Automatic reformulation using optimization bridges

Benoît Legat (UCLouvain)

July 16, 2019

Los Alamos National Laboratory, 16th July 2019

Motivation

Examples

Combining bridges

Automatic selection of bridges

Integration with JuMP

Sum-of-Squares example

Reshaping

Motivation

MathOptInterface (MOI)

MOI in a nutshell:

- `add_variable(model)`.
- `add_constraint(model, func, set)`, e.g. $2x + 3y = 1 \rightarrow$ `(2*x + 3*y)-in-EqualTo(1.0)`.
- `set`, `get` attributes, e.g., `ObjectiveSense`, `ObjectiveFunction`.

Extensible framework:

- Generic on attribute, function and set types. New ones can be defined independently.
- Solver-specific features easily exposed to JuMP/MOI users through custom attributes.
- Expose specialized problem structure easily through custom functions, sets (e.g. Sum-of-Squares variables/constraints).

Semidefinite programming

$$\begin{array}{lll} \underset{Q \in \mathcal{S}^n}{\text{minimize}} & \langle C, Q \rangle & \underset{y \in \mathbb{R}^n}{\text{maximize}} \quad \langle b, y \rangle \\ \text{subject to} & \langle A_i, Q \rangle = b_i & \text{subject to} \quad \sum_i A_i y_i \preceq C \\ & Q \succeq 0 & \end{array}$$

File format: SDPA

Solvers: CSDP, SDPA, DSDP, SDPLR, ...

Variables: Q block diagonal, nonnegative scalar variables (1×1 blocks) or SDP matrices.

Constraints: Affine equations.

Conic Modelling

```
using JuMP
model = Model(...)
@variable(model, -1 <= x <= 1)
@variable(model, y)
@variable(model, z <= 0)
@constraint(model, [x + y  x
                    y      x - y] in PSDCone())
@constraint(model, [x + y, z, y] in SecondOrderCone())
@objective(model, x^2 - 2x*z + z^2)
```

The gap between models and solvers

The solver interface should only support structures and the algorithm exploits:

- n solvers and m structures $\rightarrow$ mn transformations $\rightarrow$ unscalable for large m, n.
- enables evaluation of formulation quality, e.g. automatic transformation and automatic dualization.

The model should

- be independent from solvers.
- represent the structure exploitable by algorithms.
- allow reprentable structure unknown to solvers, e.g. Sum-of-Squares variables/constraints.

Bridging the gap

$$x \in S_1 \Leftrightarrow Ax \in S_2 \qquad AS_1 = S_2$$
$$A^*y \in S_1^* \Leftrightarrow y \in S_2^* \qquad S_1^* = A^*S_2^*$$

In Lagrangian:

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

Transformation of variable-in-S_2 to variable-in-S_1.

Primal Transform value v to Av.

Dual Transform dual y to $A^{-*}y$.

Transformation of f-in-S_1 constraint to Af-in-S_2 constraint.

Primal Transform value v of Af to $A^{-1}v$ of f.

Dual Transform dual y of A^*y.

Examples

Examples i

FlipSignBridge

- **Variable** $x \geq l$ substituted by $x = -y$ where $y \leq -l$.
- **Constraint** $a^\top x \leq \beta$ transformed into $-a^\top x \geq -\beta$.

VectorizeBridge

- **Variable** $x \geq l$ substituted by $x = y + l$ where $y \in \mathbb{R}^1_+$.
- **Constraint** $a^\top x \leq \beta$ transformed into $[a^\top x - \beta] \in \mathbb{R}^1_-$.

IntervalBridge

- **Constraint** $l \leq a^\top x \leq u$ transformed into $a^\top x \geq l$ and $a^\top x \leq u$.

Examples ii

IndicatorBridge

- **Constraint** y binary, $\neg y \Rightarrow a^\top x \leq \beta$ transformed into z binary, $z + y = 1$ and $z \Rightarrow a^\top x \leq \beta$.

