V_{T+1}(S_{T+1}) = 0$ for all states.
Step 1: Step backward $t = T, T-1, T-2, \ldots$
    Step 2: Loop over $S_t = \left(R_t, D_t, p_t, E_t\right)$ (four loops)
        Step 3: Loop over all decisions $x_t$ (all dimensions)
            Step 4: Take the expectation over each random dimension $\left(\hat{D}_t, \hat{p}_t, \hat{E}_t\right)$
                Compute $Q(S_t, x_t) = C(S_t, x_t) +$
                    $\boxed{\sum_{w_1=0}^{100}\sum_{w_2=0}^{100}\sum_{w_3=0}^{100}} V_{t+1}\left(S^M\left(S_t, x_t, W_{t+1} = (w_1, w_2, w_3)\right)\right) P^W(w_1, w_2, w_3)$
            End step 4;
        End Step 3;
        Find $V_t^*(S_t) = \max_{x_t} Q(S_t, x_t)$
        Store $X_t^{\pi^*}(S_t) = \arg\max_{x_t} Q(S_t, x_t)$. (This is our policy)
    End Step 2;
End Step 1;
```

Curse of dimensionality

- Notes:
  - » There are potentially three "curses of dimensionality" when using backward dynamic programming:
    - The state variable – We need to enumerate all states. If the state variable is a vector with more than two dimensions, the state space gets very big, very quickly.
    - The random information – We have to sum over all possible realizations of the random variable. If this has more than two dimensions, this gets very big, very quickly.
    - The decisions – Again, decisions may be vectors (our energy example has five dimensions). Same problem as the other two.
  - » Some problems fit this framework, but not very. However, when we can use this framework, we obtain something quite rare: an optimal policy.

Curse of dimensionality

- Strategies for approximating value functions:
  - » Backward dynamic programming
    - Exact using lookup tables
    - Backward approximate dynamic programming:
      - – Linear regression
      - – Low rank approximations
  - » Forward approximate dynamic programming
    - Approximation architectures
      - – Lookup tables
        - » Correlated beliefs
        - » Hierarchical
      - – Linear models
      - – Convex/concave
    - Updating schemes
      - – Pure forward pass TD(0)
      - – Double pass TD(1)

Backward ADP-Chapter 16

Backward ADP

- Classical backward dynamic programming
  - » Uses lookup table representations of value functions
  - » Assumes the one-step transition matrix can be computed (which is also lookup table).
  - » "Dynamic programming" does *not* suffer from the curse of dimensionality (as we show below), but lookup tables do.
  - » There are three curses of dimensionality, but often it is the state variable that causes the most problems.
  - » Backward ADP uses a sample of states rather than all the states, and a statistical model for the value of being in a state. At a minimum this fixes two of the three curses of dimensionality.

Backward ADP

- Backward approximate dynamic programming
  - » Basic idea is to step backward in time, just as we do with classical backward dynamic programming.
  - » Instead of looping over all the states, loop over a random sample.
  - » Now, use the sample of values and states to produce an approximate value function:
    - Any statistical model
    - Low-rank approximations (works well when value functions are smooth).
  - » You still need to take full expectation (although this might be approximated) and search over all actions.

Backward ADP

- Backward ADP

```
Step 0: Initialize $\bar{V}_{T+1}(S_{T+1}) = 0$ for all states.
Step 1: Step backward $t = T, T-1, T-2, \ldots$
    Step 2: Loop over a random sample of states $\hat{s}_t = (R_t, D_t, p_t, E_t)$ (one loop)
        Step 3: Loop over all decisions $x_t$ (all dimensions)
            Step 4: Take the expectation over each random dimension $(\hat{D}_t, \hat{p}_t, \hat{E}_t)$
            Compute $Q(\hat{s}_t, x_t) = C(\hat{s}_t, x_t) +$
                $\sum_{w_1=0}^{100} \sum_{w_2=0}^{100} \sum_{w_3=0}^{100} \bar{V}_{t+1}\left(S^M\left(\hat{s}_t, x_t, W_{t+1} = (w_1, w_2, w_3)\right)\right) P^W(w_1, w_2, w_3)$
            End step 4;
        End Step 3;
        Find $\hat{v}_t(\hat{s}_t) = \max_{x_t} Q(\hat{s}_t, x_t)$
    End Step 2;
    Use sampled $\hat{v}_t(\hat{s}_t)$'s to find an approximate $\bar{V}_t(s)$.
End Step 1;
```

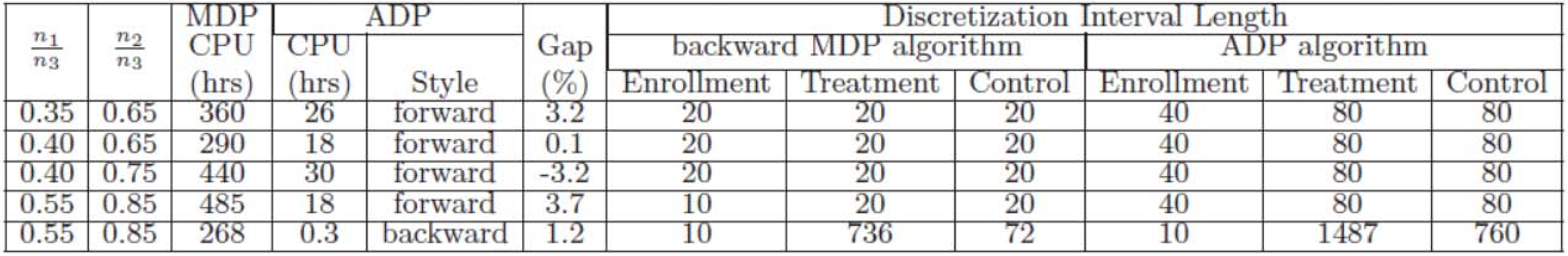
Backward ADP

- Backward ADP with the post-decision state
  - » Computing the imbedded expectation can be a pain.
  - » Instead of sampling over (pre-decision) states, sample post-decision states s_{t-1}^x.
  - » Then draw a sample W_t, and simulate our way to the next pre-decision state s_t.
  - » From s_t, compute the sampled value v_t from
    $$v_t = \max_x \left(C(s_t, x_t) + \bar{V}_t^x(s_t^x)\right)$$
  - » Do this N times and create a dataset $(s_{t-1}^{x,n}, v_t^n)_{n=1}^N$
  - » Now use this dataset to fit a statistical model for $\bar{V}_{t-1}^x(s_{t-1}^x)$.
  - » Repeat.

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	ADP		Gap	Discretization Interval Length					
		CPU	CPU			backward MDP algorithm			ADP algorithm		
		(hrs)	(hrs)	Style	(%)	Enrollment	Treatment	Control	Enrollment	Treatment	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

Backward ADP

- Energy storage problem
  - » Lookup table – 99.3 percent of optimal, .67 hours.
  - » Backward MDP 11.3 hours.

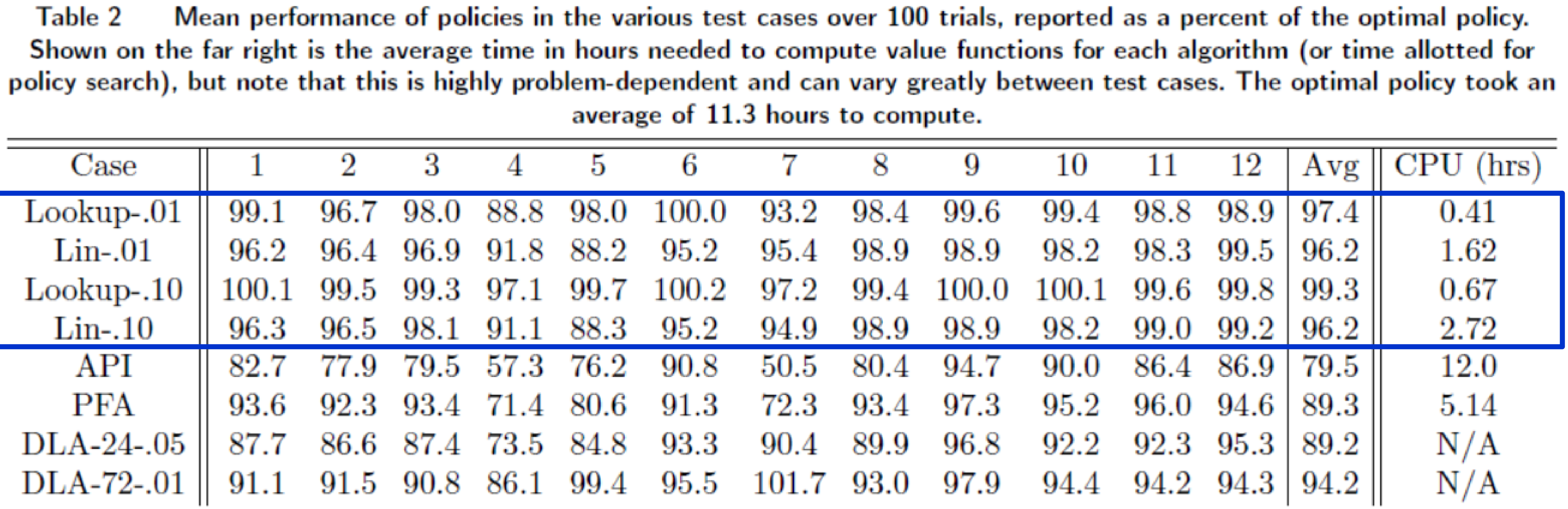
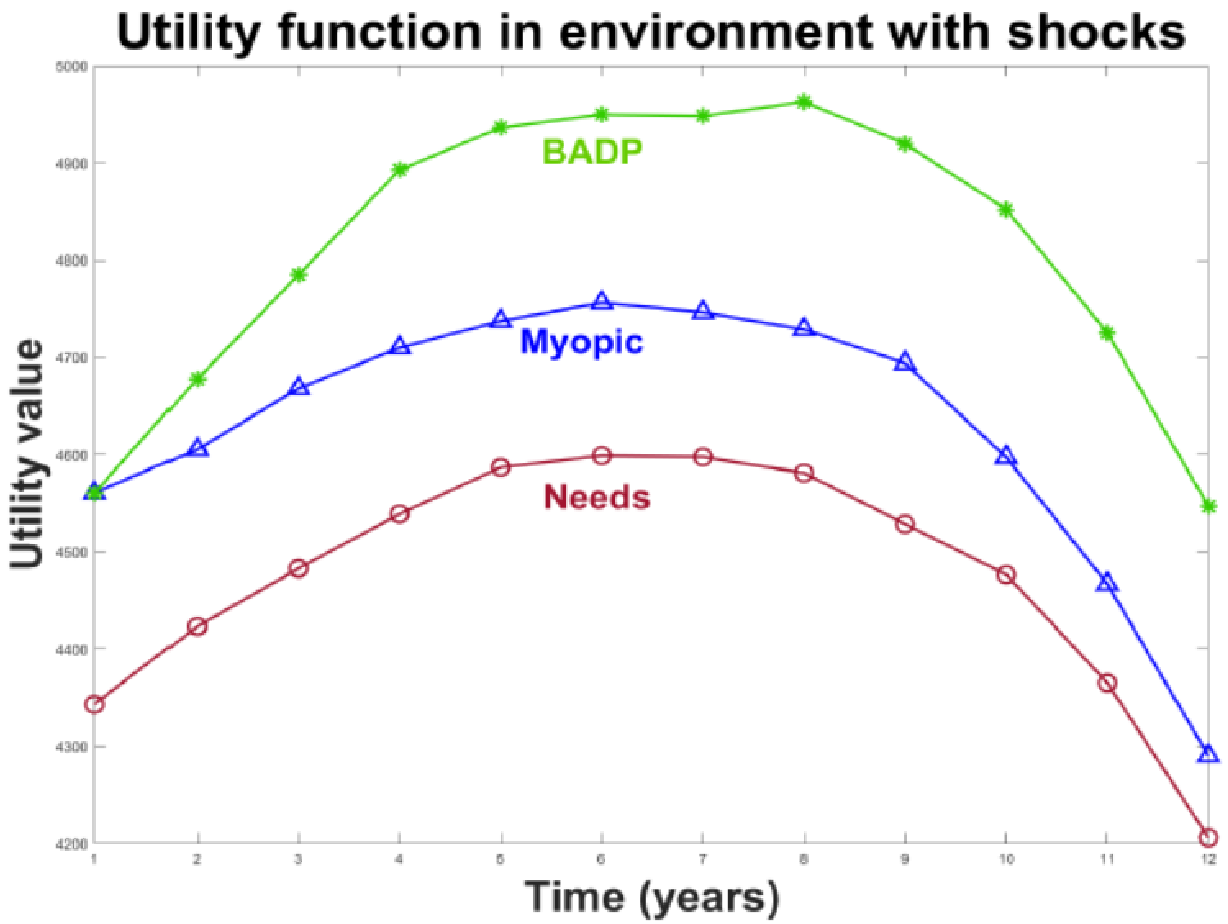


Table 2 Mean performance of policies in the various test cases over 100 trials, reported as a percent of the optimal policy. Shown on the far right is the average time in hours needed to compute value functions for each algorithm (or time allotted for policy search), but note that this is highly problem-dependent and can vary greatly between test cases. The optimal policy took an average of 11.3 hours to compute.

Case	1	2	3	4	5	6	7	8	9	10	11	12	Avg	CPU (hrs)
Lookup-.01	99.1	96.7	98.0	88.8	98.0	100.0	93.2	98.4	99.6	99.4	98.8	98.9	97.4	0.41
Lin-.01	96.2	96.4	96.9	91.8	88.2	95.2	95.4	98.9	98.9	98.2	98.3	99.5	96.2	1.62
Lookup-.10	100.1	99.5	99.3	97.1	99.7	100.2	97.2	99.4	100.0	100.1	99.6	99.8	99.3	0.67
Lin-.10	96.3	96.5	98.1	91.1	88.3	95.2	94.9	98.9	98.9	98.2	99.0	99.2	96.2	2.72
API	82.7	77.9	79.5	57.3	76.2	90.8	50.5	80.4	94.7	90.0	86.4	86.9	79.5	12.0
PFA	93.6	92.3	93.4	71.4	80.6	91.3	72.3	93.4	97.3	95.2	96.0	94.6	89.3	5.14
DLA-24-.05	87.7	86.6	87.4	73.5	84.8	93.3	90.4	89.9	96.8	92.2	92.3	95.3	89.2	N/A
DLA-72-.01	91.1	91.5	90.8	86.1	99.4	95.5	101.7	93.0	97.9	94.4	94.2	94.3	94.2	N/A

Backward ADP

- Resource allocation in Africa
  - » Extended widely cited myopic policy to a dynamic setting.



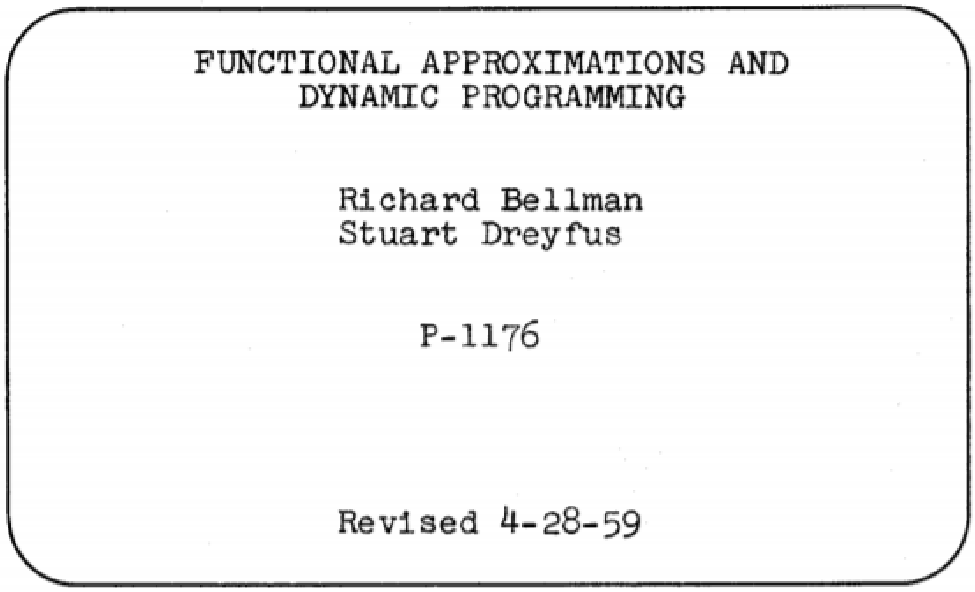
Backward ADP

- Notes
  - » By now I have applied backward ADP to approximate four projects, three with rigorous benchmarks, and one (the resource allocation problem) with a high quality benchmark (the myopic policy).
  - » Each time it seems to have worked very well.

Histories of approximate dynamic programming and reinforcement learning

Histories of ADP/reinforcement learning

- 1959 – Operations research
  - » Bellman recognizes the limitations of classical backward dynamic programming.
  - » Introduces the idea of statistically approximating value functions.
  - » This line of research quickly died out in the operations research community.



FUNCTIONAL APPROXIMATIONS AND
DYNAMIC PROGRAMMING

Richard Bellman
Stuart Dreyfus

P-1176

Revised 4-28-59

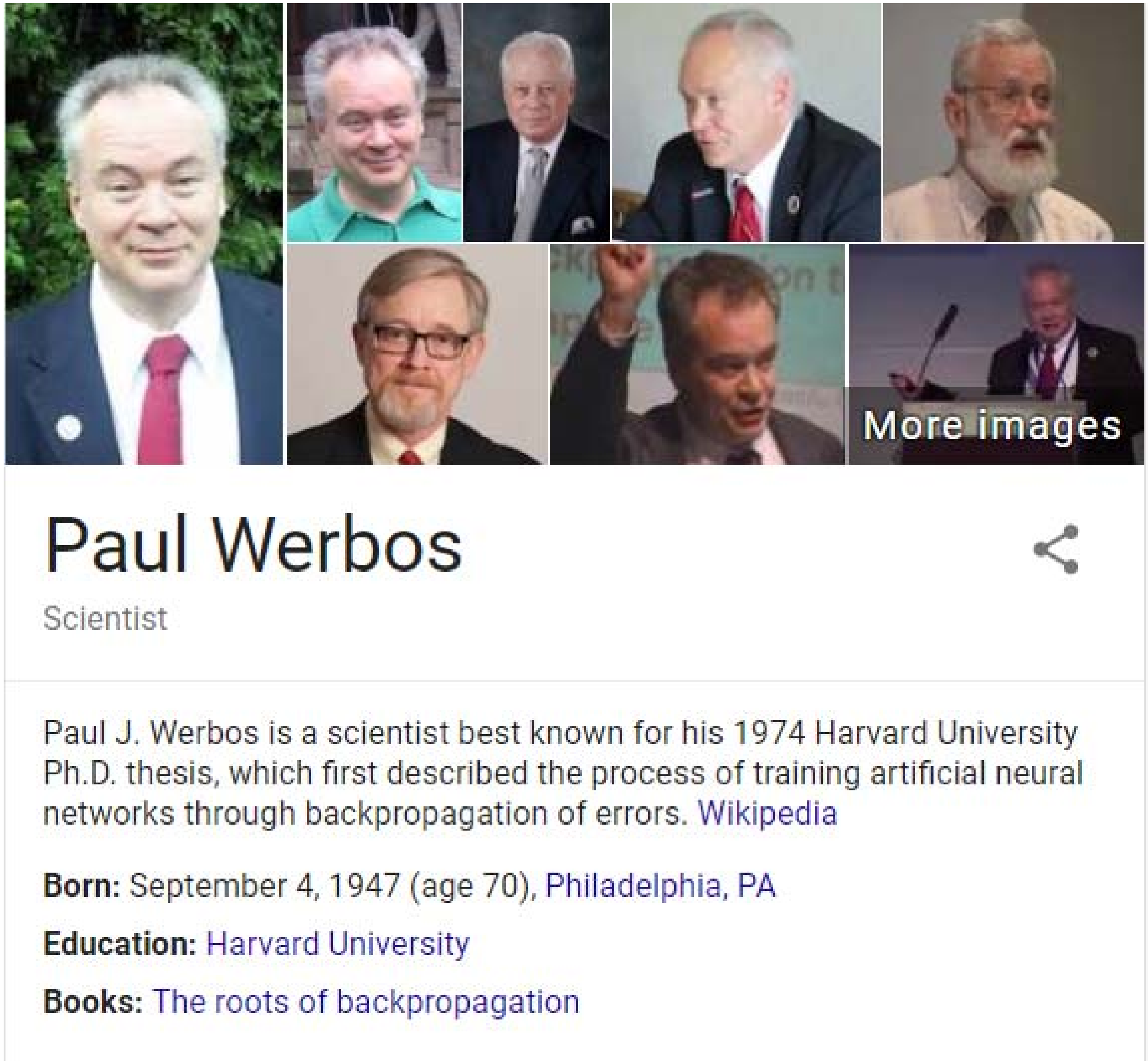
The fields of stochastic optimization

- Approximate dynamic programming/reinforcement learning
  - » 1959 paper by Bellman – first attempt at ADP
  - » ADP in control theory – 1974 dissertation of Paul Werbos
  - » Reinforcement learning in computer science – 1980 research of Rich Sutton and Andy Barto
    - 1998 book *Reinforcement Learning* establishes the field
  - » 1996 book *Neuro-Dynamic programming* – First to bridge the theory of stochastic approximation methods (Robbins and Monro) with reinforcement learning
  - » Late 1990's – ADP returns to operations research
    - 1994 dissertation of Ben van Roy
    - Late 1990's onward – Value function approximations for MDPs (discrete actions)
    - 1998 onward – use of ADP for vector-valued actions (Powell and students)
  - » 2007 ADP book by Powell; second edition in 2011.
    - Three curses of dimensionality; high dimensional decision vectors (action spaces)

Histories of ADP/reinforcement learning

- 1974 – Controls community
  - » Paul Werbos introduces "backpropagation" for approximating the "cost to go" function for continuous controls problems.
  - » Engineering controls community continues to develop these ideas, with special emphasis on the use of neural networks in two ways:
    - Actor nets – A neural network for the policy (which chooses the action given a state).
    - Critic nets – A neural network for approximating the value function (cost to go function in the language of control theory)
  - » Paul Werbos becomes an NSF program officer and continues to promote "approximate dynamic programming." Funded workshops on ADP in 2002 and 2006.
  - » 1994 – Beginning with 1994 paper of John Tsitsiklis, bridging of the heuristic techniques of Q-learning and the mathematics of stochastic approximation methods (Robbins-Monro).
  - » 1996 book "Neuro-Dynamic Programming" by Bertsekas and Tsitsiklis formally bridges Q-learning and stochastic approx. methods

Histories of ADP/reinforcement learning



Paul Werbos

Scientist

Paul J. Werbos is a scientist best known for his 1974 Harvard University Ph.D. thesis, which first described the process of training artificial neural networks through backpropagation of errors. Wikipedia

Born: September 4, 1947 (age 70), Philadelphia, PA

Education: Harvard University

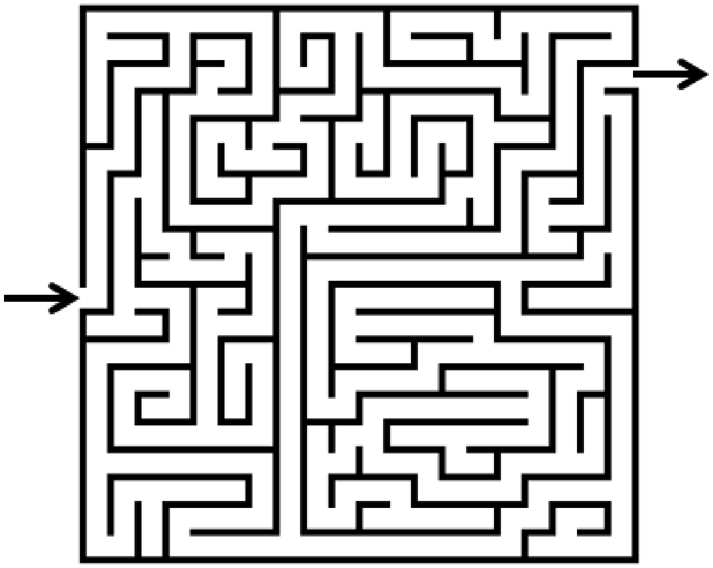
Books: The roots of backpropagation

Werbos, P. J. (1974). Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. dissertation Harvard University.

Histories of ADP/reinforcement learning

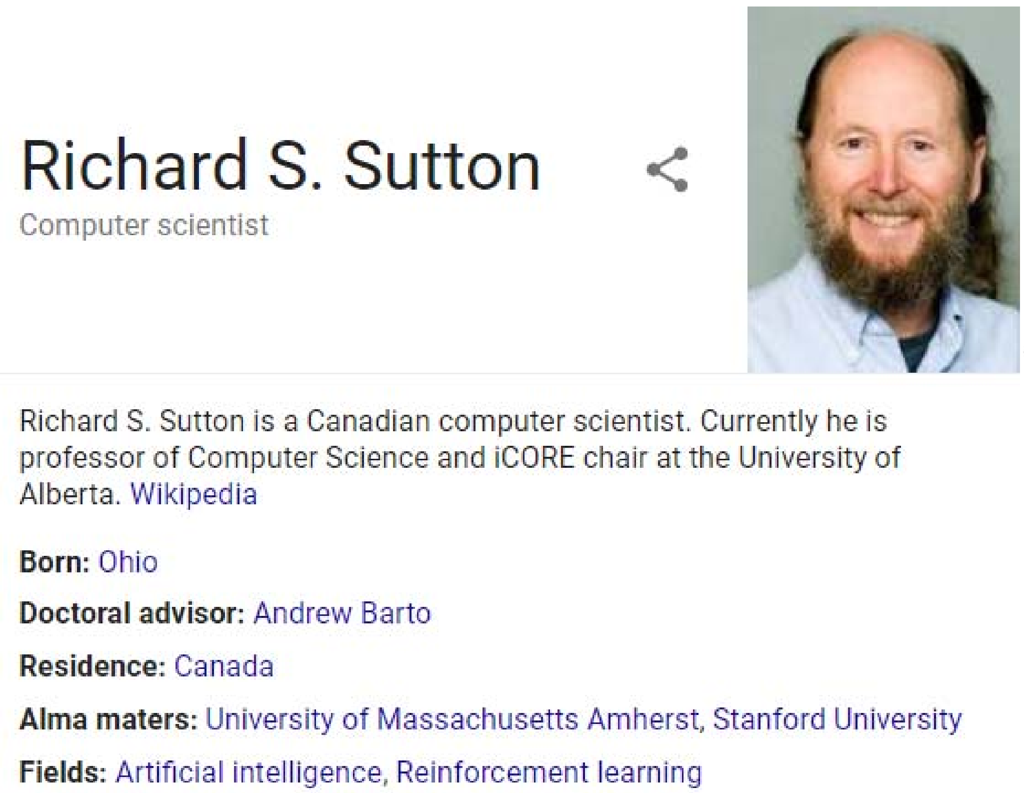
- History of Q-learning
  - » Began ~1980 with Andy Barto (supervisor) and Rich Sutton (student) studying behavior of mice in mazes.
  - » Heuristically developed basic feedback mechanism:

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



  - » Late 1980's the link to Markov decision processes was made, and the community adopted the basic notation of Markov decision processes.
  - » Bizarrely, the RL community adopted popular test problems from the controls community, which are primarily deterministic:
    - Inverted pendulum problem.
    - Hill-climbing
    - Truck backer-upper
    - Robotics applications.

Histories of ADP/reinforcement learning



Richard S. Sutton

Computer scientist

Richard S. Sutton is a Canadian computer scientist. Currently he is professor of Computer Science and iCORE chair at the University of Alberta. Wikipedia

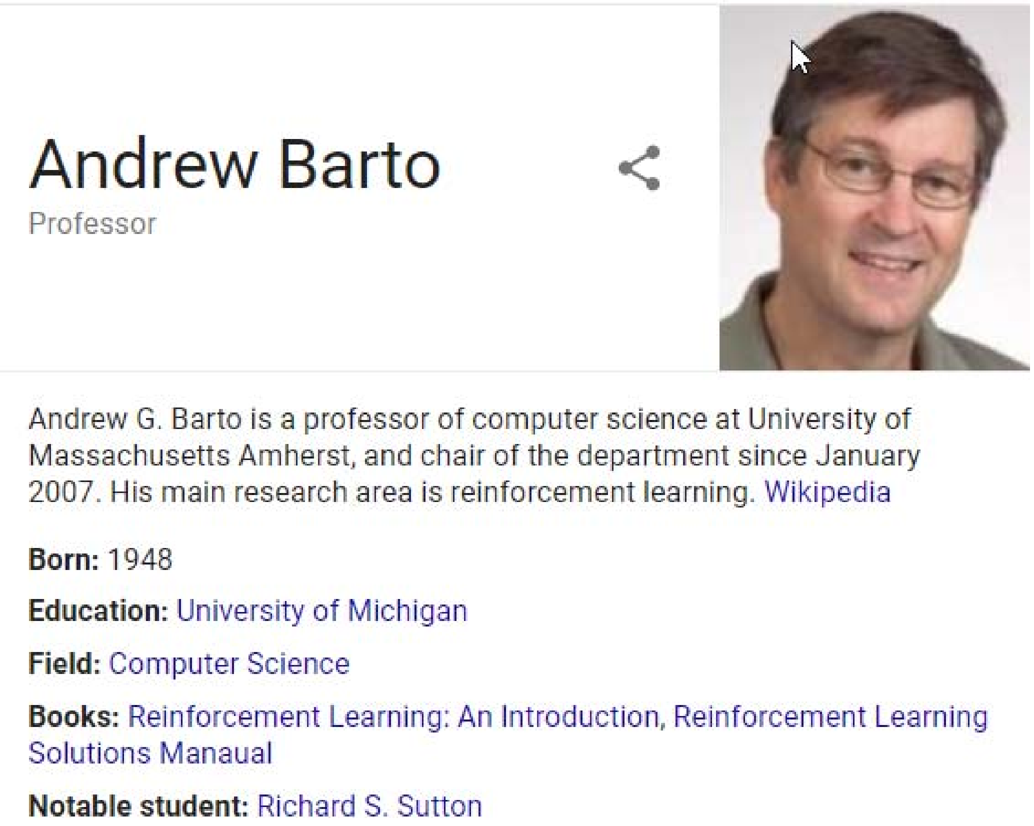
Born: Ohio

Doctoral advisor: Andrew Barto

Residence: Canada

Alma maters: University of Massachusetts Amherst, Stanford University

Fields: Artificial intelligence, Reinforcement learning



Andrew Barto

Professor

Andrew G. Barto is a professor of computer science at University of Massachusetts Amherst, and chair of the department since January 2007. His main research area is reinforcement learning. Wikipedia

Born: 1948

Education: University of Michigan

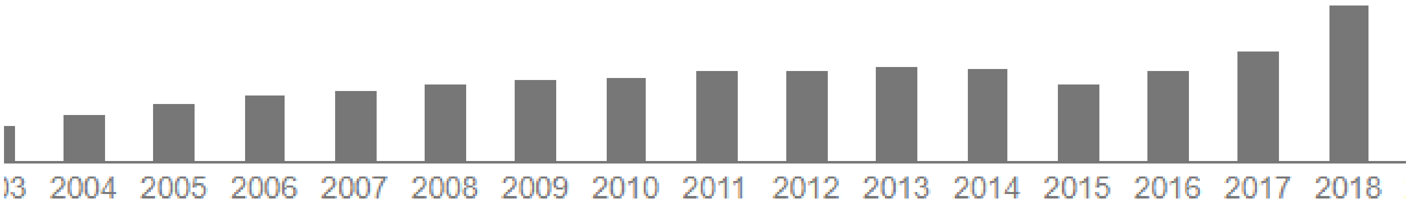
Field: Computer Science

Books: Reinforcement Learning: An Introduction, Reinforcement Learning Solutions Manaual

Notable student: Richard S. Sutton

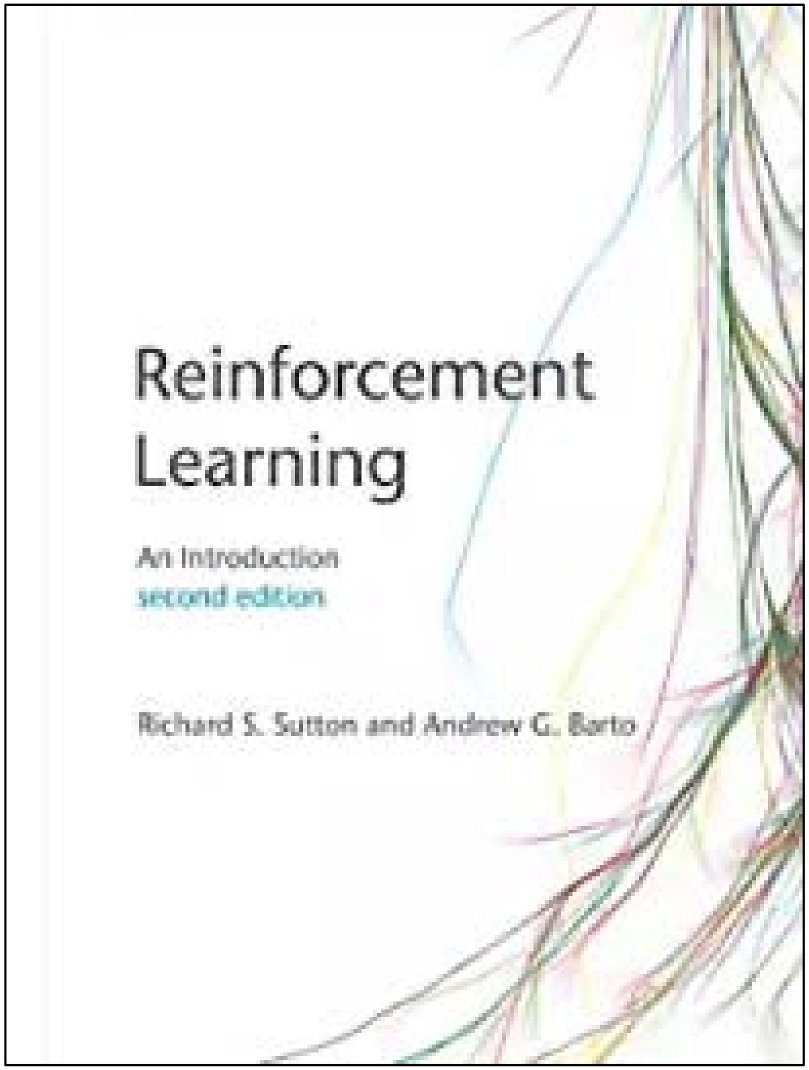
Reinforcement Learning

Cited by 29594



Histories of ADP/reinforcement learning

- Second edition of Reinforcement Learning



Reinforcement Learning

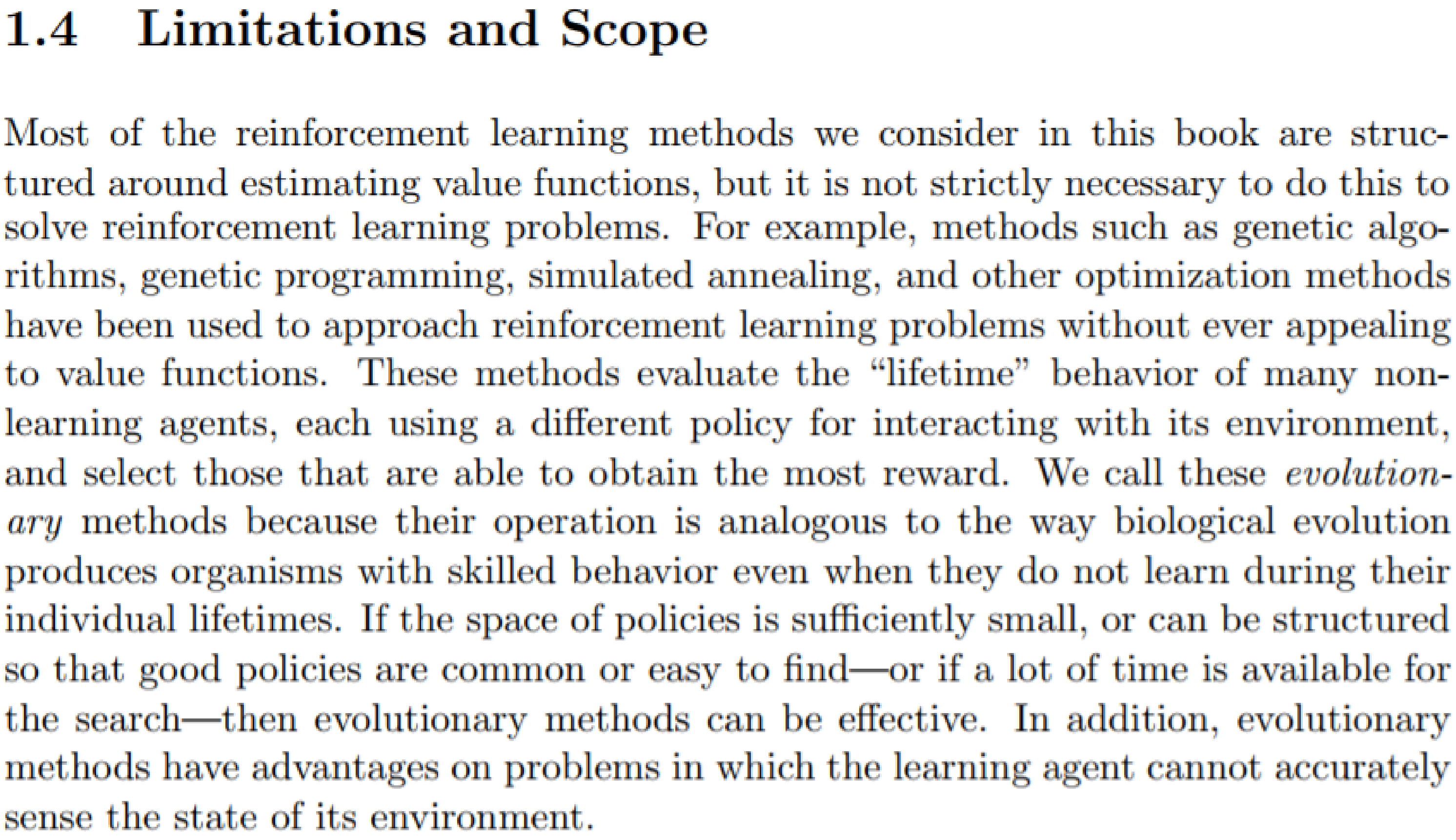
An Introduction

second edition

Richard S. Sutton and Andrew G. Barto

Histories of ADP/reinforcement learning

- From Q-learning to other policies
  - » From the second edition:



1.4 Limitations and Scope

Most of the reinforcement learning methods we consider in this book are structured around estimating value functions, but it is not strictly necessary to do this to solve reinforcement learning problems. For example, methods such as genetic algorithms, genetic programming, simulated annealing, and other optimization methods have been used to approach reinforcement learning problems without ever appealing to value functions. These methods evaluate the "lifetime" behavior of many nonlearning agents, each using a different policy for interacting with its environment, and select those that are able to obtain the most reward. We call these *evolutionary* methods because their operation is analogous to the way biological evolution produces organisms with skilled behavior even when they do not learn during their individual lifetimes. If the space of policies is sufficiently small, or can be structured so that good policies are common or easy to find—or if a lot of time is available for the search—then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems in which the learning agent cannot accurately sense the state of its environment.

  - » This hints at policy search, but still ignores lookahead policies (Monte Carlo tree search)

Histories of ADP/reinforcement learning

- 1990's – Operations research
  - » Ben van Roy (student of John Tsitsklis) developed the ideas of using regression models for solving the "curse of dimensionality problem" of dynamic programming.
  - » 1991 – Pereira and Pinto introduce the idea of Benders cuts for "solving the curse of dimensionality" for stochastic linear programs. Method called "stochastic dual decomposition procedure" (SDDP)
  - » ~2000 – Work of WBP on "adaptive dynamic programming" for high-dimensional problems in logistics.
  - » With Ben van Roy (who first introduced the term), WBP developed the idea of the post-decision state which opened the door for solving high-dimensional convex DPs.
  - » WBP switches to "approximate dynamic programming" after attending Werbos' 2002 conference on "Approximate dynamic programming and reinforcement learning"

Histories of ADP/reinforcement learning

- Today:
  - » Controls community now uses the term "adaptive dynamic programming"
    - Balanced using of "control laws" (PFAs), Hamilton/Jacobi/Bellman equations (VFAs) and "model predictive control" (DLAs)
  - » "Reinforcement learning" has spread from Q-learning to include "policy search" and Monte Carlo tree search (a form of direct lookahead – we will get to this later).
  - » 2014-2016 Two tutorials by WBP establish the "four classes of policies"
  - » "Optimization under Uncertainty" book (being written for this class) is the first to truly unify all of the different subcommunities of stochastic optimization.

Next steps

- What we are going to cover:
  - » Forward approximate dynamic programming
    - Estimating the value of a fixed policy
    - Optimizing while learning

Learning the value of a policy

TD-learning

TD-learning

- Temporal difference learning
  - » The “temporal difference” is given by

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

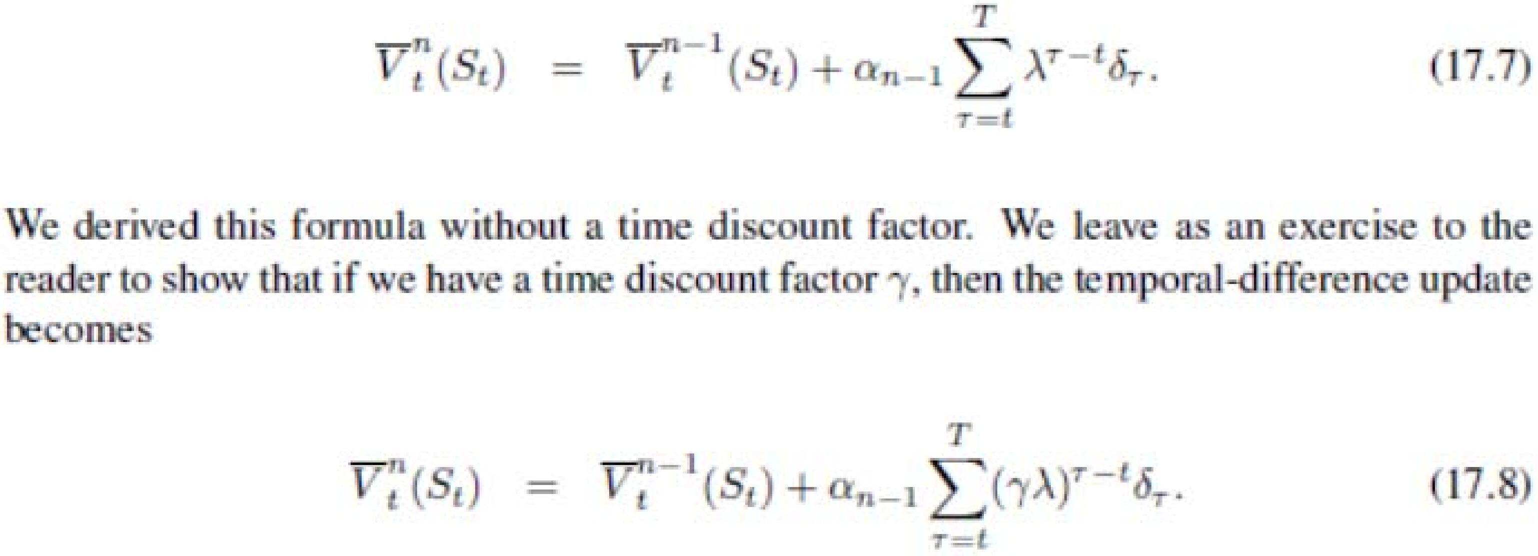
  - » In other words, this is “old estimate minus new estimate”. We can update our value function approximation using

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

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

TD-learning

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



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

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

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

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

TD-learning

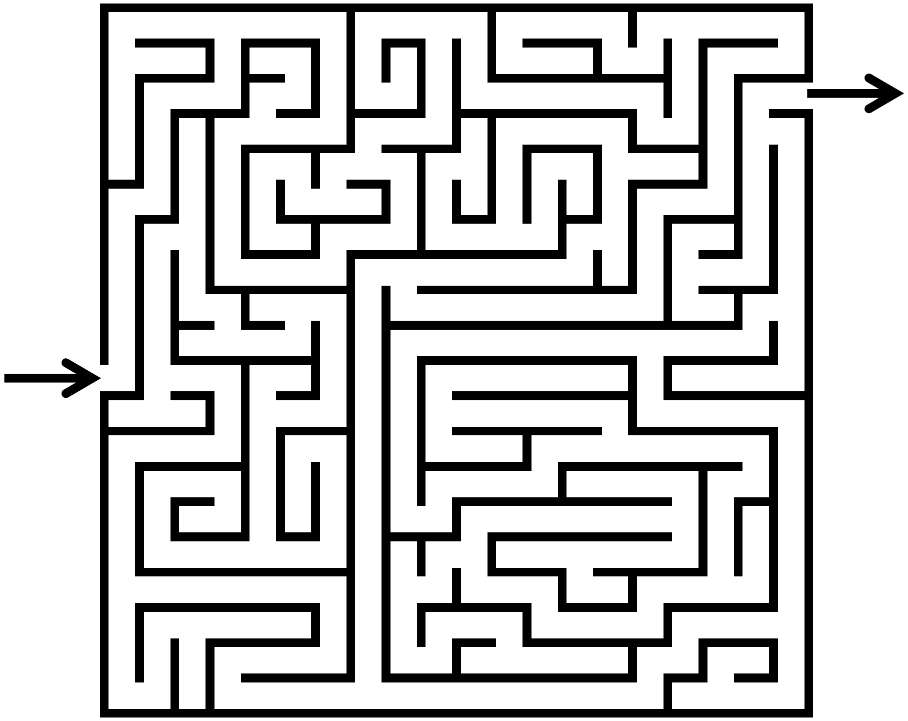
- Approximating the value function
  - » Temporal difference updates can be used in any recursive estimation algorithm:
    - Lookup tables
      - – Independent beliefs
      - – Correlated beliefs
    - Parametric models
      - – Linear
      - – Nonlinear
      - – Shallow neural networks
    - Nonparametric
      - – Kernel regression
      - – Locally linear
      - – Deep neural networks

Q-learning

"Reinforcement learning"

Q-learning

- Mouse in a maze problem

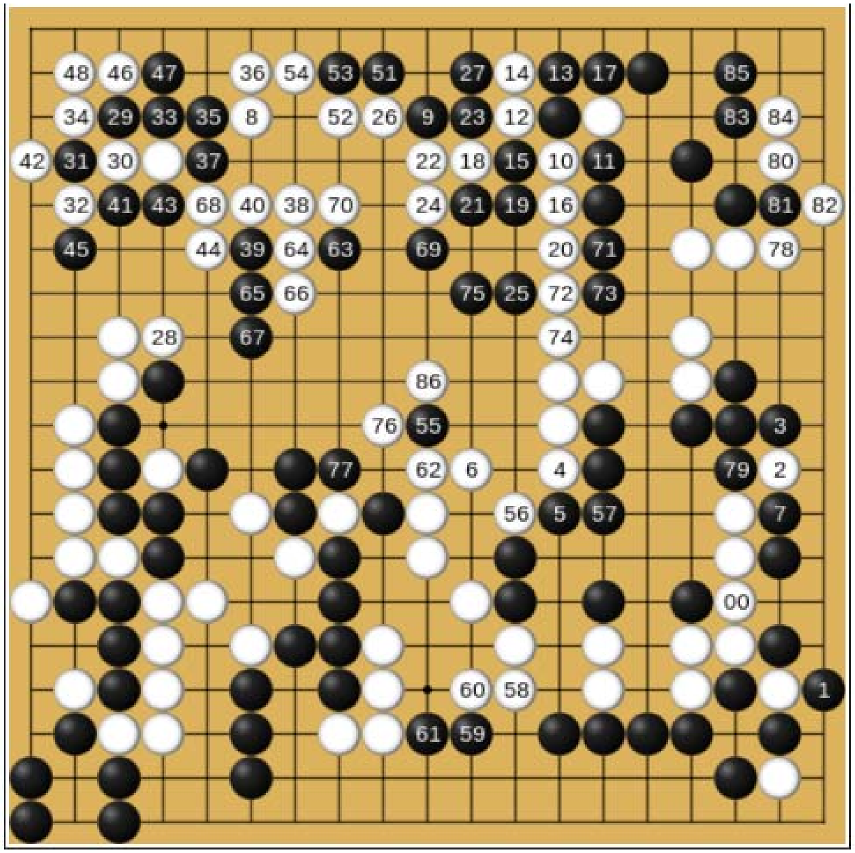


Receive reward = 1

© 2019 Warren B. Powell

Q-learning

- AlphaGo
  - » Much more complex state space.
  - » Uses hybrid of policies:
    - PFA
    - VFA
    - Lookahead (DLA)



Q-learning

- Basic Q-learning algorithm
  - » Basic update:

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

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

where

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

  - » Given a state s^n and action a^n, we simulate our way to state s'.
  - » Need to determine:
    - State sampling process/policy
    - Action sampling policy

Q-learning

- Some terms from reinforcement learning:
  - » "Behavior policy" is the policy used to choose actions
    - E.g. these are actions observed by a real system
  - » "Target policy" is the policy that we are trying to learn, which is to say the policy we want to implement.
  - » When the target policy is different from the behavior policy, then this is termed "off policy learning"
- In this course
  - » The "learning policy" is the policy (often called an algorithm) that learns the value functions (or Q-factors)
  - » The "implementation policy" is the policy determined by the value functions (or Q-factors).

Q-learning

- Learning policy
  - » This is the policy that determines what action to choose as a part of learning the Q-factors.
  - » "Exploitation":

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

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

Q-learning

- State sampling policies
  - » Trajectory following

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

    - Helps to avoid sampling states that never happen
    - Problem is that a suboptimal policy may mean that you are not sampling important states.
  - » Exploration
    - Pick a state at random
  - » Hybrid
    - Use trajectory following with randomization, e.g.

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

Q-learning

- Implementation policy
  - » This is the policy we are going to follow based on the Q-factors:

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

- The value of the implementation policy:

$$\bar{F}^{\pi,n} = \sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1}(\omega)) \quad \text{where} \quad S_{t+1} = S^M(S_t, X^{\pi}(S_t), W_{t+1}(\omega))$$

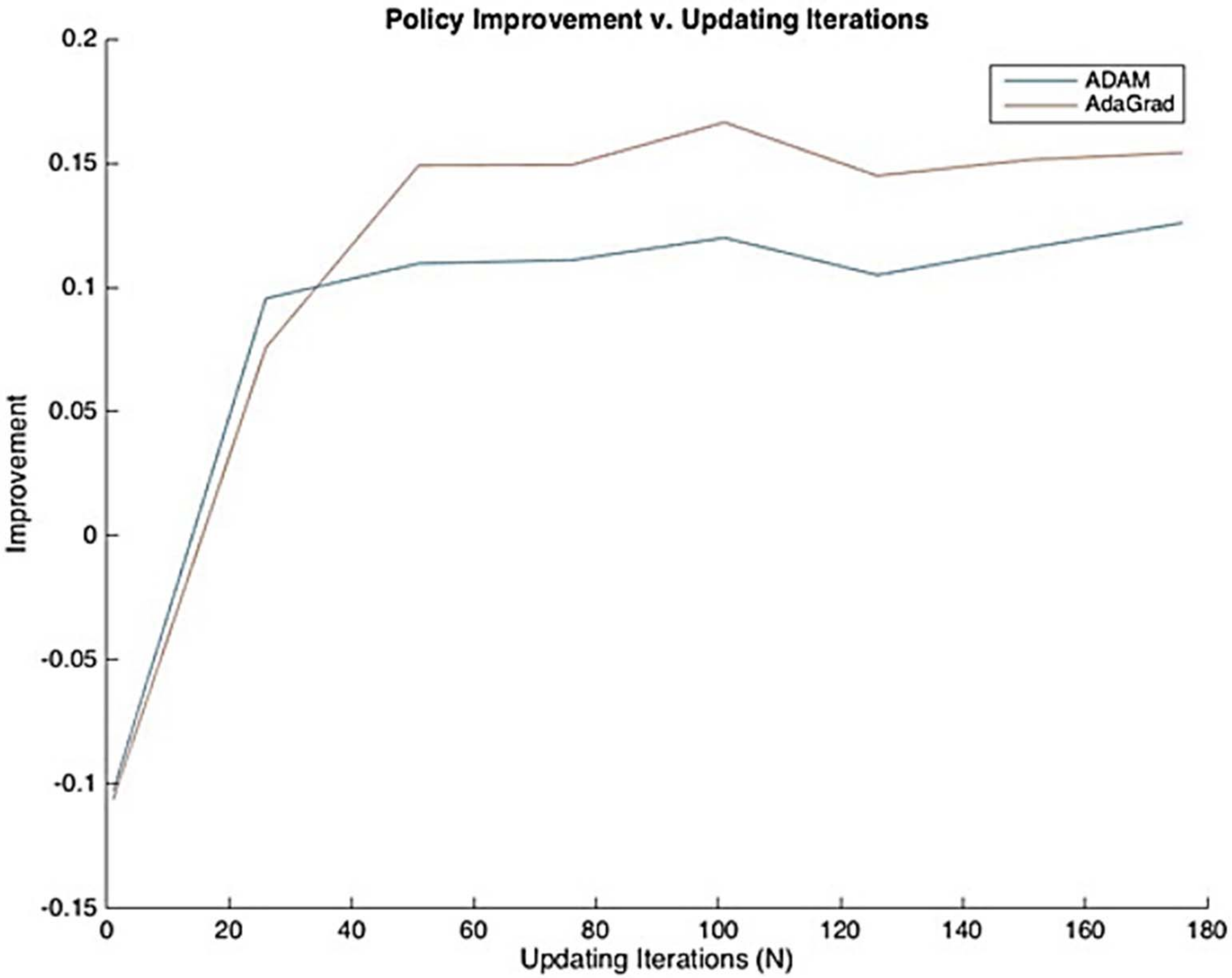
  - » or

$$\bar{F}^{\pi,n} = \frac{1}{N}\sum_{n=1}^{N-1}\sum_{t=0}^{T} C(S_t, X^{\pi}(S_t), W_{t+1}(\omega^n)) \quad \text{where} \quad S_{t+1} = S^M(S_t, X^{\pi}(S_t), W_{t+1}(\omega^n)))$$

- The goal is to find an effective learning policy so that we obtain the best implementation policy.

Q-learning

- Convergence rates:



Q-learning

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

    From a state s^n, choose action

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

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

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

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

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

Q-learning

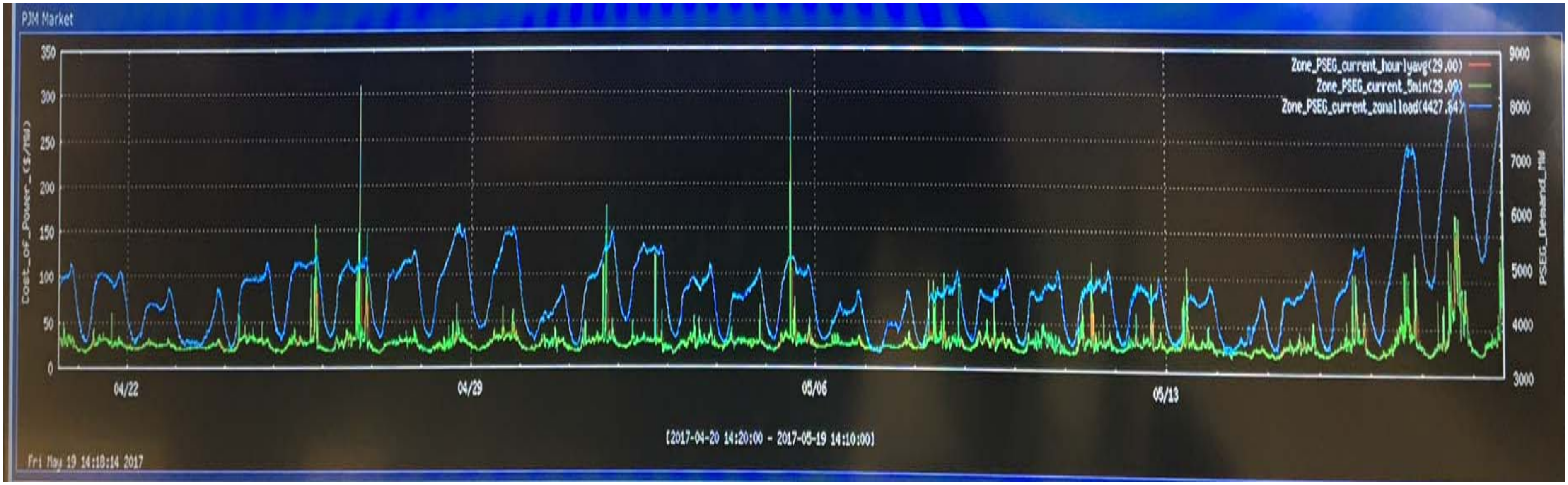
- Model-free vs. model-based
  - » Model-based means we have a mathematical statement of how the problem evolves that can be simulated in the computer.
  - » Model-free refers to a physical process that can be observed, but where we do not have equations describing the evolution over time.
    - The behavior of a human or animal
    - The behavior of the climate
    - The behavior of a complex system such as a chemical plant
  - » Q-learning is often described as "model free" because it can be learned while observing a system.
  - » The resulting policy does not require a model:
    - $A^{\pi}(s) = argmax_a \bar{Q}^n(s, a)$

Q-learning

- Notes
  - » Lookup table belief models are most popular, but do not scale (limit of 3 dimensions).
  - » Various smoothing strategies have been suggested (basically nonparametric statistics), but still limited to 3 dimensions.
  - » Need to be very careful with stepsizes. Q-learning is a form of approximate value iteration where the backward learning is slowed by the use of stepsizes.

Q-learning

- Max operator bias:
  - Second issue arises when there is randomness in the reward.
  - Imagine that we are purchasing energy at a price p_t which evolves randomly from one time period to the next.
  - Imagine buying and selling energy using real time prices:



Q-learning

- Max operator bias (cont'd)
  - » This introduces noise in $\hat{q}^n(s^n, a^n)$:

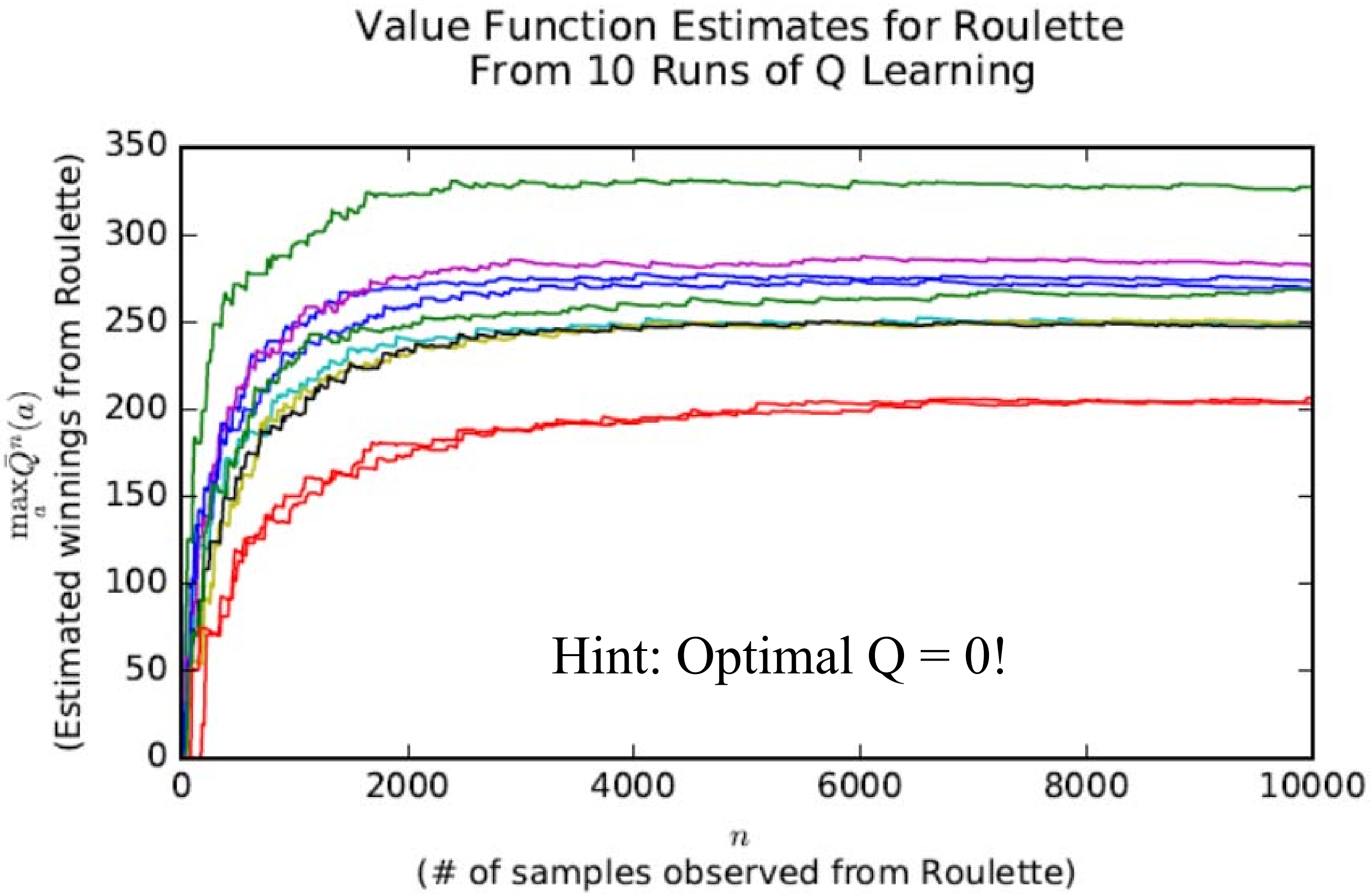
$$\hat{q}^n(s^n, a^n) = C(s^n, a^n) + \gamma \max_{a'} \bar{Q}^{n-1}(s', a')$$

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

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

Q-learning

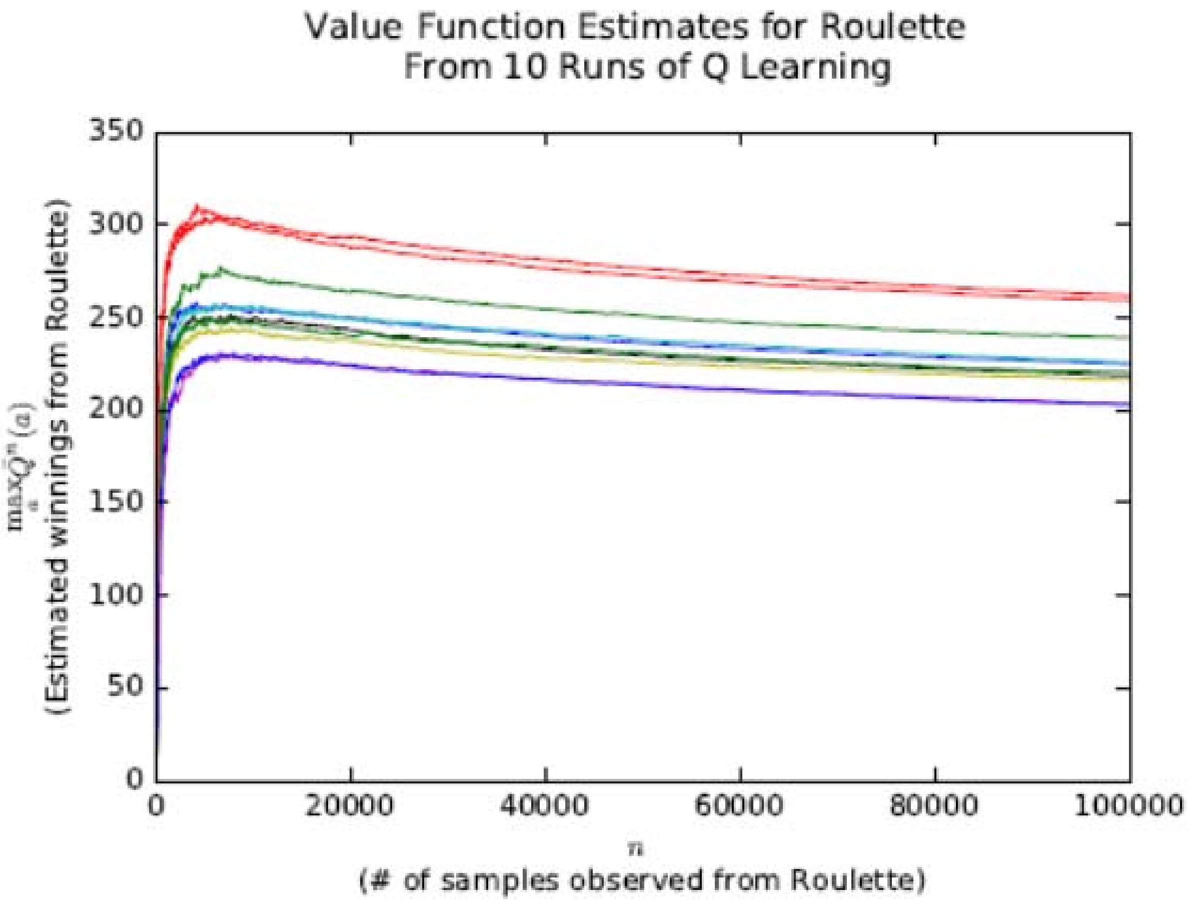
- Roulette
  - » Optimal solution is not to play – optimal value of game is zero
  - » Q-learning over 10,000 iterations



Hint: Optimal Q = 0!

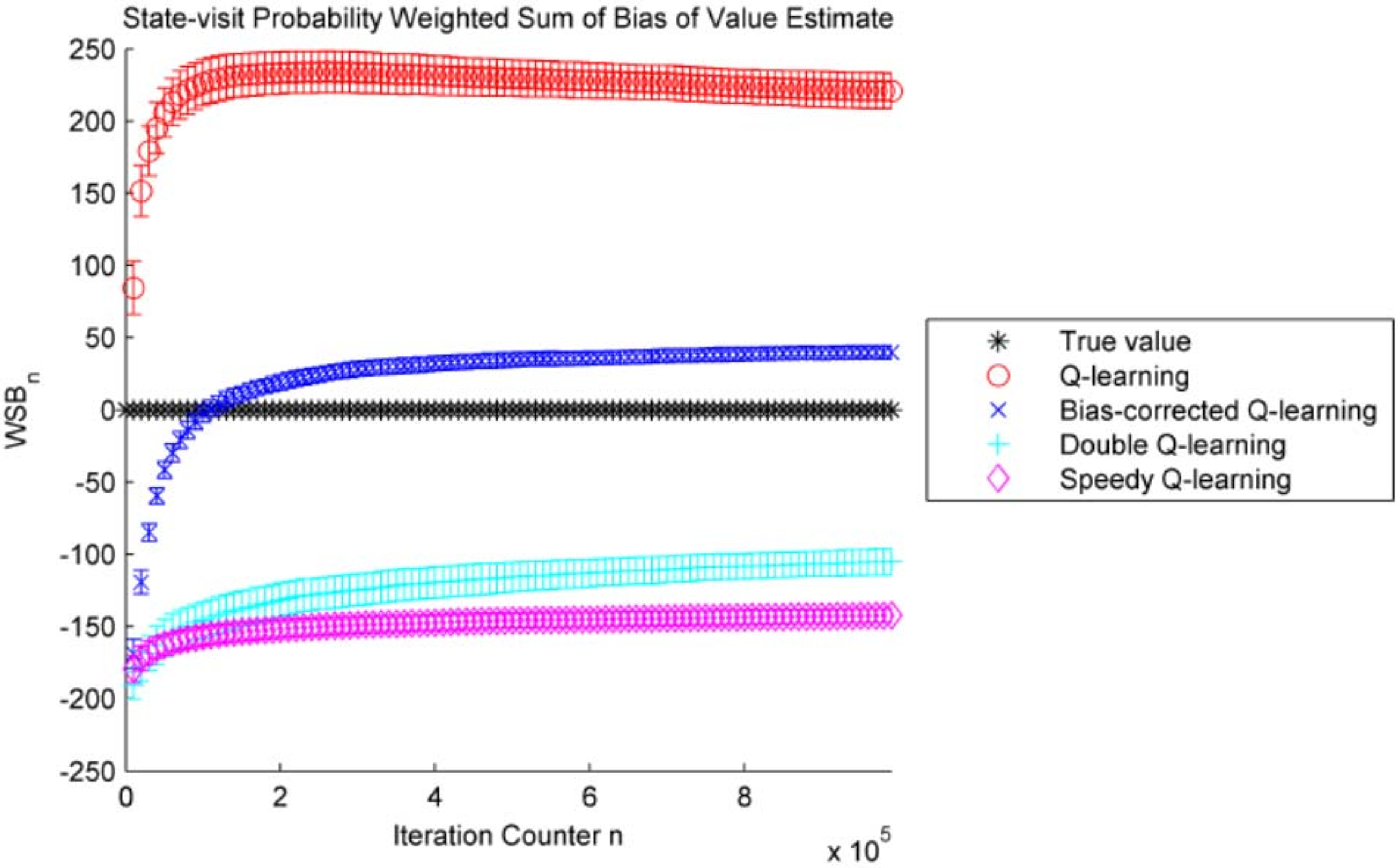
Q-learning

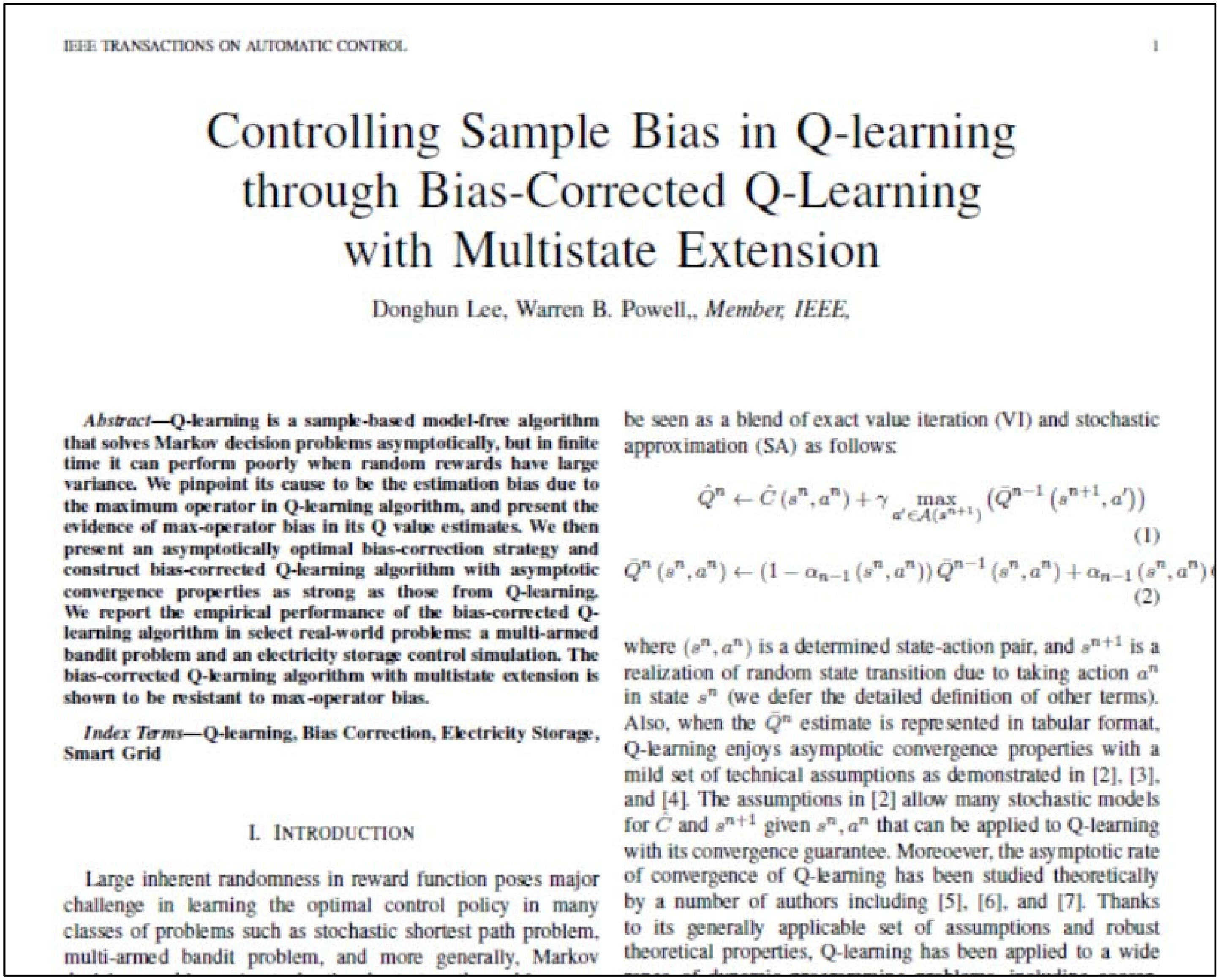
- Roulette
  - » Optimal solution is not to play – optimal value of game is zero
  - » Q-learning over 10,000 iterations



Q-learning

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





Controlling Sample Bias in Q-learning through Bias-Corrected Q-Learning with Multistate Extension

Donghun Lee, Warren B. Powell,, *Member, IEEE,*

***Abstract*—Q-learning is a sample-based model-free algorithm that solves Markov decision problems asymptotically, but in finite time it can perform poorly when random rewards have large variance. We pinpoint its cause to be the estimation bias due to the maximum operator in Q-learning algorithm, and present the evidence of max-operator bias in its Q value estimates. We then present an asymptotically optimal bias-correction strategy and construct bias-corrected Q-learning algorithm with asymptotic convergence properties as strong as those from Q-learning. We report the empirical performance of the bias-corrected Q-learning algorithm in select real-world problems: a multi-armed bandit problem and an electricity storage control simulation. The bias-corrected Q-learning algorithm with multistate extension is shown to be resistant to max-operator bias.**

***Index Terms*—Q-learning, Bias Correction, Electricity Storage, Smart Grid**

I. Introduction

Large inherent randomness in reward function poses major challenge in learning the optimal control policy in many classes of problems such as stochastic shortest path problem, multi-armed bandit problem, and more generally, Markov

be seen as a blend of exact value iteration (VI) and stochastic approximation (SA) as follows:

$$\hat{Q}^n \leftarrow \hat{C}\left(s^n, a^n\right) + \gamma \max_{a' \in A(s^{n+1})} \left(\bar{Q}^{n-1}\left(s^{n+1}, a'\right)\right) \tag{1}$$

$$\bar{Q}^n\left(s^n, a^n\right) \leftarrow \left(1 - \alpha_{n-1}\left(s^n, a^n\right)\right) \bar{Q}^{n-1}\left(s^n, a^n\right) + \alpha_{n-1}\left(s^n, a^n\right) \tag{2}$$

where (s^n, a^n) is a determined state-action pair, and s^{n+1} is a realization of random state transition due to taking action a^n in state s^n (we defer the detailed definition of other terms). Also, when the $\bar{Q}^n$ estimate is represented in tabular format, Q-learning enjoys asymptotic convergence properties with a mild set of technical assumptions as demonstrated in [2], [3], and [4]. The assumptions in [2] allow many stochastic models for $\hat{C}$ and s^{n+1} given s^n, a^n that can be applied to Q-learning with its convergence guarantee. Moreoever, the asymptotic rate of convergence of Q-learning has been studied theoretically by a number of authors including [5], [6], and [7]. Thanks to its generally applicable set of assumptions and robust theoretical properties, Q-learning has been applied to a wide

… ongoing research.

Approximate dynamic programming

Algorithms

Algorithms

- Approximate value iteration
  - » Single-pass
    - Need to define policies for choosing states and actions
  - » Double-pass with "discount" λ

- Approximate policy iteration

- Relationship to TD-learning
  - » Approximate value iteration uses TD(0) updates.
  - » Approximate policy iteration uses TD(1) updates.

Approximate value iteration

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

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

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

to obtain x^n.

Deterministic optimization

Step 3: Update the value function approximation

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

Recursive statistics

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

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

Simulation

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:

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

to obtain x^n.

Deterministic optimization

Linear model for post-decision state

Step 3: Update the value function approximation

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

Recursive statistics

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

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

"Trajectory following"

Simulation

Step 5: Return to step 1.

Approximate policy iteration

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

Step 2: Inner loop: Do for m=1,…,M:

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

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

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

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:

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

to obtain x^m.

Step 2b: Update the value function approximation using recursive least squares.

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

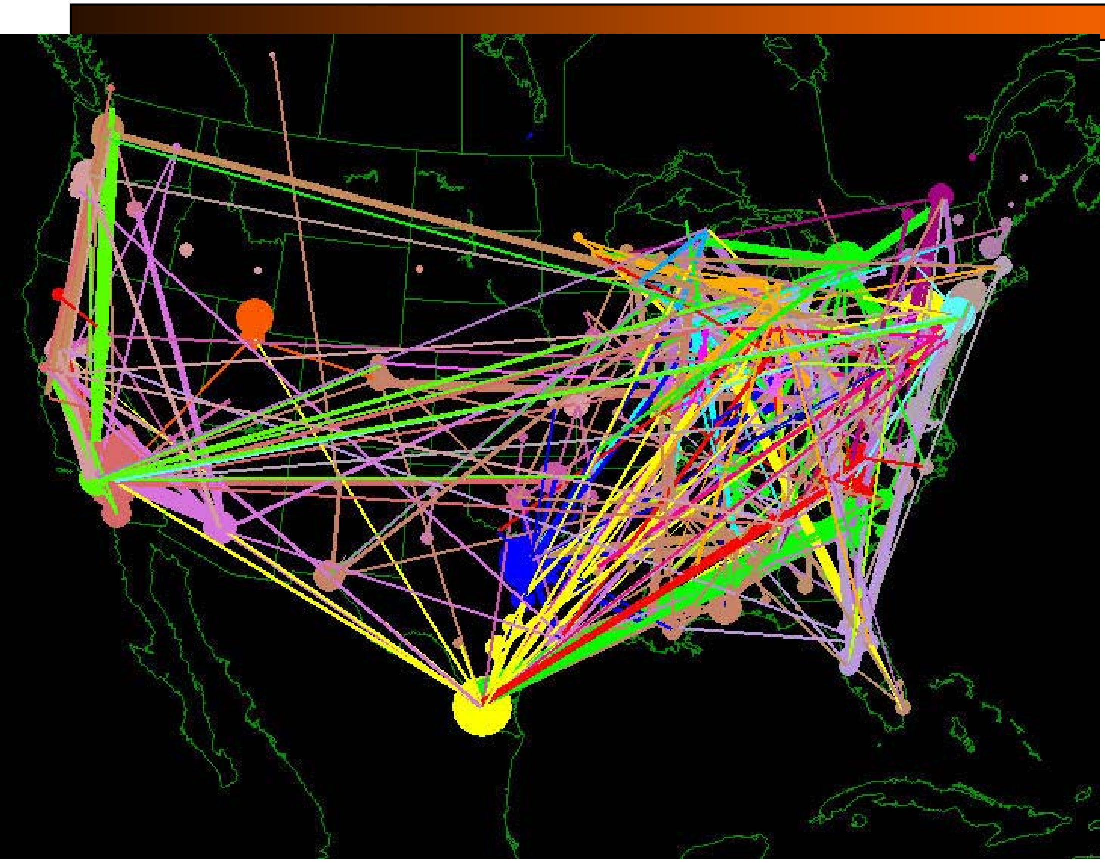
$$S^{m+1} = S^M(S^m, x^m, W(\omega^m))$$

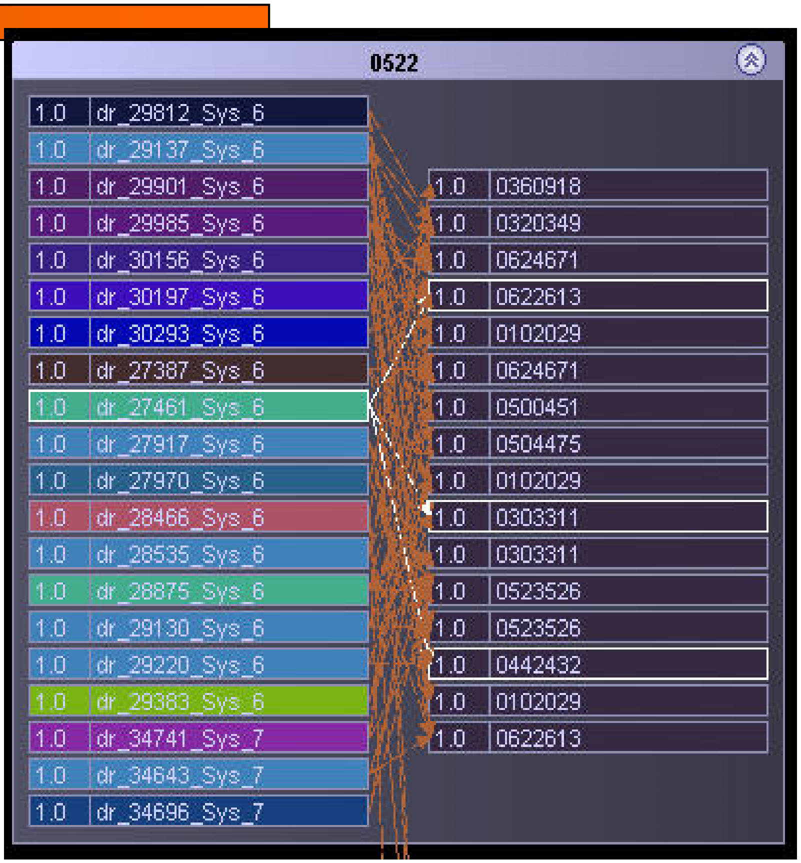
Step 3: Update $\bar{V}^n(S)$ using $\bar{V}^{n-1,M}(S)$ and return to step 1.

Approximate dynamic programming

Nomadic trucker problem
Approximate value iteration

Fleet management





0522

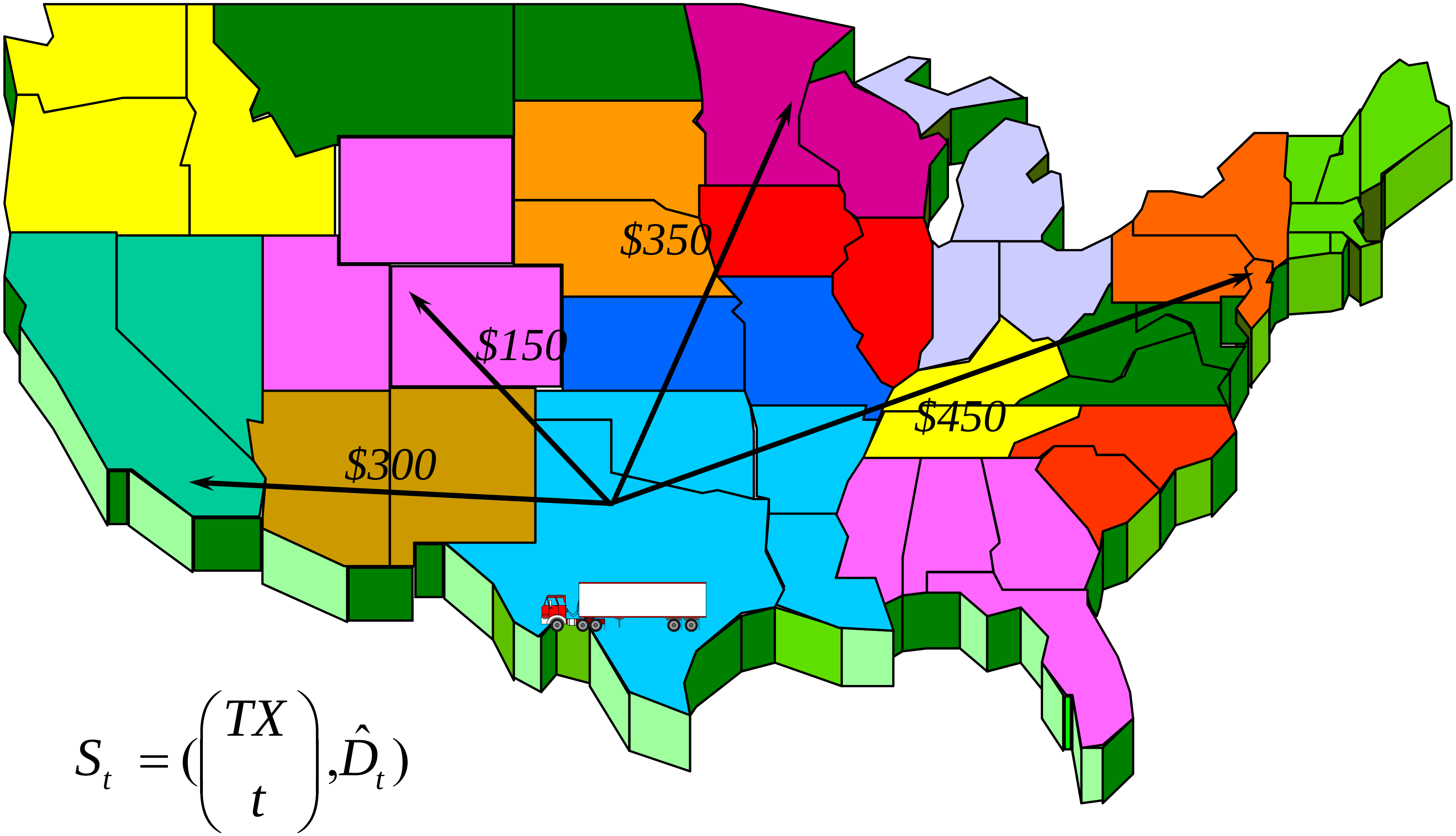
1.0	dr_29812_Sys_6
1.0	dr_29137_Sys_6
1.0	dr_29901_Sys_6
1.0	dr_29985_Sys_6
1.0	dr_30156_Sys_6
1.0	dr_30197_Sys_6
1.0	dr_30293_Sys_6
1.0	dr_27387_Sys_6
1.0	dr_27461_Sys_6
1.0	dr_27917_Sys_6
1.0	dr_27970_Sys_6
1.0	dr_28466_Sys_6
1.0	dr_28535_Sys_6
1.0	dr_28875_Sys_6
1.0	dr_29130_Sys_6
1.0	dr_29220_Sys_6
1.0	dr_29383_Sys_6
1.0	dr_34741_Sys_7
1.0	dr_34643_Sys_7
1.0	dr_34696_Sys_7

1.0	0360918
1.0	0320349
1.0	0624671
1.0	0622613
1.0	0102029
1.0	0624671
1.0	0500451
1.0	0504475
1.0	0102029
1.0	0303311
1.0	0303311
1.0	0523526
1.0	0523526
1.0	0442432
1.0	0102029
1.0	0622613

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

Approximate dynamic programming

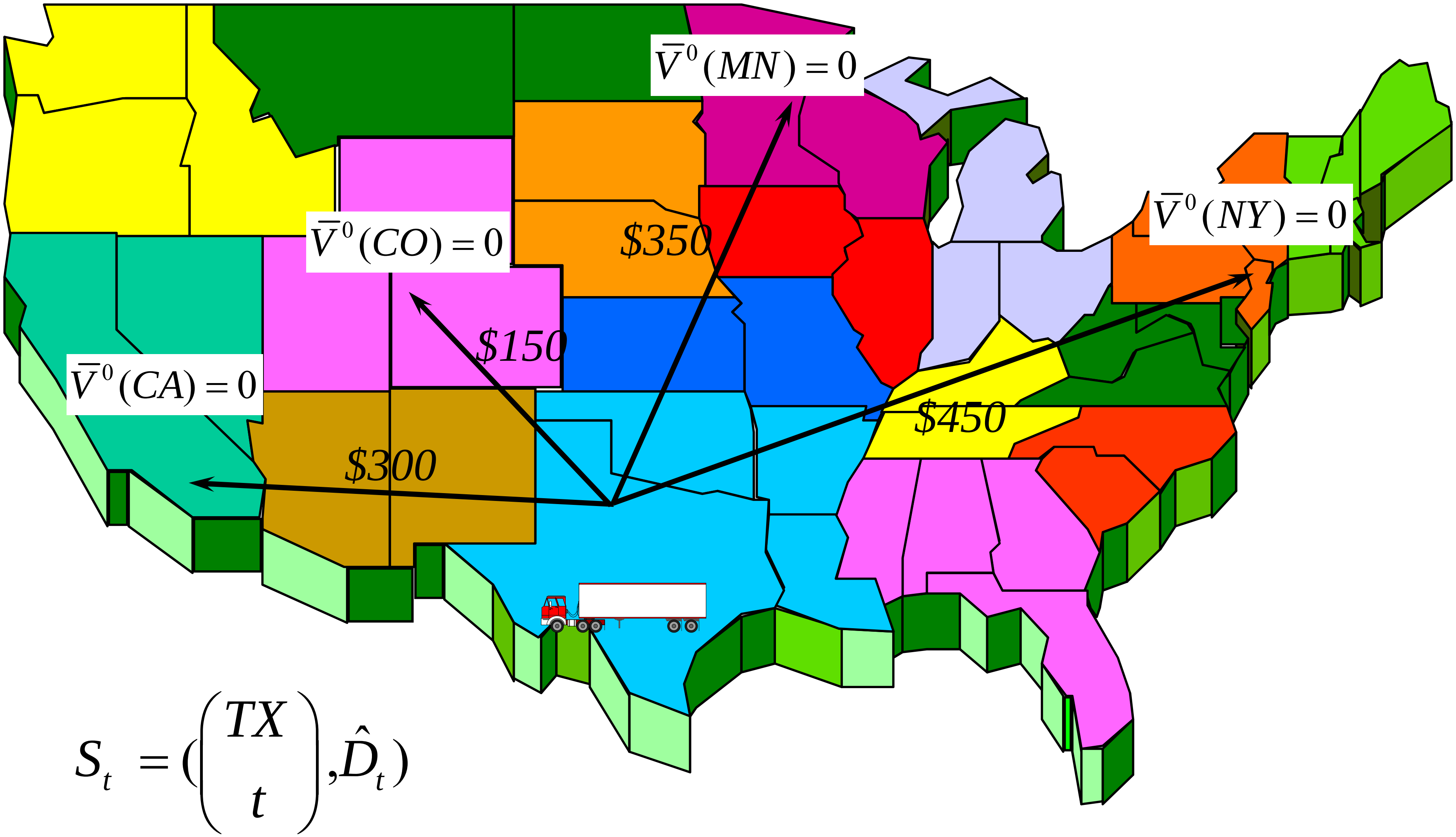
- Pre-decision state: we see the demands



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

Approximate dynamic programming

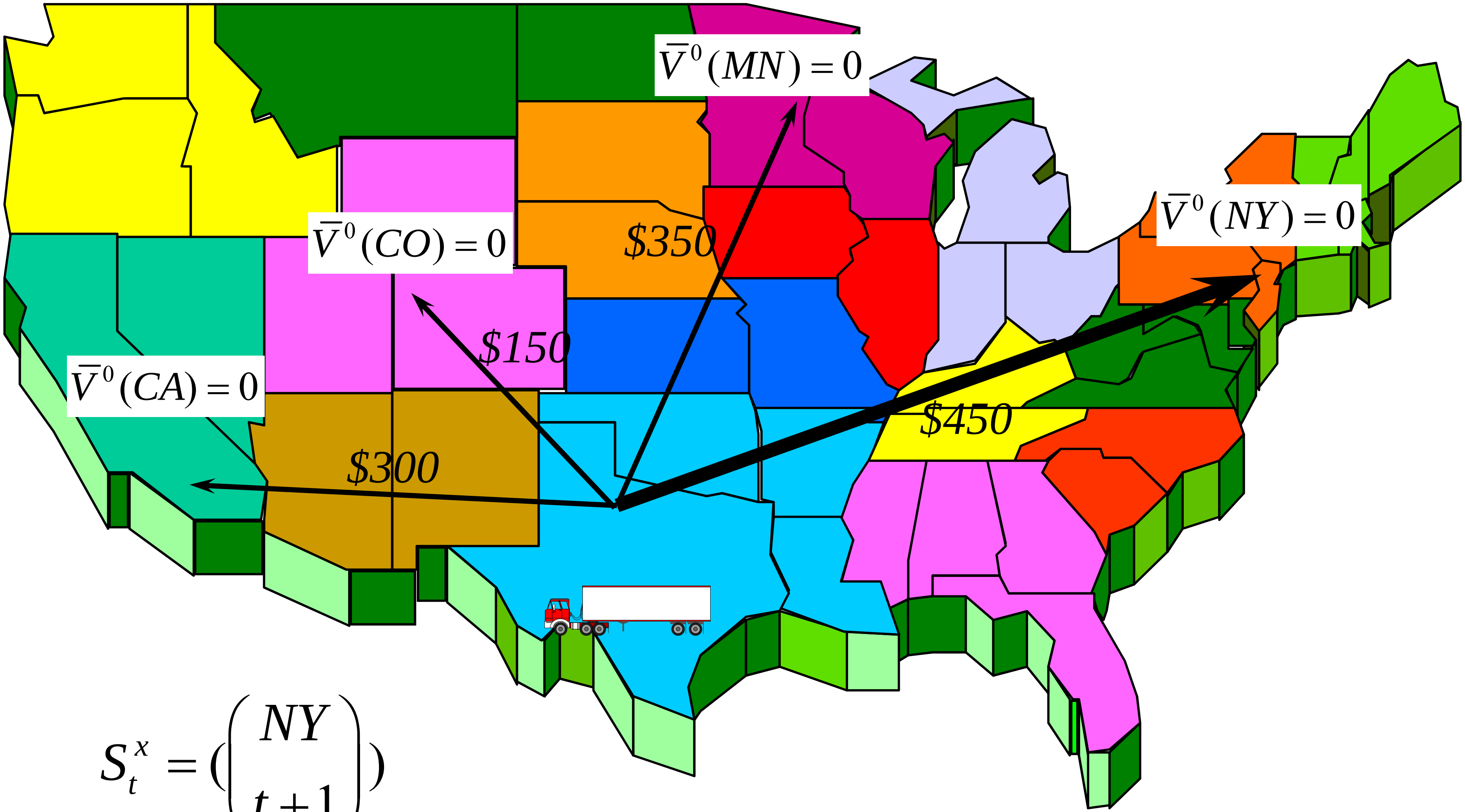
- We use initial value function approximations…



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

Approximate dynamic programming

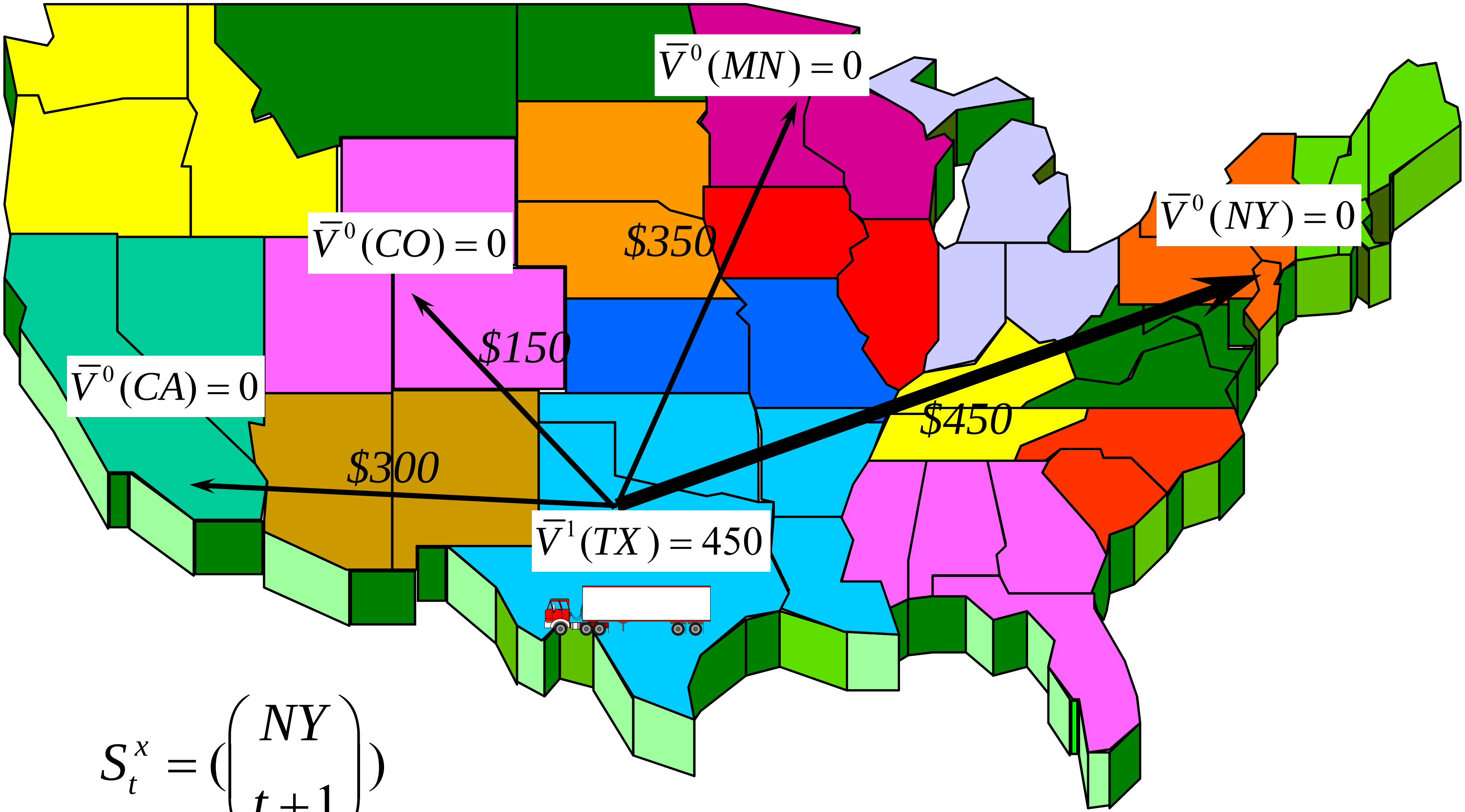
- … and make our first choice: x^1



$$S_t^x = \left(\begin{pmatrix} NY \\ t+1 \end{pmatrix}\right)$$

Approximate dynamic programming

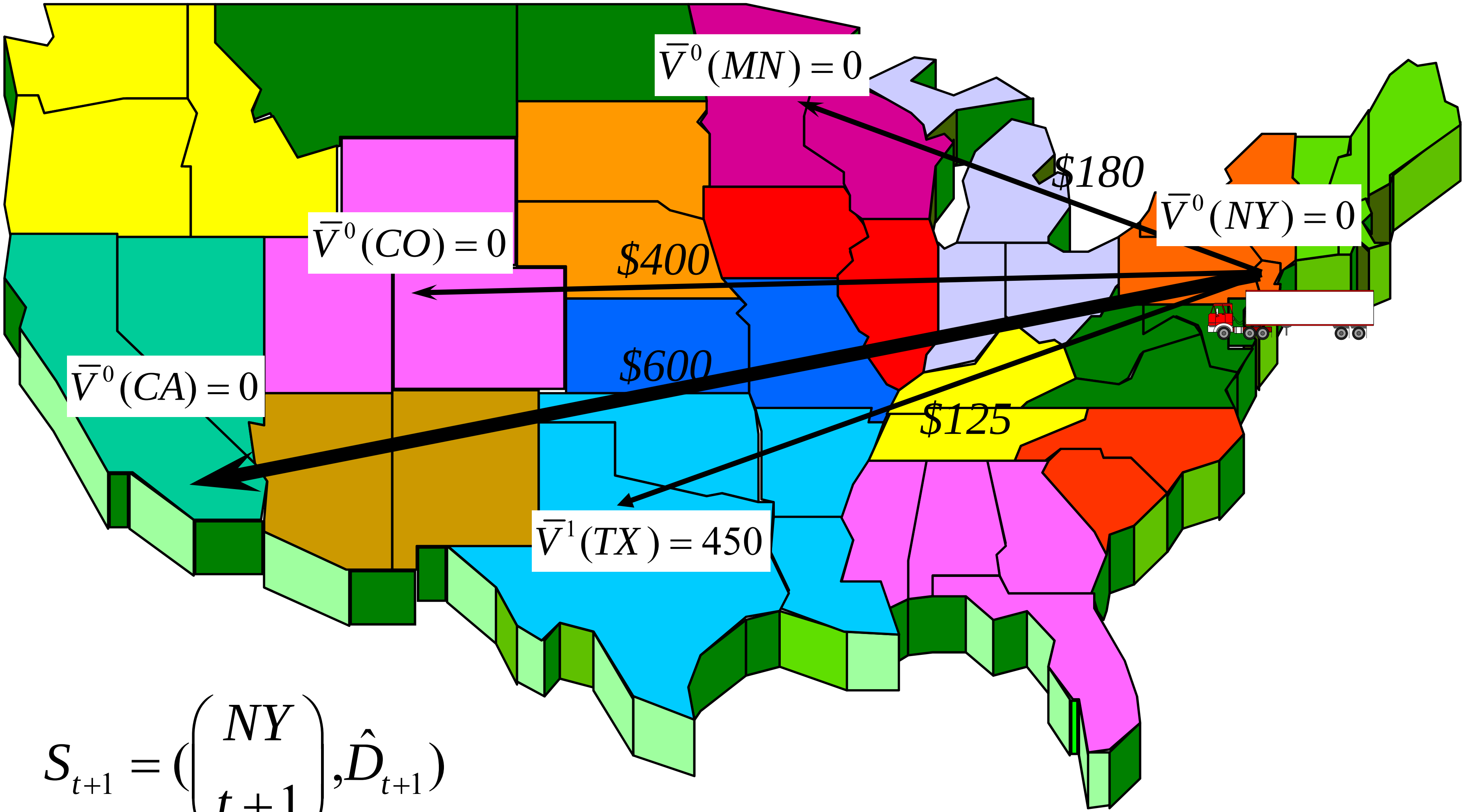
- Update the value of being in Texas.



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

Approximate dynamic programming

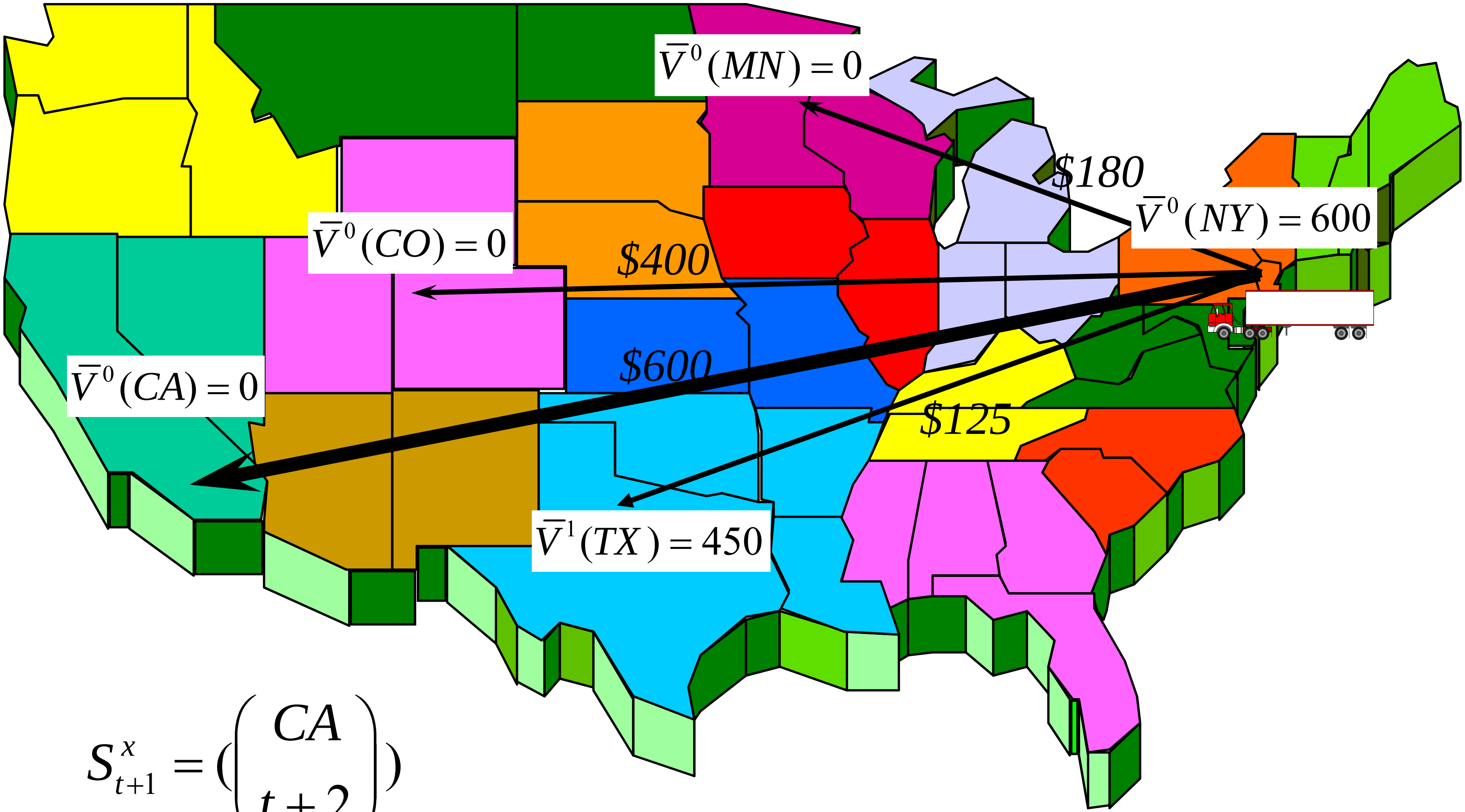
- Now move to the next state, sample new demands and make a new decision



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

Approximate dynamic programming

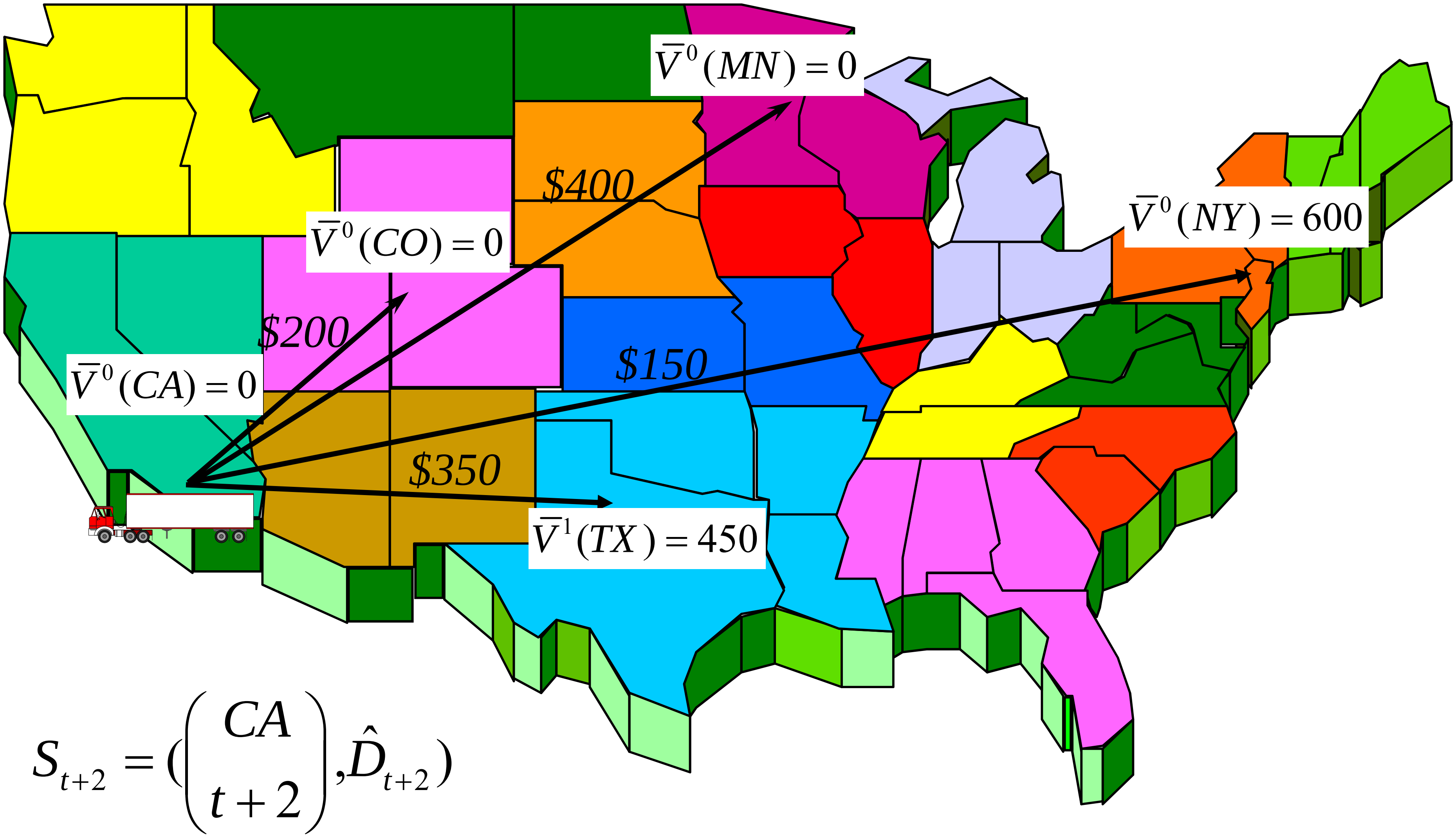
- Update value of being in NY



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

Approximate dynamic programming

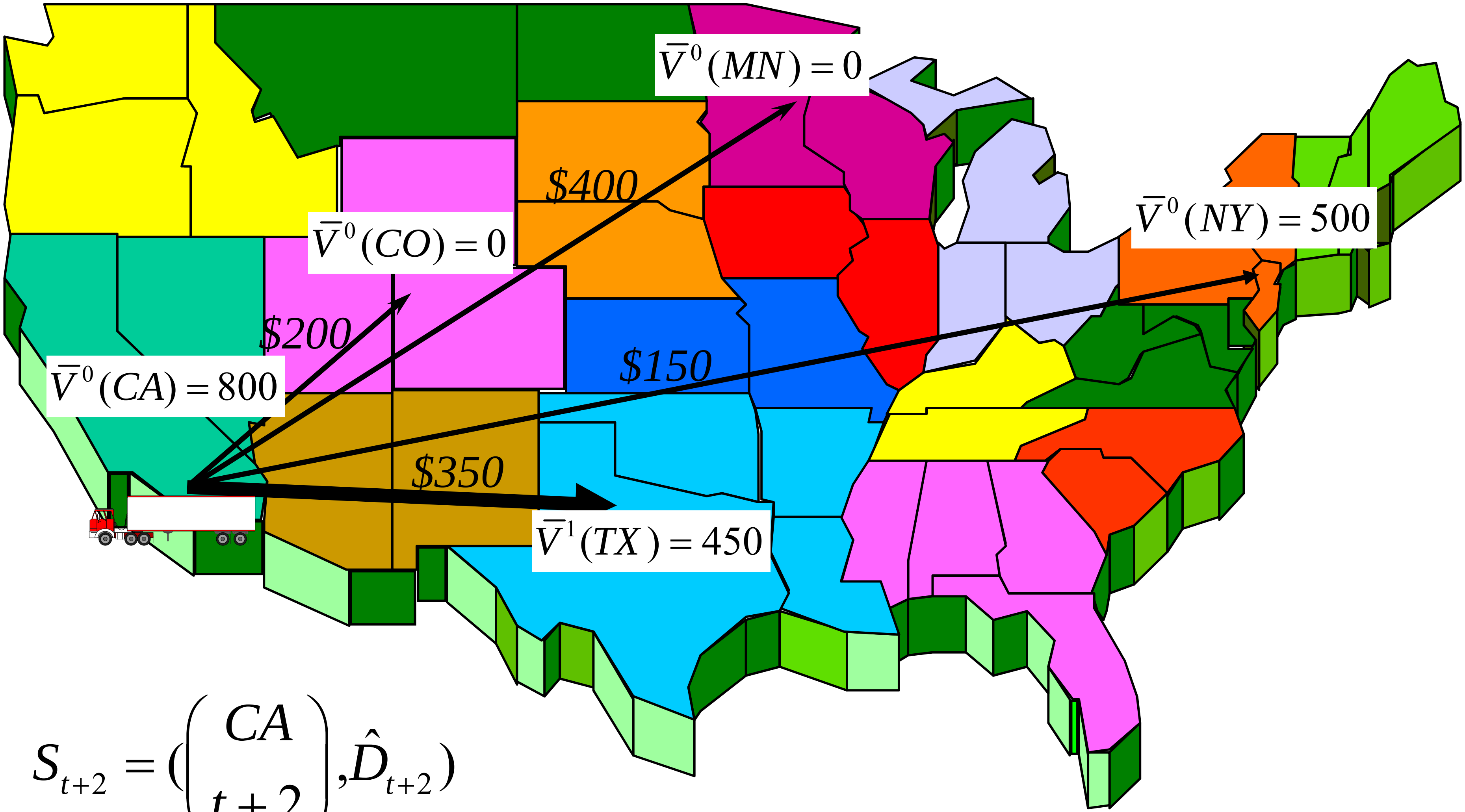
- Move to California.



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

Approximate dynamic programming

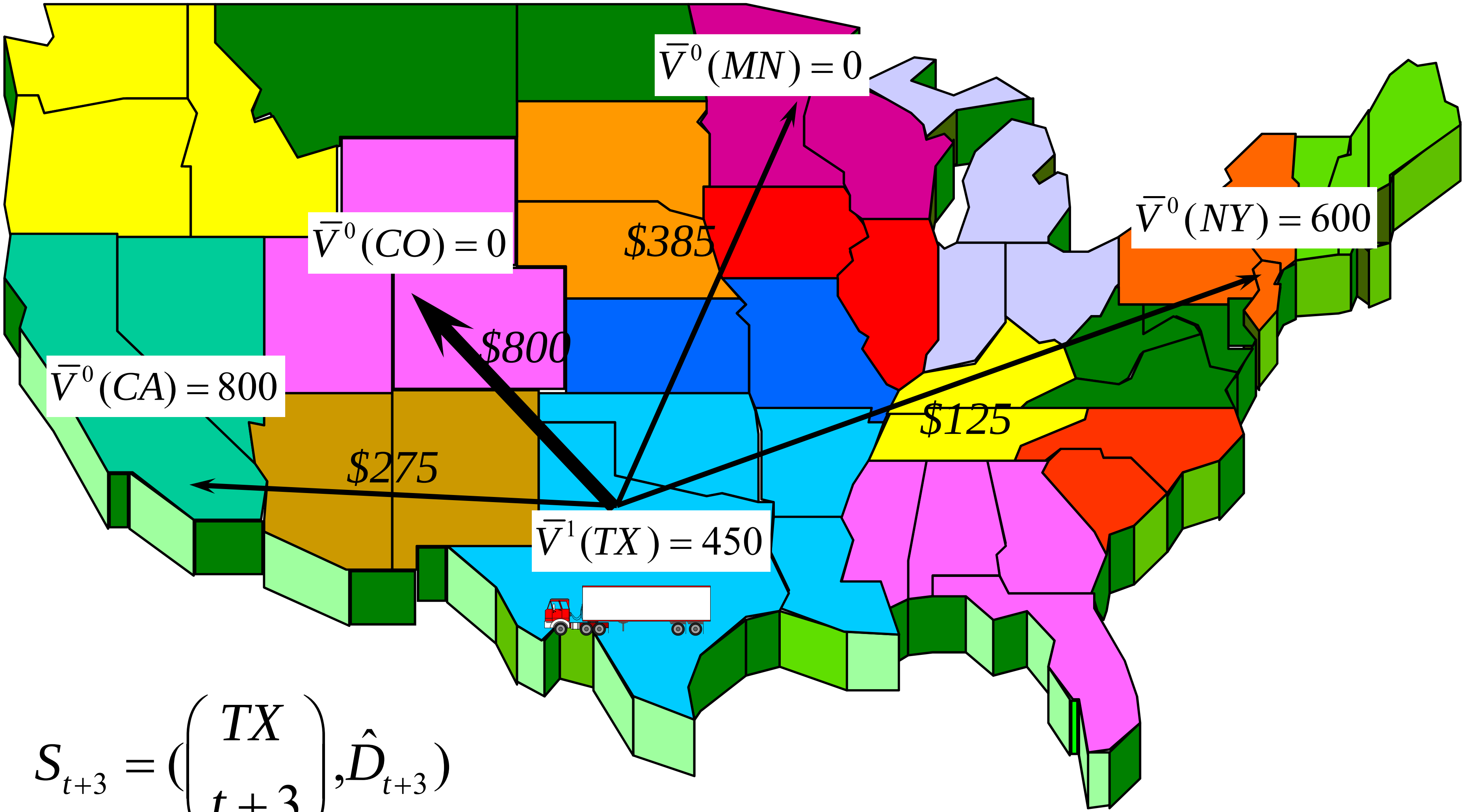
- Make decision to return to TX and update value of being in CA



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

Approximate dynamic programming

- An updated value of being in TX



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

Approximate dynamic programming

- Updating the value function:

Old value:

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

New estimate:

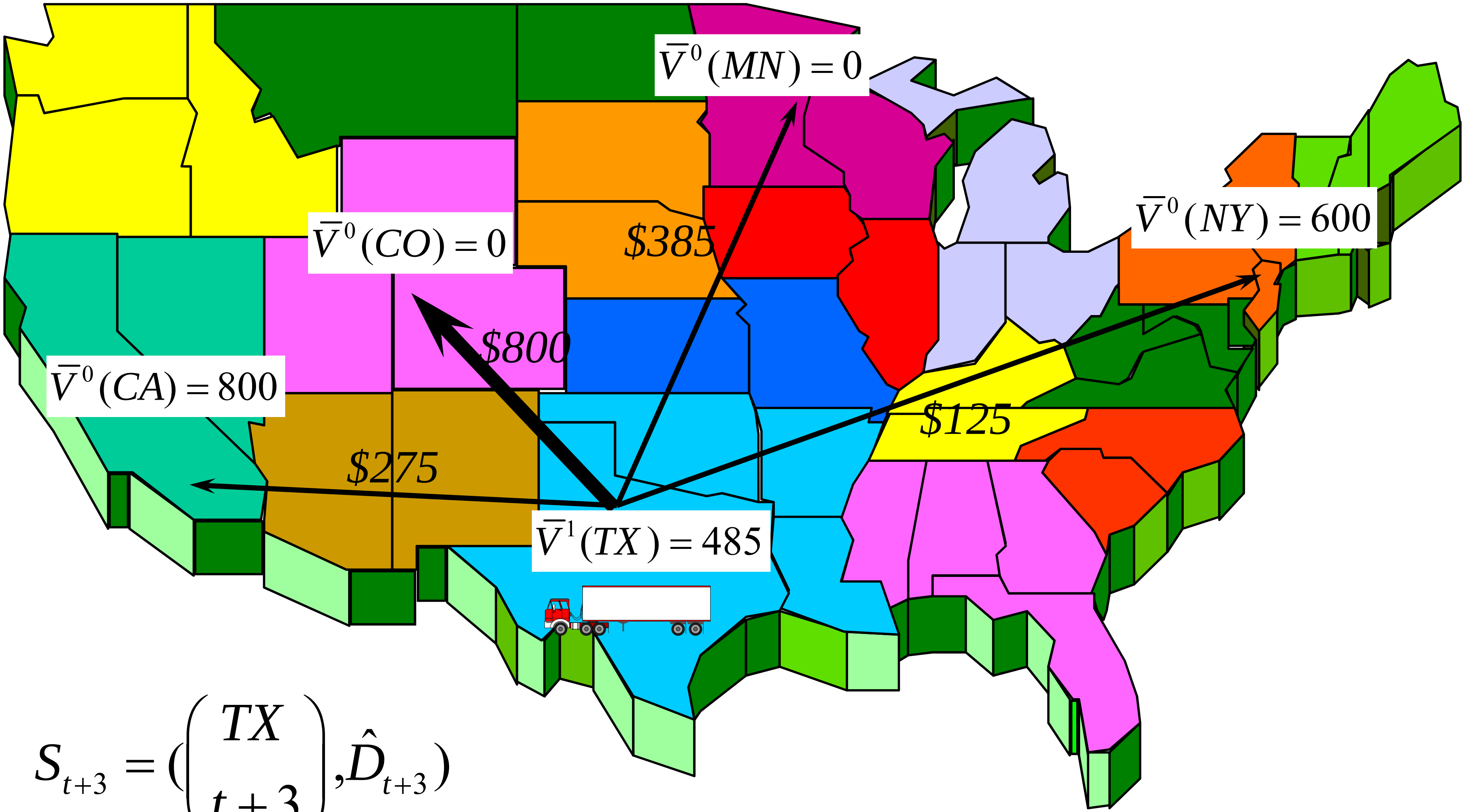
$$\hat{v}^2(TX) = \$800$$

How do we merge old with new?

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

Approximate dynamic programming

- An updated value of being in TX

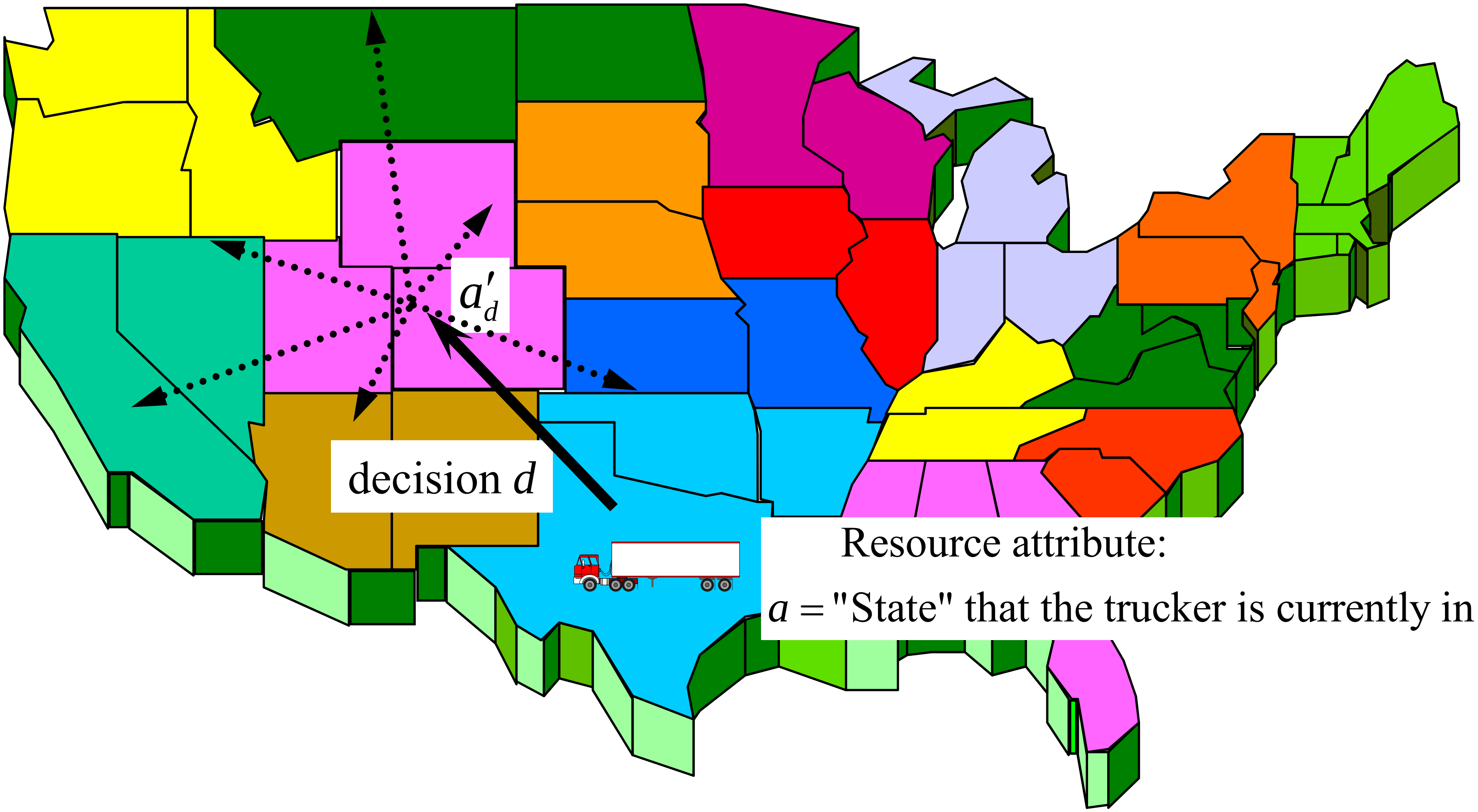


$$S_{t+3} = (\begin{pmatrix} TX \\ t+3 \end{pmatrix}, \hat{D}_{t+3})$$

Approximate dynamic programming

Hierarchical learning

Hierarchical learning



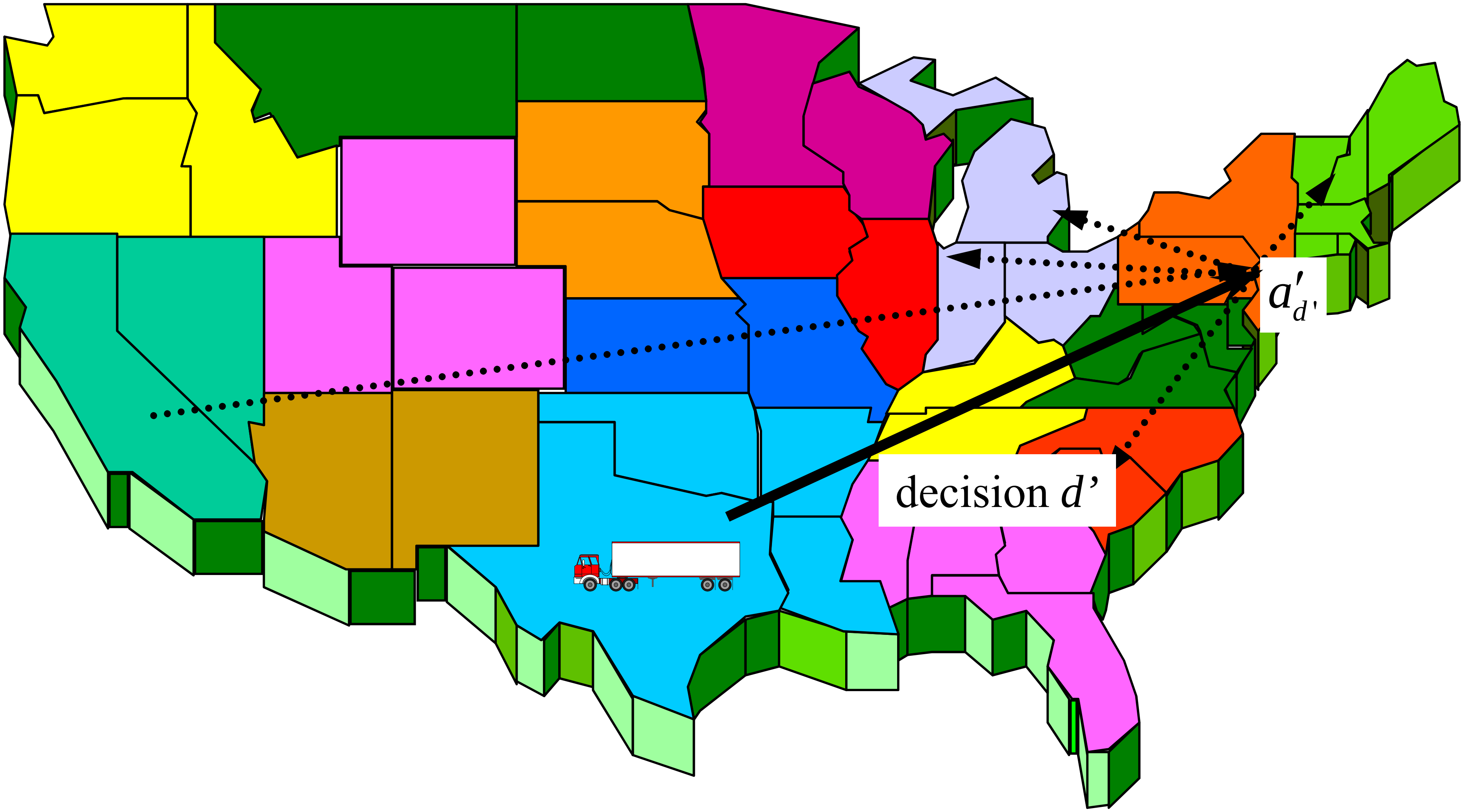
a'_d

decision d

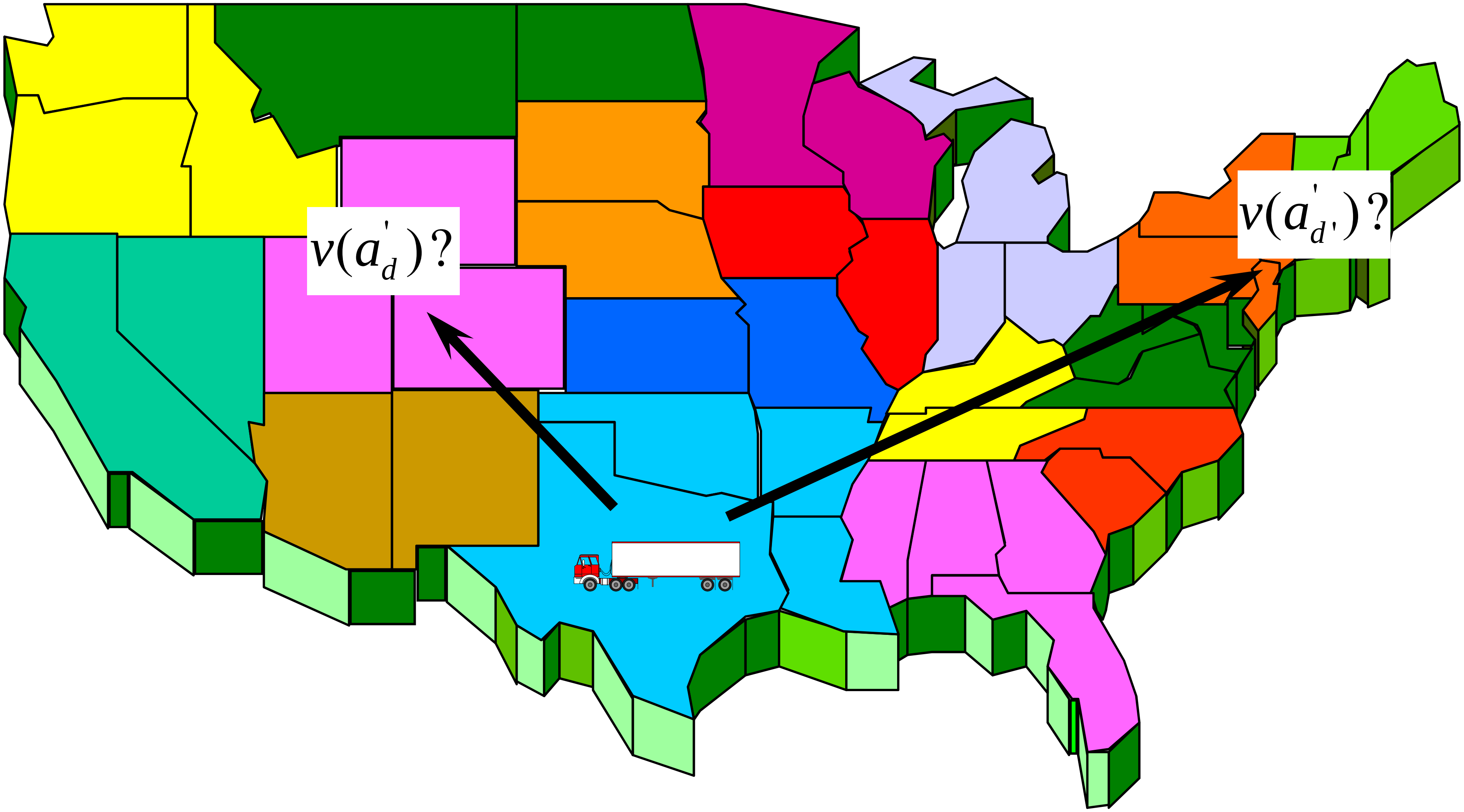
Resource attribute:

$a =$ "State" that the trucker is currently in

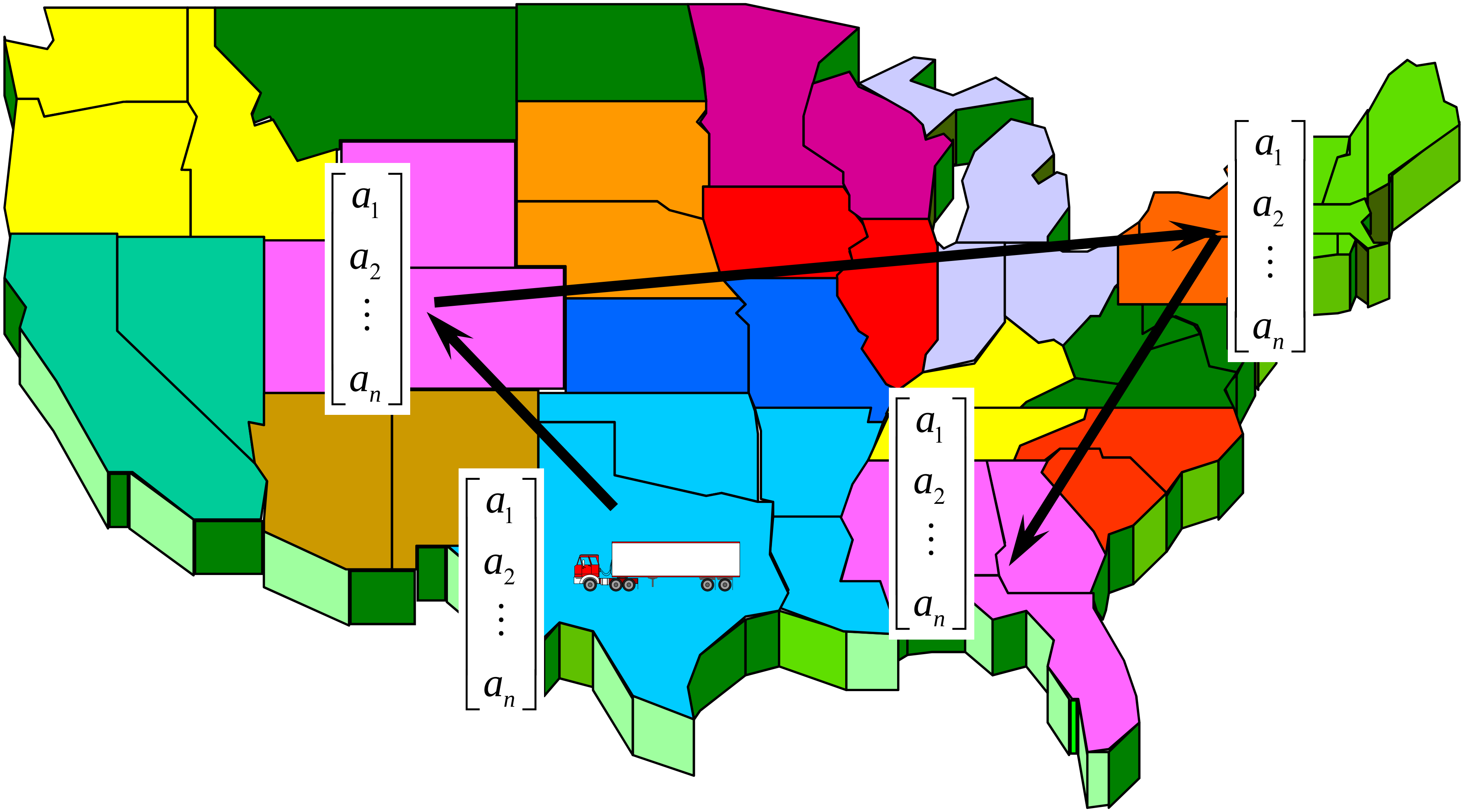
Hierarchical learning



Hierarchical learning



Hierarchical learning


$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$
$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$
$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$
$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$

Hierarchical learning

- Our optimization problem at time t looks like:

$$V_t(S_t) = \max_x \left(C_t(S_t, x_t) + \sum_{a\in\mathcal{A}} \bar{v}_{ta} \cdot R^x_{ta} \right)$$

There are a lot of these attributes!

» We had to develop novel machine learning strategies to estimate this function, since the attribute space was very large.

Hierarchical learning

- Different levels of aggregation:

$$a = \begin{bmatrix}\text{Time}\\ \text{Region Location}\\ \text{Region Domicile}\\ \text{Type}\end{bmatrix}\begin{bmatrix}\text{Time}\\ \text{Region Location}\\ \text{Type}\end{bmatrix}\begin{bmatrix}\text{Time}\\ \text{Region Location}\end{bmatrix}\begin{bmatrix}\text{Time}\\ \text{Area Location}\end{bmatrix}$$

$$\left|\mathcal{A}\right| \approx \qquad \boldsymbol{\mathit{3{,}293{,}136}} \qquad \boldsymbol{\mathit{33{,}264}} \qquad \boldsymbol{\mathit{5{,}544}} \qquad \boldsymbol{\mathit{672}}$$

Hierarchical learning

- Estimating value functions
  - » Most aggregate level

$$\bar{v}^{n}([Location]) = (1-\alpha)\bar{v}^{n-1}([Location]) + \alpha\hat{v}(\begin{bmatrix} Location \\ Fleet \\ Domicile \\ DOThrs \\ DaysFromHome \end{bmatrix})$$

Hierarchical learning

- Estimating value functions
  - » Middle level of aggregation

$$\bar{v}^n\left(\begin{bmatrix} Location \\ Fleet \end{bmatrix}\right) = (1-\alpha)\bar{v}^{n-1}\left(\begin{bmatrix} Location \\ Fleet \end{bmatrix}\right) + \alpha\hat{v}\left(\begin{bmatrix} Location \\ Fleet \\ Domicile \\ DOThrs \\ DaysFromHome \end{bmatrix}\right)$$

Hierarchical learning

- Estimating value functions
  - » Most disaggregate level

$$\bar{v}^n\left(\begin{bmatrix} Location \\ Fleet \\ Domicile \end{bmatrix}\right) = (1-\alpha)\bar{v}^{n-1}\left(\begin{bmatrix} Location \\ Fleet \\ Domicile \end{bmatrix}\right) + \alpha\hat{v}\left(\begin{bmatrix} Location \\ Fleet \\ Domicile \\ DOThrs \\ DaysFromHome \end{bmatrix}\right)$$

Hierarchical learning

- Adaptive hierarchical estimation procedure developed as part of this project (George, Powell and Kulkarni, 2008)
  - » Use weighted sum across different levels of aggregation.

$$\bar{v}_a = \sum_g w_a^{(g)} \bar{v}_a^{(g)} \qquad \sum_g w_a^{(g)} = 1$$

where

$$w_a^{(g)} \propto \left(Var\left(\bar{v}_a^{(g)} \right) + \left(\beta_a^{(g)} \right)^2 \right)^{-1}$$

Estimate of variance - $(\sigma_a^2)^{(g)}$

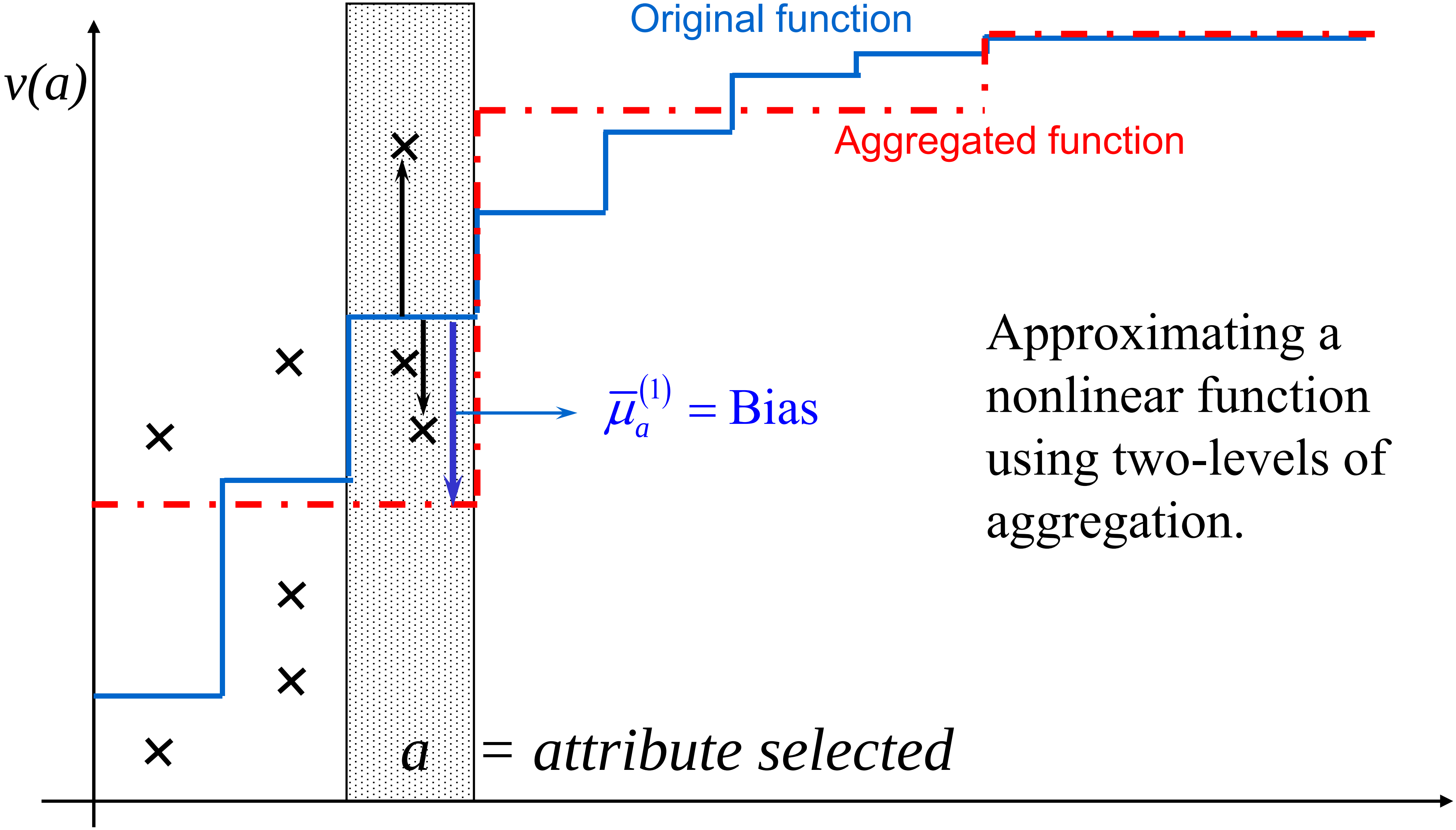
Estimate of bias

Both can be computed using simple recursive formulas.

George, A., W.B. Powell and S. Kulkarni, "Value Function Approximation Using Hierarchical Aggregation for Multiattribute Resource Management," Journal of Machine Learning Research, Vol. 9, pp. 2079-2111 (2008).

Hierarchical learning

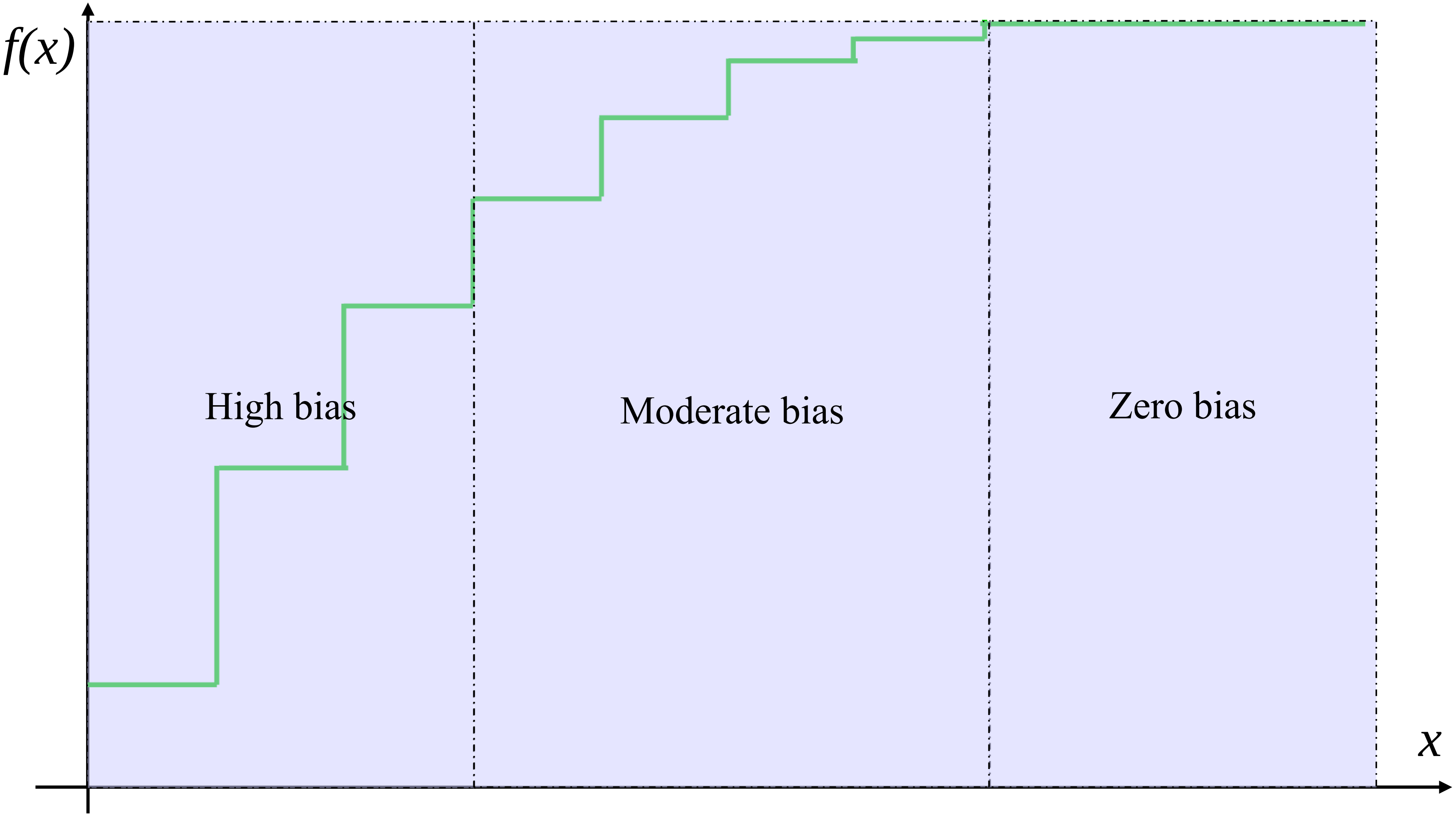
- Hierarchical aggregation



Approximating a nonlinear function using two-levels of aggregation.

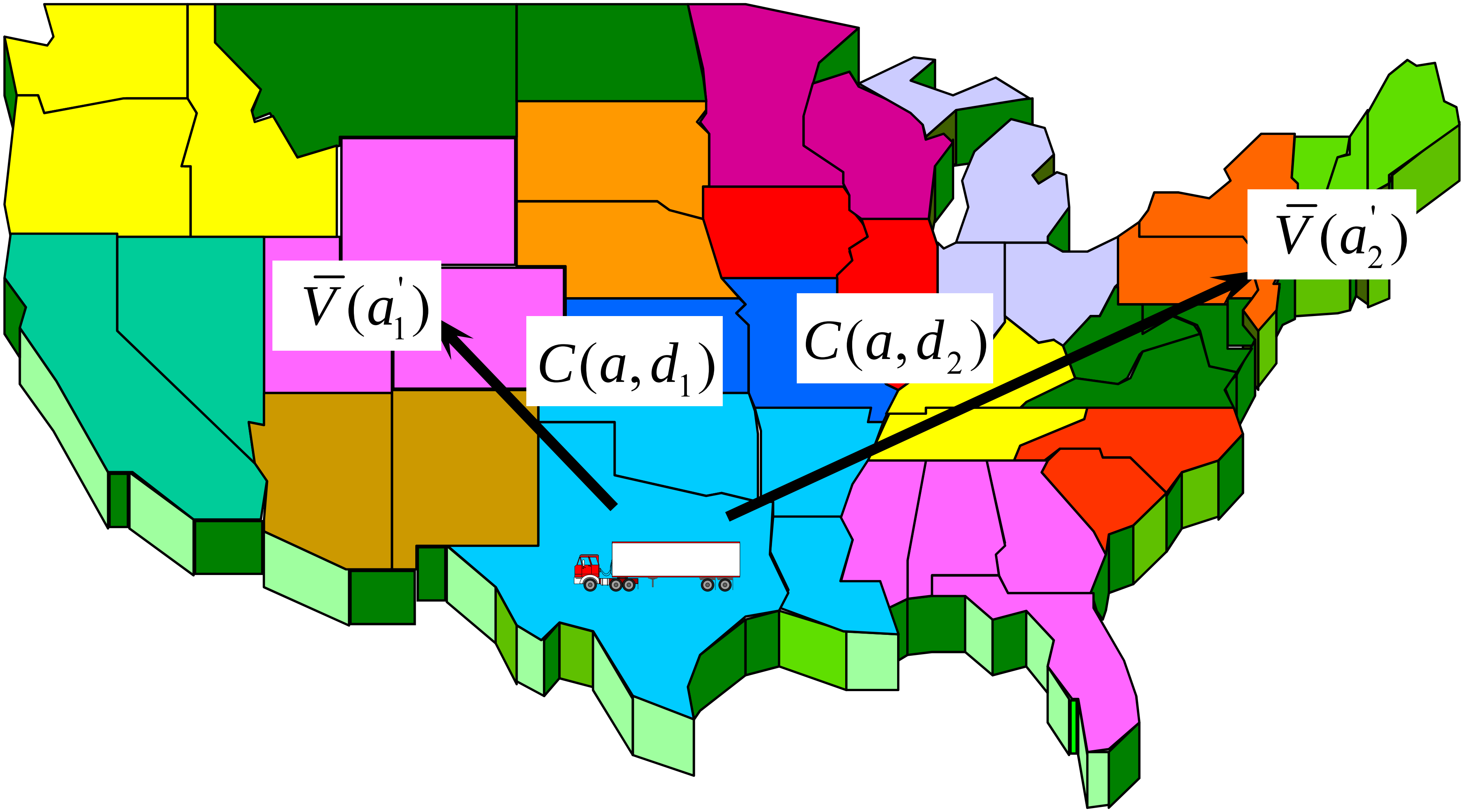
Hierarchical learning

- Hierarchical aggregation

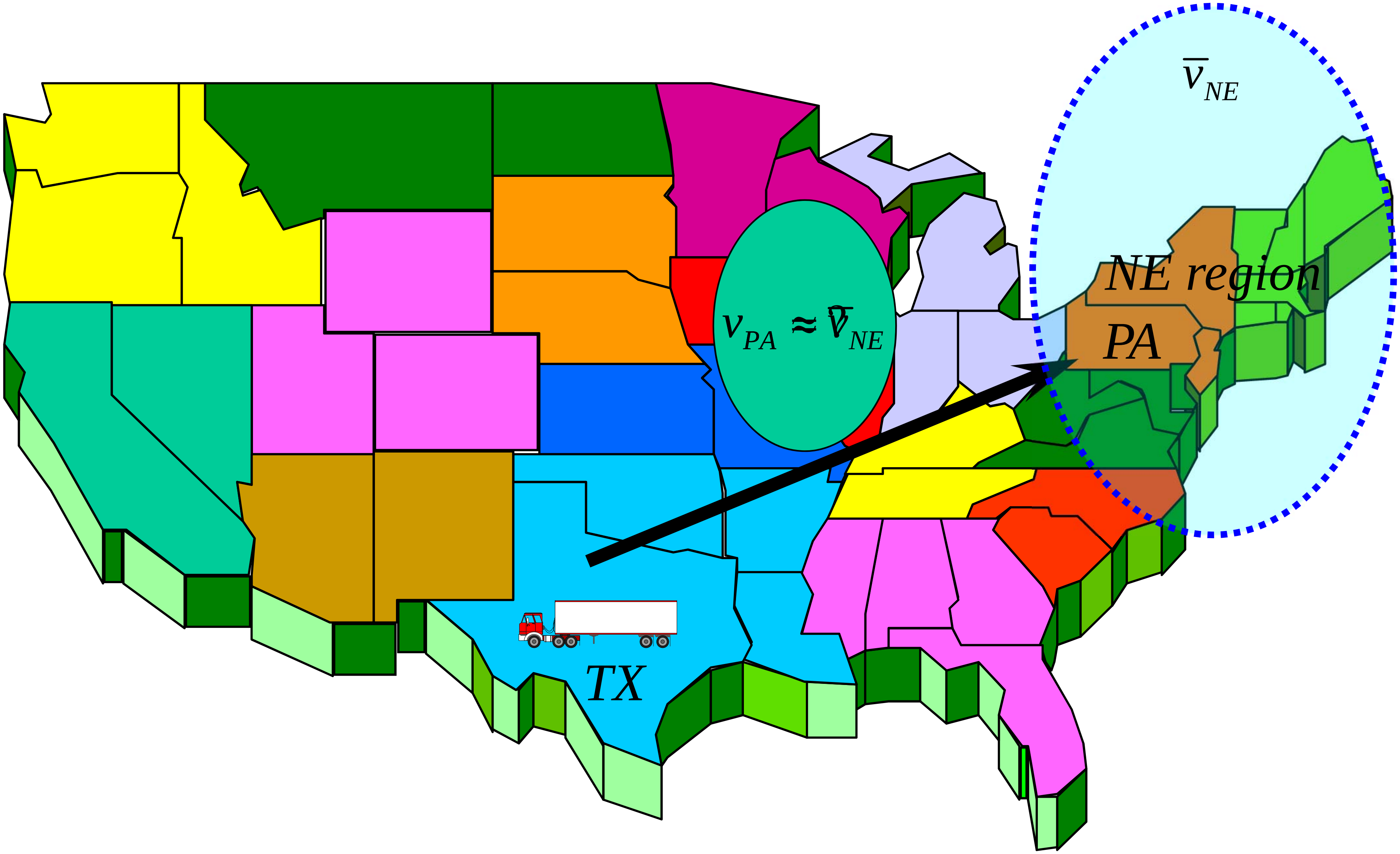


© 2019 Warren B. Powell

Hierarchical learning

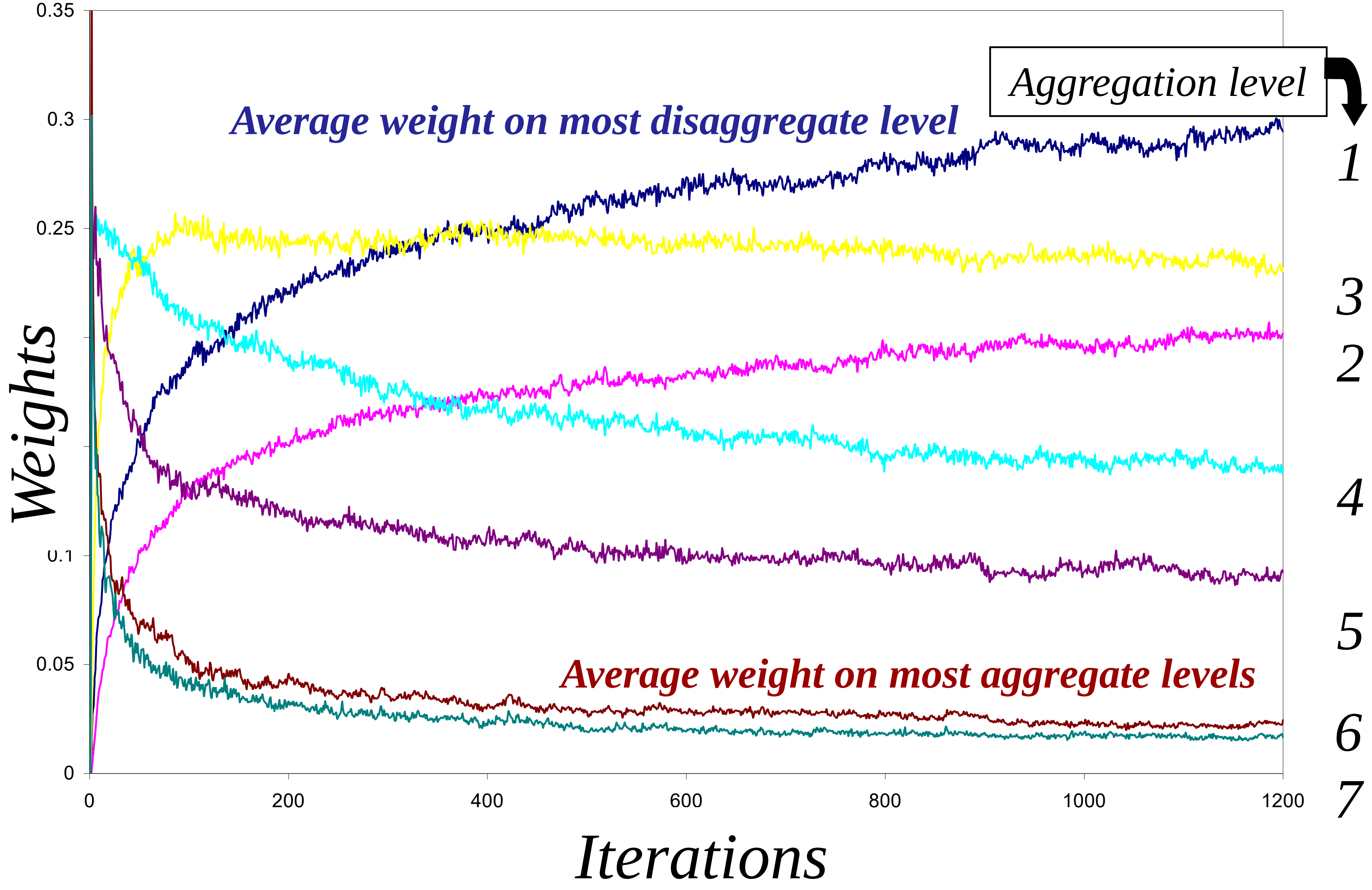


Hierarchical learning



Hierarchical learning

- Hierarchical aggregation

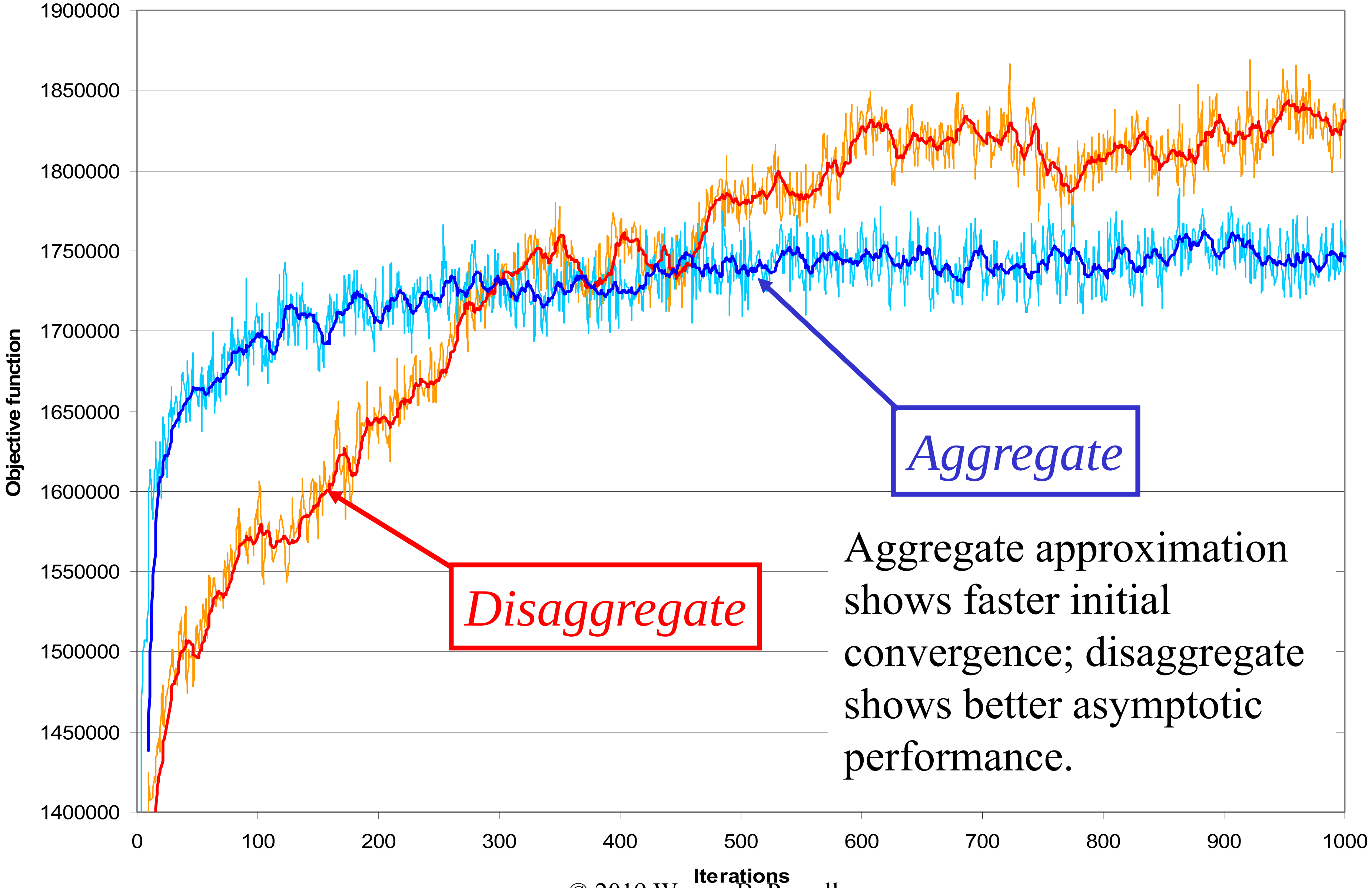


Hierarchical learning

- Notes:
  - » In the early iterations, we do not have enough data to provide estimates at the detail level.
  - » So, we put more weight on the most aggregate estimates.
  - » As the algorithm progresses and we gain more information, we can put more weight on the more disaggregate estimates.
  - » But the weights depend on how much data we have in different regions.
  - » This type of adaptive learning, from coarse-grained to fine-grained, is common across all learning problems in stochastic optimization.

Hierarchical learning

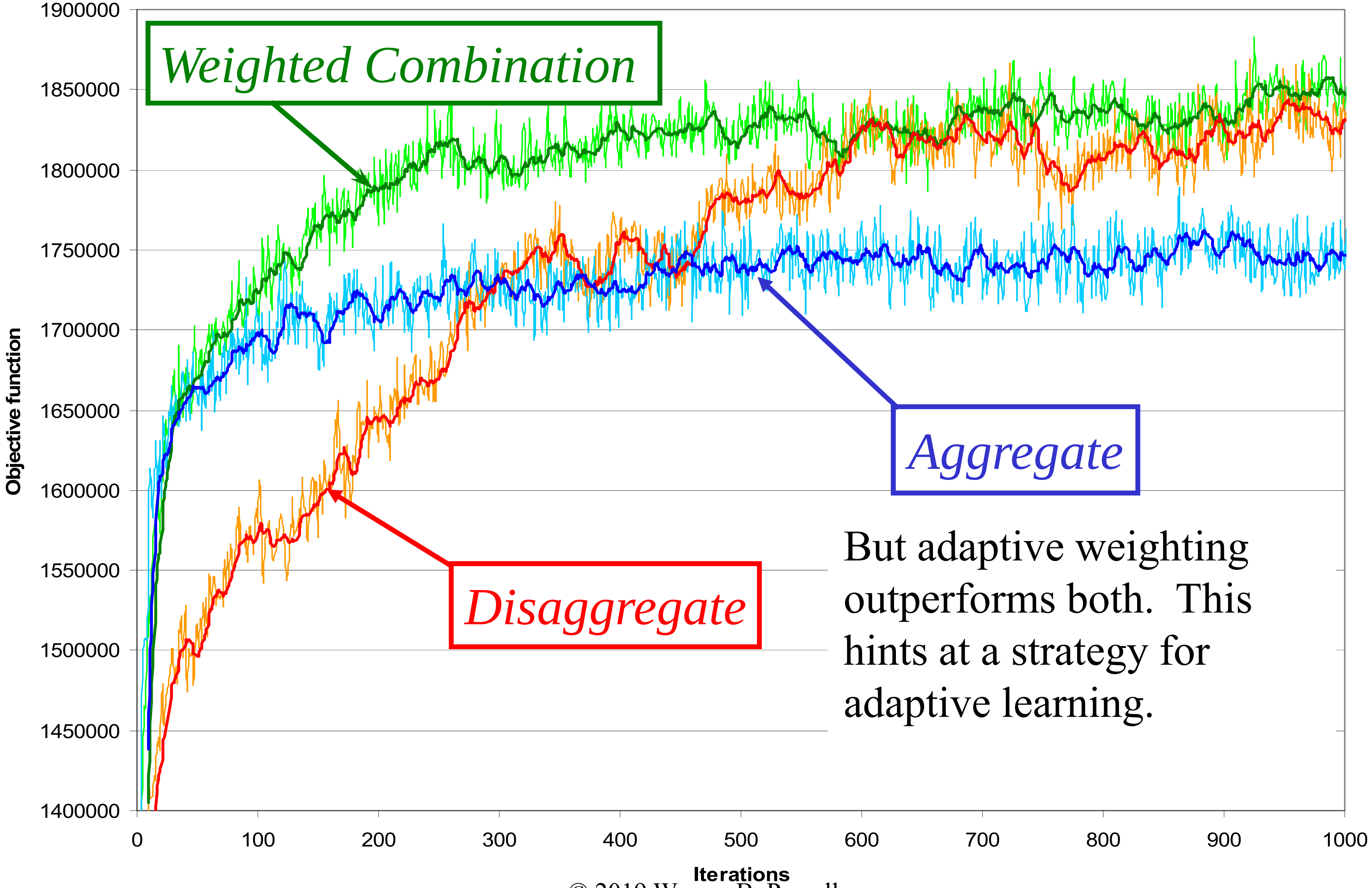
- Hierarchical aggregation



Aggregate approximation shows faster initial convergence; disaggregate shows better asymptotic performance.

Hierarchical learning

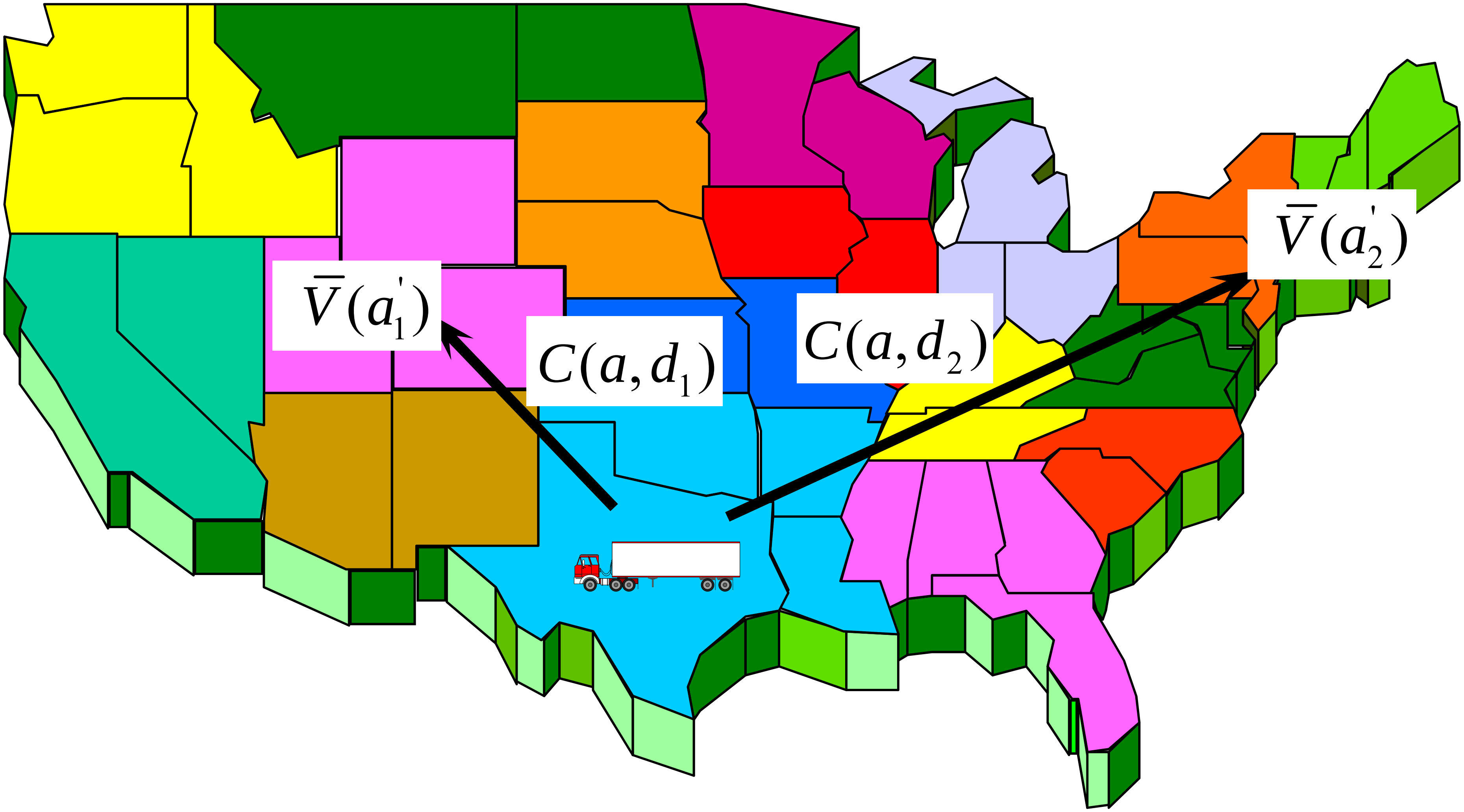
- Hierarchical aggregation



But adaptive weighting outperforms both. This hints at a strategy for adaptive learning.

The exploration-exploitation problem

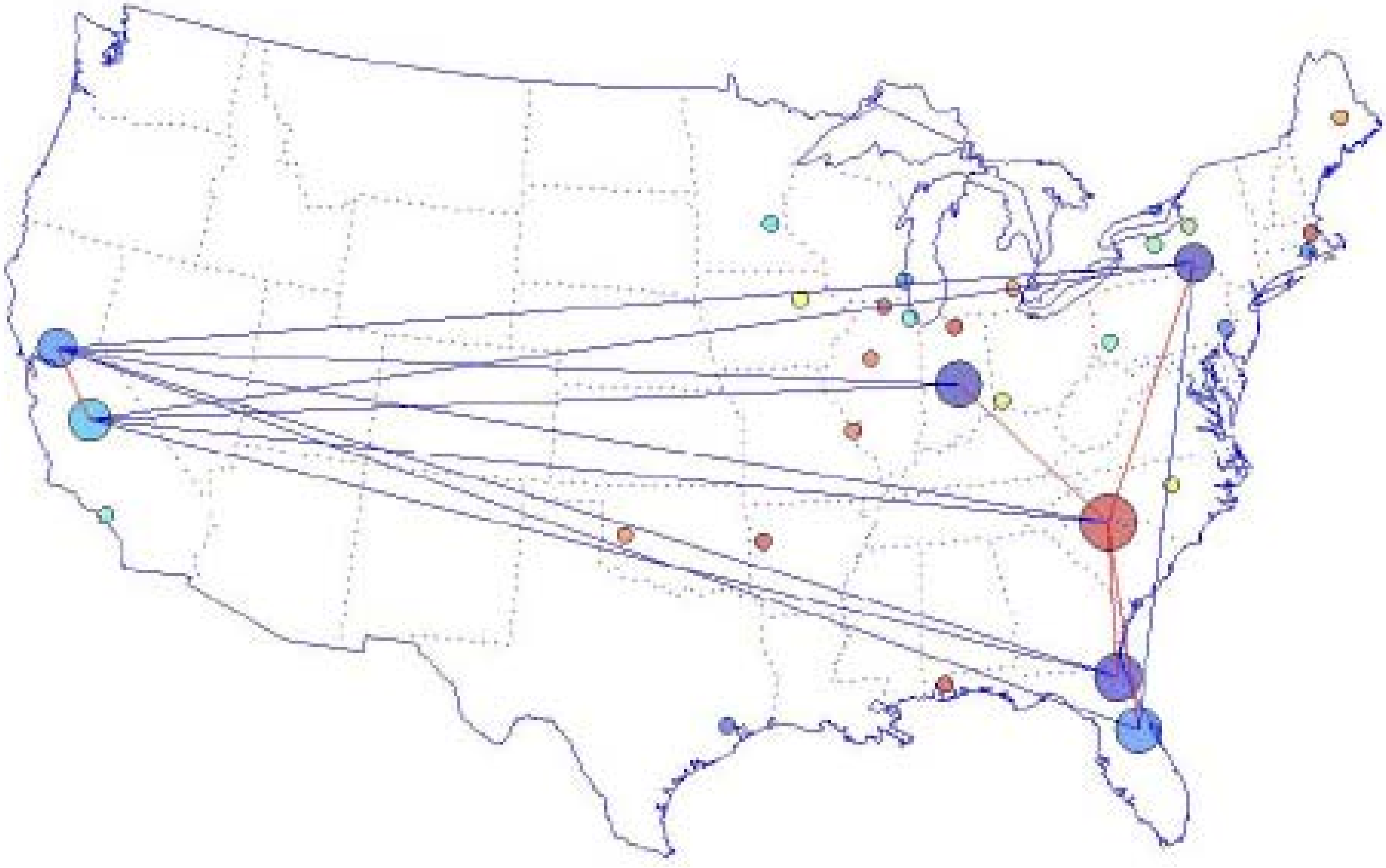
Exploration vs. exploitation



Exploration vs. exploitation

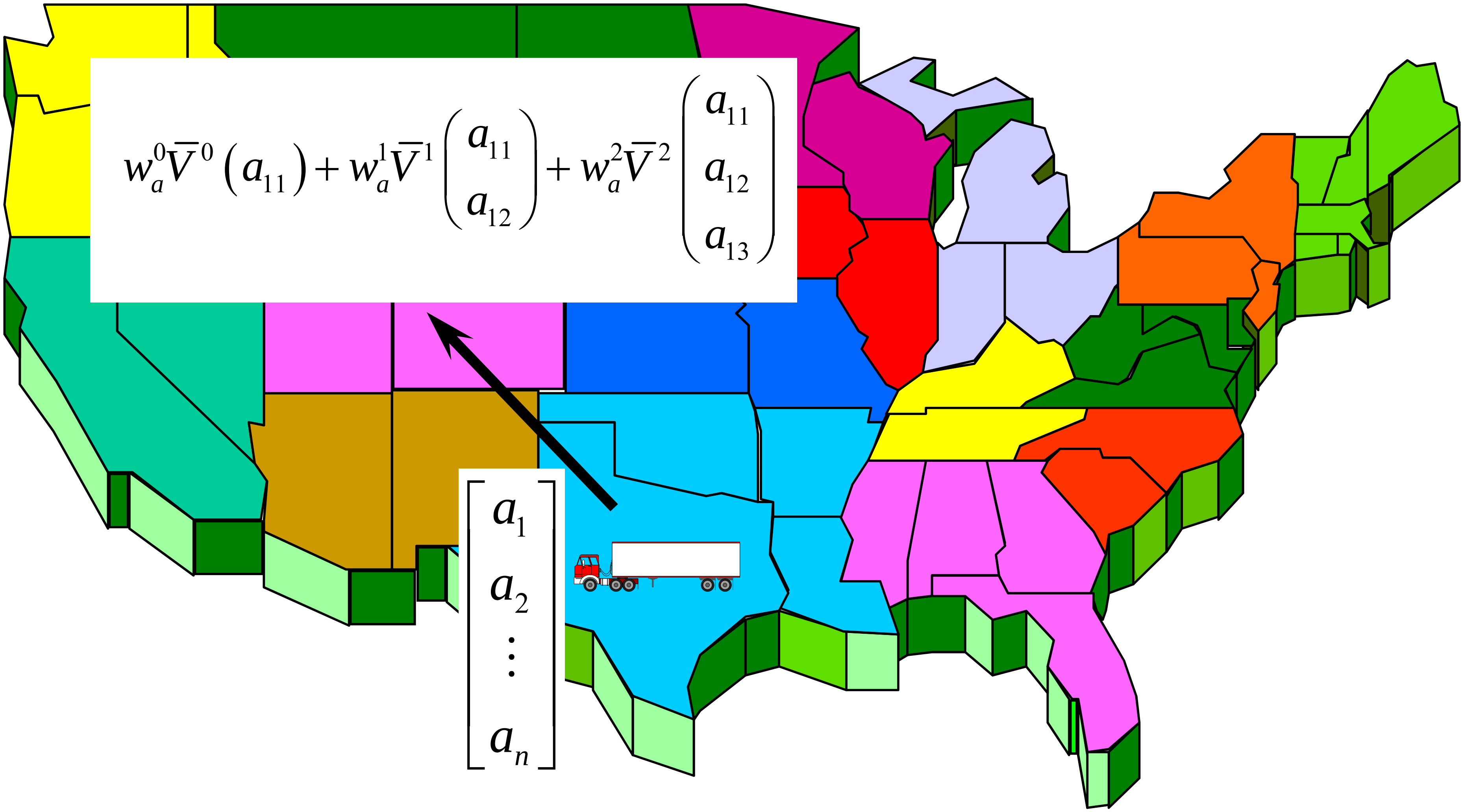
- What decision do we make?
  - » The one we think is best?
    - Exploitation
  - » Or do we make a decision just to try something and learn more about the result?
    - Exploration
  - » This is the reason that the "exploration vs. exploitation" problem is so well known in ADP/RL.

Exploration vs. exploitation



Pure exploitation

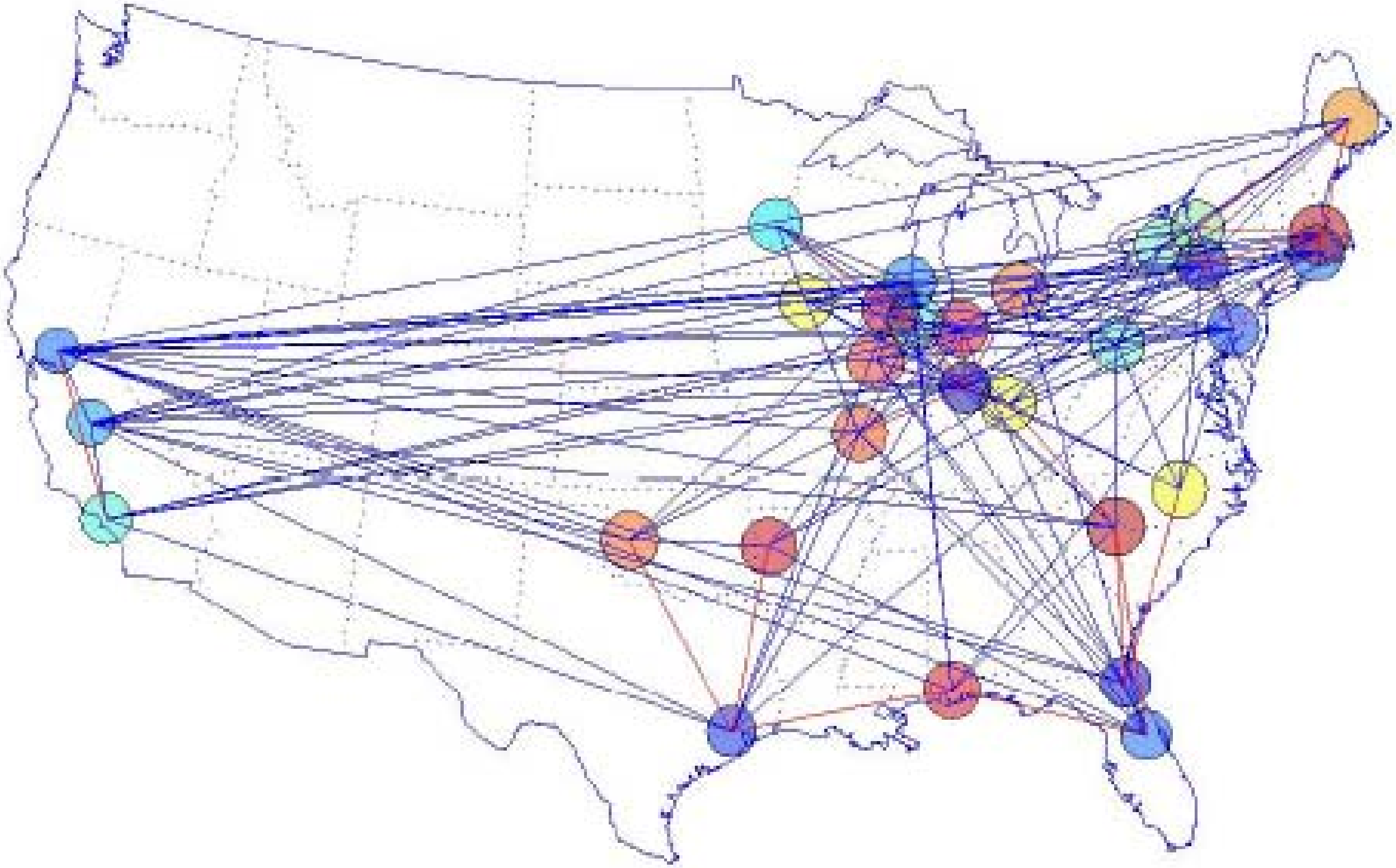
Exploration vs. exploitation



$$w_a^0\overline{V}^0\left(a_{11}\right)+w_a^1\overline{V}^1\begin{pmatrix}a_{11}\\a_{12}\end{pmatrix}+w_a^2\overline{V}^2\begin{pmatrix}a_{11}\\a_{12}\\a_{13}\end{pmatrix}$$

$$\begin{bmatrix}a_1\\a_2\\\vdots\\a_n\end{bmatrix}$$

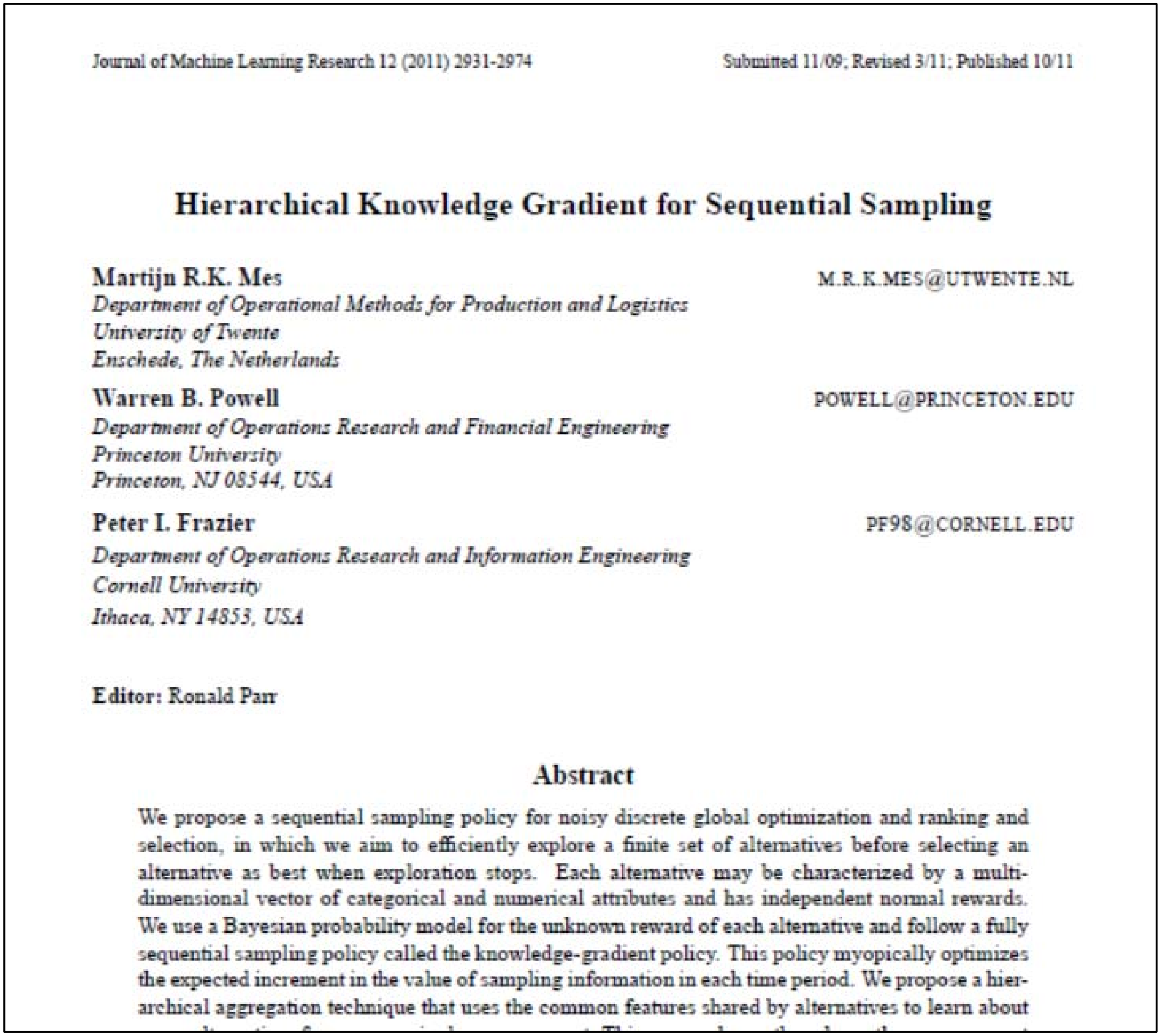
Exploration vs. exploitation



Pure exploitation with generalized learning.

Exploration vs. exploitation

- Notes
  - » This is a learning problem in the presence of a physical state (the location)
  - » Above, we are using a pure exploitation strategy, but with generalized learning (visiting one location teaches us about another location).
  - » An active area of research, with painfully little progress, is how to do active learning for state-dependent problems (in general) and more specifically, problems with a physical state.
  - » Note that we have avoided modeling the uncertainty in the value functions as part of the state variable. This is a historical oversight.



Journal of Machine Learning Research 12 (2011) 2931-2974 Submitted 11/09; Revised 3/11; Published 10/11

Hierarchical Knowledge Gradient for Sequential Sampling

Martijn R.K. Mes M.R.K.MES@UTWENTE.NL
Department of Operational Methods for Production and Logistics
University of Twente
Enschede, The Netherlands

Warren B. Powell POWELL@PRINCETON.EDU
Department of Operations Research and Financial Engineering
Princeton University
Princeton, NJ 08544, USA

Peter I. Frazier PF98@CORNELL.EDU
Department of Operations Research and Information Engineering
Cornell University
Ithaca, NY 14853, USA

Editor: Ronald Parr

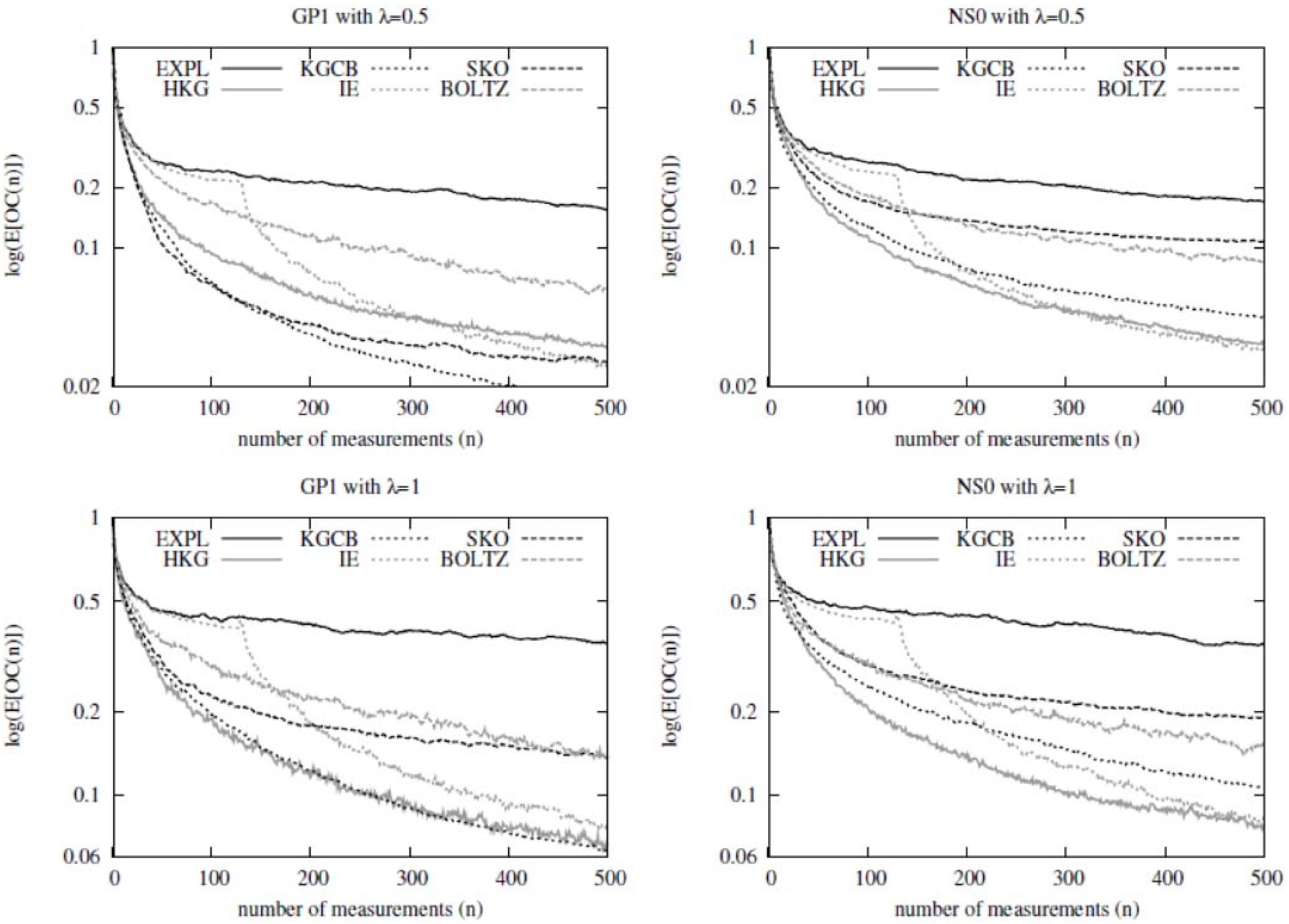
Abstract

We propose a sequential sampling policy for noisy discrete global optimization and ranking and selection, in which we aim to efficiently explore a finite set of alternatives before selecting an alternative as best when exploration stops. Each alternative may be characterized by a multi-dimensional vector of categorical and numerical attributes and has independent normal rewards. We use a Bayesian probability model for the unknown reward of each alternative and follow a fully sequential sampling policy called the knowledge-gradient policy. This policy myopically optimizes the expected increment in the value of sampling information in each time period. We propose a hierarchical aggregation technique that uses the common features shared by alternatives to learn about

Optimal learning with a single physical state and hierarchical learning.

Exploration vs. exploitation

- Comparison of policies for pure learning problems

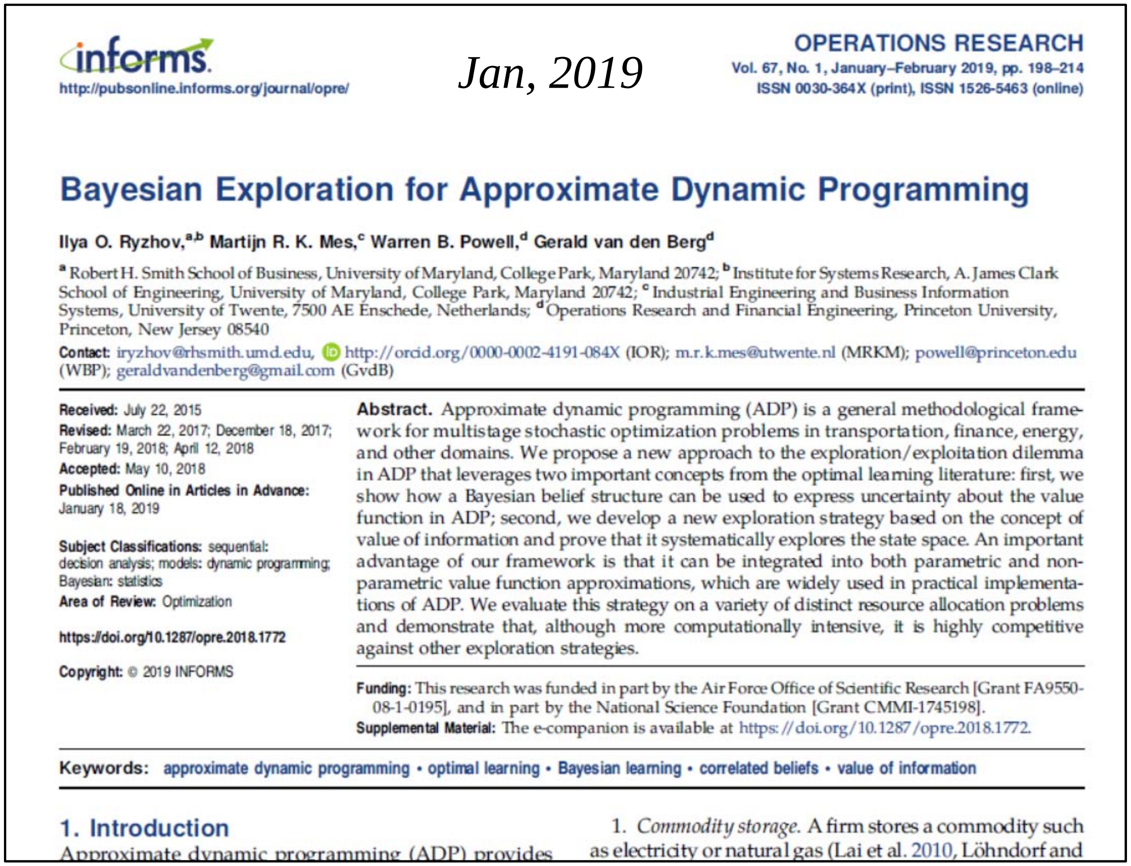


Exploration vs. exploitation

- Notes:
  - » We have extensive research on pure learning problems.
  - » Very little has been done with problems that combine a belief state (which is the basis of any active learning problem) and a physical state.
  - » Central to the value of making a decision that is balancing the value of information is how this information is used in future decisions.
    - E.g. if we learn more about the cost c_{ij} by going from i to j, then this is only useful if we return to i so that we can use this information.
    - For this reason, the presence of generalized learning architectures is key.

Exploration vs. exploitation

- What about learning?
  - » “Active learning” with general dynamic programs is a very young field.
  - » Typically “exploration vs. exploitation” issues are solved with CFAs using approximate value functions as part of the estimate of the value of a decision.



Jan, 2019

informs.
http://pubsonline.informs.org/journal/opre/

OPERATIONS RESEARCH
Vol. 67, No. 1, January–February 2019, pp. 198–214
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Bayesian Exploration for Approximate Dynamic Programming

Ilya O. Ryzhov,[a,b] Martijn R. K. Mes,[c] Warren B. Powell,[d] Gerald van den Berg[d]

[a] Robert H. Smith School of Business, University of Maryland, College Park, Maryland 20742; [b] Institute for Systems Research, A. James Clark School of Engineering, University of Maryland, College Park, Maryland 20742; [c] Industrial Engineering and Business Information Systems, University of Twente, 7500 AE Enschede, Netherlands; [d] Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08540

Contact: iryzhov@rhsmith.umd.edu, http://orcid.org/0000-0002-4191-084X (IOR); m.r.k.mes@utwente.nl (MRKM); powell@princeton.edu (WBP); geraldvandenberg@gmail.com (GvdB)

Received: July 22, 2015
Revised: March 22, 2017; December 18, 2017; February 19, 2018; April 12, 2018
Accepted: May 10, 2018
Published Online in Articles in Advance: January 18, 2019

Subject Classifications: sequential: decision analysis; models: dynamic programming; Bayesian: statistics
Area of Review: Optimization

https://doi.org/10.1287/opre.2018.1772

Copyright: © 2019 INFORMS

Abstract. Approximate dynamic programming (ADP) is a general methodological framework for multistage stochastic optimization problems in transportation, finance, energy, and other domains. We propose a new approach to the exploration/exploitation dilemma in ADP that leverages two important concepts from the optimal learning literature: first, we show how a Bayesian belief structure can be used to express uncertainty about the value function in ADP; second, we develop a new exploration strategy based on the concept of value of information and prove that it systematically explores the state space. An important advantage of our framework is that it can be integrated into both parametric and nonparametric value function approximations, which are widely used in practical implementations of ADP. We evaluate this strategy on a variety of distinct resource allocation problems and demonstrate that, although more computationally intensive, it is highly competitive against other exploration strategies.

Funding: This research was funded in part by the Air Force Office of Scientific Research [Grant FA9550-08-1-0195], and in part by the National Science Foundation [Grant CMMI-1745198].
Supplemental Material: The e-companion is available at https://doi.org/10.1287/opre.2018.1772.

Keywords: approximate dynamic programming • optimal learning • Bayesian learning • correlated beliefs • value of information

1. Introduction

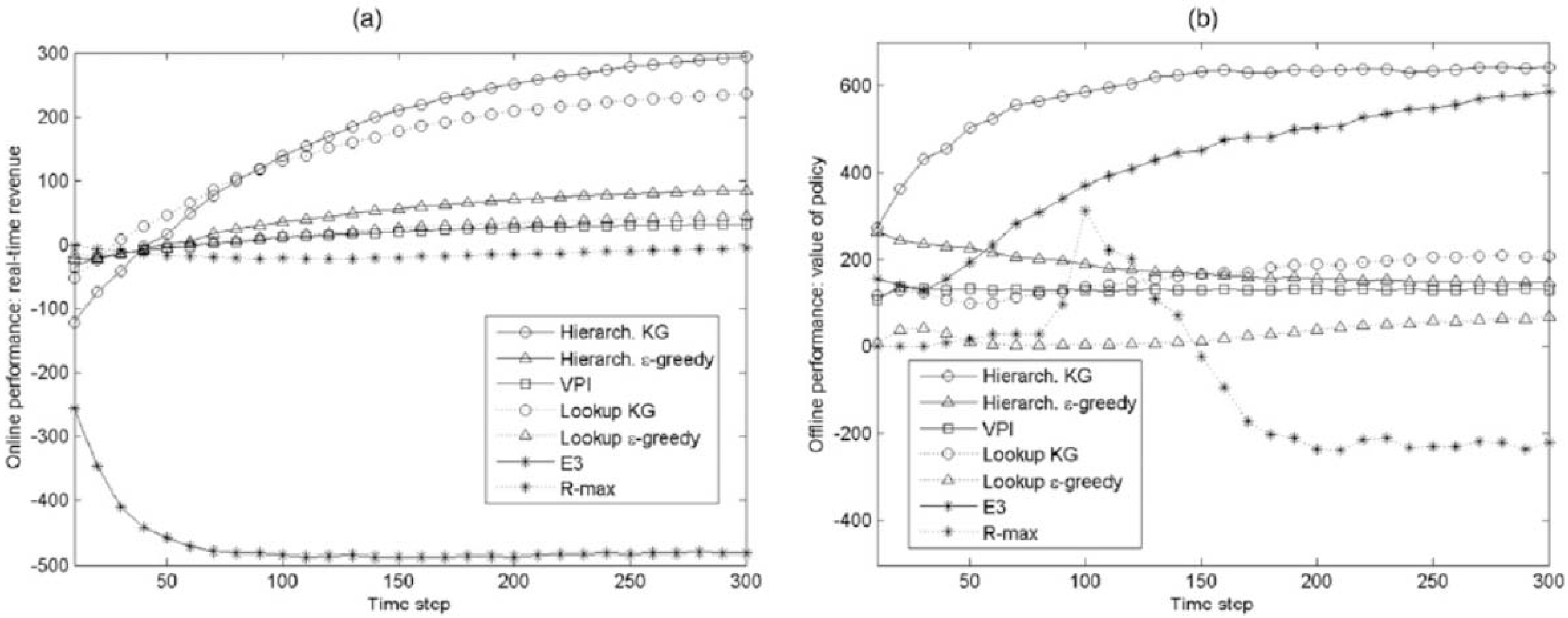
Approximate dynamic programming (ADP) provides

1. *Commodity storage.* A firm stores a commodity such as electricity or natural gas (Lai et al. 2010, Löhndorf and

Exploration vs. exploitation

- Some numerical experiments from Ryzhov et al. paper.

Figure 2. Experimental Comparisons for Commodity Storage with Stochastic Price



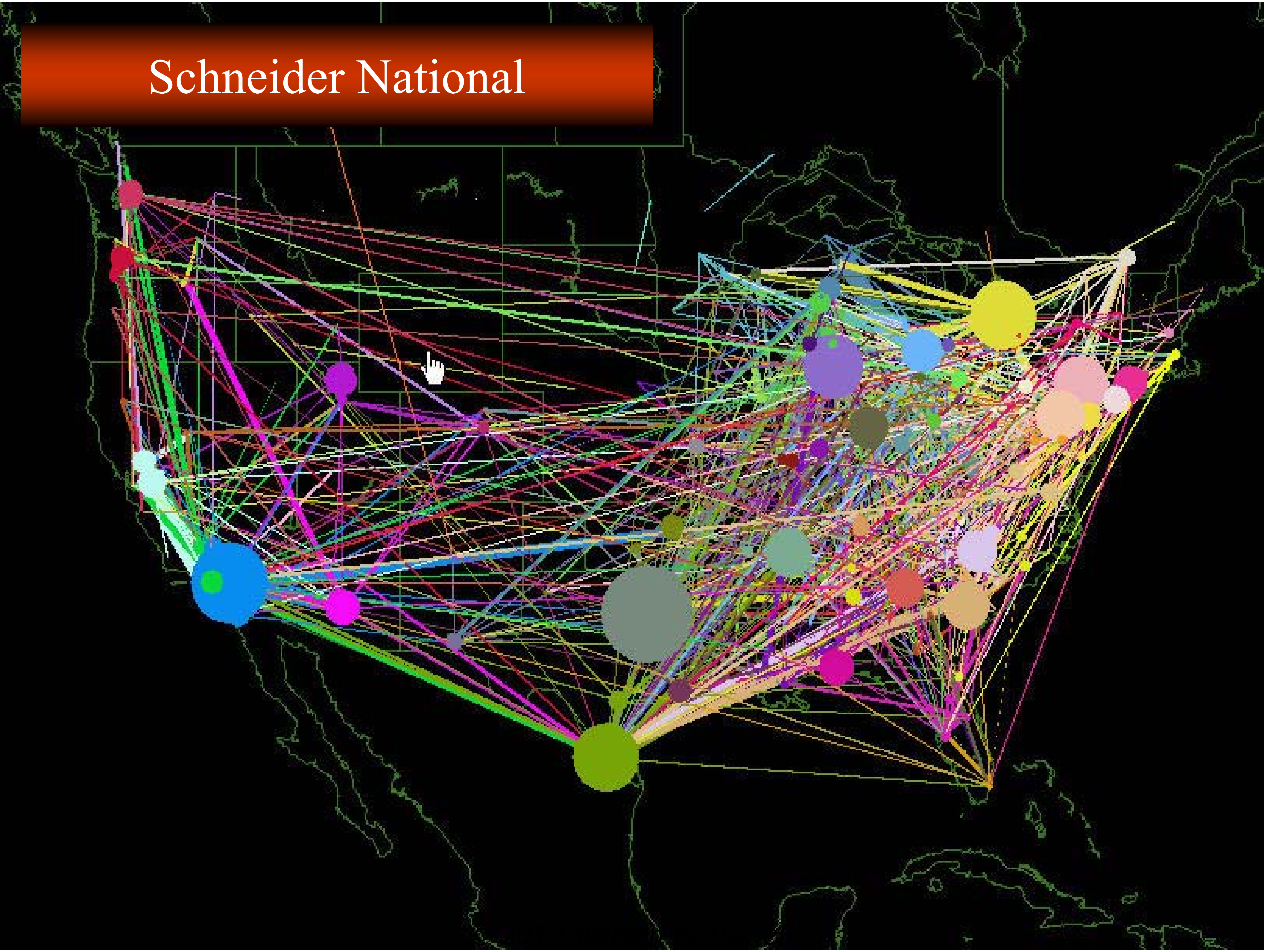
Notes. (a) Online performance. (b) Offline performance.

Exploration vs. exploitation

- Notes:
  - » The exploration vs. exploitation problem is well known in approximate dynamic programming, but lacks the elegant solution of pure learning problems.
  - » Most algorithms use fairly simple heuristics to balance exploration and exploitation.
  - » We have been pursuing research in optimal learning:
    - First paper: optimally sampling a function represented by a hierarchical belief model.
    - Second paper (in preparation): optimal learning with a physical state (the truck)
  - » Our research in optimal learning with a physical state is modest. The challenge is our ability to learn from one physical state, and generalize to others.
  - » Open question right now: how much does active learning contribute when there is a physical state? This is likely to be very problem-dependent.

From one truck to many

Schneider National

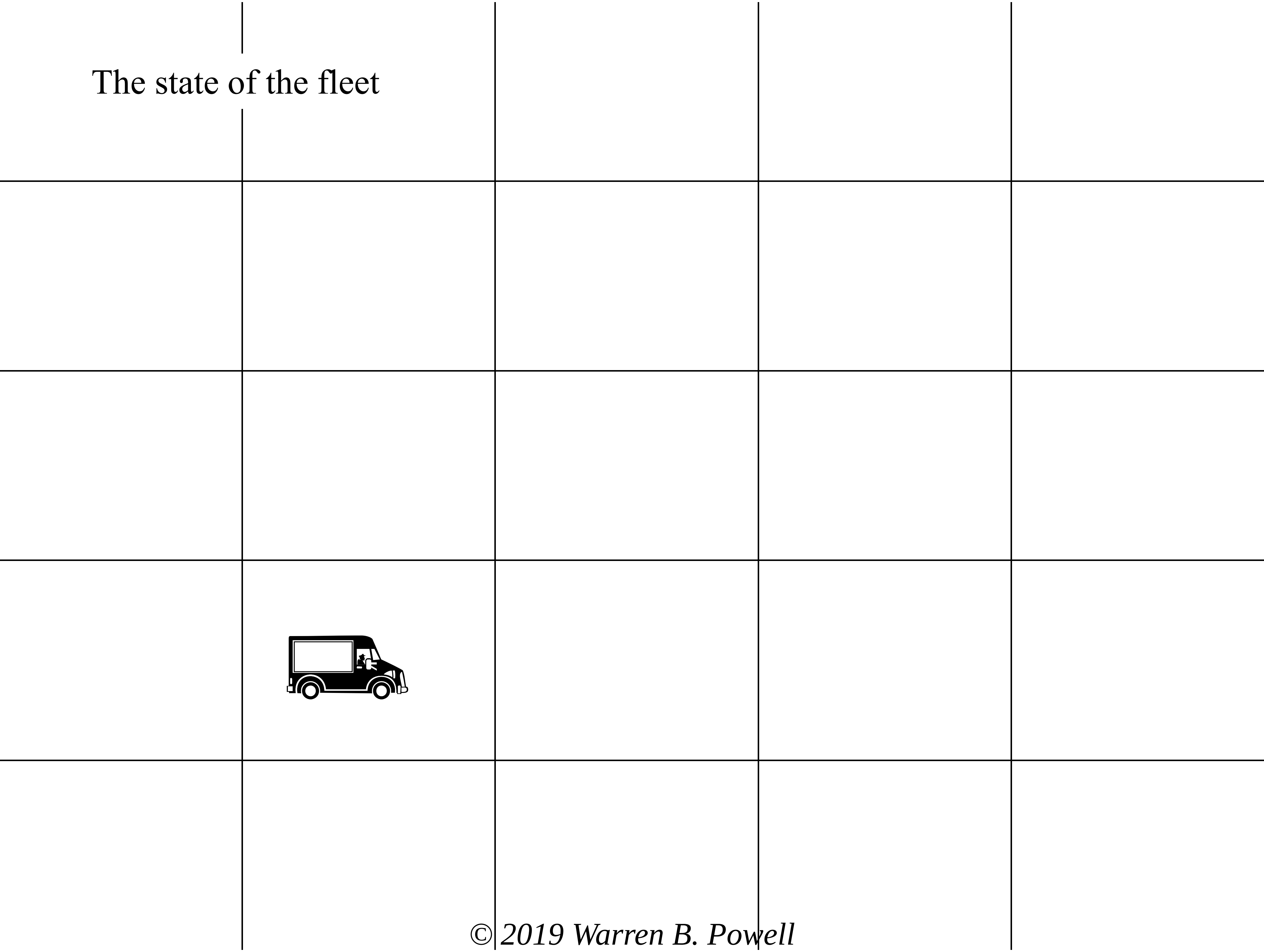


SCHNEIDER
Week 45
1 Way Rank
Dedic Rank
Trkfl Rank
Brck Rank
Comb Rank
Miles 18 4 1 3 1
Working units 18 1
By industry
Auto Asmby
Auto Parts
Retail
Paper
Miles 8 27 1 9 33
Nov YOY
1 Way
Dedic
Trkfl
Growth
SCHNEIDER

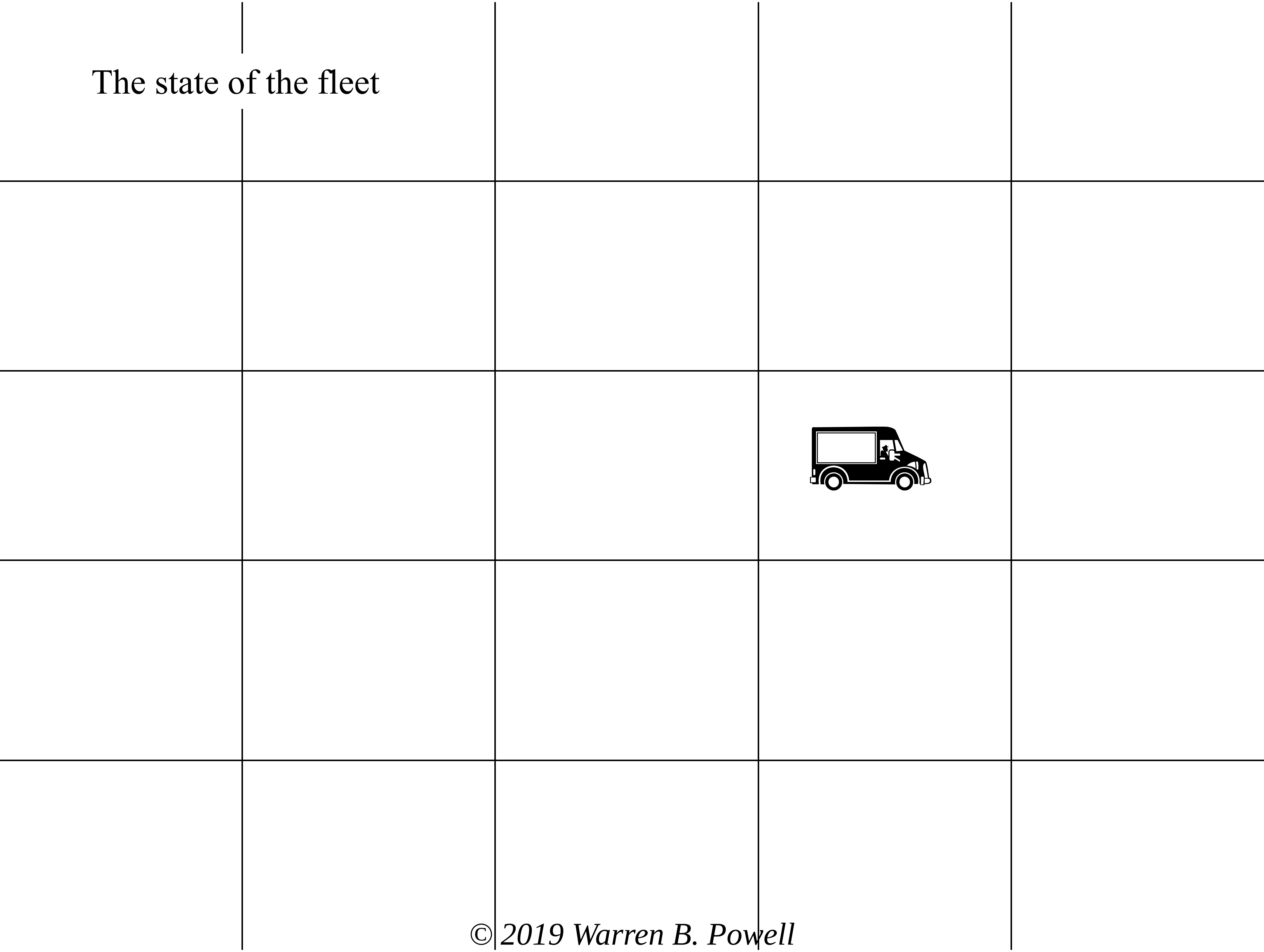
Optimizing fleets

- From one truck to many trucks
  - » If there is one truck and N "states" (locations), then our dynamic program (post-decision state) has N states.
  - » But what if there is more than one truck? Then we have to capture the state of the fleet.

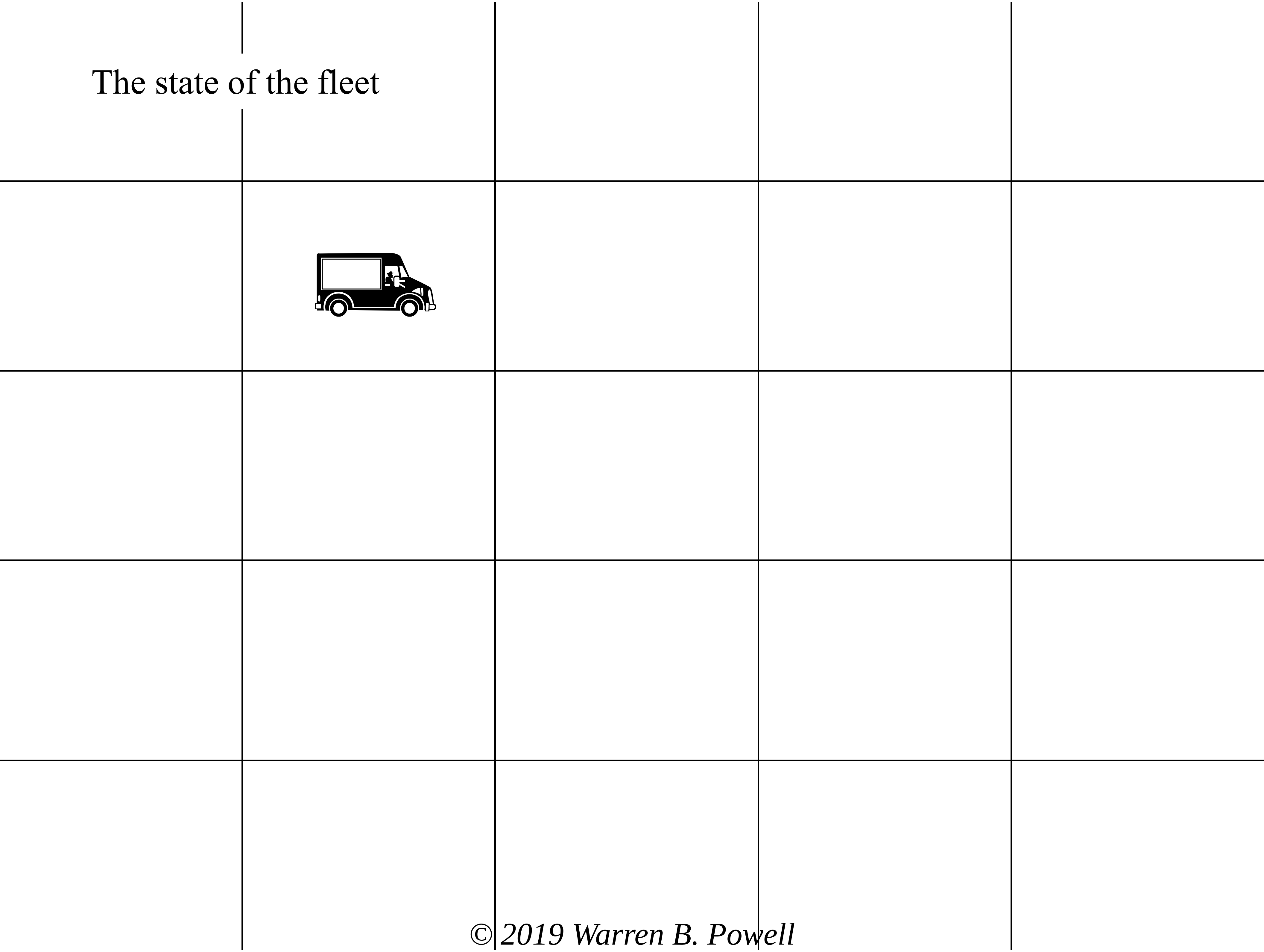
The state of the fleet



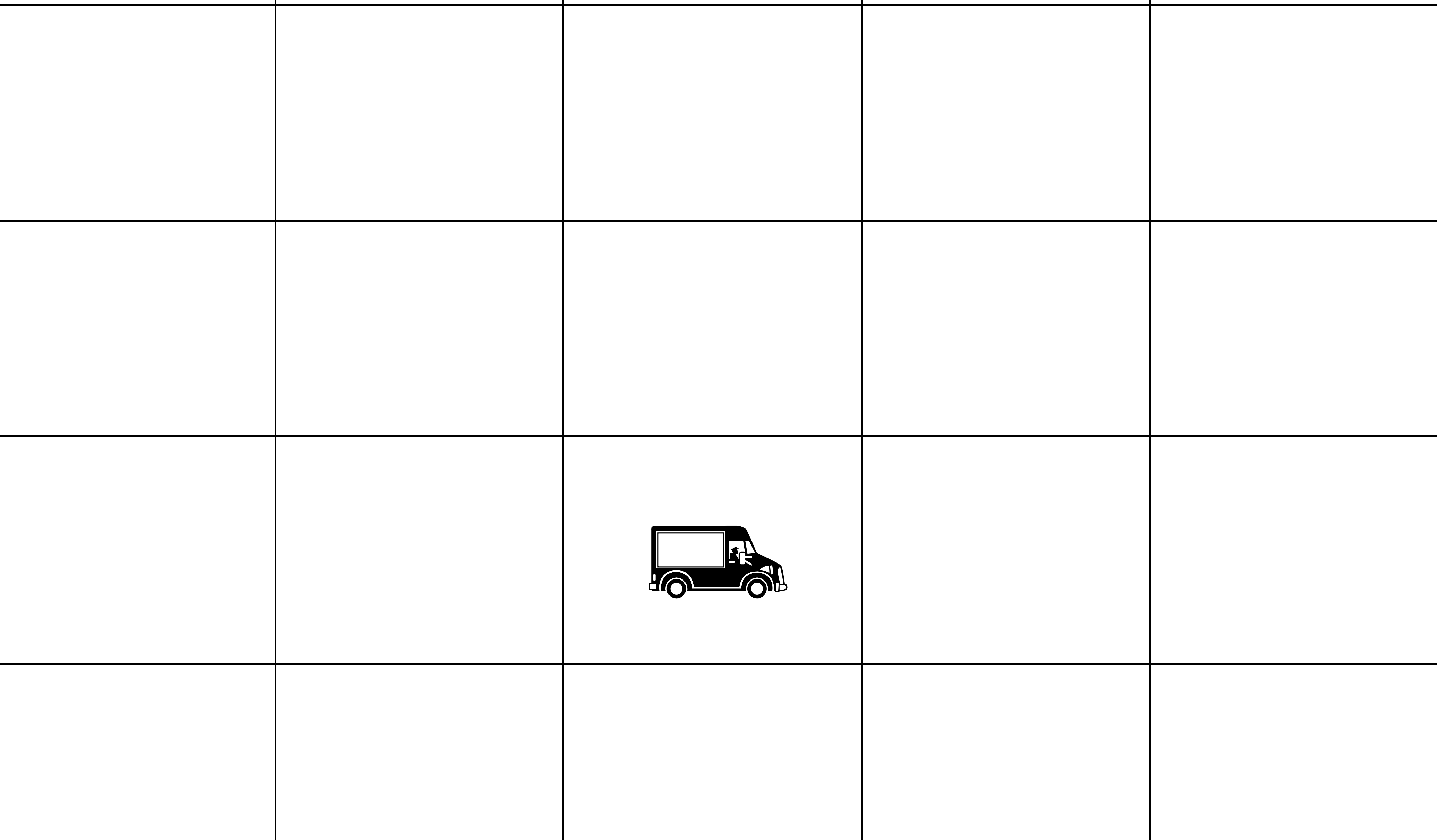
The state of the fleet



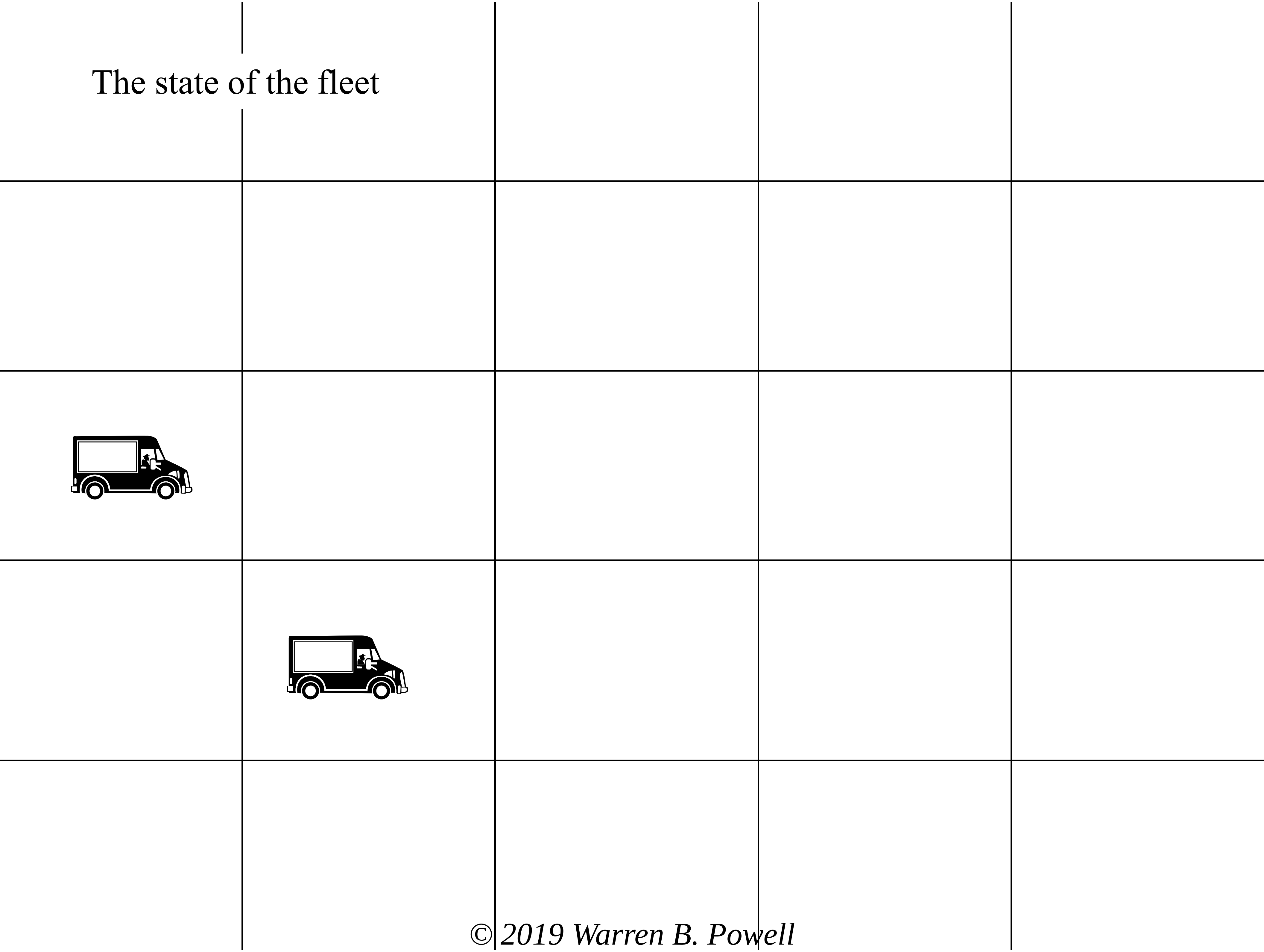
The state of the fleet



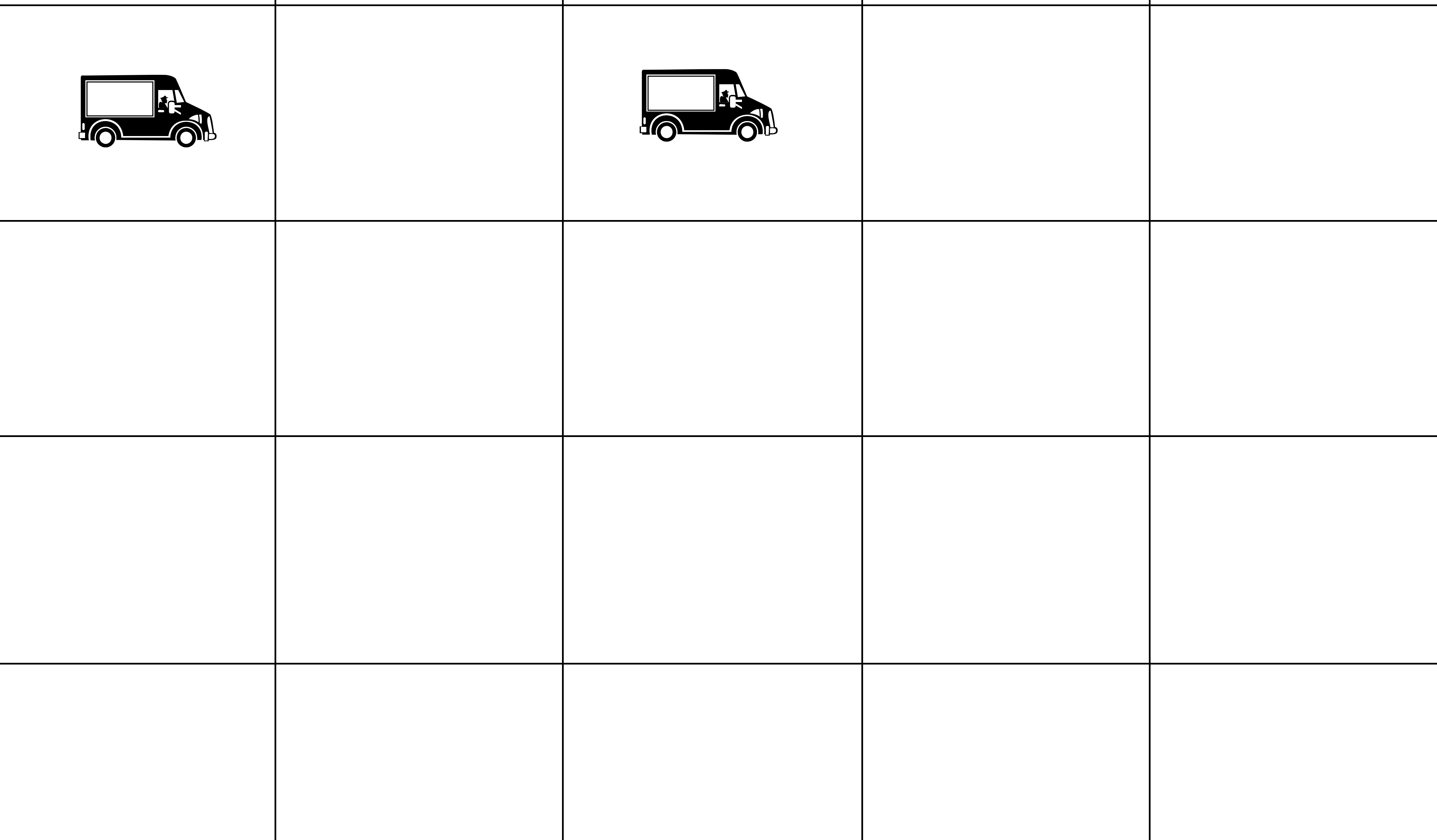
The state of the fleet



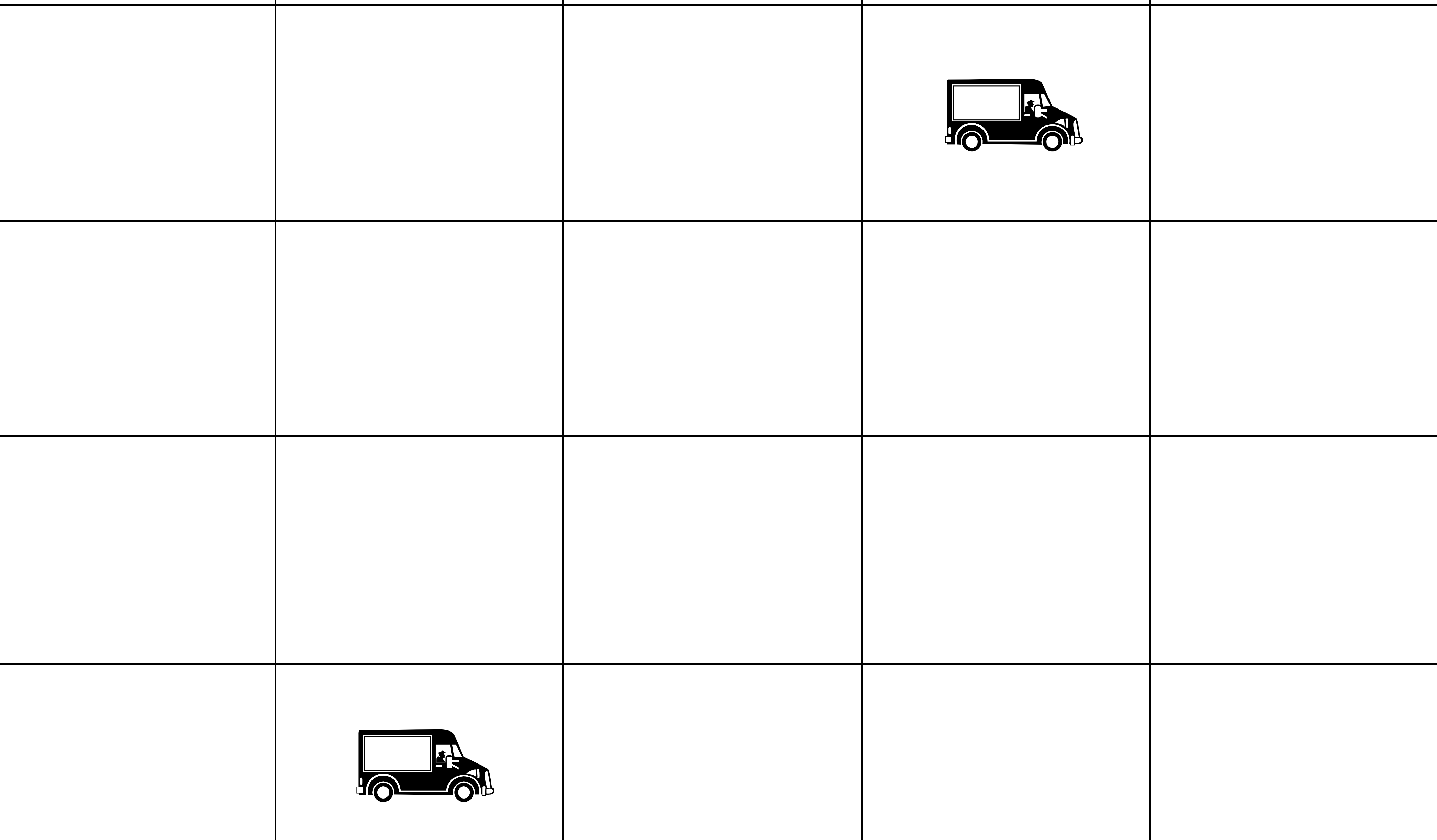
The state of the fleet



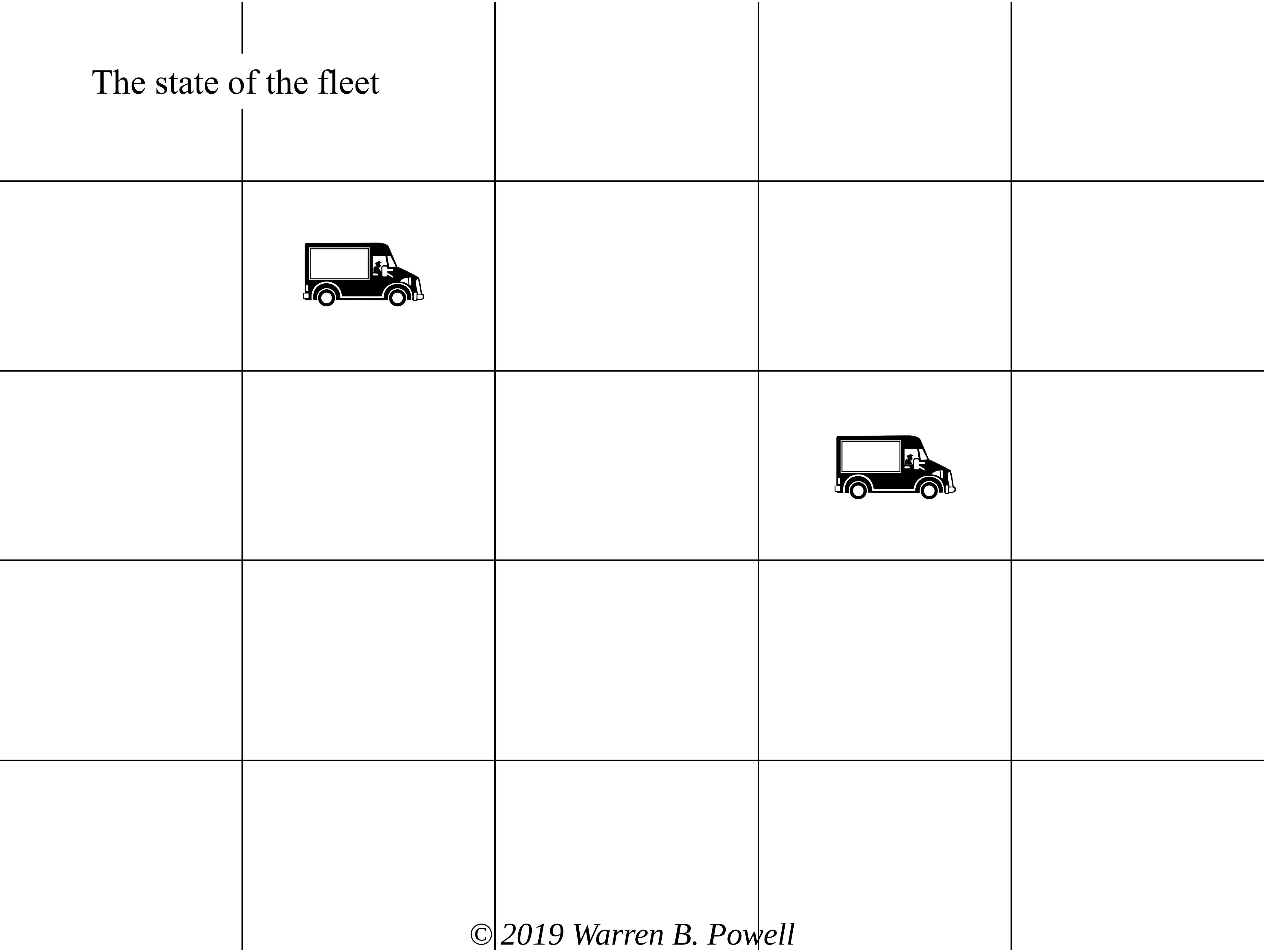
The state of the fleet



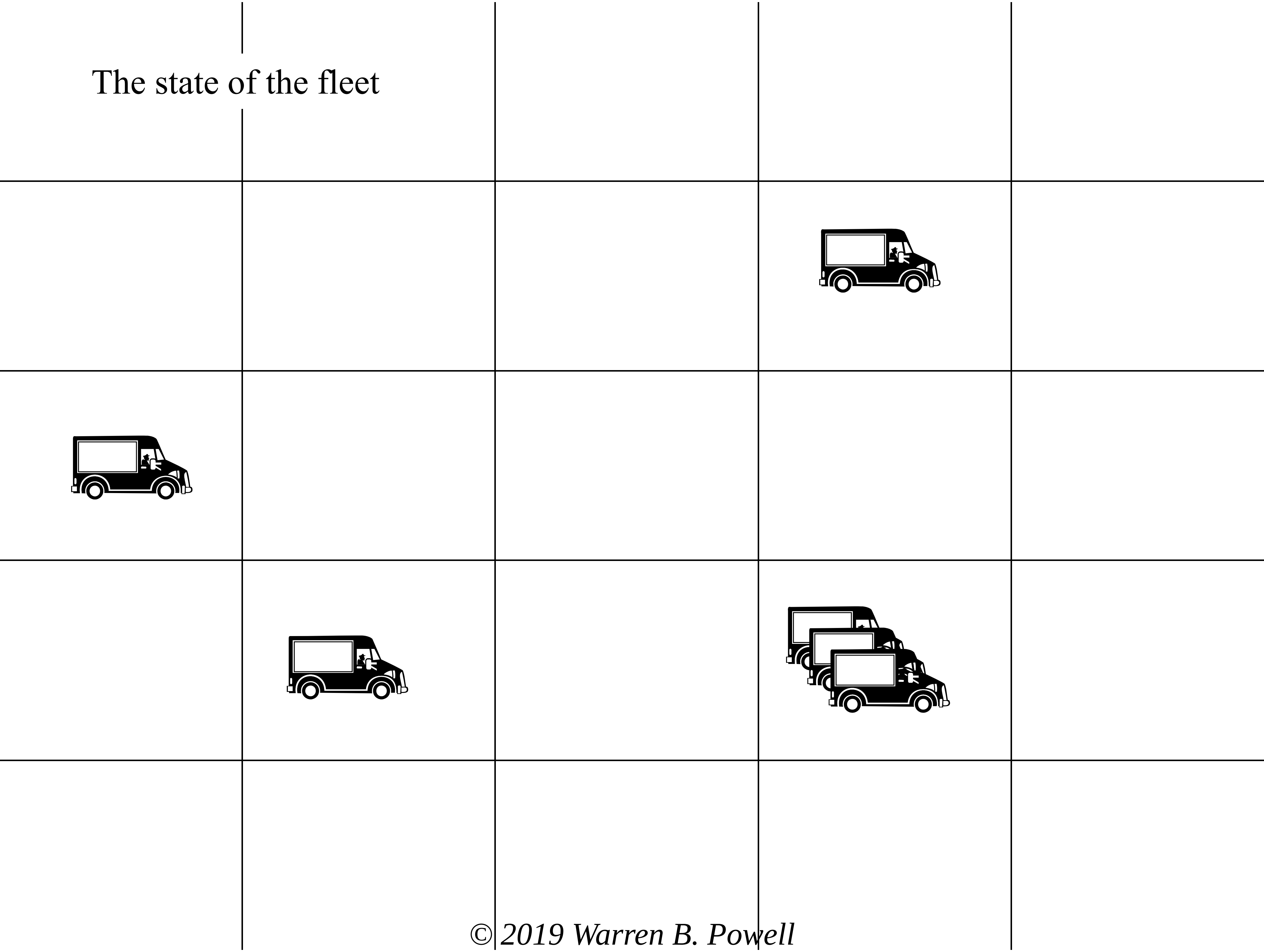
The state of the fleet



The state of the fleet



The state of the fleet



Optimizing fleets

- What if we have N > 1 trucks?

$$|States| = \binom{No.\ trucks\ + |Locations| - 1}{|Locations| - 1}$$

Number of resources	Attribute space	State space
1	1	1
1	100	100
1	1000	1,000
5	10	2,002
5	100	91,962,520
5	1000	8,416,958,750,200
50	10	12,565,671,261
50	100	13,419,107,273,154,600,000,000,000,000,000,000,000,000
50	1000	109,740,941,767,311,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000

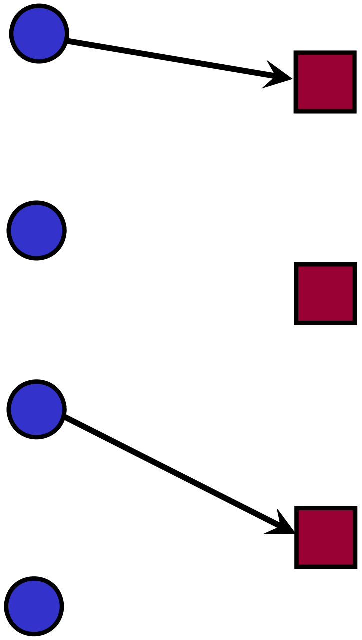
Real problems: 500 to 5,000 trucks, attribute space 50,000 up to 10^{20} attributes.

Optimizing fleets

- The pre-decision state: drivers and loads

Drivers Loads

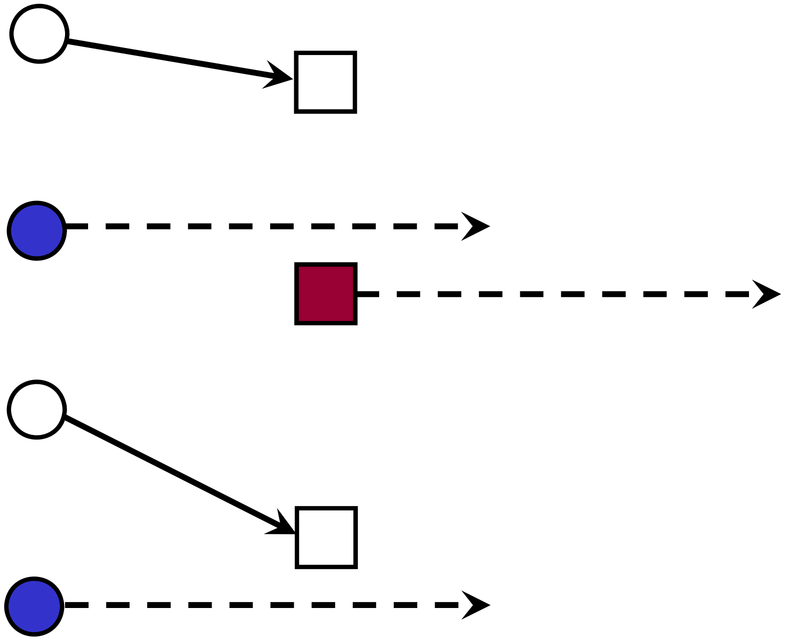
$$S_t = (R_t, D_t)$$



Optimizing fleets

- The post-decision state - drivers and loads after a decision is made:

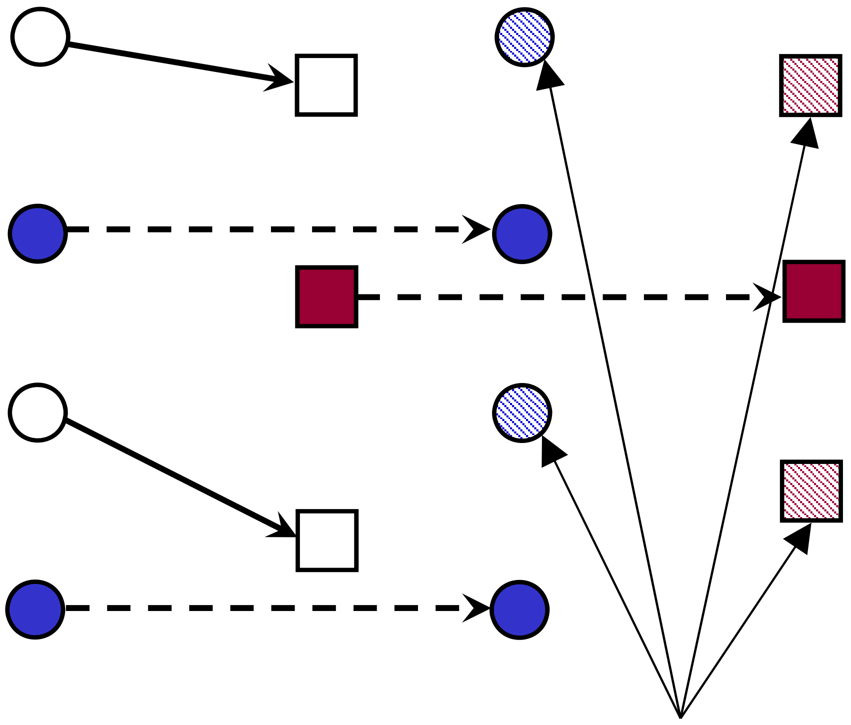
$$S_t^x = S^{M,x}(S_t, x_t)$$



Optimizing fleets

- The transition: Adding new information

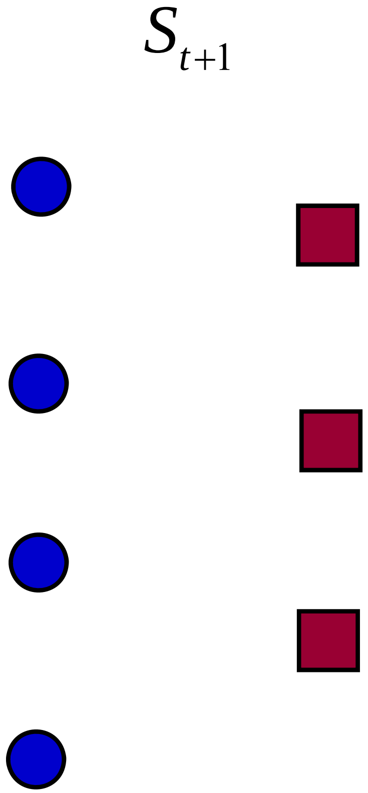
S_t^x $\qquad$ $S_{t+1} = S^{M,W}(S_t^x, W_{t+1})$



$W_{t+1} = (\hat{R}_{t+1}, \hat{D}_{t+1})$

Optimizing fleets

- The next pre-decision state

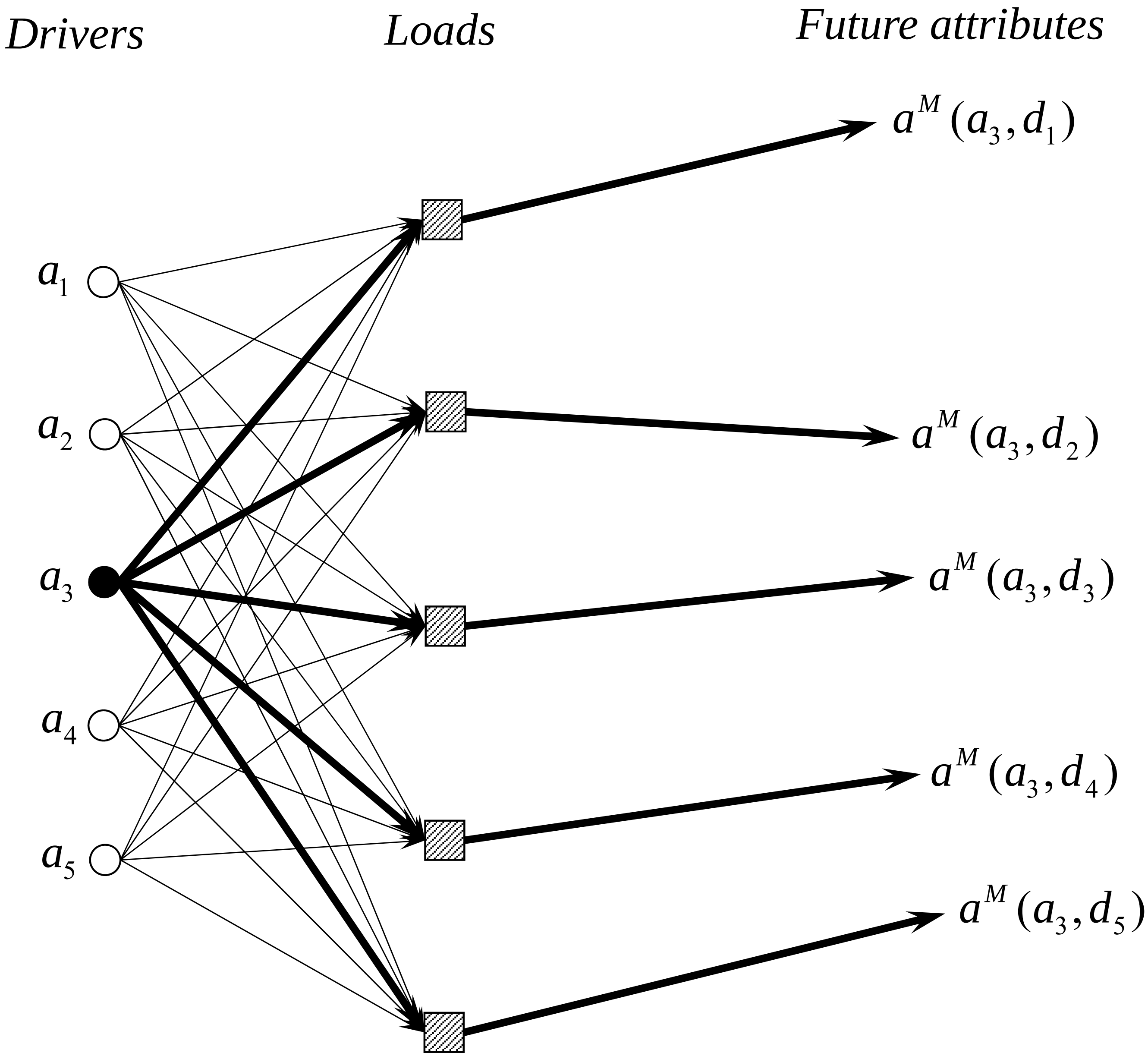


Optimizing fleets

- Assignment network
  - » Capture the value of downstream driver.

$a^M(a_3, d_1) =$ Attribute vector of driverin the future given a decision d.

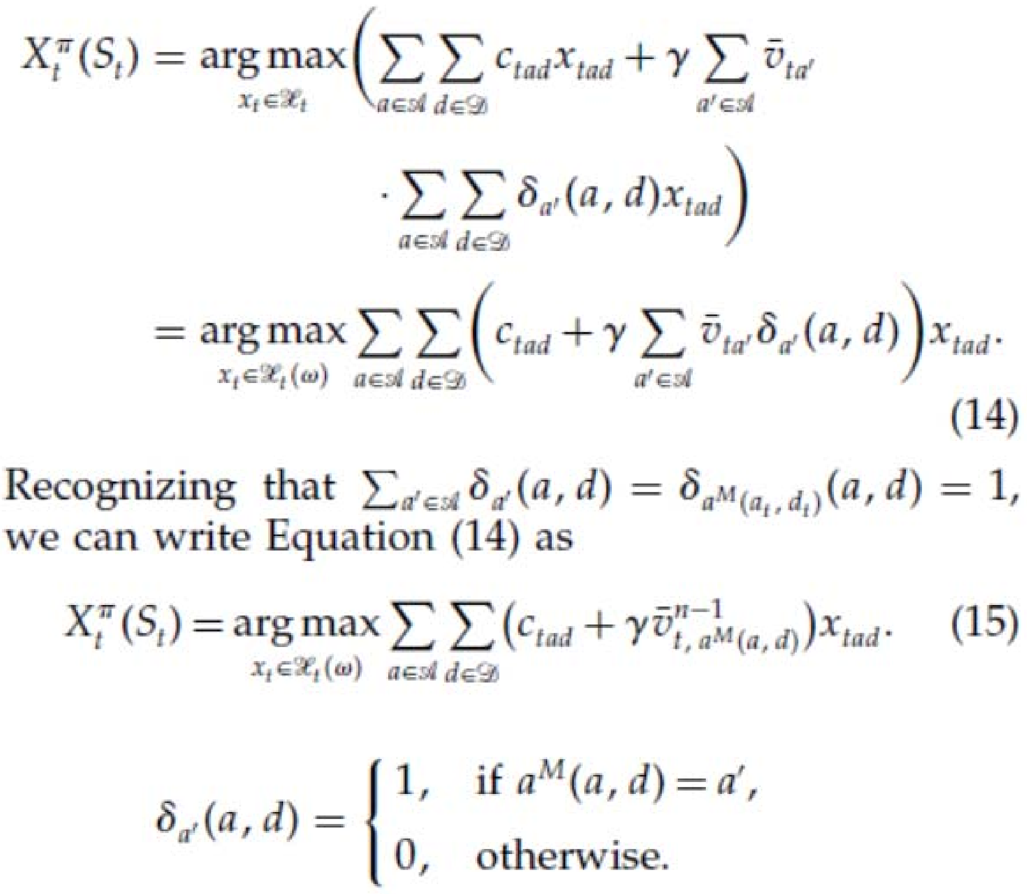
  - » Add this value to the assignment arc.



© 2019 Warren B. Powell

Optimizing fleets

- The assignment problem
  - » We now have a basic assignment problem, but where we have to capture the downstream value of a truck:



$$X_t^\pi(S_t) = \arg\max_{x_t\in\mathcal{X}_t}\left(\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}} c_{tad}x_{tad} + \gamma\sum_{a'\in\mathcal{A}}\bar{v}_{ta'} \cdot \sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}}\delta_{a'}(a,d)x_{tad}\right)$$

$$= \arg\max_{x_t\in\mathcal{X}_t(\omega)}\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}}\left(c_{tad}+\gamma\sum_{a'\in\mathcal{A}}\bar{v}_{ta'}\delta_{a'}(a,d)\right)x_{tad}. \qquad (14)$$

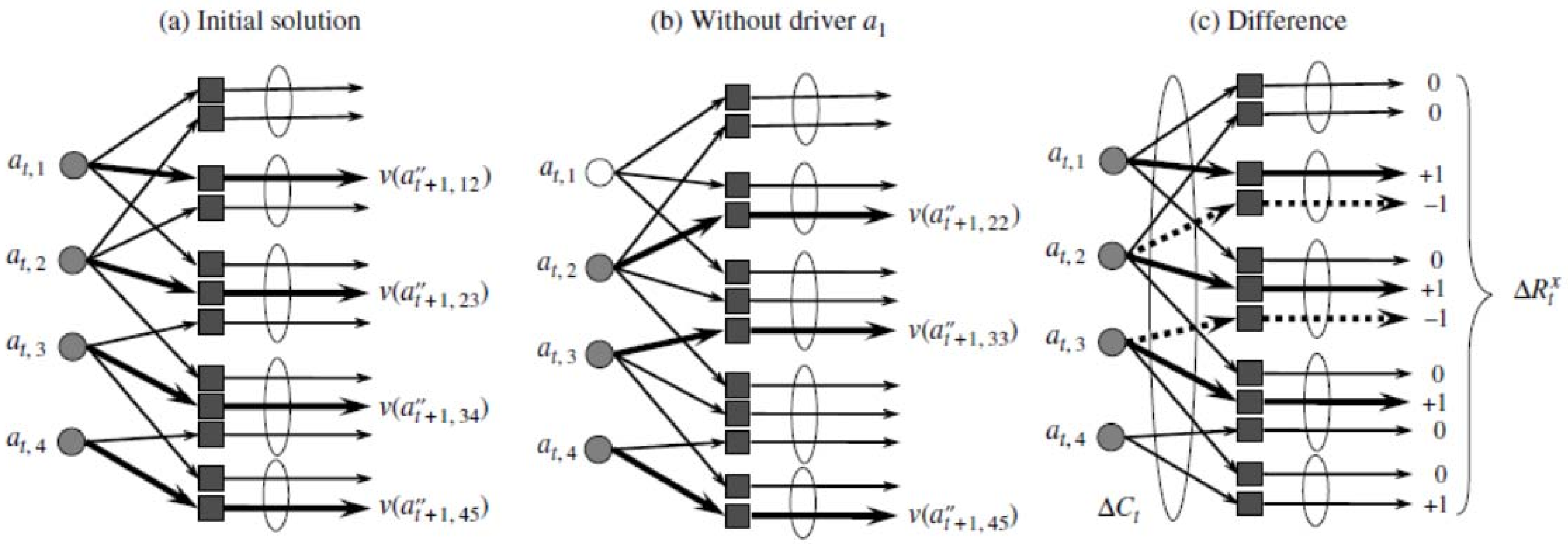
Recognizing that $\sum_{a'\in\mathcal{A}}\delta_{a'}(a,d) = \delta_{a^M(a_t,d_t)}(a,d) = 1$, we can write Equation (14) as

$$X_t^\pi(S_t) = \arg\max_{x_t\in\mathcal{X}_t(\omega)}\sum_{a\in\mathcal{A}}\sum_{d\in\mathcal{D}}\left(c_{tad}+\gamma\bar{v}^{n-1}_{t,a^M(a,d)}\right)x_{tad}. \qquad (15)$$

$$\delta_{a'}(a,d) = \begin{cases} 1, & \text{if } a^M(a,d) = a', \\ 0, & \text{otherwise.} \end{cases}$$

Optimizing fleets

- 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:



Optimizing fleets

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:

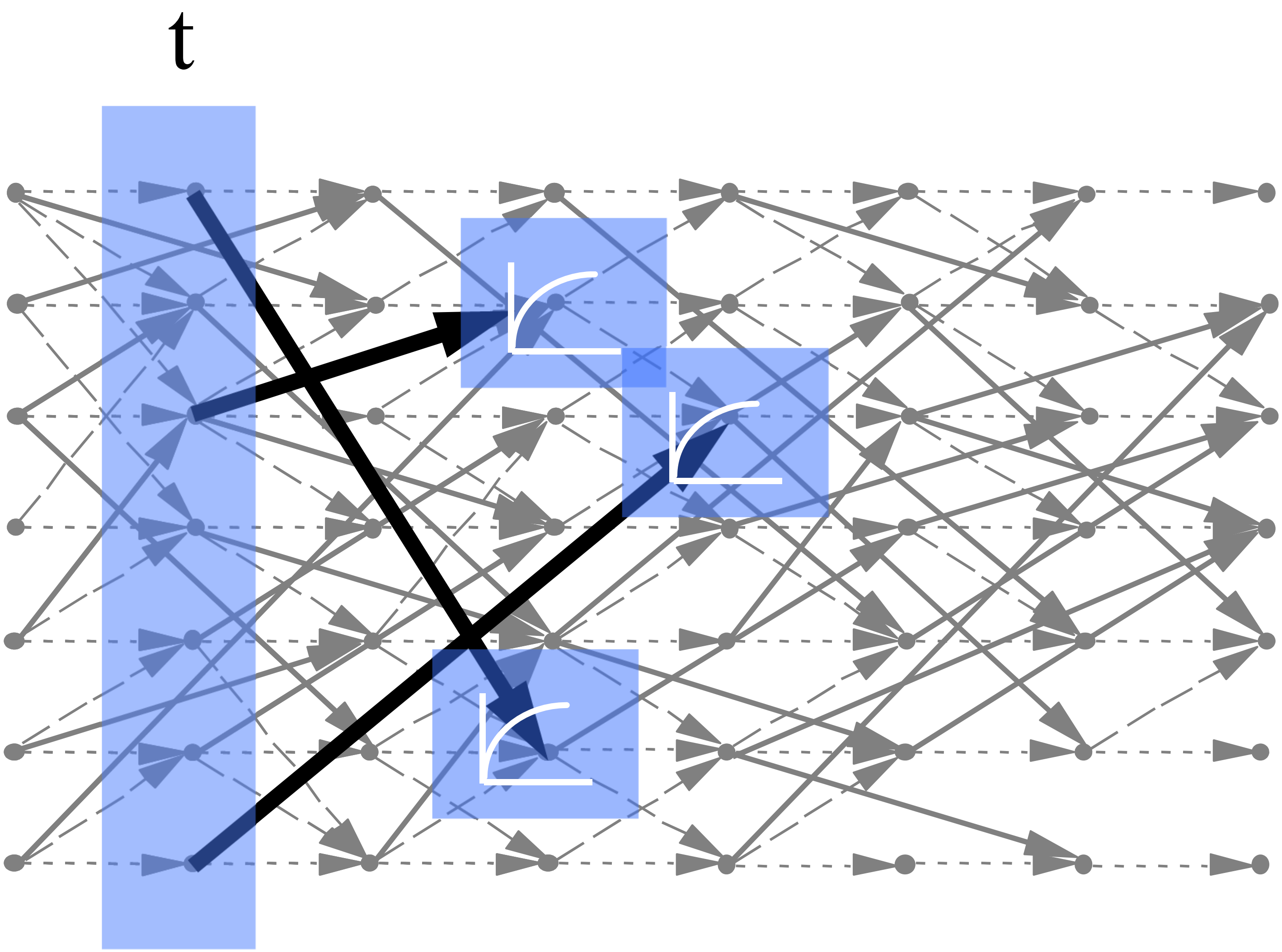
$$S_{t+1}^n = S^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Simulation

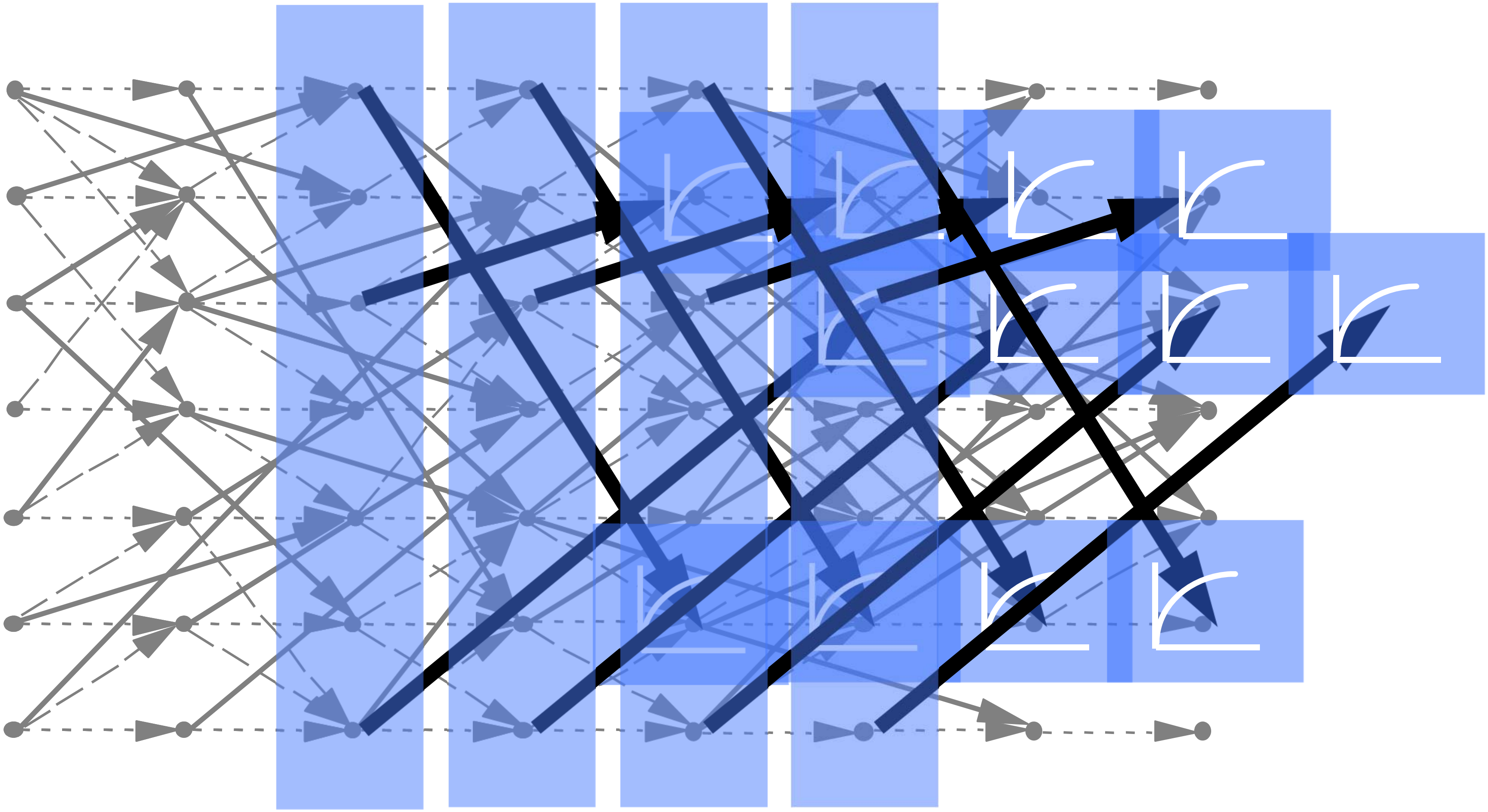
Step 5: Return to step 1.

"on policy learning"

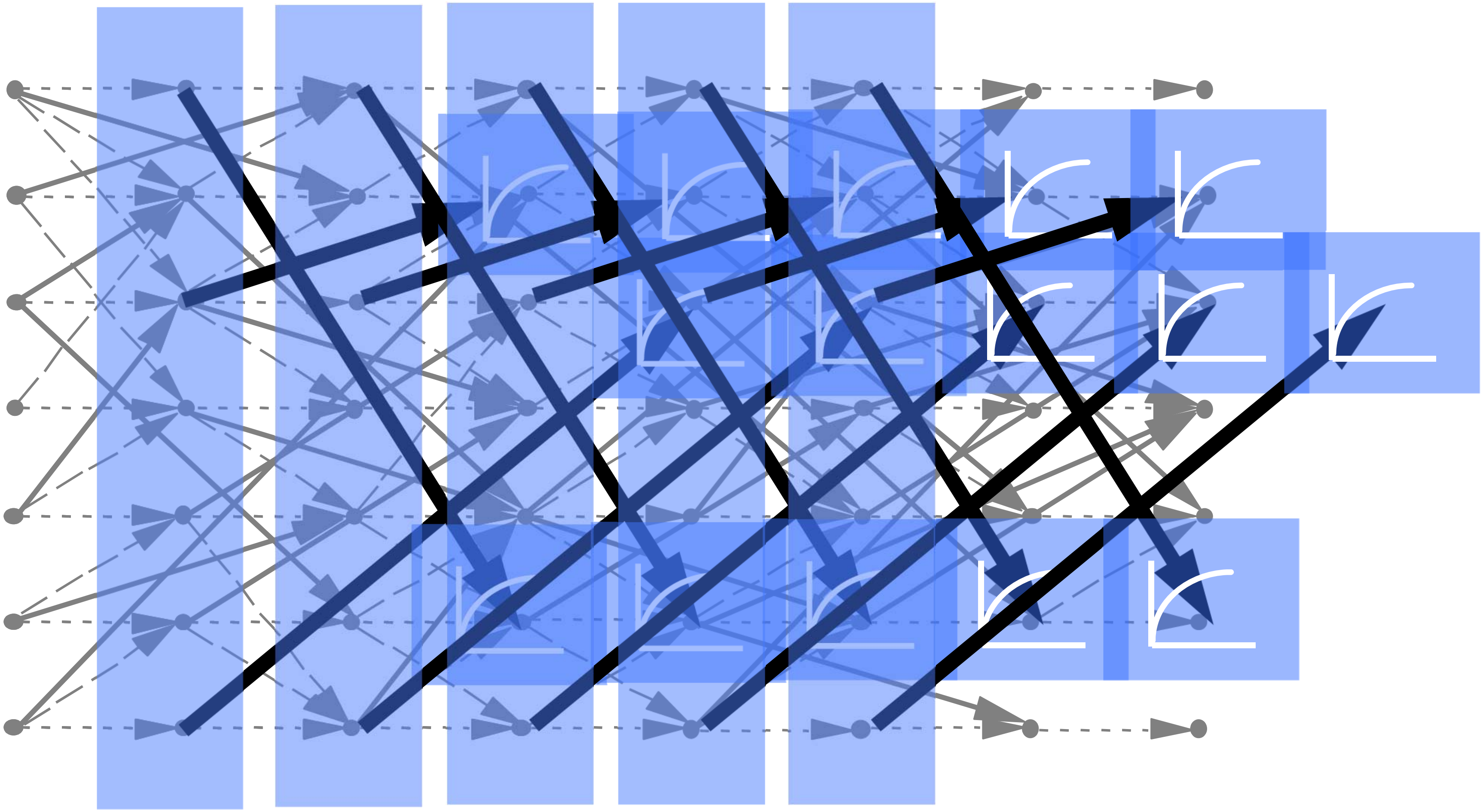
Optimizing fleets



Optimizing fleets

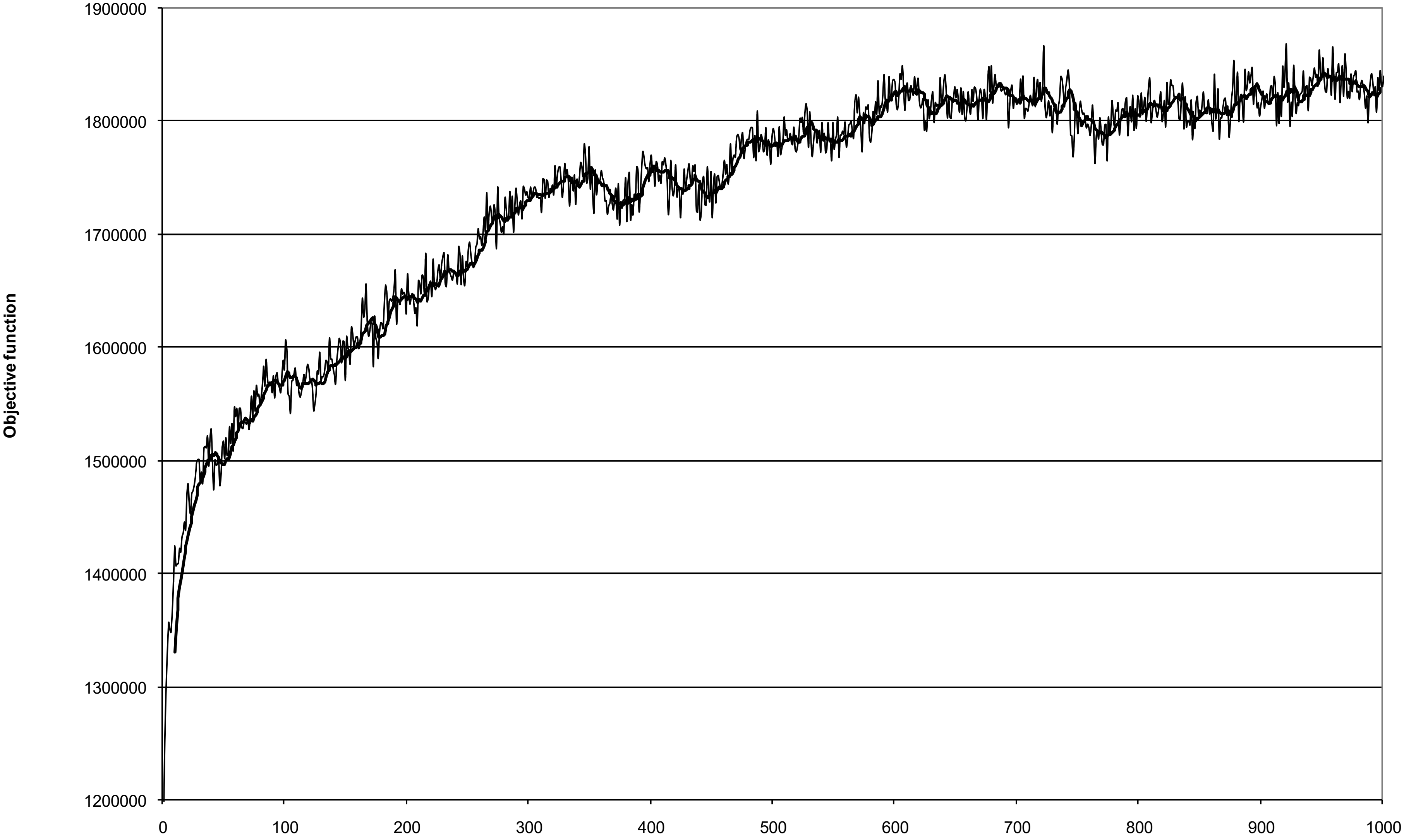


Optimizing fleets



Approximate dynamic programming

- … a typical performance graph.

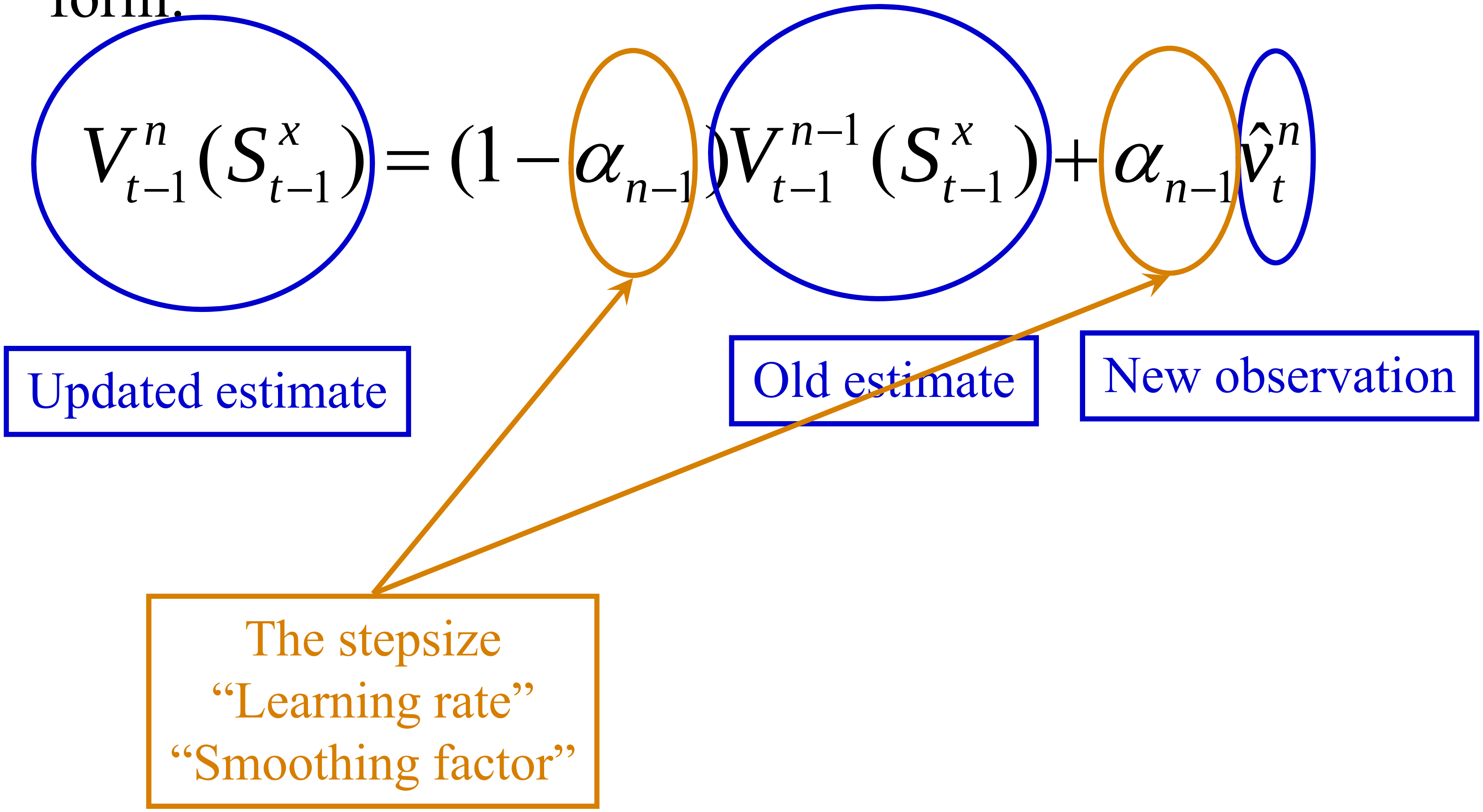


Approximate dynamic programming

Stepsizes for forward APD

Stepsizes

- Stepsizes:
  - » Approximate value iteration requires updates of the form:



$$V^{n}_{t-1}(S^{x}_{t-1}) = (1-\alpha_{n-1})V^{n-1}_{t-1}(S^{x}_{t-1}) + \alpha_{n-1}\hat{v}^{n}_{t}$$

Updated estimate

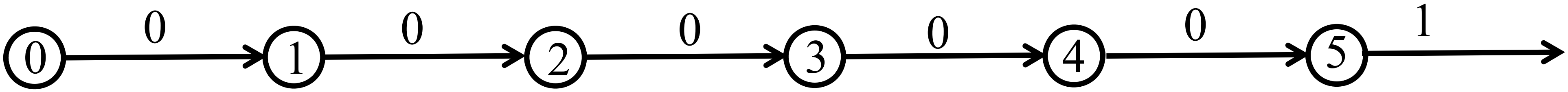
Old estimate

New observation

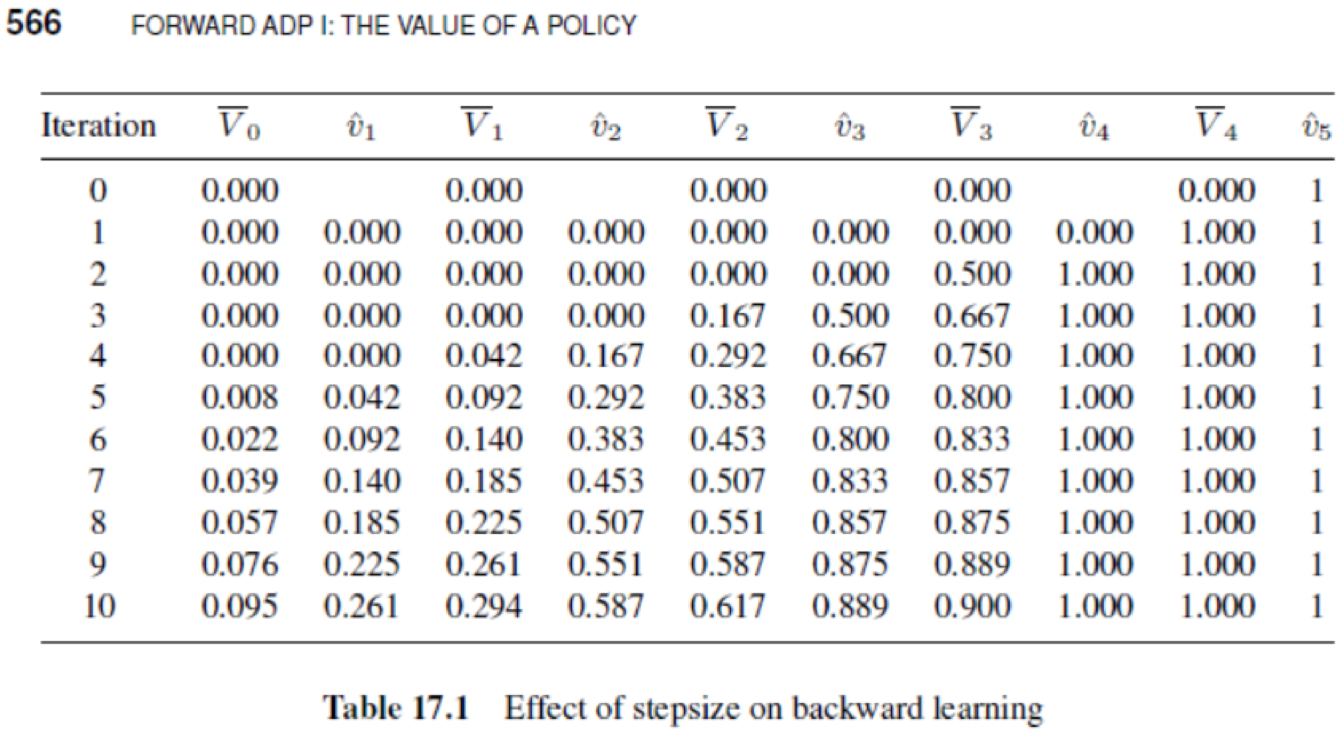
The stepsize
“Learning rate”
“Smoothing factor”

Stepsizes

- Single state, single action Markov chain
  - » Updating – receives reward=1 at last node. All other rewards = 0.



  - » Same as adding up random rewards with mean of 1. Noise may be zero, or quite high.



566 FORWARD ADP I: THE VALUE OF A POLICY

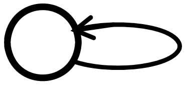
Iteration	$\bar{V}_0$	$\hat{v}_1$	$\bar{V}_1$	$\hat{v}_2$	$\bar{V}_2$	$\hat{v}_3$	$\bar{V}_3$	$\hat{v}_4$	$\bar{V}_4$	$\hat{v}_5$
0	0.000		0.000		0.000		0.000		0.000	1
1	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	1
2	0.000	0.000	0.000	0.000	0.000	0.000	0.500	1.000	1.000	1
3	0.000	0.000	0.000	0.000	0.167	0.500	0.667	1.000	1.000	1
4	0.000	0.000	0.042	0.167	0.292	0.667	0.750	1.000	1.000	1
5	0.008	0.042	0.092	0.292	0.383	0.750	0.800	1.000	1.000	1
6	0.022	0.092	0.140	0.383	0.453	0.800	0.833	1.000	1.000	1
7	0.039	0.140	0.185	0.453	0.507	0.833	0.857	1.000	1.000	1
8	0.057	0.185	0.225	0.507	0.551	0.857	0.875	1.000	1.000	1
9	0.076	0.225	0.261	0.551	0.587	0.875	0.889	1.000	1.000	1
10	0.095	0.261	0.294	0.587	0.617	0.889	0.900	1.000	1.000	1

Table 17.1 Effect of stepsize on backward learning

Stepsizes

- Bound on performance using 1/n:

Single state, single action

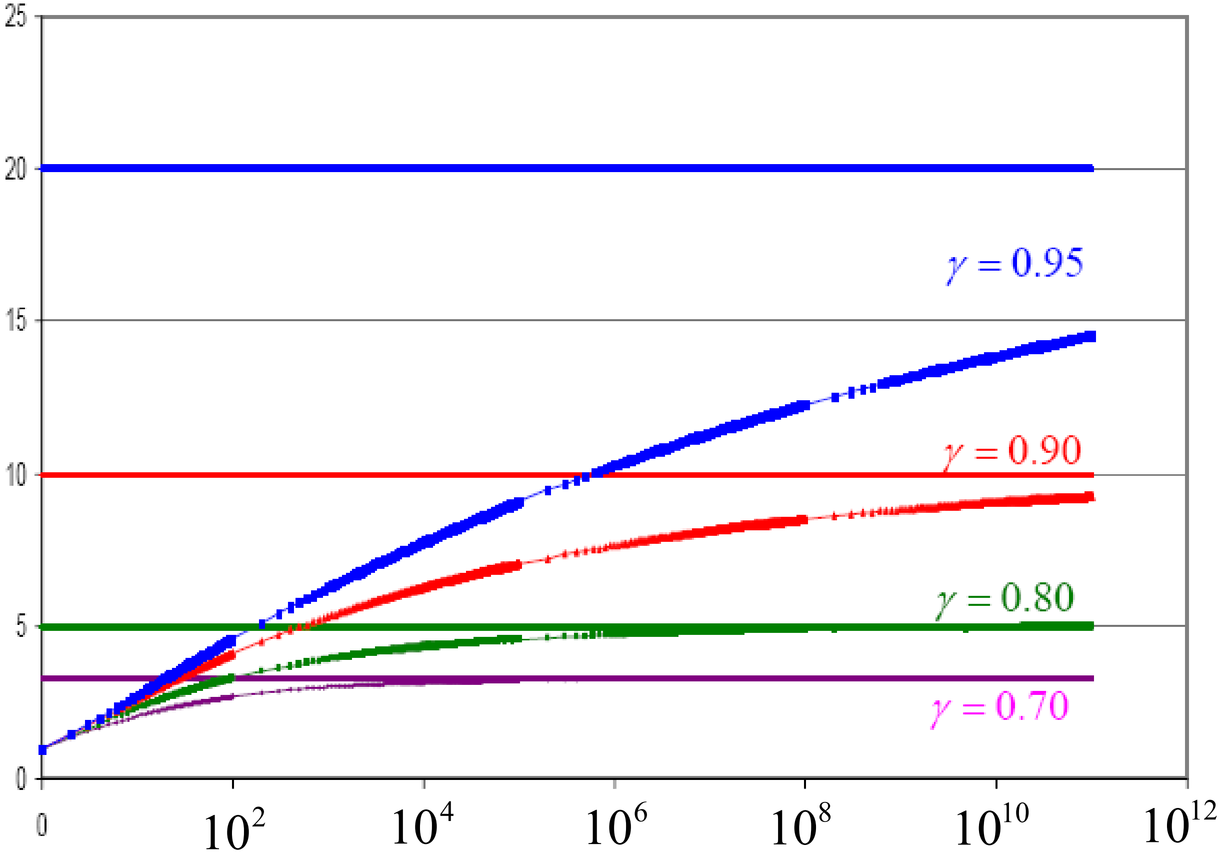


$$V = \sum_{n=0}^{\infty} \gamma^n c = \frac{c}{1-\gamma}$$

$$\hat{v}^{n+1} = c + \gamma \bar{v}^n$$

$$\bar{v}^{n+1} = (1-\alpha_n)\bar{v}^n + \alpha_n \hat{v}^{n+1}$$

$$\nu^L(n) = \frac{c}{1-\gamma}\left(1 - \left(\frac{1}{1+n}\right)^{1-\gamma}\right)$$



© 2019 Warren B. Powell

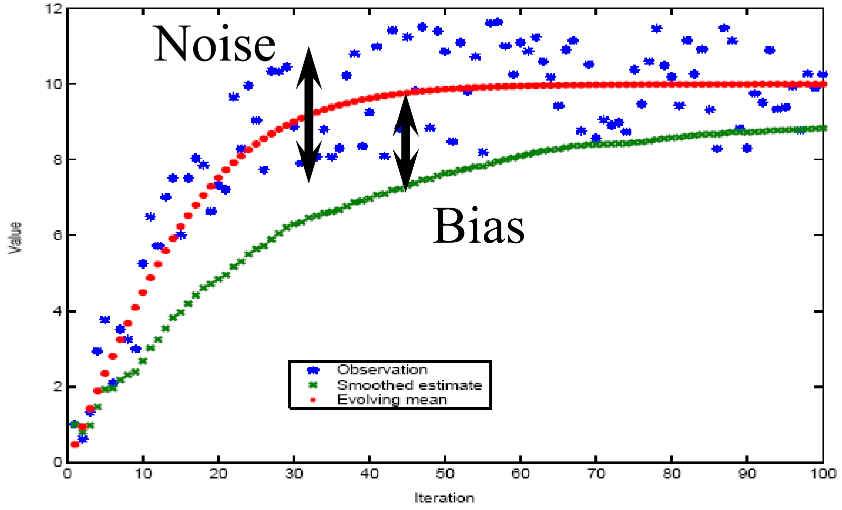
Stepsizes

- Bias-adjusted Kalman filter (BAKF)

Estimate of the variance

$$\alpha_n = 1 - \frac{\sigma^2}{\left(1+\lambda^{n-1}\right)\sigma^2 + \left(\beta^n\right)^2}$$

Estimate of the bias



where:

$$\lambda^n = \left(1-\alpha_n\right)^2 \lambda^{n-1} + \left(\alpha_n\right)^2$$

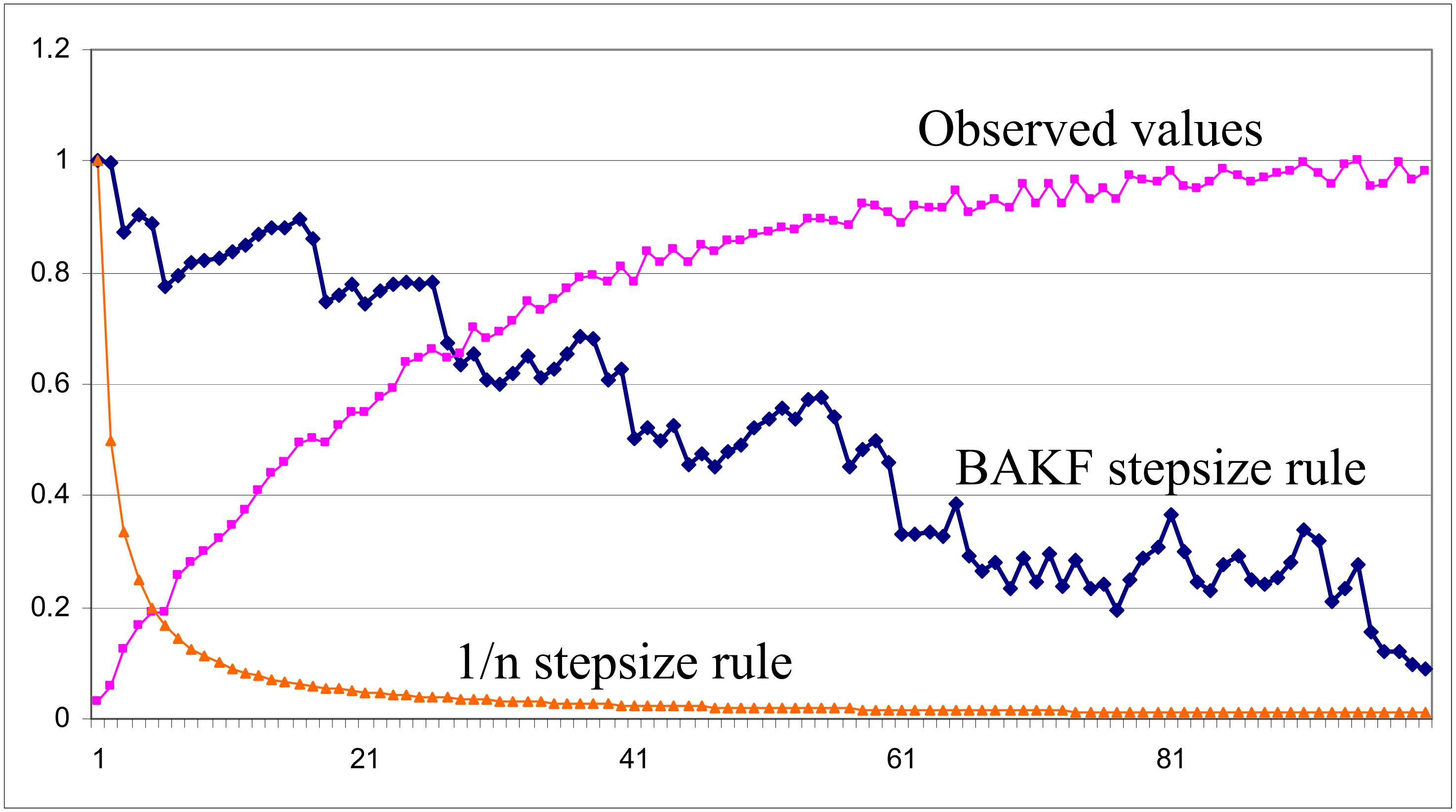
As σ^2 increases, stepsize decreases toward $1/n$

As β^n increases, stepsize increases toward 1.

At all times, $\alpha_n \geq \dfrac{1}{n}$

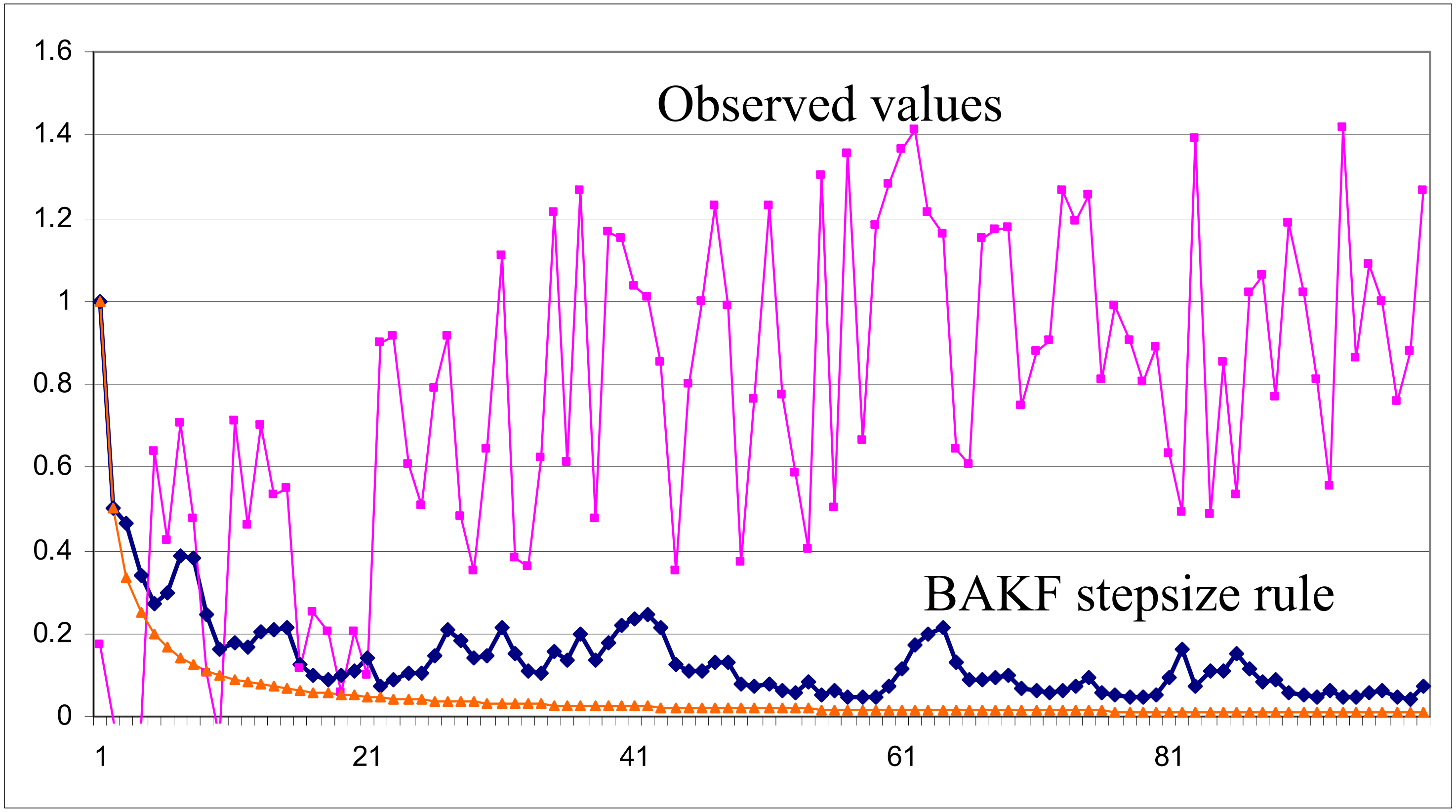
Stepsizes

- The bias-adjusted Kalman filter



Stepsizes

- The bias-adjusted Kalman filter



Stepsizes

- Notes:
  - » Because β and σ^2 have to be estimated from data, it is a stochastic stepsize policy. The challenge is estimating β .
  - » This stepsize rule is designed for a nonstationary time series. It does not recognize the feedback that arises in approximate value iteration.

Optimizing fleets

- The effect of stepsizes

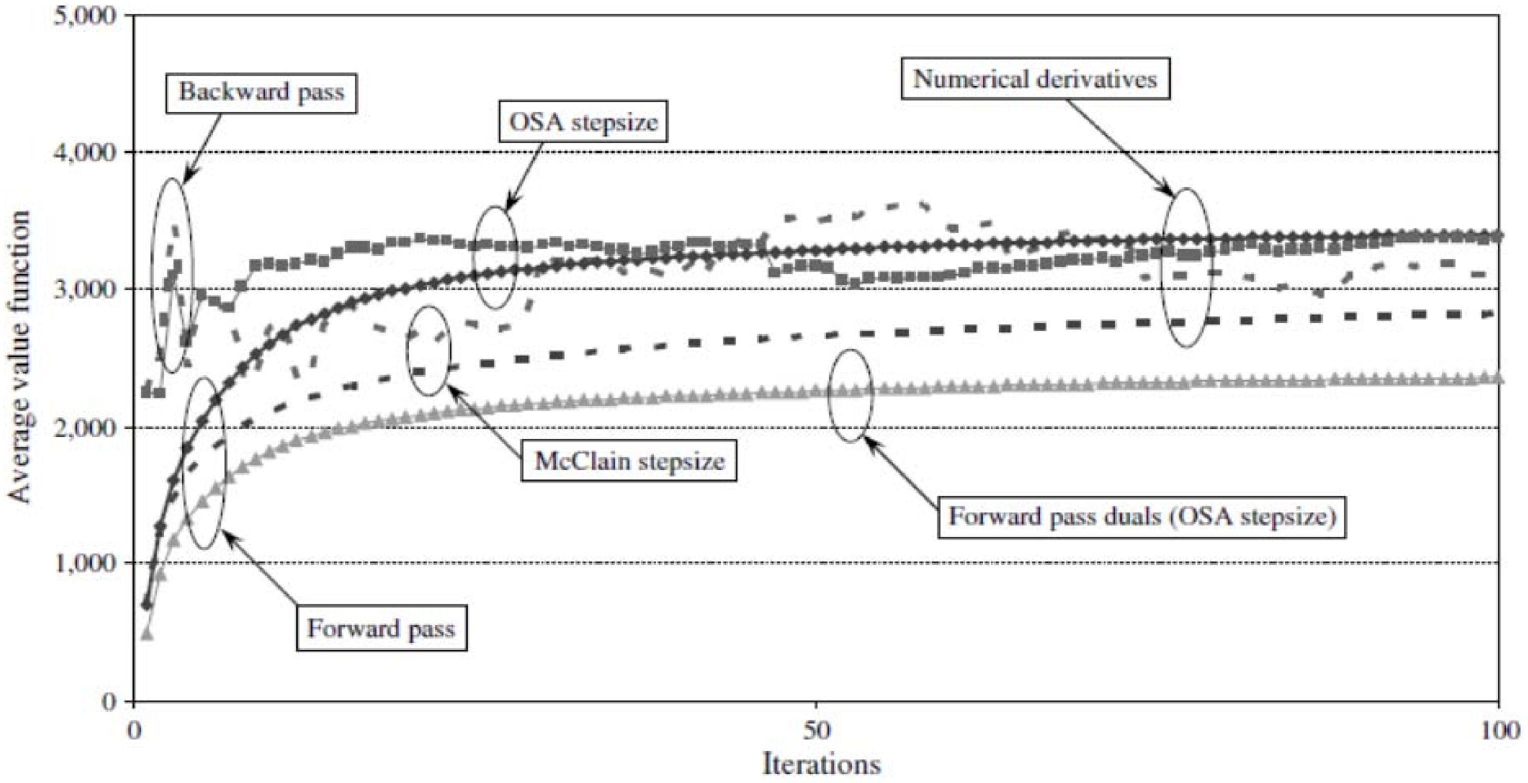
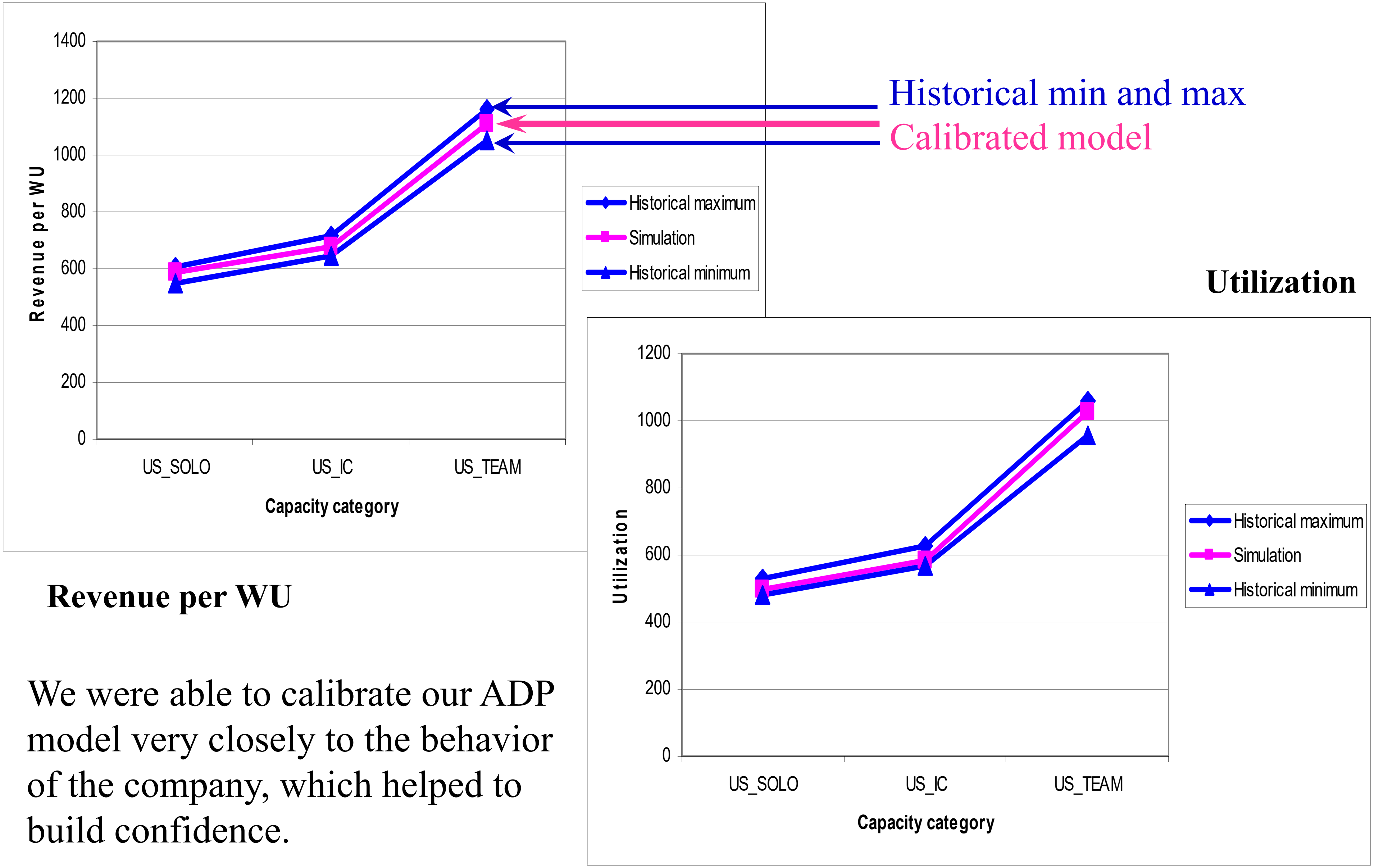


Figure 5 Average Value Function When We Use Forward and Backward Passes, Numerical Derivatives and Dual Variables, and the OSA Stepsize or the McClain Stepsize

Schneider National case study

Schneider National



Historical min and max

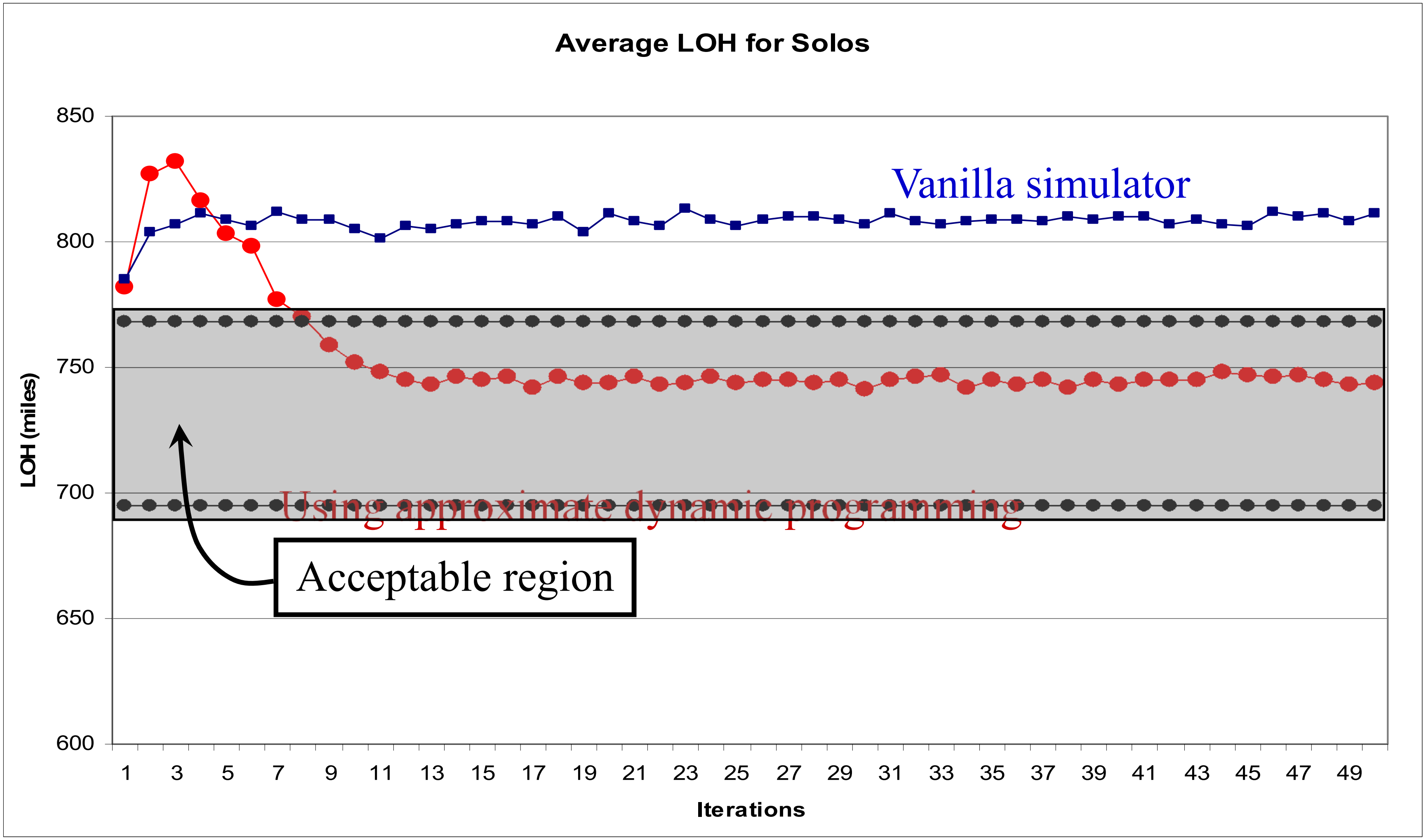
Calibrated model

Utilization

Revenue per WU

We were able to calibrate our ADP model very closely to the behavior of the company, which helped to build confidence.

Schneider National

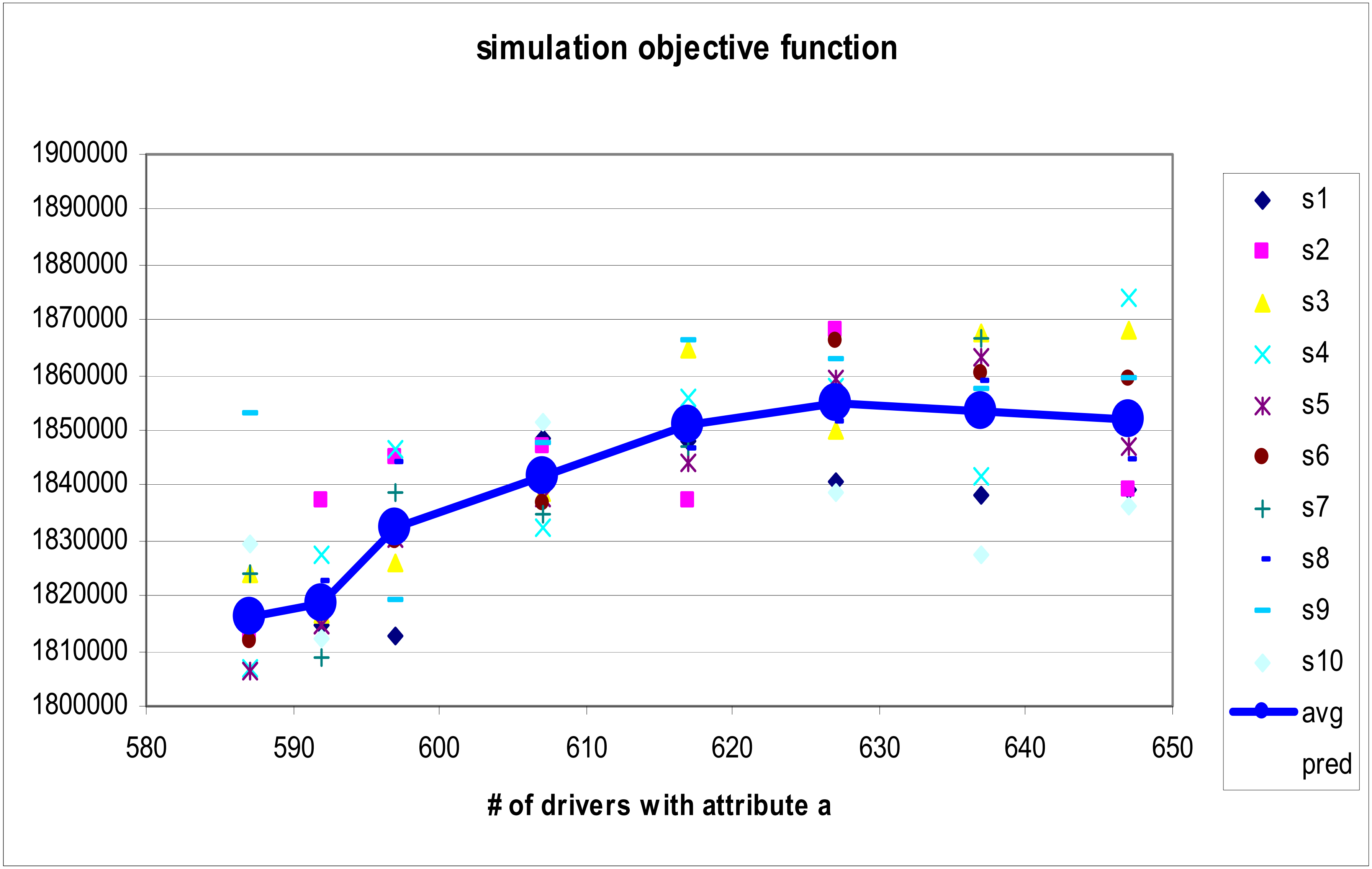


The adaptive learning of value functions produced results that more accurately matched historical performance.

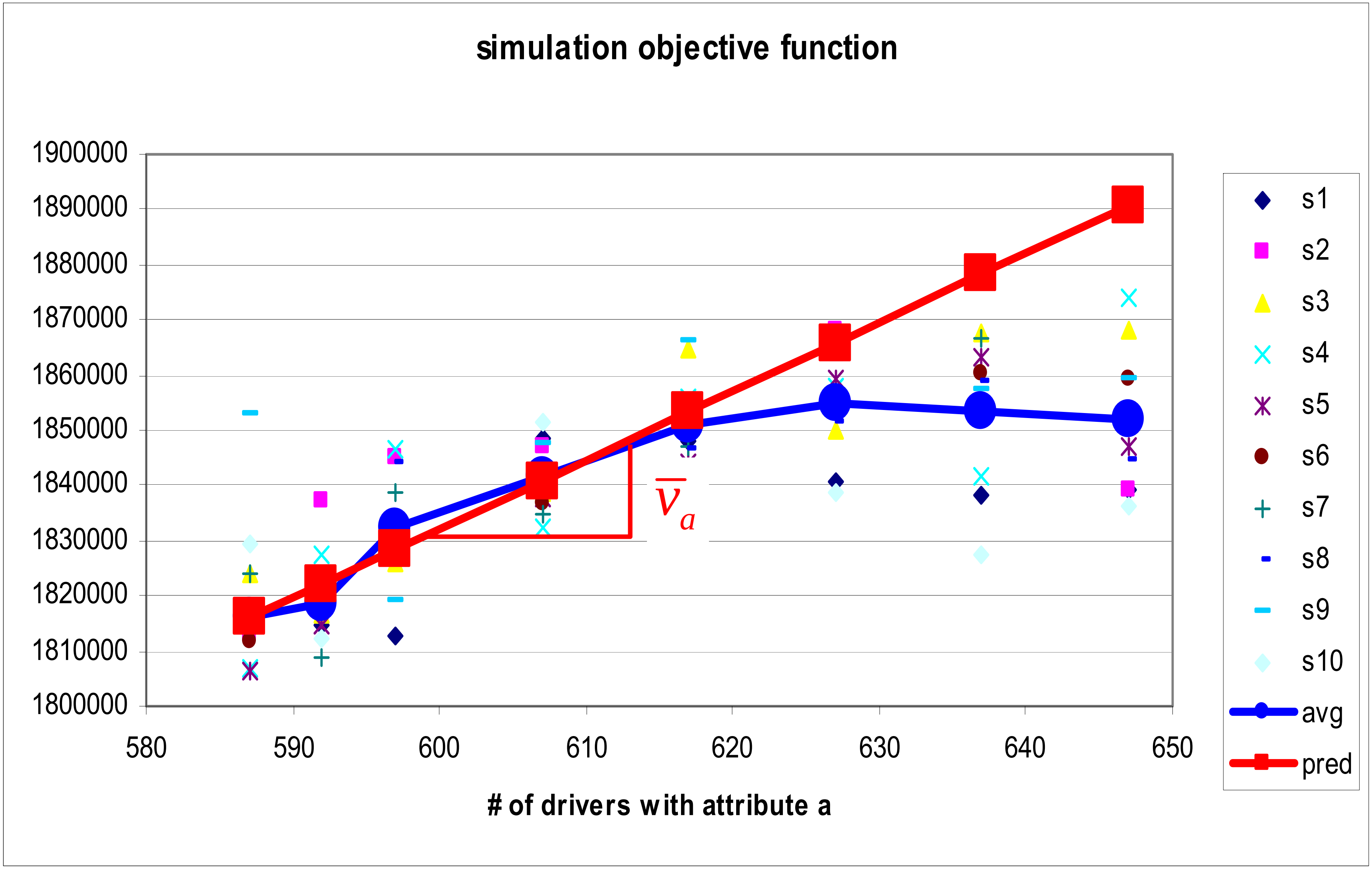
ADP for trucking

- Notes:
  - » We can use the value functions to estimate the marginal value of drivers whose home is in a particular region.
  - » In the next slides, we are going to show two ways to estimate these values:
    - In the first, we are going to run a series of (very expensive) simulations where we hire an increasing number of drivers in a region. Each time, we have to reoptimize the entire fleet over many iterations. We would have to repeat this for each possible home location.
    - In the second, we are going to estimate the marginal value by using the value function approximations to estimate the value of drivers in *each* region, from one run of the model.

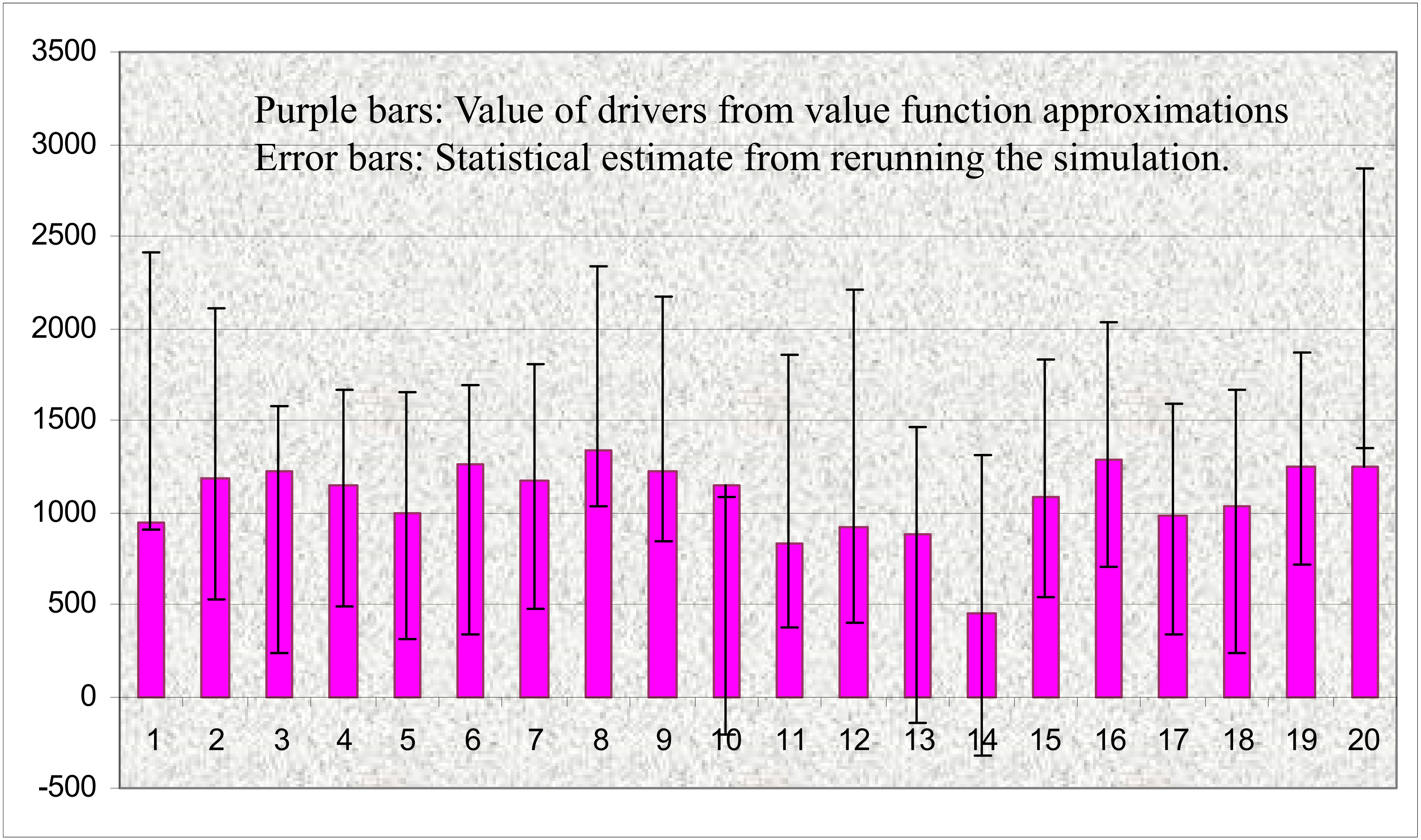
Case study: truckload trucking



Case study: truckload trucking



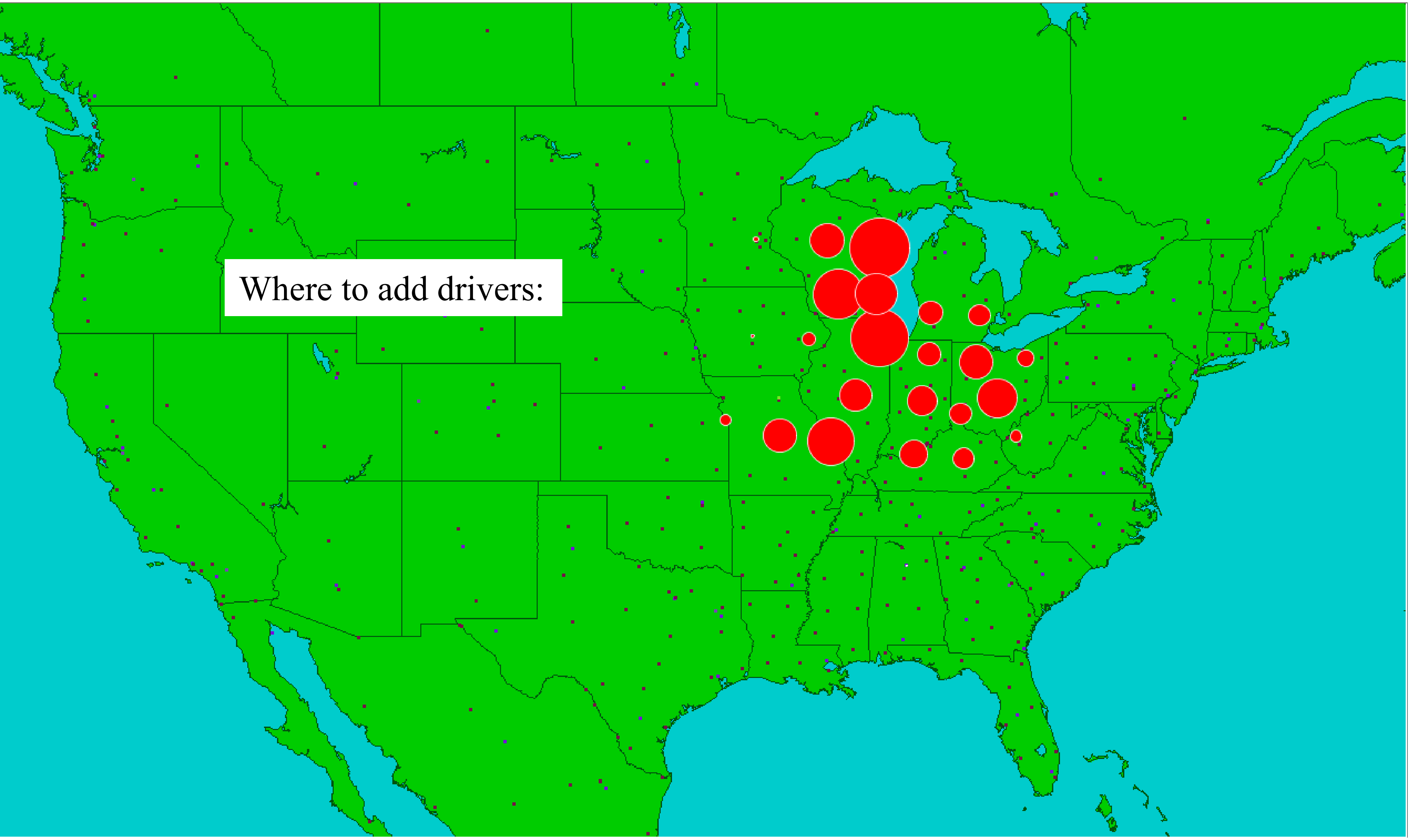
Case study: truckload trucking



This shows that the value functions provide reasonable approximations of the

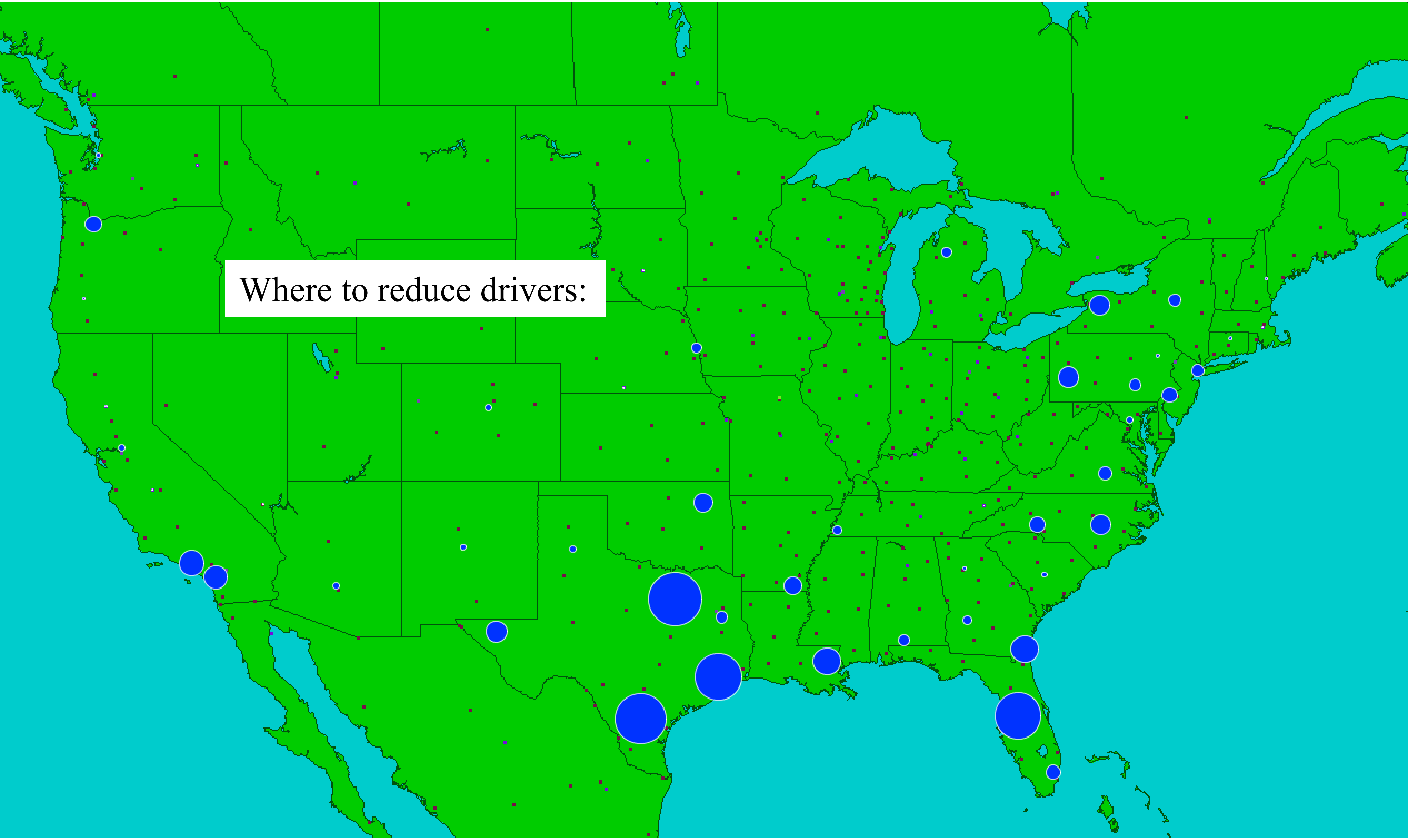
© 2019 Warren B. Powell

Case study: truckload trucking



Where to add drivers:

Case study: truckload trucking


Where to reduce drivers: