Stochastic Dynamic Programmin

V. Leclère

December 8th 2023

École des Ponts
ParisTech

Presentation Outline

Presentation Outline

Presentation Outline

1. Stochastic Dynamic Programming
   - Stochastic optimal control problem
   - Dynamic Programming principle
   - Example
2. Extending the usage of dynamic programming
   - More flexibility in the framework
   - Continuous state space
3. Structured problems
   - Linear Quadratic case
   - Linear convex case

Stochastic Controlled Dynamic System

A discrete time controlled stochastic dynamic system is defined by its *dynamic*

$$\boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1})$$

and initial state

$$\boldsymbol{x}_0 = \boldsymbol{\xi}_0$$

The variables

- $\boldsymbol{x}_t$ is the *state* of the system,
- $\boldsymbol{u}_t$ is the *control* applied to the system at time t,
- $\boldsymbol{\xi}_t$ is an exogeneous noise.

Usually, $\boldsymbol{x}_t \in \mathbb{X}_t$ and $\boldsymbol{u}_t$ belongs to a set depending upon the state: $\boldsymbol{u}_t \in U_t(\boldsymbol{x}_t)$.

Examples

- Stock of water in a dam:
  - $\boldsymbol{x}_t$ is the amount of water in the dam at time t,
  - $\boldsymbol{u}_t$ is the amount of water turbined at time t,
  - $\boldsymbol{\xi}_{t+1}$ is the inflow of water in $[t, t+1[$.
- Boat in the ocean:
  - $\boldsymbol{x}_t$ is the position of the boat at time t,
  - $\boldsymbol{u}_t$ is the direction and speed chosen for $[t, t+1[$,
  - $\boldsymbol{\xi}_{t+1}$ is the wind and current for $[t, t+1[$.
- Subway network:
  - $\boldsymbol{x}_t$ is the position and speed of each train at time t,
  - $\boldsymbol{u}_t$ is the acceleration chosen at time t,
  - $\boldsymbol{\xi}_{t+1}$ is the delay due to passengers and incident on the network for $[t, t+1[$.

More considerations about the state

- Physical state: the physical value of the controlled system. e.g. amount of water in your dam, position of your boat...
- Information state: physical state and information you have over noises. e.g.: amount of water and weather forecast...
- Knowledge state: your current belief over the actual information state (in case of noisy observations). Represented as a distribution law over information states.

The state, in the Dynamic Programming sense, is the information required to define an optimal solution.

Optimization Problem

$$
\begin{aligned}
\min_{\boldsymbol{u}} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = \boldsymbol{\xi}_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{\xi}_0, \cdots, \boldsymbol{\xi}_t)
\end{aligned}
$$

Optimization Problem

$$
\begin{aligned}
\min_{\boldsymbol{u}} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = \boldsymbol{\xi}_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{\xi}_0, \cdots, \boldsymbol{\xi}_t)
\end{aligned}
$$

1. We want to minimize the expectation of the sum of costs.
2. The system follows a dynamic given by the function f_t.
3. There are stagewise constraints on the controls and costs.
4. The controls are functions of the past noises (= non-anticipativity).

Optimization Problem

$$
\begin{aligned}
\min_{\Phi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = \boldsymbol{\xi}_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \Phi(\boldsymbol{\xi}_0, \cdots, \boldsymbol{\xi}_t)
\end{aligned}
$$

1. We want to minimize the expectation of the sum of costs.
2. The system follows a dynamic given by the function f_t.
3. There are stagewise constraints on the controls and costs.
4. The controls are functions of the past noises
(= non-anticipativity).

Optimization Problem with independence of noises

Assuming stagewise independence of the noises, we can compress information in the following way:

$$
\begin{aligned}
\min_{\Phi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t\big(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}\big) + K\big(\boldsymbol{x}_T\big)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = \boldsymbol{\xi}_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \Phi_t(\boldsymbol{\xi}_0, \cdots, \boldsymbol{\xi}_t)
\end{aligned}
$$

Optimization Problem with independence of noises

Assuming stagewise independence of the noises, we can compress information in the following way:

$$
\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = \boldsymbol{\xi}_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t)
\end{aligned}
$$

Keeping only the state

For notational ease, we want to formulate Problem (**??**) only with states.
Let $\mathcal{X}_t(x_t, \xi_{t+1})$ be the reachable states, i.e.,

$$\mathcal{X}_t(x_t, \xi_{t+1}) := \Big\{ x_{t+1} \in \mathbb{X}_{t+1} \quad \Big| \quad \exists u_t \in \mathcal{U}_t(x_t, \xi_{t+1}), \quad x_{t+1} = f_t(x_t, u_t, \xi_{t+1}) \Big\}.$$

And $c_t(x_t, x_{t+1}, \xi_{t+1})$ the transition cost from x_t to x_{t+1}, i.e.,

$$c_t(x_t, x_{t+1}, \xi_{t+1}) := \min_{u_t \in U_t(x_t, \xi_{t+1})} \Big\{ L_t(x_t, u_t, \xi_{t+1}) \quad \Big| \quad x_{t+1} = f_t(x_t, u_t, \xi_{t+1}) \Big\}.$$

Then, under independance of noises, the optimization problem reads

$$\begin{aligned} \min_{\psi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{\xi}_{t+1}) + K(x_T) \Big] \\ s.t. \quad & \boldsymbol{x}_{t+1} \in \mathcal{X}_t(\boldsymbol{x}_t, \boldsymbol{\xi}_{t+1}), \qquad x_0 = \xi_0 \\ & \boldsymbol{x}_{t+1} = \psi_t(\boldsymbol{x}_t, \boldsymbol{\xi}_{t+1}) \end{aligned}$$

Keeping only the state

For notational ease, we want to formulate Problem (**??**) only with states.
Let $\mathcal{X}_t(x_t, \xi_{t+1})$ be the reachable states, i.e.,

$$\mathcal{X}_t(x_t, \xi_{t+1}) := \Big\{ x_{t+1} \in \mathbb{X}_{t+1} \quad | \quad \exists u_t \in \mathcal{U}_t(x_t, \xi_{t+1}), \quad x_{t+1} = f_t(x_t, u_t, \xi_{t+1}) \Big\}.$$

And $c_t(x_t, x_{t+1}, \xi_{t+1})$ the transition cost from x_t to x_{t+1}, i.e.,

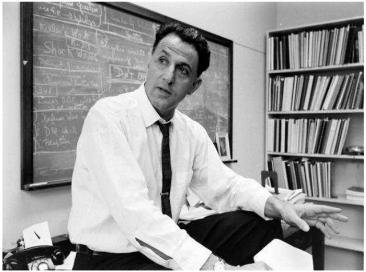
$$c_t(x_t, x_{t+1}, \xi_{t+1}) := \min_{u_t \in U_t(x_t, \xi_{t+1})} \Big\{ L_t(x_t, u_t, \xi_{t+1}) \quad | \quad x_{t+1} = f_t(x_t, u_t, \xi_{t+1}) \Big\}.$$

Then, under independance of noises, the optimization problem reads

$$\begin{aligned} \min_{\psi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} c_t(\boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{\xi}_{t+1}) + K(x_T) \Big] \\ s.t. \quad & \boldsymbol{x}_{t+1} \in \mathcal{X}_t(\boldsymbol{x}_t, \boldsymbol{\xi}_{t+1}), \qquad x_0 = \boldsymbol{\xi}_0 \\ & \boldsymbol{x}_{t+1} = \psi_t(\boldsymbol{x}_t, \boldsymbol{\xi}_{t+1}) \end{aligned}$$

Presentation Outline

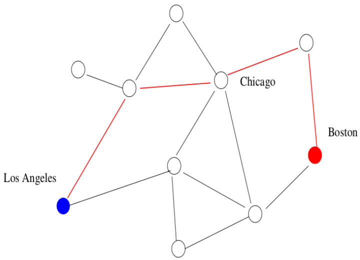
Bellman's Principle of Optimality



Richard Ernest Bellman (August 26, 1920 – March 19, 1984)

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision (Richard Bellman)

The shortest path on a graph illustrates Bellman's Principle of Optimality



For an auto travel analogy, suppose that the fastest route from Los Angeles to Boston passes through ***Chicago****.*
The principle of optimality translates to obvious fact that the Chicago to Boston portion of the route is also the fastest route for a trip that starts from Chicago and ends in Boston. (Dimitri P. Bertsekas)

Idea behind dynamic programming

If noises are time independent, then

1. The cost to go at time t depends only upon the current state.
2. We can compute recursively the cost to go for each position, starting from the terminal state and computing optimal trajectories backward.

Optimal cost-to-go of being in state x at time t is:
At time t, V_{t+1} gives the cost of the future.

Dynamic Programming is a time decomposition method.

Idea behind dynamic programming

If noises are time independent, then

1. The cost to go at time t depends only upon the current state.
2. We can compute recursively the cost to go for each position, starting from the terminal state and computing optimal trajectories backward.

Optimal cost-to-go of being in state x at time t is:
At time t, V_{t+1} gives the cost of the future.

Dynamic Programming is a time decomposition method.

Idea Behind Dynamic Programming

$$
\min_{u_0 \in U_0(x_0)} \mathbb{E}\Bigg[L_0(x_0, u_0, \xi_1) + \min_{u_1, \dots u_{T-1}} \mathbb{E}\Big[\sum_{t=1}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{w}_{t+1}) + K(\boldsymbol{x}_T) \Big] \Bigg]
$$

$$
\begin{aligned}
s.t. \qquad & \boldsymbol{x}_1 = f_0(x_0, u_0, \boldsymbol{\xi}_1) \\
& \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) \in X_{t+1}, \\
& \boldsymbol{u}_t \in U_t(\boldsymbol{x}_t) \\
& \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{\xi}_0, \cdots, \boldsymbol{\xi}_t)
\end{aligned}
$$

Idea Behind Dynamic Programming

$$
\begin{aligned}
\min_{u_0 \in U_0(x_0)} \mathbb{E}\Big[L_0(x_0, u_0, \boldsymbol{\xi}_1) + \min_{u_1,\dots u_{T-1}} & \mathbb{E}\Big[\sum_{t=1}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{w}_{t+1}) + K(\boldsymbol{x}_T)\Big]\Big] \\
s.t. \qquad & \boldsymbol{x}_1 = f_0(x_0, u_0, \boldsymbol{\xi}_1) \\
& \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) \in X_{t+1}, \\
& \boldsymbol{u}_t \in U_t(\boldsymbol{x}_t) \\
& \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{x}_t)
\end{aligned}
$$

Independence of noises

Idea Behind Dynamic Programming

$$\min_{u_0\in U_0(x_0)} \mathbb{E}\Bigg[L_0(x_0,u_0,\boldsymbol{\xi}_1) + \underbrace{\begin{array}{rl} \displaystyle\min_{\boldsymbol{u}_1,\dots \boldsymbol{u}_{T-1}} & \displaystyle\mathbb{E}\Big[\sum_{t=1}^{T-1} L_t(\boldsymbol{x}_t,\boldsymbol{u}_t,\boldsymbol{w}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\ s.t. & \boldsymbol{x}_1 = f_0(x_0,u_0,\boldsymbol{\xi}_1) \\ & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t,\boldsymbol{u}_t,\boldsymbol{\xi}_{t+1}) \in X_{t+1}, \\ & \boldsymbol{u}_t \in U_t(\boldsymbol{x}_t) \\ & \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{x}_t) \end{array}}_{=:V_1(\boldsymbol{x}_1)}\Bigg]$$

Independence of noises

Definition of Bellman Value Function

The Bellman's value function $V_{t_0}(x)$ is defined as the value of the problem starting at time t_0 from the state x.
More precisely we have

$$
\begin{aligned}
V_{t_0}(x) = \min \quad & \mathbb{E}\Big[\sum_{t=t_0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_{t_0} = x \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{\xi}_0, \cdots, \boldsymbol{\xi}_t)
\end{aligned}
$$

Bellman's recursion

The core idea of Bellman's recursion is to see the total (expected) cost as the sum of the current cost and the future cost:

$$
\begin{aligned}
V_t(x_t) = \min_{u_t} \quad & \mathbb{E}\Big[L_t(x, u, \xi_{t+1}) + V_{t+1}(x_{t+1})\Big] \\
& x_{t+1} = f_t(x_t, u_t, \xi_{t+1}) \\
& u_t \in \mathcal{U}_t(x_t) \\
& x_{t+1} \in X_{t+1}
\end{aligned}
$$

And we know the final cost function:

$$
V_T(x_T) = K(x_T).
$$

Dynamic Programming Algorithm - Discrete Case

Data: Problem parameters
Result: optimal strategy and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**

$$V_t(x) = \min_{u \in \mathcal{U}_t(x)} \mathbb{E}\Big[L_t(x, u, \boldsymbol{\xi}_{t+1}) + V_{t+1}\Big(\underbrace{f_t(x, u, \boldsymbol{\xi}_{t+1})}_{x_{t+1}}\Big)\Big]$$

Algorithm 1: Classical stochastic DP algorithm

Dynamic Programming Algorithm - Discrete Case

Data: Problem parameters
Result: optimal strategy and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    $V_t(x) = +\infty$;
    for $u \in \mathcal{U}(x)$ **do**
      $$Q_t(x,u) = \mathbb{E}\Big[L_t(x,u,\boldsymbol{\xi}_{t+1}) + V_{t+1}\Big(\underbrace{f_t(x,u,\xi_{t+1})}_{x_{t+1}}\Big)\Big]$$
      if $Q_t(x,u) < V_t(x)$ **then**
        $V_t(x) = Q_t(x,u)$;
        $\pi_t(x) = u$;

Algorithm 1: Classical stochastic DP algorithm

Dynamic Programming Algorithm - Discrete Case

Data: Problem parameters
Result: optimal strategy and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    $V_t(x) = +\infty$;
    for $u \in \mathcal{U}(x)$ **do**
      for $\xi \in \Xi_{t+1}$ **do**
        $x^\xi_{t+1} = f_t(x, u, \xi)$;
        if $x^\xi_{t+1} \in X_t$ **then**
          $\dot{Q}_t(x, u, \xi) = L_t(x, u, \boldsymbol{\xi}_{t+1}) + V_{t+1}(x^\xi_{t+1})$
        else
          $\dot{Q}_t(x, u, \xi) = +\infty$
      $Q_t(x, u) = \sum_{\xi \in \Xi_{t+1}} \mathbb{P}(\boldsymbol{\xi}_{t+1} = \xi)\dot{Q}_t(x, u, \xi)$;
      if $Q_t(x, u) < V_t(x)$ **then**
        $V_t(x) = Q_t(x, u)$; $\pi_t(x) = u$;

Algorithm 1: Classical stochastic DP algorithm

3 curses of dimensionality

Complexity $= O(T \times |\mathbb{X}_t| \times |\mathbb{U}_t| \times |\Xi_t|)$

Linear in the number of time steps, but we have 3 curses of dimensionality :

1. State. Complexity is exponential in the dimension of $\mathbb{X}_t$ e.g. 3 independent states each taking 10 values leads to a loop over 1000 points.
2. Decision. Complexity is exponential in the dimension of $\mathbb{U}_t$.
   $\rightsquigarrow$ due to exhaustive minimization of inner problem. Can be accelerated using faster method (e.g. MILP solver).
3. Expectation. Complexity is exponential in the dimension of Ξ_t.
   $\rightsquigarrow$ due to expectation computation. Can be accelerated through Monte-Carlo approximation (still at least 1000 points)

In practice, DP is not used for a state of dimension more than 5.

3 curses of dimensionality

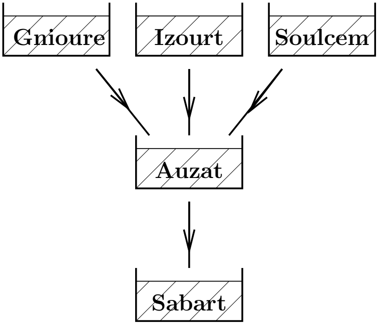
Complexity $= O(T \times |\mathbb{X}_t| \times |\mathbb{U}_t| \times |\Xi_t|)$

Linear in the number of time steps, but we have 3 curses of dimensionality :

1. State. Complexity is exponential in the dimension of $\mathbb{X}_t$
   e.g. 3 independent states each taking 10 values leads to a loop over 1000 points.
2. Decision. Complexity is exponential in the dimension of $\mathbb{U}_t$.
   $\rightsquigarrow$ due to exhaustive minimization of inner problem. Can be accelerated using faster method (e.g. MILP solver).
3. Expectation. Complexity is exponential in the dimension of Ξ_t.
   $\rightsquigarrow$ due to expectation computation. Can be accelerated through Monte-Carlo approximation (still at least 1000 points)

In practice, DP is not used for a state of dimension more than 5.

Illustrating dynamic programming with the damsvalley example



Illustrating the curse of dimensionality

We are in dimension 5 (not so high in the world of big data!) with 52 timesteps (common in energy management) plus 5 controls and 5 independent noises.

1. We discretize each state's dimension in 100 values: $|\mathbb{X}_t| = 100^5 = 10^{10}$
2. We discretize each control's dimension in 100 values: $|U_t| = 100^5 = 10^{10}$
3. We use optimal quantization to discretize the noises' space in 10 values: $|\Xi_t| = 10$

Number of flops: $\mathcal{O}(52 \times 10^{10} \times 10^{10} \times 10) \approx \mathcal{O}(10^{23})$.
In the TOP500, the best computer computes 10^{17} flops/s.
Even with the most powerful computer, it takes at least 12 days to solve this problem.

Presentation Outline

A storage management example

A producer that needs to satisfy a weekly demand over 12 weeks.

- Storage capacity of 100 units, starting with 50 units.
- The producer can produce 0 (cost 0), 10 (cost 20) or 20 (cost 30) or 25 (cost 45) units per week.
- Demand is random and follows a stagewise independent uniform distribution on $\{0, 10, 20, 30, 40\}$.
- Storage cost 0.1 per unit per week.
- Unmet demand is lost and costs 5 per unit.
- Products remaining at the end are sold at 1 per unit.
- During a given week:
  - producer decide how much to produce during the week
  - demand is revealed and should be met with current stock and production
  - remaining stock is stored (at a cost), stock above capacity is lost

Exercise

1. Formulate the problem as a stochastic dynamic program, underlying state, decision and noise.
2. Write the dynamic programming (Bellman's) equation.
3. Solve the problem with your favorite programming language.

Presentation Outline

Presentation Outline

Requirements of stochastic DP

$$\begin{aligned} \min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t\big(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}\big) + K\big(\boldsymbol{x}_T\big)\Big] \\ s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\ & \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\ & \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t) \end{aligned}$$

Assumptions:

- The noise are stagewise-independent.
- The only constraint linking stages is the dynamic equation: no coupling between stages.
- The cost function is additive over stages.
- We consider the expectation of costs.

Dynamic Programming Algorithm - Discrete Case

Data: Problem parameters
Result: optimal strategy and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**

$$V_t(x) = \min_{u \in \mathcal{U}_t(x)} \mathbb{E}\Big[L_t(x, u, \boldsymbol{\xi}_{t+1}) + V_{t+1}\Big(\underbrace{f_t(x, u, \boldsymbol{\xi}_{t+1})}_{x_{t+1}} \Big) \Big]$$

Algorithm 2: Classical stochastic DP algorithm

Dynamic Programming Algorithm - Discrete Case

Data: Problem parameters
Result: optimal strategy and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    $V_t(x) = +\infty$;
    for $u \in \mathcal{U}(x)$ **do**
      $Q_t(x,u) = \mathbb{E}\Big[L_t(x,u,\boldsymbol{\xi}_{t+1}) + V_{t+1}\Big(\underbrace{f_t(x,u,\xi_{t+1})}_{x_{t+1}}\Big)\Big]$
      if $Q_t(x,u) < V_t(x)$ **then**
        $V_t(x) = Q_t(x,u)$;
        $\pi_t(x) = u$;

Algorithm 2: Classical stochastic DP algorithm

Dynamic Programming Algorithm - Discrete Case

Data: Problem parameters
Result: optimal strategy and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    $V_t(x) = +\infty$;
    for $u \in \mathcal{U}(x)$ **do**
      for $\xi \in \Xi_{t+1}$ **do**
        $x_{t+1}^{\xi} = f_t(x, u, \xi)$;
        if $x_{t+1}^{\xi} \in X_t$ **then**
          $\dot{Q}_t(x, u, \xi) = L_t(x, u, \boldsymbol{\xi}_{t+1}) + V_{t+1}(x_{t+1}^{\xi})$
        else
          $\dot{Q}_t(x, u, \xi) = +\infty$
      $Q_t(x, u) = \sum_{\xi \in \Xi_{t+1}} \mathbb{P}(\boldsymbol{\xi}_{t+1} = \xi)\dot{Q}_t(x, u, \xi)$;
      if $Q_t(x, u) < V_t(x)$ **then**
        $V_t(x) = Q_t(x, u)$; $\pi_t(x) = u$;

Algorithm 2: Classical stochastic DP algorithm

Markovian noise

Assume that $(\xi_t)_t$ is a Markovian noise, i.e. ξ_t only depends on x_t.

- We can recover the previous setting by defining an extended state

$$\tilde{x}_t = (x_t, \xi_t)$$

- Bellman equation then becomes:

$$V_t(x_t, \xi_t) := \min_{u_t \in \mathcal{U}_t(x_t)} \mathbb{E}\Big[L_t(x_t, u_t, \xi_{t+1}) + V_{t+1}(x_{t+1}) \mid \xi_t = \xi_t\Big]$$

More precisely, it means that:

1. The value function V_t (and the optimal policy π_t) depends on both the current physical state x_t and the current noise ξ_t.
2. The probability used to average the cost to go in the algorithm is the conditional probability given ξ_t.

Markovian noise

Assume that $(\boldsymbol{\xi}_t)_t$ is a Markovian noise, i.e. $\boldsymbol{\xi}_t$ only depends on $\boldsymbol{x}_t$.

- We can recover the previous setting by defining an extended state

$$\tilde{x}_t = (\boldsymbol{x}_t, \boldsymbol{\xi}_t)$$

- Bellman equation then becomes:

$$V_t(x_t, \xi_t) := \min_{u_t \in \mathcal{U}_t(x_t)} \mathbb{E}\Big[L_t(x_t, u_t, \boldsymbol{\xi}_{t+1}) + V_{t+1}(\boldsymbol{x}_{t+1}) \mid \boldsymbol{\xi}_t = \xi_t \Big]$$

More precisely, it means that:

1. The value function V_t (and the optimal policy π_t) depends on both the current physical state x_t and the current noise ξ_t.
2. The probability used to average the cost to go in the algorithm is the conditional probability given ξ_t.

Markovian noise

Assume that $(\boldsymbol{\xi}_t)_t$ is a Markovian noise, i.e. $\boldsymbol{\xi}_t$ only depends on $\boldsymbol{x}_t$.

- We can recover the previous setting by defining an extended state

$$\tilde{x}_t = (\boldsymbol{x}_t, \boldsymbol{\xi}_t)$$

- Bellman equation then becomes:

$$V_t(x_t, \xi_t) := \min_{u_t \in \mathcal{U}_t(x_t)} \mathbb{E}\Big[L_t(x_t, u_t, \boldsymbol{\xi}_{t+1}) + V_{t+1}(\boldsymbol{x}_{t+1}) \mid \boldsymbol{\xi}_t = \xi_t\Big]$$

More precisely, it means that:

1. The value function V_t (and the optimal policy π_t) depends on both the current physical state x_t and the current noise ξ_t.
2. The probability used to average the cost to go in the algorithm is the conditional probability given ξ_t.

Coupling control

Consider the following problem, with stagewise independent noise:

$$
\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t) \\
& \|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\| \leq \delta
\end{aligned}
$$

How can we solve this problem using Dynamic Programming?

Coupling control

Consider the following problem, with stagewise independent noise:

$$
\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T) \Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t) \\
& \|\boldsymbol{u}_t - \boldsymbol{u}_{t-1}\| \leq \delta
\end{aligned}
$$

How can we solve this problem using Dynamic Programming?

Delayed control

Consider the following problem, with stagewise independent noise:

$$
\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t\big(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}\big) + K\big(\boldsymbol{x}_T\big)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t\big(\boldsymbol{x}_t, \boldsymbol{u}_{t-2}, \boldsymbol{\xi}_{t+1}\big), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t)
\end{aligned}
$$

How can we solve this problem using Dynamic Programming?

Delayed control

Consider the following problem, with stagewise independent noise:

$$\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_{t-2}, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t)
\end{aligned}$$

How can we solve this problem using Dynamic Programming?

Bankruptcy

Consider the following problem, with stagewise independent noise:

$$
\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t)
\end{aligned}
$$

In addition, we assume that we start with a capital C_0, and that we must never, under any circonstance, have a negative capital.
How can we solve this problem using Dynamic Programming?

Bankruptcy

Consider the following problem, with stagewise independent noise:

$$\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \xi_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \xi_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t)
\end{aligned}$$

In addition, we assume that we start with a capital C_0, and that we must never, under any circonstance, have a negative capital. How can we solve this problem using Dynamic Programming?

Maximizing probability

Consider the following problem, with stagewise independent noise:

$$\begin{aligned}
\min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} L_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}) + K(\boldsymbol{x}_T)\Big] \\
s.t. \quad & \boldsymbol{x}_{t+1} = f_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_{t+1}), \qquad \boldsymbol{x}_0 = x_0 \\
& \boldsymbol{u}_t \in \mathcal{U}_t(\boldsymbol{x}_t), \quad \boldsymbol{x}_t \in X_t \\
& \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t)
\end{aligned}$$

We are now reconsidering our objective function, and want to replace the expectation by the probability of the accumulated, at the end of the period, to be negative.
How can we solve this problem by Dynamic Programming?

Presentation Outline

Dynamic Programming Algorithm - Discrete Case - HD

Data: Problem parameters
Result: optimal trajectory and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    $V_t(x) = \mathbb{E}\Big[\min_{y \in \mathcal{X}_t(x, \boldsymbol{\xi}_{t+1})} \Big(c_t(x, y, \boldsymbol{\xi}_{t+1}) + V_{t+1}(y)\Big)\Big]$

Algorithm 3: Classical stochastic dynamic programming algorithm

Dynamic Programming Algorithm - Discrete Case - HD

Data: Problem parameters
Result: optimal trajectory and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    for $\xi \in \Xi_t$ **do**

$$\hat{V}_t(x,\xi) = \min_{y \in \mathcal{X}_t(x,\xi)} c_t(x,y,\xi) + V_{t+1}(y)$$

$$V_t(x) = V_t(x) + \mathbb{P}(\xi)\hat{V}_t(x,\xi)$$

Algorithm 3: Classical stochastic dynamic programming algorithm

Dynamic Programming Algorithm - Discrete Case - HD

Data: Problem parameters
Result: optimal trajectory and value;
$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $x \in \mathbb{X}_t$ **do**
    for $\xi \in \Xi_t$ **do**
      $\hat{V}_t(x,\xi) = \infty$;
      for $y \in \mathcal{X}_t(x,\xi)$ **do**
        $v_y = c_t(x,y,\xi) + V_{t+1}(y)$;
        if $v_y < \hat{V}_t(x,\xi)$ **then**
          $\hat{V}_t(x,\xi) = v_y$;
          $\psi_t(x,\xi) = y$;
      $V_t(x) = V_t(x) + \mathbb{P}(\xi)\hat{V}_t(x,\xi)$

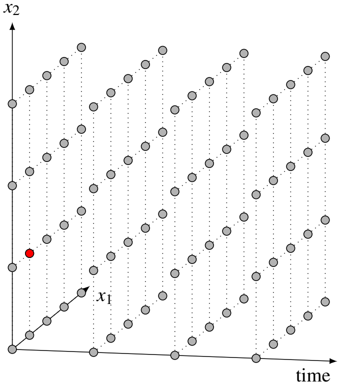
Algorithm 3: Classical stochastic dynamic programming algorithm

Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

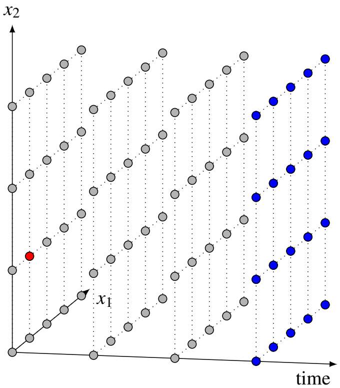


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

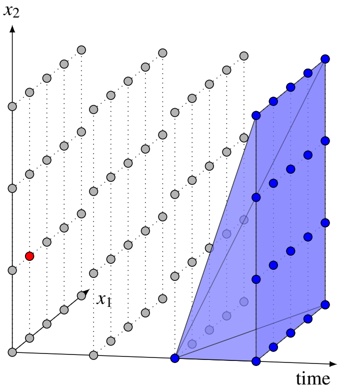
    for $\xi \in \Xi_t$ **do**

      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$

      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

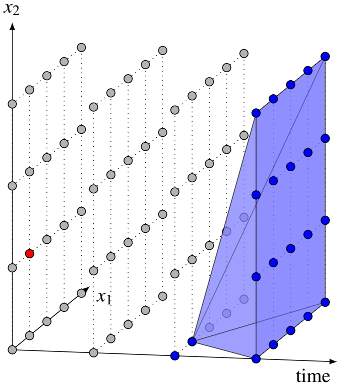
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

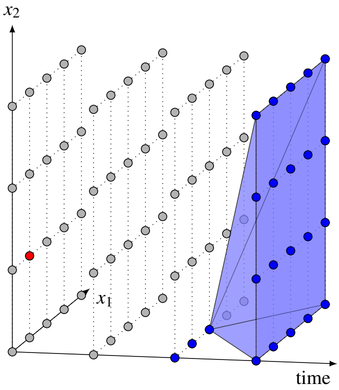
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

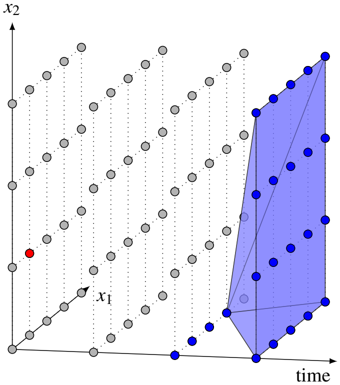
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

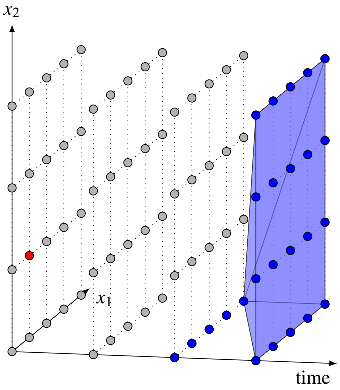
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

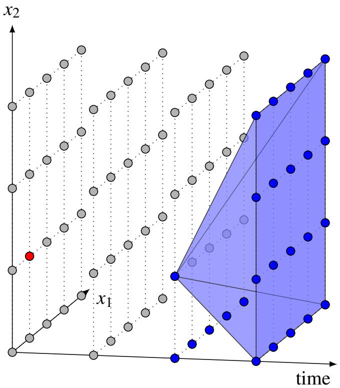


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$
      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

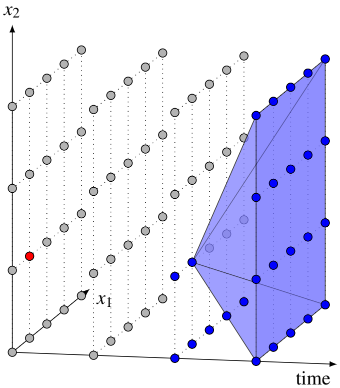
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X_{t-1}^D$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

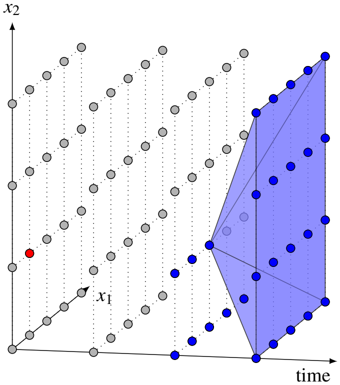


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

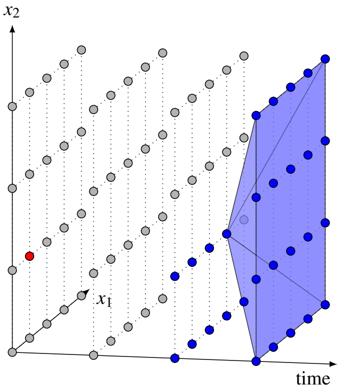


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$
      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

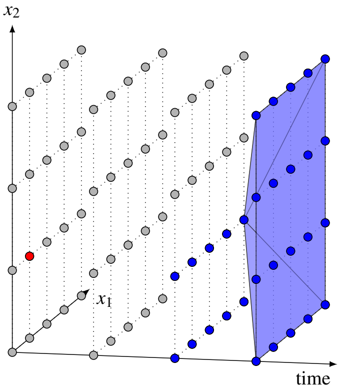
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

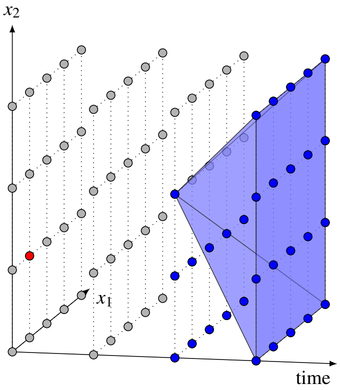
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

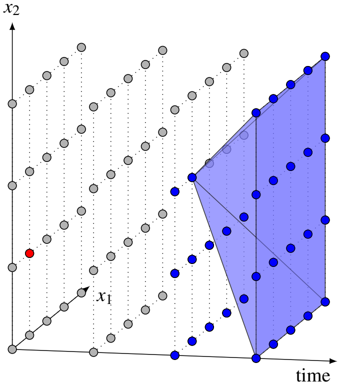


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

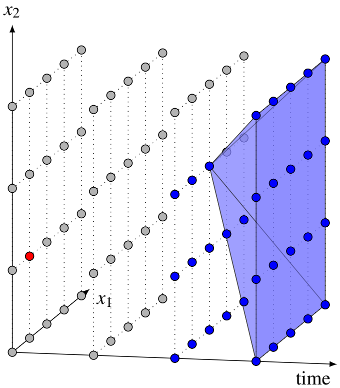


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

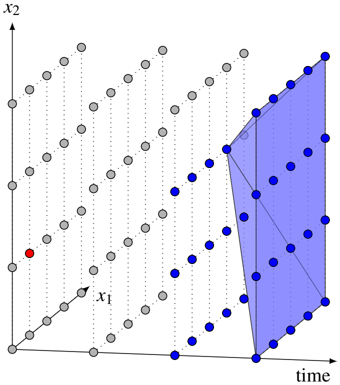
    for $\xi \in \Xi_t$ **do**

      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

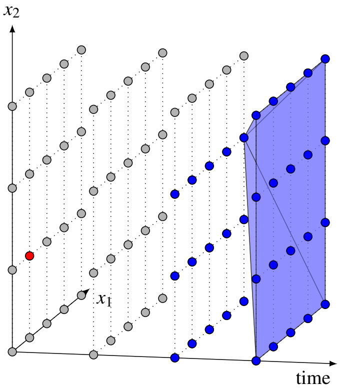
    for $\xi \in \Xi_t$ **do**

      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

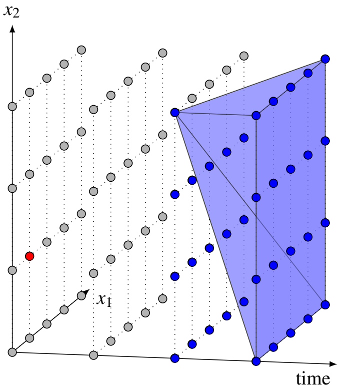
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

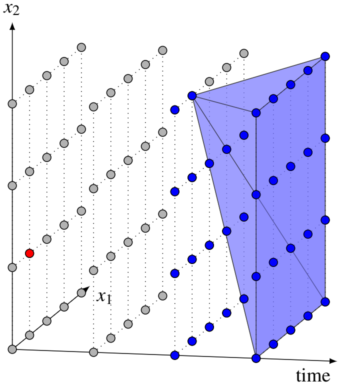
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

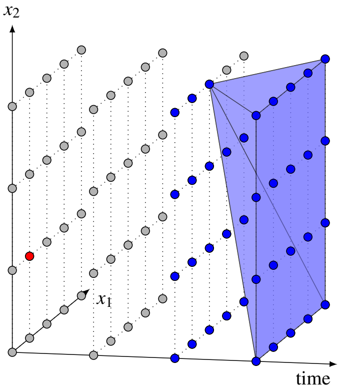
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

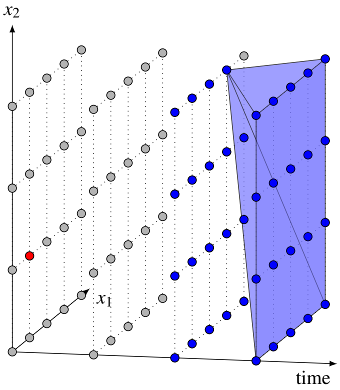
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

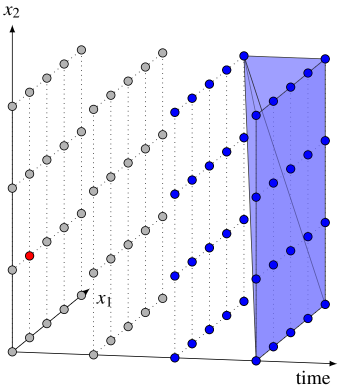


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

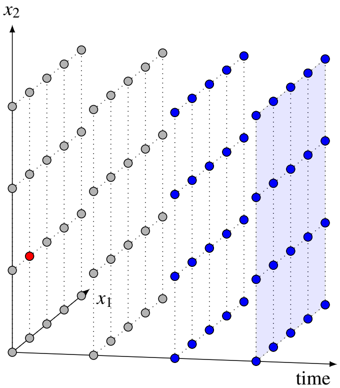
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

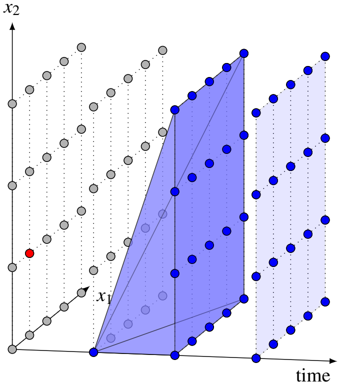
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:= \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t = \xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

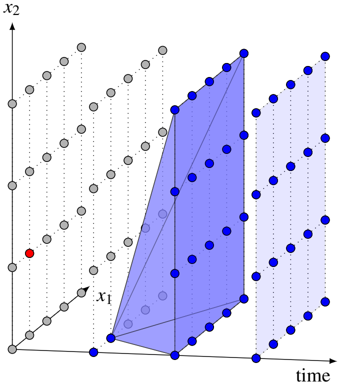
    for $\xi \in \Xi_t$ **do**

      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

      $$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

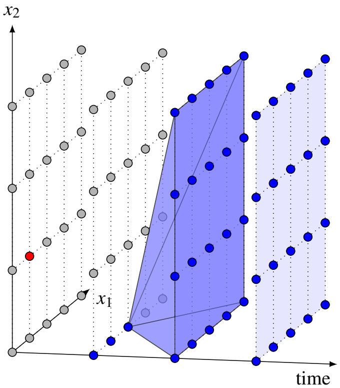


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

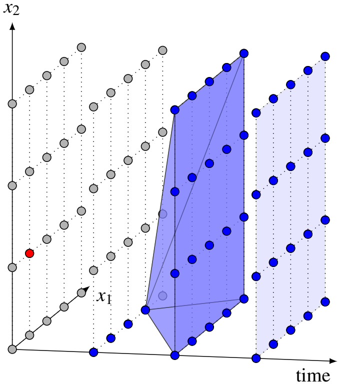


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$
      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

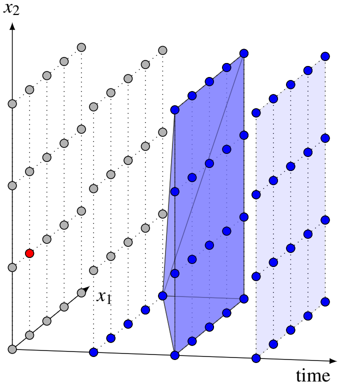
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

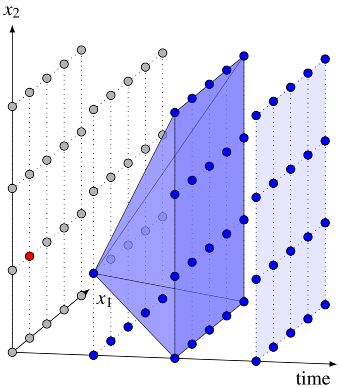


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$
      $\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

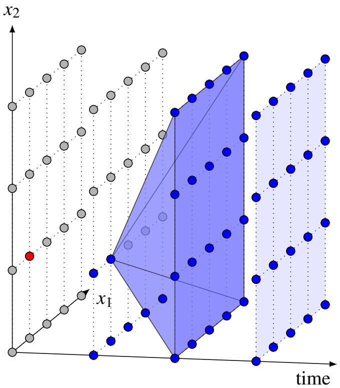
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

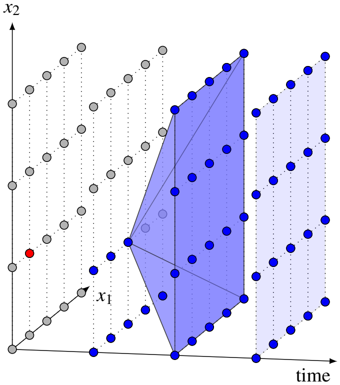


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

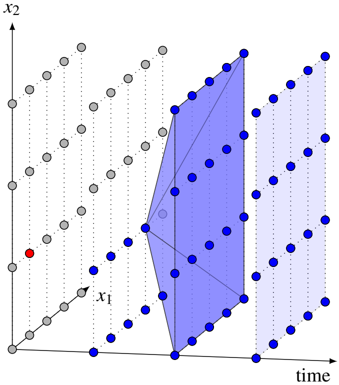
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

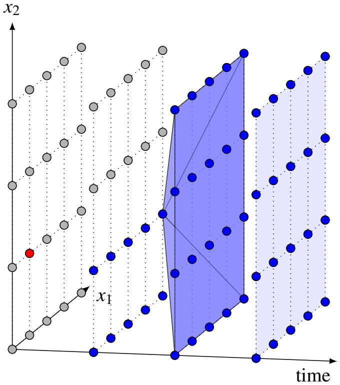
    for $\xi \in \Xi_t$ **do**

      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$

      $\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

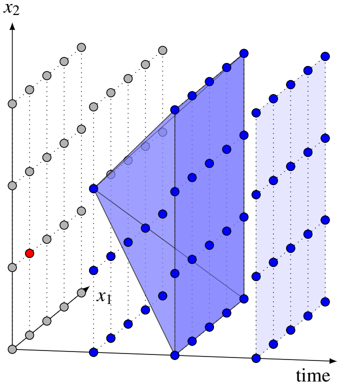
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

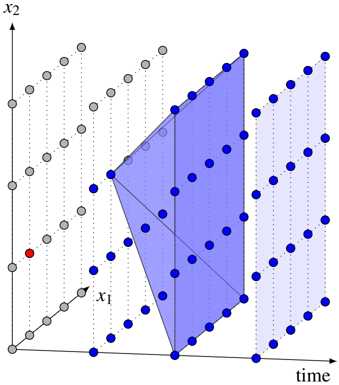
    for $\xi \in \Xi_t$ **do**

      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$

      $\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

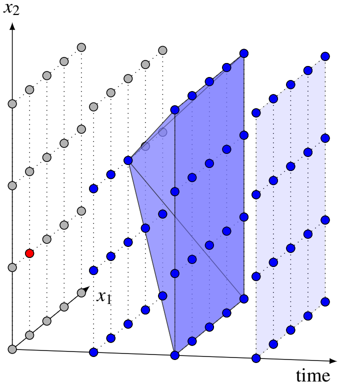
    for $\xi \in \Xi_t$ **do**

      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:= \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$

      $\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t = \xi)} \dot{v}_\xi$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

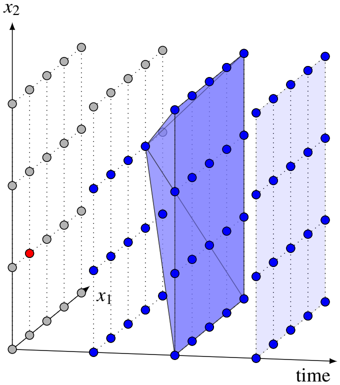


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$
      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

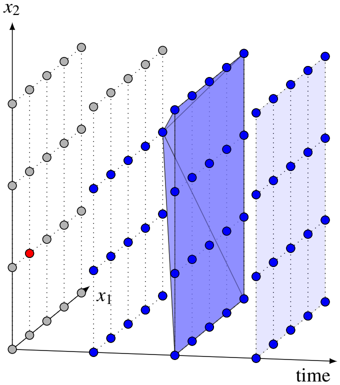
    for $\xi \in \Xi_t$ **do**

      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$

      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

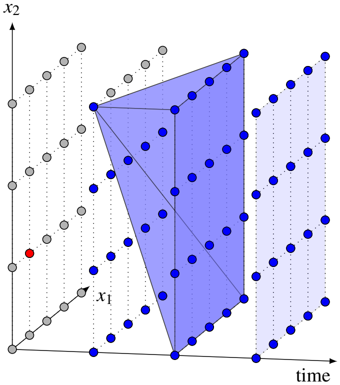
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

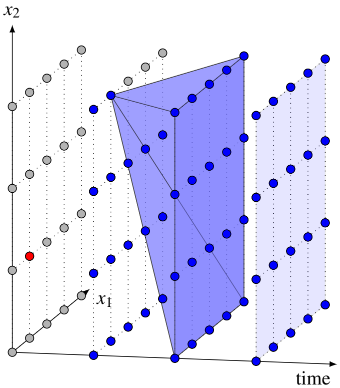
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) += \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

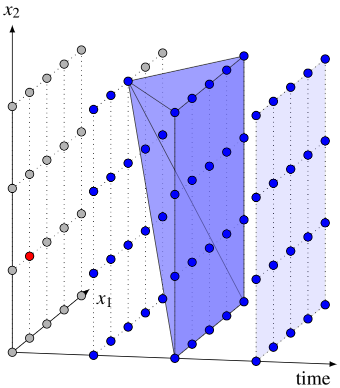


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$
      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

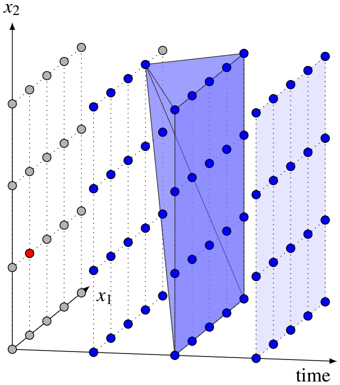
$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:= \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP

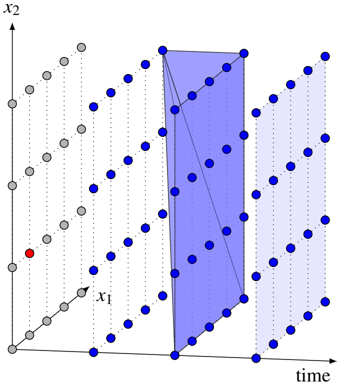


Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$
for $t : T-1 \to 1$ **do**
  for $x_{in} \in X^D_{t-1}$ **do**
    for $\xi \in \Xi_t$ **do**
      $\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$
      $\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$
  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

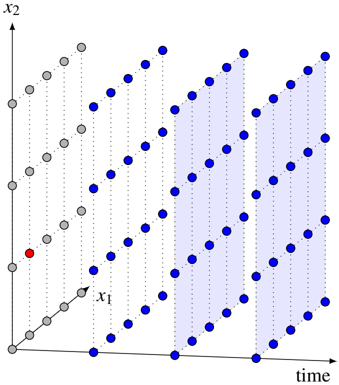
    for $\xi \in \Xi_t$ **do**

      $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

      $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Discretized Stochastic Dynamic Programming

The simplest DP algorithm is obtained by discretizing the state set, and then doing a single backward pass over the grid.

$\tilde{V}_t \equiv 0$

for $t : T-1 \to 1$ **do**

  for $x_{in} \in X^D_{t-1}$ **do**

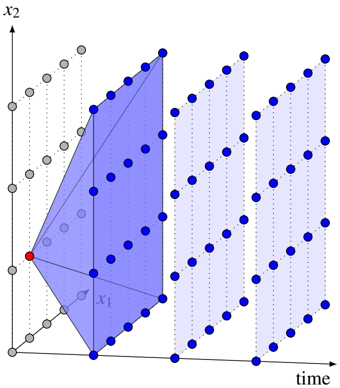
    for $\xi \in \Xi_t$ **do**

$$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

$$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

  Extend definition of $\tilde{V}_t$ to X_t by interpolation

Algorithm 1: Discretized SDP



Cost-to-go induced policy and Forward Bellman operator

- The point of most DP methods is to produce approximations $\tilde{V}_t$ of the true value function[1] V_t.
- From any approximation $\tilde{V}_t$ of V_t, we can define a cost-to-go induced policy ψ_t by solving the stage problem:

$$\min_{x_{out}, u_t \in \mathcal{X}_t(x_{in}, \xi_t)} \underbrace{\ell_{t+1}(x_{in}, x_t, u_t, \xi_t)}_{\text{transition costs}} + \underbrace{\tilde{V}(x_{out})}_{\text{cost-to-go}}$$

- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.

➥ Often used to pass information from long-term to short-term problems.

[1] Sometimes it can be of $\dot{V}_t$ instead

Cost-to-go induced policy and Forward Bellman operator

- The point of most DP methods is to produce approximations $\tilde{V}_t$ of the true value function[1] V_t.
- From any approximation $\tilde{V}_t$ of V_t, we can define a cost-to-go induced policy ψ_t by solving the stage problem:

$$\min_{x_{out}, u_t \in \mathcal{X}_t(x_{in}, \xi_t)} \underbrace{\ell_{t+1}(x_{in}, x_t, u_t, \xi_t)}_{\text{transition costs}} + \underbrace{\tilde{V}(x_{out})}_{\text{cost-to-go}}$$

- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.

➥ Often used to pass information from long-term to short-term problems.

[1]Sometimes it can be of $\dot{V}_t$ instead

Cost-to-go induced policy and Forward Bellman operator

- The point of most DP methods is to produce approximations $\tilde{V}_t$ of the true value function[1] V_t.
- From any approximation $\tilde{V}_t$ of V_t, we can define a cost-to-go induced policy ψ_t by solving the stage problem:

$$\min_{x_{out}, u_t \in \mathcal{X}_t(x_{in}, \xi_t)} \underbrace{\ell_{t+1}(x_{in}, x_t, u_t, \xi_t)}_{\text{transition costs}} + \underbrace{\tilde{V}(x_{out})}_{\text{cost-to-go}}$$

- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.

➥ Often used to pass information from long-term to short-term problems.

[1] Sometimes it can be of $\dot{V}_t$ instead

Cost-to-go induced policy and Forward Bellman operator

- The point of most DP methods is to produce approximations $\tilde{V}_t$ of the true value function[1] V_t.
- From any approximation $\tilde{V}_t$ of V_t, we can define a cost-to-go induced policy ψ_t by solving the stage problem:

$$\min_{x_{out}, u_t \in \mathcal{X}_t(x_{in}, \xi_t)} \underbrace{\ell_{t+1}(x_{in}, x_t, u_t, \xi_t)}_{\text{transition costs}} + \underbrace{\tilde{V}(x_{out})}_{\text{cost-to-go}}$$

- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.

➥ Often used to pass information from long-term to short-term problems.

[1] Sometimes it can be of $\dot{V}_t$ instead

Presentation Outline

1. Stochastic Dynamic Programming
   - Stochastic optimal control problem
   - Dynamic Programming principle
   - Example
2. Extending the usage of dynamic programming
   - More flexibility in the framework
   - Continuous state space
3. Structured problems
   - Linear Quadratic case
   - Linear convex case

Presentation Outline

Linear Quadratic case

$$\begin{aligned} \min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} \boldsymbol{x}_t^\top Q_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top R_t \boldsymbol{u}_t + \boldsymbol{x}_T^\top Q_T \boldsymbol{x}_T\Big] \\ s.t. \quad & \boldsymbol{x}_{t+1} = A_t \boldsymbol{x}_t + B_t \boldsymbol{u}_t + \boldsymbol{\xi}_t, \qquad \boldsymbol{x}_0 = x_0 \\ & \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t) \end{aligned}$$

Under stagewise independence of the (centered) noise we can show that:

1. The value function is quadratic: $V_t(x_t) = x_t^\top K_t x_t + k_t$.
2. The optimal policy is linear: $\pi_t(x_t) = L_t x_t$.
3. With explicit (Riccati) formulas for K_t and L_t.

$$\begin{cases} K_T = Q_T, k_T = 0 \\ K_t = Q_t + A_t^\top K_{t+1} A_t - A_t^\top K_{t+1} B_t (R_t + B_t^\top K_{t+1} B_t)^{-1} B_t^\top K_{t+1} A_t \\ L_t = -(R_t + B_t^\top K_{t+1} B_t)^{-1} B_t^\top K_{t+1} A_t \end{cases}$$

➥ Can be solved for large dimension (say $n \sim 10^4$).

Linear Quadratic case

$$\begin{aligned} \min_{\pi} \quad & \mathbb{E}\Big[\sum_{t=0}^{T-1} \boldsymbol{x}_t^\top Q_t \boldsymbol{x}_t + \boldsymbol{u}_t^\top R_t \boldsymbol{u}_t + \boldsymbol{x}_T^\top Q_T \boldsymbol{x}_T\Big] \\ s.t. \quad & \boldsymbol{x}_{t+1} = A_t \boldsymbol{x}_t + B_t \boldsymbol{u}_t + \boldsymbol{\xi}_t, \qquad \boldsymbol{x}_0 = x_0 \\ & \boldsymbol{u}_t = \pi_t(\boldsymbol{x}_t) \end{aligned}$$

Under stagewise independence of the (centered) noise we can show that:

1. The value function is quadratic: $V_t(x_t) = x_t^\top K_t x_t + k_t$.
2. The optimal policy is linear: $\pi_t(x_t) = L_t x_t$.
3. With explicit (Riccati) formulas for K_t and L_t.

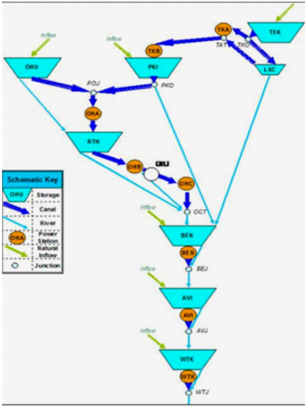
$$\begin{cases} K_T = Q_T, k_T = 0 \\ K_t = Q_t + A_t^\top K_{t+1} A_t - A_t^\top K_{t+1} B_t (R_t + B_t^\top K_{t+1} B_t)^{-1} B_t^\top K_{t+1} A_t \\ L_t = -(R_t + B_t^\top K_{t+1} B_t)^{-1} B_t^\top K_{t+1} A_t \end{cases}$$

➥ Can be solved for large dimension (say $n \sim 10^4$).

Presentation Outline

From Dynamic Programming to SDDP

- DP is a flexible tool, hampered by the curses of dimensionality
- Numerical illustration (7 dams):
  - $T = 52$ weeks
  - $|S| = 100^7$ possible states
  - $|U| = 10^7$ possible controls
  - $|\xi_t| = 10$ (10^{52} scenarios)



➥ ≈ 2 days on today's fastest super-computer (3.10^6 years for 10 dams)

➥ Can be solved[2] in ≈ 10 minutes

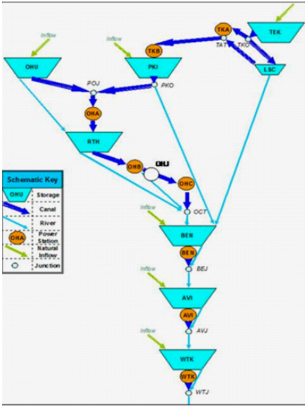
[2] Approximately, depending on the problem and precision required...

From Dynamic Programming to SDDP

- DP is a flexible tool, hampered by the curses of dimensionality
- Numerical illustration (7 dams):
  - $T = 52$ weeks
  - $|S| = 100^7$ possible states
  - $|U| = 10^7$ possible controls
  - $|\xi_t| = 10$ (10^{52} scenarios)

➥ ≈ 2 days on today's fastest super-computer (3.10^6 years for 10 dams)

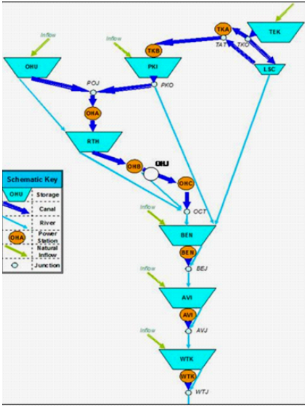
➥ Can be solved[2] in ≈ 10 minutes



[2] Approximately, depending on the problem and precision required...

From Dynamic Programming to SDDP

- DP is a flexible tool, hampered by the curses of dimensionality
- Numerical illustration (7 dams):
  - $T = 52$ weeks
  - $|S| = 100^7$ possible states
  - $|U| = 10^7$ possible controls
  - $|\xi_t| = 10$ (10^{52} scenarios)



➥ ≈ 2 days on today's fastest super-computer (3.10^6 years for 10 dams)

➥ Can be solved[2] in ≈ 10 minutes

[2] Approximately, depending on the problem and precision required...

How can we be so much faster ?

- Structural assumptions:
  - convexity
  - continuous state
  - ➥ duality tools
- Sampling instead of exhaustive computation
- Iteratively refining value function estimation at "the right places" only
- LP solvers

➥ Stochastic Dual Dynamic Programming (SDDP) which
  - has been around for 30 years
  - is widely used in the energy community
  - has lots of extensions and variants
  - some convergence results, mainly asymptotic

Indepen
Finitely suppo
Conve
Discrete
State discr
Progres
Maximur

The setting

1. We are in a finite-time, stagewise independent framework.
2. The state and control variables are continuous and bounded.
3. The costs are convex (jointly in state and control).
4. The dynamic is linear.
5. The constraint on control is convex.
6. We are in a relatively complete recourse framework.

Then, we can show that, the value function are convex, and we can approximate them by polyhedral functions.

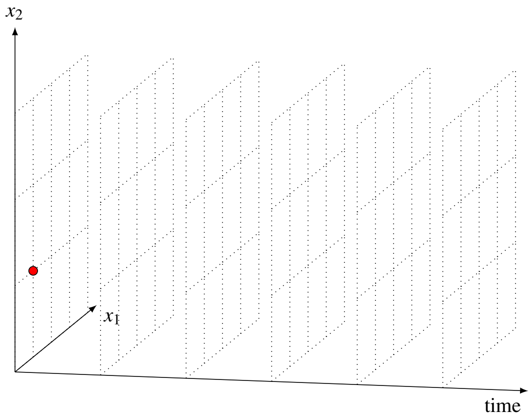
Stochastic Dual Dynamic Programming: principle

The main idea is to update approximations of the value functions by adding cuts, in order to refine the approximations. We iterate the following steps:

Forward pass Given approximations of the value functions, we simulate the policy induced by these approximations, and obtain a trajectory.

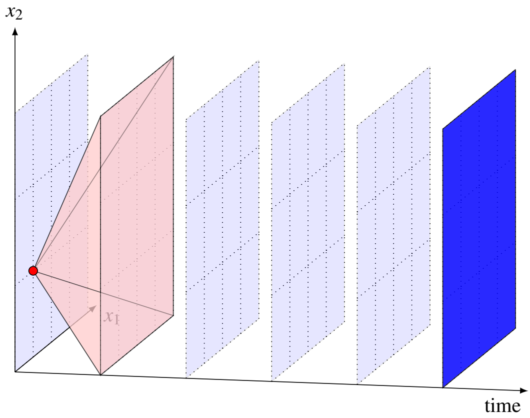
Backward pass We refine the approximations by adding cuts, in order to make the approximations more precise around the trajectory.

Stochastic Dual Dynamic Programming



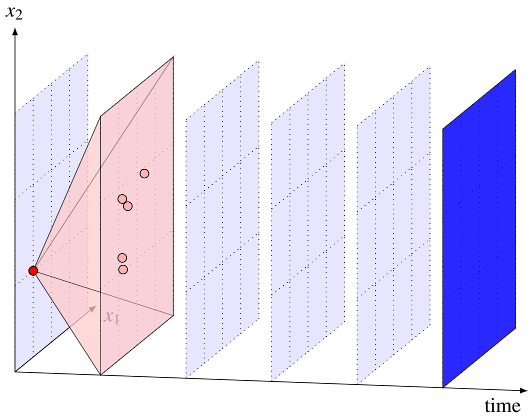
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



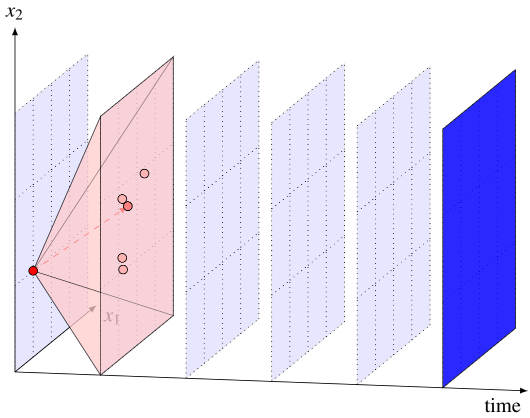
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



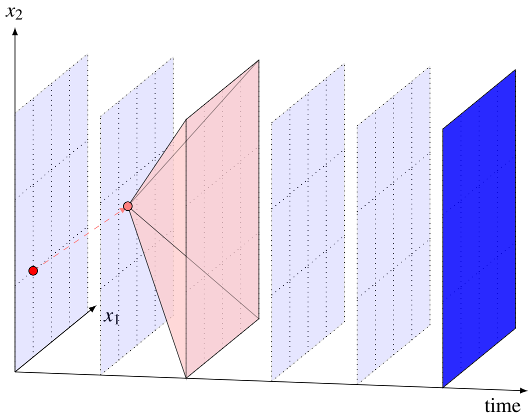
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



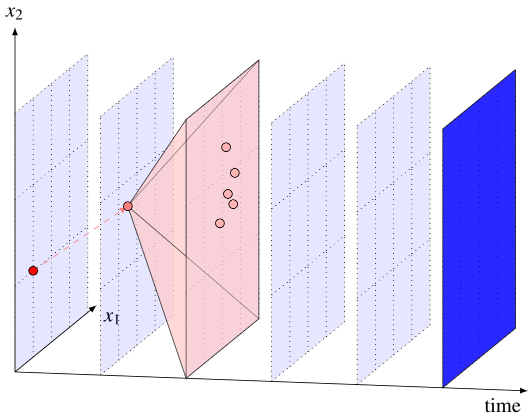
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



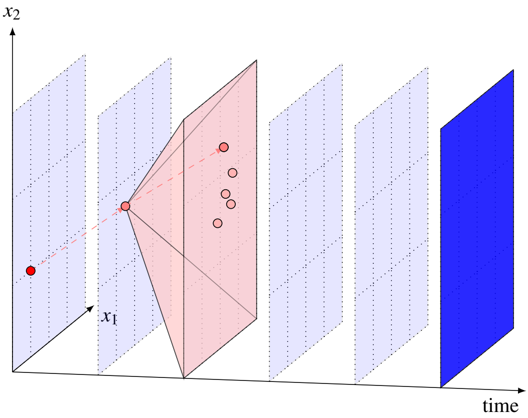
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



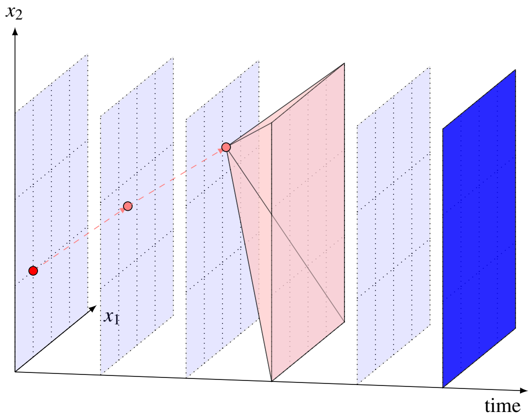
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



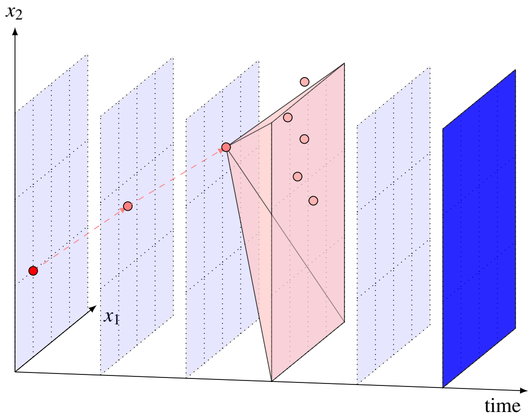
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



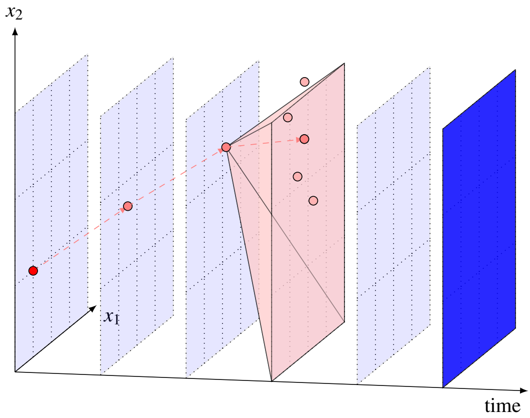
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



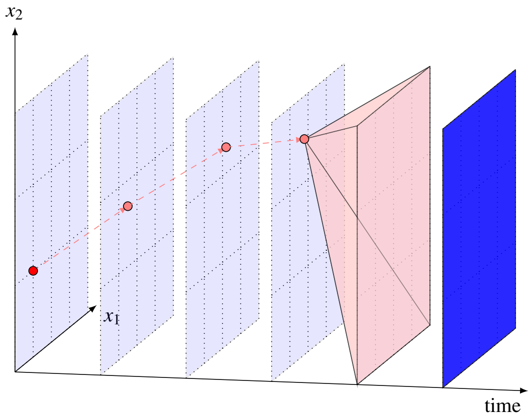
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



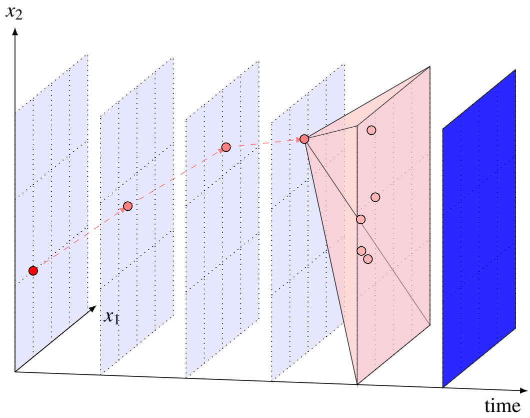
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



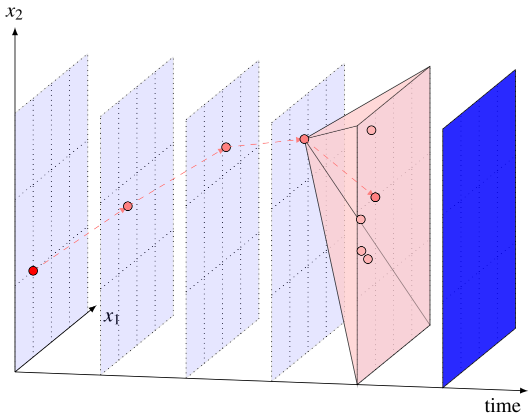
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



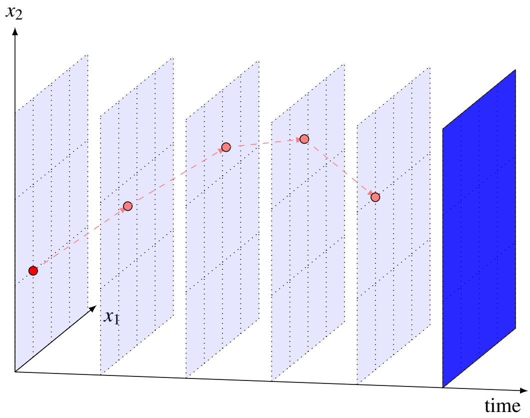
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



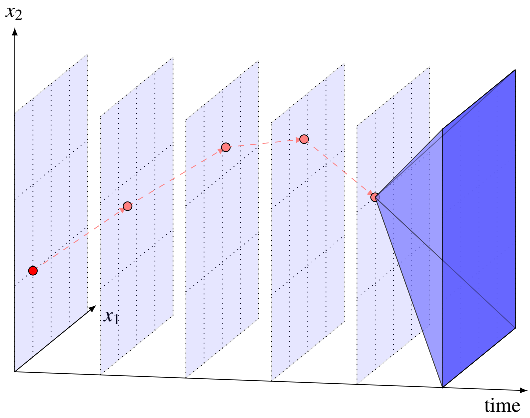
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



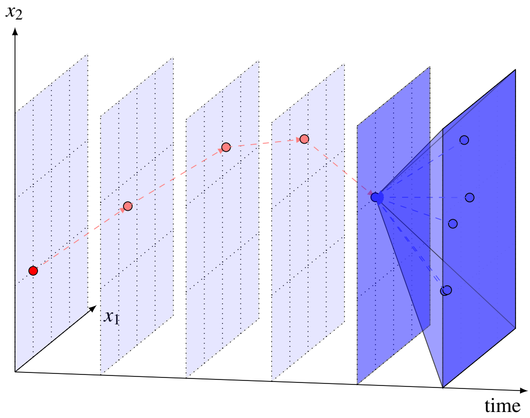
First forward pass : computing trajectory

Stochastic Dual Dynamic Programming



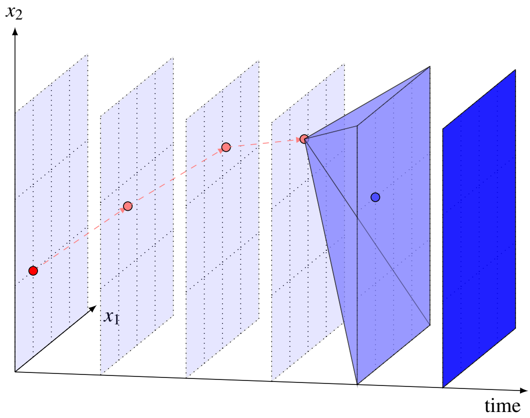
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



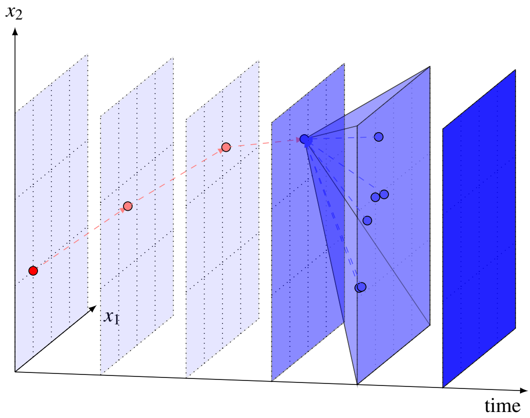
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



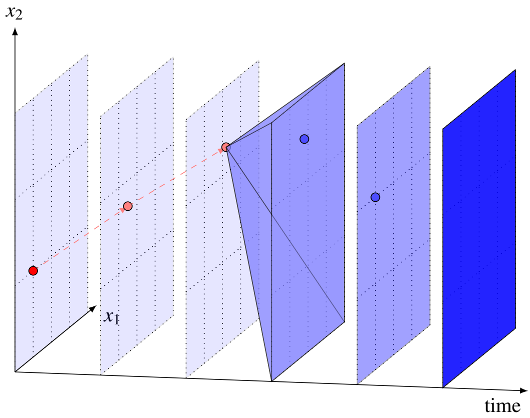
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



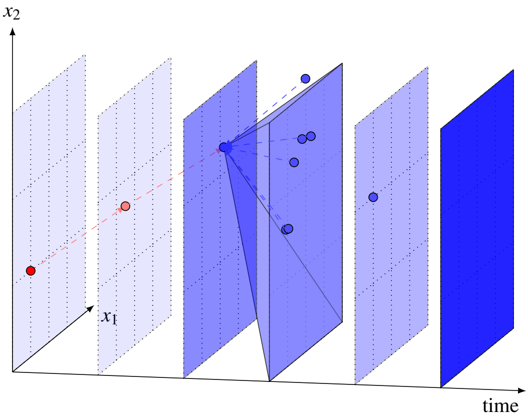
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



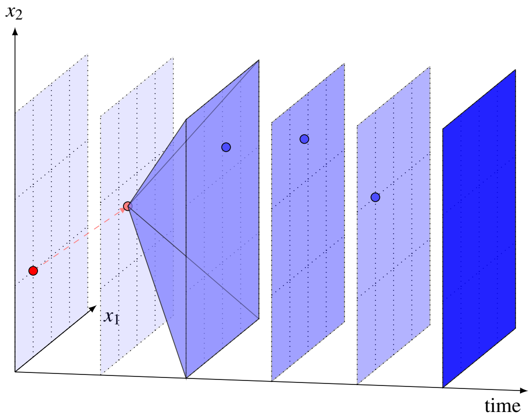
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



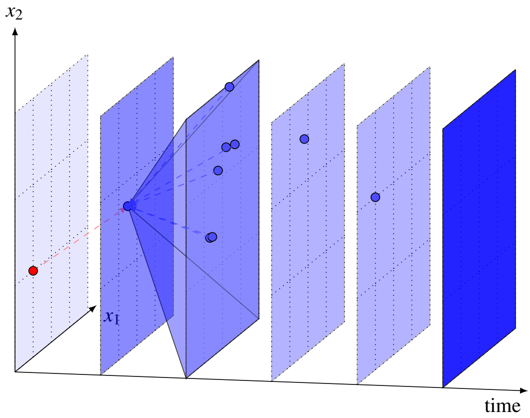
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



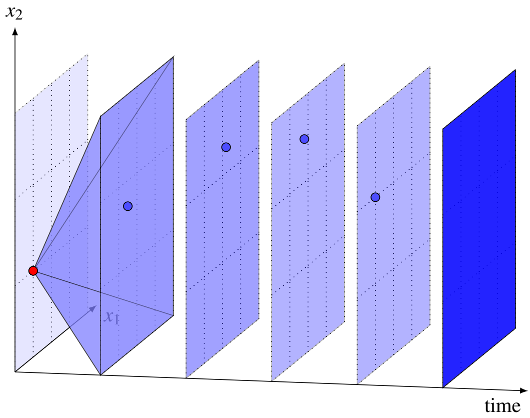
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



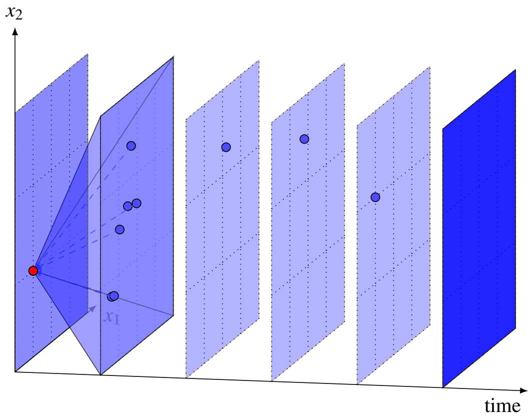
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



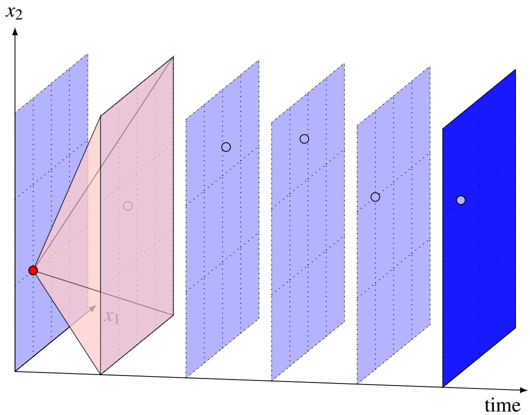
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



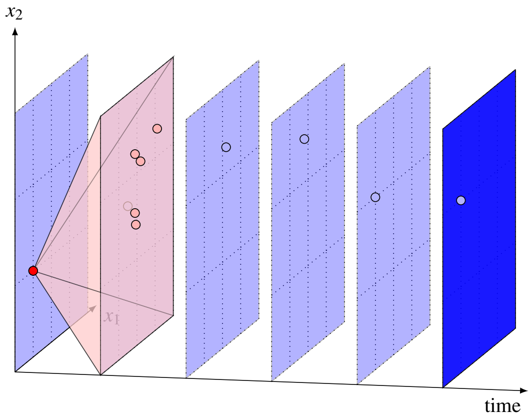
First backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



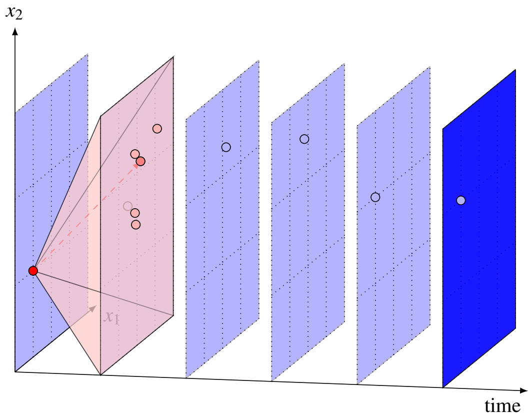
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



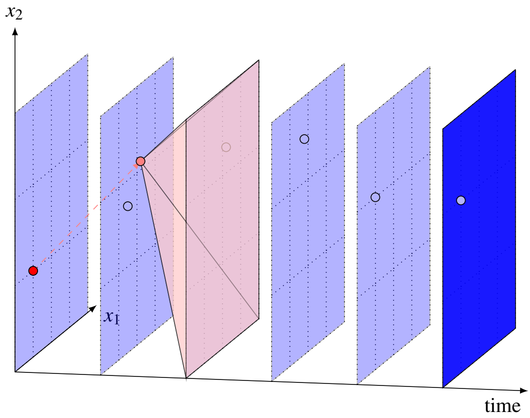
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



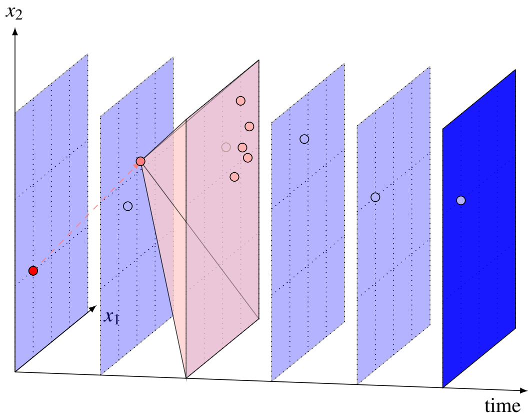
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



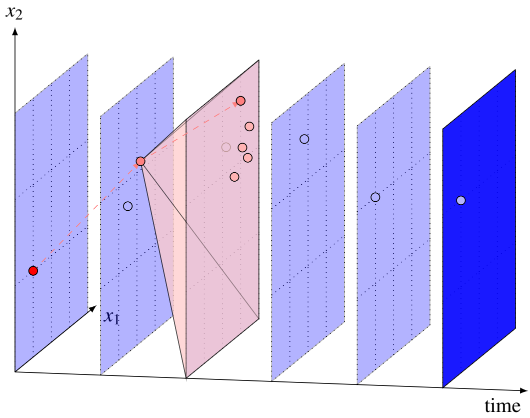
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



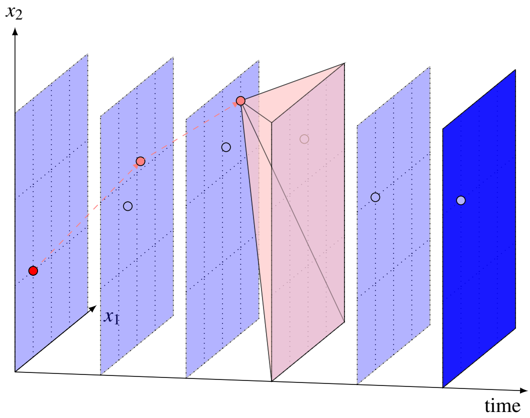
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



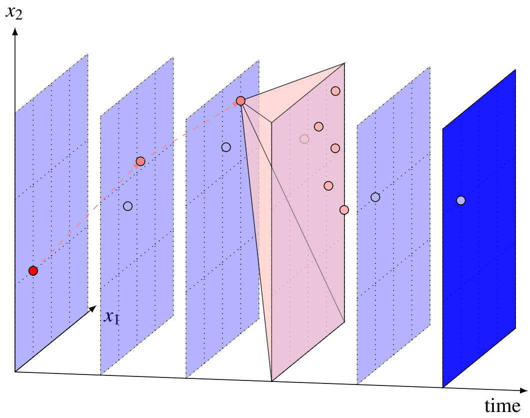
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



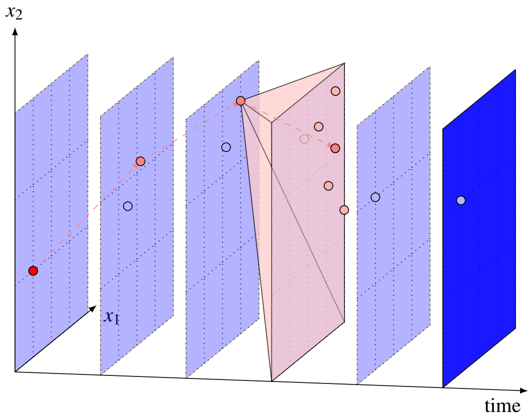
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



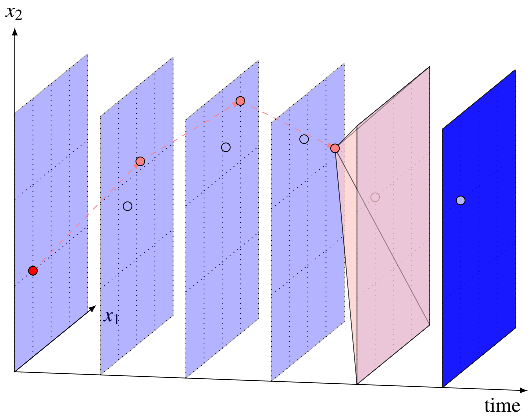
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



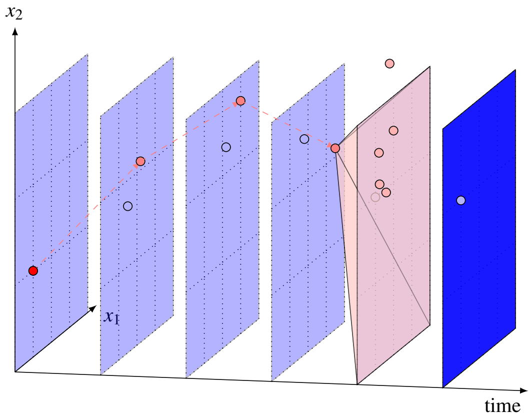
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



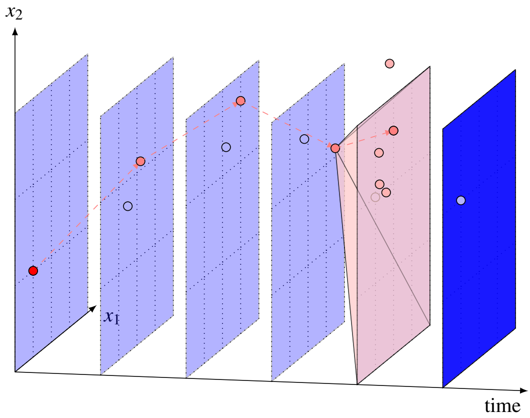
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



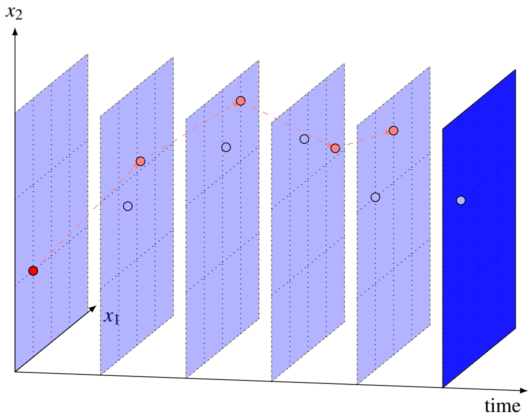
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



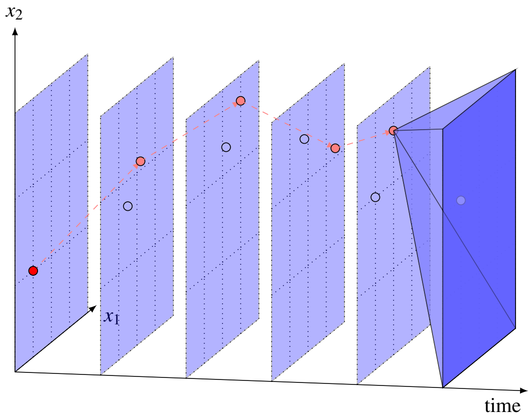
second forward pass : computing trajectory

Stochastic Dual Dynamic Programming



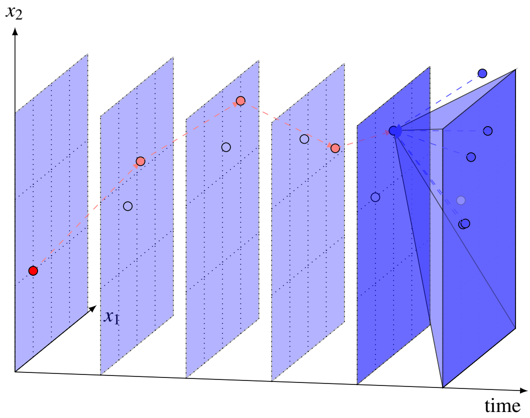
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



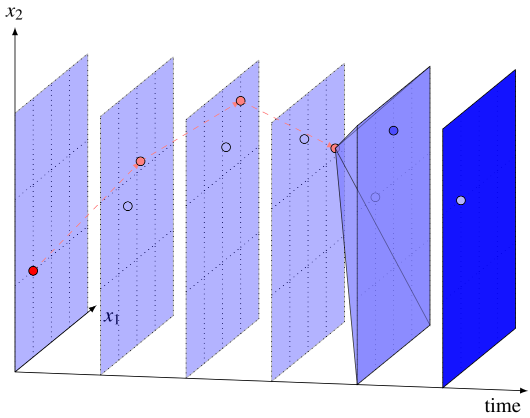
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



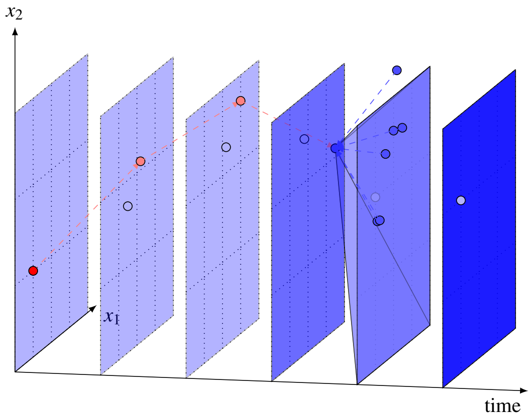
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



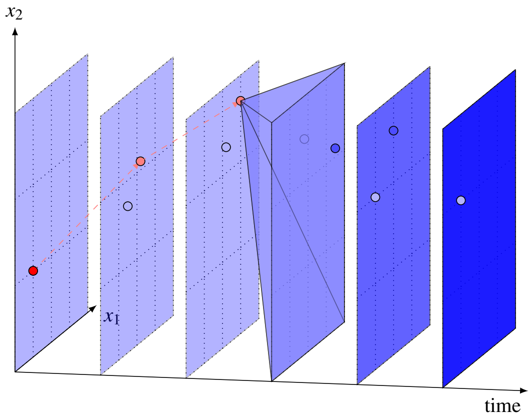
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



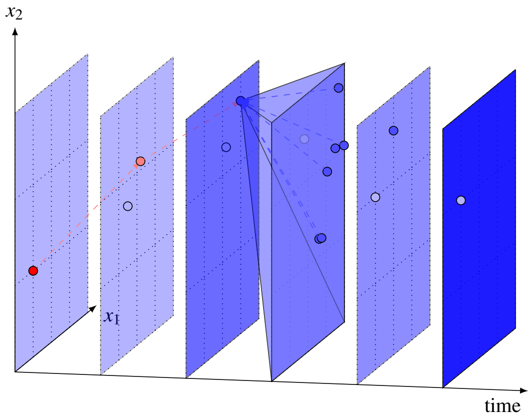
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



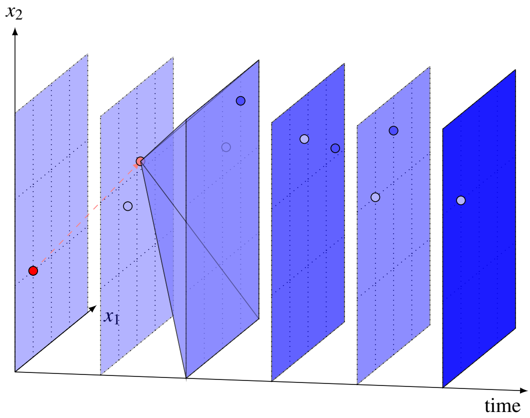
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



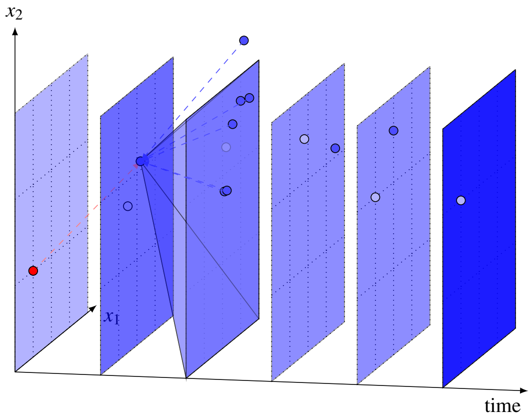
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



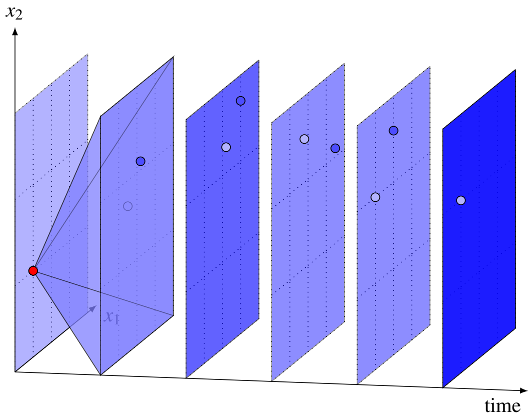
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



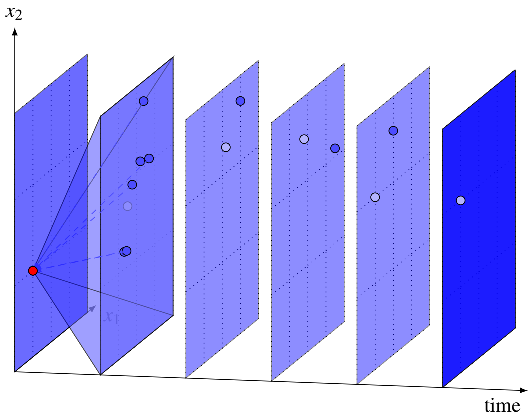
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



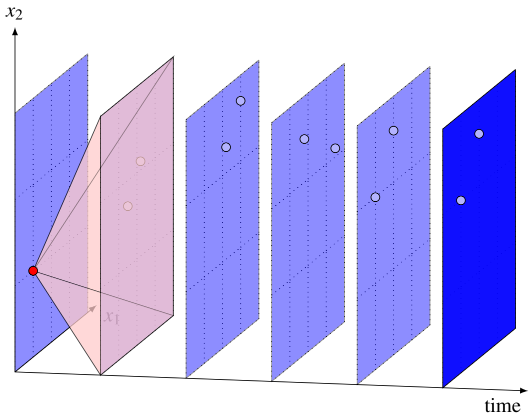
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



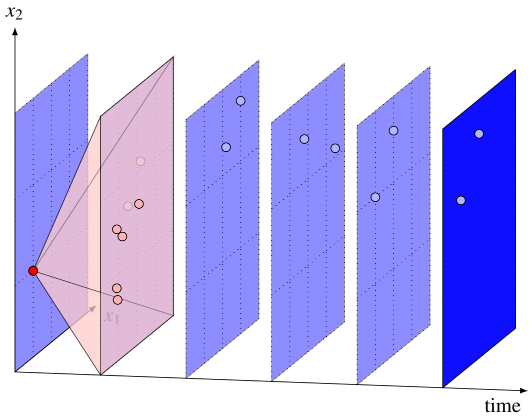
second backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



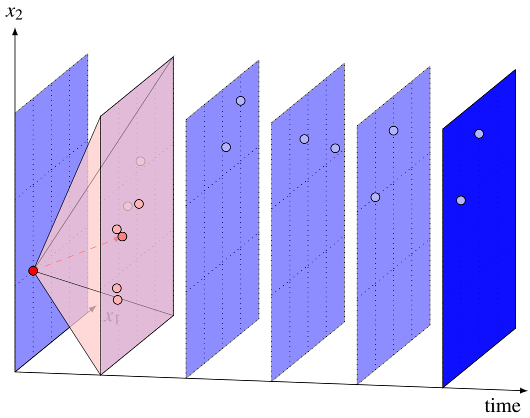
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



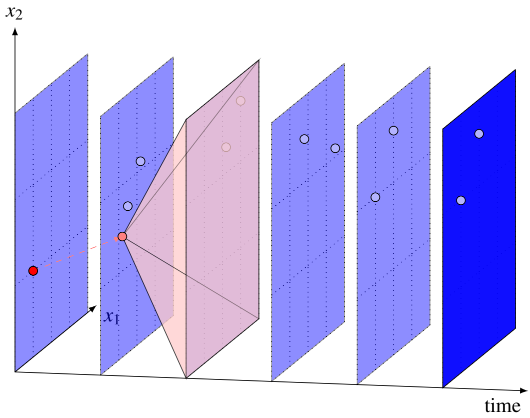
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



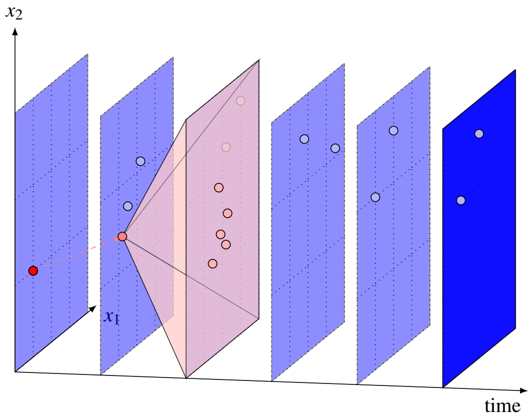
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



