Trajectory Following Dynamic Programming algorithms

(a.k.a SDDP & friends)

Vincent Leclère
CERMICS
École des Ponts

July 22nd, 2023

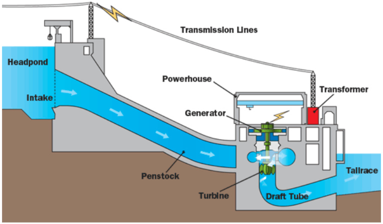
Motivations

- An hydroelectric stock

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} - \boldsymbol{u}_t + \boldsymbol{\xi}_t$$

where, at time t:

  - $\boldsymbol{s}_t$ is the amount of water
  - $\boldsymbol{u}_t$ is the water turbined
  - $\boldsymbol{\xi}_t$ is the inflow
  - $\boldsymbol{p}_t$ is the price



$$\begin{aligned}
\underset{(\boldsymbol{u}_t)_{t=1:T}}{\text{Min}} \quad & \mathbb{E}\left[\sum_{t=1}^{T} -\boldsymbol{p}_t \boldsymbol{u}_t + K(\boldsymbol{s}_T)\right] \\
\text{s.t.} \quad & \boldsymbol{s}_0 = s_{init} && \text{(initial stock)} \\
& \boldsymbol{s}_t = \boldsymbol{s}_{t-1} - \boldsymbol{u}_t + \boldsymbol{\xi}_t && \text{(dynamic)} \\
& 0 \leq \boldsymbol{s}_t \leq \bar{s}_t && \text{(state constraints)} \\
& \sigma(\boldsymbol{u}_t) \subset \sigma(\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_t) && \text{(information constraints)}
\end{aligned}$$

Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\left[\min_{\boldsymbol{u}_t}\left\{ \underbrace{-\boldsymbol{p}_t \boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \right\}\right]$$

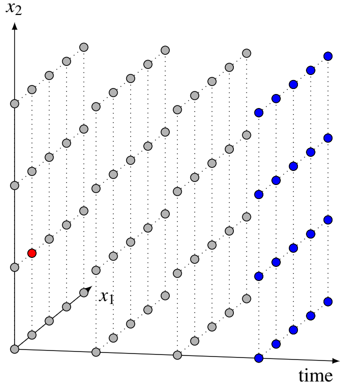
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\left[\min_{\boldsymbol{u}_t}\left\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \right\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



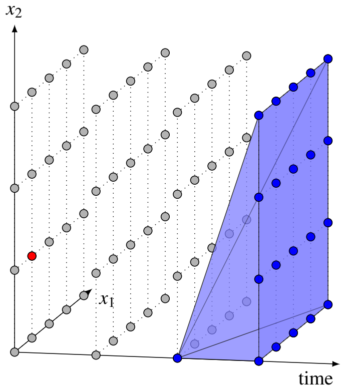
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



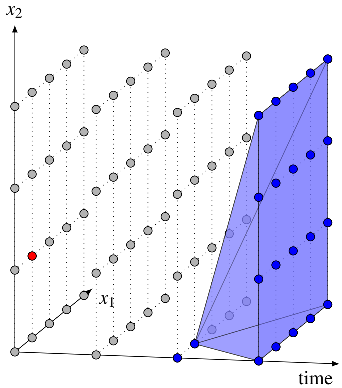
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\left[\min_{\boldsymbol{u}_t}\left\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\right\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



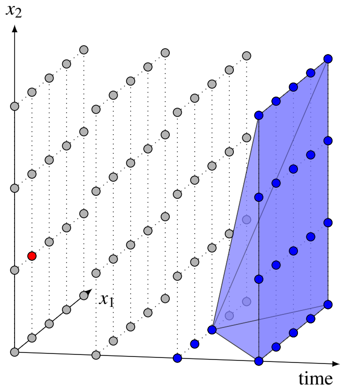
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\left[\min_{u_t}\left\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \right\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



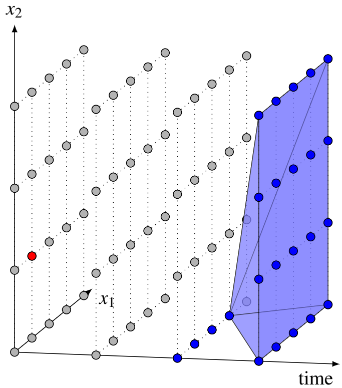
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



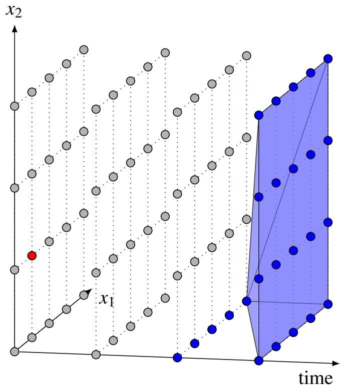
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$

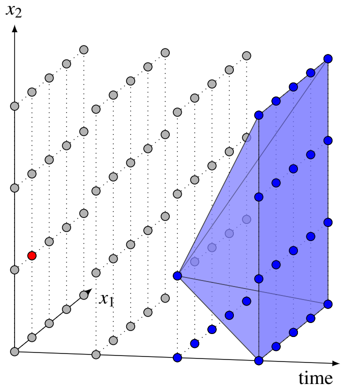
for $t : T-1 \to 0$ **do**

  for $s \in S$ **do**

    for $\xi \in \Xi$ **do**

      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$

      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



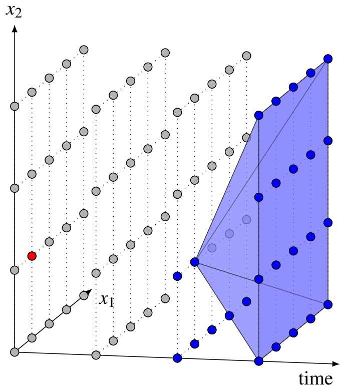
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



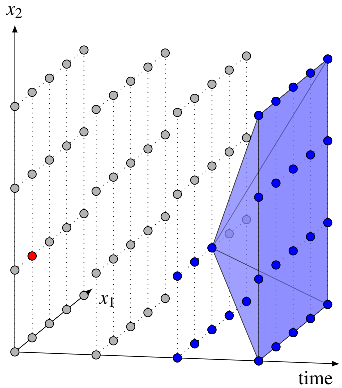
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



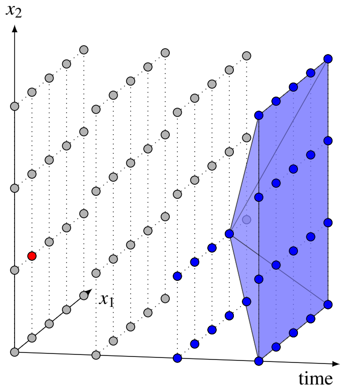
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t \in [T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t} \Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\} \Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u \in \mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



Dynamic Programming

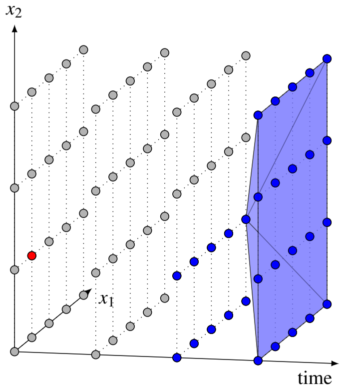
Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

---

1 $V_T \equiv K$; $V_t \equiv 0$
2 **for** $t : T-1 \to 0$ **do**
3 **for** $s \in S$ **do**
4 **for** $\xi \in \Xi$ **do**
5 $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
6 $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



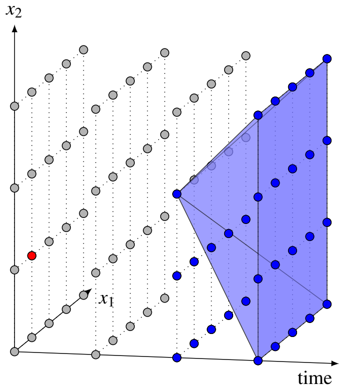
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

```
V_T ≡ K ; V_t ≡ 0
for t : T − 1 → 0 do
    for s ∈ S do
        for ξ ∈ Ξ do
            v̂ = min_{u∈U} −p_t u + V_{t+1}(s − u + ξ)
            V_t(s) += P(ξ_t = ξ) v̂
```



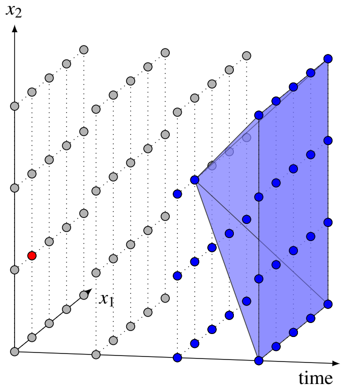
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



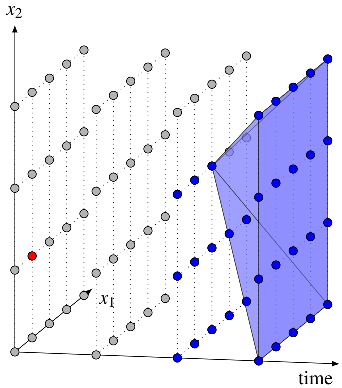
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



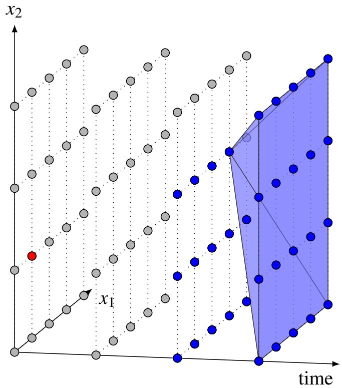
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



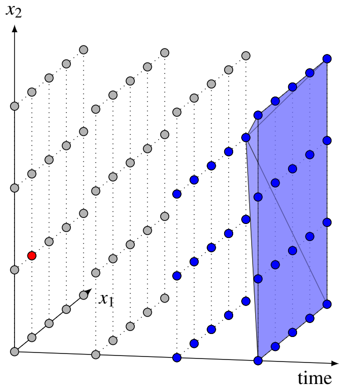
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



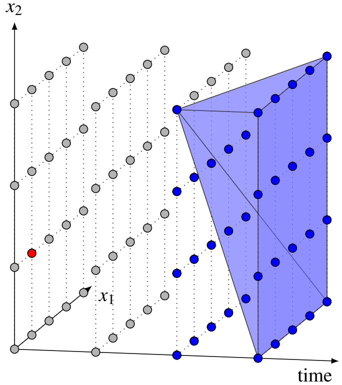
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t \boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u \in \mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



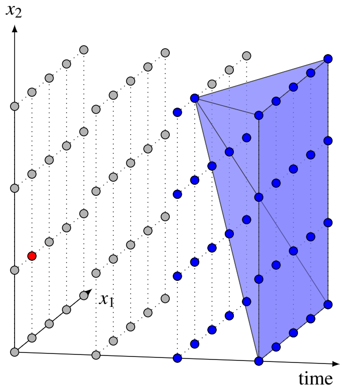
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



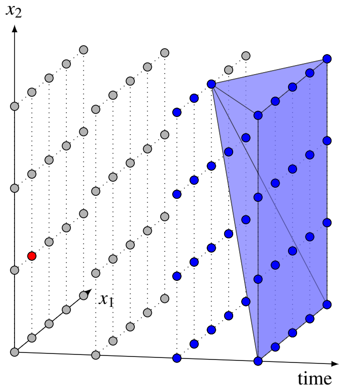
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

```
V_T ≡ K ; V_t ≡ 0
for t : T − 1 → 0 do
    for s ∈ S do
        for ξ ∈ Ξ do
            v̂ = min_{u∈U} −p_t u + V_{t+1}(s − u + ξ)
            V_t(s) += P(ξ_t = ξ) v̂
```



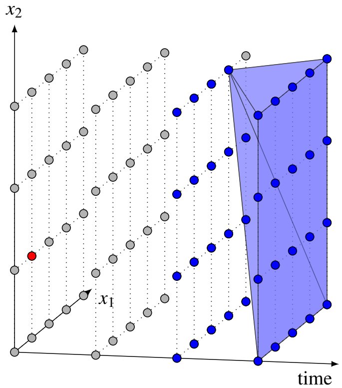
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



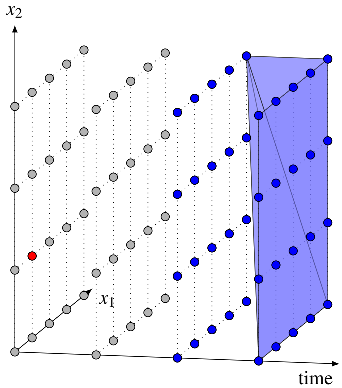
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



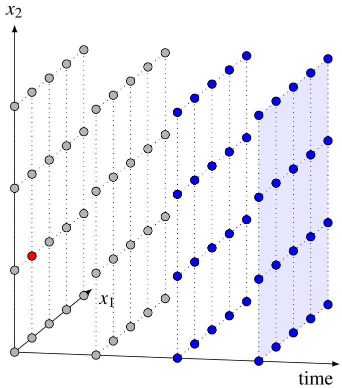
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\left[\min_{u_t}\left\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \right\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) += \mathbb{P}(\xi_t = \xi)\hat{v}$



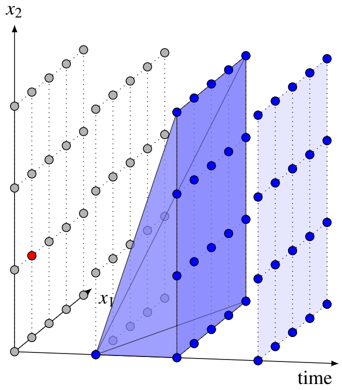
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t \boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



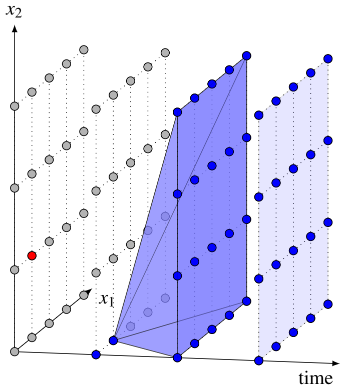
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

1 $V_T \equiv K$; $V_t \equiv 0$
2 **for** $t : T-1 \to 0$ **do**
3   **for** $s \in S$ **do**
4   **for** $\xi \in \Xi$ **do**
5       $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
6       $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



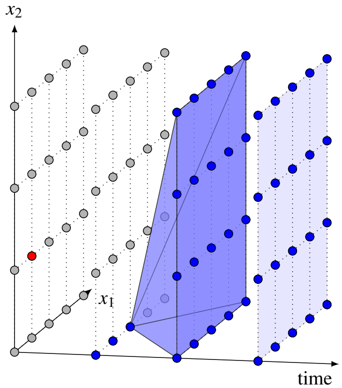
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\left[\min_{\boldsymbol{u}_t}\left\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \right\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

1 $V_T \equiv K$; $V_t \equiv 0$
2 **for** $t : T-1 \to 0$ **do**
3   **for** $s \in S$ **do**
4   **for** $\xi \in \Xi$ **do**
5       $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
6       $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



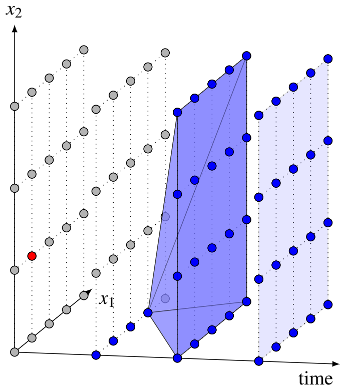
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



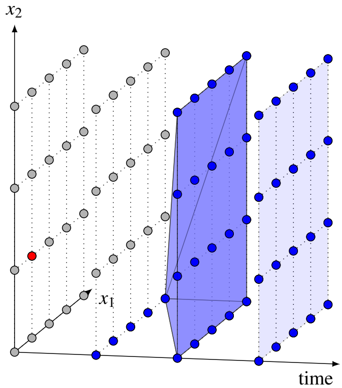
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



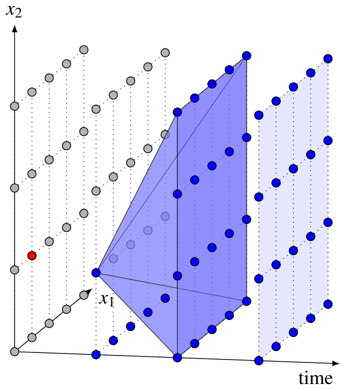
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



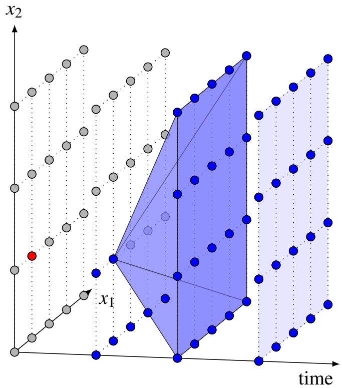
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$

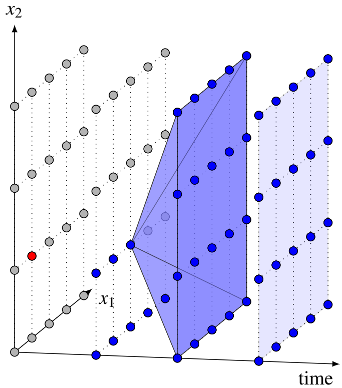
for $t : T-1 \to 0$ **do**

  for $s \in S$ **do**

    for $\xi \in \Xi$ **do**

      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$

      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{\underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$

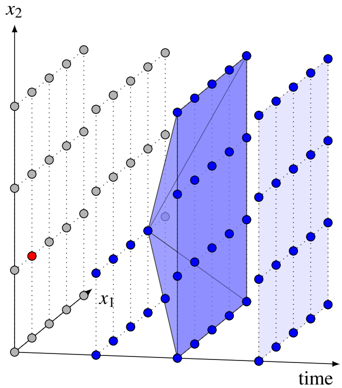
for $t : T-1 \to 0$ **do**

  for $s \in S$ **do**

    for $\xi \in \Xi$ **do**

      $\hat v = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$

      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat v$



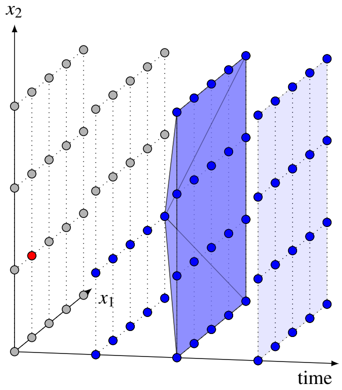
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



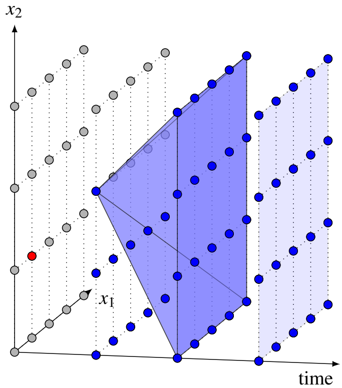
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t \boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



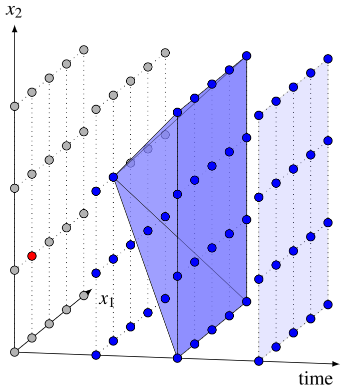
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



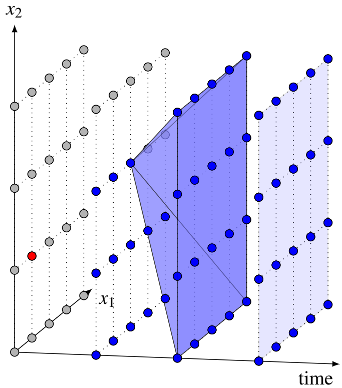
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t \boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



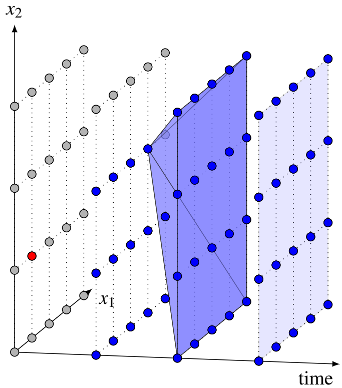
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\left[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - \boldsymbol{u}_t + \boldsymbol{\xi}_t)}_{\text{cost-to-go}} \Big\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



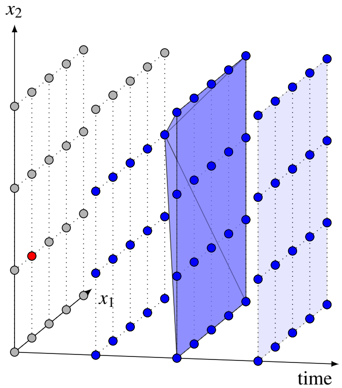
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



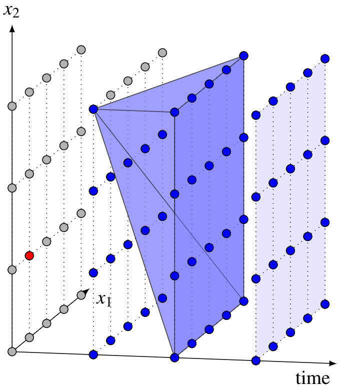
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{\underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

1. $V_T \equiv K$; $V_t \equiv 0$
2. **for** $t : T-1 \to 0$ **do**
3.   **for** $s \in S$ **do**
4.   **for** $\xi \in \Xi$ **do**
5. $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
6. $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



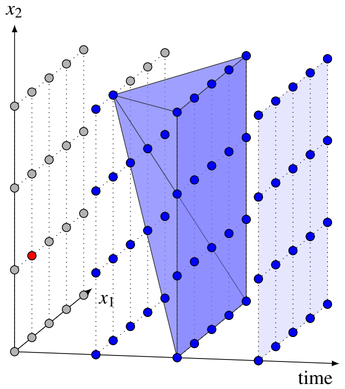
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{ \underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\boldsymbol{\xi}_t = \xi)\hat{v}$



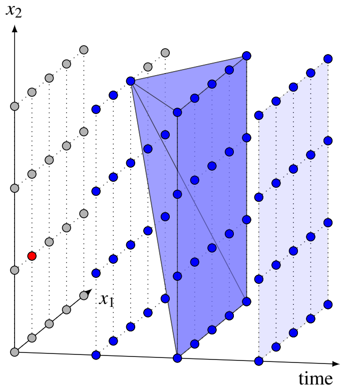
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\boldsymbol{\xi}_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\boldsymbol{\xi}_t}\Big[\min_{\boldsymbol{u}_t}\Big\{\underbrace{-\boldsymbol{p}_t\boldsymbol{u}_t}_{\text{current cost}} + \underbrace{V_{t+1}(s-\boldsymbol{u}_t+\boldsymbol{\xi}_t)}_{\text{cost-to-go}}\Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

```
V_T ≡ K ; V_t ≡ 0
for t : T − 1 → 0 do
    for s ∈ S do
        for ξ ∈ Ξ do
            v̂ = min_{u∈U} −p_t u + V_{t+1}(s − u + ξ)
            V_t(s) += P(ξ_t = ξ) v̂
```



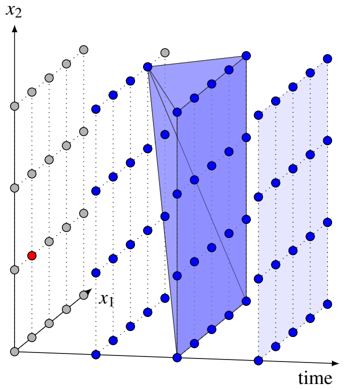
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$
for $t : T-1 \to 0$ **do**
  for $s \in S$ **do**
    for $\xi \in \Xi$ **do**
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$
      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



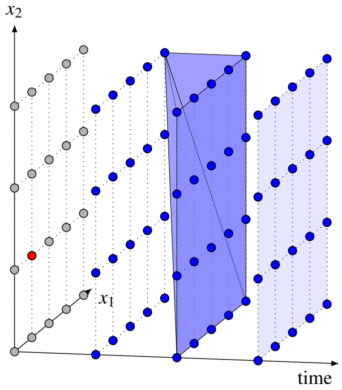
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\left[\min_{u_t}\left\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \right\}\right]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

1 $V_T \equiv K$; $V_t \equiv 0$
2 **for** $t : T-1 \to 0$ **do**
3   **for** $s \in S$ **do**
4   **for** $\xi \in \Xi$ **do**
5       $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s-u+\xi)$
6       $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



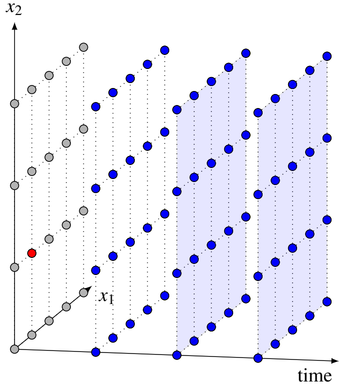
Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

```
V_T ≡ K ; V_t ≡ 0
for t : T − 1 → 0 do
    for s ∈ S do
        for ξ ∈ Ξ do
            v̂ = min_{u∈𝒰} −p_t u + V_{t+1}(s − u + ξ)
            V_t(s) += ℙ(ξ_t = ξ) v̂
```



Dynamic Programming

Under a crucial stagewise independence assumption (*i.e.* $(\xi_t)_{t\in[T]}$ is a sequence of independent random variables), we have the Bellman equation

$$V_t(s) = \mathbb{E}_{\xi_t}\Big[\min_{u_t}\Big\{ \underbrace{-p_t u_t}_{\text{current cost}} + \underbrace{V_{t+1}(s - u_t + \xi_t)}_{\text{cost-to-go}} \Big\}\Big]$$

Algorithm 1: Discretized Stochastic Dynamic Programming

$V_T \equiv K$; $V_t \equiv 0$

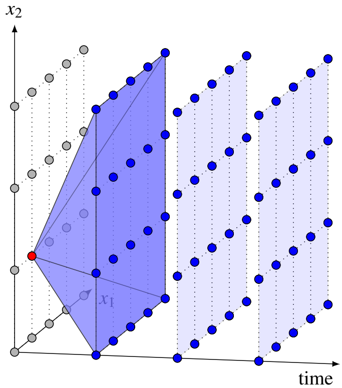
for $t : T-1 \to 0$ **do**

  for $s \in S$ **do**

    for $\xi \in \Xi$ **do**

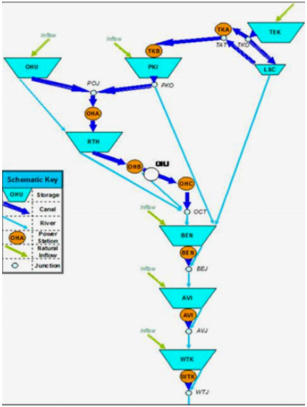
      $\hat{v} = \min_{u\in\mathcal{U}} -p_t u + V_{t+1}(s - u + \xi)$

      $V_t(s) \mathrel{+}= \mathbb{P}(\xi_t = \xi)\hat{v}$



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[1] in ≈ 10 minutes

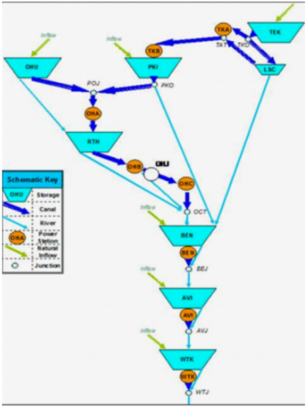
[1] Approximately, depending on the problem and precision required...

