Week 11

Forward approximate dynamic programming
Approximate dynamic programming

Exploiting monotonicity
Exploiting monotonicity

Monotonicity in dynamic programming

» There are many problems where the value function increases (or decreases) monotonically with all dimensions of a state variable:

» Operations research
- Optimal equipment replacement – Replace when a measure of deterioration exceeds some point
- Dispatching customers in a queue – Costs increase monotonically with the number of customers in the queue. Dispatch when the queue is over some number.
Exploiting monotonicity

Monotonicity in dynamic programming (cont’d)

» Energy
  • Buy when electricity price is below one number, sell when above another number:

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

Monotonicity in dynamic programming (cont’d)

» Health
  • Dosage of a diabetes drug increases with blood sugar.
  • Dosage of statins (for reducing cholesterol) increase as the cholesterol level increases (and also increases with age, weight).

» Finance
  • The value of holding cash in a mutual fund increases with the amount of redemptions, stock indices, and interest rates.
  • The value of holding an option increases with price and volatility.
Exploiting monotonicity

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

The exact value function

© 2019 Warren B. Powell
Exploiting monotonicity

Approximate value function without monotonicity
Exploiting monotonicity

- Maintaining monotonicity

○ ○ = observations

0

5

10

© 2019 Warren B. Powell
Exploiting monotonicity

- Maintaining monotonicity

\( \Pi_M \)

\( 0 \)

\( 5 \)

\( 10 \)

© 2019 Warren B. Powell
Exploiting monotonicity

Maintaining monotonicity

○ ○ = observations

0

5

10

"© 2019 Warren B. Powell"
Exploiting monotonicity

Maintaining monotonicity

= observations

0

5

10

© 2019 Warren B. Powell
Exploiting monotonicity

Monotonicity Preservation Example

© 2019 Warren B. Powell
Exploiting monotonicity

Benchmarking

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

Observations

» Initial experiments using various machine learning approximations produced poor results (60-80 percent of optimal)

» Vanilla lookup table works poorly (within computing budgets that spanned days)

» Lookup table with monotonicity worked quite well

» We have run this algorithm on problems with up to 7 dimensions (but that is our limit)
Backward ADP

Backward ADP for a clinical trial problem

» Problem is to learn the value of a new drug within a budget of patients to be tested.

» Backward MDP required 268-485 hours.

» Forward ADP exploiting monotonicity (we will cover this later) required 18-30 hours.

» Backward ADP required 20 minutes, with a solution that was 1.2 percent within optimal.

<table>
<thead>
<tr>
<th>$\frac{n_1}{n_3}$</th>
<th>$\frac{n_2}{n_3}$</th>
<th>MDP CPU (hrs)</th>
<th>ADP CPU (hrs)</th>
<th>Style</th>
<th>Gap (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.35</td>
<td>0.65</td>
<td>360</td>
<td>26</td>
<td>forward</td>
<td>3.2</td>
</tr>
<tr>
<td>0.40</td>
<td>0.65</td>
<td>290</td>
<td>18</td>
<td>forward</td>
<td>0.1</td>
</tr>
<tr>
<td>0.40</td>
<td>0.75</td>
<td>440</td>
<td>30</td>
<td>forward</td>
<td>-3.2</td>
</tr>
<tr>
<td>0.55</td>
<td>0.85</td>
<td>485</td>
<td>18</td>
<td>forward</td>
<td>3.7</td>
</tr>
<tr>
<td>0.55</td>
<td>0.85</td>
<td>268</td>
<td>0.3</td>
<td>backward</td>
<td>1.2</td>
</tr>
</tbody>
</table>

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

© 2019 Warren B. Powell
Approximate dynamic programming

Driverless EV problem
Driverless EV optimization

Current Uber logic:
» Show nearest 8 drivers.
» Contact closest driver to confirm assignment.
» If driver does not confirm, contact second closest driver.

Limitations:
» Ignores potential future opportunities for each driver.
Driverless EV optimization

Assigning a car in a less dense area allows a closer car to handle potential demands in more dense areas.

Closest car... but it moves a car away from a busy downtown area and strands another car in a low density area.
Driverless EV optimization
Driverless EV optimization

The central operator should think about:

• Should it accept this trip?
• What is the best car to assign to the trip considering
  – The type of car
  – The charge level of the battery

» If it doesn’t assign a car to a trip, should it:
  • Sit where it is?
  • Reposition to a better location?
  • Recharge the battery?
  • Move to a parking facility?
Driverless EV optimization
Driverless EV optimization
The assignment of cars to riders evolves over time, with new riders arriving, along with updates of cars available.
Driverless EV optimization

- Policies based on value function approximations
  \[ X_t^{VFA}(S^n_t) = \arg \max_{x_t} \left( C(S^n_t, x_t) + \tilde{V}^{n-1}_t(S^{M,x_t}(S^n_t, x_t)) \right) \]

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

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

© 2019 Warren B. Powell
Driverless EV optimization

\[ v(a_1') \]

\[ v(a_1'') \]

\[ \mathbf{c}_{\mathbf{t}_{a_1}, \mathbf{r}_1} \]

\[ \mathbf{c}_{\mathbf{t}_{a_1}, \mathbf{r}_2} \]
Driverless EV optimization
Driverless EV optimization

© 2019 Warren B. Powell
Driverless EV optimization
Driverless EV optimization
Driverless EV optimization

New car assignments:

New future values

\(a_1\) \(a_2\) \(a_3\) \(a_4\)
Driverless EV optimization
Driverless EV optimization

New solution

$\text{New solution}$
Driverless EV optimization

\[ \text{Difference} = \hat{v}(a_1) \]

\[ \hat{v}(a_1) \rightarrow v(a'_1) - v(a''_1) \]

\[ a_1 \rightarrow v(a'_1) \]

\[ a_2 \rightarrow v(a'_2) - v(a''_2) \]

\[ a_3 \rightarrow v(a'_3) - v(a''_3) \]

\[ a_4 \rightarrow v(a'_4) - v(a''_4) \]
Driverless EV optimization

Difference in assignment costs

Changes in future values

\[ v(a_1') - v(a_2') \]

\[ v(a_2') - v(a_3') \]

\[ v(a_3') - v(a_4') \]

\[ v(a_1) - v(a_2) \]
Driverless EV optimization

Assignment network

» Capture the value of downstream driver.

