Recent Advances in Mixed Integer Programming Modeling and Computation

Juan Pablo Vielma
Massachusetts Institute of Technology

MIT Center for Transportation & Logistics.
Cambridge, Massachusetts, June, 2017.
(Nonlinear) Mixed Integer Programming (MIP)

\[
\begin{align*}
\text{min} & \quad f(x) \\
\text{s.t.} & \quad x \in C \\
& \quad x_i \in \mathbb{Z} \quad i \in I
\end{align*}
\]

Mostly *convex* \( f \) and \( C \).

NP-hard = Challenge Accepted!
50+ Years of MIP = Significant Solver Speedups

- Algorithmic Improvements (Machine Independent):
  - CPLEX
  - GUROBI

- Also convex nonlinear:
  - GUROBI
      \[V.,\] Dunning, Huchette, Lubin, 2015)
Widespread Use of Linear/Quadratic MIP Solvers

http://www.gurobi.com/company/example-customers
State of MIP Solvers

• Mature: Linear and Quadratic (Conic Quadratic/SOCP)
  – Commercial:
    - Gurobi
    - CPLEX
    - FICO
  – “Open Source”
    - SCIP
    - CBC
    - GLPK

• Emerging: Convex Nonlinear (e.g. SDP)
  – Open-Source + Commercial linear MIP Solver > Commercial
    - Knitro
    - Bonmin

Accessing MIP Solvers = Modelling Languages

• User-friendly algebraic modelling languages (AML):
  - Standalone and Fast
  - Based on General Language and Versatile

• Fast and Versatile, but complicated
  - Proprietary low-level C/C++ solver interphases.
  - C/C++ Coin-OR interphases and frameworks.

• 21st Century AMLs:
Outline

• JuMP ↔ julia overview.

• Advanced MIP formulations.

• Convex nonlinear MIP solvers.

• Optimal Control with Julia, JuMP and Pajarito.

• Other applications if time permits.
Why Julia and JuMP?

- **Julia** [http://julialang.org](http://julialang.org)
  - 21st century programming language
  - MIT licensed (and developed): free and open source
  - (Almost) as fast as C (LLVM JIT) and as easy as Matlab

- **JuMP**
  - Julia-based algebraic modelling language for optimization
  - Easy and natural syntax for linear, quadratic and conic (e.g. SDP) mixed-integer optimization.
  - Modular, extensible, easy to embed (e.g. simulation, visualization, etc.) and FAST.
  - Solver-independent access to advanced MIP features (e.g. cutting plane callbacks)
JuliaOpt’s packages can be loosely grouped into two sets. The first set are standalone Julia packages:

- **Overview of Packages**
- **Getting Started**

[PDF](http://dx.doi.org/10.1287/ijoc.2014.0623).

“Computing in Operations Research using Julia”

Case study

- Advanced language features like metaprogramming that enable interesting possibilities

Julia is a high-level, high-performance dynamic programming language for technical computing” ([http://julialang.org](http://julialang.org)). It is free (open source) and supports Windows, OSX, and Linux. It has a familiar syntax, works well with external libraries, is fast, and has

What is Julia?

1. You can find downloads and installation instructions for Julia
2. To install a JuliaOpt package, simply use the

```
julia> Pkg.add("JuMP")
```

```
julia> Pkg.add("Optim")
```

```
julia> Pkg.update() # Get latest package info
```

Open-source package directory. If you want to use an external

- solvers will automatically be downloaded and installed in your Julia

- solvers


We have a collection of JuliaOpt examples in the form of Jupyter

- notebooks, including:

- The source for these notebooks is available here ([https://github.com/JuliaOpt/juliaopt-notebooks](https://github.com/JuliaOpt/juliaopt-notebooks)).

