Building Coluna.jl, a branch-cut-and-price framework in Julia

Natacha Javerzat, Guillaume Marques, Vitor Nesello, Artur Pessoa, Ruslan Sadykov, and François Vanderbeck
A Julia framework to optimize block-structured MILP problems.

Coluna.jl

- Dantzig-Wolfe decomposition
- Benders decomposition

- Linking constraints
- Linking variables

Tractable subproblems

© Atoptima
Generalized Assignment Problem

Let:

- $x_{mj} = 1$ if job $j$ assigned to machine $m$;
- $0$ otherwise
- $c_{mj}$ cost
- $w_{mj}$ weight
- $Q_m$ capacity of machine $m$

```
Let:
x_{mj} = 1 if job j assigned to machine m;
0 otherwise
c_{mj} cost
w_{mj} weight
Q_m capacity of machine m
```

Example of solution

```
m = 3
m = 2
m = 1

Example of solution
```

```
min \sum_{m \in M} \sum_{j \in J} c_{mj} x_{mj}

s.t. \sum_{j \in J} x_{mj} \geq 1 \quad j \in J

\sum_{i \in I} w_{mj} x_{mj} \leq Q_m \quad m \in M

x_{mj} \in \{0, 1\}
```

© Atoptima
using JuMP, GLPK

M = 1:3 # Machines
J = 1:15 # Jobs

model = Model(GLPK.Optimizer)

@variable(model, x[m in M, j in J], Bin)

@constraint(model, cov[j in J], sum(x[m, j] for m in M) >= 1)
@constraint(model, knp[m in M],
            sum(w[m, j] * x[m, j] for j in J) <= Q[m])

@objective(model, Min, sum(c[m, j] * x[m, j] for m in M, j in J));
using JuMP, GLPK

M = 1:3  # Machines
J = 1:15  # Jobs

model = Model(GLPK.Optimizer)

@variable(model, x[m in M, j in J], Bin)

@constraint(model, cov[j in J], sum(x[m, j] for m in M) >= 1)
@constraint(model, knp[m in M],
    sum(w[m, j] * x[m, j] for j in J) <= Q[m])

@objective(model, Min, sum(c[m, j] * x[m, j] for m in M, j in J));
using JuMP, GLPK, BlockDecomposition, Coluna

@axis(M, 1:3) # Annotated machines
J = 1:15 # Jobs
coluna = optimizer_with_attributes(Coluna.Optimizer, ...)

model = BlockModel(coluna)

@variable(model, x[m in M, j in J], Bin)

@constraint(model, cov[j in J], sum(x[m, j] for m in M) >= 1)
@constraint(model, knp[m in M],
    sum(w[m, j] * x[m, j] for j in J) <= Q[m])

@objective(model, Min, sum(c[m, j] * x[m, j] for m in M, j in J));
@dantzig_wolfe_decomposition(model, decomposition, M)
Generalized Assignment Problem

```plaintext
using JuMP, HiGHS

M = 1:3 # Machines
J = 1:15 # Jobs

model = Model(HiGHS.Optimizer)

@variable(model, x[m in M, j in J], Bin)
@constraint(model, cov[j in J], sum(x[m, j] for m in M) >= 1)
@constraint(model, knp[m in M],
            sum(w[m,j] * x[m, j] for j in J) <= Q[m])
@objective(model, Min, sum(c[m, j] * x[m, j] for m in M, j in J));

using JuMP, HiGHS, BlockDecomposition, Coluna

@axis(M, 1:3) # Annotated machines
J = 1:15 # Jobs

coluna = optimizer_with_attributes(Coluna.Optimizer, ...)

model = BlockModel(coluna)

@variable(model, x[m in M, j in J], Bin)
@constraint(model, cov[j in J], sum(x[m, j] for m in M) >= 1)
@constraint(model, knp[m in M],
            sum(w[m,j] * x[m, j] for j in J) <= Q[m])
@objective(model, Min, sum(c[m, j] * x[m, j] for m in M, j in J));
@dantzig_wolfe_decomposition(model, decomposition, M)
```
Coluna environment