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)

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

➥ Can be solved[1] in $\approx$ 10 minutes



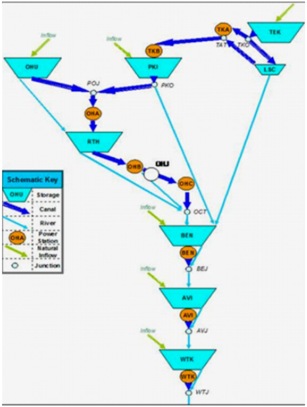
[1] Approximately, depending on the problem and precision required...

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)

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

➥ Can be solved[1] in $\approx$ 10 minutes



[1] 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

Some TFDP algorithms

Algorithm's name	Node selection: Choice ξ_t^k	$\mathcal{F}_t$	$\underline{V}_t^k$	$\overline{V}_t^k$	Hypothesis	Complexity known
SDDP	Random sampling	Exact	Benders cuts	V_t	Convex	✔
EDDP	Explorative	Exact	Benders cuts	V_t	Convex	✔
APSDDP	Random sampling	Exact	Adaptive partition	V_t	Linear	✖
SDDiP	Random sampling	Exact	Lagrangian or integer cuts	V_t	Mixed Integer Linear	✖
MIDAS	Random sampling	Exact	Step cuts	V_t	Monotonic Mixed Integer	✖
SLDP	Random sampling	Exact	Reverse norm cuts	V_t	Non-Convex	✖
BDZ17	Problem child	Exact	Benders cuts	Epigraph as convex hull	Convex	✖
BDZ18	Problem child	Exact	Benders $\times$ Epigraph	Hypograph $\times$ Benders	Convex-Concave	✖
RDDP	Deterministic	Exact	Benders cuts	Epigraph as convex hull	Robust	✖
ISDDP	Random sampling	Inexact	Inexact Lagrangian cuts	V_t	Convex	✖
TDP	Problem child	Exact	Benders cuts	Min of quadratic	Convex	✖
ZS19	Random or Problem	Regularized	Generalized conjugacy cuts	Norm cuts	Mixed Integer Convex	✔
NDDP	Random or Problem	Regularized	Benders cuts	Norm cuts	Distributionally Robust	✔
DSDDP	Random sampling	Exact	Benders cuts	Fenchel transform	Linear	✖

Contents

Problem setting

- The risk-neutral Multistage Stochastic Program considered reads

$$\begin{aligned} \min \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] && \text{(MSP)} \\ \text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \in [T] \\ & \boldsymbol{x}_t, \boldsymbol{u}_t \preceq \sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]}) && \forall t \in [T], \end{aligned}$$

- where:
  - $\boldsymbol{x}_t$ is the state, that convey information from the past,
  - $\boldsymbol{u}_t$ the control, which only impact stage t,
  - $\boldsymbol{\xi}_t$ the (exogeneous) noise.
- Note that:
  - finite, discrete time
  - contraints are stagewise independent
  - $\boldsymbol{x}_t \preceq \sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]})$ means that $\boldsymbol{x}_t$ is measurable w.r.t. $\sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]})$

Problem setting

- The risk-neutral Multistage Stochastic Program considered reads

$$\begin{aligned} \min \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] && \text{(MSP)} \\ \text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \in [T] \\ & \boldsymbol{x}_t, \boldsymbol{u}_t \preceq \sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]}) && \forall t \in [T], \end{aligned}$$

- where:
  - $\boldsymbol{x}_t$ is the state, that convey information from the past,
  - $\boldsymbol{u}_t$ the control, which only impact stage t,
  - $\boldsymbol{\xi}_t$ the (exogeneous) noise.
- Note that:
  - finite, discrete time
  - contraints are stagewise independent
  - $\boldsymbol{x}_t \preceq \sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]})$ means that $\boldsymbol{x}_t$ is measurable w.r.t. $\sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]})$

Problem setting II

We often encouter MSPs with more compact formulation than:

$$\begin{aligned} \min \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] & & \text{(MSP)} \\ \text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) & \forall t \in [T] & \\ & \boldsymbol{x}_t, \boldsymbol{u}_t \preceq \sigma(\{\boldsymbol{\xi}_\tau\}_{\tau\in[t]}) & \forall t \in [T] & \end{aligned}$$

Here are some examples:

- without u_t: use the cheapest control getting you from x_{t-1} to x_t;
- with explicit dynamic: $x_{t+1} = \mathrm{dyn}_t(x_t, u_t, \xi_t)$;
- with cost depending only on the control u_t or the out-state x_t;
- a linear setting I favor:
  - $\ell_t(\boldsymbol{x}_{t-1}\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) := \boldsymbol{c}_t{}^\top \boldsymbol{u}_t$,
  - $\mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) := \big\{\boldsymbol{x}_t \in \mathbb{R}_+^{n_t} \mid \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{x}_{t-1} + \boldsymbol{T}_t\boldsymbol{u}_t\big\} = \boldsymbol{d}_t$.

➥ For DP approaches, it is worth it to keep in mind the difference between state and control variables.

Problem setting II

We often encouter MSPs with more compact formulation than:

$$\begin{aligned} \min \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] & & \text{(MSP)} \\ \text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) & \forall t \in [T] & \\ & \boldsymbol{x}_t, \boldsymbol{u}_t \preceq \sigma(\{\boldsymbol{\xi}_\tau\}_{\tau \in [t]}) & \forall t \in [T] & \end{aligned}$$

Here are some examples:

- without u_t: use the cheapest control getting you from x_{t-1} to x_t;
- with explicit dynamic: $x_{t+1} = \mathrm{dyn}_t(x_t, u_t, \xi_t)$;
- with cost depending only on the control u_t or the out-state x_t;
- a linear setting I favor:
  - $\ell_t(\boldsymbol{x}_{t-1}\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) := \boldsymbol{c}_t{}^\top \boldsymbol{u}_t$,
  - $\mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) := \big\{\boldsymbol{x}_t \in \mathbb{R}_+^{n_t} \mid \ \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{x}_{t-1} + \boldsymbol{T}_t \boldsymbol{u}_t\big\} = \boldsymbol{d}_t$.

➥ For DP approaches, it is worth it to keep in mind the difference between state and control variables.

Dynamic Programming principle

The main idea of Dynamic Programming is that, under stagewise independence, we can look for an optimal solution as a function of the state instead of the past noises.

$$
\begin{aligned}
\min_{\boldsymbol{x}_{1:T}, \boldsymbol{u}_{1:T}} \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] && \text{(MSP)} \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \in [T] \\
& \boldsymbol{x}_t, \boldsymbol{u}_t \preceq \sigma(\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_t) && \forall t \in [T].
\end{aligned}
$$

Dynamic Programming principle

The main idea of Dynamic Programming is that, under stagewise independence, we can look for an optimal solution as a function of the state instead of the past noises.

$$
\begin{aligned}
\min_{\Phi_{1:T}} \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] && \text{(MSP)} \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Phi_t(\boldsymbol{\xi}_1, \dots, \boldsymbol{\xi}_t) && \forall t \in [T].
\end{aligned}
$$

Dynamic Programming principle

The main idea of Dynamic Programming is that, under stagewise independence, we can look for an optimal solution as a function of the state instead of the past noises.

$$
\begin{aligned}
\min_{\Psi_{1:T}} \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] && \text{(MSP)} \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \in [T].
\end{aligned}
$$

Dynamic Programming equation

$$
\begin{aligned}
\min_{\Psi_{1:T}} \quad & \mathbb{E}\left[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\right] & \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) & \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) & \forall t \in [T]
\end{aligned}
$$

Dynamic Programming equation

$$
\begin{aligned}
\min_{\Psi_{1:T}} \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) \qquad \forall t \in [T] \\
= \min_{\Psi_{1:T}} \quad & \mathbb{E}\Big[\ell_1(x_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \boldsymbol{\xi}_1) + \mathbb{E}\Big[\sum_{t=2}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \Big| \boldsymbol{\xi}_1\Big]\Big] \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) \qquad \forall t \in [T]
\end{aligned}
$$

Dynamic Programming equation

$$
\begin{aligned}
\min_{\Psi_{1:T}} \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big] \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) \qquad \forall t \in [T]
\end{aligned}
$$

$$
\begin{aligned}
= \min_{\Psi_{1:T}} \quad & \mathbb{E}\Big[\ell_1(x_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \boldsymbol{\xi}_1) + \mathbb{E}\Big[\sum_{t=2}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big| \boldsymbol{\xi}_1\Big]\Big] \\
\text{s.t.} \quad & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \in [T] \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) \qquad \forall t \in [T]
\end{aligned}
$$

$$
= \mathbb{E}\Big[\min_{\boldsymbol{x}_1, \boldsymbol{u}_1 \in \mathcal{X}_1(x_0, \boldsymbol{\xi}_1)} \ell_1(x_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \boldsymbol{\xi}_1) + \underbrace{\min_{\Psi_{2:T}} \mathbb{E}\Big[\sum_{t=2}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big| \boldsymbol{\xi}_1\Big]\Big] \quad \text{s.t.} \quad (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1})}_{:= V_2(\boldsymbol{x}_1; \boldsymbol{\xi}_1)}
$$

Dynamic Programming equation

$$\min_{\Psi_{1:T}} \quad \mathbb{E}\Big[\sum_{t=1}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t)\Big]$$

$$\text{s.t.} \quad (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \in [T]$$

$$(\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) \qquad \forall t \in [T]$$

$$= \min_{\Psi_{1:T}} \quad \mathbb{E}\Big[\ell_1(x_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \boldsymbol{\xi}_1) + \mathbb{E}\Big[\sum_{t=2}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \Big| \boldsymbol{\xi}_1\Big]\Big]$$

$$\text{s.t.} \quad (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \in [T]$$

$$(\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(x_{t-1}) \qquad \forall t \in [T]$$

$$= \mathbb{E}\Big[\min_{\boldsymbol{x}_1, \boldsymbol{u}_1 \in \mathcal{X}_1(x_0, \boldsymbol{\xi}_1)} \ell_1(x_0, \boldsymbol{x}_1, \boldsymbol{u}_1, \boldsymbol{\xi}_1) + \underbrace{\min_{\Psi_{2:T}} \mathbb{E}\Big[\sum_{t=2}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \Big| \not{\boldsymbol{\xi}_1}\Big] \quad \text{s.t.} \quad (\boldsymbol{x}_t, \boldsymbol{u}_t) = \Psi_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1})}_{:= V_2(\boldsymbol{x}_1; \not{\boldsymbol{\xi}_1})}\Big]$$

Backward Bellman operators and Dynamic Programming

Define the cost-to-go, or value function

$$
\begin{aligned}
\dot{V}_{t_0}(x, \xi \quad) = \min \quad & \mathbb{E}\Big[\sum_{t=t_0}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \mid \boldsymbol{\xi}_{t_0} = \xi \quad \Big] \\
\text{s.t.} \quad & \boldsymbol{x}_{t_0-1} = x \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \geq t_0 \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) \preceq \sigma(\boldsymbol{\xi}_{[t]}) && \forall t \geq t_0
\end{aligned}
$$

Backward Bellman operators and Dynamic Programming

Define the cost-to-go, or value function

$$
V_{t_0}(x) = \mathbb{E}_{\boldsymbol{\xi}_{t_0}}\Big[\dot{V}_{t_0}(x, \boldsymbol{\xi}_{t_0}) = \min \quad \mathbb{E}\Big[\sum_{t=t_0}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \mid \boldsymbol{\xi}_{t_0} = \boldsymbol{\xi}_{t_0}\Big]\Big]
$$

$$
\begin{aligned}
\text{s.t.} \quad & \boldsymbol{x}_{t_0-1} = x \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) && \forall t \geq t_0 \\
& (\boldsymbol{x}_t, \boldsymbol{u}_t) \preceq \sigma(\boldsymbol{\xi}_{[t]}) && \forall t \geq t_0
\end{aligned}
$$

Backward Bellman operators and Dynamic Programming

Define the cost-to-go, or value function

$$V_{t_0}(x) = \mathbb{E}_{\xi_{t_0}}\Big[\dot{V}_{t_0}(x, \xi_{t_0}) = \min \quad \mathbb{E}\Big[\sum_{t=t_0}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \mid \boldsymbol{\xi}_{t_0} = \xi_{t_0}\Big]\Big]$$

$$\begin{aligned} \text{s.t.} \quad & \boldsymbol{x}_{t_0-1} = x \\ & (\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) & \forall t \geq t_0 \\ & (\boldsymbol{x}_t, \boldsymbol{u}_t) \preceq \sigma(\boldsymbol{\xi}_{[t]}) & \forall t \geq t_0 \end{aligned}$$

Assuming that $(\boldsymbol{\xi}_\tau)_{\tau \in [T]}$ is stagewise independent, we have

$$\begin{aligned} \dot{V}_t &= \dot{\mathcal{B}}_t(V_{t+1}) \\ V_t &= \mathcal{B}_t(V_{t+1}) := \mathbb{E}\big[\dot{V}_{t+1}(\cdot, \boldsymbol{\xi}_t)\big] \end{aligned}$$

Backward Bellman operators and Dynamic Programming

Define the cost-to-go, or value function

$$V_{t_0}(x) = \mathbb{E}_{\boldsymbol{\xi}_{t_0}}\Big[\dot V_{t_0}(x, \xi_{t_0}) = \min \quad \mathbb{E}\Big[\sum_{t=t_0}^{T} \ell_t(\boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\xi}_t) \mid \boldsymbol{\xi}_{t_0} = \xi_{t_0}\Big]\Big]$$

$$\text{s.t.} \quad \boldsymbol{x}_{t_0-1} = x$$

$$(\boldsymbol{x}_t, \boldsymbol{u}_t) \in \mathcal{X}_t(\boldsymbol{x}_{t-1}, \boldsymbol{\xi}_t) \qquad \forall t \geq t_0$$

$$(\boldsymbol{x}_t, \boldsymbol{u}_t) \preceq \sigma(\boldsymbol{\xi}_{[t]}) \qquad \forall t \geq t_0$$

Assuming that $(\boldsymbol{\xi}_\tau)_{\tau\in[T]}$ is stagewise independent, we have

$$\dot V_t = \dot{\mathcal{B}}_t(V_{t+1})$$

$$V_t = \mathcal{B}_t(V_{t+1}) := \mathbb{E}\big[\dot V_{t+1}(\cdot, \boldsymbol{\xi}_t)\big]$$

where the pointwise Backward Bellman operator $\dot{\mathcal{B}}_t$ is defined

$$\dot{\mathcal{B}}_t(\tilde V) := \begin{cases} \mathbb{R}^{n_t} \times \Xi_t & \to \mathbb{R} \cup \{+\infty\} \\ (x_{t-1}, \xi_t) & \mapsto \min\limits_{x_t, u_t \in \mathcal{X}_t(x_{t-1}, \xi_t)} \underbrace{\ell_{t+1}(x_{t-1}, x_t, u_t, \xi_t)}_{\text{transition costs}} + \underbrace{\tilde V(x_t)}_{\text{cost-to-go}} \end{cases}$$

Contents

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.

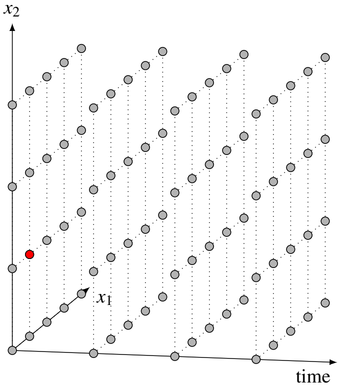
Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

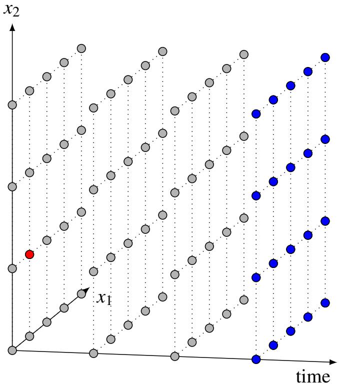
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

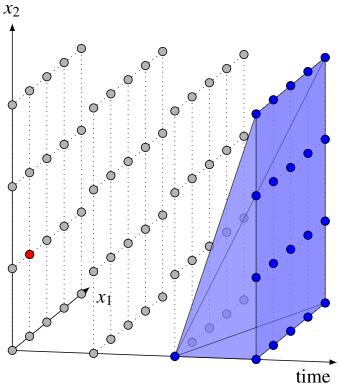
  for $x_{in} \in X^D_{t-1}$ **do**

    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

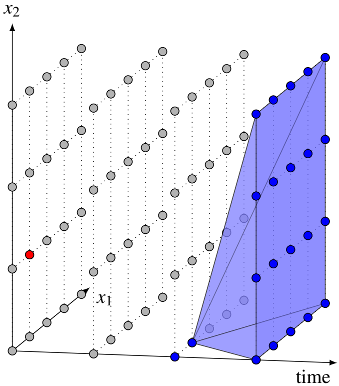


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

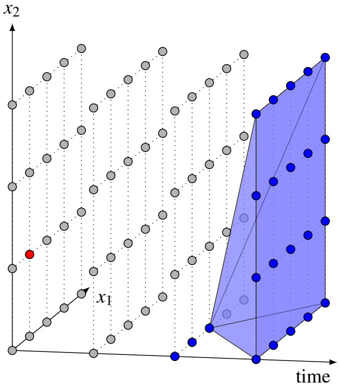
  for $x_{in} \in X^D_{t-1}$ **do**

    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

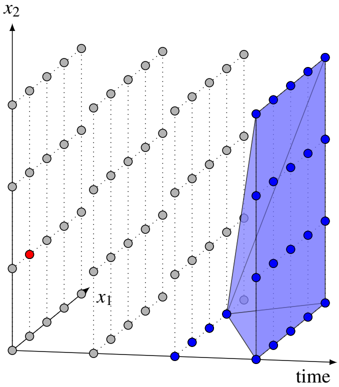


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

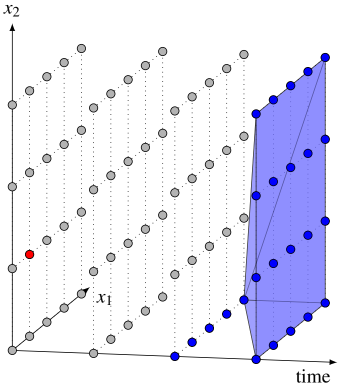
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

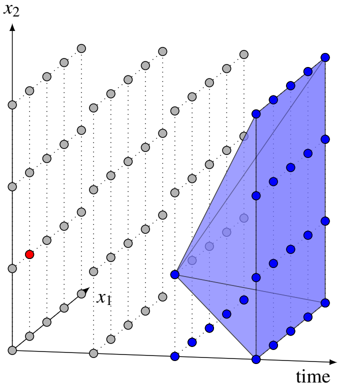
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

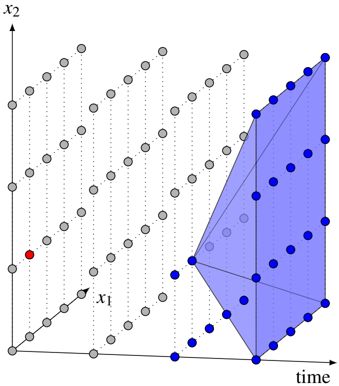
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

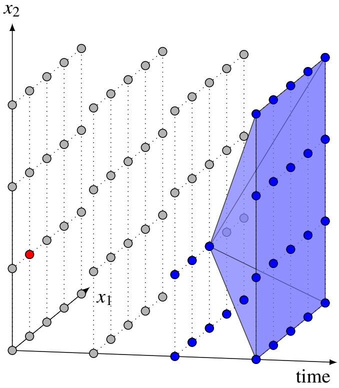
  for $x_{in} \in X^D_{t-1}$ **do**

    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

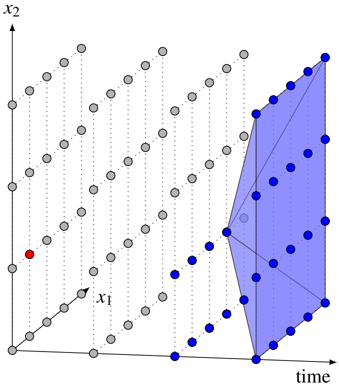


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

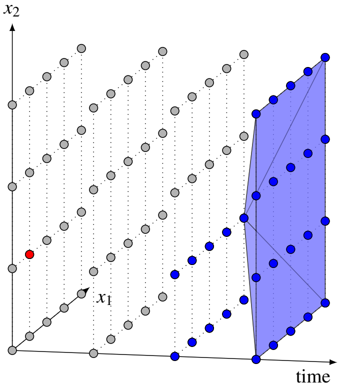
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

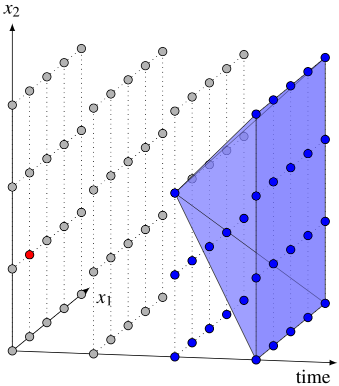
  for $x_{in} \in X^D_{t-1}$ **do**

    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

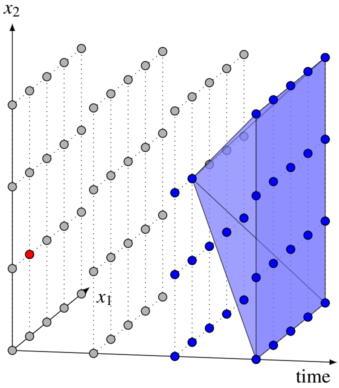


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

1 $\tilde{V}_t \equiv 0$

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

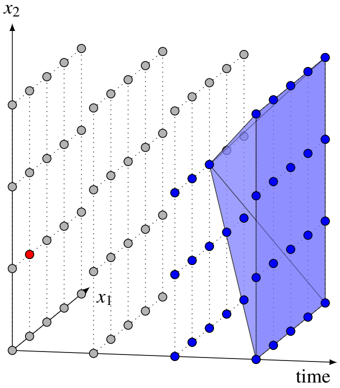
3   **for** $x_{in} \in X^D_{t-1}$ **do**

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

5 $$\dot{v}_\xi = \underbrace{\min_{x_{out} \in \mathcal{X}_t(x_{in},\xi)} \ell_t(x_{in}, x_{out}, \xi) + \tilde{V}_{t+1}(x_{out})}_{:=\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)}$$

6 $$\tilde{V}_t(x_{in}) \mathrel{+}= \underbrace{\pi_\xi}_{:=\mathbb{P}(\boldsymbol{\xi}_t=\xi)} \dot{v}_\xi$$

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

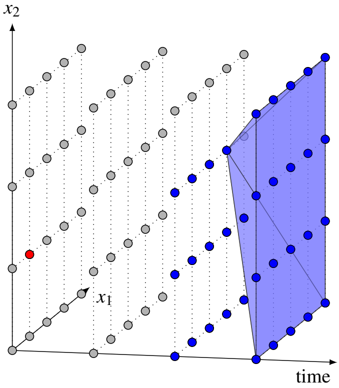


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

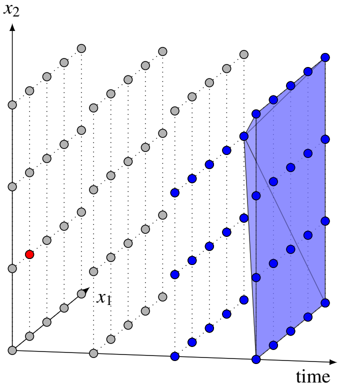
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

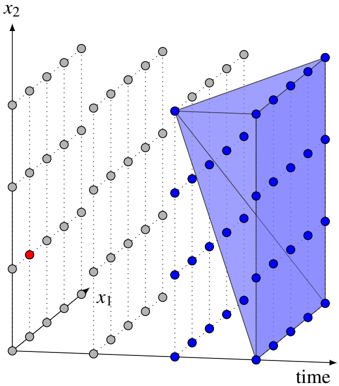
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

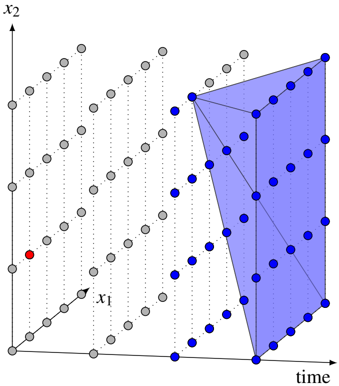
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