We have a workshop ([https://www.youtube.com/watch?v=nnL7yLMVu6c](https://www.youtube.com/watch?v=nnL7yLMVu6c)) from JuliaCon 2015:

- [presentation](https://www.youtube.com/watch?v=7LNeR299q88) and

- Overviews

- Extensive Stack of Modelling and Solver Packages

- **JuliaOpt**

- **JuMP**

- **Convex.jl**

- **MathProgBase.jl**

- **Cbc.jl**

- **Clp.jl**

- **CPLEX.jl**

- **ECOS.jl**

- **GLPK.jl**

- **Gurobi.jl**

- **Ipopt.jl**

- **KNITRO.jl**

- **Mosek.jl**

- **LsqFit.jl**

- **Optim.jl**

- **SCS.jl**

- **AmplNLWriter.jl**

- **CoinOptServices.jl**

- **NLopt.jl**

- **JuMP extensions for**: block stochastic optimization, robust optimization, chance constraints, **piecewise linear optimization**, polynomial optimization, multi-objective optimization, discrete time stochastic optimal control, **sum of squares optimization**, etc.

- Useful Julia Packages: **Multivariate Polynomials**, etc.

Advanced MIP Formulations
Simple Formulation for Univariate Functions

\[ z = f(x) \]

\[
\begin{pmatrix}
  x \\
  z
\end{pmatrix} = \sum_{j=1}^{5} \begin{pmatrix}
  d_j \\
  f(d_j)
\end{pmatrix} \lambda_j
\]

\[ 1 = \sum_{j=1}^{5} \lambda_j, \quad \lambda_j \geq 0 \]

\[ y \in \{0, 1\}^4, \quad \sum_{i=1}^{4} y_i = 1 \]

\[ 0 \leq \lambda_1 \leq y_1 \]

\[ 0 \leq \lambda_2 \leq y_1 + y_2 \]

\[ 0 \leq \lambda_3 \leq y_2 + y_3 \]

\[ 0 \leq \lambda_4 \leq y_3 + y_4 \]

\[ 0 \leq \lambda_5 \leq y_4 \]

Size = \( O(\# \text{ of segments}) \)

Non-Ideal: Fractional Extreme Points
Mixed-Integer Models for Piecewise Linear Optimization

2. Modeling Piecewise Linear Functions

An appropriate way of modeling a piecewise linear function \( f: \mathcal{D} \rightarrow \mathbb{R}^n \) is to model its epigraph given by \( \text{epi}(f) = \{ (x, z) \in \mathcal{D} \times \mathbb{R} : f(x) \leq z \} \). For example, the epigraph of the function in Figure 2(a) is depicted in Figure 2(b).

For simplicity, we assume that the function domain \( \mathcal{D} \) is bounded and \( f \) is only used in a constraint of the form \( f(x) \leq 0 \) or as an objective function that is being minimized. We then need a model of \( \text{epi}(f) \) since \( f(x) \leq 0 \) can be modeled as \((x, z) \in \text{epi}(f), z \leq 0\) and the minimization of \( f \) can be achieved by minimizing \( z \) subject to \((x, z) \in \text{epi}(f)\).

For continuous functions we can also work with its graph, but modeling the epigraph will allow us to extend most of the results to some discontinuous functions and will simplify the analysis of formulation properties.

### Advanced Formulation for Univariate Functions

\[
\begin{align*}
z &= f(x) \\
\begin{pmatrix} x \\ z \end{pmatrix} &= \sum_{j=1}^{5} \begin{pmatrix} d_j \\ f(d_j) \end{pmatrix} \lambda_j \\
1 &= \sum_{j=1}^{5} \lambda_j, \quad \lambda_j \geq 0 \\
y &\in \{0, 1\}^2 \\
0 &\leq \lambda_1 + \lambda_5 \leq 1 - y_1 \\
0 &\leq \lambda_3 \leq y_1 \\
0 &\leq \lambda_4 + \lambda_5 \leq 1 - y_2 \\
0 &\leq \lambda_1 + \lambda_2 \leq y_2
\end{align*}
\]

Size = \(O(\log_2 \# \text{ of segments})\)