» $a^M(a_3, d_1) = \text{Attribute vector of driver in the future given a decision } d.$

» Add this value to the assignment arc.
Driverless EV optimization

Finding the marginal value of a driver:

» Dual variables
  - Can provide unreliable estimates.
  - Need to get the marginal value of drivers who are not actually there.

» Numerical derivatives:
Driverless EV optimization

Estimating average values:

\[ \bar{v}^n (a) = (1 - \alpha) \bar{v}^{n-1} (a) + (\alpha) \hat{v}^n (a) \]

- Old estimate of the value of a car with attribute \( a \)
- New estimate of the value of a cars with attribute \( a \)
- Attribute of car
Driverless EV optimization

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

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

Step 1: Start with a pre-decision state \( S^t_n \).

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

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

to obtain \( x^n \).

Step 3: Update the value function approximation

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

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

\[
S^t_{n+1} = S^{M}(S^t_n, x^n_t, W_{t+1}(\omega^n))
\]

Step 5: Return to step 1.

“on policy learning”
Driverless EV optimization

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

© 2019 Warren B. Powell
Driverless EV optimization

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

*Profit versus number of iterations*
Driverless EV optimization

- Three levels of aggregation - Better

![Graph showing profit versus number of iterations](image-url)
Driverless EV optimization

Five levels of aggregation – old VFAs

Profit versus number of iterations
Driverless EV optimization

- Five levels of aggregation – new VFAs

![Graph showing profit versus number of iterations](image)
Driverless EV optimization

- Trip requests over time
  - Challenge is to recharge during off-peak periods
Driverless fleets of EVs using ADP

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

Heuristic recharging logic vs. Recharging controlled by approximate dynamic programming

- Trip requests
- Actual trips
- Total battery charge
- Recharging during peak period
- No recharging during peak period
The economics of driverless fleets

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

Exploiting convexity in an inventory problem
Exploiting convexity

Single inventory problem

» With pre-decision state:
\[ x_t^n = \arg \max_{x_t \in X} \left( C(S_t^n, x_t) + \gamma \mathbb{E}\left\{ \bar{V}_{t+1}^{n-1} \left( S_{t+1}^n(S_t^n, x_t, W_{t+1}) \right) \left| S_t \right\} \right) \]

» With post-decision state:
\[ x_t^n = \arg \max_{x_t \in X} \left( C(S_t^n, x_t) + \gamma \bar{V}_t^{x, n-1} \left( S_t^x(S_t^n, x_t) \right) \right) \]
Exploiting convexity

Updating strategies

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

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

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

- Now compute the marginal value using

\[
\hat{\nu}^n_t = \frac{\hat{V}_t^n (S_t^n + \delta) - \hat{V}_t^n (S_t^n)}{\delta}
\]

- Use this to update the piecewise linear value functions.
Exploiting convexity

Ω Updating strategies

» Double pass:

• Perform forward pass simulating the policy for \( t = 0, ..., T \):

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

• Now perturb state by \( \delta \) and solve again:

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

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

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

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

\[
\hat{\nu}_t^n = \delta C(S_t^n, x_t^n) + \hat{\nu}_{t+1}^n
\]

\[
\bar{V}_{t-1}^n(S_{t-1}^n) \leftarrow U^V (\bar{V}_{t-1}^{n-1}(S_{t-1}^n), S_{t-1}^n, \hat{\nu}_t^n)
\]
Exploiting convexity

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

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

From pre- to post-decision state

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

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

© 2019 Warren B. Powell
Exploiting convexity

Updating strategies

» The choice between a pure forward pass (TD(0)) versus a double pass (TD(1)) is highly problem dependent. Every problem has a natural “horizon.”

» We have found that in our fleet management problems, the pure forward pass works fine and is much easier to implement.

» For energy storage problems, it is essential that we use a double pass, since a decision at time $t$ can have an impact hundreds of time periods into the future.
Exploiting convexity

- It is important to maintain concavity:
Exploiting convexity

A concave function…

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

\[ \bar{v}^n_{it} = (1 - \alpha)\bar{v}^{n-1}_{it} + \alpha \bar{v}^n_{it} \]
Exploiting convexity

\[ \bar{v}_{it}^n = (1 - \alpha)\bar{v}_{it}^{n-1} + \alpha \bar{v}_{it}^n \]
Exploiting convexity

\[ \bar{v}_{it}^n = (1 - \alpha)\bar{v}_{it}^{n-1} + \alpha \hat{v}_{it}^n \]
Exploiting convexity

Ways for maintaining concavity (monotonicity in the slopes):

» CAVE algorithm – Use updated slope at one point to update over a range $+/−\delta^n$ which shrinks with iterations (shrinking factor is tunable parameter). Expand $\delta$ if needed to ensure concavity is maintained.
  • Works well! But we could never prove convergence.