- BlockDecomposition
- JuMP
- MOI
- Coluna
- HiGHS
- GLPK
- Gurobi
- CPLEX
- Callbacks
- Custom algorithms
When calling **BlockModel**: 

```julia
function optimize_hook!(m::JuMP.Model)
    # [...] automatic decomposition if activated
    register_decomposition(m)
    return JuMP.optimize!(m, ignore_optimize_hook = true)
end
```

When calling `@dantzig_wolfe_decomposition`:

```
model.ext[:decomposition_tree] =
```

```
```

© Atoptima
Danztig-Wolfe reformulation

When calling JuMP.optimize!

- BlockDecomposition hook annotates all the variables and constraints
- Coluna receives the formulation and the annotations from JuMP/MathOptInterface
- Coluna reformulates the formulation

```plaintext
mutable struct Reformulation
    parent::Formulation{Original}
    master::Formulation{DwMaster}
    dw_pricing_subprs::Dict/FormId, Formulation{DwSp})
# [...]
end
```

\[
\text{cov}[1:15] \sum_{m \in M} \sum_{k \in K^m} \bar{x}_m^k \lambda_k^m \geq 1 \quad j \in J
\]

\[
\sum_{k} \lambda_k^m \leq 1 \quad m \in M
\]

\[
\lambda_k^m \geq 0
\]

\[\lambda_k^m\] is nb of times solution \(\bar{x}_k^m\) is used.

master formulation

pricing subproblems

\[\text{knp}[1], \text{knp}[2], \text{knp}[3]\]
Benders reformulation

- Use an `@axis` to define the index-set of subproblems
- Write the compact model with JuMP
- Call the `@benders_decomposition` macro

\[
\begin{align*}
\text{min} & \quad cy + fx \\
\text{s.t.} & \quad Ay \geq a \\
& \quad Ty + Dx \geq b \\
& \quad Ex \geq e \\
& \quad x, y \geq 0
\end{align*}
\]

original formulation

\[
\begin{align*}
\text{min} & \quad cy + \sum_k \eta^k \\
\text{s.t.} & \quad Ay \geq a \\
& \quad < \text{Benders cuts} > \\
& \quad \eta \in \mathbb{R}, y \geq 0
\end{align*}
\]

master formulation (1st level)

\[
\begin{align*}
\text{min} & \quad fx \\
\text{s.t.} & \quad Dx \geq b - T\bar{y} \\
& \quad Ex \geq e \\
& \quad x \geq 0
\end{align*}
\]

separation subproblems (2nd level)

where \( \bar{x} \) is the first-level solution
Top algorithm of Coluna

```python
bcp = Coluna.Algorithm.TreeSearchAlgorithm(
    conqueralg = ColCutGenConquer(
        colgen = ColumnGeneration(
            max_nb_iterations = 1000
        ),
        primal_heuristics = [DefaultRestrictedMasterHeuristic()],
    ),
    cutgen = CutCallbacks()
    dividealg = StrongBranching(
        phases = [...]
        rules = [...]
        selection_criterion = MostFractionalCriterion()
    ),
    explorestrategy = DepthFirstStrategy(),
    maxnumnodes::Int = 50,
    branchingtreefile = "tree.dot"
)```

Diagram:
- `TreeSearchAlgorithm`
  - `StrongBranching`
    - `DepthFirstStrategy`
  - `ColCutGenConquer`
    - `ColCutGenConquer`
  - `ColumnGeneration`
  - `SolveIpForm`
  - `CutCallbacks`
  - `SolveLpForm`
  - `SolveIpForm`
Branch-cut-and-price output

Coluna
Version 0.7.0 | https://github.com/atoptima/Coluna.jl

***********************************************************************
**** B&B tree root node
**** Local DB = -Inf, global bounds: [-Inf, Inf], time = 0.00 sec.
***********************************************************************

<st= 1> <it=  1> <et= 0.00> <mst= 0.00> <sp= 0.00> <cols= 5> <al= 0.00> <DB=-129136.2200> <mlp=100000.0000> <PB=Inf>
<st= 1> <it=  2> <et= 0.01> <mst= 0.00> <sp= 0.00> <cols= 5> <al= 0.00> <DB=-148537.5000> <mlp=30340.5500> <PB=Inf>
<st= 1> <it=  3> <et= 0.01> <mst= 0.00> <sp= 0.00> <cols= 5> <al= 0.00> <DB=-139032.5600> <mlp=20319.9200> <PB=Inf>
<st= 1> <it=  4> <et= 0.01> <mst= 0.00> <sp= 0.00> <cols= 5> <al= 0.00> <DB=-124563.8557> <mlp= 3253.8629> <PB=Inf>
<st= 1> <it=  5> <et= 0.01> <mst= 0.00> <sp= 0.00> <cols= 5> <al= 0.00> <DB=  313.9200> <mlp=  410.1222> <PB=Inf>

Cut separation callback adds 0 new essential cuts and 1 new facultative cuts.
avg. viol. = 0.54, max. viol. = 0.54, zero viol. = 0.

<st= 1> <it=  1> <et= 1.71> <mst= 0.00> <sp= 0.00> <cols= 2> <al= 0.00> <DB=  395.4634> <mlp=  399.3076> <PB=Inf>
<st= 1> <it=  2> <et= 1.71> <mst= 0.00> <sp= 0.00> <cols= 1> <al= 0.00> <DB=  398.1515> <mlp=  399.0815> <PB=Inf>
<st= 1> <it=  3> <et= 1.71> <mst= 0.00> <sp= 0.00> <cols= 0> <al= 0.00> <DB=  398.9050> <mlp=  398.9050> <PB=Inf>

Cut separation callback adds 0 new essential cuts and 1 new facultative cuts.
avg. viol. = 0.50, max. viol. = 0.50, zero viol. = 0.

Cut separation callback adds 0 new essential cuts and 1 new facultative cuts.

<st= 1> <it=  1> <et= 1.77> <mst= 0.00> <sp= 0.00> <cols= 5> <al= 0.00> <DB=-14668.8378> <mlp= 1528.5356> <PB=Inf>
<st= 1> <it=  2> <et= 1.77> <mst= 0.00> <sp= 0.00> <cols= 4> <al= 0.00> <DB=  433.7789> <mlp=  444.2689> <PB=Inf>
<st= 1> <it=  3> <et= 1.77> <mst= 0.00> <sp= 0.00> <cols= 3> <al= 0.00> <DB=  434.5150> <mlp=  440.3950> <PB=Inf>
<st= 1> <it=  4> <et= 1.78> <mst= 0.00> <sp= 0.00> <cols= 3> <al= 0.00> <DB=  436.0250> <mlp=  440.3950> <PB=Inf>
<st= 1> <it=  5> <et= 1.78> <mst= 0.00> <sp= 0.00> <cols= 4> <al= 0.00> <DB=  433.6600> <mlp=  440.3950> <PB=Inf>
<st= 1> <it=  6> <et= 1.78> <mst= 0.00> <sp= 0.00> <cols= 2> <al= 0.00> <DB=  438.6150> <mlp=  440.3950> <PB=Inf>
<st= 1> <it=  7> <et= 1.78> <mst= 0.00> <sp= 0.00> <cols= 0> <al= 0.00> <DB=  440.3950> <mlp=  440.3950> <PB=Inf>

Cut separation callback adds 0 new essential cuts and 0 new facultative cuts.

***********************************************************************
**** B&B tree node N°3, parent N°1, depth 1, 1 untreated node
**** Local DB = 440.3950, global bounds: [440.3950, 444.4000], time = 1.78 sec.
**** Branching constraint: x[3,1]<0.0
***********************************************************************

<st= 1> <it=  1> <et= 1.78> <mst= 0.00> <sp= 0.00> <cols= 4> <al= 0.00> <DB=-433.7450> <mlp= 443.7450> <PB=Inf>
<st= 1> <it=  2> <et= 1.79> <mst= 0.00> <sp= 0.00> <cols= 3> <al= 0.00> <DB=  438.6300> <mlp=  441.2575> <PB=Inf>
Algorithms provided by Coluna

**Column-and-cut generation**
- Column Generation (API)
  - Auto smoothing stabilization
  - Identical subproblems
  - Multi-stage
- Pricing callback
  - Initial columns callback
  - Lazy-cut callback
  - User-cut callback
    - Robust cuts
    - Non-robust cuts
  - Restricted master heuristic
- Columns cleanup

Based on [Pessoa et al., 2018], [Poggi de Aragão and Uchoa 2003], [Jepsen et al., 2006]

**Benders cut generation (API)**
- Multi-cut
- Integration with B&B
- Stabilization

Based on [Bonami et al. 2020]
Algorithms provided by Coluna

**Strong Branching**
- Selection criterion (API)
  - Most fractional
  - First found
  - Least Fractional
  - Closest To Non Zero Integer
- Scores (API)
  - Product score
  - Tree Depth score
- Rules (API)
  - Single variable

**Tree search (API)**
- Depth-First Search
- Best-Bound Search
- *LDS (Draft PR)*

**Presolve (dev)**
- Single row elimination
- Variable fixing
- Variable bounds strengthening
- Propagation between master/pricing subproblems

Inspired from [Pecin et al., 2017], [Le Bodic & George Nemhauser, 2017], [Achterberg 2007], [Kullmann, 2009]

Inspired from [Achterberg et al., 2020] & [Unpublished work]
Coluna architecture

MOI

solver.jl

DynamicSparse Arrays

MathProg

Algorithm (implementations)

ColunaBase

ColGen

Benders

Branching

TreeSearch
Algorithms architecture

Generic functions + Interface

ColGen

<Generic functions>
run_colgen
run_colgen_phase
run_colgen_iteration

<Interface>
AbstractColGenCtx

Default implementation

Algorithm

ColGenCtx

PrinterColGenCtx
Algorithms architecture

Generic functions + Interface

« textbook algorithm »

algorithmic logic

documented interface

@mustimplement "ColGen" update_reduced_costs!(
    context, phase, red_costs
) = nothing

Default implementation

« algorithm details »

algorithm runs
with the formulation representation
provided by MathProg
Algorithms architecture

Generic functions + Interface

« textbook algorithm »

algorithmic logic

documented interface

Default implementation

« algorithm details »

algorithm runs with the formulation representation provided by MathProg

Modular & easy to test

- unit tests of the logic with mocks
- e2e tests

- unit tests
- integration tests with MathProg
Algorithms architecture

Generic functions + Interface

ColGen

<Generic functions>
run_colgen
run_colgen_phase
run_colgen_iteration

<Interface>
AbstractColGenCtx

Default implementation

Algorithm

ColGenCtx

PrinterColGenCtx

MockColGenTestCtx
- All features for the textbook Branch-Cut-and-Price are available
- Now focus on performance features
When should I give it a try?

- Julia as high-level language of your project
- Trying Dantzig-Wolfe/Benders decomposition on your problem (POC)
- Prototyping (features: callbacks, advanced parameters)
- Building a viable project (confirmed Julia devs who feel comfortable with theory)
Coluna.jl

Open-source
https://github.com/atoptima/Coluna.jl

Registered
] add BlockDecomposition, Coluna

Contributors
Natacha Javerzat, Guillaume Marques, Vitor Nesello, Artur Pessoa, Ruslan Sadykov, François Vanderbeck
References


Level up your decision-making with optimization intelligence