LABORATORY OF APPLIED MATHEMATICAL PROGRAMMING AND STATISTICS

LAMPS

PUC RIO

Solving Large-scale problems using JuMP

Thuener Silva

JuMP Developers meet-up

Santiago, March 13, 2019

Agenda

- LAMPS
- Research Projects
- Benchmarks
- SDDP
- Since then

LAMPS

More than 20 students most Ph.D. and M.Sc. candidates and researchers

6 professors from PUC-Rio from different backgrounds mainly in Optimization, Statistics and Temporal series

Mostly problems in energy, finance and oil and gas production

Why we use JuMP?

- JuMP was the major reason to migrate and convert every project in our laboratory(LAMPS) to Julia

- Versatile and easy to use (even for undergrad)

- Why get stuck with many different languages and solvers

Research Projects

- Churn and Fraud Detection in real time
- Incorporating the effect of climate variability and contingencies in the optimal contracting strategy of transmission-usage amounts
- Stochastic Dual Dynamic Programming Dispatch Tool
- Optimization model with uncertainty in real time for offshore platforms

Migrating to Julia/JuMP

As I finished my Ph.D. my advisor insists to port everything to Julia(the hole SDDP)

Cuts	Julia(sec.)	C++(sec.)
1	6.8	1.7
5	7.6	4.3
10	10.6	37.1
15	55.3	54.0
20	69.8	68.0
25	84.8	81.8
30	98.9	97.3

Humanitarian

Different languages

Inst.	Gap(%)			Time(sec.)		
	Julia	C++	Mosel	Julia	C++	Mosel
v10e20_s1	0.9	0.9	**0.37**	**22**	77	64
v10e20_s2	**0.68**	**0.68**	0.76	**41**	126	100
v10e20_s2	**0.07**	**0.07**	0.69	**25**	92	78
v10e20_s4	0.84	0.84	**0.41**	**71**	146	133
v10e20_s5	**0.41**	**0.41**	0.57	**118**	154	171
v12e25D	0.5	**0.49**	0.72	**23**	38	33
v13e30_s1	**0.58**	**0.58**	**N/A**	**11**	28	**N/A**
v13e30_s2	0.8	0.8	**0.51**	**192**	376	239
v13e30_s3	**0.0**	**0.0**	0.37	**14**	31	44
v13e30_s4	**0.0**	**0.0**	0.6	**51**	97	60
v13e30_s5	**0.0**	**0.0**	0.46	**92**	150	135

Cplex 12.7.0 / Xpress 8.0

Humanitarian

Different solvers in Julia/JuMP

Inst.	Gap(%)			Time(sec.)		
	Cplex	Gurobi	Xpress	Cplex	Gurobi	Xpress
v10e20_s1	0.9	0.53	**0.37**	**22**	124	60
v10e20_s2	0.68	0.77	**0.0**	**41**	163	117
v10e20_s2	**0.07**	0.67	0.69	**25**	79	70
v10e20_s4	0.84	**0.15**	0.41	**71**	127	126
v10e20_s5	0.41	**0.0**	0.57	**118**	222	151
v12e25D	0.5	**0.24**	0.41	**23**	49	31
v13e30_s1	0.58	0.89	**0.33**	**11**	39	20
v13e30_s2	0.8	0.62	**0.47**	**192**	245	240
v13e30_s3	**0.0**	0.15	0.01	**14**	51	43
v13e30_s4	**0.0**	0.18	0.84	**51**	74	**51**
v13e30_s5	**0.0**	0.47	0.78	**92**	118	109

Gurobi 7.0.2 / Cplex 12.7.0 / Xpress 8.0

Hydrotermal Dispach

Inst.	Matlab			Julia		
	CP	BB	BC	CP	BB	BC
3 Bus, k = 0	0.1	0.5	0.4	**0.0**	**0.0**	**0.0**
3 Bus, k = 1	0.1	0.4	0.4	**0.0**	**0.0**	**0.1**
24 Bus reduced,k = 0	0.0	0.8	0.2	**0.0**	**0.1**	**0.0**
24 Bus reduced, k = 1	35.0	4.1	67.2	**12.3**	**2.0**	**81.2**
24 Bus, k = 0	0.0	2.0	1.6	**0.0**	**0.1**	**0.3**
24 Bus, k = 1	223.2	62.0	1030.2	**11.6**	**39.0**	**121.8**

Cplex 12.7.0

Machine Learning

C++/Cplex(Concert)

points\dim	10	30	100	500	5000
10	0.4	0.5	0.7	1.8	15.3
15	1.0	1.2	1.7	5.5	53.1
20	1.8	2.4	3.8	12.7	
30	5.0	7.2	12.4	47.6	

Julia/JuMP(Cplex)

points\dim	10	30	100	500	5000
10	0.3	0.3	0.5	1.8	19.4
15	0.5	0.8	1.5	6.1	69.6
20	1.1	1.7	3.5	14.9	
30	3.7	6.2	12.9	60.0	

SDDP

How to solve multistage stochastic problems?

$$\max_{\substack{\mathbf{A}_1\mathbf{x}_1=\mathbf{b}_1\\ \mathbf{x}_1\geq 0}} \mathbf{c}_1^\top \mathbf{x}_1 + \mathbb{E}\left[\max_{\substack{\mathbf{A}_2\mathbf{x}_2=\mathbf{b}_2-\mathbf{B}_2\mathbf{x}_1\\ \mathbf{x}_2\geq 0}} \mathbf{c}_2^\top \mathbf{x}_2 + \ldots + \mathbb{E}\left[\max_{\substack{\mathbf{A}_T\mathbf{x}_T=\mathbf{b}_T-\mathbf{B}_T\mathbf{x}_{T-1}\\ \mathbf{x}_T\geq 0}} \mathbf{c}_T^\top \mathbf{x}_T \,\middle|\, \xi_{T-1}\right] \ldots \,\middle|\, \xi_1\right]$$

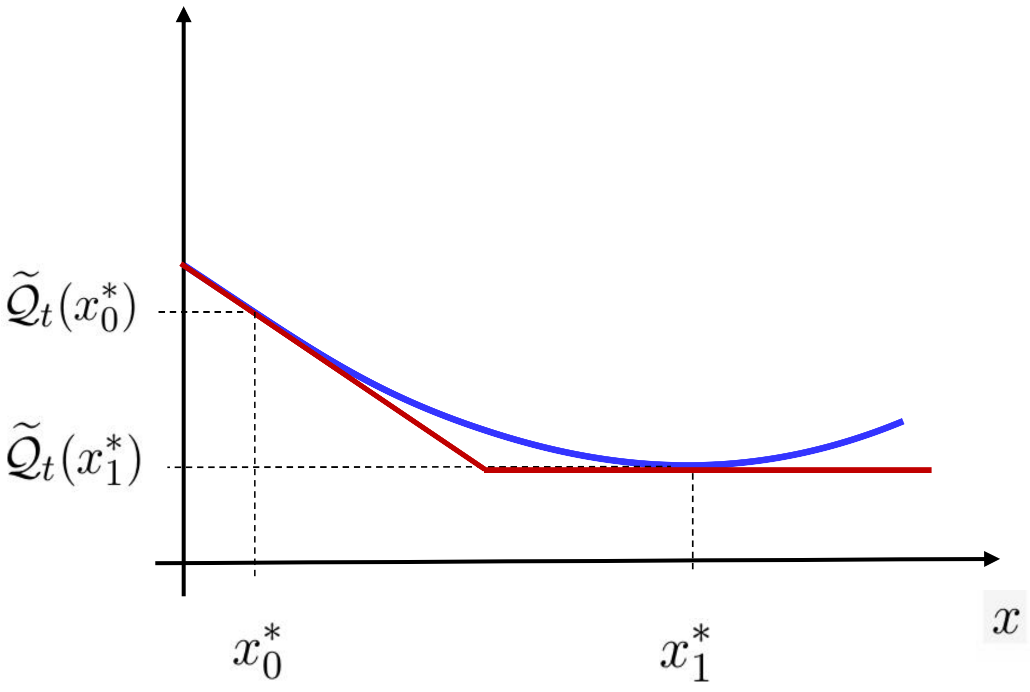
Bellman equations and cost function

$$\begin{aligned} Q_t(\mathbf{x}_{t-1}, \xi_{ts}) = \max_{\mathbf{x}_t} \; & \mathbf{c}_{ts}^\top \mathbf{x}_t + \mathcal{Q}_{t+1}(\mathbf{x}_t) \\ \text{s.t. } & \mathbf{A}_{ts}\mathbf{x}_t = \mathbf{b}_{ts} - \mathbf{B}_{ts}\mathbf{x}_{t-1} \\ & \mathbf{x}_t \geq 0 \end{aligned}$$

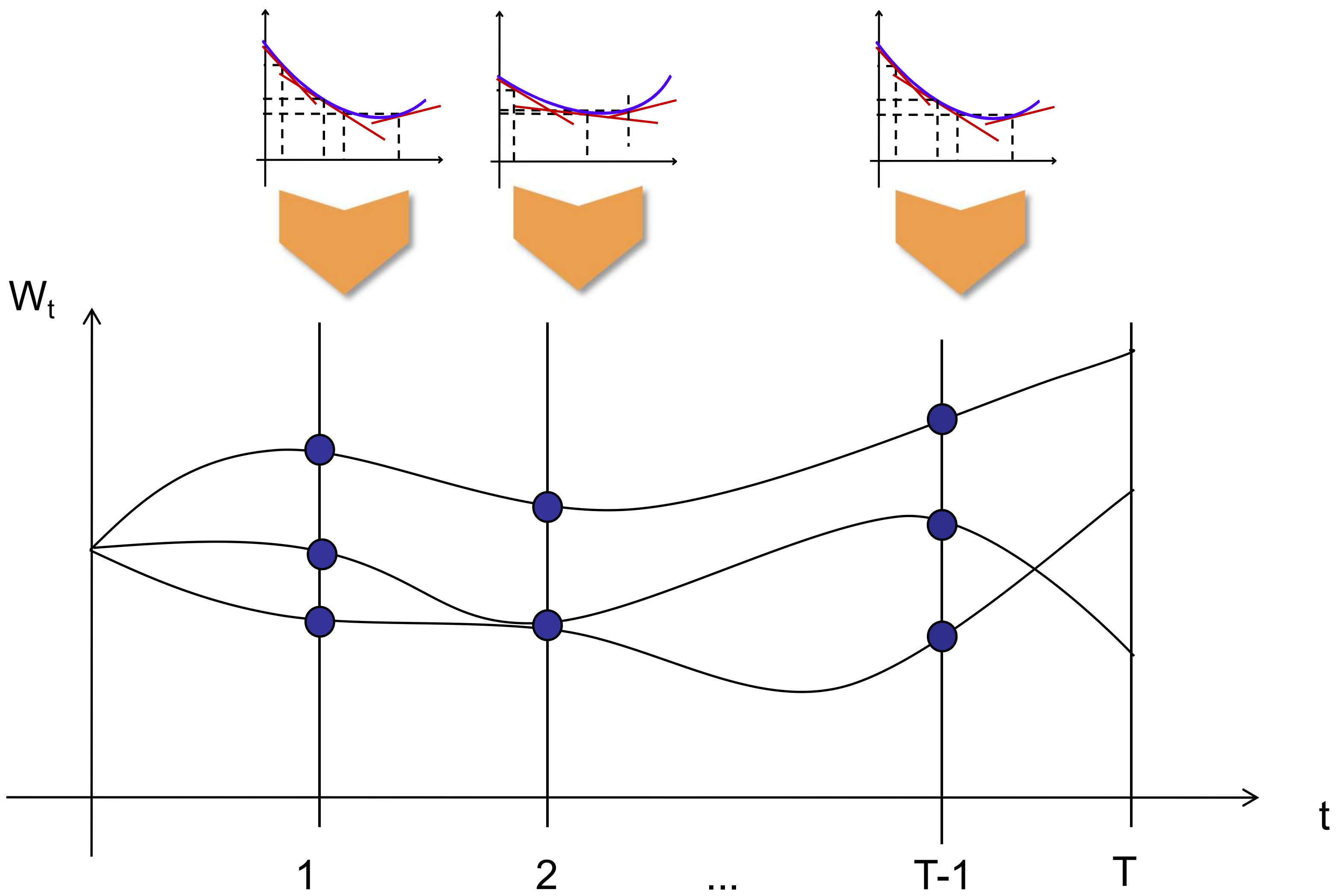
SDDP

Approximate the future cost function using a piecewise linear function

$$\mathfrak{Q}_t(\mathbf{x}_{t-1}) = \max_{l\in\mathcal{I}_t}\{\widetilde{\mathcal{Q}}_t(\mathbf{x}_{t-1,l}) + \widetilde{\mathbf{g}}_{tl}^\top(\mathbf{x}_{t-1} - \mathbf{x}_{t-1,l})\},\ \forall t \in \mathcal{H}$$



SDDP



SDDP Issues

- The first issue was memory consumption our model were consuming more than 128Gb of memory(Computational bottleneck)
- Couldn't remove constraints making difficult to remove cuts to optimize memory consumption
- Performance decrease compared to Low-level API

Solution

How can I solve those problems?

- I liked JuMP very much but for SDDP there were some problems

- I still manage to maintain the **construction** of the problem **using JuMP** but to change RHS and add constraints I had to use Cplex low-level API

- Choosing solver. I did benchmark tests to choose the best solver: Cplex was the best with Gurobi soon after

Solution

The difference of using low-level API (1355 cuts)

CPLEX API	
1360 MB	350 min
JuMP	
1514 MB	826 min

Adding constraints(Backward) was 2.2 times slower

Upper bound evaluation(chgrhs) was 3.41 times slower

JuMP.jl 0.18.1 and Cplex.jl 0.3.2

JuMP has a limit?

Memory consumption and time of @constraint #969

Closed Thuener opened this issue on Feb 20, 2017 · 24 comments

Thuener commented on Feb 20, 2017 • edited

I'm having some issues with memory consumption on JuMP. I have a problem that has too many constraints and the JuMP structures for macro **@constraint** is consuming too much memory.

Benchmark JuMP/Julia

```
m = Model(solver=CplexSolver())
@variable(m, 0 <= x[1:N] <= 1)
@variable(m, 0 <= θ <= 1000)

@objective(m, Max, θ )
```

Vectorized

```
@constraint(m, θ .<= coef*x  )
```

Scalar 1

```
for i in 1:size(coef,1)
  @constraint(m, θ <= sum(coef[i,j]*x[j] for j = 1:size(coef,2)))
end
```

Scalar 2

```
@constraint(m,[i = 1:size(coef,1)],
  θ <= sum(coef[i,j]*x[j] for j = 1:size(coef,2)))
```

Cplex API

```
rhs = zeros(C)
coef = hcat(-coef,ones(C));
CPLEX.add_constrs!(m.internalModel.inner, coef, '<', rhs)
```

Performance evolution

Adding constraints time(sec.) for each Julia, JuMP and Cplex versions

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 0.6.3 JuMP 0.18.1 Cplex 0.3.2			
12.5	5.0	5.1	2.0
12.5	4.9	5.0	1.8
Julia 0.7.0 JuMP 0.18.5 Cplex 0.4.3			
11.1	4.9	4.9	2.7
11.1	4.8	4.8	2.7
Julia 1.0.3 JuMP 0.18.5 Cplex 0.4.3			
11.1	4.9	5.2	2.6
10.9	5.1	5.0	2.7
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
22.4	**13.1**	**13.7**	2.7
21.1	**13.2**	**13.8**	2.7

C = 300,000

N = 100

This model is solved in 5 seconds

Cplex 12.7.0

Memory evolution

Adding constraints MB for each Julia, JuMP and Cplex versions

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 0.6.3 JuMP 0.18.1 Cplex 0.3.2			
1069	1125	1148	**348**
992	1118	1123	**347**
Julia 0.7.0 JuMP v0.18.5 Cplex 0.4.3			
1074	1111	1135	364
966	1130	1203	347
Julia 1.0.3 JuMP 0.18.5 Cplex 0.4.3			
1030	1198	1178	348
967	1234	1136	347
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
1115	**924**	1011	303
1047	**986**	1004	347

Direct Model

Time

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
20.7	12.1	12.7	2.5
20.0	12.2	12.7	2.6

Memory

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
1206	965	963	304
1130	1022	885	384

Direct Model

Direct Model

Time

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
20.7	12.1	12.7	2.5
20.0	12.2	12.7	2.6

Memory

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
1206	965	963	304
1130	1022	885	384

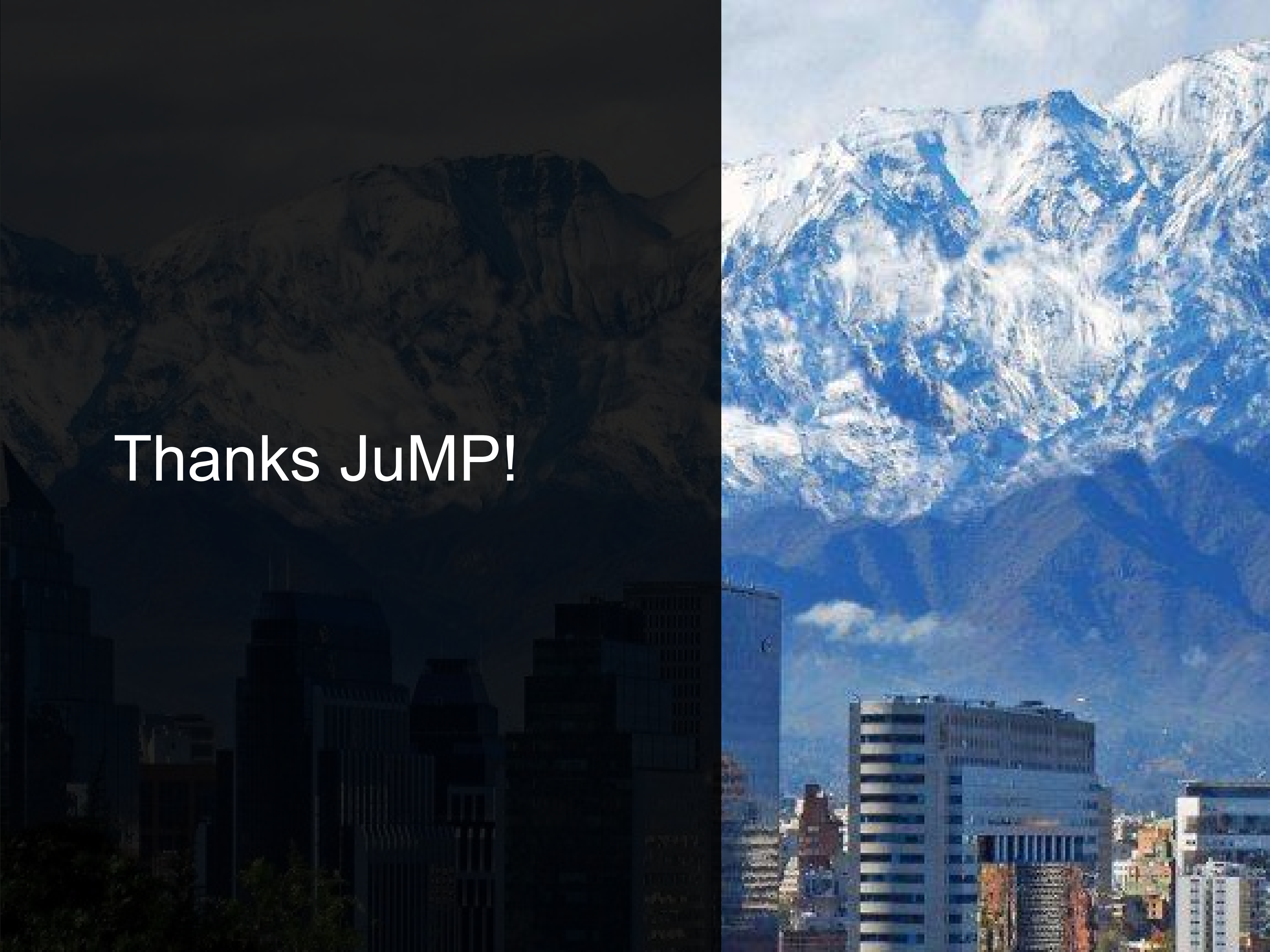
Direct Model

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
12.0	**6.0**	6.5	2.5
12.2	**6.0**	6.3	2.5

Vec.	Scalar 1	Scalar 2	Cplex API
Julia 1.0.3 JuMP 0.19.0 Cplex v0.4.3			
565	**495**	498	**304**
545	**495**	498	**384**

Since then

- Julia and JuMP change the way we develop software
- Now all our projects and courses are develop in Julia
- We are constructing frameworks in Julia using JuMP
- Our development and research is much faster making possible to construct big research with a small team
- LAMPS have more than **15** publications using JuMP and at least **14** in development


Thanks JuMP!

SDDP.jl

SDDP.jl the power of a framework

- At one point we had several SDDP frameworks in Julia
- Oscar comes with SDDP.jl and won the best SDDP framework in my opinion. Greate generic implementation many features
- Why I still don't use? I need to be able to construct different and efficient approaches fast. It is difficult to do this in generic frameworks

Research Projects

- Churn and Fraud Detection in real time
  - Is a challenge to process and optimize data with millions of transactions

- Incorporating the effect of climate variability and contingencies in the optimal contracting strategy of transmission-usage amounts
  - High-frequency series with a lot of data, even some database can't deal with it

- Stochastic Dual Dynamic Programming Dispatch Tool
  - Deal with Brazil huge hydro and thermoelectric generation (also Chile)

- Optimization model with uncertainty in real time for offshore platforms
  - Binaries to represent production curves per scenario (Special Ordered Set)

SDDP for portfolio

Portfolio allocation problem

$$
\begin{aligned}
& Q_t^j(\mathbf{u}_{t-1}, \mathbf{r}_t) = \\
& \max_{\mathbf{u}_t, \mathbf{b}_t, \mathbf{d}_t} \quad \sum_{k \in \mathcal{K}} \mathbb{E}\left[Q_{t+1}^k(\mathbf{u}_t, \mathbf{r}_{t+1}) \middle| K_{t+1} = k\right] \mathbf{p}_j(k) \\
& \quad \text{s.t.} \quad \rho_{\mathbf{p}_j}\left[\mathbf{r}_{t+1}^\top \mathbf{u}_t\right] + \mathbf{c}^\top (\mathbf{b}_t + \mathbf{d}_t) \leq \gamma\big((\mathbf{1} + \mathbf{r}_t)^\top \mathbf{u}_{t-1}\big) \\
& \qquad\qquad u_{0,t} + (\mathbf{1} + \mathbf{c})^\top \mathbf{b}_t - (\mathbf{1} - \mathbf{c})^\top \mathbf{d}_t = u_{0,t-1} \\
& \qquad\qquad u_{i,t} - b_{i,t} + d_{i,t} = (1 + r_{i,t})\, u_{i,t-1}, \quad \forall i \in \mathcal{A} \\
& \qquad\qquad \mathbf{u}_t, \mathbf{b}_t, \mathbf{d}_t \geq 0,
\end{aligned}
$$

Fewer stages, more realizations, more states, and Markov states

For example, 96 stages, 100 assets, and 3 Markov states. For this case, we have **288 JuMP** models and at each iteration, adding at least one cut to each