» Leveling algorithm – Update at a point $x^n$.
  • Force monotonicity in slopes by increasing/decreasing slopes farther from $x^n$ as needed.
  • Works fine, without tunable parameters

» SPAR algorithm – Perform nearest point projection onto space of monotone functions.
  • Nice theoretical convergence proof – could never get it to work.
Exploiting convexity

- Derivatives are used to estimate a piecewise linear approximation
Exploiting convexity

With luck, your objective function improves

![Graph showing the objective function improving over iterations]
Exploiting convexity

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

- Stochastic, time-dependent problems

![Graph showing benchmark storage problems](image-url)
Exploiting convexity

Stochastic, time-dependent problems
Approximate dynamic programming

Resource allocation (freight cars) for two-stage problem
Forecasts of Car Demands

Forecasts vs. Actuals for Car Demand from May 9th to June 27th, 2000.

© 2004 Warren B. Powell
Two-stage problems

- Two-stage resource allocation under uncertainty

© 2004 Warren B. Powell
Optimization frameworks

We obtain piecewise linear recourse functions for each region.
Optimization frameworks

- The function is piecewise linear on the integers.

![Graph showing profits vs. number of vehicles at a location.](image)
Optimization frameworks

Using standard network transformation:

Each link captures the marginal reward of an additional car.
Two-stage problems
Two-stage problems
Two-stage problems

\[ R_1^n \rightarrow \]
\[ R_2^n \rightarrow \]
\[ R_3^n \rightarrow \]
\[ R_4^n \rightarrow \]
\[ R_5^n \rightarrow \]
Two-stage problems

We estimate the functions by sampling from our distributions.

Marginal value:

\[ v_1(\omega^n) \rightarrow R_1^n \rightarrow D_1(\omega^n) \]
\[ v_2(\omega^n) \rightarrow R_2^n \rightarrow D_2(w^n) \]
\[ v_3(\omega^n) \rightarrow R_3^n \rightarrow D_3(w^n) \]
\[ v_4(\omega^n) \rightarrow R_4^n \rightarrow \ldots \]
\[ v_5(\omega^n) \rightarrow R_5^n \rightarrow D_C(w^n) \]
Two-stage problems

- The time $t$ subproblem:

\[
\tilde{V}_{ta}^n(R_{t1}, R_{t2}, R_{t3})
\]

Gradients:

- $(\hat{v}_{t1}^{n-}, \hat{v}_{t1}^{n+})$ to $R_{t1}$
- $(\hat{v}_{t2}^{n-}, \hat{v}_{t2}^{n+})$ to $R_{t2}$
- $(\hat{v}_{t3}^{n-}, \hat{v}_{t3}^{n+})$ to $R_{t3}$
Two-stage problems

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

\[ V_{ta}^n (R_{t1}, R_{t2}, R_{t3}) \]

Gradients:

\[ (\hat{v}_{t3}^{n+}) \]

The right derivative (the value of one more unit of that resource) is a flow augmenting path from that node to the supersink.
Two-stage problems

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

\[ \bar{V}_{it}^k (R_{1t}) \]
Two-stage problems

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

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

Approximate value function vs. Number of resources

- Exact
- 1 Iter
- 2 Iter
- 5 Iter
- 10 Iter
- 15 Iter
- 20 Iter

Variable Value, $s$

Functional Value, $f(s) = \ln(1+s)$

© 2004 Warren B. Powell
What we can prove

- SPAR algorithm

- Leveling algorithm

But so far, our fastest convergence is with the original version, called the CAVE algorithm, for which convergence has not been proven.
Approximate dynamic programming

Blood management problem
Management multiple resources
Blood management

- Managing blood inventories over time

Week 0, Week 1, Week 2, Week 3

\( x_0 \rightarrow S_0 \rightarrow \hat{R}_1, \hat{D}_1 \rightarrow S_1 \rightarrow x_1 \rightarrow S_2 \rightarrow \hat{R}_2, \hat{D}_2 \rightarrow S_2^x \rightarrow x_2 \rightarrow S_3 \rightarrow \hat{R}_3, \hat{D}_3 \rightarrow S_3^x \rightarrow x_3 \rightarrow \)

\( t=0 \rightarrow t=1 \rightarrow t=2 \rightarrow t=3 \)

© 2018 Warren B. Powell
\[ S_t = \begin{pmatrix} R_t \\ \hat{D}_t \end{pmatrix} \]

\[ R_t^{x} \]

\[ R_{t,(AB+,0)} \to AB+,0 \]
\[ R_{t,(AB+,1)} \to AB+,1 \]
\[ R_{t,(AB+,2)} \to AB+,2 \]
\[ R_{t,(O-,0)} \to O-,0 \]
\[ R_{t,(O-,1)} \to O-,1 \]
\[ R_{t,(O-,2)} \to O-,2 \]

\[ \hat{D}_{t,AB+} \]
\[ \hat{D}_{t,AB-} \]
\[ \hat{D}_{t,A+} \]
\[ \hat{D}_{t,A-} \]
\[ \hat{D}_{t,B+} \]
\[ \hat{D}_{t,B-} \]
\[ \hat{D}_{t,O+} \]
\[ \hat{D}_{t,O-} \]

\[ \text{Satisfy a demand} \]

\[ \text{Hold} \]

© 2018 Warren B. Powell
Solve this as a linear program.
Dual variables give value additional unit of blood..

\[ F(R_t) \]
Updating the value function approximation

- Estimate the gradient at $R_t^n$
Updating the value function approximation

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

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

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

\[ \overline{V}_{t-1}(R_{t-1}^{x}) \]

\[ \overline{V}_{t-1}^{n-1}(R_{t-1}^{x}) \]
Updating the value function approximation

Notes:

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

- Derivatives are used to estimate a piecewise linear approximation.

\[ \overline{V}_t(R_t) \]

© 2018 Warren B. Powell
Approximate value iteration

Step 1: Start with a pre-decision state $S^n_t$

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

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

to obtain $x^n_t$ and dual variables $\hat{\omega}_i^n$.

Step 3: Update the value function approximation

$$\bar{V}^n_{t-1}(S^{x^n}_{t-1}) = (1 - \alpha_{n-1})\bar{V}^{n-1}_{t-1}(S^{x^n}_{t-1}) + \alpha_{n-1}\hat{\omega}^n_t$$

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

$$S^n_{t+1} = S^M(S^n_t, x^n_t, W_{t+1}(\omega^n))$$

Step 5: Return to step 1.

© 2018 Warren B. Powell
Approximate value iteration

Step 1: Start with a pre-decision state $S^*_t$

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

$$\max_x \left( C_t(S^*_t, x_t) + \tilde{V}^{n-1}_t(S^M_t(x(S^*_t, x_t))) \right)$$

to obtain $x^n_t$ and dual variables $\tilde{\nu}^n_{ii}$. 

Deterministic optimization
Approximate value iteration

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

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

\[
\max_{x_t} \left( C_t(S_t^n, x_t) + \mathbb{V}_{t-1}^{n-1}(S_t^{M, x}(S_t^n, x_t)) \right)
\]

to obtain \( x_t^n \) and dual variables \( \hat{\nu}_{ii}^n \).
Approximate value iteration

Step 1: Start with a pre-decision state $S^n_t$

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

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

to obtain $x^n_t$ and dual variables $\hat{\nu}^n_{ii}$.

Step 3: Update the value function approximation

Step 4: Obtain Monte Carlo sample of $\hat{V}^n_t(S^n_t, x_t)$ and compute the next pre-decision state:

Step 5: Return to step 1.
Iterative learning
Iterative learning
Iterative learning
Iterative learning
Approximate dynamic programming

Grid level storage
Imagine 25 large storage devices spread around the PJM grid:
Optimizing battery storage
Value function approximations
Value function approximations

Monday

Time:

05

010

015

020

© 2019 Warren B. Powell
Value function approximations
Value function approximations

© 2019 Warren B. Powell
Exploiting concavity

Derivatives are used to estimate a piecewise linear approximation
Approximate dynamic programming

... a typical performance graph.
Grid level storage control

Without storage
Grid level storage control

- With storage
Approximate dynamic programming

Locomotive application
The locomotive assignment problem
The locomotive assignment problem

Horsepower  Locomotives

4400  4400  6000  4400  5700  4600  6200

Consist-breakup costs
Shop routing bonuses/ penalties
Leader logic

© 2019 Warren B. Powell
The locomotive assignment problem

Locomotives coming in on same train

Horsepower  Locomotives

4400  
4400  
6000  
4400  
4400  
5700  
4600  
6200  

© 2019 Warren B. Powell
The locomotive assignment problem

Train reward function

<table>
<thead>
<tr>
<th>Horsepower</th>
<th>Locomotives</th>
</tr>
</thead>
<tbody>
<tr>
<td>4400</td>
<td>4400</td>
</tr>
<tr>
<td>4400</td>
<td>4400</td>
</tr>
<tr>
<td>6000</td>
<td>4400</td>
</tr>
<tr>
<td>4400</td>
<td>4400</td>
</tr>
<tr>
<td>5700</td>
<td>5700</td>
</tr>
<tr>
<td>4600</td>
<td>4600</td>
</tr>
<tr>
<td>6200</td>
<td>6200</td>
</tr>
</tbody>
</table>

© 2019 Warren B. Powell
The locomotive assignment problem

- The train reward function

![Diagram showing the train reward function with goals and minimums labeled](image-url)
The locomotive assignment problem

Train may need 12,130 horsepower. Solutions:

1. $4400 + 4400 + 6000 = 14,400$
2. $4400 + 4400 + 4400 = 13,200$
3. $6000 + 6200 = 12,200$

© 2019 Warren B. Powell
The locomotive assignment problem

Train may need 12,130 horsepower. Solutions:

- $4400 + 4400 + 6000 = 14,400$
- $4400+4400+4400 = 13,200$
- $6000+6200 = 12,200$

© 2019 Warren B. Powell
The locomotive assignment problem

<table>
<thead>
<tr>
<th>Horsepower</th>
<th>Locomotives</th>
</tr>
</thead>
<tbody>
<tr>
<td>4400</td>
<td>💜</td>
</tr>
<tr>
<td>4400</td>
<td>💜</td>
</tr>
<tr>
<td>6000</td>
<td>💜</td>
</tr>
<tr>
<td>4400</td>
<td>💜</td>
</tr>
<tr>
<td>4400</td>
<td>💜</td>
</tr>
<tr>
<td>5700</td>
<td>💜</td>
</tr>
<tr>
<td>4600</td>
<td>💜</td>
</tr>
<tr>
<td>6200</td>
<td>💜</td>
</tr>
</tbody>
</table>

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

© 2019 Warren B. Powell
The locomotive assignment problem

Locomotive subproblem can be solved quickly using Cplex
Iterative learning

- Approximate dynamic programming
Iterative learning

- Approximate dynamic programming
Iterative learning

- Approximate dynamic programming
Approximate dynamic programming

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

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

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

to obtain $x_t^n$. 

© 2019 Warren B. Powell
Approximate dynamic programming

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

Step 2: Solve the deterministic optimization using an approximate value function:
$$\hat{v}_t^n = \max_x \left( C_t(S_t^n, x_t) + V_{t-1}^{n-1}(S_t^{M,x}(S_t^n, x_t)) \right)$$

to obtain $x_t^n$.

Step 3: Update the value function approximation
$$\bar{V}_{t-1}^{n}(S_t^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{n-1}(S_t^{x,n}) + \alpha_{n-1}\hat{v}_t^n$$

Step 4: Obtain Monte Carlo sample of $W_t(\omega^n)$ and compute the next pre-decision state:
$$S_{t+1}^n = S_t^M(S_t^n, x_t^n, W_{t+1}(\omega^n))$$

Step 5: Return to step 1.
Laboratory testing

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

- Train delay with uncertain transit times and yard delays

![Graph showing train delays using deterministically and stochastically trained VFAs](image)
How do we do it?

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

The stochastic model keeps more power in inventory. The challenge is knowing when and where.
Experiments with ADP for energy storage

“Does anything work”
Energy storage model

\[ C(S_t, x_t) = -P_t(x_t^{GR} + x_t^{GL} - \eta x_t^{RG}) \]
A storage problem

- Energy storage with stochastic prices, supplies and demands.

\[ E_{wind}^t \]
\[ P_{grid}^t \]
\[ W_{t+1} = \text{Exogenous inputs} \]
\[ S_t = \text{State variable} \]
\[ x_t = \text{Controllable inputs} \]

© 2019 Warren B. Powell
A storage problem

Bellman’s optimality equation

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

These are the “three curses of dimensionality.”
Approximate policy iteration

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

Step 2: Inner loop: Do for $m=1,\ldots,M$:

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.

© 2019 Warren B. Powell
Approximate policy iteration

- Machine learning methods (coded in R)
  - **SVR** - Support vector regression with Gaussian radial basis kernel
  - **LBF** – Weighted linear combination of polynomial basis functions
  - **GPR** – Gaussian process regression with Gaussian RBF
  - **LPR** – Kernel smoothing with second-order local polynomial fit
  - **DC-R** – Dirichlet clouds – Local parametric regression.
  - **TRE** – Regression trees with constant local fit.
Approximate policy iteration

Test problem sets

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

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

LBF – regression trees
LBF – linear basis functions
LPR – kernel smoothing
DC-R – local parametric regression
GPR - Gaussian process regression
SVR - Support vector regression

© 2019 Warren B. Powell
Energy storage applications

DC-R – local parametric regression
LPR – kernel smoothing
GPR - Gaussian process regression
SVR - Support vector regression

© 2019 Warren B. Powell
A tale of two distributions

» The *sampling distribution*, which governs the likelihood that we sample a state.

» The *learning distribution*, which is the distribution of states we would visit given the current policy.
Approximate policy iteration

Using the optimal value function

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

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

![State distribution under optimal policy](#)

![Divergence of VFA's 1](#)

![Divergence of VFA's 2](#)

VFA estimated after 50 policy iterations

VFA estimated after 51 policy iterations
Observations

» Experiments using a number of machine learning approximations produced poor results (60-80 percent of optimal)

» Policy search using a simple error correction term works surprisingly well (for this problem)

» Using low-order polynomial approximations fit using state distributions simulated using the optimal policy works poorly!
A Comparison of Approximate Dynamic Programming Techniques on Benchmark Energy Storage Problems: Does Anything Work?

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

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

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

“Is the RL glass half full, or half empty?”
Least squares approximate policy iteration

…and comparison to policy search
Least squares API vs. policy search

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

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

- Using the post-decision state

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

- In steady state, we might write

\[
\bar{V}^x(S^x_{t-1}) = \mathbb{E}\left\{ \max_x C(S^x, x) + \gamma \bar{V}^x(S^x_t) \mid S^x_t \right\}
\]
Least squares API vs. policy search

- Least squares policy iteration (Lagoudakis and Parr)
  - Bellman’s equation (infinite horizon)
    \[
    V^x(S_{t-1}^x) = \mathbb{E}[\max_x\{C(S_t, x) + \gamma V^x(S_t^x)\}|S_{t-1}^x]
    \]
  - … is equivalent to (for a fixed policy)
    \[
    \phi(S_{t-1}^x)^T \theta = \mathbb{E}[C(S_t, X^\pi(S_t|\theta)) + \gamma \phi(S_t^x)^T \theta|S_{t-1}^x].
    \]
  - Rearranging gives:
    \[
    C(S_t, X^\pi(S_t|\theta)) \overset{C_{t,i}}{=} \left(\phi(S_{t-1}^x) - \gamma \mathbb{E}[\phi(S_t^x)|S_{t-1}^x]\right)^T \theta + C(S_t, X^\pi(S_t|\theta)) - \mathbb{E}[C(S_t, X^\pi(S_t|\theta))|S_{t-1}^x]
    \]
  - … where “X” is our explanatory variable. But we cannot compute this exactly (due to the expectation) so we can sample it.

Need to sample introduces “errors in variable” formulation

© 2019 Warren B. Powell
Least squares API vs. policy search

Algorithm performance as a percent of optimal

Myopic policy

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

© 2019 Warren B. Powell
Least squares API vs. policy search

Algorithm performance as a percent of optimal

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

© 2019 Warren B. Powell
Least squares API vs. policy search

Variations of LSAPI:

Instrumental Variables Bellman Error Minimization (LSTD)

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

Least-Squares Projected Bellman Error Minimization

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

Instrumental Variables Projected Bellman Error Minimization

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

» Theorem (W. Scott and W.B.P.) Bellman error using instrumental variables and projected Bellman error minimization are the same!
Least squares API vs. policy search

Algorithm performance as a percent of optimal

Myopic policy
LSAPI

For benchmark datasets, see: http://www.castlelab.princeton.edu/datasets.htm
© 2019 Warren B. Powell
Least squares API vs. policy search

Algorithm performance as a percent of optimal

Myopic policy
LSAPI
LSAPI-Proj. Bellman

For benchmark datasets, see: [http://www.castlelab.princeton.edu/datasets.htm](http://www.castlelab.princeton.edu/datasets.htm)
Least squares API vs. policy search

Finding the best policy ("policy search")

» Assume our policy is given by

\[ X^\pi (S_t \mid \theta) = \arg \max_x \left( C(S^n_t, x_t) + \sum_f \theta_f \phi_f (S^x_t (S^n_t, x_t)) \right) \]

» We wish to minimize the function

\[
\max_\theta \mathbb{E} F(\theta) = \mathbb{E} \sum_{t=0}^T \gamma^t C \left( S_t, X^\pi (S_t \mid \theta) \right)
\]
Least squares API vs. policy search

Two solution methods

» Forward approximate dynamic programming
  • Use least squares approximate policy iteration (LSAPI) to solve
  \[
  V^x(S^x_{t-1}) = \mathbb{E}\left\{ \min_x \left( C(S_t, x_t) + V^x(S^x_t(S_t^x, x_t)) \right) \mid S^x_{t-1} \right\}
  \]
  \[
  \sum_f \theta_f \phi_f(S^x_{t-1}) \approx \mathbb{E}\left\{ \min_x \left( C(S_t, x_t) + \sum_f \theta_f \phi_f(S^x(S^n_t, x_t)) \right) \mid S^x_{t-1} \right\}
  \]

» Policy search
  • Find \( \theta \) to optimize
  \[
  \min_\theta \mathbb{E} F(\theta) = \mathbb{E} \sum_{t=0}^{T} \gamma^t C(S_t, X^\pi(S_t \mid \theta))
  \]
  where
  \[
  X^\pi(S_t \mid \theta) = \arg\min_x \left( C(S_t, x_t) + \sum_f \theta_f \phi_f(S^x(S_t, x_t)) \right)
  \]

© 2019 Warren B. Powell
Least squares API vs. policy search

Algorithm performance as a percent of optimal

Myopic policy
LSAPI
LSAPI-Inst.Var.

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

© 2019 Warren B. Powell
Least squares API vs. policy search

Algorithm performance as a percent of optimal

Policy search.

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

© 2019 Warren B. Powell
Least squares API vs. policy search

Notes:

» Using various parametric approximations for the value function did not work well (but it did work well with the trucking application!)

» Using the same functions, but fitted using policy search, worked quite well.

» We do not know how general this conclusion is. For example, when we can exploit problem structure, value function approximations, fitted using Bellman’s equation, can work quite well.

» It appears that the value function approximations need to be quite accurate, and not just locally.
Least squares API vs. policy search

Notes

» So why don’t we always do policy search?
  • It simply doesn’t work all the time.
  • Unable to handle time-dependent policies.
  • The vector $\theta$ needs to be fairly low dimensional if we do not have access to derivatives (and derivatives where the CFA term is in the objective can be tricky).

» One strategy is to use both:
  • Start by trying to fit $\theta$ to approximate the value function. Use this to get an initial value of $\theta$.
  • Then use policy search to tune it to something even better.
Approximate dynamic programming

Stochastic dual dynamic programming
ADP with Benders cuts (SDDP)

ADP with Benders decomposition

» Mario Pereira (1991) was the first to notice that we could approximate stochastic linear programs (which are convex) using Benders cuts.

» SDDP requires two approximations:
  • We are limited to solving a sampled model. That is, we are limited to a small set of sample paths.
  • We have to assume that new information arriving at time t+1 is independent of the information at time t.

» SDDP is approximate dynamic programming using a two-pass procedure:
  • We simulate forward using the current value function approximation (represented using a set of cuts).
  • We then perform a backward pass to create a new cut for each time period, averaging over the set of samples.
ADP with Benders cuts (SDDP)

ADP with Piecewise Linear Separable VFA

- Assumes separability across resource dimensions

\[ \bar{V}_t(R) := \sum_{m=1}^{\vert R \vert} \bar{V}_{t,m}(R_m) \]

- Lack of optimality bounds.

Stochastic Dual Dynamic Programming - SAA

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

\[ \bar{V}_t^k(R) := \max_{j \leq k} \{ \alpha_t^j + \langle \beta_t^j, R \rangle \} \]

- Suitable for problems with small \( \vert \Omega_t \vert \), \( t = 1, \ldots, T \).
ADP with Benders cuts (SDDP)

- The forward pass

### Forward Pass at Iteration $k$

1. Sample $\omega \in \Omega$.
2. **for** $t = 0, \ldots, T$ **do**
3. **if** ($k = 0$) **then**
4. 
   
   Select $x_t^k \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \{ C(S_t(\omega), x_t) \}$
5. **else**
6. 
   Select $x_t^k \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \left\{ C(S_t(\omega), x_t) + V_t^{k-1}(R_t^x) \right\}$
7. **end if**
8. Set $R_t^{x,k} \leftarrow B_t^k x_t^k$; $S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), l_{t+1}(\omega))$
ADP with Benders cuts (SDDP)

- The set of sample paths (of wind)
ADP with Benders cuts (SDDP)

Backward Pass at Iteration $k$

1: \textbf{for} $t = T, \ldots, 1$ \textbf{do}
2: 

Define $V^k_t(R^x_{t-1}, \omega_t) := \min_{x_t \in \mathcal{X}_t(R^x_{t-1}, I_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + V^k_t(R^x_t) \right\}$
ADP with Benders cuts (SDDP)

Backward Pass at Iteration $k$

1: \textbf{for} $t = T, \ldots, 1$ \textbf{do}

2: Define $V^k_t(R^x_{t-1}, \omega_t) := \min_{x_t \in \mathcal{X}_t(R^x_{t-1}, l_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + \overline{V}^k_t(R^x_t) \right\}$

3: \textbf{for all} $\omega_t \in \Omega_t$ \textbf{do}

4: Select $\beta^k_t(\omega_t) \in \partial_{R^x_t} V^k_t(R^x_{t-1}, \omega_t)$

5: \textbf{end for}

6: $\alpha^k_{t-1} \leftarrow \sum_{\omega_t \in \Omega_t} P(\omega_t) V^k_t(R^x_{t-1}, \omega_t)$

7: $h_{t-1}^k(R^x_{t-1}) := \alpha^k_{t-1} + \langle \beta^k_{t-1}, R^x_{t-1} - R^x_{t-1} \rangle$

8: $\overline{V}^k_{t-1}(R^x_{t-1}) := \max \{ \overline{V}^{k-1}_{t-1}(R^x_{t-1}), h_{t-1}^k(R^x_{t-1}) \}$

9: \textbf{end for}

10: $V^k_0 \leftarrow \left\{ \min_{x_0 \in \mathcal{X}_0(S_0)} C(S_0, x_0) + \overline{V}^k_0(R^x_0) \right\}$

© 2019 Warren B. Powell
ADP with Benders cuts (SDDP)

Backward Pass at Iteration $k$

1: \textbf{for} $t = T, \ldots, 1 \textbf{ do}$

2: \[
\text{Define } V^k_t(R^x_{t-1}, \omega_t) := \min_{x_t \in X_t(R^x_{t-1}, l_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + V^k_t(R^x_t) \right\}
\]

3: \textbf{for all } $\omega_t \in \Omega_t \textbf{ do}$

4: Select $\beta^k_t(\omega_t) \in \partial R^x_{t-1} V^k_t(R^x_{t-1}, \omega_t)$

5: \textbf{end for}

6: $\alpha^k_{t-1} \leftarrow \sum_{\omega_t \in \Omega_t} P(\omega_t) V^k_t(R^x_{t-1}, \omega_t)$; $\beta^k_{t-1} \leftarrow \sum_{\omega_t \in \Omega_t} P(\omega_t) \beta^k_t(\omega_t)$

7: $h^k_{t-1}(R^x_{t-1}) := \alpha^k_{t-1} + \langle \beta^k_{t-1}, R^x_{t-1} - R^x_{t-1, k} \rangle$

8: $V^k_{t-1}(R^x_{t-1}) := \max \{ V^k_{t-1}(R^x_{t-1}), h^k_{t-1}(R^x_{t-1}) \}$

9: \textbf{end for}

10: $V^k_0 \leftarrow \left\{ \min_{x_0 \in X^0(S_0)} C(S_0, x_0) + V^k_0(R^x_0) \right\}$

© 2019 Warren B. Powell
Regularization

» In the 1990’s, Ruszczynski showed that Benders works much better if you introduce a regularization term, which penalizes deviations from the current solution.

» This powerful idea (now widely used in statistics) has not been generalized in a practical way to multiperiod (“multistage”) problems.

» In recent work, Tsvetan Asamov introduced a practical way to do regularization for multiperiod problems (with hundreds of time periods). A key feature is that the regularization term does not depend on history.

» The method is very easy to implement.
ADP with Benders cuts (SDDP)

Forward Pass with Regularization

1: Sample $\omega \in \Omega$.
2: for $t = 0, \ldots, T$ do
3:     if $(k = 0)$ then
4:         Select $x_t^k \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \{C(S_t(\omega), x_t)\}$
5:     else
6:         if $t < T$ then
7:             Select $x_t^k \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \left\{C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x) + \frac{\rho^k}{2} \langle R_t^x - \overline{R}_t^{x,k-1}, Q_t(R_t^x - \overline{R}_t^{x,k-1}) \rangle \right\}$
8:         else
9:             Select $x_t^k \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \{C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x)\}$
10:         end if
11:     end if
12:     Set $R_t^{x,k} \leftarrow B_t^k x_t^k; S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), l_{t+1}(\omega))$
13: end for

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

We simply retain the previous resource state.
ADP with Benders cuts (SDDP)

- SDDP with regularization

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

© 2019 Warren B. Powell
ADP with Benders cuts (SDDP)

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

Slightly faster convergence using SPWL VFAs.

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

© 2019 Warren B. Powell
ADP with Benders cuts (SDDP)

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

Much faster convergence using SPWL VFAs.

Figure 8: Comparison of multistage stochastic optimization methods for $|R| = 50$
ADP with Benders cuts (SDDP)

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

SPWL VFAs still exhibit slightly faster convergence.

Figure 11: Numerical comparison of multistage stochastic optimization methods for $|R^T| = 25$
ADP with Benders cuts (SDDP)

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

SPWL VFAs still exhibit slightly faster convergence.

Figure 12: Numerical comparison of multistage stochastic optimization methods for $|R^x| = 50$
ADP with Benders cuts (SDDP)

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

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

Figure 13: Numerical comparison of multistage stochastic optimization methods for $|R_t| = 100$
# Problem classes

<table>
<thead>
<tr>
<th></th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Terminal reward</td>
<td>max_π \mathbb{E}{F(x^π, N, W)</td>
<td>S_0}}</td>
</tr>
<tr>
<td>Stochastic search</td>
<td>(1)</td>
<td>Multiarmed bandit problem</td>
</tr>
<tr>
<td>Offline dynamic programming</td>
<td>(4)</td>
<td>Online dynamic programming</td>
</tr>
<tr>
<td>Offline dynamic programming</td>
<td>max_π \mathbb{E}{C(S, X^{π \text{impl}}(S</td>
<td>θ^{\text{impl}}), W)</td>
</tr>
</tbody>
</table>

**Learning policies (“algorithms”):**
- Approximate dynamic programming
- Q-learning
- SDDP (see next slide)
Forward Pass with Regularization

1: Sample $\omega \in \Omega$.
2: for $t = 0, \ldots, T$ do
3:   if $(k = 0)$ then
4:       Select $x_t^k \in \arg \min_{x_t \in X_t(R_{t-1}^{x,k}, l_t(\omega))} \{C(S_t(\omega), x_t)\}$
5:   else
6:     if $t < T$ then
7:       Select $x_t^k \in \arg \min_{x_t \in X_t(R_{t-1}^{x,k}, l_t(\omega))} \left\{ C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x) + \frac{\rho_t}{2} \left\langle R_t^x - \overline{R}_t^{x,k-1}, Q_t(R_t^x - \overline{R}_t^{x,k-1}) \right\rangle \right\}$
8:     else
9:       Select $x_t^k \in \arg \min_{x_t \in X_t(R_{t-1}^{x,k}, l_t(\omega))} \{C(S_t(\omega), x_t) + \overline{V}_t^{k-1}(R_t^x)\}$
10:   end if
11: end if
12: Set $R_{t+1}^{x,k} \leftarrow B_t^k x_t^k; S_{t+1}(\omega) \leftarrow (R_t^{x,k} - b_{t+1}(\omega), l_{t+1}(\omega))$
13: end for

© 2019 Warren B. Powell
ADP with Benders cuts (SDDP)

Notes:

» SDDP is a form of learning policy. This is a method for learning the cuts of the value function.

» It is parameterized by the regularization term

\[
\rho^k = \rho^0 r^k
\]

\[
\{ C(S_t(\omega), x_t) + \bar{V}_{\bar{t}}^{k-1}(R^x_t) + \frac{\sigma^k}{2} \langle R^x_t - \bar{R}^{x,k-1}_t, Q_t(R^x_t - \bar{R}^{x,k-1}_t) \rangle \}
\]

» So our learning policy is determined by:
  • The SDDP structure.
  • The parameters of the regularization term \( \theta = (\rho^0, r) \)
  • The graphs above represent optimization of the learning policy.

» Then we have to test the cuts as an implementation policy, and compare to other VFA approximations.
ADP with Benders cuts (SDDP)

- The forward pass

**Forward Pass at Iteration $k$**

1: Sample $\omega \in \Omega$.
2: for $t = 0, \ldots, T$ do
3: if ($k = 0$) then
4: Select $x^k_t \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \{ C(S_t(\omega), x_t) \}$
5: else The value function approximation defines the implementation policy!
6: Select $x^k_t \in \arg \min_{x_t \in \mathcal{X}_t(R_{t-1}^{x,k}, l_t(\omega))} \left\{ C(S_t(\omega), x_t) + V_{t-1}^{k-1}(R^x_t) \right\}$
7: end if
8: Set $R_{t}^{x,k} \leftarrow B_t x^k_t$; $S_{t+1}(\omega) \leftarrow (R_{t}^{x,k} - b_{t+1}(\omega), l_{t+1}(\omega))$
9: end for
ADP with Benders cuts (SDDP)

More complex information processes

» We can easily approximate convex functions (using Benders or separable, piecewise linear value functions).

» But what if we have an external information process that affects the system?
  • Weather
  • Economy

» In the next slide, we show how the algorithm changes when we no longer enjoy “intertemporal independence.”

» Assume that the information state is “weather” that may be sunny, cloudy or stormy.
ADP with Benders cuts (SDDP)

1: for $t = T, \ldots, 1$ do
2: 
   Define $V_t^k(R_{t-1}^x, \omega_t) := \min_{x_t \in \mathcal{X}_t(R_{t-1}^x, l_t(\omega_t))} \left\{ C(S_t(\omega_t), x_t) + \overline{V}_t^k(R_t^x, l_t^x(\omega_t)) \right\}$
3:   for all $\omega_t \in \Omega_t$ do
4:     Select $\beta_t^k(\omega_t) \in \partial_{R_t^x} V_t^k(R_{t-1}^x, \omega_t)$
5:   end for
6:   for all $l_{t-1}^x \in \mathcal{I}_{t-1}(\Omega_{t-1})$ do
7:     $\alpha_{t-1}^k(l_{t-1}^x) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | l_{t-1}^x) V_t^k(R_{t-1}^x, \omega_t)$;
8:     $\beta_{t-1}^k(l_{t-1}^x) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | l_{t-1}^x) \beta_t^k(\omega_t)$
9:     $h_{t-1}^k(R_{t-1}^x, l_{t-1}^x) := \alpha_{t-1}^k(l_{t-1}^x) + \langle \beta_{t-1}^k(l_{t-1}^x), R_{t-1}^x - R_{t-1}^{x,k} \rangle$
10:    $\overline{V}_{t-1}^k(R_{t-1}^x, l_{t-1}^x) := \max \left\{ \overline{V}_{t-1}^{k-1}(R_{t-1}^x, l_{t-1}^x), h_{t-1}^k(R_{t-1}^x, l_{t-1}^x) \right\}$
11:   end for
12: $\overline{R}_{t-1}^x, k \leftarrow R_{t-1}^x, k, \ t = 0, \ldots, T - 1$; $\overline{V}_0^k \leftarrow \left\{ \min_{x_0 \in \mathcal{X}_0(S_0)} C(S_0, x_0) + \overline{V}_0^k(R_0^x) \right\}$

Weather states
ADP with Benders cuts (SDDP)

More complex information processes (cont’d)

» In this algorithm, we are using a lookup table function for weather:

\[
\alpha^k_{t-1}(l^x_{t-1}) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | l^x_{t-1}) V^k_t(R^x_{t,k}, \omega_t);
\]

\[
\beta^k_{t-1}(l^x_{t-1}) \leftarrow \sum_{\omega_t \in \Omega_t} \mathbb{P}(\omega_t | l^x_{t-1}) \beta^k_t(\omega_t)
\]

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