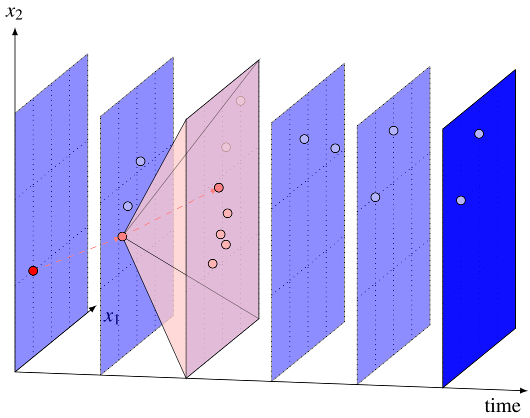
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



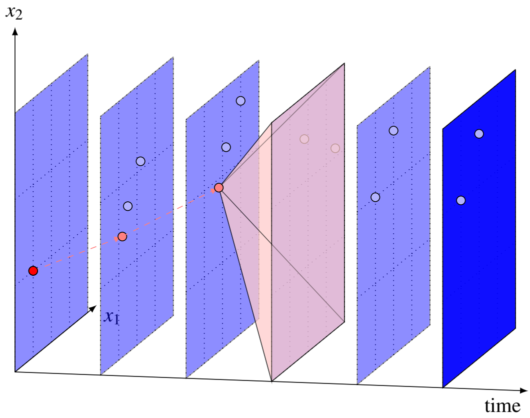
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



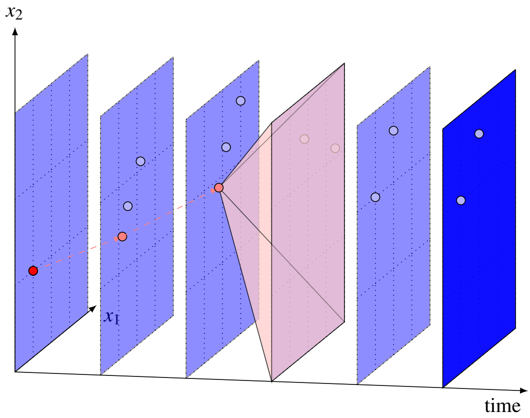
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



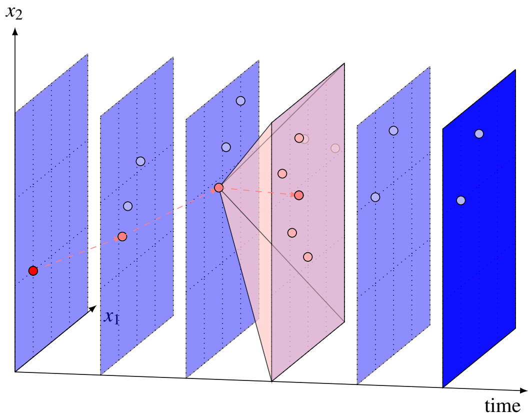
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



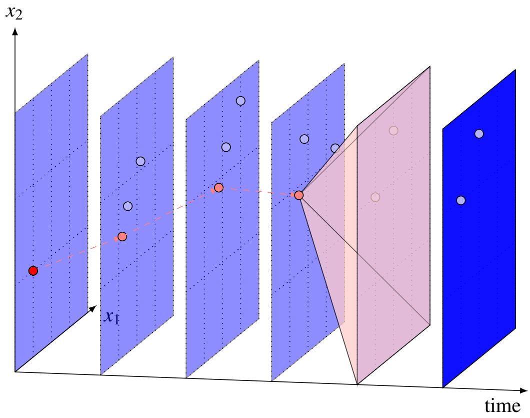
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



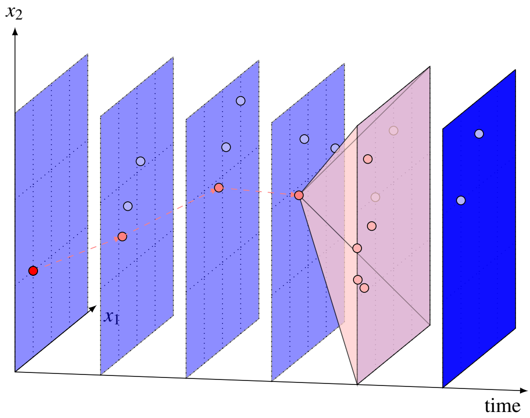
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



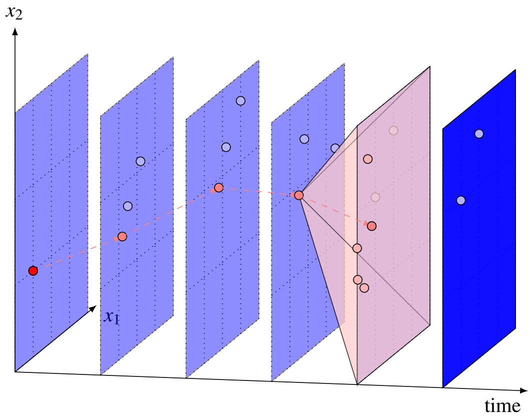
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



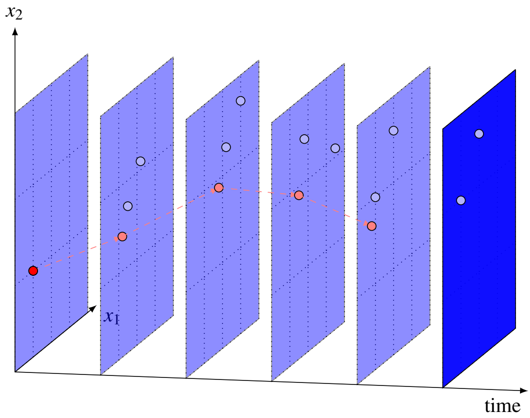
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



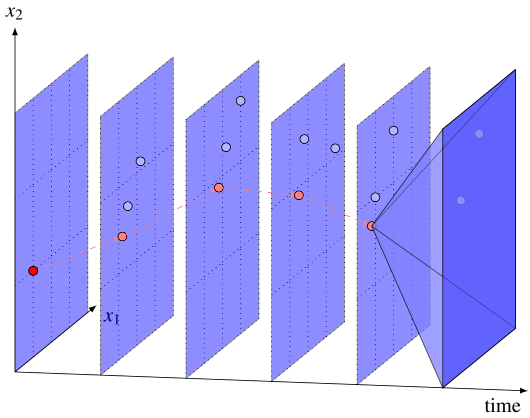
third forward pass : computing trajectory

Stochastic Dual Dynamic Programming



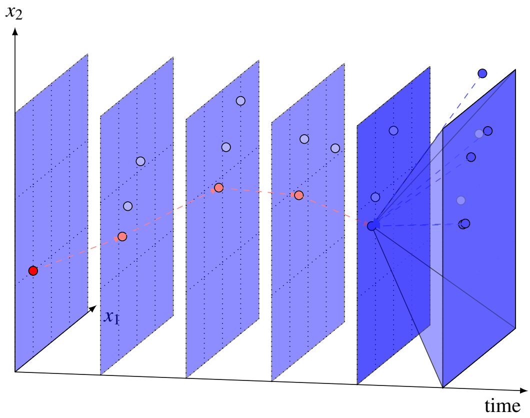
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



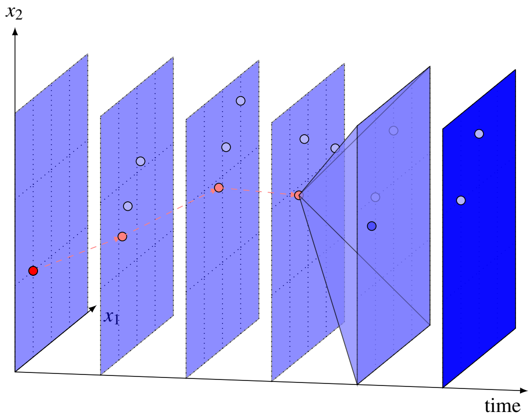
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



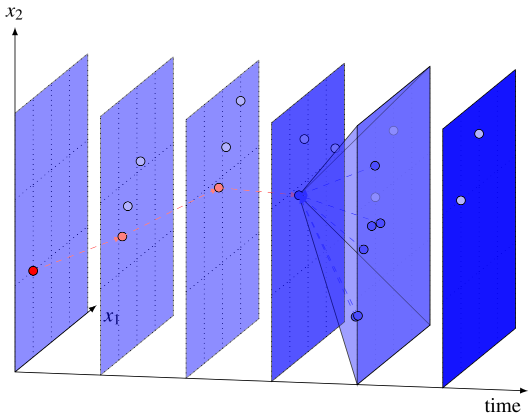
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



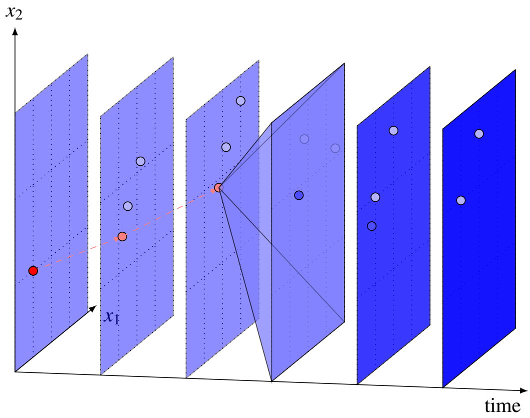
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



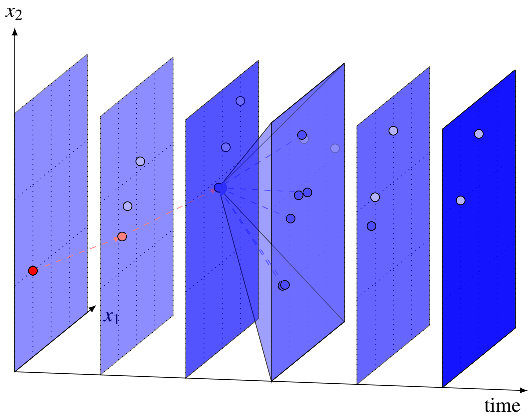
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



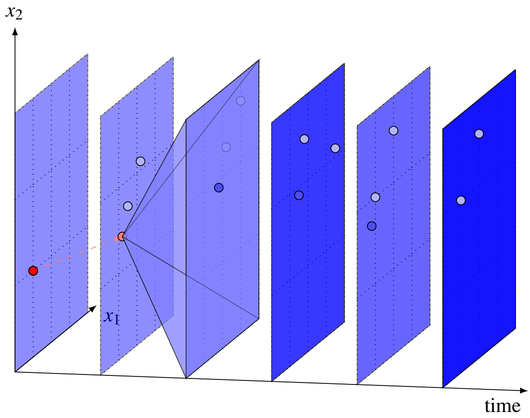
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



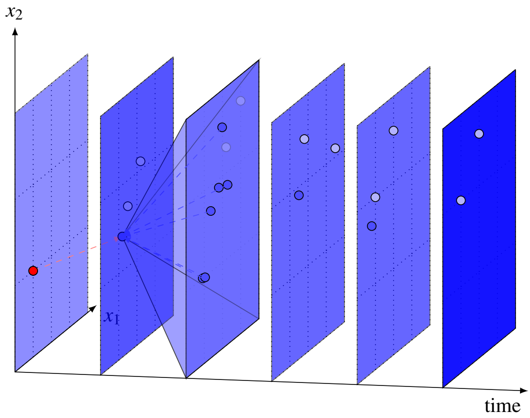
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



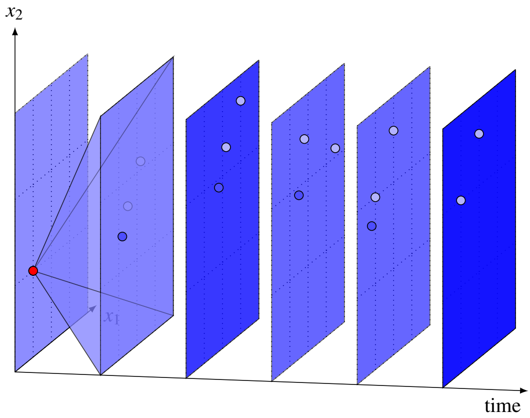
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



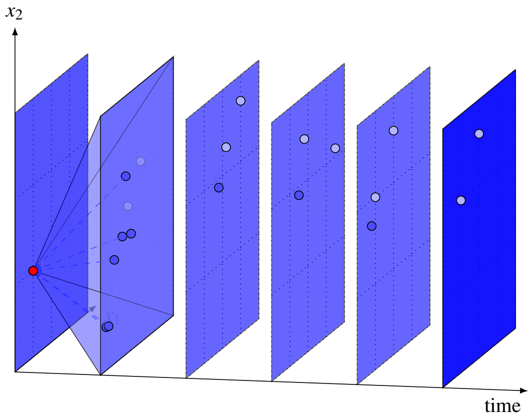
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



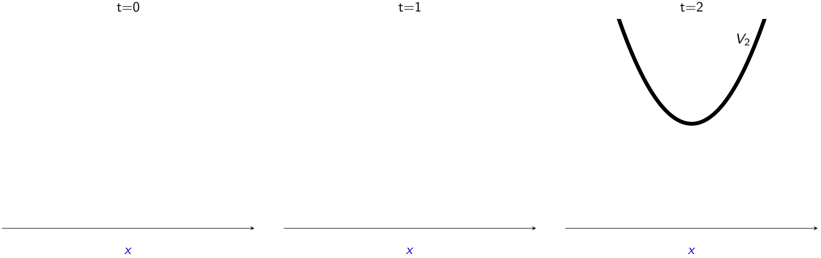
third backward pass : refining approximation (adding cuts)

Stochastic Dual Dynamic Programming



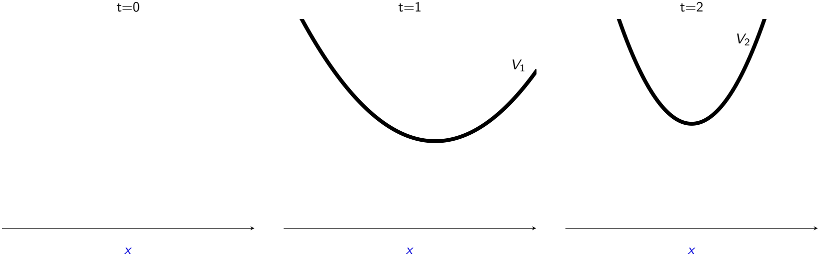
And so on...

SDDP



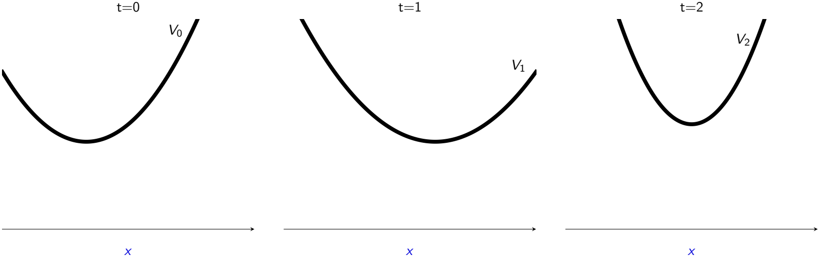
Final Cost $V_2 = V_2$

SDDP



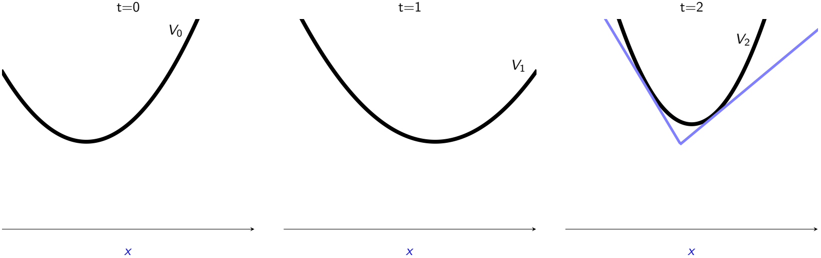
Real Bellman function $V_1 = \mathcal{B}_1(V_2)$

SDDP



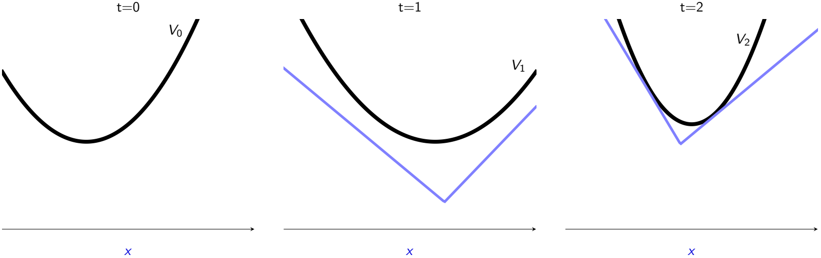
Real Bellman function $V_0 = \mathcal{B}_0(V_1)$

SDDP



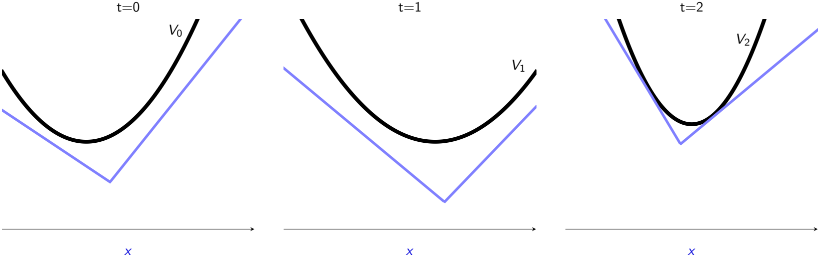
Lower polyhedral approximation $\underline{V}_2$ of V_2

SDDP



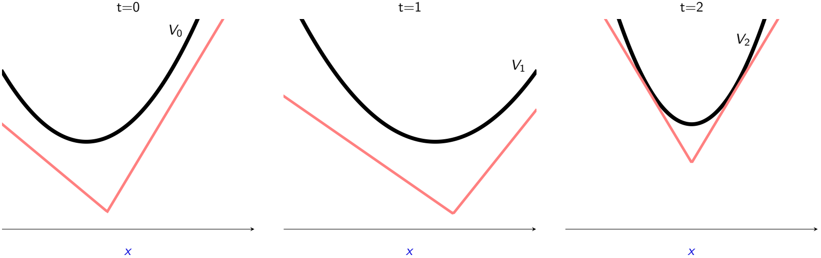
Lower polyhedral approximation $\underline{V}_1 = \mathcal{B}_t(\underline{V}_2)$ of V_1

SDDP



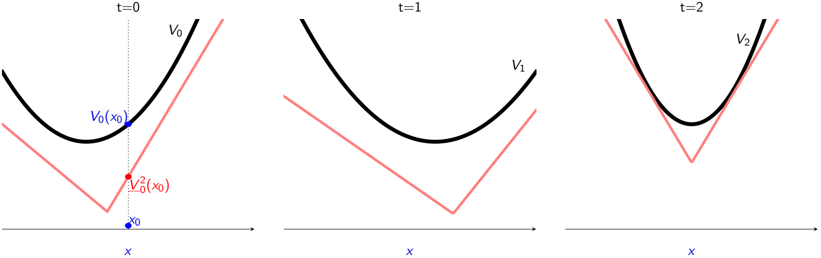
Lower polyhedral approximation $\underline{V}_0 = \mathcal{B}_t(\underline{V}_1)$ of V_0

SDDP



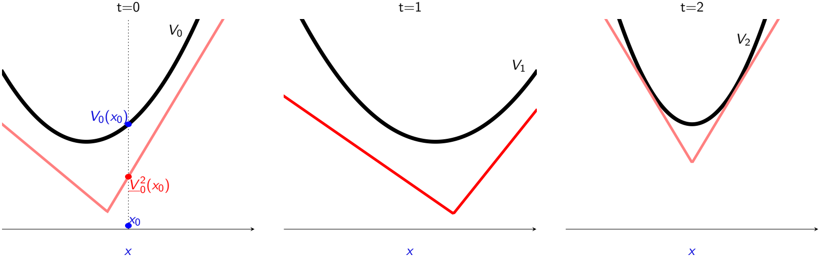
Assume that we have lower polyhedral approximations of V_t

SDDP



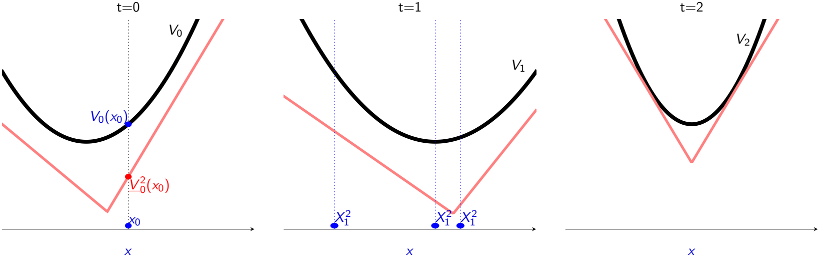
Obtain a lower bound on the value of our problem

SDDP



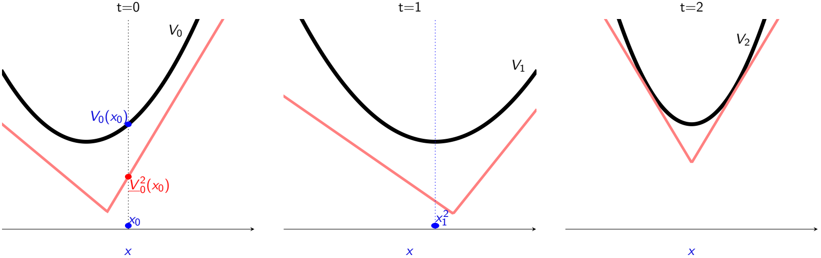
Apply $\mathcal{F}_0\big(\underline{V}_1^{(2)}\big)(x_0)$ and obtain $\boldsymbol{X}_1^{(2)}$

SDDP



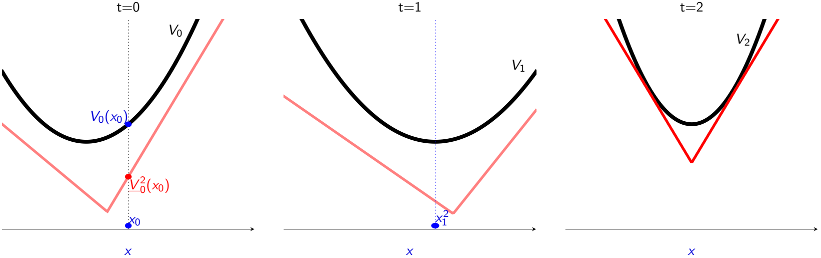
Apply $\mathcal{F}_0(\underline{V}_1^{(2)})(x_0)$ and obtain $\boldsymbol{X}_1^{(2)}$

SDDP



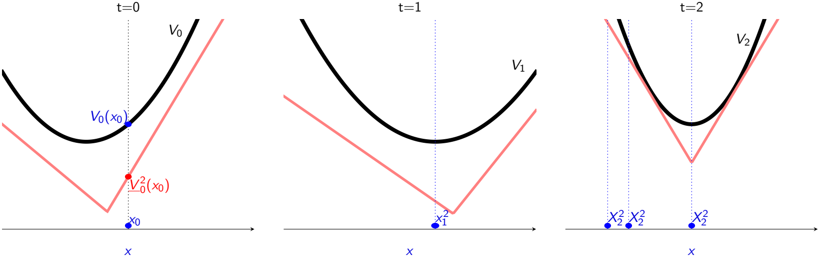
Draw a random realisation $x_1^{(2)}$ of $\boldsymbol{X}_1^{(2)}$

SDDP



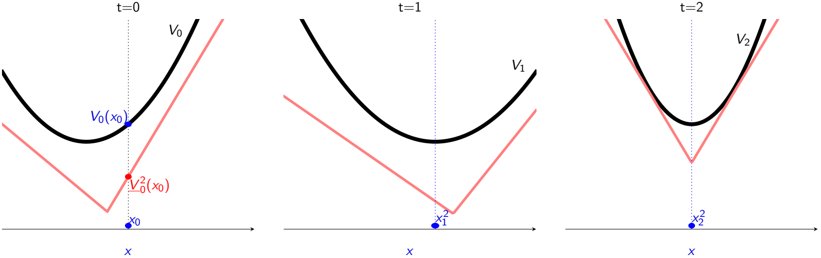
We apply $\mathcal{F}_1\big(\underline{V}_1^{(2)}\big)\big(x_1^{(2)}\big)$ and obtain $\boldsymbol{X}_2^{(2)}$

SDDP



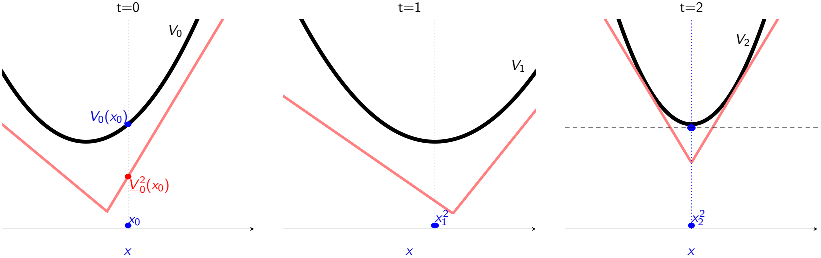
We apply $\mathcal{F}_1(\underline{V}_1^{(2)})(x_1^{(2)})$ and obtain $\boldsymbol{X}_2^{(2)}$

SDDP



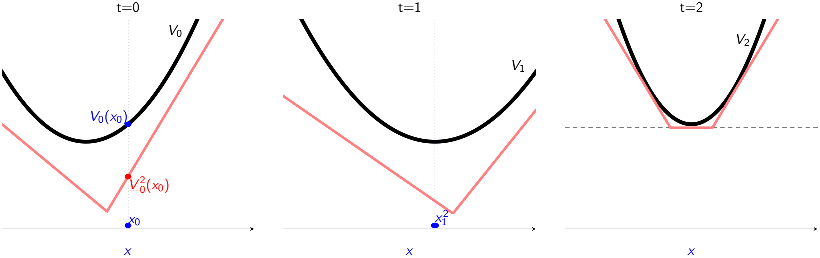
Draw a random realisation $x_2^{(2)}$ of $\boldsymbol{X}_2^{(2)}$

SDDP



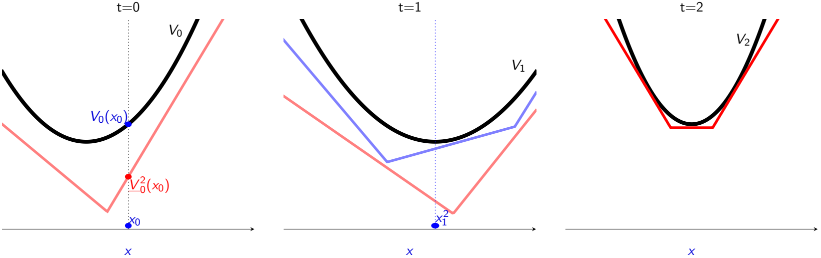
Compute a cut for V_2 at $x_2^{(2)}$

SDDP



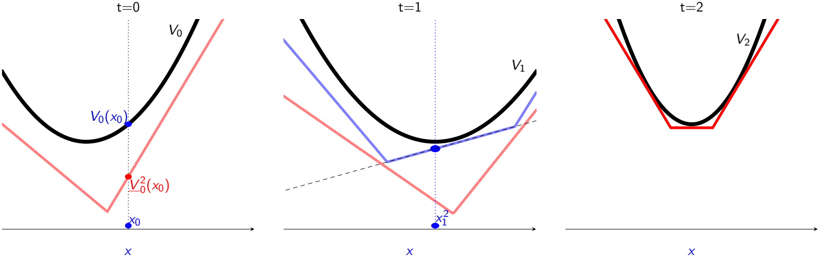
Add the cut to $\underline{V}_2^{(2)}$ which gives $\underline{V}_2^{(3)}$

SDDP



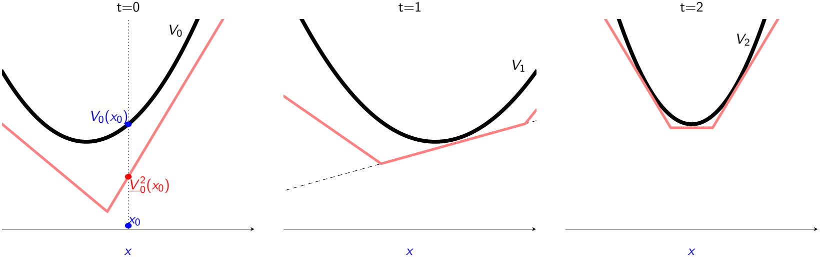
A new lower approximation of V_1 is $\mathcal{B}_1(\underline{V}_2^{(3)})$

SDDP



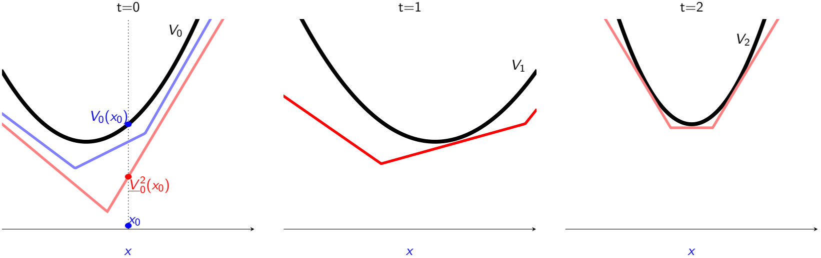
Compute the face active at $x_1^{(2)}$

SDDP



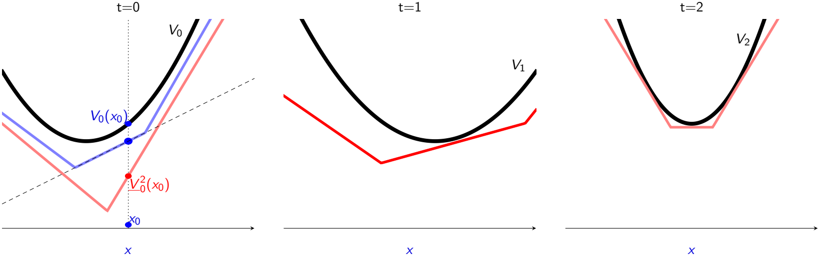
Add the cut to $\underline{V}_1^{(2)}$ which gives $\underline{V}_1^{(3)}$

SDDP



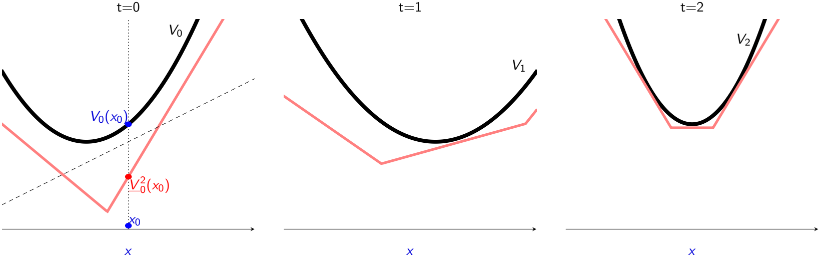
A new lower approximation of V_0 is $\mathcal{B}_0(\underline{V}_1^{(3)})$

SDDP



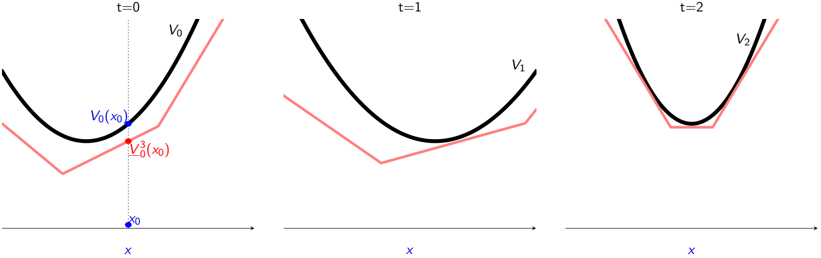
Compute the face active at x_0

SDDP



Compute the face active at x_0

SDDP



Obtain a new lower bound