Automatic reformulation using constraint bridges

Benoît Legat (UCL)

June 29, 2018

Université catholique de Louvain (UCL)

Motivation

Consider interval constraints:

```
@constraint(m, 0 <= 2x + 3y <= 1)
```

and second order cone (SOC) constraints:

```
@constraint(m, x'*x <= t^2)
@constraint(m, [t; x] in MOI.SecondOrderCone(length(x)+1))
```

- Solver A: supports interval constraints and quadratic constraints
- Solver B: does not supports interval constraints and support SOC constraints.

What should JuMP do ?

Solution 1

Disallow using interval constraints

Issues

- Solver A benefits from knowing more stucture
- Does not work for SOC constraints

Solution 2

The user needs to enter the form supported by the solver

Issues

- The user needs to read solvers docs
- Some transformations are not easy, let alone transforming duals
- Cannot write solver independent code

Similar to MathProgBase LinearQuadraticModel/ConicModel/NLPModel with JuMP v0.18 traits.

Solution 3

Write transformations in JuMP

Issues

- Bloat JuMP code (need to transform duals!)
- Unfair: specific transformations are included and some are not
- Not extensible/distributed

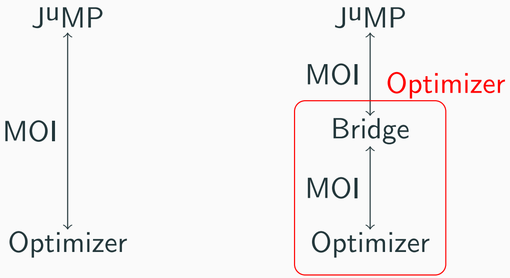
Similar to handling of PSD constraints in JuMP v0.18.

Solution 4: Constraint Bridges

- Transparent
- Lightweight
- Complete
- Extensible

Transparent

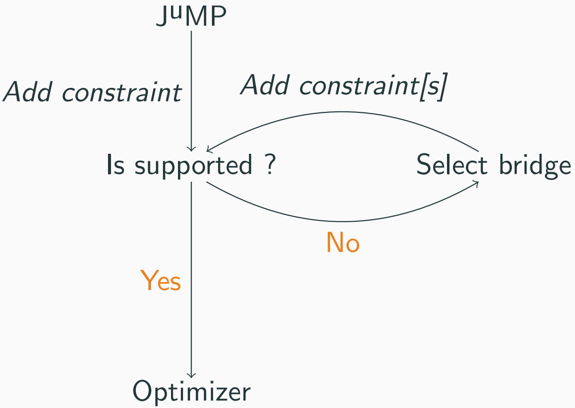
Transparently bridge constraints by adding an MOI layer



Lightweight

Transformed *on the fly*, no copy needed.

MathProgBase bridges: model-wise → need full copy.



Complete

If the underlying optimizer fully implements MOI, the bridged optimizer should too!

Bridges keep indices of created constraints and variables and implements

- transforming constraint primal and constraint dual,
- deleting the constraint,
- modifying the constraint,
- remove indices of created constraints and variables from `MOI.ListOfVariableIndices`, ...

Keep original constraint for gett `MOI.ConstraintFunction`, `MOI.ConstraintSet`, ...

Transforming constraint duals

Linear bridge from $*$-in-S_1 to $*$-in-S_2.
Suppose

$$x \in S_1 \Leftrightarrow Ax \in S_2 \qquad AS_1 = S_2$$

Hence

$$A^* y \in S_1 \Leftrightarrow y \in S_2 \qquad S_1^* = A^* S_2^*$$

In Lagrangian:

$$\langle Ax, y \rangle_2 = \langle x, A^* y \rangle_1$$

Extensible

Custom bridges can be added. How do we select bridges ?

Example
Root-Det constraint: $t \leq \sqrt[d]{\det(X)}$, $X \in \mathbb{R}^{d \times d}$.

Geometric-Mean constraint: $x \geq 0, t \leq \sqrt[n]{x_1 x_2 \cdots x_n}$

- Bridge 1: Root-Det $\to$ PSD to get eigenvalues and GeoMean with eigenvalues.
- Bridge 2: Root-Det $\to$ Rotated SOC.
- Bridge 3: Root-Det $\to$ Power Cone (see Ulf's talk on Wednesday).

Which one to choose ?

Select bridge that minimize the number of bridges needed ?

What do to for Bridge 2 and 3 ? Add cost to bridges ?

What is our graph ?

Nodes
Each F-in-S constraint types. Need to go beyond MOI's F and S.

It can by anything for extensibility.

Infinitely many nodes, we need to be lazy!

Edge
Each bridge b defined possible infinitely many edges.

For each F-in-S supported by bridge B: multi-ouput edge between F-in-S and all added constraint types ($\mathcal{A}(B, F, S)$).

Given F-in-S, finitely many bridges supporting F-in-S: $\mathcal{B}(F, S)$.

Shortest Path Problem

Need to solve

$$d(F,S) = \begin{cases} 0 & \text{if } F\text{-in-}S \text{ are supported by optimizer} \\ 1 + \min_{B \in \mathcal{B}(F,S)} \sum_{(F',S') \in \mathcal{A}(B,F,S)} d(F',S') & \text{otherwise} \end{cases}$$

Shortest path algrithms ?

- Breath-First Search : For edges with cost 1
- Dijkstra : For edges with nonnegative cost
- Bellman-Ford : For edges with any real cost (+ negative cycles)

Choice: a modified *Bellman-Ford algorithm*.

Classical Bellman-Ford algorithm

- `N`: set of nodes
- `E`: set of edges
- `d`: distance
- `b`: next node

```
for _ in 1:length(N)-1:
  for each edge u=>v with weight w in E
    if d[u] + w < d[v]:
      d[v] = d[u] + w
      b[v] = u
    end
  end
end
```

Complexity $\mathcal{O}(|N| \cdot |E|)$

Target constraint types

Invariant: if $d(F, S)$ defined, it is correct.

F-in-S constraint added by user

$\rightarrow$ generate $\mathcal{C}$: list of needed new entries in d.

Algorithm 1 recursive add F-in-S

```
add F-in-S to C
for B ∈ B(F, S) do
    for (F', S') ∈ A(B, F, S) do
        if F'-in-S' not supported and d(F', S') not defined then
            recursive add F'-in-S'
        end if
    end for
end for
```

Modified Bellman-Ford algorithm

```
changed ← true
while changed do
    changed ← false
    for F-in-S ∈ C do
        for B ∈ B(F, S) do
            u ← 1 + Σ_{(F′,S′)∈A(B,F,S)} d(F′, S′)
            if u < d(F, S) then
                d(F, S) ← u
                b(F, S) ← B
                changed ← true
            end if
        end for
    end for
end while
```

Future work

- Should GeoMean be bridged to RSOC or Power Cone ? Is adding weights the right solution ?
- Disciplined Convex Programming : Bridge between NonlinearFunction-in-S and convex constraints.
  Issue: cannot determine added constraint types only with NonlinearFunction type.