Ideal: Integral Extreme Points

- V. and Nemhauser 2011.
Formulation Improvements can be Significant

Transportation Problems with Concave Costs using GUROBI
More Advanced Small/Strong Formulation

- Modeling Finite Alternatives = Unions of Polyhedra

\[ x \in \bigcup_{i=1}^{n} P_i \subseteq \mathbb{R}^d \]

\[ \text{gr} (f) = \bigcup_{i=1}^{n} P_i \]
Many Techniques Based on Geometry/Graphs

- Somewhat complicated, but worth it!
- Also nonlinear MIP formulations.
- V. ’15; Huchette, Dey and V. ’16, Huchette and V. ‘16; Huchette and V. ‘17; V. ‘17a and V. ’17b.
Some Easily Accessible Through JuMP Extensions

• PiecewiseLinearOpt.jl (Huchette and V. 2017)

\[
\begin{align*}
\min & \quad \exp(x + y) \\
\text{s.t.} & \quad x, y \in [0, 1]
\end{align*}
\]

using JuMP, PiecewiseLinearOpt

```julia
using JuMP, PiecewiseLinearOpt
m = Model()
@variable(m, x)
@variable(m, y)

z = piecewiselinear(m, x, y, 0:0.1:1, 0:0.1:1, (u,v) -> exp(u+v))
@objective(m, Min, z)
```

Automatically select $\Delta$

Automatically construct formulation (easily chosen)
Convex Nonlinear MIP Solvers
Nonlinear MIP B&B Algorithms

• NLP (QCP) Based B&B
  – Few cuts = high speed.
  – Possible slow convergence.

• (Dynamic) LP Based B&B
  – Extended or Lifted relaxation.
  – Static relaxation
    • Mimic NLP B&B.
  – Dynamic relaxation
    • Standard LP B&B

\[
\begin{align*}
\text{max} & \quad \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} & \quad Ax + Dz \leq b, \\
& \quad g_i(x) \leq 0, \ i \in I, \quad x \in \mathbb{Z}^n \\
& \quad x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}
\end{align*}
\]
Second Order Conic or Conic Quadratic Problems

- Problems using Euclidean norm:
  - e.g. Portfolio Optimization Problems

\[
\begin{align*}
\text{max} & \quad \bar{a}x \\
\text{s.t.} & \quad \left\| Q^{1/2}x \right\|_2 \leq \sigma \\
& \quad \sum_{j=1}^{n} x_j = 1, \quad x \in \mathbb{R}_+^n \\
& \quad x_j \leq z_j \quad \forall j \in [n] \\
& \quad \sum_{j=1}^{n} z_j \leq K, \quad z \in \{0, 1\}^n
\end{align*}
\]

- $\bar{a}$ expected returns.
- $Q^{1/2}$ square root of covariance matrix.
- $K$ maximum number of assets.
- $\sigma$ maximum risk.
LP v/s NLP B&B for CPLEX v11 for n = 20 and 30

- Results from V., Ahmed and Nemhauser 2008.
Lifted or Extended Approximations

- Projection = multiply constraints.
- V., A. and N. 2008:
  - Extremely accurate, but static and complex approximation by Ben-Tal and Nemirovski
- V., Dunning, Huchette and Lubin 2016: Simple, dynamic and good approximation:

\[
\|y\|_2 \leq y_0 \quad \text{or} \quad y_i^2 \leq z_i \cdot y_0 \quad \forall i \in [n]
\]

\[
\sum_{i=1}^{n} z_i \leq y_0
\]

Image from Lipton and Regan, https://rjlipton.wordpress.com
CPLEX v12.6 for n = 20, 30, 40, 50 and 60
Gurobi v5.6.3 for n = 20, 30, 40, 50 and 60
All Major Solvers Now Implement Lifted LP

- First Talks:
  - SIAM Optimization (SIOPT), May 2014 ≈ two weeks coding.
  - IBM Thomas J. Watson Research Center, December 2014.


- GUROBI v6.5, October 2015.

- FICO v8.0, May 2016.

- SCIP v4.0, March 2017.
Comparison with specialized MISOCP solvers
Termination statuses and shifted geometric mean of solve time on 120 MISOCPs, for SCIP and CPLEX MISOCP solvers, and default MSD and iterative Pajarito solvers using CPLEX and MOSEK termination status counts

<table>
<thead>
<tr>
<th>solver</th>
<th>conv</th>
<th>wrong</th>
<th>not conv</th>
<th>limit</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCIP</td>
<td>78</td>
<td>1</td>
<td>0</td>
<td>41</td>
<td>43.36</td>
</tr>
<tr>
<td>CPLEX</td>
<td>96</td>
<td>3</td>
<td>5</td>
<td>16</td>
<td>14.30</td>
</tr>
<tr>
<td>Paj-iter</td>
<td>96</td>
<td>1</td>
<td>0</td>
<td>23</td>
<td>38.70</td>
</tr>
<tr>
<td>Paj-MSD</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>18.12</td>
</tr>
</tbody>
</table>
Flexible Architecture Thanks to Julia-Opt Stack

- Fastest Open Source MISOCP Solver!
- Pajarito can also solve MISDPs and MI-“EXP”
Optimal Control with Julia, JuMP and Pajarito

Joey Huchette ≈ two weeks for SIOPT ‘17
Trajectory Planning with Collision Avoidance

• Motivating: Steering a quadcopter through obstacles [Deits/Tedrake:2015]

• Position described by polynomials:
  – \((p^x(t), p^y(t))\) \(t \in [0,1]\)
  – avoid obstacles
  – initial/terminal conditions
  – minimize “jerk” of path

• Solution approach:
  – split domain into “safe” polyhedrons +
    discretize time into intervals
  – “smooth” piecewise polynomial trajectories
    in each interval, which chose polyhedron

variables = polynomials

Mixed-Integer Polynomial Programming
Disjunctive \textit{Polynomial} Optimization Formulation

Variables = Polynomials: \( \{ p_i : [T_i, T_{i+1}] \rightarrow \mathbb{R}^2 \}^N_{i=1} \)

\[
\begin{align*}
\min_{\mathbf{p}} & \quad \sum_{i=1}^{N} \| p_i'''(t) \|^2 \\
\text{s.t.} & \quad p_1(0) = X_0, \; p'(0) = X'_0, \; p'''(0) = X''_0 \\
& \quad p_N(1) = X_f, \; p'_N(1) = X'_f, \; p'''_N(1) = X''_f \\
& \quad p_i(T_{i+1}) = p_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \\
& \quad p'_i(T_{i+1}) = p'_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \\
& \quad p''_i(T_{i+1}) = p''_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \\
& \quad \bigvee_{r=1}^{R} [A^r p_i(t) \leq b^r] \quad \text{for } t \in [T_i, T_{i+1}] \quad \forall i \in \{1, \ldots, N\}
\end{align*}
\]

Avoid Collision = Remain in Safe Regions
**Diisjunctive Polynomial Optimization Formulation**

Variables = Polynomials: \( \{ p_i : [T_i, T_{i+1}] \rightarrow \mathbb{R}^2 \}_{i=1}^{N} \)

\[
\min \sum_{i=1}^{N} \| p_i'''(t) \|^2
\]

s.t.
- \( p_1(0) = X_0, p'(0) = X_0', p''(0) = X_0'' \)
- \( p_N(1) = X_f, p'_N(1) = X_f', p''_N(1) = X_f'' \)
- \( p_i(T_{i+1}) = p_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \)
- \( p'_i(T_{i+1}) = p'_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \)
- \( p''_i(T_{i+1}) = p''_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \)

\( b_j^r + M_j^r (1 - z_{i,r}) - A_j^r p_i(t) \geq 0 \quad \text{for } t \in [T_i, T_{i+1}] \quad \forall i, j, r \)

\( \sum_{r=1}^{R} z_{i,r} = 1 \quad \forall i, z \in \{0, 1\}^{N \times R} \)

Avoid Collision = Remain in Safe Regions
**Disjunctive Polynomial Optimization Formulation**

**Mixed-Integer Sum-of-Squares**

Variables = Polynomials: \( \{ p_i : [T_i, T_{i+1}] \rightarrow \mathbb{R}^2 \} \)

\[
\min_p \sum_{i=1}^{N} \| p_i'''(t) \|^2
\]

s.t. \( p_1(0) = X_0, p_1'(0) = X_0', p_1''(0) = X_0'' \)
\( p_N(1) = X_f, p_N'(1) = X_f', p_N''(1) = X_f'' \)
\( p_i(T_{i+1}) = p_{i+1}(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \)
\( p_i'(T_{i+1}) = p_{i+1}'(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \)
\( p_i''(T_{i+1}) = p_{i+1}''(T_{i+1}) \quad \forall i \in \{1, \ldots, N\} \)
\[
b_j^r + M_j^r (1 - z_{i,r}) - A_j^r p_i(t) \text{ is SOS for } t \in [T_i, T_{i+1}] \quad \forall i, j, r
\]
\[
\sum_{r=1}^{R} z_{i,r} = 1 \quad \forall i, z \in \{0, 1\}^{N \times R}
\]

**Initial/Terminal Conditions**

**Interstitial Smoothing Conditions**

**Avoid Collision = Remain in Safe Regions**
• **Sufficient** condition for non-negative polynomial:
  – Sum of Squares: \( f(x) = \sum_{i} g_i^2(x) \)
  – SDP representable for fixed degree:
    degree \( \leq k \) \( \rightarrow \) \((k - 1) \times (k - 1)\) matrices

• **MI-SOS:**
  – Low degree polynomials (\( \leq 3 \)):
    • MI-SOCP: solvable by Gurobi/CPLEX
    • Deits/Tedrake: 2015
  – Higher degree polynomials:
    • MI-SDP: solvable by Pajarito
Results for 9 Regions and 8 time steps

- Infeasible for degree $\leq 3$ (MI-SOCP)
- Pajarito results for degree 5:

First Feasible Solution: 58 seconds
Optimal Solution: 651 seconds
model = SOSModel(solver=PajaritoSolver())

@polyvar(t)
Z = monomials([t], 0:r)

@variable(model, H[1:N,boxes], Bin)

p = Dict()
for j in 1:N
    @constraint(model, sum(H[j,box] for box in boxes) == 1)
    p[(:x,j)] = @polyvariable(model, _, Z)
    p[(:y,j)] = @polyvariable(model, _, Z)
    for box in boxes
        xl, xu, yl, yu = box.xl, box.xu, box.yl, box.yu
        @polyconstraint(model, p[(:x,j)] >= Mxl + (xl-Mxl)*H[j,box], domain = (t >= T[j] && t <= T[j+1]))
        @polyconstraint(model, p[(:x,j)] <= Mxu + (xu-Mxu)*H[j,box], domain = (t >= T[j] && t <= T[j+1]))
        @polyconstraint(model, p[(:y,j)] >= Myl + (yl-Myl)*H[j,box], domain = (t >= T[j] && t <= T[j+1]))
        @polyconstraint(model, p[(:y,j)] <= Myu + (yu-Myu)*H[j,box], domain = (t >= T[j] && t <= T[j+1]))
    end
end

for ax in (:x,:y)
    @constraint(model, p[(ax,1)][[0], [t]] == X₀[ax])
    @constraint(model, differentiate(p[(ax,1)],t)[[0], [t]] == X₀ʹ[ax])
    @constraint(model, differentiate(p[(ax,1)],t,2)[[0], [t]] == X₀ʹʹ[ax])
    for j in 1:N-1
        @constraint(model, p[(ax,j)][[T[j+1]], [t]] == p[(ax,j+1)][[T[j+1]], [t]])
        @constraint(model, differentiate(p[(ax,j)],t)[[T[j+1]], [t]] == differentiate(p[(ax,j+1)],t)[[T[j+1]], [t]])
        @constraint(model, differentiate(p[(ax,j)],t,2)[[T[j+1]], [t]] == differentiate(p[(ax,j+1)],t,2)[[T[j+1]], [t]])
    end
    @constraint(model, p[(ax,N)][[1], [t]] == X₁[ax])
    @constraint(model, differentiate(p[(ax,N)],t)[[1], [t]] == X₁ʹ[ax])
    @constraint(model, differentiate(p[(ax,N)],t,2)[[1], [t]] == X₁ʹʹ[ax])
end

@variable(model, γ[Keys(p)] ≥ 0)
for (key,val) in p
    @constraint(model, γ[key] ≥ norm(differentiate(val, t, 3)))
end
@objective(model, Min, sum(γ))

function eval_poly(r)
    for i in 1:N
        if T[i] <= r <= T[i+1]
            return PP[(:x,i)][[r], [t]], PP[(:y,i)][[r], [t]]
            break
        end
    end
end
using SFML

const window_width = 800
const window_height = 600

window = RenderWindow("Helicopter", window_width, window_height)
event = Event()

rects = RectangleShape[]
for box in boxes
    rect = RectangleShape()
xl = (window_width/M)*box.xl
xu = (window_width/M)*box.xu
yl = window_height*(domain.yu-box.yl)
yu = window_height*(domain.yu-box.yu)
set_size(rect, Vector2f(xu-xl, yu-yl))
set_position(rect, Vector2f(xl, yl))
set_fillcolor(rect, SFML.white)
push!(rects, rect)
end

type Helicopter
shape::CircleShape
past_path::Vector{Vector2f}
path_func::Function
end

const radius = 10
heli = Helicopter(CircleShape(), Vector2f[Vector2f(X₀[:x]*window_width, X₀[:y]*window_height)], eval_poly)
set_position(heli.shape, Vector2f(window_width/2, window_height/2))
set_radius(heli.shape, radius)
set_fillcolor(heli.shape, SFML.red)
set_origin(heli.shape, Vector2f(radius, radius))

function update_heli!(heli::Helicopter, tm)
    (_x,_y) = heli.path_func(tm)
x = window_width / M * _x
y = window_height * (1-_y)
pt = Vector2f(x,y)
set_position(heli.shape, pt)
    # move(heli.shape, pt-heli.past_path[end])
push!(heli.past_path, pt)
    get_position(heli.shape)
    nothing
end

const maxtime = 10.0
make_gif(window, window_width, window_height, 1.05*maxtime, "foobarbat.gif", 0.05)
clock = Clock()
restart(clock)

while isopen(window)
    frametime = as_seconds(get_elapsed_time(clock))
    @show normalizedtime = Tmin + (frametime / maxtime)*(Tmax-Tmin)
    (normalizedtime >= Tmax) && break
    while pollevent(window, event)
        if get_type(event) == EventType.CLOSED
            close(window)
        end
    end
    clear(window, SFML.blue)
    for rect in rects
        draw(window, rect)
    end
    update_heli!(heli, normalizedtime)
    draw(window, heli.shape)
    display(window)
end
Helicopter Game / Flappy Bird

- 60 horizontal segments, obstacle every 5 = 80 sec. to opt.
Summary

• Advances in MIP = Advanced Formulations + Advanced Solvers + Easy Access Through JuMP

• More information:
  – Advanced Formulations: 15.083 +
    • Mixed integer linear programming formulation techniques. V. ’15.
  – Algorithms/Solvers: 15.083 +
    • M. Lubin’s thesis defense: Monday, June 5, 1:00 PM, E62-550
  – Julia and JuMP: 15.083 + webpages +

• 15.083: Integer Programming and Combinatorial Optimization
  – Spring 2018: Formulations + Algorithms +
• MIP Formulations Survey:

• Other advanced MIP formulation techniques:
• Convex Nonlinear MIP Solvers:
References

- Julia:
  - [https://julialang.org](https://julialang.org)

- JuMP: