Tutorial: Approximate Dynamic Programming for High-Dimensional Resource Allocation

Informs National Meeting
Pittsburgh, 2006

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

© 2005 Warren B. Powell, Princeton University
The fractional jet ownership industry
Schneider National
Resource allocation

\[ a = \text{Attributes of the resource being managed.} \]
Resource allocation

\[ x_{ad} = \begin{cases} 
1 & \text{If we apply decision } d \text{ to resource } a \\
0 & \text{Otherwise}
\end{cases} \]
Planning for a Risky World

**Weather**

- Robust design of emergency response networks.
- Design of financial instruments to hedge against weather emergencies to protect individuals, companies, and municipalities.
- Design of sensor networks and communication systems to manage responses to major weather events.

**Disease**

- Models of disease propagation for response planning.
- Management of medical personnel, equipment and vaccines to respond to a disease outbreak.
- Robust design of supply chains to mitigate the disruption of transportation systems.
Blood management
Real-time control of financial systems
- Real-time asset pricing
- Real-time portfolio management

Robust planning of financial systems
- Risk measurement and mitigation
- Design of hedging strategies
- Robust portfolio design
Managing financial portfolios

- Money can be invested and then reinvested....
Machine learning

- Teaching computers to play games
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Languages

The languages of “optimization over time”

<table>
<thead>
<tr>
<th>Discipline</th>
<th>Engineering</th>
<th>OR/AI/Probability</th>
<th>OR/Math programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision (English)</td>
<td>Control</td>
<td>Action</td>
<td>Decision</td>
</tr>
<tr>
<td>Decision (math)</td>
<td>$u$</td>
<td>$a$</td>
<td>$x$</td>
</tr>
<tr>
<td>&quot;Value function&quot; (English)</td>
<td>Cost-to-go</td>
<td>Value function</td>
<td>Recourse function</td>
</tr>
<tr>
<td>&quot;Value function&quot; (Math)</td>
<td>$J$</td>
<td>$V$</td>
<td>$Q$</td>
</tr>
<tr>
<td>State variable</td>
<td>$x$</td>
<td>$S$</td>
<td>Huh? (oh, &quot;tenders&quot;)</td>
</tr>
<tr>
<td>Optimality equations</td>
<td>Hamilton-Jacobi</td>
<td>Bellman</td>
<td>Huh?</td>
</tr>
</tbody>
</table>
“Approximate dynamic programming” has been discovered independently by different communities under different names:

» Neuro-dynamic programming
» Reinforcement learning
» Forward dynamic programming
» Adaptive dynamic programming
» Heuristic dynamic programming
» Iterative dynamic programming
Languages

- How to land a plane:

  » Control: angle, velocity, acceleration, pitch, yaw…
  » Noise: wind, measurement

\[ V_t(x_t) = \max_u \left( C(x_t, u_t) + EV_{t+1}(x_{t+1}) \right) \]
Languages

- Where to send a plane:
  
  » Control: Where to send the plane to accomplish a goal.
  » Noise: demands on the system, equipment failures.
  
  \[ V_t(S_t) = \max_a \left( C(S_t, a_t) + EV_{t+1}(S_{t+1}) \right) \]
Languages

- How to manage a fleet of aircraft:

  - Control: Which plane to assign to each customer.
  - Noise: demands on the system, equipment failures.

\[ V_t(S_t) = \max_x \left( C(S_t, x_t) + EV_{t+1}(S_{t+1}) \right) \]
A progression of models

- Major problem classes

<table>
<thead>
<tr>
<th></th>
<th>Simple attributes</th>
<th>Complex attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single entity</td>
<td>Textbook Markov decision process</td>
<td>Classical AI applications</td>
</tr>
<tr>
<td>Multiple entities</td>
<td>Classical OR applications</td>
<td>Opportunity for combining AI/OR</td>
</tr>
</tbody>
</table>
Sample applications

■ Single entity problems
  » Playing a board game
  » Routing a truck around the country
  » Planning a set of courses through college

■ Asset acquisition (single asset class)
  » Maintaining product inventories
  » Purchasing commodity futures (oil, orange juice, …)

■ Managing multiple resource classes
  » Blood inventories
  » Fleet management (with different equipment types)

■ Managing multiple, discrete resources
  » Locomotives, jets, people
Single entity problems
Single entity problems
Single entity problems
Single entity problems
Single entity problems
Asset acquisition problems

- Inventory problem

\[ R_{t+1} = \left[ R_t + x_t - \hat{D}_{t+1} \right]^+ \]

\[ P_{t+1} = P_t + \hat{P}_t \]
Multiple inventory types

- Managing blood inventories
Multiple inventory types

- Managing blood inventories over time

Week 0  Week 1  Week 2

© 2006 Warren B. Powell
Multiple discrete assets
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Laying the foundation

Dynamic programming review:

Let:

- \( S_t = \) "State" of our "system" at time \( t \).
- \( x_t = \) "Action" that we take to change the system.

\[ C(S_t, x_t) = \text{Contribution earned when we take action } x \text{ from state } S_t. \]

A myopic policy maximizes our one-period reward, but ignores the downstream impact:

\[ x_t = \text{arg max}_x C_t(S_t, x_t) \]

Generally, this is not going to work very well.
Laying the foundation

- Modeling dynamic programs

$t = 0$ (Interval from 0 to 1) $t = 1$ (Interval from 1 to 2)

Decisions $x_0$ Information $W_1$ Decisions $x_1$ Information $W_2$
Dynamic programming review:

We need to model the transition from state $S_t$ to $S_{t+1}$. It is standard to assume that we have access to a one-step transition matrix:

$$p(S_{t+1} \mid S_t, x_t) = \text{Probability that action } x_t \text{ takes us from state } S_t \text{ to state } S_{t+1}$$

It is common in Markov decision processes to assume that the transition matrix is given as data.
Laying the foundation