FreeBridge

- **Variable** $x \in \mathbb{R}$ substituted by $x = y + z$ where $y \in \mathbb{R}_+$ and $z \in \mathbb{R}_-$.

SlackBridge

- **Constraint** $f \in S$ transformed into $f = x$ for variable $x \in S$.
- **Objective** $\min f$ transformed into $\min x$ with $f \leq x$.

RSOC to PSD

Transforms rotated second order cone into positive semidefinite cone. By Schur Complement: $\| x \|_2^2 < 2tu, t, u > 0$ iff

$$\begin{cases} u > 0 \\ 2tu > x^\top x \end{cases} \Leftrightarrow \begin{cases} uI \succ 0 \\ t - x^\top (2uI)^{-1} x > 0 \end{cases} \Leftrightarrow \begin{pmatrix} t & x^\top \\ x & 2uI \end{pmatrix} \succ 0.$$

Dual of u
Diagonal at $(2,2),(3,3),\ldots,(n,n)$ is $2u\mathbf{1}$ so dual of u is scalar product between $(2\mathbf{1})^\top$ and dual of diagonal at $(2,2),(3,3),\ldots,(n,n)$.

Dual of x
Twice dual of $(1,2),(1,3),\ldots,(1,n)$. Pay attention to scalar product of vectorized PSD cone.

Quadratic to RSOC

Transforms quadratic constraints into rotated second order cone constraints.

$$\frac{1}{2}x^T Qx + a^T x + \beta \leq 0$$

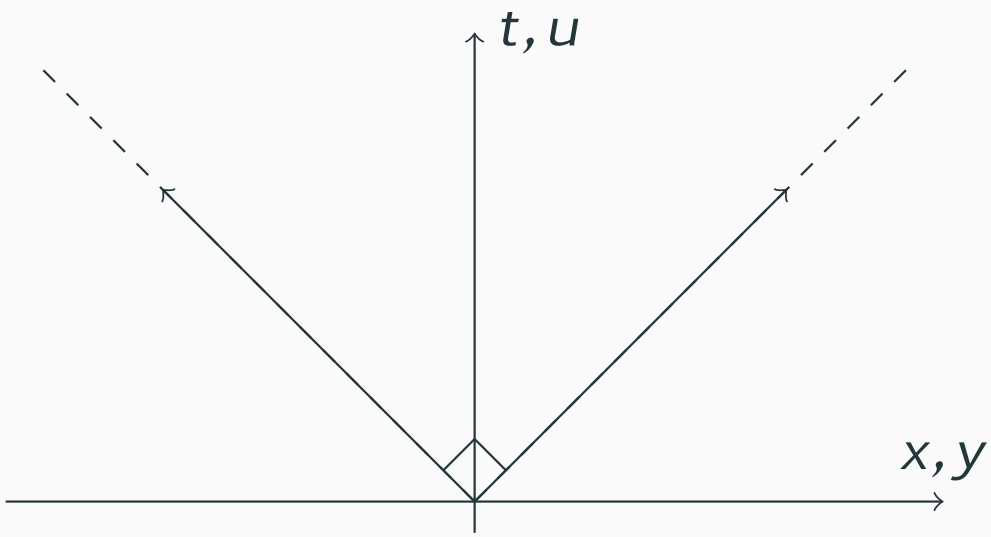
Q PSD $\rightarrow$ Cholesky $Q = U^\top U$:

$$\| Ux \|_2^2 \leq 2(-a^T x - \beta)$$

so $(1, -a^T x - \beta, Ux) \in$ RSOC.

Second order cone and complementarity slackness

If $(t,x),(u,y) \in$ SOC and $(t,x)\cdot(u,y)=0$, then (u,y) is parallel to $(t,-x)$ or (t,x) is zero or (u,y) is zero.



Lemma

If $(1, s, x), (v, u, y) \in$ RSOC and $(1, s, x) \cdot (v, u, y) = 0$, then $y = -ux$ and $v = u \parallel x \parallel_2^2 / 2$.

Proof (sketch)

$$
\begin{aligned}
0 &= (1, s, x) \cdot (v, u, y) \\
&= v + su + x \cdot y \\
\text{(Cauchy-Schwarz)} &\geq v + su - \parallel x \parallel_2 \parallel y \parallel_2 \\
\text{(RotatedSOC)} &\geq v + su - 2\sqrt{suv} \\
\text{(AM-GM)} &\geq v + su - 2\frac{v + su}{2} \\
&= 0.
\end{aligned}
$$

See MOI QuadtoSOCBridge for details.

Quadratic to RSOC duals

Consider $z^T U^\top U z/2 + a^\top z + \beta \leq 0$ transformed, so $(1, -a^T z - \beta, Uz) = (1, s, x) \in$ RSOC.

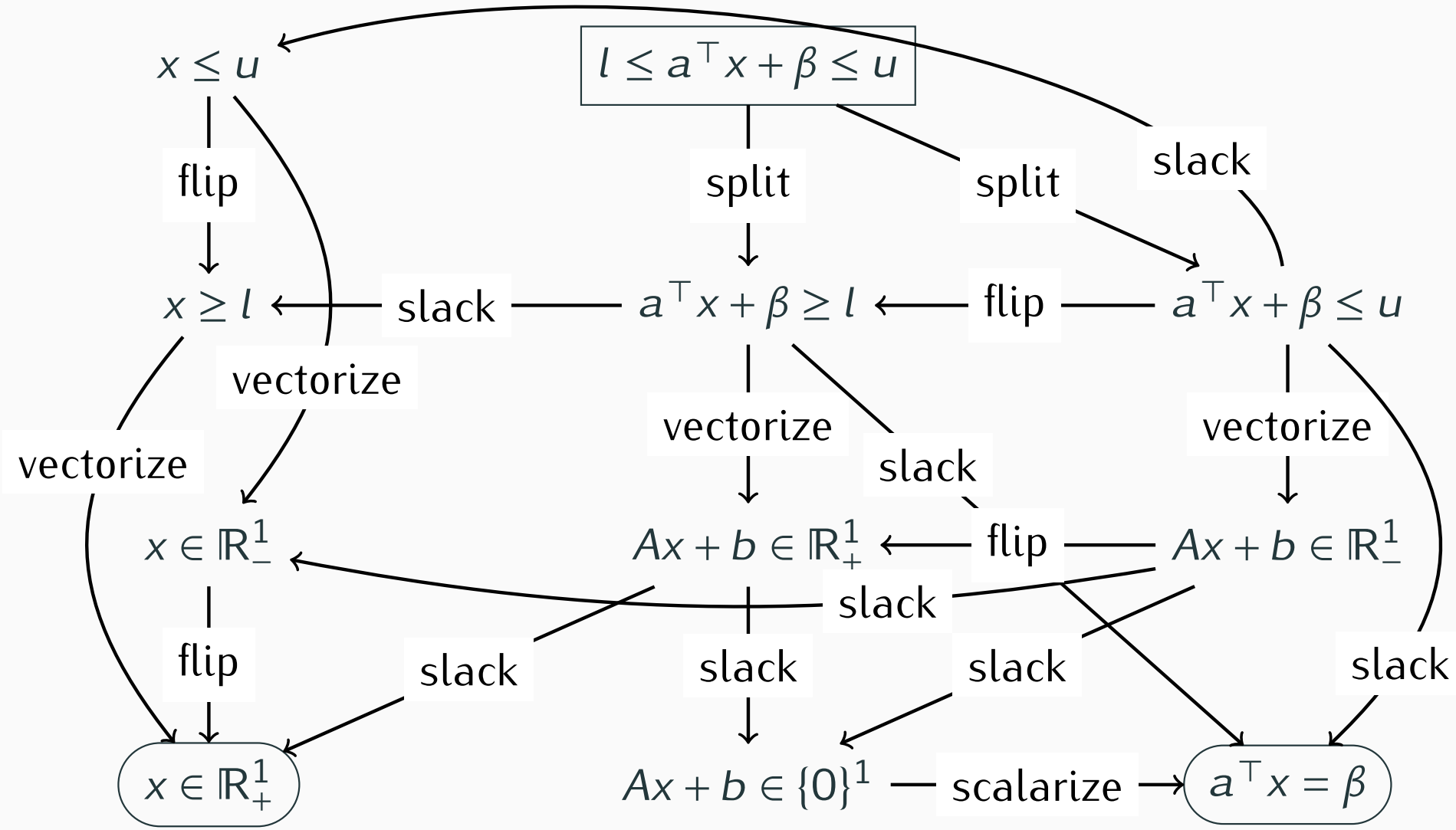
RSOC in Lagrangian:

$$\begin{aligned}
(1,s,x)\cdot(v,u,y) &= u(1,s,x)\cdot(\| x \|_2^2/2, 1, -x) \\
&= u(\| x \|_2^2/2 + s - \| x \|_2^2) \\
&= u(-\| x \|_2^2/2 + s) \\
&= u(-\| Uz \|_2^2/2 - a^\top z - \beta) \\
&= -u(z^T U^\top U z/2 + a^\top z + \beta)
\end{aligned}$$

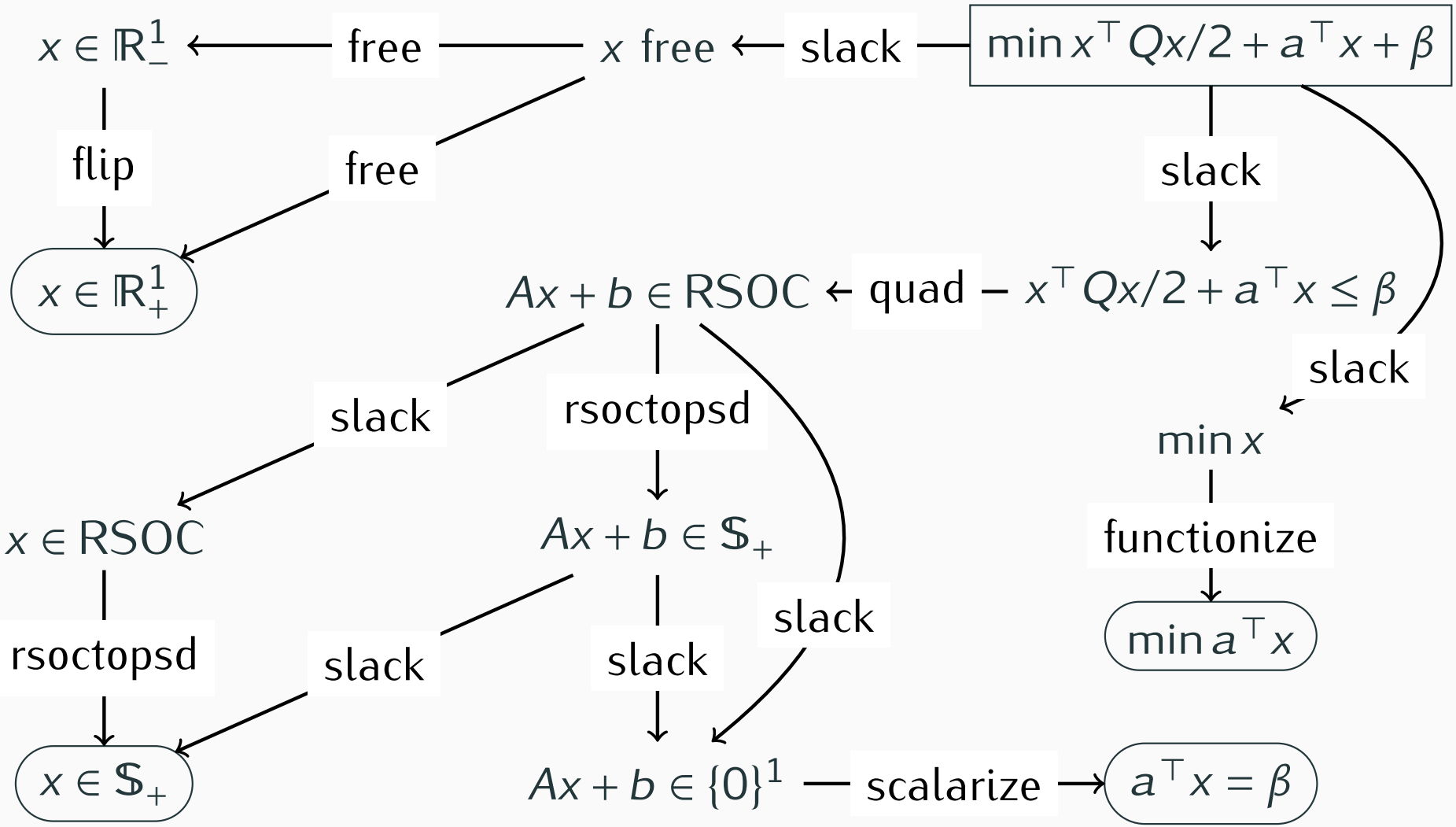
So the dual of the quadratic constraint is $-u$.

Combining bridges

Interval constraint for SDPA



Quadratic objective for SDPA



Automatic selection of bridges

Shortest path in directed Hypergraph

Nodes
Node for each set S (variable-in-S).

Node for each constraint F-in-S.

Node for each objective function F.

Types F and S are not limited to those defined in MOI.

Infinitely many nodes, we need to be lazy.

Edges
Each bridge defined possible infinitely many edges.

For each edge and ingoing node: outgoing nodes are

- variable-in-S created.
- constraints F-in-S created.
- objective F set.

Shortest Path Problem

Need to solve

$$d_{\mathcal{V}}(S) = 1 + \min_{B \in \mathcal{B}_{\mathcal{V}}(S)} \sum_{S' \in \mathcal{A}_{\mathcal{V}}(B,S)} d_{\mathcal{V}}(S') + \sum_{(F',S') \in \mathcal{A}_{\mathcal{C}}(B,S)} d_{\mathcal{C}}(F',S')$$

$$d_{\mathcal{C}}(F,S) = 1 + \min_{B \in \mathcal{B}_{\mathcal{C}}(F,S)} \sum_{S' \in \mathcal{A}_{\mathcal{V}}(B,F,S)} d_{\mathcal{V}}(S') + \sum_{(F',S') \in \mathcal{A}_{\mathcal{C}}(B,F,S)} d_{\mathcal{C}}(F',S')$$

$$d_{\mathcal{O}}(F) = 1 + \min_{B \in \mathcal{B}_{\mathcal{O}}(F)} \sum_{S' \in \mathcal{A}_{\mathcal{V}}(B,F)} d_{\mathcal{V}}(S') + \sum_{(F',S') \in \mathcal{A}_{\mathcal{C}}(B,F)} d_{\mathcal{C}}(F',S') + d_{\mathcal{O}}(\mathcal{A}_{\mathcal{O}}(B,F))$$

with

- $d_{\mathcal{V}}(S) = 0$ if constrained variables in S supported by optimizer.
- $d_{\mathcal{C}}(F,S) = 0$ if F-in-S constraints supported by optimizer.
- $d_{\mathcal{O}}(F) = 0$ if objective function F supported by the optimizer.

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: $d_{\mathcal{V}}$, $d_{\mathcal{C}}$, $d_{\mathcal{O}}$ are correct when defined.

constrained variable in S added by user
$\rightarrow$ generate $\mathcal{V}$ (resp. $\mathcal{C}$): list of needed new entries to compute in $d_{\mathcal{V}}$ (resp. $d_{\mathcal{C}}$).

F-in-S constraint added by user
$\rightarrow$ generate $\mathcal{V}$ (resp. $\mathcal{C}$): list of needed new entries to compute in $d_{\mathcal{V}}$ (resp. $d_{\mathcal{C}}$).

Objective function F constraint added by user
$\rightarrow$ generate $\mathcal{V}$ (resp. $\mathcal{C}$, $\mathcal{O}$): list of needed new entries to compute in $d_{\mathcal{V}}$ (resp. $d_{\mathcal{C}}$, $d_{\mathcal{O}}$).

Add target variable type

Algorithm 1 recursive add S

add S to $\mathcal{V}$
for $B \in \mathcal{B}_{\mathcal{V}}(S)$ **do**
  for $S' \in \mathcal{A}_{\mathcal{V}}(B, S)$ **do**
    if $d_{\mathcal{V}}(S')$ not defined **then**
      recursive add S'
    end if
  end for
  for $(F', S') \in \mathcal{A}_{\mathcal{C}}(B, S)$ **do**
    if $d_{\mathcal{C}}(F', S')$ not defined **then**
      recursive add F'-in-S'
    end if
  end for
end for

Add target constraint type

Algorithm 2 recursive add F-in-S

add F-in-S to $\mathcal{C}$
for $B \in \mathcal{B}_{\mathcal{C}}(F, S)$ **do**
  for $S' \in \mathcal{A}_{\mathcal{V}}(B, F, S)$ **do**
    if $d_{\mathcal{V}}(S')$ not defined **then**
      recursive add S'
    end if
  end for
  for $(F', S') \in \mathcal{A}_{\mathcal{C}}(B, F, S)$ **do**
    if $d_{\mathcal{C}}(F', S')$ not defined **then**
      recursive add F'-in-S'
    end if
  end for
end for

Modified Bellman-Ford algorithm

while $d_{\mathcal{V}}$ or $d_{\mathcal{C}}$ changed **do**
  for $S \in \mathcal{V}$, $B \in \mathcal{B}_{\mathcal{V}}(S)$ **do**
    $u \leftarrow 1 + \sum_{S' \in \mathcal{A}_{\mathcal{V}}(B,S)} d_{\mathcal{V}}(S') + \sum_{(F',S') \in \mathcal{A}_{\mathcal{C}}(B,S)} d_{\mathcal{C}}(F',S')$
    if $u < d_{\mathcal{V}}(S)$ **then**
      $d_{\mathcal{V}}(S) \leftarrow u$
    end if
  end for
  for F-in-$S \in \mathcal{C}$, $B \in \mathcal{B}_{\mathcal{C}}(F,S)$ **do**
    $u \leftarrow 1 + \sum_{S' \in \mathcal{A}_{\mathcal{V}}(B,F,S)} d_{\mathcal{V}}(S') + \sum_{(F',S') \in \mathcal{A}_{\mathcal{C}}(B,F,S)} d_{\mathcal{C}}(F',S')$
    if $u < d_{\mathcal{C}}(F,S)$ **then**
      $d_{\mathcal{C}}(F,S) \leftarrow u$
    end if
  end for
end while

Integration with JuMP

Extending JuMP macros

```
@constraint(model, [x + 1, x - y] in MOI.Zeros())
```

Implementation:

```
function build_constraint(
    _error::Function,
    func::Vector{<:AbstractJuMPScalar},
    set::MOI.AbstractVectorSet)
    return VectorConstraint(x, set)
end
```

Extending JuMP macros: Custom set

```
@constraint(model, [x + 1, x - y] in SecondOrderCone())
```

Implementation:

```
function build_constraint(_error::Function,
                          f::AbstractVector,
                          s::AbstractVectorSet)
    set = moi_set(s, length(f))
    return build_constraint(_error, f, set)
end
function moi_set(::SecondOrderCone, dim::Int)
  return MOI.SecondOrderCone(dim)
end
```

Extending JuMP macros: PSD cone

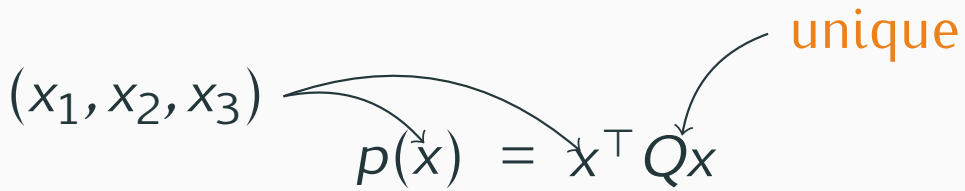
```
using LinearAlgebra # For Symmetric
@constraint(model, Symmetric([x + 1 x - y
                              x - y y]) in PSDCone())
```

Implementation:

```
function build_constraint(_error::Function,
                          Q::Symmetric,
                          ::PSDCone)
    n = LinearAlgebra.checksquare(Q)
    func = [Q[i, j] for j in 1:n for i in 1:j]
    set = MOI.PositiveSemidefiniteConeTriangle(n)
    VectorConstraint(func, set,
                     SymmetricMatrixShape(n))
end
```

Sum-of-Squares example

Nonnegative quadratic forms into sum of squares



$$x_1^2 + 2x_1x_2 + 5x_2^2 + 4x_2x_3 + x_3^2 = x^\top \begin{pmatrix} 1 & 1 & 0 \\ 1 & 5 & 2 \\ 0 & 2 & 1 \end{pmatrix} x$$

$$p(x) \geq 0 \ \forall x \iff Q \succeq 0$$

cholesky

$$(x_1 + x_2)^2 + (2x_2 + x_3)^2 \longleftarrow x^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix} x$$

Nonnegative polynomial into sum of squares

$$(x_1, x_2, x_3) \longrightarrow (x_1, x_1x_2, x_2) \longrightarrow p(x) = X^\top Q X \quad \textit{not} \text{ unique}$$

$$x_1^2 + 2x_1^2x_2 + 5x_1^2x_2^2 + 4x_1x_2^2 + x_2^2 = X^\top \begin{pmatrix} 1 & 1 & 0 \\ 1 & 5 & 2 \\ 0 & 2 & 1 \end{pmatrix} X$$

$$p(x) \geq 0 \ \forall x \impliedby Q \succeq 0 \qquad \Big\downarrow \text{cholesky}$$

$$(x_1 + x_1x_2)^2 + (2x_1x_2 + x_2)^2 \longleftarrow X^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix}^\top \begin{pmatrix} 1 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix} X$$

When is nonnegativity equivalent to sum of squares ?

Determining whether a polynomial is nonnegative is NP-hard.

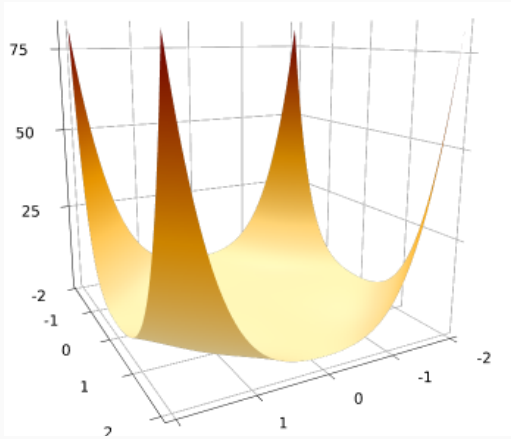
Hilbert 1888
Nonnegativity of $p(x)$ of n variables and degree $2d$ is equivalent to sum of squares in the following three cases:

- $n = 1$: Univariate polynomials
- $2d = 2$: Quadratic polynomials
- $n = 2$, $2d = 4$: Bivariate quartics

Motzkin 1967
First explicit example:

$$x_1^4 x_2^2 + x_1^2 x_2^4 + 1 - 3x_1^2 x_2^2 \geq 0 \quad \forall x$$

but is not a sum of squares.



Sum-of-Squares bridges

Polynomial $p \in \Sigma$ (p is SOS) iff $p = X^\top Q x$ with $Q \in \mathbb{S}_+$ (Q is PSD). Hence $\Sigma = A\mathbb{S}_+$.

SOSPolnomialBridge: Transformation of variable-in-Σ to variable-in-$\mathbb{S}_+$.

Transformation of contraint F-in-Σ: SlackBridge + SOSPolnomialBridge.

Result transformations

Constraint Attribute
Examples: ConstraintPrimal, ConstraintDual, ConstraintFunction, ConstraintSet, ...

Redirected to bridge when constraint is bridged.

New attributes:

- GramMatrixAttribute: Gram matrix Q indexed by X.
- MomentMatrixAttribute: Moment matrix index by X, dual of constraint $Q \in \mathbb{S}_+$.
- MomentsAttribute: Vector of moments, dual of constraint $p = X^\top Q X$.

Sum-of-Squares constraint macro

```
@constraint(model, p in SOSCone())
```

Implementation:

```
function JuMP.build_constraint(_error::Function, p,
                                cone::SOSCone; kws...)
    coefs = coefficients(p)
    monos = monomials(p)
    set = JuMP.moi_set(cone, monos; kws...)
    shape = PolyJuMP.PolynomialShape(monos)
    return PolyJuMP.bridgeable(
        JuMP.VectorConstraint(coefs, set, shape),
        JuMP.moi_function_type(typeof(coefs)),
        typeof(set)
    )
end
```

Reshaping

Reshaping sets

```
function reshape_set(set::MOI.AbstractScalarSet,
                     ::ScalarShape)
    return set
end
function reshape_set(
    ::MOI.PositiveSemidefiniteConeTriangle,
    ::SymmetricMatrixShape
)
    return PSDCone()
end
```

Reshaping polynomial sets

```
function JuMP.reshape_set(set::SOSPolynomialSet,
                          ::PolyJuMP.PolynomialShape)
    return set.cone
end
```

Reshaping results

```
function reshape_vector(vectorized_form::Vector{T},
        shape::SymmetricMatrixShape) where T
    matrix = Matrix{T}(undef, shape.side_dimension,
                       shape.side_dimension)
    k = 0
    for j in 1:shape.side_dimension
        for i in 1:j
            k += 1
            matrix[j, i] = matrix[i, j] =
                vectorized_form[k]
        end
    end
    return Symmetric(matrix)
end
```

Reshaping polynomial results

```
function JuMP.reshape_set(set::SOSPolynomialSet,
                          ::PolyJuMP.PolynomialShape)
    return set.cone
end
function JuMP.reshape_vector(x::Vector,
                             shape::PolynomialShape)
    return polynomial(x, shape.monomials)
end
function JuMP.reshape_vector(x::Vector,
                             shape::MomentsShape)
    return measure(x, shape.monomials)
end
```

Dual shape

In JuMP:

```
dual_shape(shape::AbstractShape) = shape
```

In PolyJuMP:

```
function JuMP.dual_shape(shape::PolynomialShape)
    return MomentsShape(shape.monomials)
end
function JuMP.dual_shape(shape::MomentsShape)
    return PolynomialShape(shape.monomials)
end
```

Conclusion

JuMP and MathOptInterface provide an optimization framework extensible by design.

Bridges allow to communicate custom structure to solvers from solver-independent models.