Unlocking the Power of Google OR-Tools with MathOptInterface.jl

Warren Ochibobo, Google Nairobi
Thibaut Cuvelier, Google Paris — tcuvelier@google.com

https://developers.google.com/optimization

Google's OR Team

Routing: logistics, Google Street View

- Open-source solver
- B2B API: GMPRO

Concrete applications:

- Workforce scheduling (API)
- Shipping network design (API)

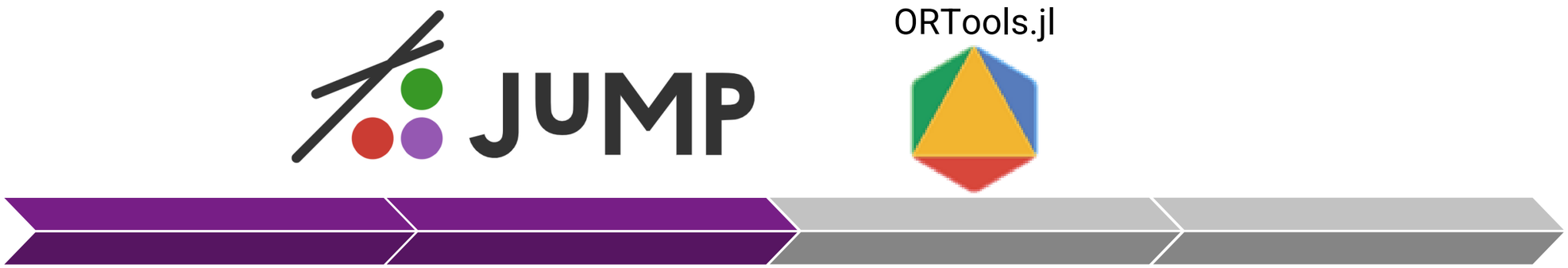
Low-level solvers:

- Glop: LP solver (simplex)
- CP-SAT: CP solver using SAT, won more than 10 gold medals at the MiniZinc competition
- PDLP: LP solver (first order)
- MathOpt: modelling layer

One open-source product: OR-Tools

- Accessible in many languages: C++, Python, Java, C#

Table of Contents



Problem model	JuMP	Solver	Solution
Scheduling, routing, etc.	Julia modelling layer: describe your problem	C++ code that solves the model	

This is a least-squares model in JuMP

```
m, n = size(A)
model = Model(Ipopt.Optimizer)
@variable(model, x[1:n])
@variable(model, residuals[1:m])
@constraint(model, residuals == A * x - b)
@objective(model, Min, sum(residuals.^2))
optimize!(model)
```

This is (part of) a least-squares model with MathOptInterface

```
model = …
x = MOI.add_variables(model, n)
residuals = MOI.add_variables(model, m)
MOI.add_constraint(
    model,
    MOI.ScalarAffineFunction(
        [MOI.ScalarAffineTerm(1.0, residuals[1]),
         MOI.ScalarAffineTerm(A[1, 1], x[1])],
        b[1],
    ),
    MOI.EqualTo(0.0),
)
```

This is (part of) a least-squares model with MathOpt (C++)

```
math_opt::Model model;
std::vector<math_opt::Variable> x;
for (int i = 0; i < n; ++i) {
  x.push_back(model.AddContinousVariable());
}
for (int j = 0; j < m; ++j) {
  LinearExpression expr(b[0]);
  expr.AddInnerProduct("A[j, :]", x);
  model.AddLinearConstraint(expr == 0.0)
}
```

OR-Tools & MathOpt

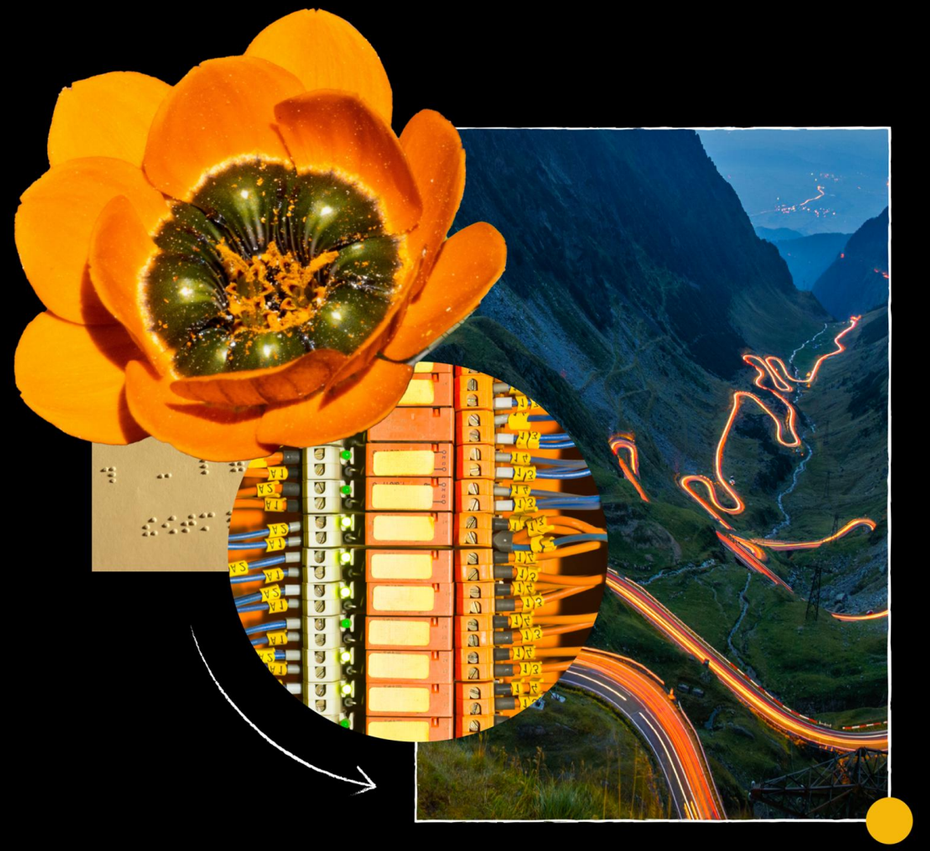
Different from MathOptInterface.jl!

- MathOpt is part of OR-Tools
- C++ modelling layer
  - Similar to MathOptInterface.jl

Many interfaces:

- Protocol Buffers
  - A kind of binary JSON/XML
- C++
  - Higher performance

Clipart generated by Gemini



Hence: ORTools.jl!

- Wraps OR-Tools' MathOpt
  - Including all of OR-Tools' solvers: CP-SAT, Glop, PDLP
  - Including other supported solvers, such as Gurobi, GPLK, or SCIP
- JuMP users thus have access to many new solvers!

Clipart generated by Gemini

Hence: ORTools.jl!

- Quite a technical challenge!
- C++: hard to access from Julia
  - OR-Tools is a C++-first library: bypass C++ via ProtoBuf and a small C interface
- Julia side:
  - ProtoBuf.jl: not as easy to use as in other languages
  - MathOptInterface.jl: not an easy API!
    But we expected this one ☺

Clipart generated by Gemini

CMake and binary building

- CMake is the de-facto standard to build C++ code

- **OR-Tools has a large CMake-based infrastructure** to cross-compile all our binaries
  - Quite custom in some parts
  - Some code runs on the host (code generators), a potentially different platform than the target

CMake and binary building

- **Julia has Yggdrasil/BinaryBuilder.jl** to handle native dependencies (“JLLs”)
  - Large infrastructure for cross-compilation
  - Automates a lot of cross-compilation
    Nice if you have no code generator!

- BinaryBuilder.jl defines ARM as without crypto extensions
  - Some OR-Tools dependencies need them
  - Most people have them
  - If you run an optimisation solver: you typically have them

CMake and binary building

- Solution? *(Work in progress!)*
  - Use JLLs whenever possible (ORTools_jll.jl)
  - Download prebuilt dependencies from GitHub otherwise (ORToolsBinaries.jl, temporary name)
    - Assume ARM means crypto extensions
    - Currently, GitHub has 114 binary packages for v9.14!
    - Downloads the binaries upon package building, like in the good old days (pre-JLLs)
  - Use extension packages to find binaries
  - Bonus: no need to have binaries to use the package!

C++ and Protocol Buffers

- Google uses Protocol Buffers for mostly everything, especially RPC

- MathOpt has three main interfaces:
  - **Protocol Buffers**: pure data interface; used for the Java interface of MathOpt
  - **C++**: user-facing interface, best performance (no intermediate data structure)
  - **Elemental in C**: in development when we started ORTools.jl; now used for the Python interface of MathOpt

C++ and Protocol Buffers

- **ProtoBuf.jl**: code generator for Julia, community-maintained
  - Few public users, thus reliability unknown
  - **In practice?** Works very well! A bit hard to have the compiler working due to dependencies

- **Missing feature**: mutability
  - All languages have some kind of mutability (sometimes through builders)
  - ORTools.jl builds many objects incrementally, due to the structure of MathOptInterface.jl
  - Hence, duplication of generated code

MathOptInterface

- Do we need an **incremental interface or a one-shot** one?
  - MathOpt's **natural API is closer to incremental...**
    but we use Protocol Buffers, closer to one-shot
  - Incremental seems **less daunting** to implement, no single huge function
    You can do it step by step: variables, constraints, attributes
  - **We can update** the underlying data structure: better fit for incremental
  - Solving a model requires **serialising** the model (Protocol Buffers), then deserialising it on the C++ side: better fit for one-shot

How do we develop OR-Tools?

- And… huh… semver anyone?

- **Julia mandates semver for packages**
- Google cannot have semver anywhere
  - **We work in a monorepo**, no internal notion of “version”
  - Breaking changes happen all the time!

- **How does OR-Tools do its versioning?**
  - **More or less semver**: let’s hope we don’t break too many people at once
  - If we are planning major changes that break many user, we wait for a major version…
    - Internal code drifts away from OR-Tools!
    - A lot of manual work to update OR-Tools from internal changes

What's next for ORTools.jl?

- Missing features for existing solvers
  - CP-SAT: no access to the CP part, only IP models WIP
  - PDLP: limited by ProtoBuf sizes (2 GB only!) Designed
  - Remote solves
- More testing!
  - Benchmarking too
- Better/faster implementation
  - Maybe use Elemental for local solves
  - Maybe use the C++ interfaces via libcxxwrap

Clipart generated by Gemini

What's next for ORTools.jl?

- CP-SAT status: nearly there, as of November 18th, 2025!
  - Variables: OK
  - MOI constraints: OK (AllDifferent working!)
  - Attributes: OK
  - Solver status/solution: in flight
  - You can soon solve IPs and basic CPs!

- For the next iterations:
  - CP constraints: design stage
  - Interval variables: design stage
  - Quadratic expressions, callbacks

Clipart generated by Gemini

What do you want next for ORTools.jl?

- OR-Tools has **more solvers and features** than currently available:
  - **Routing** solver: for VRP-like problems, based on local search
  - **Set-cover** solver: extremely fast, based on local search
  - **Bin-packing** solvers: LP/MIP-based
  - **Graph** algorithms: including DAGs
- Most of it is available in other languages, but not (yet?) in Julia

Clipart generated by Gemini

Thank You