Bellman’s equation:

» Standard form:

\[
V_t(S_t) = \max_x \left( C_t(S_t, x_t) + \sum_{s'} p(s' \mid S_t, x_t) V_{t+1}(S_{t+1} = s') \right)
\]

» Expectation form:

\[
V_t(S_t) = \max_x \left( C_t(S_t, x_t) + E \left\{ V_{t+1}(S_{t+1}(S_t, x_t)) \mid S_t \right\} \right)
\]
Use weather report

Forecast sunny .6
Forecast cloudy .3
Forecast rain .1

Decision nodes
Outcome nodes

- Schedule game
- Cancel game

- Rain .8  -$2000
- Clouds .2  $1000
- Sun .0  $5000

- Rain .8  -$200
- Clouds .2  -$200
- Sun .0  -$200
- Rain .1  -$200

- Clouds .5  $1000
- Sun .4  $5000
- Rain .1  -$200

- Clouds .5  -$200
- Sun .4  -$200
- Rain .1  -$200

- Clouds .2  $1000
- Sun .7  $5000
- Rain .1  -$200

- Clouds .2  -$200
- Sun .7  -$200

- Rain .2  -$2000
- Clouds .3  $1000
- Sun .5  $5000

- Rain .2  -$200
- Clouds .3  -$200
- Sun .5  -$200

- Sun .5  -$200
Do not use weather report

Weather report

Forecast sunny .6

Schedule game

Cancel game

Forecast cloudy .3

Schedule game

Cancel game

Forecast rain .1

Schedule game

Cancel game

Schedule game

Cancel game

-$1400

-$200

$2300

-$200

$3500

-$200

$2400

-$200

-$200
Do not use weather report

Use weather report

Forecast sunny .6

Schedule game

$2400

Cancel game

-$200

Forecast cloudy .3

$2300

Forecast rain .1

-$200

$3500
Use weather report

Do not use weather report

$2770

$2400
The curses of dimensionality

- We just solved Bellman’s equation:

\[
V_t(S_t) = \max_{x \in \mathcal{X}} C_t(S_t, x_t) + E \{ V_{t+1}(S_{t+1}) \mid S_t \}
\]

- We found the value of being in each state by stepping backward through the tree.
The curses of dimensionality

What happens if we apply this idea to our blood problem?

» State variable is:
  • The supply of each type of blood, along with its age
    – 8 blood types
    – 6 ages
    – = 48 “blood types”
  • The demand for each type of blood
    – 8 blood types

» Decision variable is how much of 48 blood types to supply to 8 demand types.
  • 216- dimensional decision vector

» Random information
  • Blood donations by week (8 types)
  • New demands for blood (8 types)
The curses of dimensionality

The challenge of dynamic programming:

\[
V_t(S_t) = \max_{x \in \mathcal{X}} \left( C_t(S_t, x_t) + E \{ V_{t+1}(S_{t+1}) \mid S_t \} \right)
\]
The curses of dimensionality

The computational challenge:

\[ V_t(S_t) = \max_{x \in \mathcal{X}} \left( C_t(S_t, x_t) + E\{ V_{t+1}(S_{t+1}) | S_t \} \right) \]
$V_t(S_t) = \max_{x \in \mathcal{X}} \left( C_t(S_t, x_t) + E \left\{ V_{t+1}(S_{t+1}) \mid S_t \right\} \right)$
Forecast sunny .6
Rain .8   -$2000
Clouds .2  $1000
Sun .0  $5000
Rain .8  -$200
Clouds .2  -$200
Sun .0  -$200

Forecast rain .1
Rain .1  -$2000
Clouds .2 $1000
Sun .7  $5000
Rain .1 -$200
Clouds .2 -$200
Sun .7 -$200

Forecast cloudy .3
Rain .2 -$2000
Clouds .3 $1000
Sun .5  $5000
Rain .2 -$200
Clouds .3 -$200
Sun .5 -$200

Schedule game
Cancel game

- Decision nodes
- Outcome nodes
Pre- and post-decision states

New concept:

» The “pre-decision” state variable:

- \( S_t^i = \) The information required to make a decision \( x_t \)
- Same as a “decision node” in a decision tree.

» The “post-decision” state variable:

- \( S_t^x = \) The state of what we know immediately after we make a decision.
- Same as an “outcome node” in a decision tree.
Pre- and post-decision states

- Modeling dynamic programs

\[ t = 0 \quad \text{(Interval from 0 to 1)} \quad t = 1 \quad \text{(Interval from 1 to 2)} \]

Decisions \( x_0 \)  \quad \text{Information} \quad W_1  \quad \text{Decisions} \quad x_1  \quad \text{Information} \quad W_2
Pre- and post-decision states

Post-decision states and Q-learning

» A popular algorithm in ADP is known as “Q-learning.”
» Instead of learning a value function, we learn the value of being in a state and taking an action:

Given a state \( n \), pick an action \( x^n \) and an outcome \( \omega^n \):

\[
\hat{q}_t^n = C(S^n_t, x^n_t) + V_{t+1} \left( S_{t+1}^n(S^n_t, x^n_t, W_{t+1}(\omega^n)) \right)
\]

Next update the estimate of the value of being in state \( S^n_t \) and taking action \( x^n_t \):

\[
Q_t^n(S^n_t, x^n_t) = (1 - \alpha_{n-1})Q_{t-1}^{n-1}(S^n_t, x^n_t) + \alpha_{n-1}\hat{q}_t^n
\]

» A state-action pair \((S, x)\) is mathematically equivalent to a pre-decision state, but computationally it is completely different.
Pre- and post-decision states

- Post-decision states and Q-learning

Pre-decision state | State | Action | Post-decision state
---|---|---|---

3⁹ states | 3⁹ × 9 state-action pairs | 3⁹ states

© 2006 Warren B. Powell
Pre- and post-decision states

“Resources”
Pre- and post-decision states

“Tasks”
Pre- and post-decision states

Pre- and post-decisions for a fleet management problem

\[ R_t = (R_t^{Drivers}, R_t^{Loads}) \]
Pre- and post-decision states programming

\[ R^x_t = R^{M,x}_t (R_t, x_t) \]
Pre- and post-decision states

\[ R_t^x \quad \quad \quad R_{t+1} = R^{M,W}_{t+1}(R_t^x, W_{t+1}) \]

\[ W_{t+1} = (\hat{R}_{t+1}^{\text{truck}}, \hat{R}_{t+1}^{\text{load}}) \]
Pre- and post-decision states

\[ R_{t+1} \]
System dynamics

- It is traditional to assume you are given the one-step transition matrix:

\[ p(S_{t+1} \mid S_t, x_t) = \text{Probability that action } x_t \text{ takes us from state } S_t \text{ to state } S_{t+1} \]

  » Computing the transition matrix is impossible for the vast majority of problems.

- We are going to assume that we are given a transition function:

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

  » This is at the heart of any simulation model.
  » Often rule-based. Very easy to compute, even for large-scale problems.
System dynamics

- Working with pre- and post-decision states
  - The “usual” transition function:

\[
S_{t+1} = S^M \left( S_t, x_t, W_{t+1} \right) \quad \text{From } S_t \text{ to } S_{t+1}.
\]

- The transition function broken into two steps:

\[
\begin{align*}
S^x_t &= S^{M,x} \left( S_t, x_t \right) \quad \text{The pure effect of a decision} \\
S_{t+1} &= S^{M,W} \left( S^x_t, W_{t+1} \right) \quad \text{The effect of the exogenous information}
\end{align*}
\]
Bellman’s equations

Bellman’s equations broken into stages:

» Optimization problem (making the decision):

\[ V_t(S_t) = \max_x \left( C_t(S_t, x_t) + V^x_t(S^{M,x}_t(S_t, x_t)) \right) \]

• Note: this problem is deterministic!

» Simulation problem (the effect of exogenous information):

\[ V^x_t(S^x_t) = \mathbb{E}\left\{ V_{t+1}(S^{M,W}_{t+1}(S^x_t, W_{t+1})) \mid S^x_t \right\} \]
Bellman’s equations

■ Challenges

» For most practical problems, we are not going to be able to compute $V_t^x(S_t^x)$.

$$V_t(S_t) = \max_x \left( C_t(S_t, x_t) + V_t^x(S_t^x) \right)$$

» Concept: replace it with an approximation $\overline{V}_t(S_t^x)$ and solve

$$V_t(S_t) = \max_x \left( C_t(S_t, x_t) + \overline{V}_t(S_t^x) \right)$$

» So now we face:
  • What should the approximation look like?
  • How do we estimate it?
ADP our way

The general algorithm

» Given the post-decision state $S_{t-1}^x$, sample the new information $W_t(\omega^n)$ and compute the pre-decision state

$$S_t^n = S_{t-1}^{M,W} \left( S_{t-1}^{x,n}, W_t(\omega^n) \right)$$

» Solve the optimization using an approximate value function:

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

to obtain $x_t^n$. Next post-decision state is $S_t^{x,n} = S_{t}^{M,x}(S_t^n, x_t)$

» Smooth the sample to form an approximate expectation:

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

To update $\overline{V}_{t-1}^{n}(S_{t-1}^{x,n})$
Competing updating methods

Comparison to other methods:

» Classical MDP (value iteration)

\[ V^n(S) = \max_x \left( C(S, x) + EV^{n-1}(S_{t+1}) \right) \]

» Classical ADP (pre-decision state):

\[
\hat{v}_t^n = \max_x \left( C_t(S^n_t, x_t) + \sum_{s'} p(s' \mid S^n_t, x_t) \bar{V}^{n-1}_{t+1}(s') \right)
\]

\[ V^n_t(S^n_t) = (1 - \alpha_{n-1}) \bar{V}^{n-1}_t(S^n_t) + \alpha_{n-1} \hat{v}_t^n \]

» Our method:

\[
\hat{v}_t^n = \max_x \left( C_t(S^n_t, x_t) + \bar{V}^{x,n-1}_{t-1}(S_{t-1}^{x,n}) \right)
\]

\[ \bar{V}^n_{t-1}(S_{t-1}^{x,n}) = (1 - \alpha_{n-1}) \bar{V}^{n-1}_{t-1}(S_{t-1}^{x,n}) + \alpha_{n-1} \hat{v}_t^n \]
Approximate Dynamic Programming

Solving the curses of dimensionality

Warren B. Powell

November 2, 2006

(c) Warren B. Powell, 2006 All rights reserved.

Department of Operations Research and Financial Engineering
Princeton University, Princeton, NJ 08544
Nomadic trucker illustration

- The previous post-decision state: trucker in Texas
Nomadic trucker illustration

- Pre-decision state: we see the demands
Nomadic trucker illustration

- We use initial value function approximations…
Nomadic trucker illustration

... and make our first choice: $x^1$
Nomadic trucker illustration

- Update the value of being in Texas.
Now move to the next state, sample new demands and make a new decision.
Nomadic trucker illustration

- Update value of being in NY

\[ V^0(\text{NY}) = 500 \]

\[ V^0(\text{CO}) = 0 \]

\[ V^0(\text{MI}) = 0 \]

\[ V^0(\text{CA}) = 0 \]

\[ V^1(\text{TX}) = 450 \]

\[ V(\text{CO}) = 0 \]

\[ V(\text{MI}) = 500 \]

\[ V(\text{CA}) = 125 \]

\[ V(\text{TX}) = 180 \]

Nomadic trucker illustration
Nomadic trucker illustration

- Move to California.

© 2006 Warren B. Powell
Nomadic trucker illustration

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

$V^0(CA) = 750$
$V^0(CO) = 0$
$V^0(MI) = 0$
$V^0(NY) = 500$
$V^1(TX) = 450$
$V^1(CA) = 300$
$V^1(CO) = 150$
$V^1(MI) = 300$
$V^1(NY) = 750$

© 2006 Warren B. Powell
Nomadic trucker illustration

- Back in TX, we repeat the process, observing a different set of demands.
Nomadic trucker illustration

- We get a different decision and a new estimate of the value of being in TX.
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?
\[ \bar{V}^2(TX) = (1 - \alpha)\bar{V}^1(TX) + (\alpha)\hat{v}^2(TX) \]
\[ = (0.90)\$450 + (0.10)\$800 \]
\[ = \$485 \]
Nomadic trucker illustration

- An updated value of being in TX

- $V^0(CO) = 0$
- $V^0(MI) = 0$
- $V^0(NY) = 600$
- $V^0(CA) = 750$
- $V^1(TX) = 485$
- $V^0(A) = 0$
- $V(CO) = 0$
- $V(MI) = 0$
- $V(NY) = 600$
- $V(CA) = 750$
- $V(TX) = 485$
- $V(A) = 0$

© 2006 Warren B. Powell
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
The states of our system

Now let’s take a look at what we just did:

\[ a_t = \text{Attribute of our nomadic trucker at time } t \text{ before the decision is made.} \]

\[ \hat{D}_t = \text{Vector of demands that are revealed at time } t. \]

\[ S_t = (a_t, \hat{D}_t) \quad \text{Pre-decision state variable.} \]

\[ a_t^x = \text{Attribute of our nomadic trucker at time } t \text{ after the decision is made.} \]

\[ S_t^x = a_t^x \quad \text{Post-decision state variable.} \]
A single, complex entity

- Pre- and post-decision attributes for our nomadic truck driver:

<table>
<thead>
<tr>
<th></th>
<th>City</th>
<th>ETA</th>
<th>Equip</th>
<th>$t = 40$ Pre-decision</th>
<th>$t = 40$ Decision</th>
<th>$t = 40$ Post-decision</th>
<th>$t = 50$ Pre-decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>City</td>
<td>Dallas</td>
<td>Chicago</td>
<td>Good</td>
<td>$41.2$</td>
<td>$-$</td>
<td>$54.7$</td>
<td>$-$</td>
</tr>
<tr>
<td>ETA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Equip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Multiple, complex entities

Notation for multiple entities:

» The truck state vector:

\[ a = \text{The attributes of the truck} \]
\[ a \in \mathcal{A} \quad \text{The attribute space} \]
\[ R_{ta}^{\text{truck}} = \text{The number of trucks with attribute } a \]
\[ R_{t}^{\text{truck}} = \left( R_{ta}^{\text{truck}} \right)_{a \in \mathcal{A}} \quad \text{The truck state vector} \]

» The information process:

\[ \hat{R}_{ta}^{\text{truck}} = \text{The change in the number of trucks with attribute } a. \]
Multiple, complex entities

- Modeling the fleet management problem:
  - The load state vector:
    - \( b = \) The attributes of a load to be moved.
    - \( b \in B \) The attribute space
    - \( R_{tb}^{load} = \) The number of tasks with attribute \( b \)
    - \( R_t^{load} = \left( R_{tb}^{load} \right)_{b \in B} \) The load state vector
  - The information process:
    - \( \hat{R}_{tb}^{load} = \) The change in the number of loads with attribute \( b \).
Multiple, complex entities

■ Modeling the fleet management problem:
  » The resource state vector (a.k.a. “physical state”)

\[
R_t = \left( R_t^{\text{truck}}, R_t^{\text{load}} \right)
\]

» The information process:

\[
\hat{R}_t = \text{The number of new arrivals (of drivers and loads) during time interval } t.
\]
\[
= \left( \hat{R}_t^{\text{truck}}, \hat{R}_t^{\text{load}} \right)
\]
\[
= W_t
\]
The states of our system

- The state of a single, simple entity:

\[ a = \text{[Location]} \]

\[ a \in \mathcal{A} \quad |\mathcal{A}| \approx 100 - 10,000 \]
The states of our system

- The state of a single, complex entity:

\[ a = \begin{bmatrix}
\text{Time} \\
\text{Location} \\
\text{Equipment type} \\
\text{Home base} \\
\text{Operator attributes} \\
\text{Time in service} \\
\text{Maintenance status}
\end{bmatrix} \]

\[ a \in \mathcal{A}, \quad |\mathcal{A}| \approx 10^{10} - 10^{100} \]

The curse of dimensionality!
The states of our system

- Multiple, complex entities

\[ R_t = \begin{bmatrix} R_{ta_1} \\ R_{ta_2} \\ \vdots \\ R_{ta_n} \end{bmatrix} \]

The number of *dimensions* of our state variable is equal to the size of the *state space* for a single entity problem.

The curse of curses.
The states of our system

- The three states of our system
  - The state of a single resource/entity
    \[ a_t = \begin{bmatrix} a_{t1} \\ a_{t2} \\ a_{t3} \end{bmatrix} \]
  - The state of all our resources
    \[ R_t = \begin{bmatrix} R_{ta1} \\ R_{ta2} \\ R_{ta3} \end{bmatrix} \]
  - The state of knowledge
    \[ S_t = \left( R_t, \bar{\theta}_t \right) \quad \bar{\theta}_t = \text{Estimates of "other parameters"} \]
The states of our system

Transition functions

» The attribute transition function:

\[ a_t^x = a_{M,x}^t (a_t, x_t) \] The pure effect of a decision
\[ a_{t+1}^x = a_{M,W}^t (a_t^x, W_{t+1}) \] The effect of the exogenous information

» The resource transition function

\[ R_t^x = R_{M,x}^t (R_t, x_t) \] The pure effect of a decision
\[ R_{t+1}^x = R_{M,W}^t (R_t^x, W_{t+1}) \] The effect of the exogenous information

» The general transition function:

\[ S_t^x = S_{M,x}^t (S_t, x_t) \] The pure effect of a decision
\[ S_{t+1}^x = S_{M,W}^t (S_t^x, W_{t+1}) \] The effect of the exogenous information
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Asset acquisition problems

- Inventory problem

\[ R_{t-1}^x (\hat{R}_t, \hat{D}_t) \]

\[ S_t x_t R_t^x \]

\[ t - 1 \quad t \quad t + 1 \]

\[ R_{t-1}^x = \text{Amount level over after decisions made in previous time period.} \]

\[ \hat{R}_t = \text{Random change to supply (donations, theft, cash inflow)} \]

\[ R_t = \text{Amount available at time } t \text{ before we make any decisions.} \]

\[ \hat{D}_t = \text{Demand that arises during time period } t. \]

\[ x_t^D = \text{Amount of } R_t \text{ used to satisfy demand } \hat{D}_t \text{ (} x_t^D \leq R_t, x_t^D \leq \hat{D}_t \text{)} \]

\[ x_t^O = \text{New amount ordered at time } t \text{ (for use in } t + 1). \]
Asset acquisition problems

Inventory problem

\[ R_{t-1}^x (\hat{R}_t, \hat{D}_t) S_t x_t R_t^x \]

States and transitions:

\[ R_t = R_{t-1}^x + \hat{R}_t \quad \text{Pre-decision resource state} \]

\[ R_t^x = R_t - x_t^D + x_t^O \quad \text{Post-decision resource state} \]

\[ S_t = (R_t, \hat{D}_t) \quad \text{State before we make a decision} \]

\[ S_t^x = (R_t^x) \quad \text{Post-decision state} \]
Asset acquisition problems

The ADP algorithm

Given $R_{t-1}^{x,n}$, randomly sample $\hat{R}_t^n$ and $\hat{D}_t^n$

$$R_t^n = R_{t-1}^{x,n} + \hat{R}_t$$

$$S_t = \left( R_t^n, \hat{D}_t^n \right)$$

Now find the decision:

$$x_t^n = \text{arg max}_{x_D^n, x_O^n} \left( px_t^D - cx_O + \bar{V}_t \left( R_t - x_D^n + x_O^n \right) \right)$$
Asset acquisition problems

- Updating the value function

Let

\[ F_t(R_t) = \max_{x_t^O, x_t^D} \left( px_t^D - cx_t^O + \bar{V}_t (R_t - x_t^D + x_t^O) \right) \]

Find the gradient with respect to \( R_t \):

\[ \hat{V}_t^n = \frac{dF_t(S_t, x_t^n)}{dR_t} \]

Instead of using the value of being in a state, we are using its derivative.

Use \( \hat{V}_t^n \) to update \( \bar{V}_{t-1}(R_{t-1}^x) \) at \( R_{t-1}^x = R_{t-1}^{x,n} \)
Asset acquisition problems

- Estimate the gradient at $R_t^n$
Asset acquisition problems

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

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

Approximate

\( \chi^* \)
Asset acquisition problems

Approximate
Asset acquisition problems
Asset acquisition problems
Asset acquisition problems

Approximate
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Blood management

- Managing blood inventories
Blood management

- Managing blood inventories over time
\( \hat{D}_{t,AB^+} \)

Satisfy a demand

Hold
\[ R_t \]

- \( R_{t,(AB+,0)} \) → \( AB+,0 \)
- \( R_{t,(AB+,1)} \) → \( AB+,1 \)
- \( R_{t,(AB+,2)} \) → \( AB+,2 \)
- ... → \( AB+,3 \)

\[ R_{t}^x \]

- \( R_{t,(AB-,0)} \) → \( AB-,0 \)
- \( R_{t,(AB-,1)} \) → \( AB-,1 \)
- \( R_{t,(AB-,2)} \) → \( AB-,2 \)
- ... → \( AB-,3 \)

\[ \hat{D}_t \]
Solve this as a linear program.
Dual variables give value additional unit of blood.

Dual variables = stochastic gradient.
Blood management

- Sequencing of decisions and information
Blood management
Blood management

$R_{t,1}^{x,n} \rightarrow \bullet$

$R_{t,2}^{x,n} \rightarrow \bullet$

$R_{t,3}^{x,n} \rightarrow \bullet$

$R_{t,4}^{x,n} \rightarrow \bullet$

$R_{t,5}^{x,n} \rightarrow \bullet$

© 2006 Warren B. Powell
Blood management

- We estimate the functions by sampling from our distributions.

Marginal value:

\[ \hat{v}_{t,1}(\omega^n) \rightarrow R_{t,1}^{x,n} \]

\[ \hat{v}_{t,2}(\omega^n) \rightarrow R_{t,2}^{x,n} \]

\[ \hat{v}_{t,3}(\omega^n) \rightarrow R_{t,3}^{x,n} \]

\[ \hat{v}_{t,4}(\omega^n) \rightarrow R_{t,4}^{x,n} \]

\[ \hat{v}_{t,5}(\omega^n) \rightarrow R_{t,5}^{x,n} \]
Blood management

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

\[
\overline{V}_{t-1,AB+}(R^x_{t-1,AB+})
\]

\[
R^{x,n}_{t-1,AB+}
\]
Blood management

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

\[ \overline{V}_{t-1, AB+} \left( R_{t-1, AB+}^x \right) \]

\[ \bar{V}_{t-1, AB+} \left( R_{t-1, AB+}^x \right) \]

Left derivative \quad Right derivative

\[ \tilde{v}_{-k} \quad \tilde{v}_{+k} \]

\[ R_{t-1, AB+}^{x,n} \]

© 2006 Warren B. Powell
Blood management

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

\[ t \]
Blood management
Blood management
Blood management
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Approximating value functions

- There are two ways of approximating the value function
  - Table lookup
    - Compute one value for each state.
    - Number of values we need to estimate = number of states.
  - Everything else:
    - Regression functions
    - Nonparametric statistics
    - Neural networks
Approximating value functions

Table lookup

» For a discrete state \( S_{t-1}^{x,n} \) we can update the value of being in this state:

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

» ADP replaces the statistical problem of looping over all states with the statistical problem of estimating the value of being in each state.

» We need to estimate not just the value of states we do visit, but also states that we might visit.
Aggregation for table lookup

\[ \bar{V}(a_1') \quad C(a, d_1) \quad C(a, d_2) \quad \bar{V}(a_2') \]
Aggregation for table lookup

\[ \nu_{PA} \approx \nu_{NE} \]

\( \overline{\nu}_{NE} \)

NE region

PA

TX
Aggregation for table lookup

- Different levels of aggregation

\[ A \approx \begin{bmatrix}
600,000 \\
6,000 \\
2,000
\end{bmatrix} \]

\[ a = \begin{bmatrix}
\text{Time} \\
\text{Location} \\
\text{Fleet} \\
\text{Domicile} \\
\text{Days from home}
\end{bmatrix} \]
Aggregation for table lookup

- Updating the value of a driver:

\[
\begin{align*}
\bar{v}^n(L, F, D) &= (1 - \alpha)\bar{v}^{n-1}(L, F, D) + \alpha \hat{v}(L, F, D, \text{DOThrs}, \text{DaysFromHome}) \\
&= (0.90)\$2000 + (0.10)\$2500 \\
&= \$2050
\end{align*}
\]

Value function approximation may have fewer attributes than driver.

Drivers may have very detailed attributes.
Aggregation for table lookup

- Estimating value functions
  » Most aggregate level

$$\bar{v}^n([Location]) = (1 - \alpha)\bar{v}^{n-1}([Location]) + \alpha\hat{v}([Fleet, Domicile, DOThrs, DaysFromHome])$$
Aggregation for table lookup

- Estimating value functions
  » Middle level of aggregation

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

» Most disaggregate level

\[
\bar{v}^n \left( \begin{bmatrix} Location \\ Fleet \\ Domicile \end{bmatrix} \right) = (1 - \alpha) \bar{v}^{n-1} \left( \begin{bmatrix} Location \\ Fleet \\ Domicile \end{bmatrix} \right) + \alpha \hat{v} \left( \begin{bmatrix} Location \\ Fleet \\ Domicile \\ DOThrs \\ DaysFromHome \end{bmatrix} \right)
\]
Aggregation for table lookup

- Using different levels of aggregation:
  - Pick the (single) level of aggregation that produces the best overall results.
  - Pick the level of aggregation that produces the lowest variance for each state.
  - Use a weighted sum of estimates at each level of aggregation (weight depends only on the level of aggregation):

\[
\overline{v}_a^n = \sum_g w^{(g,n)} \overline{v}_a^{(g,n)}
\]

- Use a weighted sum, but where the weights depend on the state (attribute):

\[
\overline{v}_a^n = \sum_g w^{(g,n)} \overline{v}_a^{(g,n)}
\]
Aggregation for table lookup

- State-dependent weighted aggregation:
  - There may be hundreds of thousands of weights, so these have to be easy to compute.

\[
\bar{v}_a = \sum_g w_a^{(g)} \bar{v}_a^{(g)} \quad \sum_g w_a^{(g)} = 1
\]

where

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

Variance of estimate \hspace{1cm} Estimate of bias squared

Both can be computed using simple recursive formulas.
Aggregation for table lookup

Weight on most disaggregate level

Weight on most aggregate levels

Iterations

© 2006 Warren B. Powell
Aggregation for table lookup
Aggregation for table lookup

- Weighted Combination
- Aggregate
- Disaggregate
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Basis functions

- Aggregation works well when we have a state (attribute) space with very little structure.
- But what if we have some structure? Consider our inventory problem:

\[
x^n_t = \arg \max_x \left( px^D_t - cx^O_t + \bar{V}_t (R^x_t) \right)
\]
Basis functions

- Approximating the value function:
  - We have to exploit the structure of the value function (e.g. concavity).
  - We might approximate the value function using a simple polynomial
    \[ \overline{V}_t (R_t | \theta) = \theta_0 + \theta_1 R_t + \theta_2 R_t^2 \]
  - .. or a complicated one:
    \[ \overline{V}_t (R_t | \theta) = \theta_0 + \theta_1 R_t + \theta_2 R_t^2 + \theta_3 \ln(R_t) + \theta_4 \sin(R_t) \]
  - Sometimes, they get really messy:
\[
\bar{V}_t(R | \theta) = \theta^{(0)} + \sum_s^{t+2} \sum_{t'=t}^{t+3} \theta_{t,st}^{(1)} R_{t, st} + \sum_w \sum_{t'=t}^{t+3} \theta_{t, wt}^{(1)} R_{t, wt} \\
+ \sum_w \sum_{t'}^{t+2} \theta_{t, wt}^{(2)} R_{t, wt}^2 + \sum_s \sum_{t'}^{t+2} \theta_{t, st}^{(2)} R_{t, st}^2 \\
+ \sum_s \theta_{t s}^{(3)} \left( R_{t, st} - \frac{1}{S} \sum_{s'} R_{t, s't} \right)^2 \\
+ \sum_s \theta_{t s}^{(4)} \left( \sum_{t'=t}^{t+1} R_{t, st'} \right) - \frac{1}{2S} \sum_{s'} \sum_{t'=t}^{t+1} R_{t, s't'}^2 \\
+ \sum_s \theta_{t s}^{(5)} \left( \sum_{t'=t}^{t+2} R_{t, st'} \right) - \frac{1}{3S} \sum_{s'} \sum_{t'=t}^{t+2} R_{t, s't'}^2 \\
+ \sum_w \sum_s \theta_{t, ws}^{(ws, 1)} R_{t, wt} R_{t, st} \\
+ \sum_w \sum_s \theta_{t, ws}^{(ws, 2)} R_{t, wt} \left( \sum_{t'=t}^{t+2} R_{t, st'} \right) \\
+ \sum_w \sum_s \theta_{t, ws}^{(ws, 3)} \left( \sum_{t'=t}^{t+3} R_{t, wt'} \right) \left( \sum_{t'=t}^{t+2} R_{t, st'} \right) \\
+ \sum_w \sum_s \theta_{t, ws}^{(ws, 4)} \left( \sum_{t'=t}^{t+2} R_{t, wt'} \right) \left( \sum_{t'=t}^{t+2} R_{t, st'} \right) \\
+ \sum_w \sum_s \theta_{t, sw}^{(ws, 5)} R_{t, s,t+2} R_{t, wt} R_{t, w,t+2} \\
\]
Basis functions

- We can write a model of the observed value of being in a state as:

\[
\hat{v} = \theta_0 + \theta_1 R_t + \theta_2 R_t^2 + \theta_3 \ln(R_t) + \theta_4 \sin(R_t) + \epsilon
\]

- This is often written as a generic regression model:

\[
Y = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \theta_4 X_4
\]

- The ADP community refers to the independent variables as basis functions:

\[
Y = \theta_0 \varphi_0(R) + \theta_1 \varphi_1(R) + \theta_2 \varphi_2(R) + \theta_3 \varphi_3(R) + \theta_4 \varphi_4(R)
\]

\[
= \sum_{f \in \mathcal{F}} \theta_f \varphi_f(R) \quad \varphi_f(R) \text{ are also known as features.}
\]
Basis functions

Methods for estimating $\theta$

» Generate observations $\hat{\nu}^1, \hat{\nu}^2, \ldots, \hat{\nu}^N$, and use traditional regression methods to fit $\theta$.

» Use recursive statistics - update $\theta^n$ after each iteration:

$$\theta^n = \theta^{n-1} - \alpha_{n-1} \left( \overline{V}^{n-1}(S^n \mid \theta^{n-1}) - \hat{\nu}^n \right) \nabla \overline{V}^{n-1}(S^n \mid \theta^{n-1})$$

$$= \theta^{n-1} - \alpha_{n-1} \left( \overline{V}^{n-1}(S^n \mid \theta^{n-1}) - \hat{\nu}^n \right) \begin{pmatrix} \phi_1(S) \\ \phi_2(S) \\ \phi_F(S) \end{pmatrix}$$

Error \hspace{1cm} Basis functions
Basis functions

Notes:

» When using basis functions, we are basically drawing on the entire field of statistics.
» Designing basis functions (independent variables) is mostly art.
» In special cases, the resulting algorithm can produce optimal solutions.
» Most of the time, we are hoping for “good” solutions.
» In some cases, it can work terribly.
» As a general rule – you have to use problem structure. Value function approximations have to capture the right structure. Blind use of polynomials will rarely be successful.
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Stepsizes

- Stepsizes:
  » Fundamental to ADP is an updating equation that looks like:

\[
V_{t-1}^n(S_{t-1}^x) = (1 - \alpha_{n-1})V_{t-1}^{n-1}(S_{t-1}^x) + \alpha_{n-1}\hat{\nu}_t^n
\]

Updated estimate \hspace{1cm} Old estimate \hspace{1cm} New observation

The stepsize
“Learning rate”
“Smoothing factor”
Stepsizes

■ Theory:
  » Many convergence results require:

\[
\sum_{n=1}^{\infty} \alpha_{n-1} = \infty
\]

\[
\sum_{n=1}^{\infty} (\alpha_{n-1})^2 < \infty
\]

  » For example:

\[
\alpha_{n-1} = \frac{1}{n}
\]

■ Practice
  » Constant stepsizes
  » Various stepsize rules
    • Deterministic
    • Stochastic
Rate of convergence

Smoothed estimate using $1/n$
Rate of convergence

The challenge of stepsizes:
» When have we converged?

We need to improve our understanding of adaptive stepsizes.
Stepsizes

- Deterministic stepsize rules:

\[ \alpha^n = \frac{1}{n} \text{ or } \frac{1}{n^\beta} \]  
Basic averaging

\[ \alpha^n = \frac{a}{a + n - 1} \]  
Slows the rate of descent

\[ \alpha^n = \frac{b}{b + a} \]  
"Search then converge"

\[ \alpha^n = \frac{n}{b + a + n^\beta} \]  

\[ \alpha^n = \frac{\alpha^n}{1 + \alpha^n - \bar{\alpha}} \]  
McClain's formula
Stepsizes

Increasing “a”
Stepsizes

- Estimates of value functions tend to go through a transient phase before settling in:
Stepsizes

- Stochastic stepsize rules:

  Let: \( \varepsilon^n = \text{New observation} - \text{old estimate} \)

  Kesten's rule:

  \[
  \alpha^n = \alpha^0 \frac{a}{a + K^n} \quad K^n = K^{n-1} + 1_{\{\varepsilon^n \varepsilon^{n-1} < 0\}}
  \]

  Tunable parameters

  Stochastic gradient adaptive stepsize (Benveniste et al.)

  \[
  \alpha^n = \left[\alpha^{n-1} + a \varepsilon^n \varphi^n\right]_{\alpha^{\min}}^{\alpha^{\max}}
  \]

  where:

  \[
  \varphi^n = (1 - \alpha^{n-1}) \varphi^n + \varepsilon^n
  \]
Stepsizes

Kalman filter for zero-mean drift:

Based on model that assumes we are estimating a parameter $\theta^n$ that is evolving randomly over time:

$$
\theta^{n+1} = \theta^n + \delta^n \quad E\delta^n = 0, \quad \text{Var}\delta^n = (\sigma^\delta)^2
$$

$$
\hat{\theta}^{n+1} = \theta^n + \epsilon^n \quad E\epsilon^n = 0, \quad \text{Var}\epsilon^n = \sigma^2
$$

Stepsise using Kalman filter:

$$
\alpha^n = \frac{(\sigma^\delta)^2}{(\sigma^\delta)^2 + \sigma^2}
$$

where:

$$
(\bar{\beta}^n)^2 = (1 - \alpha^n)(\bar{\beta}^{n-1})^2 + \beta^2
$$
Stepsizes

- Kalman filter for nonzero-mean drift:

In dynamic programming, observations of the value function often drift upward (or downward), giving us

$$\theta^{n+1} = \theta^n + \delta^n \quad E\delta^n \neq 0, \quad \text{Var}\delta^n = (\sigma^\delta)^2$$
Stepsizes

Kalman filter for nonzero-mean drift:

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

where: $$\lambda^n = (1 - \alpha^n)^2 \lambda^{n-1} + (\epsilon_n)$$

As $$\sigma^2$$ increases, stepsize decreases; as $$\beta^n$$ increases, stepsize increases.
Stepsizes

- Deterministic data: predictions and stepsizes
Stepsizes

Low noise stochastic data: predictions
Stepsizes

- Low noise stochastic data: stepsizes
Stepsizes

- High noise stochastic data: predictions
Stepsizes

- High noise stochastic data: stepsizes
Stepsizes

- Finite horizon batch replenishment

![Graph showing stepsizes with comparison to true value and optimal stepsize. The graph compares the optimized harmonic stepsize sequence to the true value.](image-url)
Stepsizes

- Finite horizon batch replenishment problem

![Graph showing percentage error from optimal versus average number of observations per state. The graph compares different methods: OSA, Kalman, STC, and Berveniste. The OSA method shows the least percentage error.](image)
Stepsizes

A steady state problem:

![Graph showing relative error for different stepsizes and discount values]
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
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
Exploration vs. exploitation

- Exploration vs. exploitation with the nomadic trucker

  » Pure exploitation
Information collection

Pure exploitation
Exploration vs. exploitation

- The state variable:

\[ S_t = \left( a_t, \left( \bar{V}_{t}^{n-1}, \bar{\sigma}_{t}^{n-1} \right) \right) \]

"Resource state"  Knowledge state
Resource allocation

\[
w_a^0 \bar{V}^0 (a_{11}) + w_a^1 \bar{V}^1 \begin{pmatrix} a_{11} \\ a_{12} \end{pmatrix} + w_a^2 \bar{V}^2 \begin{pmatrix} a_{11} \\ a_{12} \\ a_{13} \end{pmatrix}
\]
Exploration vs. exploitation

Strategies for overcoming the exploitation trap

» Generalization:
  • Visit one state, learn something about other states
  • Exploitation with more general learning
Information collection

Pure exploration
Outline

- The languages of dynamic programming
- The post-decision state variable
- The states of our system
- Examples:
  - An inventory problem
  - Managing blood inventories
- Approximating value functions
  - Table lookup
  - Basis functions
- Stepsizes
- Exploration vs. exploitation
- Simulating complex systems
Aggregation

Benefits:
» Sometimes we can dramatically reduce the state space using aggregation.

Functions that depend on the state variable:
» Value function
» Transition function
» Contribution function
» Decision function (e.g. constraints)
Aggregation

- Aggregation:
  - Exact methods
    - We have to use the same level of aggregation throughout (in particular, the transition matrix and value function).

\[
V_t(S_t) = \max_x \left( C_t(S_t, x_t) + \sum_{s'} p(s'|S_t, x_t) V_{t+1}(s') \right)
\]

Same level of aggregation
Aggregation

Approximate DP

» We only need to discretize the value function. We can capture the full state variable in the transition function:

• Decision function:

\[ x_t = \arg \max_x \left( C_t(S_t, x_t) + \tilde{V}_t(S_t^x) \right) \]

• Transition functions

\[
\begin{align*}
S_t &= S_{t-1}^{M,W}(S_t^x, W_t(\omega)) \\
S_t^x &= S_{t}^{M,x}(S_t, x_t)
\end{align*}
\]
Resource allocation

\[ \overline{V} \begin{pmatrix} \text{Location} \\ \text{Fleet type} \end{pmatrix} \]

Location
ETA
Fleet
Domicile
DOThrs
DaysFromHome
Equip status
Resource allocation

<table>
<thead>
<tr>
<th>Location'</th>
<th>ETA'</th>
<th>Fleet</th>
<th>Domicile</th>
<th>DOThrs'</th>
<th>DaysFromHome'</th>
<th>Equip status'</th>
</tr>
</thead>
</table>

© 2006 Warren B. Powell
Asset acquisition problems

Inventory problem

\[ x_t = \arg \max_x \left\{ C_t\left(\begin{array}{c} $1,283.57 \\ $5.493 \\ $459.72 \end{array}\right), (x) + V_t\left(\begin{array}{c} $1,200 + x \\ $5.500 \end{array}\right)\right\} \]

Detailed state (R,P,D)

Aggregated post-decision state
Low aggregation
Slow, but small errors

Exact results

High aggregation
Fast, but high errors due to aggregation

Low aggregation
Slow, but small errors
High aggregation produces fast run times with much smaller statistical errors.

Low aggregation produces high errors due to statistical problems.

ADP

Exact results
Schneider National
Case study: truckload trucking

Revenue per WU

Utilization

Historical maximum
Simulation
Historical minimum

Historical maximum
Simulation
Historical minimum
Case study: truckload trucking

Average LOH for Solos

Vanilla simulator

Using approximate dynamic programming

Acceptable region