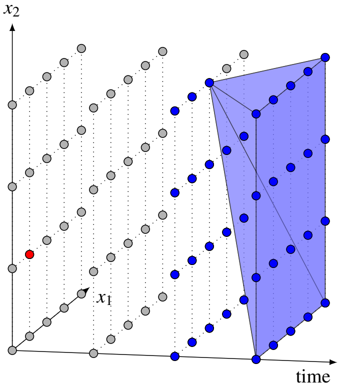
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

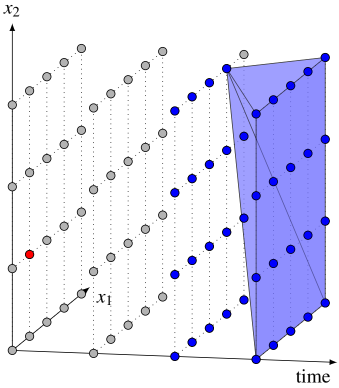
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

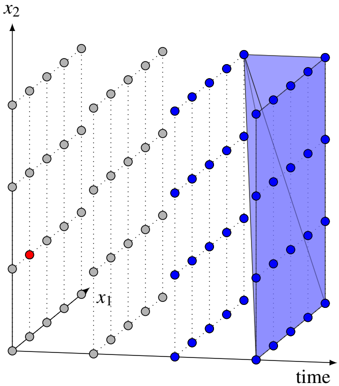
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

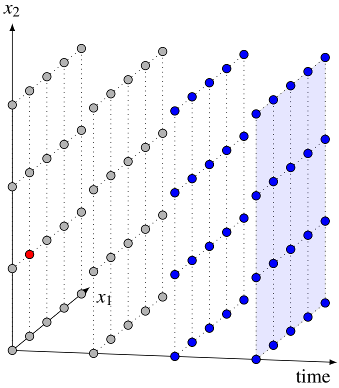
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

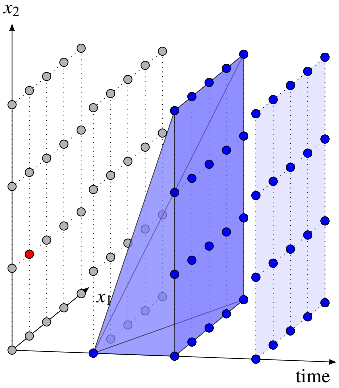
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

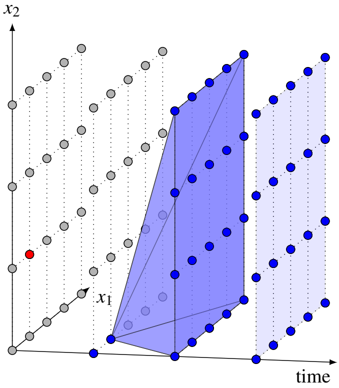
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

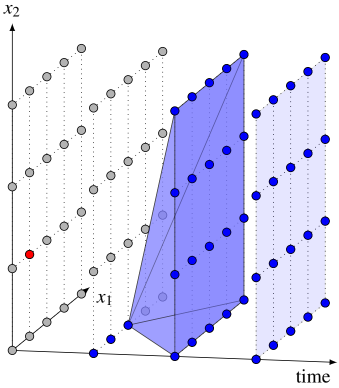
  for $x_{in} \in X^D_{t-1}$ **do**

    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

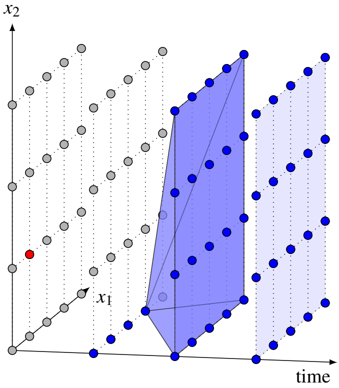


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

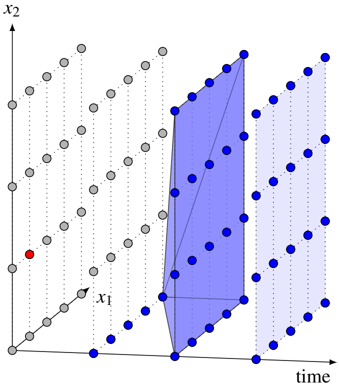
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

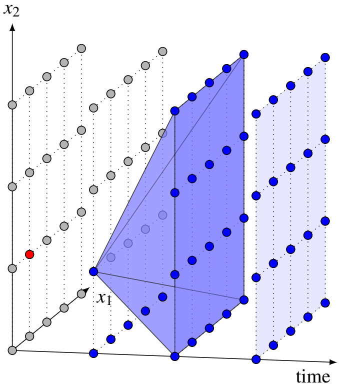
  for $x_{in} \in X^D_{t-1}$ **do**

    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

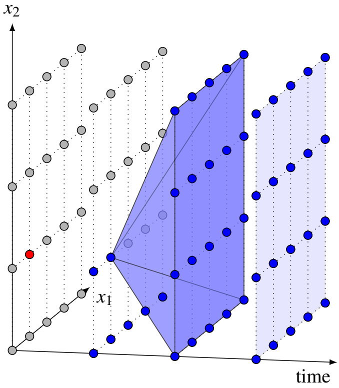


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

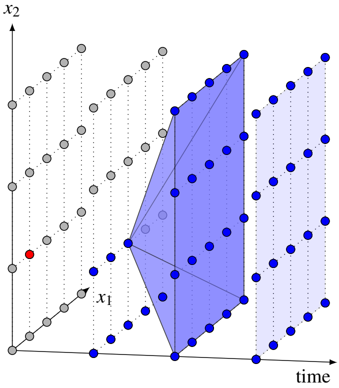
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

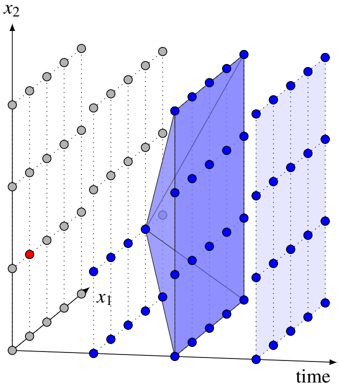
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

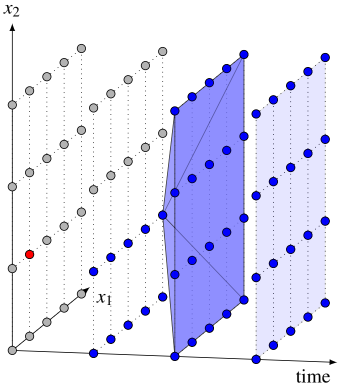
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

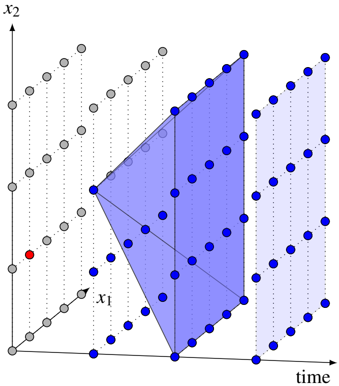
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

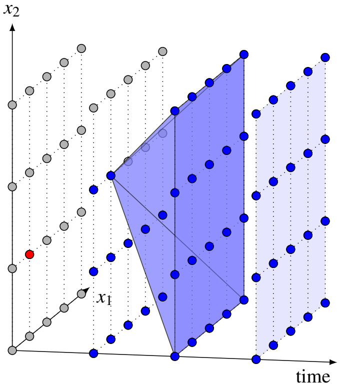
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

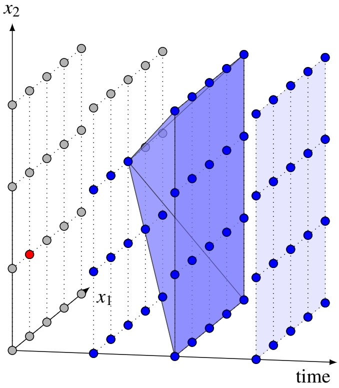
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

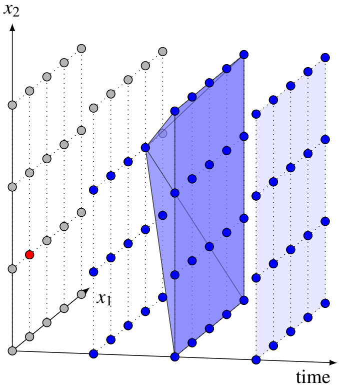
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

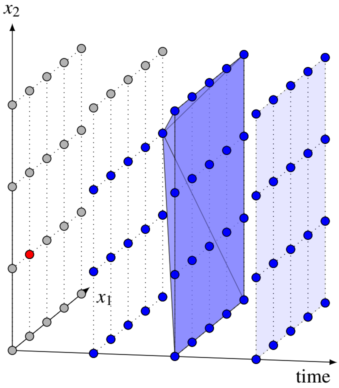
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

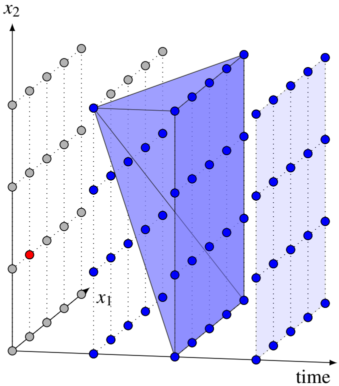
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

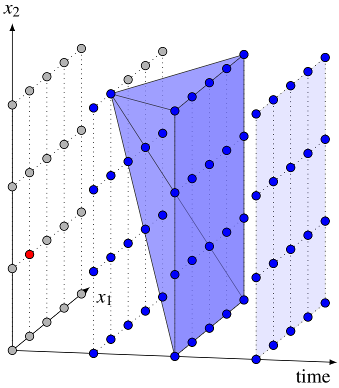
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

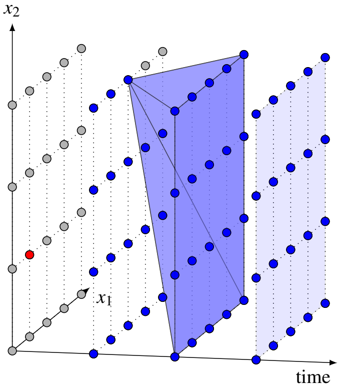
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

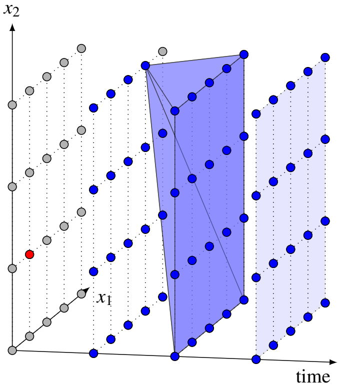
  for $x_{in} \in X_{t-1}^D$ **do**

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

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

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

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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

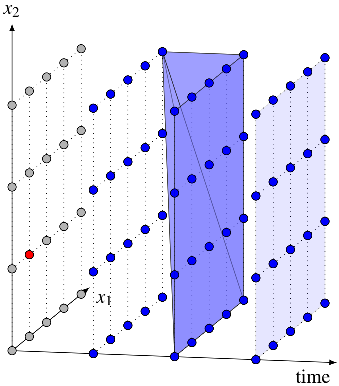
  for $x_{in} \in X^D_{t-1}$ **do**

    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

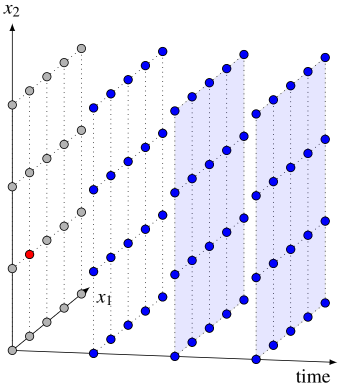


Discretized Stochastic Dynamic Programming

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

Algorithm 1: Discretized SDP

$\tilde V_t \equiv 0$
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



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.

Algorithm 1: Discretized SDP

$\tilde{V}_t \equiv 0$

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

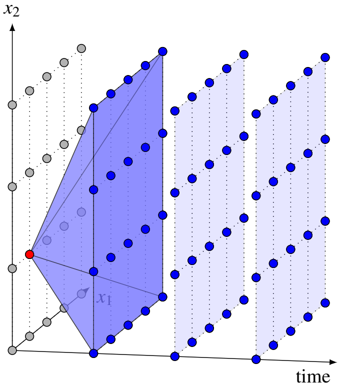
  for $x_{in} \in X^D_{t-1}$ **do**

    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



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[2] 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 $\dot{\mathcal{B}}_t(\tilde{V}_t)(x, \xi)$:

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

- A Forward Bellman operator $\mathcal{F}_t$ take as argument a cost-to-go approximation $\tilde{V}_t$ and return an optimal out-state[3] x_{out}.
- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.
- More precisely, given a scenario $(\check{\xi}_1, \dots, \check{\xi}_T)$, we have the following trajectory induced by $\tilde{V}_{[T]}$:

$$\check{x}_0 = x_0, \quad \check{x}_t = \mathcal{F}_t(\tilde{V}_t)(\check{x}_{t-1}, \check{\xi}_t)$$

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

[3] For technical reason, given the same $\tilde{V}$, $x_\in$ and ξ it should return the same x_{out}

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[2] 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 $\dot{\mathcal{B}}_t(\tilde{V}_t)(x, \xi)$:

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

- A Forward Bellman operator $\mathcal{F}_t$ take as argument a cost-to-go approximation $\tilde{V}_t$ and return an optimal out-state[3] x_{out}.
- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.
- More precisely, given a scenario $(\check{\xi}_1, \dots, \check{\xi}_T)$, we have the following trajectory induced by $\tilde{V}_{[T]}$:

$$\check{x}_0 = x_0, \quad \check{x}_t = \mathcal{F}_t(\tilde{V}_t)(\check{x}_{t-1}, \check{\xi}_t)$$

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

[3]For technical reason, given the same $\tilde{V}$, $x_{\in}$ and ξ it should return the same x_{out}

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[2] 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 $\dot{\mathcal{B}}_t(\tilde{V}_t)(x,\xi)$:

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

- A Forward Bellman operator $\mathcal{F}_t$ take as argument a cost-to-go approximation $\tilde{V}_t$ and return an optimal out-state[3] x_{out}.
- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.
- More precisely, given a scenario $(\check{\xi}_1, \dots, \check{\xi}_T)$, we have the following trajectory induced by $\tilde{V}_{[T]}$:

$$\check{x}_0 = x_0, \quad \check{x}_t = \mathcal{F}_t(\tilde{V}_t)(\check{x}_{t-1}, \check{\xi}_t)$$

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

[3] For technical reason, given the same $\tilde{V}$, $x_{\in}$ and ξ it should return the same x_{out}

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[2] 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 $\dot{\mathcal{B}}_t(\tilde{V}_t)(x, \xi)$:

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

- A Forward Bellman operator $\mathcal{F}_t$ take as argument a cost-to-go approximation $\tilde{V}_t$ and return an optimal out-state[3] x_{out}.
- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.
- More precisely, given a scenario $(\check{\xi}_1, \dots, \check{\xi}_T)$, we have the following trajectory induced by $\tilde{V}_{[T]}$:

$$\check{x}_0 = x_0, \quad \check{x}_t = \mathcal{F}_t(\tilde{V}_t)(\check{x}_{t-1}, \check{\xi}_t)$$

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

[3] For technical reason, given the same $\tilde{V}$, $x_\in$ and ξ it should return the same x_{out}

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[2] 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 $\dot{\mathcal{B}}_t(\tilde{V}_t)(x, \xi)$:

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

- A Forward Bellman operator $\mathcal{F}_t$ take as argument a cost-to-go approximation $\tilde{V}_t$ and return an optimal out-state[3] x_{out}.
- Thus a (sequence of) value functions approximations yields a policy, which can be simulated to obtain trajectories and costs.
- More precisely, given a scenario $(\check{\xi}_1, \dots, \check{\xi}_T)$, we have the following trajectory induced by $\tilde{V}_{[T]}$:

$$\check{x}_0 = x_0, \quad \check{x}_t = \mathcal{F}_t(\tilde{V}_t)(\check{x}_{t-1}, \check{\xi}_t)$$

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

[3] For technical reason, given the same $\tilde{V}$, $x_{\in}$ and ξ it should return the same x_{out}

Trajectory Following Dynamic Programming algorithms

TFDP algorithms iteratively refine outer-approximations of the cost-to-go functions:

1. using the current outer-approximation we compute a trajectory ($\rightsquigarrow$ forward phase)
2. around the computed trajectory we refine the outer-approximations ($\rightsquigarrow$ backward phase)

A few comments:

- The forward phase depends on two elements:
  - the chosen forward operator $\mathcal{F}_t$
  - the node-selection ξ_t^k method
- Outer approximations are defined as maximum of elementary functions called cuts.

Trajectory Following Dynamic Programming algorithms

TFDP algorithms iteratively refine outer-approximations of the cost-to-go functions:

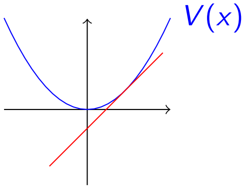
1. using the current outer-approximation we compute a trajectory ($\rightsquigarrow$ forward phase)
2. around the computed trajectory we refine the outer-approximations ($\rightsquigarrow$ backward phase)

A few comments:

- The forward phase depends on two elements:
  - the chosen forward operator $\mathcal{F}_t$
  - the node-selection ξ_t^k method
- Outer approximations are defined as maximum of elementary functions called cuts.

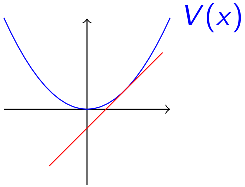
Example of cuts

1. Affine Benders cut
2. Affine Lagrangian cuts
3. Affine integer cuts

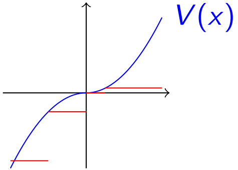


Example of cuts

1. Affine Benders cut
2. Affine Lagrangian cuts
3. Affine integer cuts

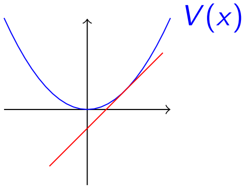


4. Step cuts

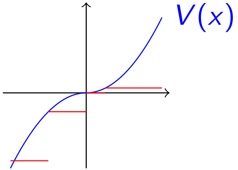


Example of cuts

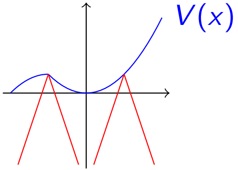
1. Affine Benders cut
2. Affine Lagrangian cuts
3. Affine integer cuts



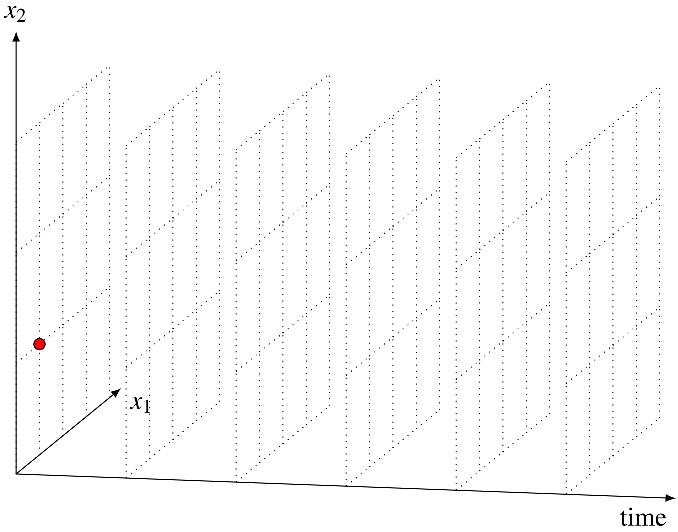
4. Step cuts



5. Lipschitz-cuts

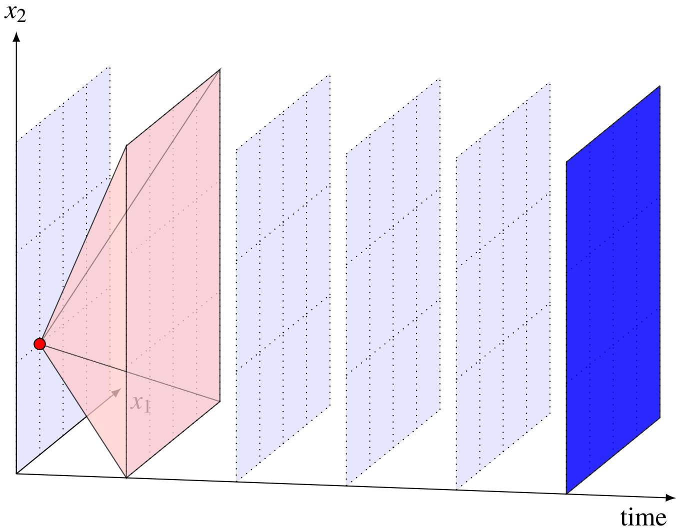


Trajectory Following Dynamic Programming



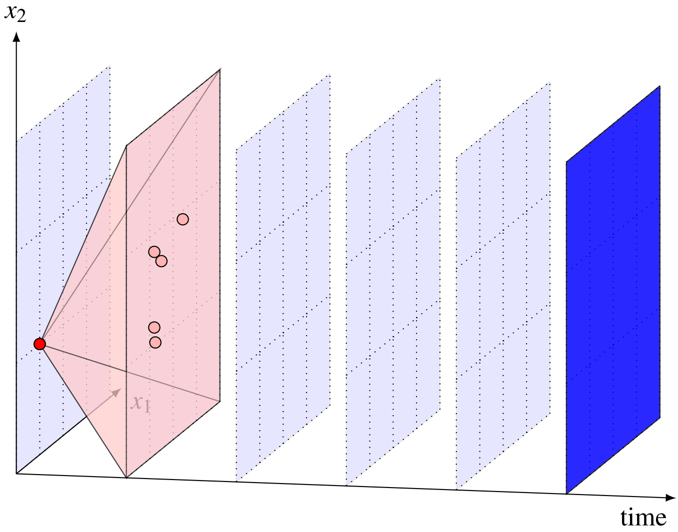
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



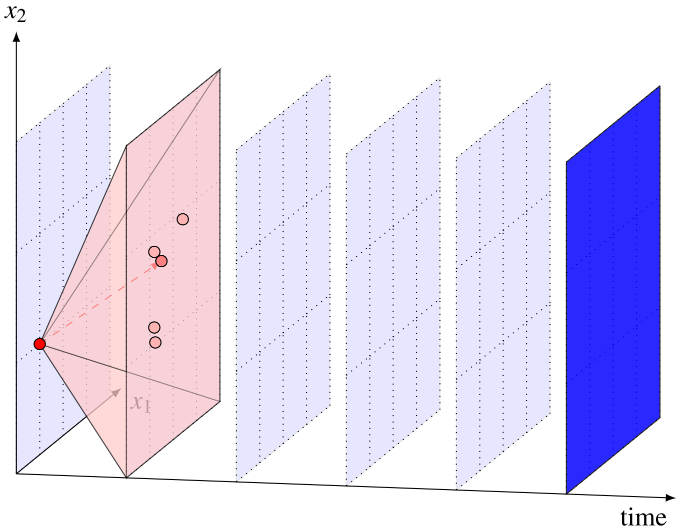
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



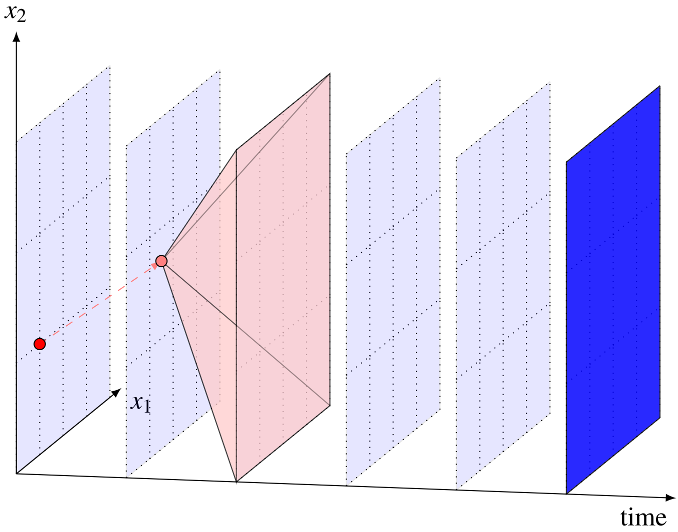
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



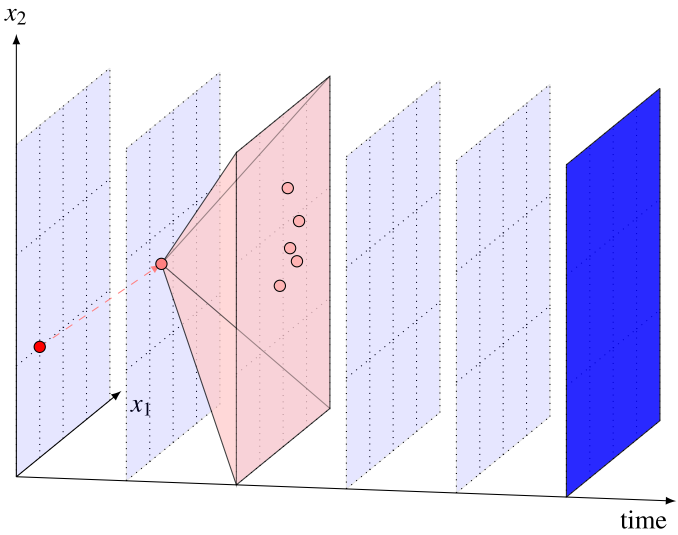
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



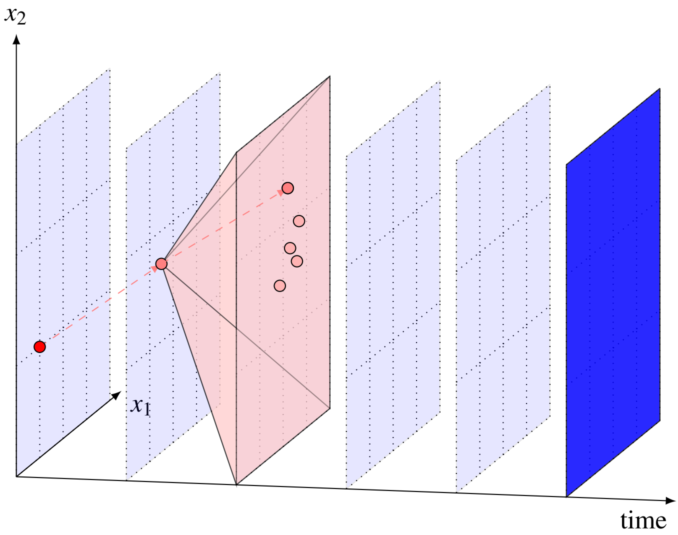
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



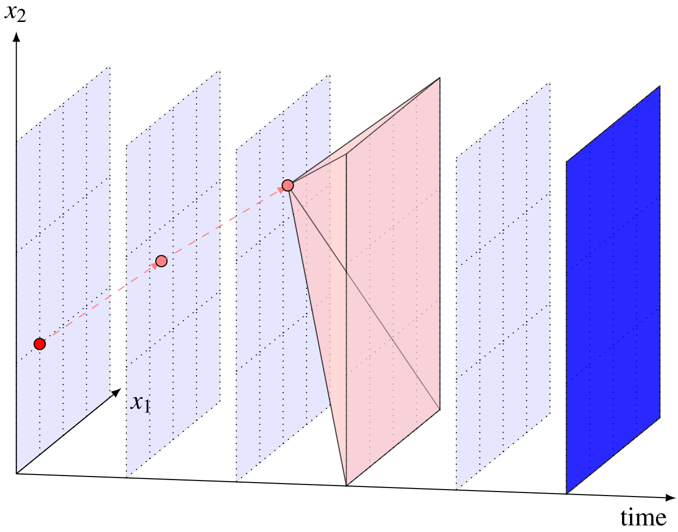
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



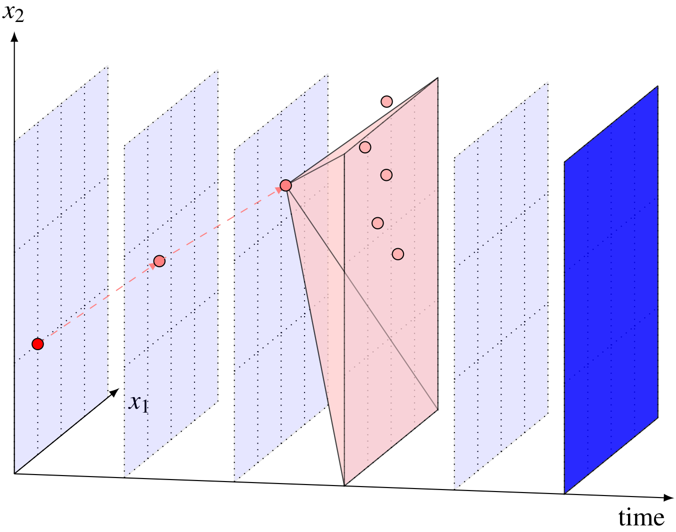
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



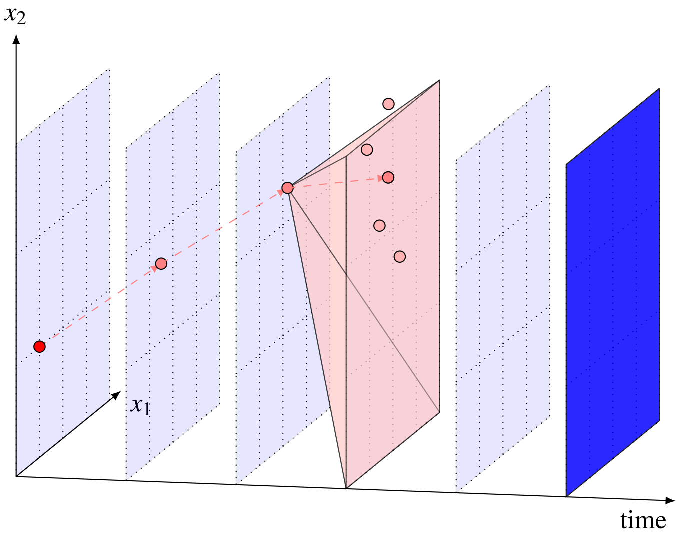
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



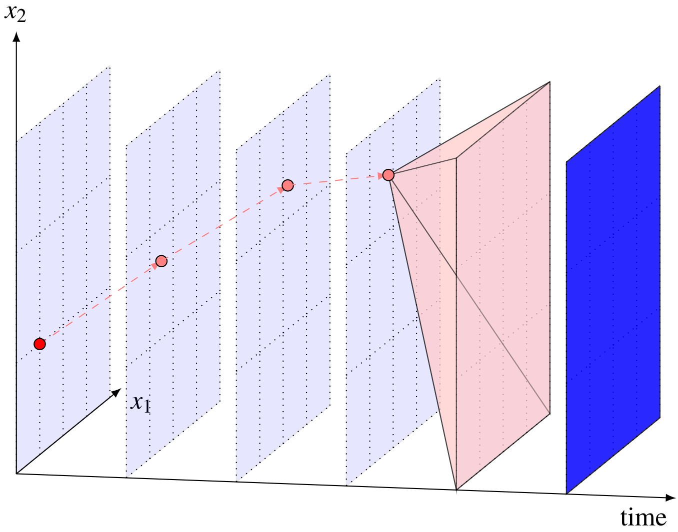
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



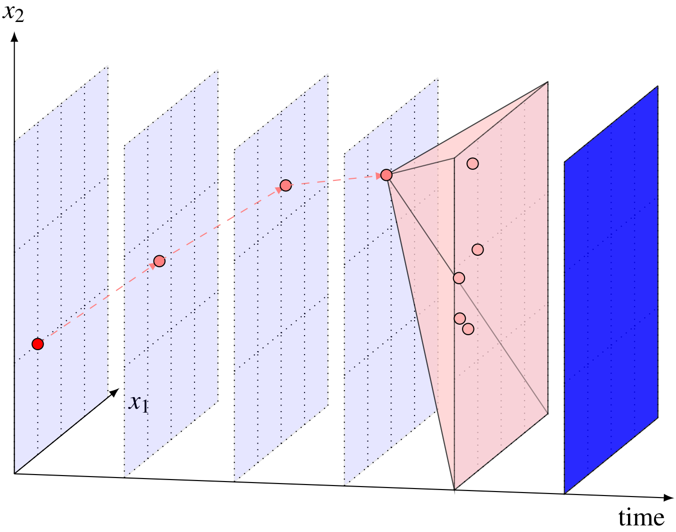
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



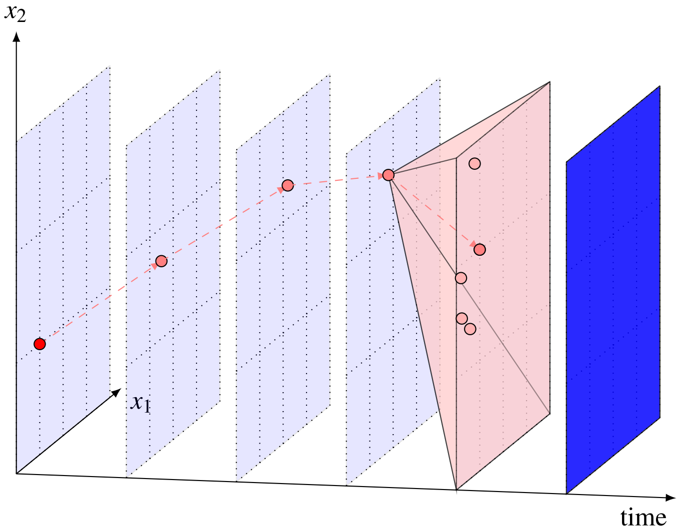
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



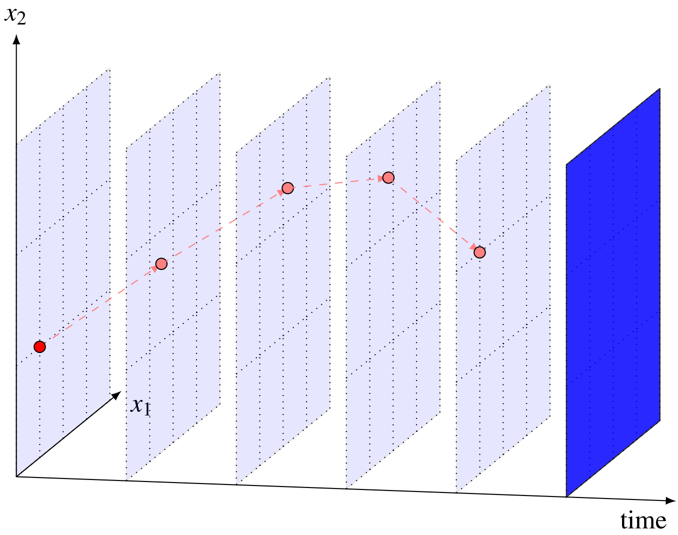
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



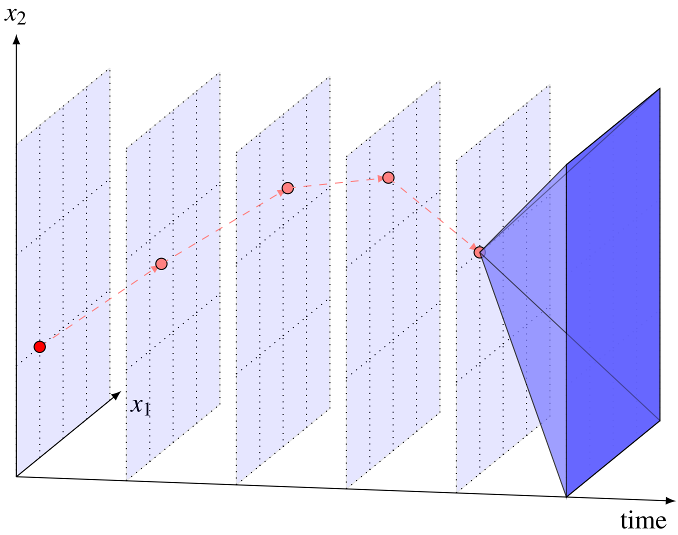
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



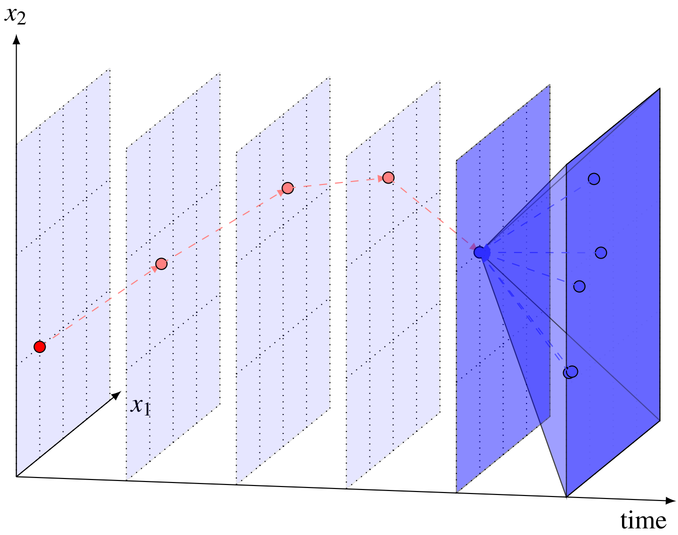
First forward pass : computing trajectory

Trajectory Following Dynamic Programming



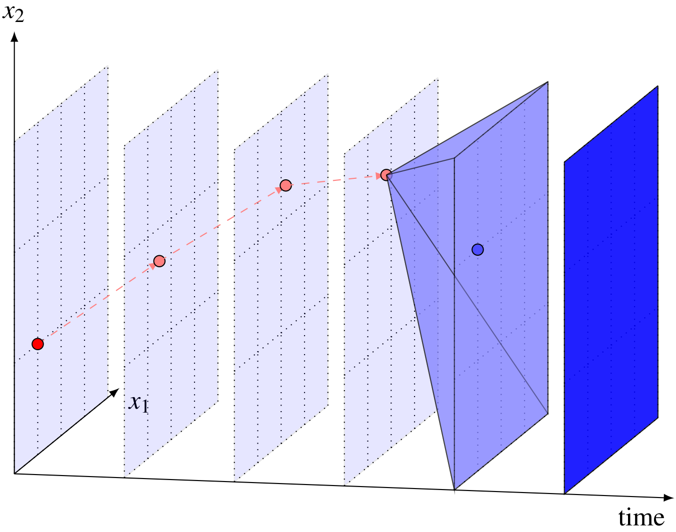
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



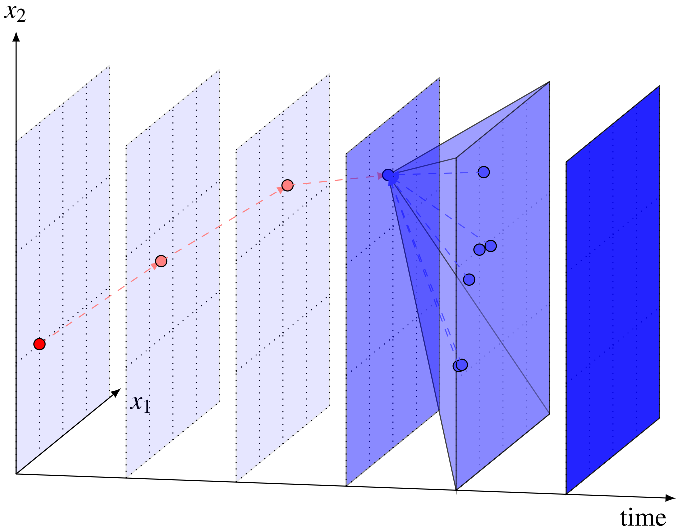
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



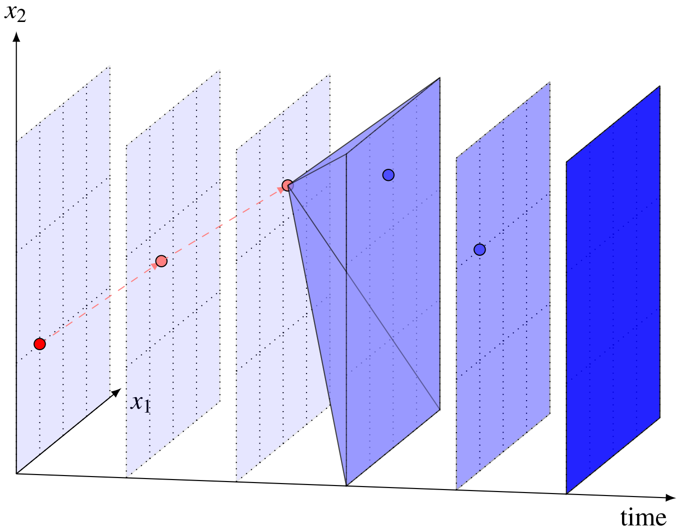
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



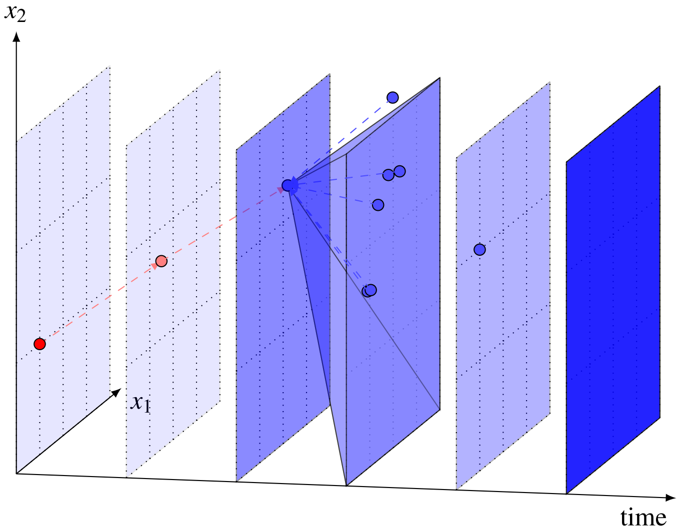
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



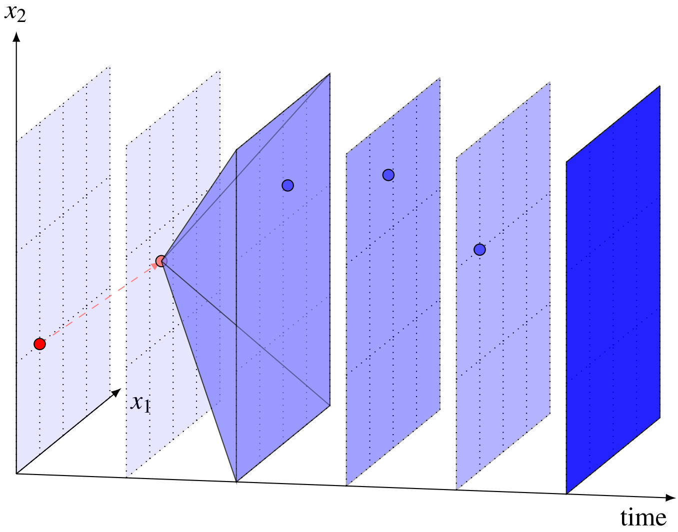
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



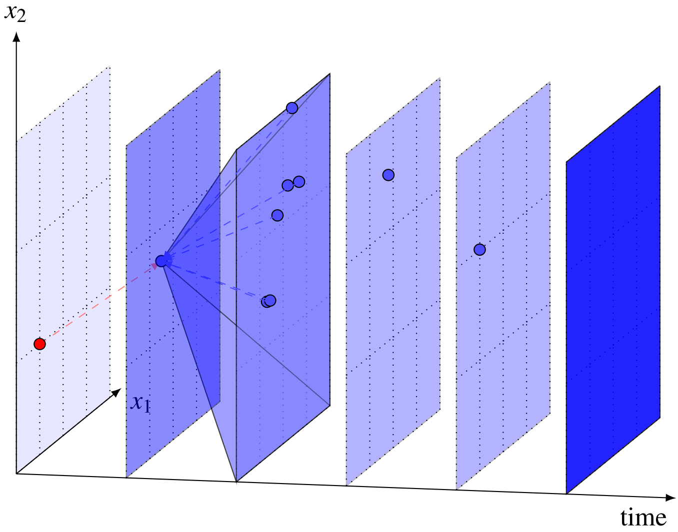
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



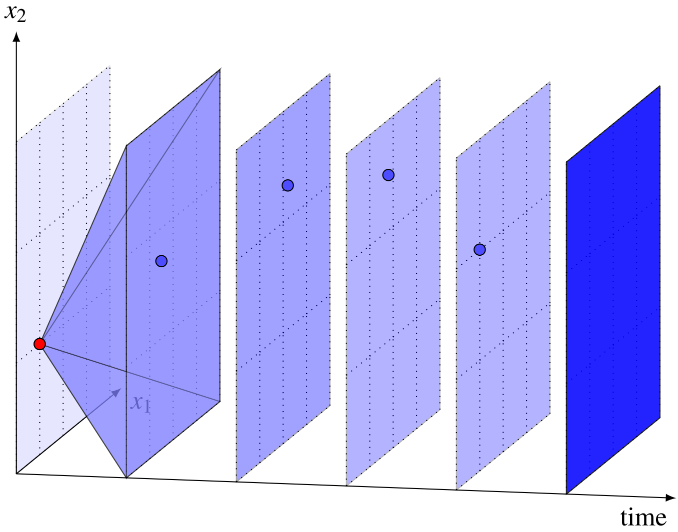
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



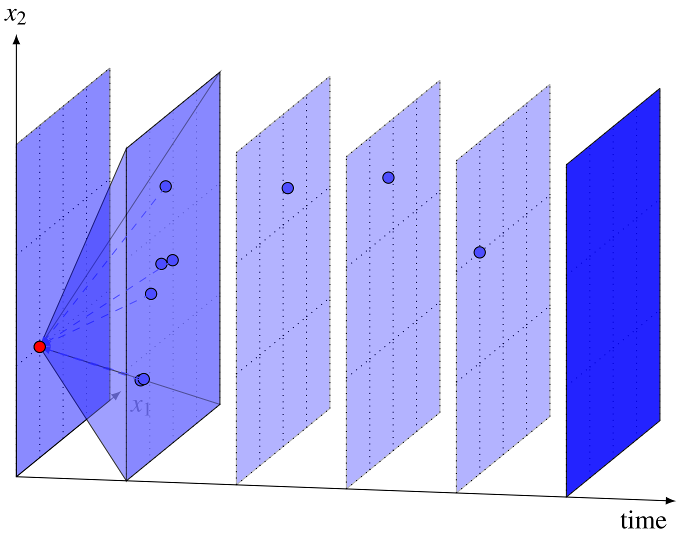
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



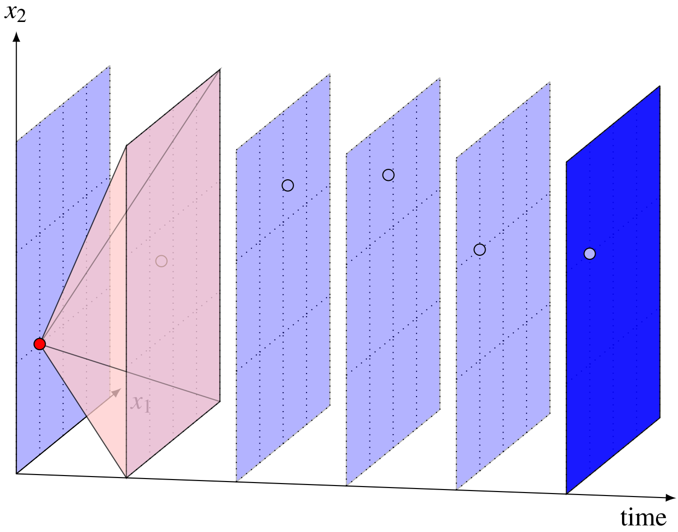
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



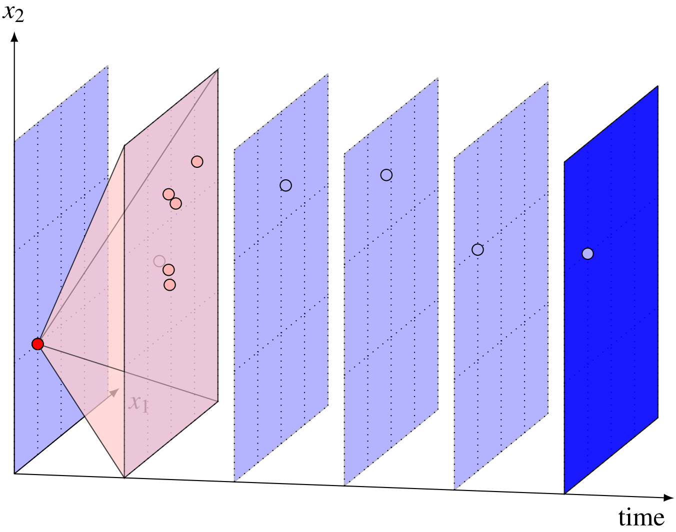
First backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



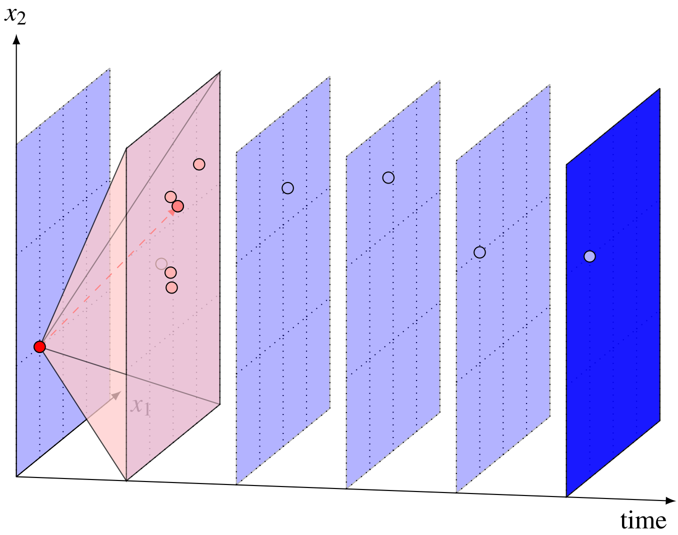
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



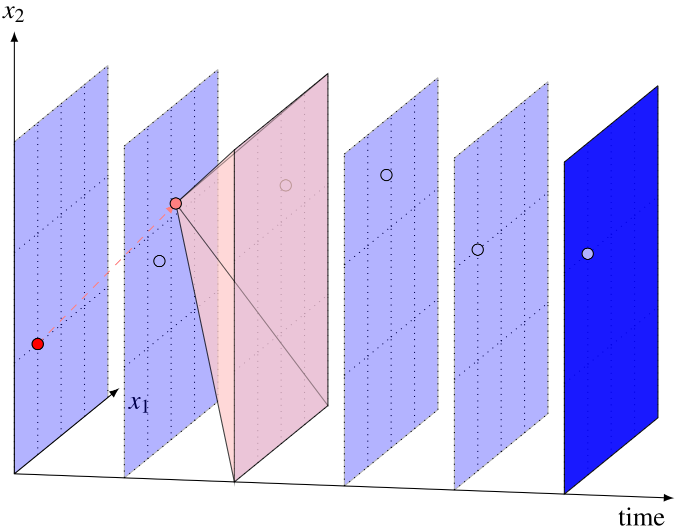
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



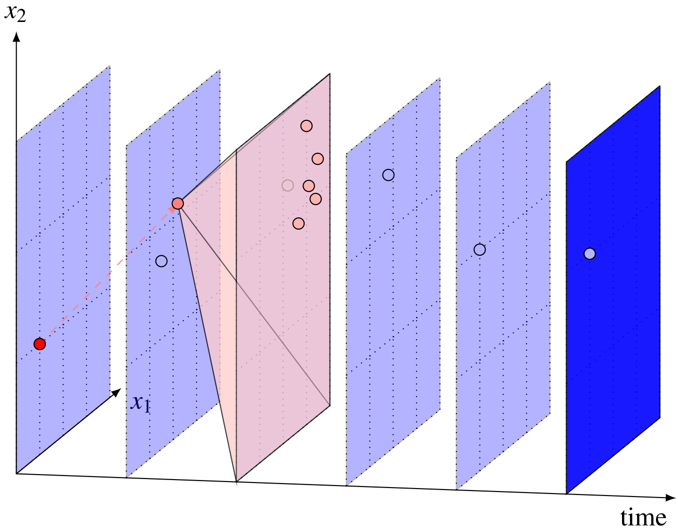
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



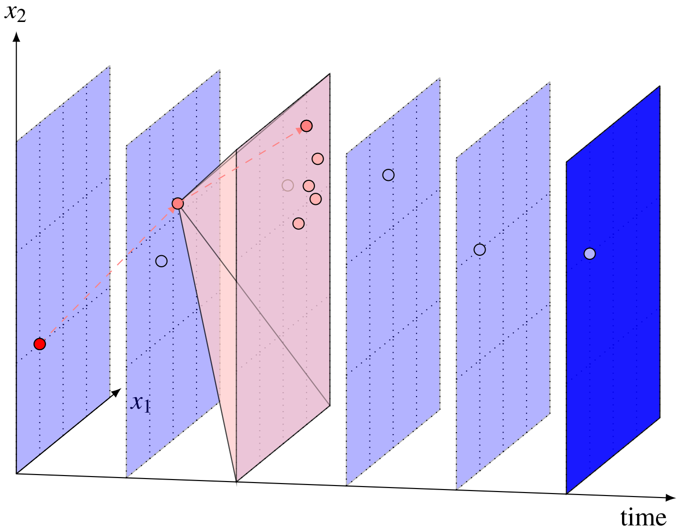
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



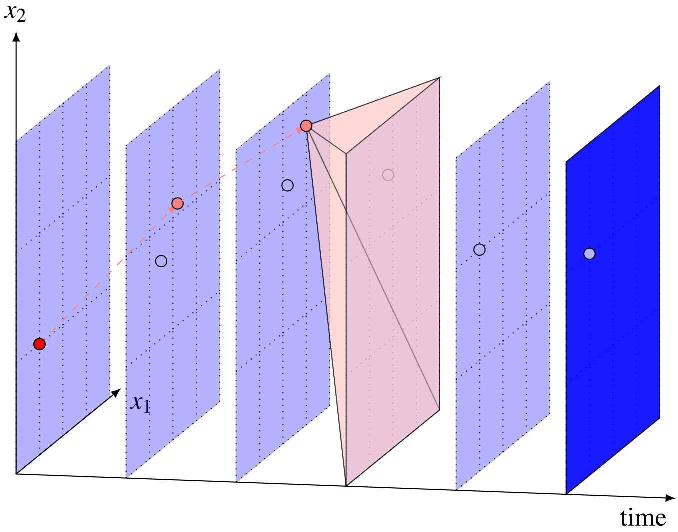
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



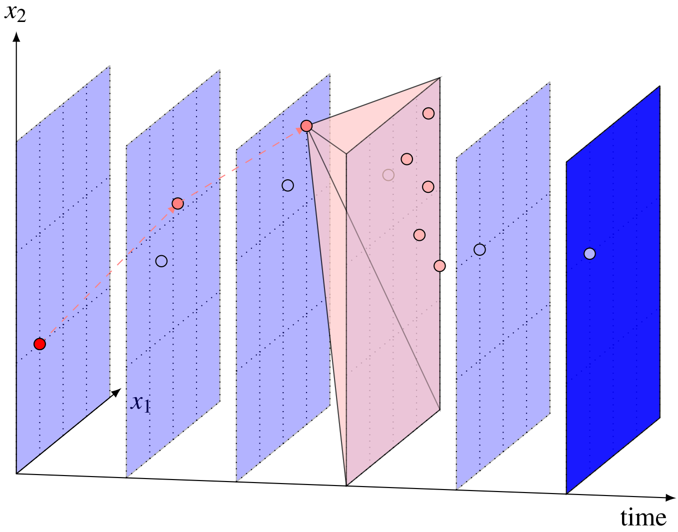
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



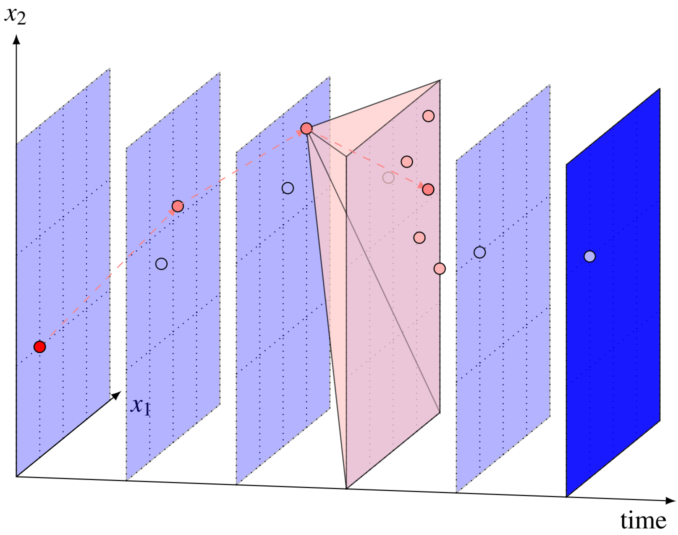
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



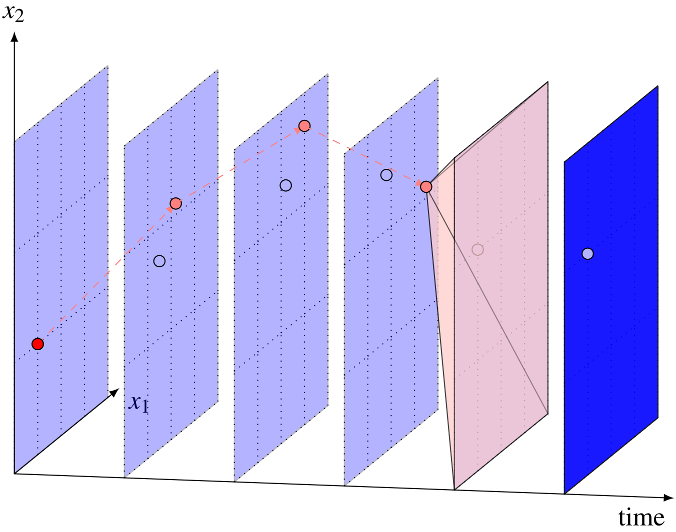
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



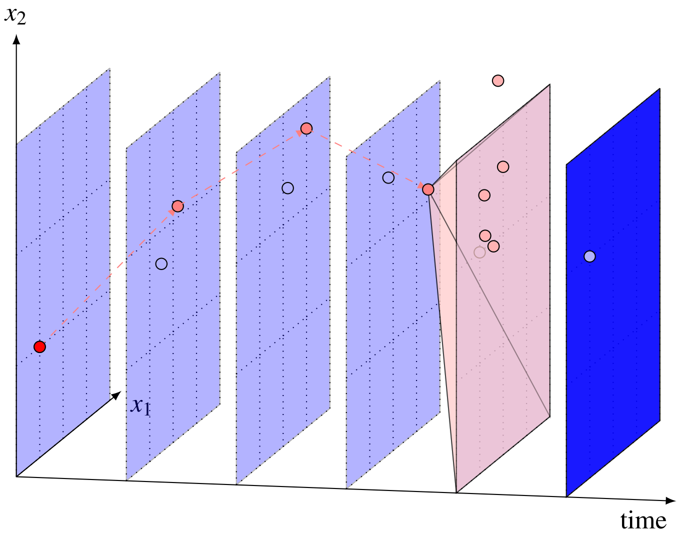
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



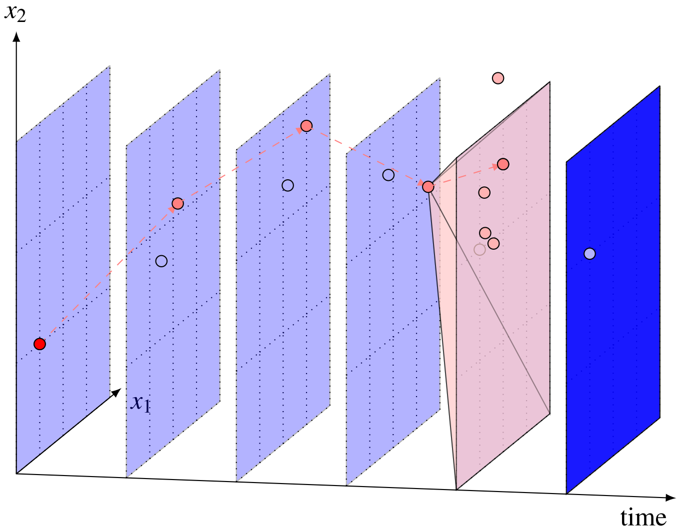
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



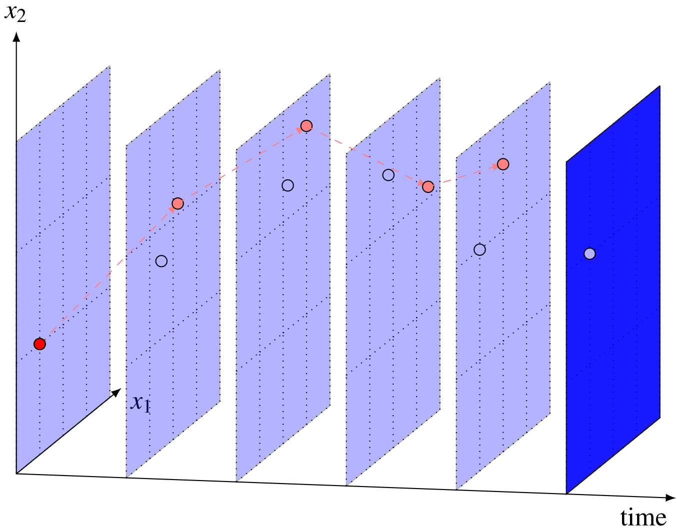
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



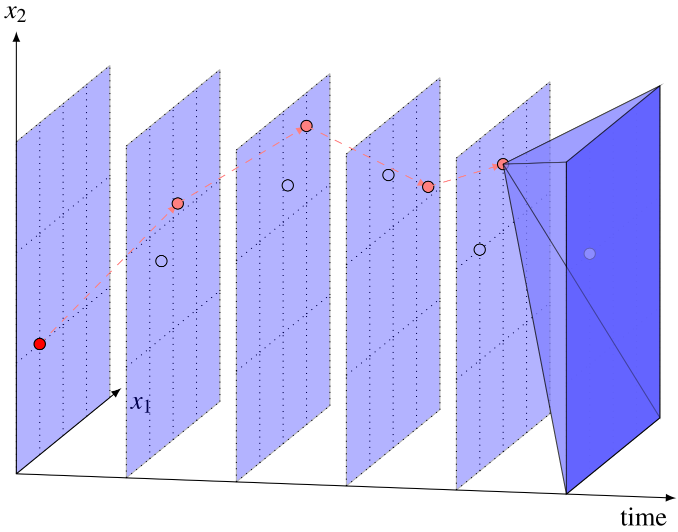
second forward pass : computing trajectory

Trajectory Following Dynamic Programming



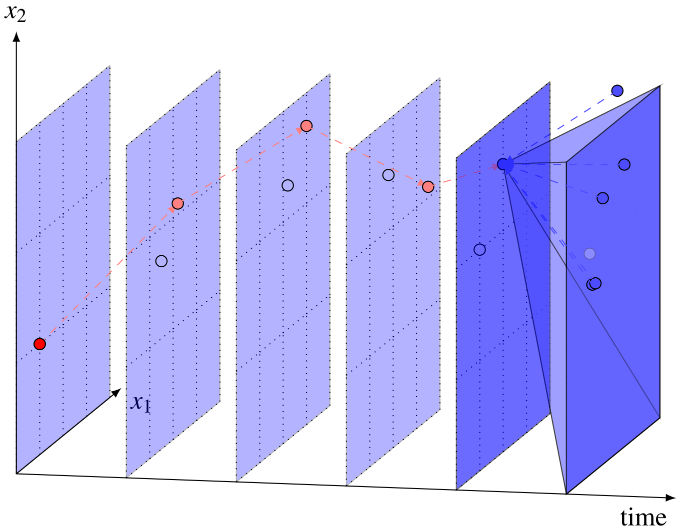
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



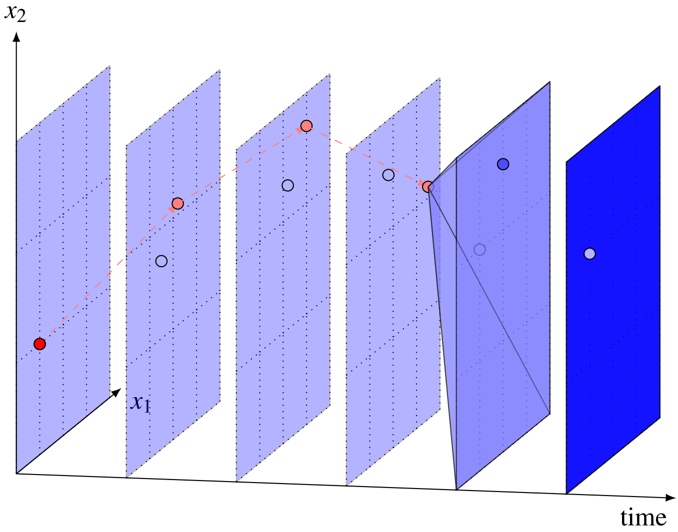
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



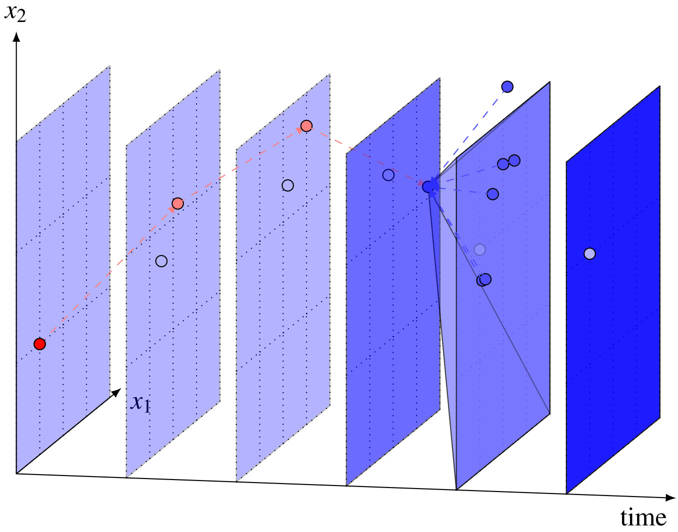
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



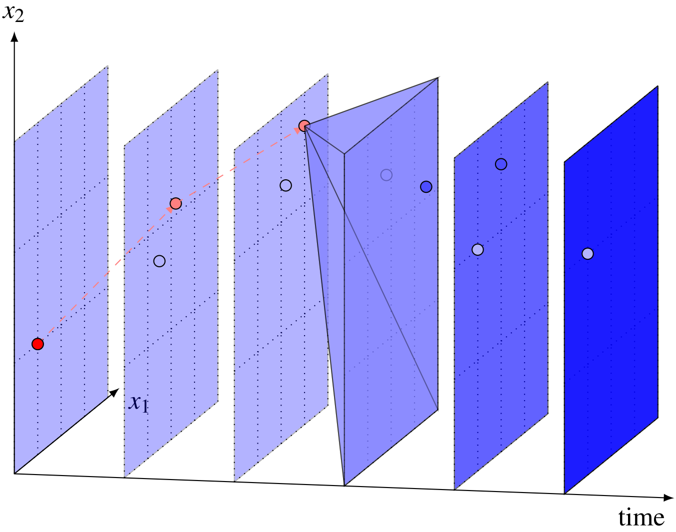
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



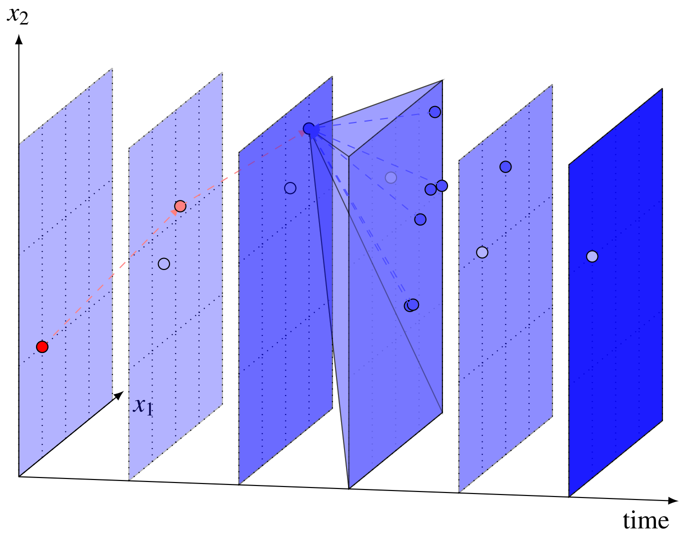
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



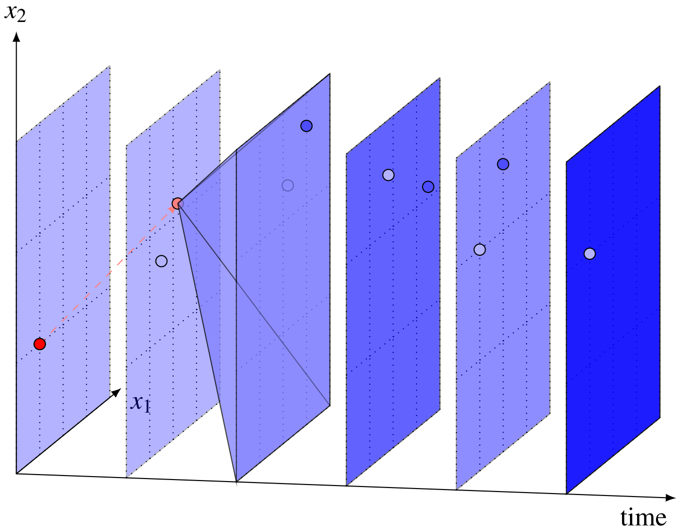
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



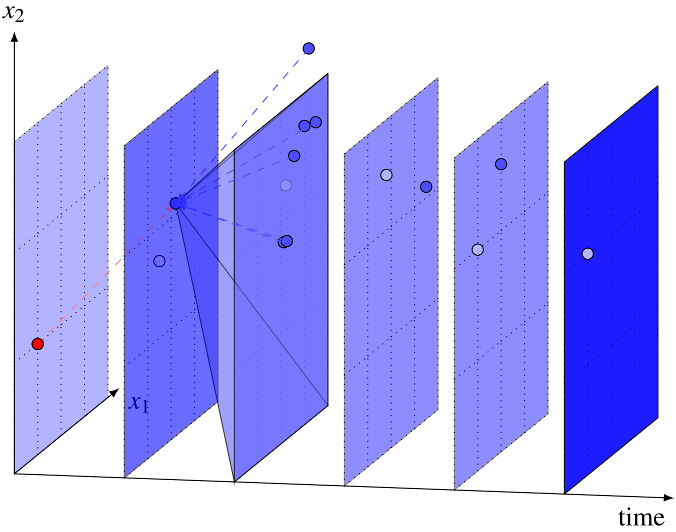
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



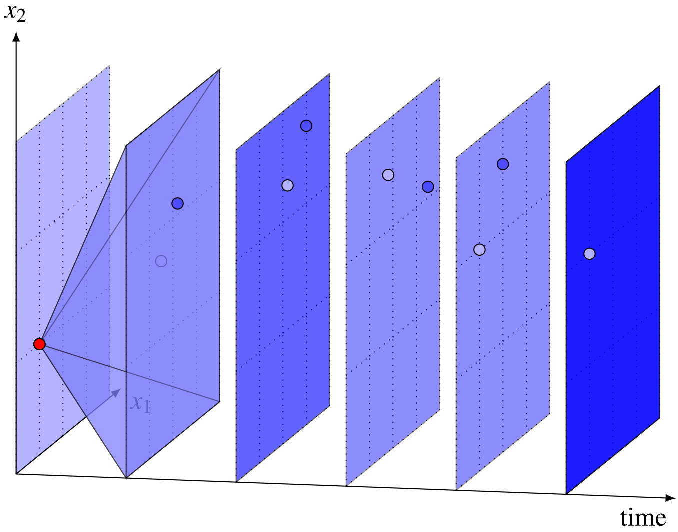
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



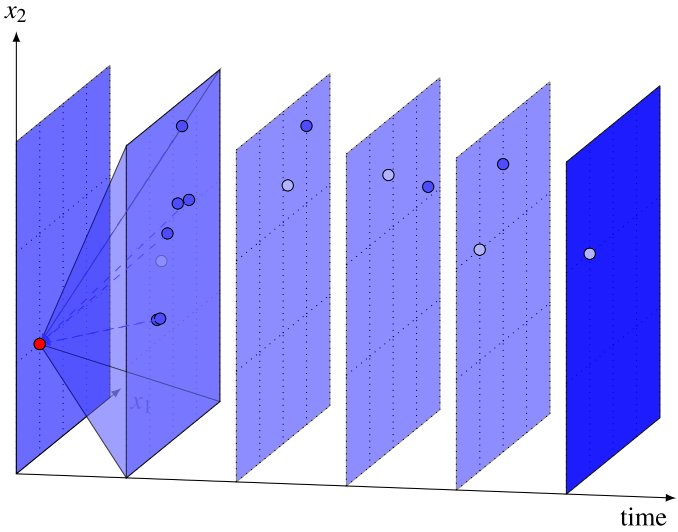
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



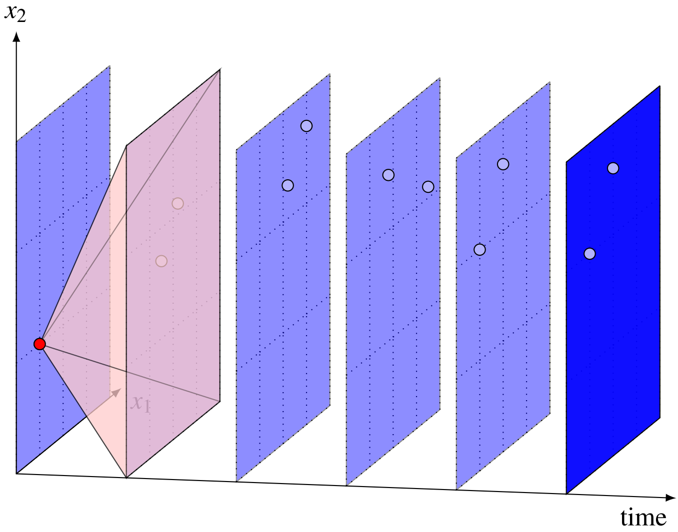
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



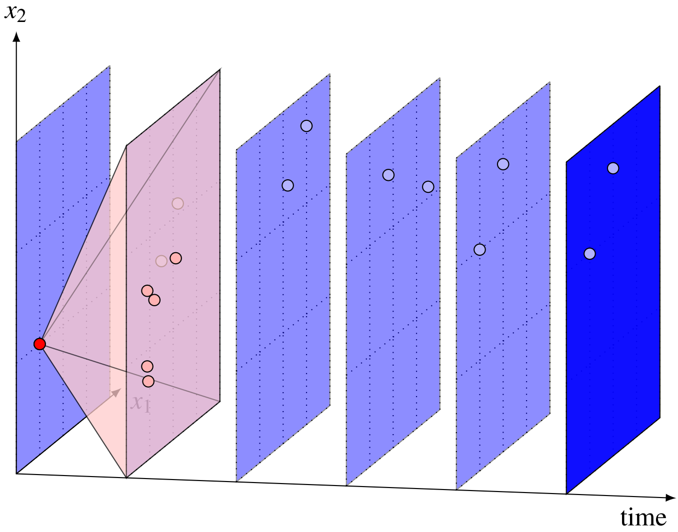
second backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



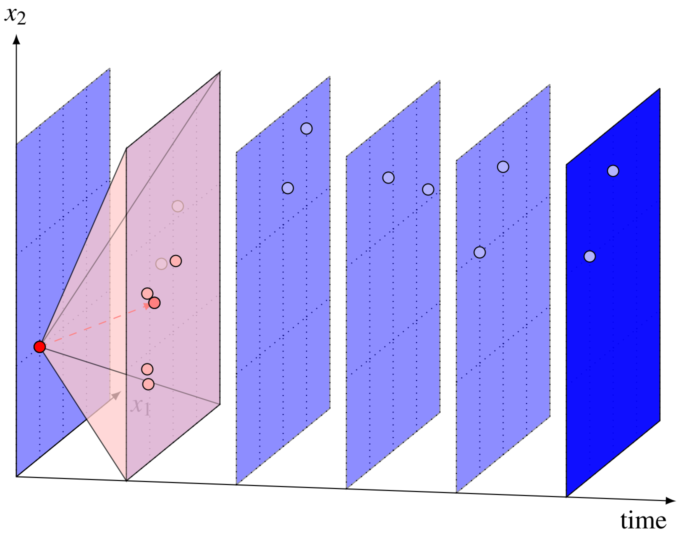
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



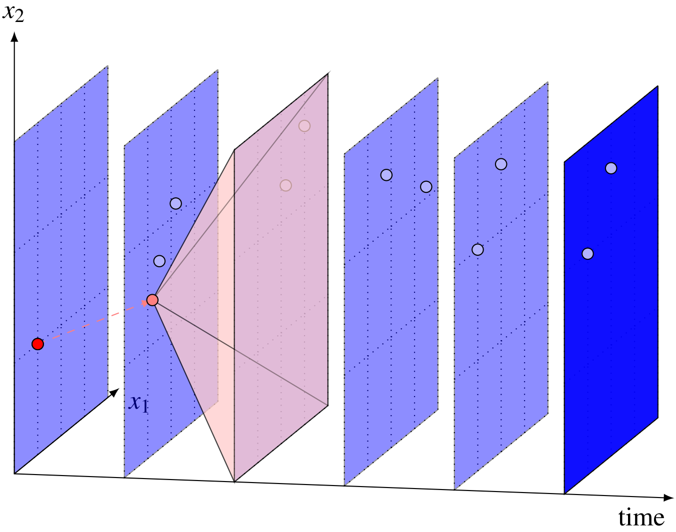
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



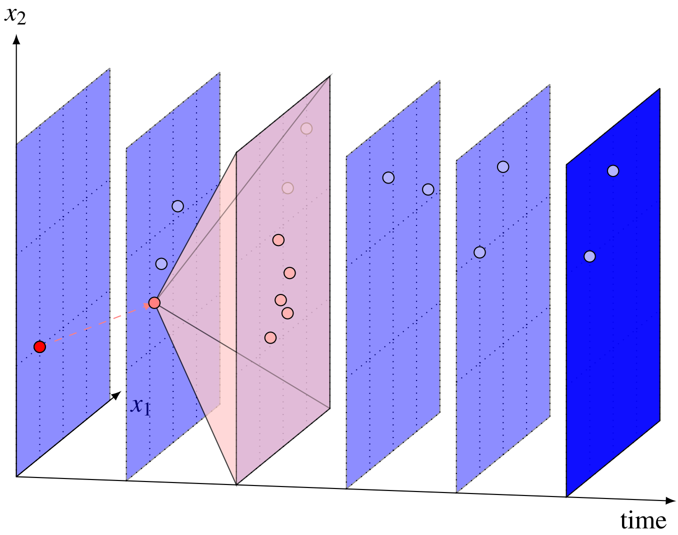
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



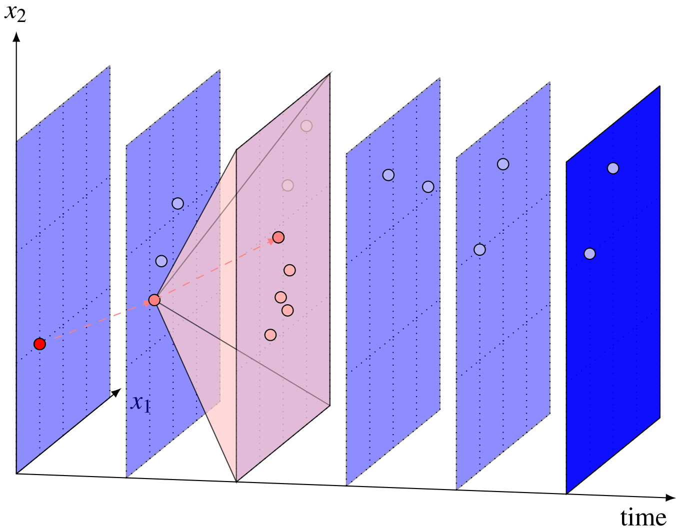
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



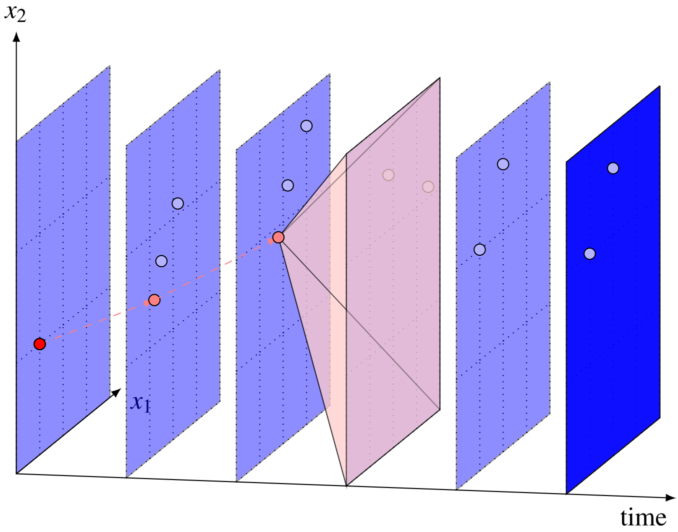
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



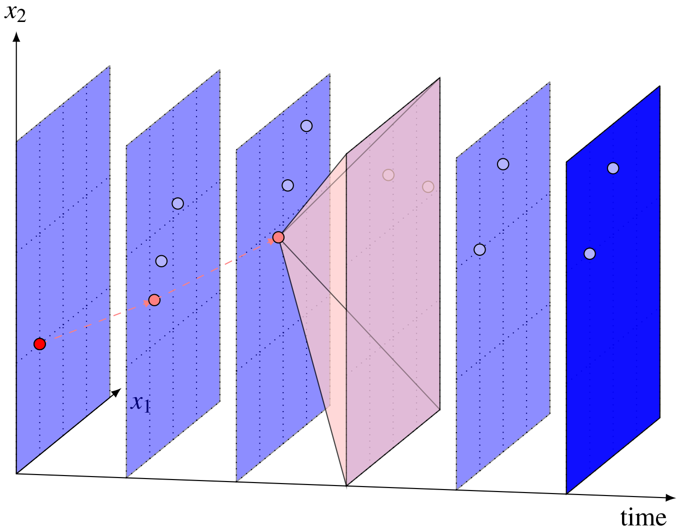
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



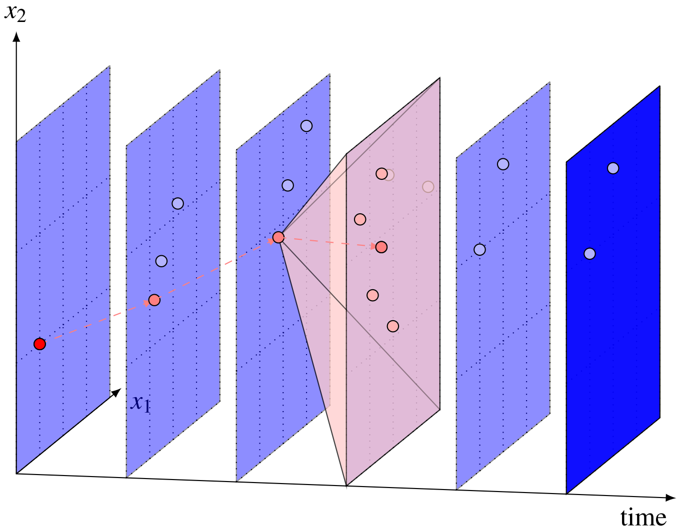
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



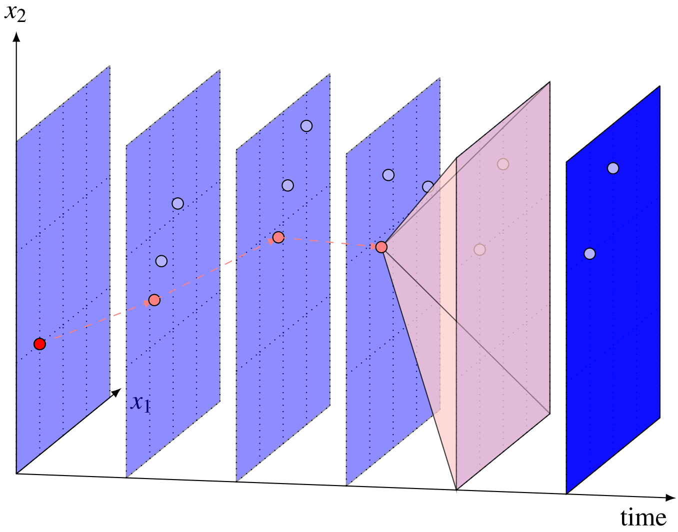
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



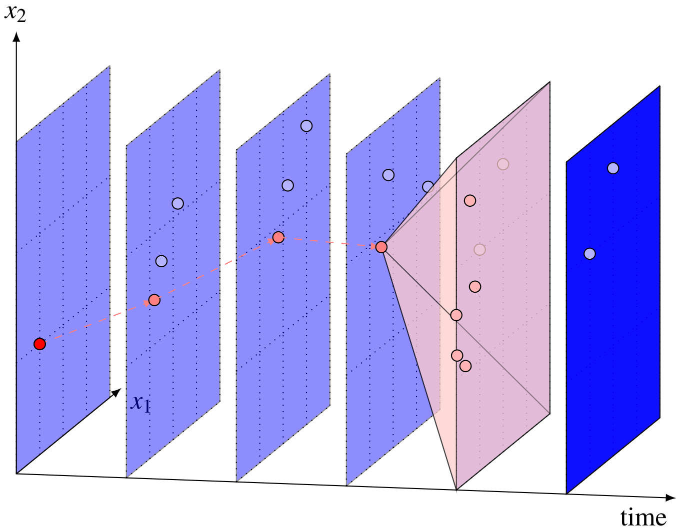
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



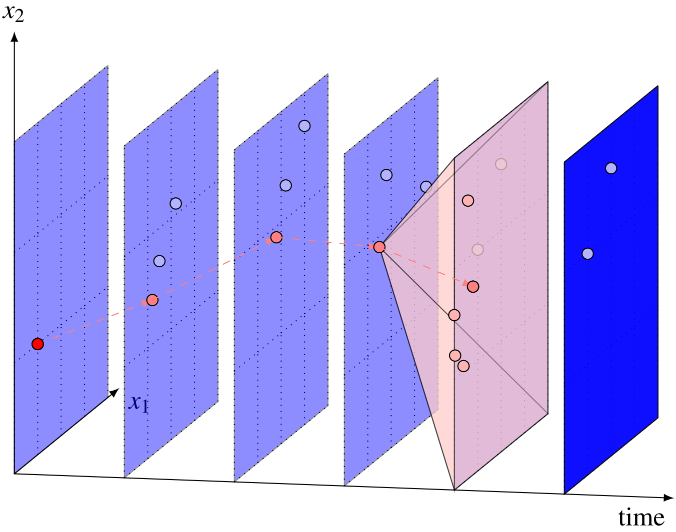
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



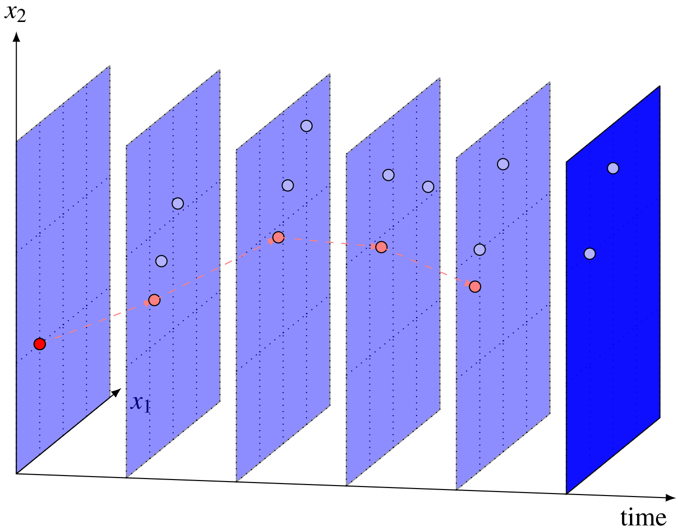
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



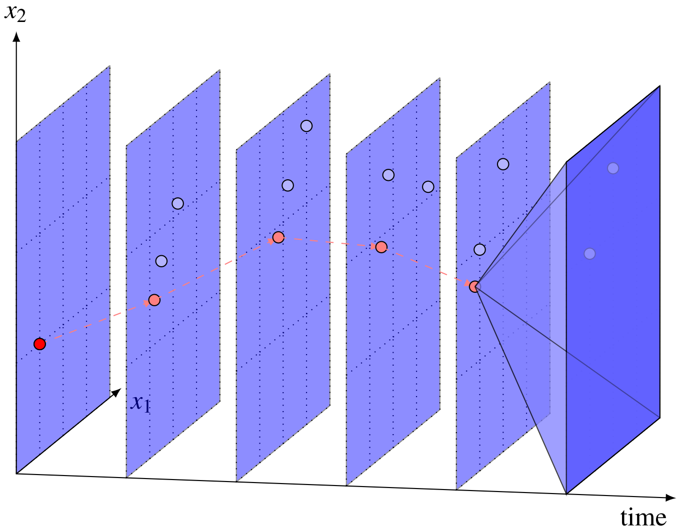
third forward pass : computing trajectory

Trajectory Following Dynamic Programming



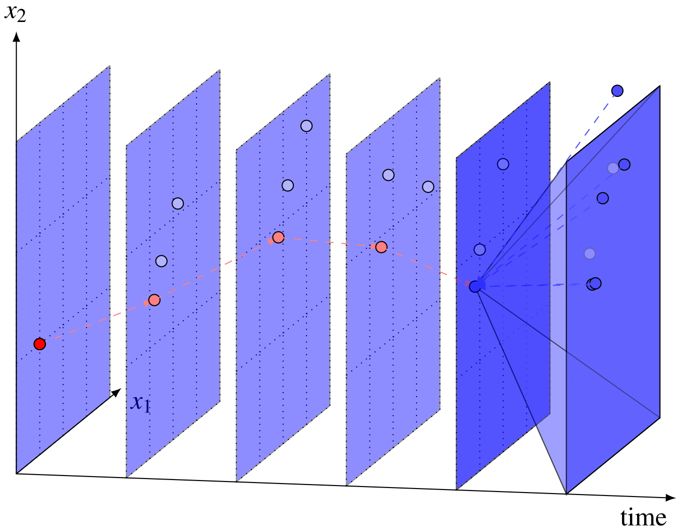
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



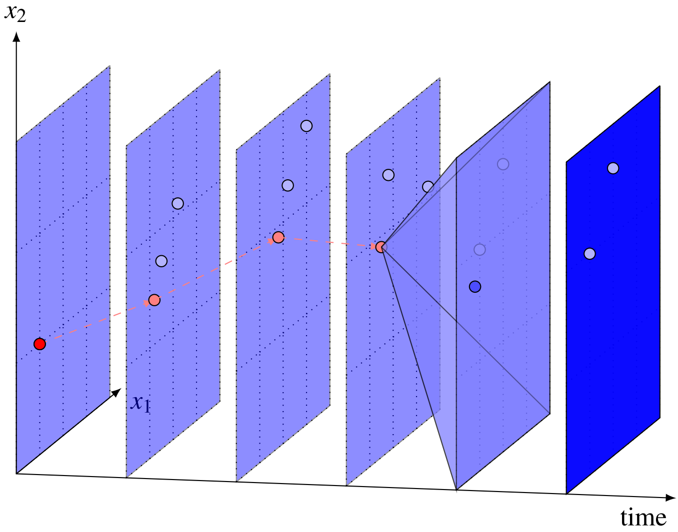
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



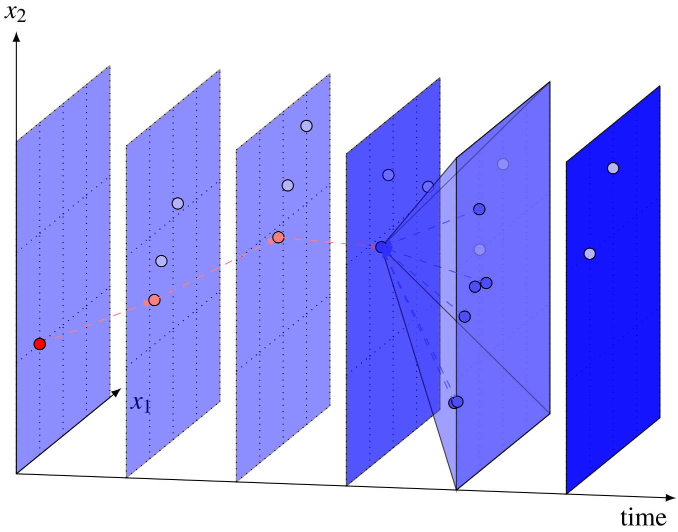
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



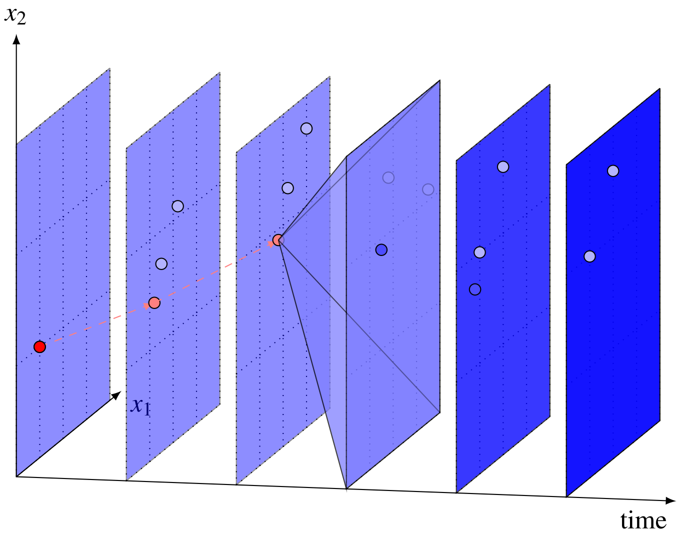
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



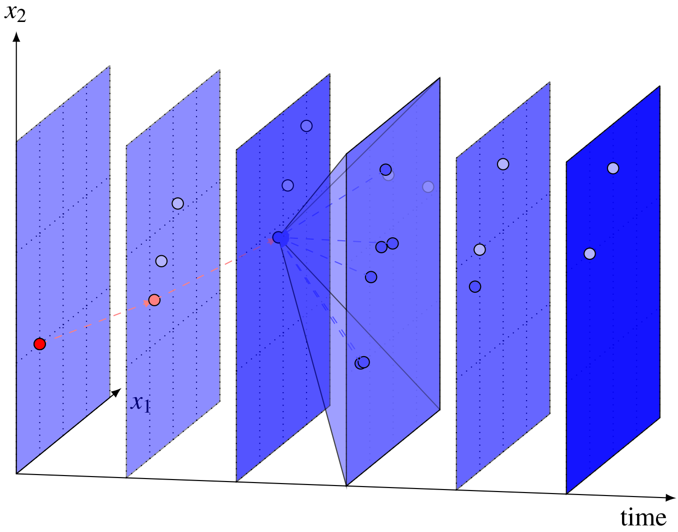
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



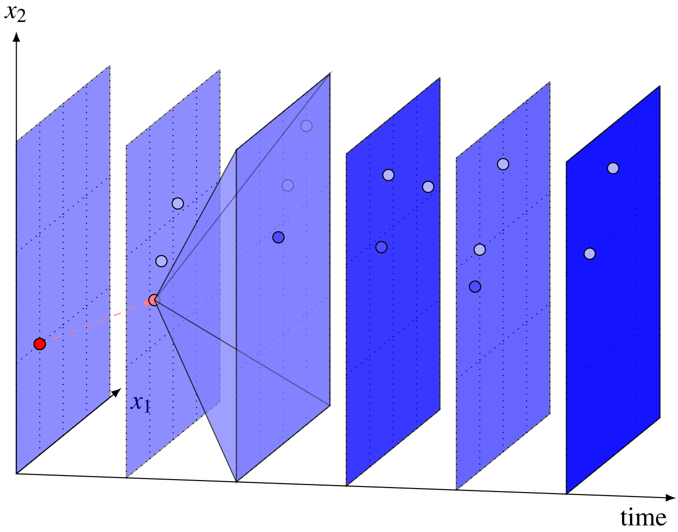
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



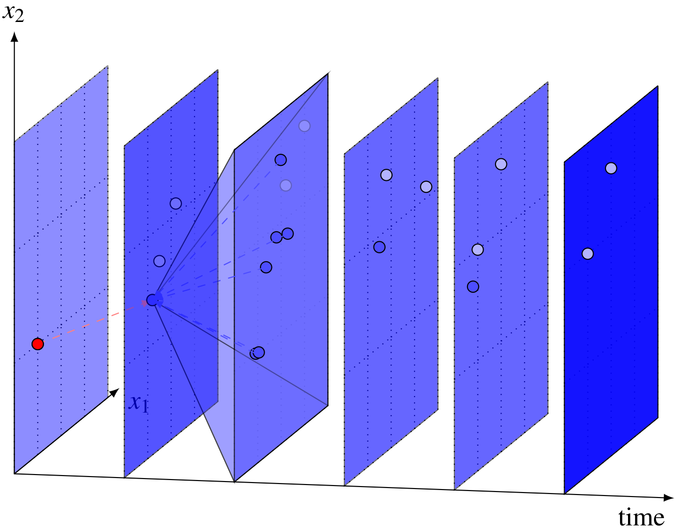
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



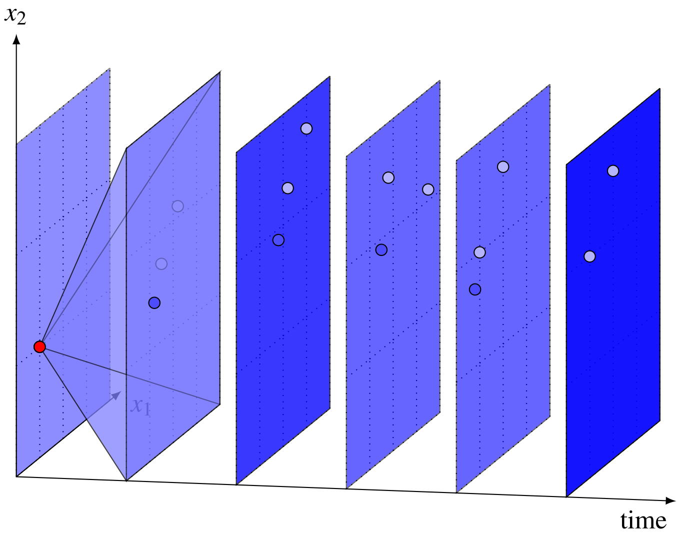
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



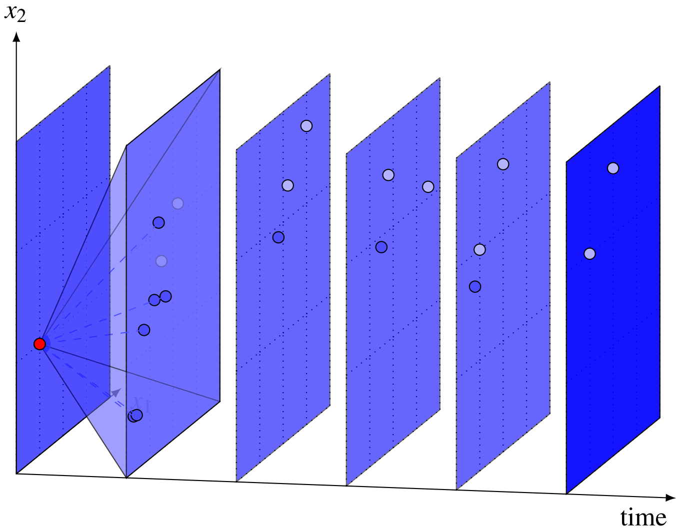
third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



third backward pass : refining approximation (adding cuts)

Trajectory Following Dynamic Programming



And so on...

Algorithm 2: A general framework for TFDP algorithms

$\underline{V}_t^0 \equiv -\infty$ and $\overline{V}_t^0 \equiv +\infty$ for $t \in [T]$;

for $k \in \mathbb{N}$ **do**

/* Forward phase: compute trajectory */

Set $x_0^k = x_0$;

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

Choose $\xi_t^k \in \mathrm{supp}(\boldsymbol{\xi}_t)$; (node seletion)

$x_t^k = \mathcal{F}_t(\underline{V}_{t+1}^{k-1})(x_{t-1}^k, \xi_t^k)$; (forward operator)

/* Backward phase: update approximations */

Set $\underline{V}_T^k \equiv \overline{V}_T^k \equiv 0$;

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

$f_t^k \leftarrow \underline{L}_t$-Lipschitz on X_t^r, valid and $\underline{\gamma}$-tight cut of $\mathcal{B}_t(\underline{V}_{t+1}^k)$ at x_{t-1}^k;

$\underline{V}_t^k \leftarrow \max(\underline{V}_t^{k-1}, f_t^k)$;

Define monotonous, $\bar{L}$-Lipschitz, valid, $\bar{\gamma}$-tight, $\overline{V}_t^k$;

TFDP convergence

We assume that:

- we have relatively complete recourse (RCR);
- the state remains in a *compact* set;
- we can compute Lipschitz cuts with uniformly bounded constant;
- the cuts are *exact* and *tight* where they are computed.

Then the lower-bound computed are valid and converging toward the true value, and the induced policy converged to an optimal policy.
We even have some (poor) complexity results.

To be continued

More on that during my talk Tuesday at 14:50 - Ballroom C

TFDP convergence

We assume that:

- we have relatively complete recourse (RCR);
- the state remains in a *compact* set;
- we can compute Lipschitz cuts with uniformly bounded constant;
- the cuts are *exact* and *tight* where they are computed.

Then the lower-bound computed are valid and converging toward the true value, and the induced policy converged to an optimal policy.
We even have some (poor) complexity results.

To be continued

More on that during my talk Tuesday at 14:50 - Ballroom C

TFDP convergence

We assume that:

- we have relatively complete recourse (RCR);
- the state remains in a *compact* set;
- we can compute Lipschitz cuts with uniformly bounded constant;
- the cuts are *exact* and *tight* where they are computed.

Then the lower-bound computed are valid and converging toward the true value, and the induced policy converged to an optimal policy.
We even have some (poor) complexity results.

To be continued

More on that during my talk Tuesday at 14:50 - Ballroom C

TFDP convergence

We assume that:

- we have relatively complete recourse (RCR);
- the state remains in a *compact* set;
- we can compute Lipschitz cuts with uniformly bounded constant;
- the cuts are *exact* and *tight* where they are computed.

Then the lower-bound computed are valid and converging toward the true value, and the induced policy converged to an optimal policy.
We even have some (poor) complexity results.

To be continued

More on that during my talk Tuesday at 14:50 - Ballroom C

Comparing DP and SDDP

	DP	SDDP
Independence assumption	Yes 👎	Yes 👎
Finitely supported noise	Yes 👎	Yes 👎
Structural assumptions	No 👍	Yes 👎
Discrete control	Yes 👍	No 👎
State discretization	Yes 👎	No 👍
Progressive results	No 👎	Yes 👍
Maximum state dimension	≈ 5 👎	≈ 30 👍
Maximum control dimension	≈ 5 👎	≈ 1000 👍

Contents

Risk neutral linear setting

$$\min_{\boldsymbol{x}_{[T]},\boldsymbol{u}_{[T]}} \quad \mathbb{E}\Big[\sum_{t=1}^{T} \boldsymbol{p}_t^\top \boldsymbol{u}_t\Big] \tag{MSLP}$$

$$\text{s.t.} \quad \boldsymbol{A}_t\boldsymbol{x}_t + \boldsymbol{B}_t\boldsymbol{x}_{t-1} + \boldsymbol{T}_t\boldsymbol{u}_t = \boldsymbol{d}_t \qquad \forall t \in [T]$$

$$\underline{x}_t \leq \boldsymbol{x}_t \leq \overline{x}_t, \quad \underline{u}_t \leq \boldsymbol{u}_t \leq \overline{u}_t, \qquad \forall t \in [T]$$

$$\boldsymbol{u}_t \preceq \sigma(\boldsymbol{\xi}_{[t]}) \qquad \forall t \in [T]$$

where $\boldsymbol{\xi}_t = (\boldsymbol{A}_t, \boldsymbol{B}_t, \boldsymbol{T}_t, \boldsymbol{d}_t)$ is a random vector with support Ξ_t.

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad p_\xi^\top u + \tilde{V}_{t+1}(x_{out})$$

$$\text{s.t.} \quad A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi$$

$$\underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u}$$

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) := \sum_{\xi \in \Xi_t} p_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi)$$

Risk neutral linear setting

$$\begin{aligned} \min_{\boldsymbol{x}_{[T]},\boldsymbol{u}_{[T]}} \quad & \mathbb{E}\Big[\sum_{t=1}^{T} \boldsymbol{p}_t^\top \boldsymbol{u}_t\Big] && \text{(MSLP)} \\ \text{s.t.} \quad & \boldsymbol{A}_t \boldsymbol{x}_t + \boldsymbol{B}_t \boldsymbol{x}_{t-1} + \boldsymbol{T}_t \boldsymbol{u}_t = \boldsymbol{d}_t && \forall t \in [T] \\ & \underline{x}_t \leq \boldsymbol{x}_t \leq \overline{x}_t, \quad \underline{u}_t \leq \boldsymbol{u}_t \leq \overline{u}_t, && \forall t \in [T] \\ & \boldsymbol{u}_t \preceq \sigma(\boldsymbol{\xi}_{[t]}) && \forall t \in [T] \end{aligned}$$

where $\boldsymbol{\xi}_t = (\boldsymbol{A}_t, \boldsymbol{B}_t, \boldsymbol{T}_t, \boldsymbol{d}_t)$ is a random vector with support Ξ_t.

$$\begin{aligned} \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad & p_\xi^\top u + \tilde{V}_{t+1}(x_{out}) \\ \text{s.t.} \quad & A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi \\ & \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \end{aligned}$$

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) := \sum_{\xi \in \Xi_t} p_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi)$$

LP formulation of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

Assume that $\tilde{V}_{t+1}$ is a polyhedral function defined as:

$$\tilde{V}_{t+1} : x \mapsto \max_{\kappa \leq K} \quad {\alpha_\kappa}^\top x + \beta_\kappa$$

Then, we can write $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ as a linear program:

$$\begin{aligned} \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad & {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out}) \\ \text{s.t.} \quad & A_\xi x_{out} + T_\xi u = d_\xi - B_\xi x_{in} \\ & \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \end{aligned}$$

LP formulation of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

Assume that $\tilde{V}_{t+1}$ is a polyhedral function defined as:

$$\tilde{V}_{t+1} : x \mapsto \max_{\kappa \leq K} \quad {\alpha_\kappa}^\top x + \beta_\kappa$$

Then, we can write $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ as a linear program:

$$\begin{aligned} \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad & {p_\xi}^\top u + \theta \\ \text{s.t.} \quad & A_\xi x_{out} + T_\xi u = d_\xi - B_\xi x_{in} \\ & \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \\ & {\alpha_\kappa}^\top x_{out} + \beta_\kappa \leq \theta \qquad \forall \kappa \leq K \end{aligned}$$

LP formulation of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

Assume that $\tilde{V}_{t+1}$ is a polyhedral function defined as:

$$\tilde{V}_{t+1} : x \mapsto \max_{\kappa \leq K} \quad {\alpha_\kappa}^\top x + \beta_\kappa$$

Then, we can write $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ as a linear program:

$$\begin{aligned} \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad & {p_\xi}^\top u + \theta \\ \text{s.t.} \quad & A_\xi x_{out} + T_\xi u = d_\xi - B_\xi x_{in} \\ & \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \\ & {\alpha_\kappa}^\top x_{out} + \beta_\kappa \leq \theta \qquad \forall \kappa \leq K \end{aligned}$$

➥ Computing $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ consists in solving a LP.

Some properties of $\dot{\mathcal{B}}_t$

$$\begin{aligned} \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad & {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out}) \\ \text{s.t.} \quad & A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi \\ & \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \end{aligned}$$

We have that:

- if $V^\flat_{t+1} \leq \tilde{V}_{t+1}$, then $\dot{\mathcal{B}}_t(V^\flat_{t+1}) \leq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})$,
- if $\tilde{V}_{t+1}$ is convex, so is $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$,
- if $\tilde{V}_{t+1}$ is polyhedral, so is $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$.

Same properties[4] hold true for $\mathcal{B}_t$ instead of $\dot{\mathcal{B}}_t$.

[4] finite support assumption required for polyhedrality

Some properties of $\dot{\mathcal{B}}_t$

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad p_\xi{}^\top u + \tilde{V}_{t+1}(x_{out})$$

$$\text{s.t.} \quad A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi$$

$$\underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u}$$

We have that:

- if $V^\flat_{t+1} \leq \tilde{V}_{t+1}$, then $\dot{\mathcal{B}}_t(V^\flat_{t+1}) \leq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})$,
- if $\tilde{V}_{t+1}$ is convex, so is $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$,
- if $\tilde{V}_{t+1}$ is polyhedral, so is $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$.

Same properties[4] hold true for $\mathcal{B}_t$ instead of $\dot{\mathcal{B}}_t$.

[4] finite support assumption required for polyhedrality

Convex duality to obtain cut

Consider a proper lowersemicontinuous convex function f of two variables, and g the partial infimum, i.e.

$$g : x_0 \mapsto \min_{x,y} \quad f(x,y)$$
$$\text{s.t.} \quad x = x_0 \qquad [\alpha]$$

Then convex duality theory tells us that g is convex and the optimal multiplier $\alpha \in \partial g(x_0)$ is a subgradient[5] of g at x_0.

More precisely, we have:

$$g(x) \geq g(x_0) + \alpha^\top (x - x_0) \quad \forall x$$

[5] Beware that the sign of the multiplier for an equality constraint is not clearly defined, thus depending of the Lagrangian you write / your solver implementation you might need to consider $-\alpha$

Convex duality to obtain cut

Consider a proper lowersemicontinuous convex function f of two variables, and g the partial infimum, i.e.

$$g : x_0 \mapsto \min_{x,y} \quad f(x,y)$$
$$\text{s.t.} \quad x = x_0 \qquad [\alpha]$$

Then convex duality theory tells us that g is convex and the optimal multiplier $\alpha \in \partial g(x_0)$ is a subgradient[5] of g at x_0.

More precisely, we have:

$$g(x) \geq g(x_0) + \alpha^\top (x - x_0) \quad \forall x$$

[5] Beware that the sign of the multiplier for an equality constraint is not clearly defined, thus depending of the Lagrangian you write / your solver implementation you might need to consider $-\alpha$

Convex duality to obtain cut

Consider a proper lowersemicontinuous convex function f of two variables, and g the partial infimum, i.e.

$$g : x_0 \mapsto \min_{x,y} \quad f(x,y)$$
$$\text{s.t.} \quad x = x_0 \qquad [\alpha]$$

Then convex duality theory tells us that g is convex and the optimal multiplier $\alpha \in \partial g(x_0)$ is a subgradient[5] of g at x_0.

More precisely, we have:

$$g(x) \geq g(x_0) + \alpha^\top (x - x_0) \quad \forall x$$

[5] Beware that the sign of the multiplier for an equality constraint is not clearly defined, thus depending of the Lagrangian you write / your solver implementation you might need to consider $-\alpha$

Computing a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi) := \min_{x_{in},x_{out},u} \quad {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out})$$

$$\text{s.t.} \quad A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi$$

$$\underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u}$$

$$x_{in} = x \qquad [\dot{\alpha}_\xi]$$

- By convexity duality we have that

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi) \geq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi) + {\dot{\alpha}_\xi}^\top (x_{in} - x), \qquad \forall x_{in}.$$

- By monotonicity, if $\tilde{V}_{t+1} \leq V_{t+1}$, then

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1}) \leq \dot{\mathcal{B}}_t(V_{t+1}) = \dot{V}_t$$

Computing a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

$$
\begin{aligned}
\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi) := \min_{x_{in},x_{out},u} \quad & {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out}) \\
\text{s.t.} \quad & A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi \\
& \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \\
& x_{in} = x \qquad\qquad\qquad\qquad [\dot{\alpha}_\xi]
\end{aligned}
$$

- By convexity duality we have that

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi) \geq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi) + {\dot{\alpha}_\xi}^\top (x_{in} - x), \qquad \forall x_{in}.$$

- By monotonicity, if $\tilde{V}_{t+1} \leq V_{t+1}$, then

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1}) \qquad \leq \dot{\mathcal{B}}_t(V_{t+1}) \qquad = \dot{V}_t$$

Computing a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

$$\begin{aligned}
\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x, \xi) := \min_{x_{in}, x_{out}, u} \quad & {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out}) \\
\text{s.t.} \quad & A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi \\
& \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \\
& x_{in} = x \qquad\qquad [\dot{\alpha}_\xi]
\end{aligned}$$

- By convexity duality we have that

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) \geq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x, \xi) + {\dot{\alpha}_\xi}^\top (x_{in} - x), \qquad \forall x_{in}.$$

- By monotonicity, if $\tilde{V}_{t+1} \leq V_{t+1}$, then

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1}) \leq \dot{\mathcal{B}}_t(V_{t+1}) = \dot{V}_t$$

Computing a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi) := \min_{x_{in},x_{out},u} \quad {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out})$$
$$\text{s.t.} \quad A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi$$
$$\underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u}$$
$$x_{in} = x \qquad [\dot{\alpha}_\xi]$$

- By convexity duality we have that

$$\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi) \geq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi) + {\dot{\alpha}_\xi}^\top (x_{in} - x), \qquad \forall x_{in}.$$

- By monotonicity, if $\tilde{V}_{t+1} \leq V_{t+1}$, then

$${\dot{\alpha}_\xi}^\top x_{in} + \dot{\beta}_\xi \leq \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi) \leq \dot{\mathcal{B}}_t(V_{t+1})(x_{in},\xi) = \dot{V}_t(x_{in},\xi)$$

with $\dot{\beta}_\xi = \mathcal{B}_t(\tilde{V}_{t+1})(x,\xi) - {\dot{\alpha}_\xi}^\top x$.

Computing a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$

We saw that, when solving $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x,\xi)$, we can compute a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ at x, i.e.,

$$\dot{\alpha}_\xi^\top x_{in} + \dot{\beta}_\xi \leq \dot{V}_t(x_{in},\xi), \qquad \forall x_{in}$$

As

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) = \sum_{\xi\in\Xi_t} p_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in},\xi)$$

$$V_t(\cdot) = \sum_{\xi\in\Xi_t} p_\xi \dot{V}_t(\cdot,\xi)$$

to compute a cut for $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, we have to solve $|\Xi_t|$ LPs, each of them giving a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, and average them:

$$\alpha := \sum_{\xi\in\Xi_t} p_\xi \dot{\alpha}_\xi \qquad \beta := \sum_{\xi\in\Xi_t} p_\xi \dot{\beta}_\xi$$

yielding

$$\alpha^\top x_{in} + \beta \leq V_t(x_{in}), \qquad \forall x_{in}.$$

Computing a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$

We saw that, when solving $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x, \xi)$, we can compute a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ at x, i.e.,

$$\dot{\alpha}_\xi{}^\top x_{in} + \dot{\beta}_\xi \leq \dot{V}_t(x_{in}, \xi), \qquad \forall x_{in}$$

As

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) = \sum_{\xi \in \Xi_t} p_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi)$$

$$V_t(\cdot) = \sum_{\xi \in \Xi_t} p_\xi \dot{V}_t(\cdot, \xi)$$

to compute a cut for $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, we have to solve $|\Xi_t|$ LPs, each of them giving a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, and average them:

$$\alpha := \sum_{\xi \in \Xi_t} p_\xi \dot{\alpha}_\xi \qquad \beta := \sum_{\xi \in \Xi_t} p_\xi \dot{\beta}_\xi$$

yielding

$$\alpha^\top x_{in} + \beta \leq V_t(x_{in}), \qquad \forall x_{in}.$$

Computing a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$

We saw that, when solving $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x, \xi)$, we can compute a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ at x, i.e.,

$$\dot{\alpha}_\xi^\top x_{in} + \dot{\beta}_\xi \leq \dot{V}_t(x_{in}, \xi), \qquad \forall x_{in}$$

As

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) = \sum_{\xi \in \Xi_t} p_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi)$$

$$V_t(\cdot) = \sum_{\xi \in \Xi_t} p_\xi \dot{V}_t(\cdot, \xi)$$

to compute a cut for $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, we have to solve $|\Xi_t|$ LPs, each of them giving a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, and average them:

$$\alpha := \sum_{\xi \in \Xi_t} p_\xi \dot{\alpha}_\xi \qquad \beta := \sum_{\xi \in \Xi_t} p_\xi \dot{\beta}_\xi$$

yielding

$$\alpha^\top x_{in} + \beta \leq V_t(x_{in}), \qquad \forall x_{in}.$$

Computing a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$

We saw that, when solving $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x, \xi)$, we can compute a cut of $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})$ at x, i.e.,

$$\dot{\alpha}_\xi{}^\top x_{in} + \dot{\beta}_\xi \leq \dot{V}_t(x_{in}, \xi), \qquad \forall x_{in}$$

As

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) = \sum_{\xi \in \Xi_t} p_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi)$$

$$V_t(\cdot) = \sum_{\xi \in \Xi_t} p_\xi \dot{V}_t(\cdot, \xi)$$

to compute a cut for $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, we have to solve $|\Xi_t|$ LPs, each of them giving a cut of $\mathcal{B}_t(\tilde{V}_{t+1})$ at x, and average them:

$$\alpha := \sum_{\xi \in \Xi_t} p_\xi \dot{\alpha}_\xi \qquad \beta := \sum_{\xi \in \Xi_t} p_\xi \dot{\beta}_\xi$$

yielding

$$\alpha^\top x_{in} + \beta \leq V_t(x_{in}), \qquad \forall x_{in}.$$

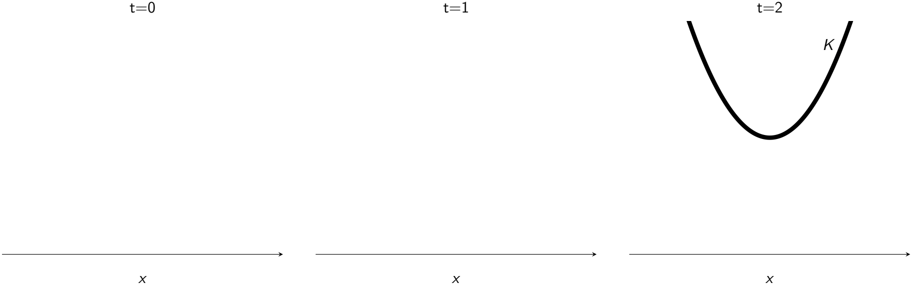
Forward Bellman operator

Note that, in order to compute $\mathcal{F}_t(\tilde{V}_{t+1})(x, \xi)$, we need to solve the same stage problem as $\dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x, \xi)$ i.e.

$$\begin{aligned} \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi) := \min_{x_{out}, u} \quad & {p_\xi}^\top u + \tilde{V}_{t+1}(x_{out}) \\ \text{s.t.} \quad & A_\xi x_{out} + B_\xi x_{in} + T_\xi u = d_\xi \\ & \underline{x} \leq x_{out} \leq \overline{x}, \quad \underline{u} \leq u \leq \overline{u} \end{aligned}$$

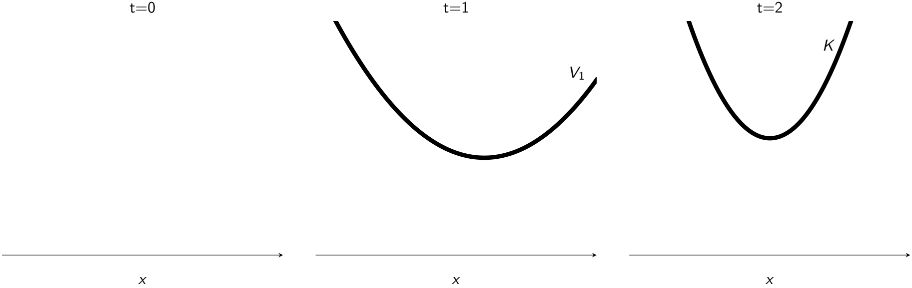
and return x_{out}.

SDDP



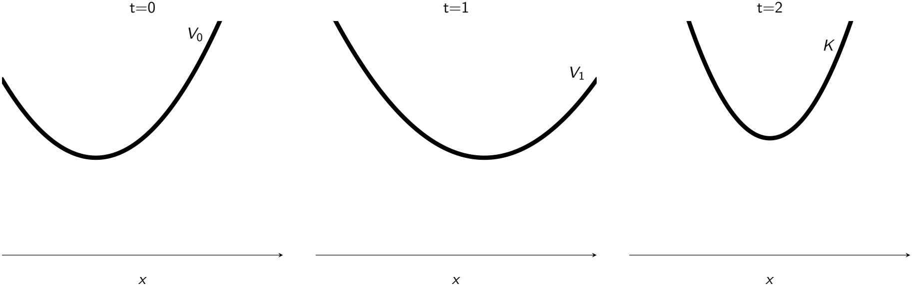
Final Cost $V_2 = K$

SDDP



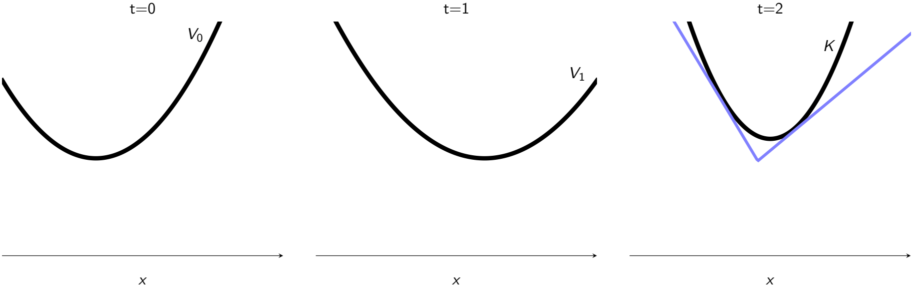
Real Bellman function $V_1 = \mathcal{B}_1(V_2)$

SDDP



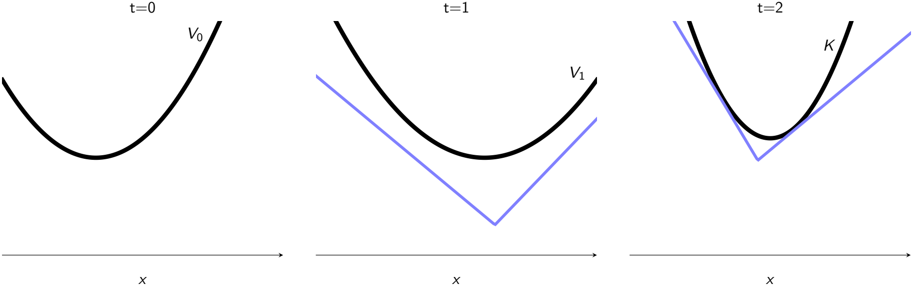
Real Bellman function $V_0 = \mathcal{B}_0(V_1)$

SDDP



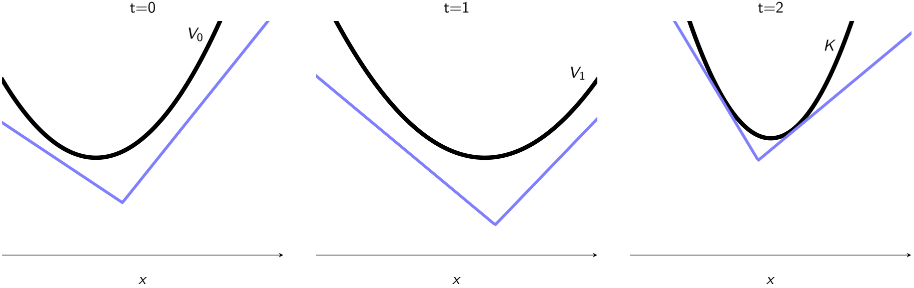
Lower polyhedral approximation $\underline{K}$ of K

SDDP



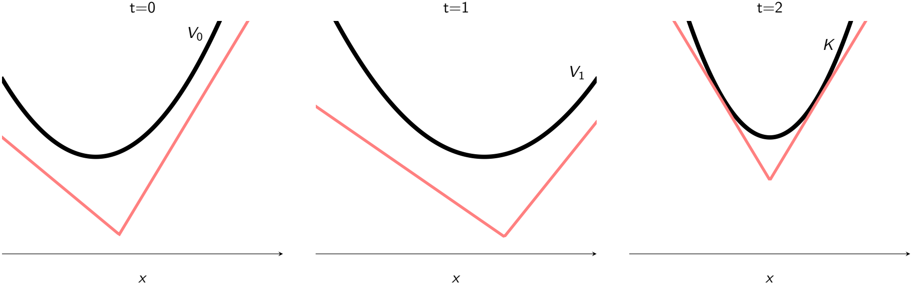
Lower polyhedral approximation $\underline{V}_1 = \mathcal{B}_t(\underline{K})$ of V_1

SDDP



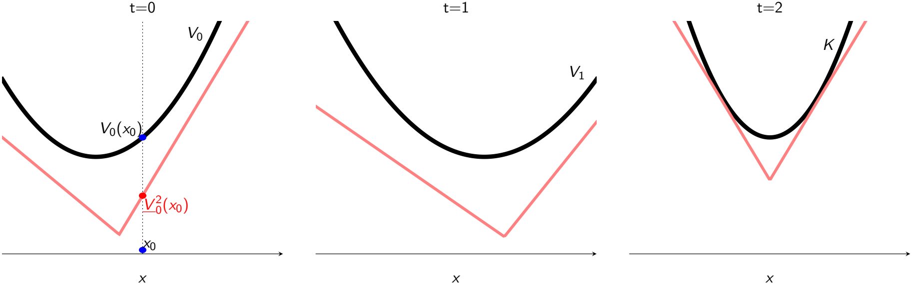
Lower polyhedral approximation $\underline{V}_0 = \mathcal{B}_t(\underline{V}_1)$ of V_0

SDDP



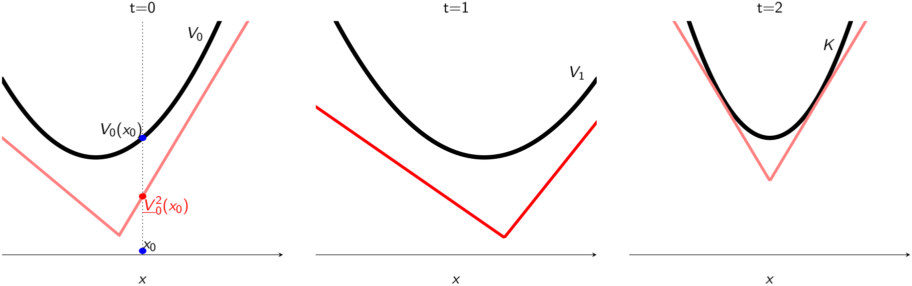
Assume that we have lower polyhedral approximations of V_t

SDDP



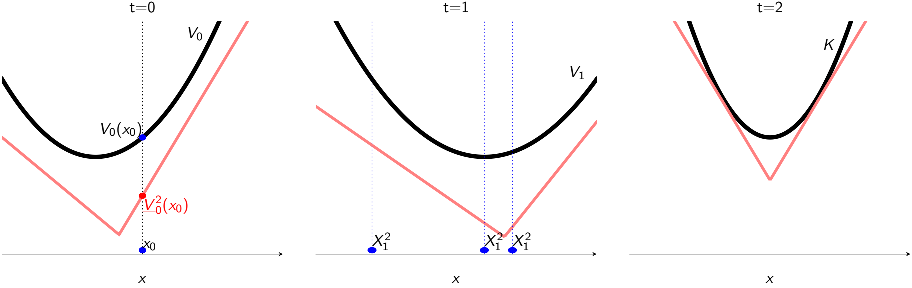
Obtain a lower bound on the value of our problem

SDDP



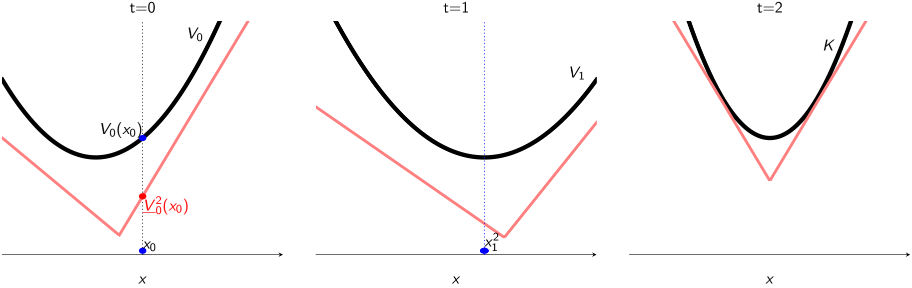
Apply $\mathcal{F}_0\big(\underline{V}_1^{(2)}\big)(x_0)$ and obtain $\boldsymbol{X}_1^{(2)}$

SDDP



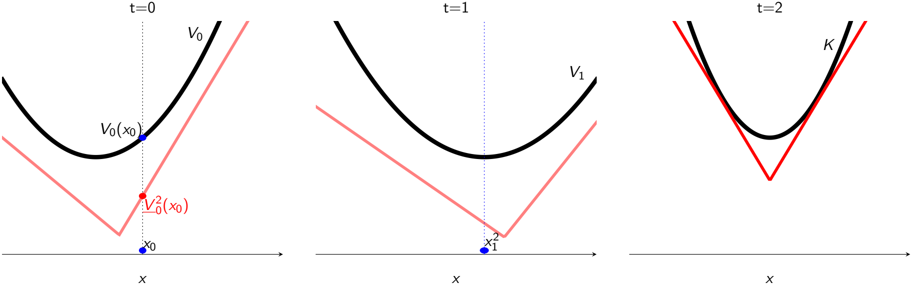
Apply $\mathcal{F}_0\big(\underline{V}_1^{(2)}\big)(x_0)$ and obtain $\boldsymbol{X}_1^{(2)}$

SDDP



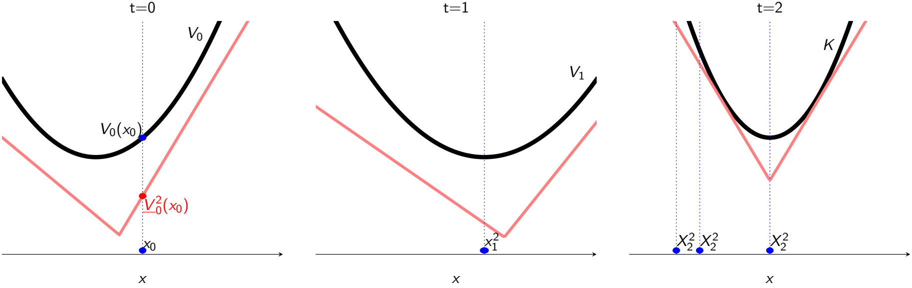
Draw a random realisation $x_1^{(2)}$ of $\boldsymbol{X}_1^{(2)}$

SDDP



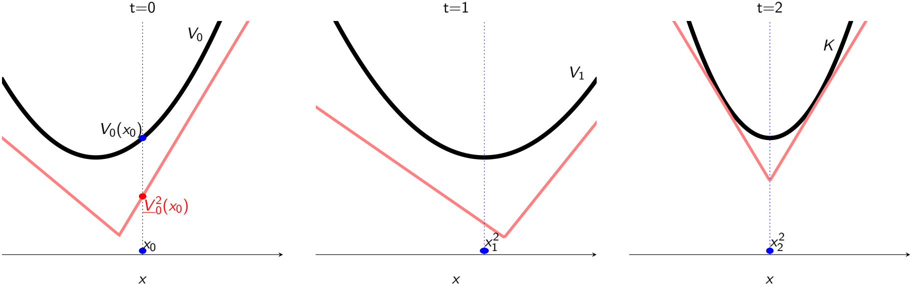
We apply $\mathcal{F}_1\big(\underline{V}_1^{(2)}\big)(x_1^{(2)})$ and obtain $\boldsymbol{X}_2^{(2)}$

SDDP



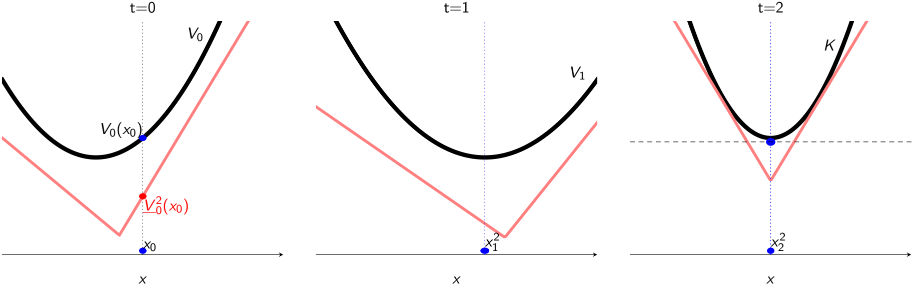
We apply $\mathcal{F}_1\big(\underline{V}_1^{(2)}\big)(x_1^{(2)})$ and obtain $\boldsymbol{X}_2^{(2)}$

SDDP



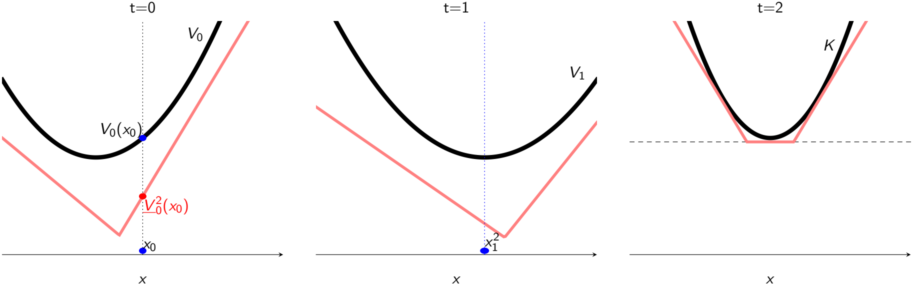
Draw a random realisation $x_2^{(2)}$ of $\boldsymbol{X}_2^{(2)}$

SDDP



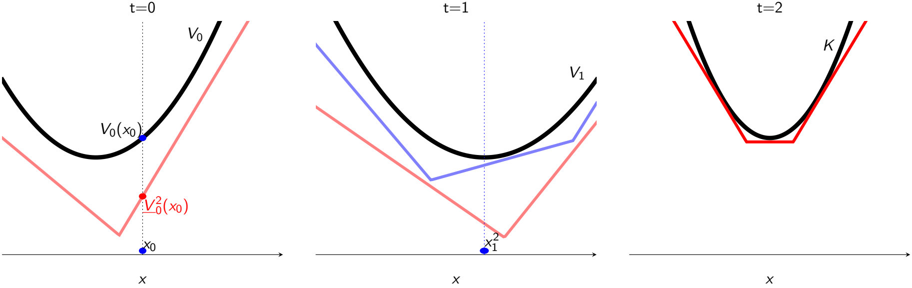
Compute a cut for K at $x_2^{(2)}$

SDDP



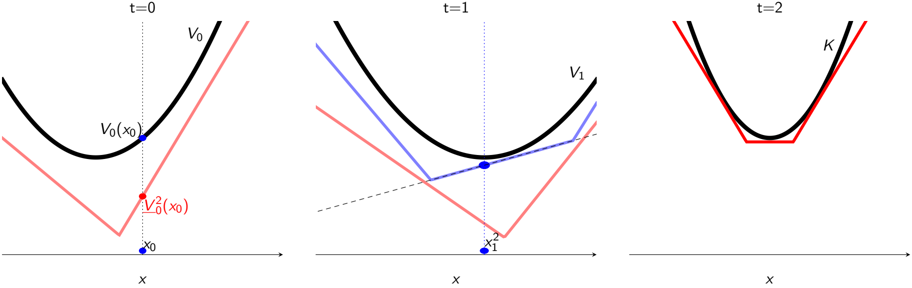
Add the cut to $\underline{V}_2^{(2)}$ which gives $\underline{V}_2^{(3)}$

SDDP



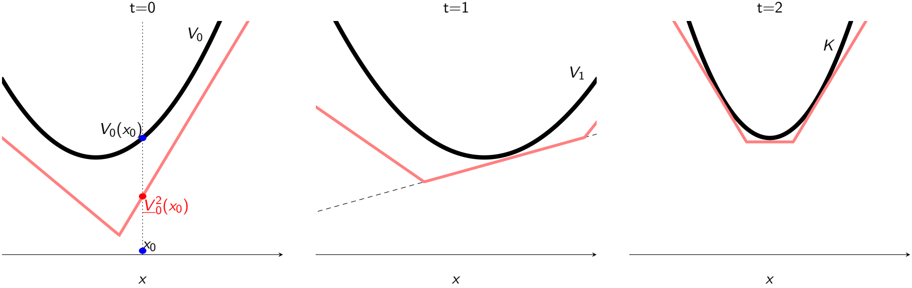
A new lower approximation of V_1 is $\mathcal{B}_1(\underline{V}_2^{(3)})$

SDDP



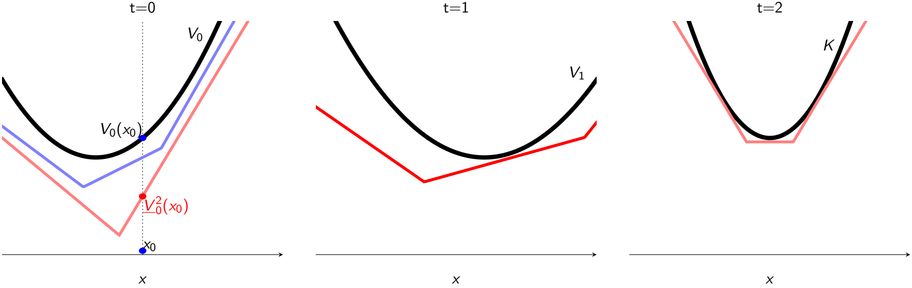
Compute the face active at $x_1^{(2)}$

SDDP



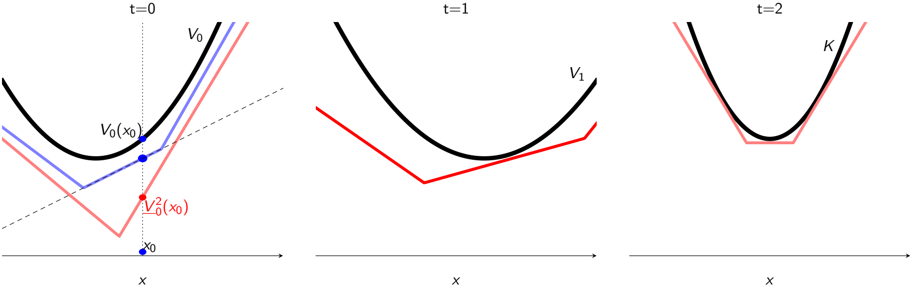
Add the cut to $\underline{V}_1^{(2)}$ which gives $\underline{V}_1^{(3)}$

SDDP



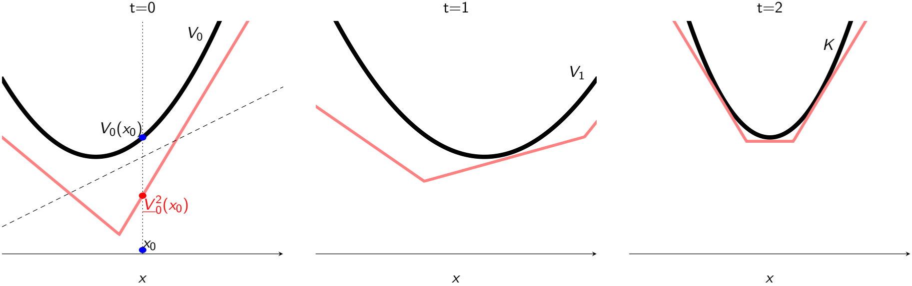
A new lower approximation of V_0 is $\mathcal{B}_0(\underline{V}_1^{(3)})$

SDDP



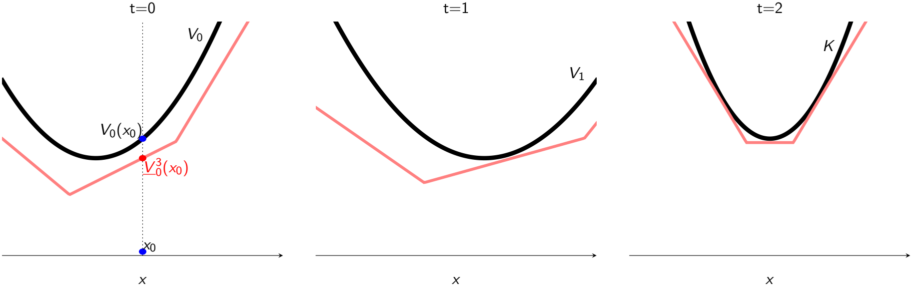
Compute the face active at x_0

SDDP



Compute the face active at x_0

SDDP



Obtain a new lower bound

Algorithm 3: SDDP algorithm

$\underline{V}_t^0 \equiv -\infty$ and $\overline{V}_t^0 \equiv +\infty$ for $t \in [T]$;

for $k \in \mathbb{N}$ **do**

  `/* Forward phase:  compute trajectory */`

  Set $x_0^k = x_0$;

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

    Randomly draw $\xi_t^k \in \mathrm{supp}(\boldsymbol{\xi}_t)$; (node seletion)

    $x_t^k = \mathcal{F}_t(\underline{V}_{t+1}^{k-1})(x_{t-1}^k, \xi_t^k)$; (forward operator)

  `/* Backward phase:  update approximations */`

  Set $\underline{V}_T^k \equiv 0$;

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

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

      Solve $\dot{\mathcal{B}}_t(\underline{V}_{t+1}^k)(x_{t-1}^k, \xi)$ for $\dot{\alpha}_\xi$ and $\dot{\beta}_\xi$;

    Compute $\alpha_t^k := \sum_{\xi \in \Xi_t} p_\xi \dot{\alpha}_{t,\xi}^k$ and, $\beta_t^k := \sum_{\xi \in \Xi_t} p_\xi \dot{\beta}_{t,\xi}^k$;

    Update $\underline{V}_t^k := \max\left(\underline{V}_t^{k-1}, \langle \alpha_t^k, \cdot \rangle + \beta_t^k\right)$;

Various numerical comments

- You need to use the same solver for training and simulating, otherwise you can go into unexplored territory.
- The forward pass requires solving T one-stage LPs; the backward pass require $T \times |\Xi_t|$ one-stage LPs.
- Most SDDP implementation ask for a lower-bound. This is not necessary if the first forward pass can be replaced by an admissible trajectory.
- Standard SDDP implementation compute $N \approx 200$ trajectories in the forward pass, and then add N cuts in the backward pass.
- An easy alternative consists in keeping the $|\Xi_t|$ per-ξ cuts of $\dot{V}_t$ instead of averaging them $\rightsquigarrow$ multicut version of SDDP.

Various numerical comments

- You need to use the same solver for training and simulating, otherwise you can go into unexplored territory.
- The forward pass requires solving T one-stage LPs; the backward pass require $T \times |\Xi_t|$ one-stage LPs.
- Most SDDP implementation ask for a lower-bound. This is not necessary if the first forward pass can be replaced by an admissible trajectory.
- Standard SDDP implementation compute $N \approx 200$ trajectories in the forward pass, and then add N cuts in the backward pass.
- An easy alternative consists in keeping the $|\Xi_t|$ per-ξ cuts of $\dot{V}_t$ instead of averaging them $\rightsquigarrow$ multicut version of SDDP.

Various numerical comments

- You need to use the same solver for training and simulating, otherwise you can go into unexplored territory.
- The forward pass requires solving T one-stage LPs; the backward pass require $T \times |\Xi_t|$ one-stage LPs.
- Most SDDP implementation ask for a lower-bound. This is not necessary if the first forward pass can be replaced by an admissible trajectory.
- Standard SDDP implementation compute $N \approx 200$ trajectories in the forward pass, and then add N cuts in the backward pass.
- An easy alternative consists in keeping the $|\Xi_t|$ per-ξ cuts of $\dot{V}_t$ instead of averaging them $\rightsquigarrow$ multicut version of SDDP.

Various numerical comments

- You need to use the same solver for training and simulating, otherwise you can go into unexplored territory.
- The forward pass requires solving T one-stage LPs; the backward pass require $T \times |\Xi_t|$ one-stage LPs.
- Most SDDP implementation ask for a lower-bound. This is not necessary if the first forward pass can be replaced by an admissible trajectory.
- Standard SDDP implementation compute $N \approx 200$ trajectories in the forward pass, and then add N cuts in the backward pass.
- An easy alternative consists in keeping the $|\Xi_t|$ per-ξ cuts of $\dot{V}_t$ instead of averaging them $\leadsto$ multicut version of SDDP.

Various numerical comments

- You need to use the same solver for training and simulating, otherwise you can go into unexplored territory.
- The forward pass requires solving T one-stage LPs; the backward pass require $T \times |\Xi_t|$ one-stage LPs.
- Most SDDP implementation ask for a lower-bound. This is not necessary if the first forward pass can be replaced by an admissible trajectory.
- Standard SDDP implementation compute $N \approx 200$ trajectories in the forward pass, and then add N cuts in the backward pass.
- An easy alternative consists in keeping the $|\Xi_t|$ per-ξ cuts of $\dot{V}_t$ instead of averaging them $\rightsquigarrow$ multicut version of SDDP.

Contents

Contents

Stopping tests

There are various ways of deciding to stop SDDP

- Statistical stopping test:
  - Estimate the cost associated to the current policy (an upper bound) by Monte Carlo and compare it to the lower bound.[6]
  - Statistically test if the lower-bound is no longer increasing
- Exact stopping test:
  - Maintain an exact upper bound and stop when the gap is small enough.
  - Computing exact upper bounds can be done using convexity or duality.
  - ➥ More on that in Bernardo da Costa talk (Tuesday 12:40-14:30 Meeting B).
- Pragmatic criterion:
  - Number of iterations
  - Time limit

[6] The correct way to do say is to set an a-priori gap ε and compare the upper end of a Monte-Carlo confidence interval of the current policy, to the (exact) lower bound.

Cut selection

- With each iteration, we add new cuts to the approximations of the value functions.
- Some of these cuts become useless as the algorithm progresses, and just burden the LP solver
- Cut selection are here to prune some of these constraints, usually in a heuristic way.
- Level-1 selection might be the most common:
  - Keep in memory all trial trajectories
  - Every $K \approx 50$ iterations, mark, for each of the past trial points, which of the cuts are active
  - Delete all inactive cuts

Node selection

For a given in-state x_{t-1}^k, and there are $|\Xi_t|$ possible out-state $x_{t-1,\xi}^k$. Choosing which one is kept is the *node selection procedure*:

1. random node selection: the noise ξ_t^k used to obtain x_t^k in the forward pass is selected randomly, independently of other node selection.
2. problem-child node selection: we choose the ξ_t^k that lead to a x_t^k maximizing the current gap estimate.
3. importance sampling node selection: the noise is selected randomly according to a specific probability measure.

Node selection

For a given in-state x_{t-1}^k, and there are $|\Xi_t|$ possible out-state $x_{t-1,\xi}^k$. Choosing which one is kept is the *node selection procedure*:

1. random node selection: the noise ξ_t^k used to obtain x_t^k in the forward pass is selected randomly, independently of other node selection.
   ➥ the most common, but hardest to study.
2. problem-child node selection: we choose the ξ_t^k that lead to a x_t^k maximizing the current gap estimate.
3. importance sampling node selection: the noise is selected randomly according to a specific probability measure.

Node selection

For a given in-state x_{t-1}^k, and there are $|\Xi_t|$ possible out-state $x_{t-1,\xi}^k$. Choosing which one is kept is the *node selection procedure*:

1. random node selection: the noise ξ_t^k used to obtain x_t^k in the forward pass is selected randomly, independently of other node selection.

➥ the most common, but hardest to study.

2. problem-child node selection: we choose the ξ_t^k that lead to a x_t^k maximizing the current gap estimate.

➥ some numerical advantages, and good theoretical guarantees.

3. importance sampling node selection: the noise is selected randomly according to a specific probability measure.

Node selection

For a given in-state x_{t-1}^k, and there are $|\Xi_t|$ possible out-state $x_{t-1,\xi}^k$. Choosing which one is kept is the *node selection procedure*:

1. random node selection: the noise ξ_t^k used to obtain x_t^k in the forward pass is selected randomly, independently of other node selection.

➡ the most common, but hardest to study.

2. problem-child node selection: we choose the ξ_t^k that lead to a x_t^k maximizing the current gap estimate.

➡ some numerical advantages, and good theoretical guarantees.

3. importance sampling node selection: the noise is selected randomly according to a specific probability measure.

➡ Can be numerically efficient, especially in the risk averse case.

Regularization

- Cutting plane algorithm are known to be unstable, and greatly benefit from regularization.
- Multiple approaches have been proposed to regularize SDDP:
  - add a quadratic penalty term to the last iterate
  - ➥ quite surprising as the state depend on the scenario
  - use a level-regularization approach
  - ➥ require upper-bounds and some parameter tweaking

➥ Still an active research area

Dual SDDP

Dual SDDP leverage Fenchel / Lagrangian duality to compute exact upper-bound.

- The basic idea is the following (MSLP case): if $V_t = \mathcal{B}_t(V_{t+1})$, then $V_t^\star = \mathcal{B}_t^\ddagger(V_{t+1}^\star)$, where $\mathcal{B}_t^\ddagger$ is an explicit Bellman operator[7]
- We can thus use SDDP on the $V_t^\star = \mathcal{B}_t^\ddagger(V_{t+1}^\star)$ recursion, which yields an exact lower bound of $V_t^\star$.
- Taking again the transform, the lower bound in the dual become an upper bound in the primal

[7] There are some technical tricks I'm glossing over...

Contents

Without Relatively Complete Recourse: Feasibility cuts

- Relatively Complete Recourse is required for SDDP to work in practice and in theory.
- Without RCR we can use feasibility cut (see standard introduction on Bender's decomposition)
- However, to ensure convergence we need to stop the forward pass as soon as we encounter a feasability cut and propagate it backward, which is time consuming (we can never reach the horizon)

➥ In practice it seems that using slack variable with high / increasing cost work best (and we can still use feasibility cuts in the end).

Non stagewise independent setting

Various ways to extend SDDP to non-stagewise independent setting:

(sampled) Nested-Benders In a fully dependent tree we associate a value function per node of the tree and iteratively add cuts.

Autoregressive Processes : for uncertainty in the right-hand side we can consider an Autoregressive process $d_t = \varepsilon_t + \beta_t + \sum_{\tau=1}^{k} \alpha_k d_{t-\tau}$, then we can consider an extended state $(\boldsymbol{x}_t, d_{t-1}, \dots d_{t-k})$, with linear dynamics and apply SDDP.

Markov Chain If the noise is a Markov Chain, or has a law which depends on a Markov Chain, we can also use a variant of SDDP. See David Wozabal talk for that.

Risk averse setting

- We consider a nested risk-averse problem, where the Bellman operator is defined as

$$\mathcal{B}_t(\tilde{V}_{t+1})(x_{in}) = \sup_{q \in \mathbb{Q}} \sum_{\xi \in \Xi_t} q_\xi \dot{\mathcal{B}}_t(\tilde{V}_{t+1})(x_{in}, \xi)$$

  where $\mathbb{Q}$ is a set of vectors representing probability measures.
- Then the DP equations holds, by construction of nested-risk measures, and we can run the SDDP algorithm almost straightforwardly.
- The only tricky point is that the averaging of cut coefficient should be done with respect to the maximazing q.

Rectangular robustness

- Consider a robust approach, and assume that the robust set is a Cartesian product $\Xi_1 \times \cdots \times \Xi_T$.
- It is equivalent to a nested risk-averse approach, where the set $\mathbb{Q}$ contains all diracs.
- The reference algorithm is the Robust Dual Dynamic Programming (RDDP) algorithm, which use a problem-child node selection approach

Infinite horizon

- There have been multiple proposition to extend SDDP to an infinite horizon framework, where we solve

$$V = \mathcal{B}(V)$$

- The core idea is to have forward pass going further and further
- An important extension is the periodic setting, which is relevant for long-term energy applications for example.

Conclusion

- TFDP algorithm are Dynamic Programming methods that iteratively refine approximations of the value functions
- They are less subject to the curse of dimensionality as:
  - they leverages structure of the problem to have global approximation
  - they smartly determine where to refine approximations along iterations
- Among them SDDP, for convex problem, is the most well-known and used algorithm
- It has numerous usefull extensions:
  - to risk-averse or distributionally robust model
  - to Markov Chain noises
  - to integer variables
  - to stochastic or infinite horizon
  - ...

Very short and partial bibliography

Pereira M. , and Pinto, L.
Multi-stage stochastic optimization applied to energy planning.
Mathematical programming, 1991

Shapiro. A.
Analysis of stochastic dual dynamic programming method.
European Journal of Operational Research 2011.

Zou, J., Ahmed, S., and Sun, X. A.
Stochastic dual dynamic integer programming.
Mathematical Programming, 2019.

Georghiou, A., Tsoukalas, A., and Wiesemann, W.
Robust dual dynamic programming.
Operations Research, 2019.

Dowson, O., and Kapelevich, L.
SDDP. jl: a Julia package for stochastic dual dynamic programming.
INFORMS Journal on Computing, 2021.

Forcier, M., and Leclere, V.
Convergence of Trajectory Following Dynamic Programming algorithms for multistage stochastic problems without finite support assumptions.
Journal of Convex Analysis, 2023 (to appear).


Anymore questions for me?