FAST, Finally An SDDP Toolbox

The first (free) Matlab toolbox to solve Multi-Stage Stochastic Programs using SDDP

Léopold Cambier

Multi-Stage Stochastic Programming

Example: Hydro-Thermal Scheduling

Over 24 hours, at each hour

- we need to satisfy a known demand D ;
- we can use fuel (p_t) for a price C or
- we can use (y_t) or store in a dam (x_t) of max. capacity W a random amount of rain $r(\xi_t)$.

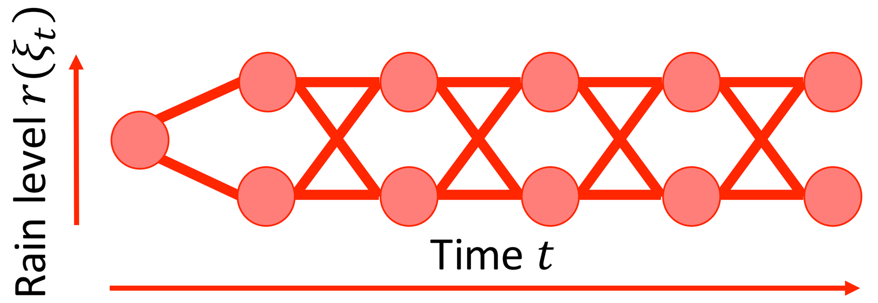
The problem is to meet all the demand at the smallest mean cost.

At each time t, the *subproblem,* looking at the future, can be formulated as

$$\begin{aligned} V_{t-1}(x_{t-1}) = \min_{x_t,y_t,p_t} \ & Cp_t + V_t(x_t) \\ \text{s.t.} \ & x_t \le W \\ & x_t = x_{t-1} - y_t + r(\xi_t) \\ & p_t + y_t \ge D \\ & x_t, y_t, p_t \ge 0 \end{aligned}$$

with $r(\xi_t)$ the random rain, where x_{t-1} is given and V_t is the expected cost of the remaining stages as a function of x_t.

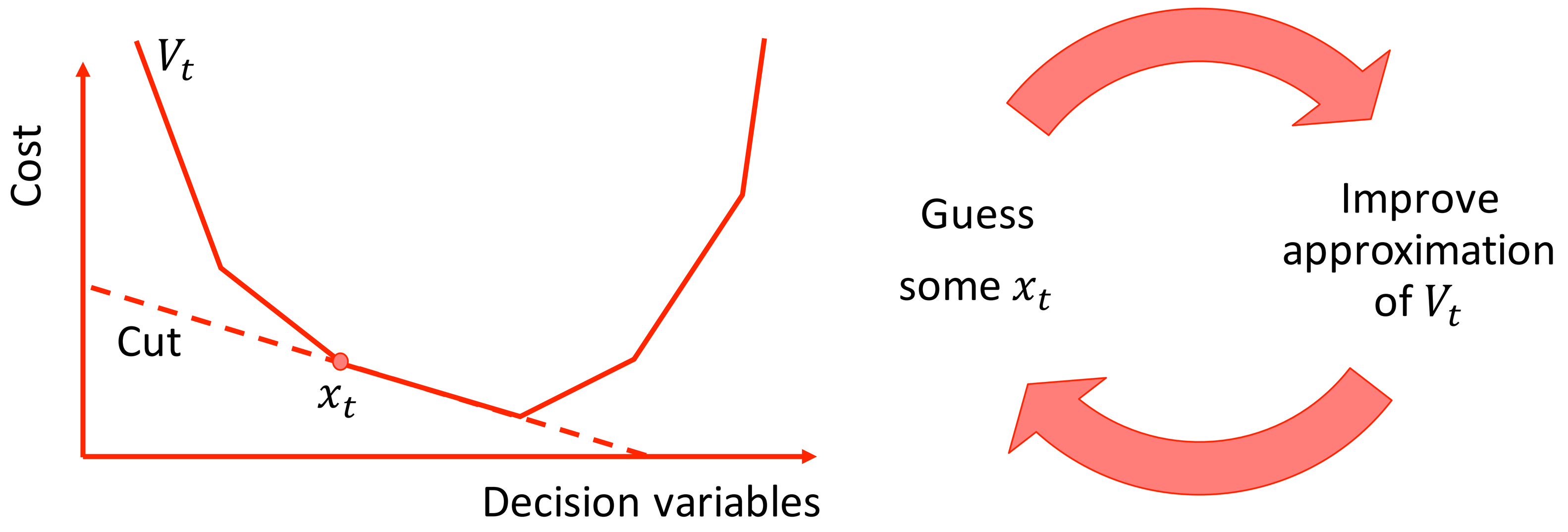
To represent time and uncertainty we use a Lattice.



We *could* model this as a plain Linear Program but there would be an exponential number of constraints and variables, due to the high number of possible paths in the Lattice.

Nested Decomposition + Monte-Carlo Estimates = SDDP

A key fact is that V_t is convex. We can thus approximate it using supporting hyper-planes (or cuts, or subgradients). This can be done by solving the individual subproblem (+ already existing cuts approximating $V_t(x_t)$) and using the dual information to build a new cut.



We can then *recursively* apply this idea to approximate the $V_t(x_t)$ at *each* node in the Lattice.

This would unfortunately require traversing all the paths of the lattice. To avoid this, we use Monte-Carlo estimates and we traverse only a few paths in the lattice. This is the idea of SDDP (Stochastic Dual Dynamic Programming).

Built-In Modeling

The toolbox, solving thus general Multi-Stage Stochastic Programs using SDDP, includes a modeling part to easily describe the subproblems, almost copy-pasting the equations ! (Compare this code to the equations on the left.)

```
function [cntr, obj] = nlds(scenario, x, y, p) % The subproblem
without V nor the cuts
C = 5 ; W = 8 ; D = 6 ;
t = scenario.time ;
rain = [2 10] ; % Rain either low of high
% The rain changes the quantity of water in the reservoir
if t == 1
    rain_effect = x(1) ==         - y(1) + mean(rain) ;
else
    rain_effect = x(t) == x(t-1) - y(t) + rain(scenario.index) ;
end
% Objective
obj = C * p(t) ;
% Constraints
cntr = [x(t) <= W, ...          % Reservoir level
        rain_effect, ...        % Influence of rain
        p(t) + y(t) >= D, ...   % Meet demand
        x(t) >= 0,...           % Positivity
        y(t) >= 0,...
        p(t) >= 0] ;
end
```

Simple Interface To Solvers

Once the problem has been modeled, running SDDP can be done in as few as 15 lines of code. The toolbox takes care of everything:

- computing the cuts to approximate V_t at each node ;
- using Monte-Carlo to avoid the curse of dimensionality ;
- interfacing with the linear solvers (Gurobi, Mosek or Linprog).

Many different options are present to tweak the algorithm.

```
H = 24 ;
R = 2 ;
% Creating a lattice with H stages and R scenarios at each time
lattice = Lattice.latticeEasy(H, R) ;
% Run SDDP
params = sddpSettings('algo.McCount',100,...
                      'stop.iterationMax',10,...
                      'stop.pereiraCoef',1,...
                      'solver','gurobi') ;
x = sddpVar(H) ; % The reservoir level at time t
y = sddpVar(H) ; % How much water we use at time t
p = sddpVar(H) ; % How much fuel we use at time t
Lattice = ...
compileLattice(lattice,@(scenario)nlds(scenario,x,y,p),params) ;
output = sddp(lattice,params) ;
% Visualise output
plotOutput(output) ;
```

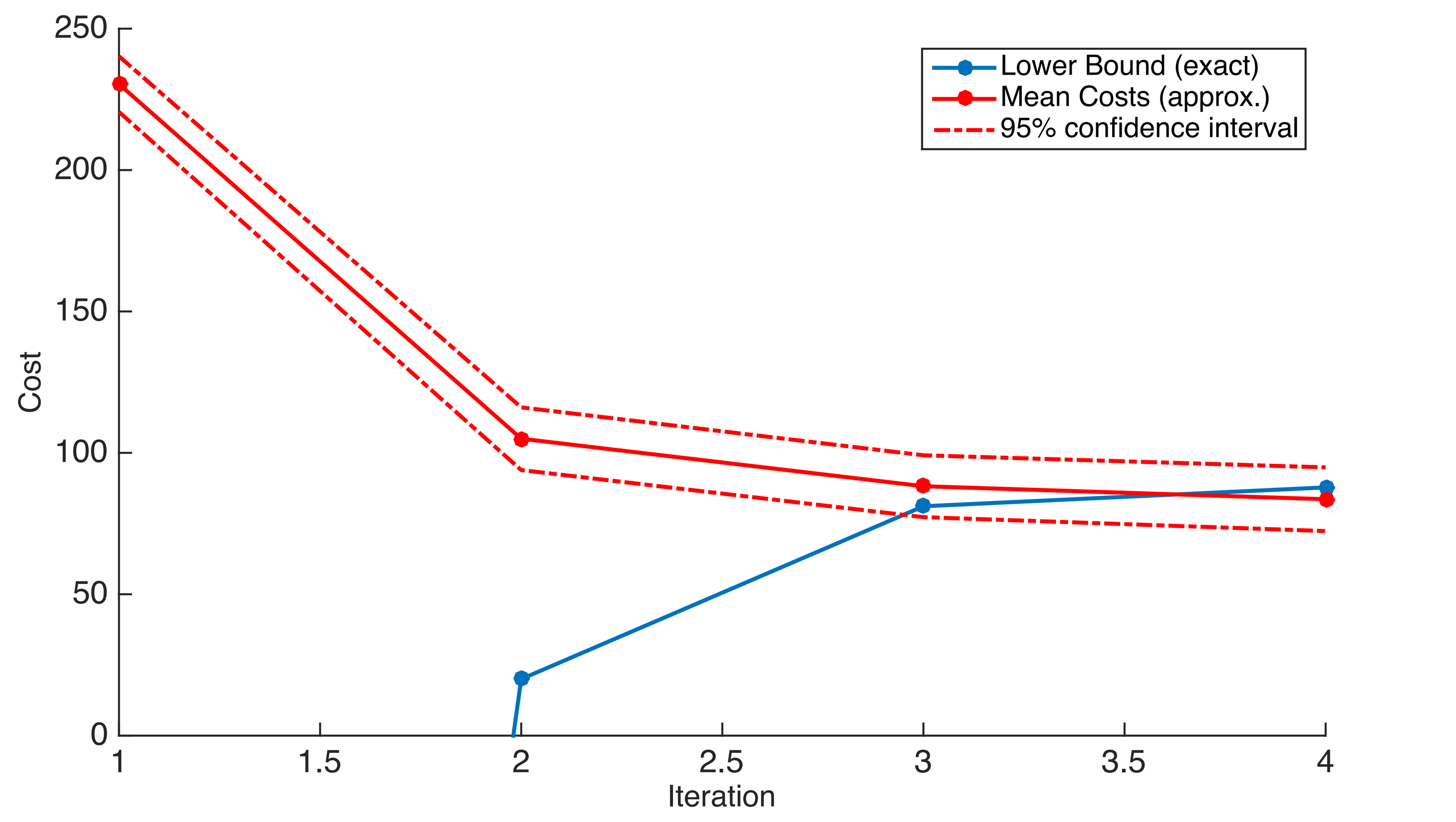
On the above problem (with approximately 2^{24} scenarios), it runs an find a solution within 5% of the optimal solution in 1 minute on a laptop. Increasing the number of Monte-Carlo samples would increase the precision of the output.

Easy-to-analyze Output

When the algorithm executes, we can compute

- an *approximate* upper-bound (the mean cost of the Monte-Carlo estimates),
- an *exact* lower-bound (the cost at time $t = 1$)

of the cost. The algorithm terminates when they are close enough.



We can then easily analyze using the toolbox:

- the evolution of the mean cost, the lowerbound, confidence intervals ;
- the solution for a given path in the lattice and the optimal policy at time $t = 1$;
- the lattice and the dual information ;
- other related quantities: wait-and-see, expected value.

Future Work

- Support for feasibility cuts (if the guessed x_t is not feasible) ;
- support for more solvers ;
- more flexible modeling tools (for instance, easy AR process definition) ;
- parallel version ;
- other (free) programming language(s).

References & Links

This toolbox has been developed in collaboration with Damien Scieur (ENS Paris/Inria).

The original paper on SDDP is

- M.V.F. Pereira and L.M.V.G. Pinto, Multi-stage stochastic optimization applied to energy planning, *Mathematical Programming*, 52, 359–375, 1991.

The toolbox website (with code examples and tutorials) is available at

- www.baemerick.be/fast

The source code is hosted on Github

- www.github.com/leopoldcambier/fast

Publication coming soon ?