

ORTools.jl: How to Access Google's Optimisation Solvers in Julia



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

<https://developers.google.com/optimization>

What is operations research?



01 Scheduling

02 Planning

03 Routing

04 Assignment

05 Packing

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

- 01 What the heck is this?

- 02 Google's MathOpt & JuMP in Julia

- 03 Development hurdles: C++ solver, Julia users

- 04 What's next?

What the heck is this?

Mathematical optimisation?

One part of operations research

Distinction between:

- **the data/problem:** scheduling, planning, etc.
- **the algorithms:** simplex, local search, etc.

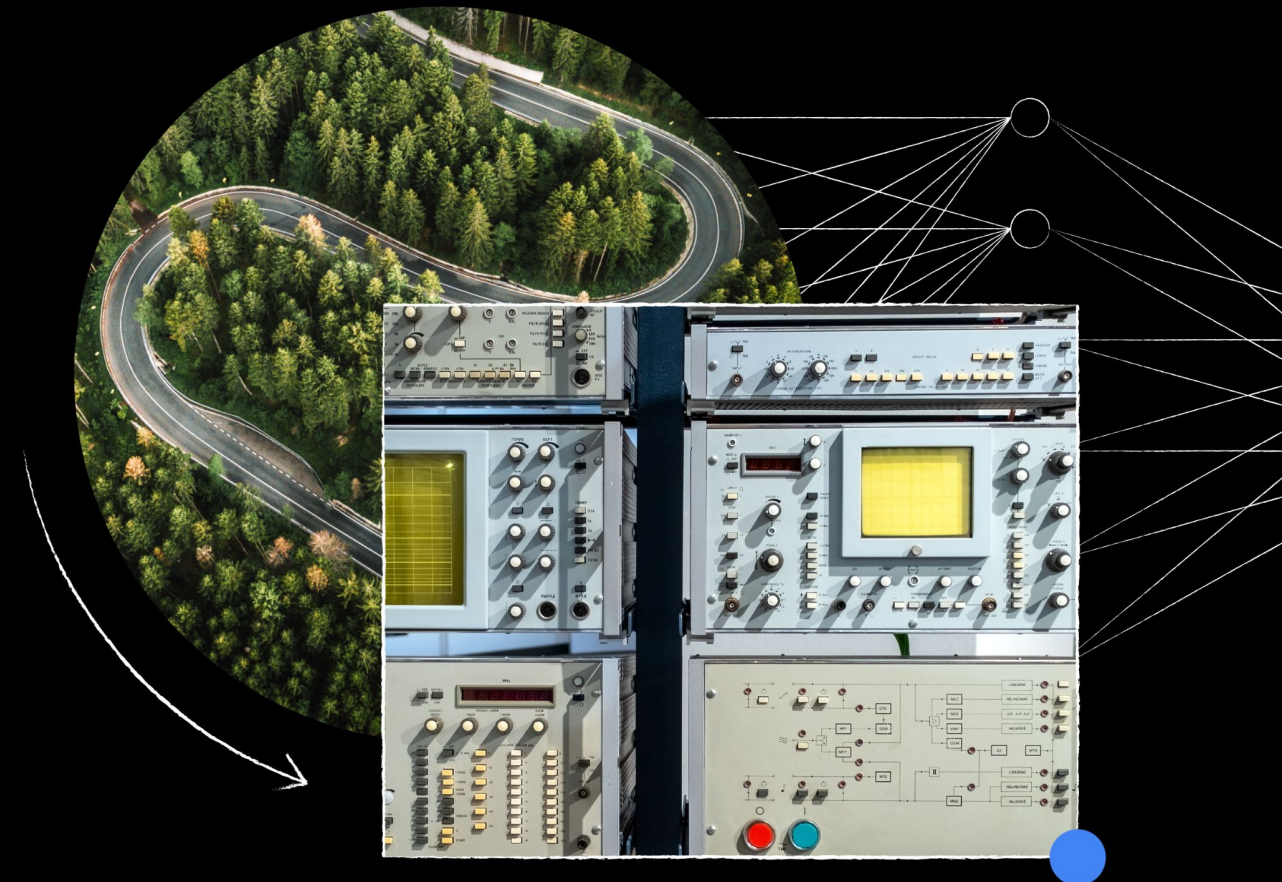
Define a problem as:

- Decisions to make — do A before B?
- Constraints to respect — do A before 10 AM
- Goal to achieve — minimise total time

Declarative programming!

Depending on how you define your problem:

- Linear programming
- Mixed-integer linear programming
- ...



Core components of an optimisation problem

Variables

What you can tune/decide

- Do task A first (yes/no)
- Number of machines of type 1 to buy (integer)
- Fraction of the queries to the first server (number)

Constraints

What you have to abide by

- Do task A before task B
- Deploy 5 TB of RAM
- (No constraint)

Objective function

What you want to minimise or maximise

- Minimise time to perform task B
- Minimise the total cost
- Minimise the difference in loads between the servers

Google's MathOpt & JuMP in Julia



Problem model

Scheduling,
routing, etc.

JuMP

Julia-based
“modelling layer”:
describe your
problem

Solver

Typically, C++ code
that solves the
model

Solution

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),
)
```

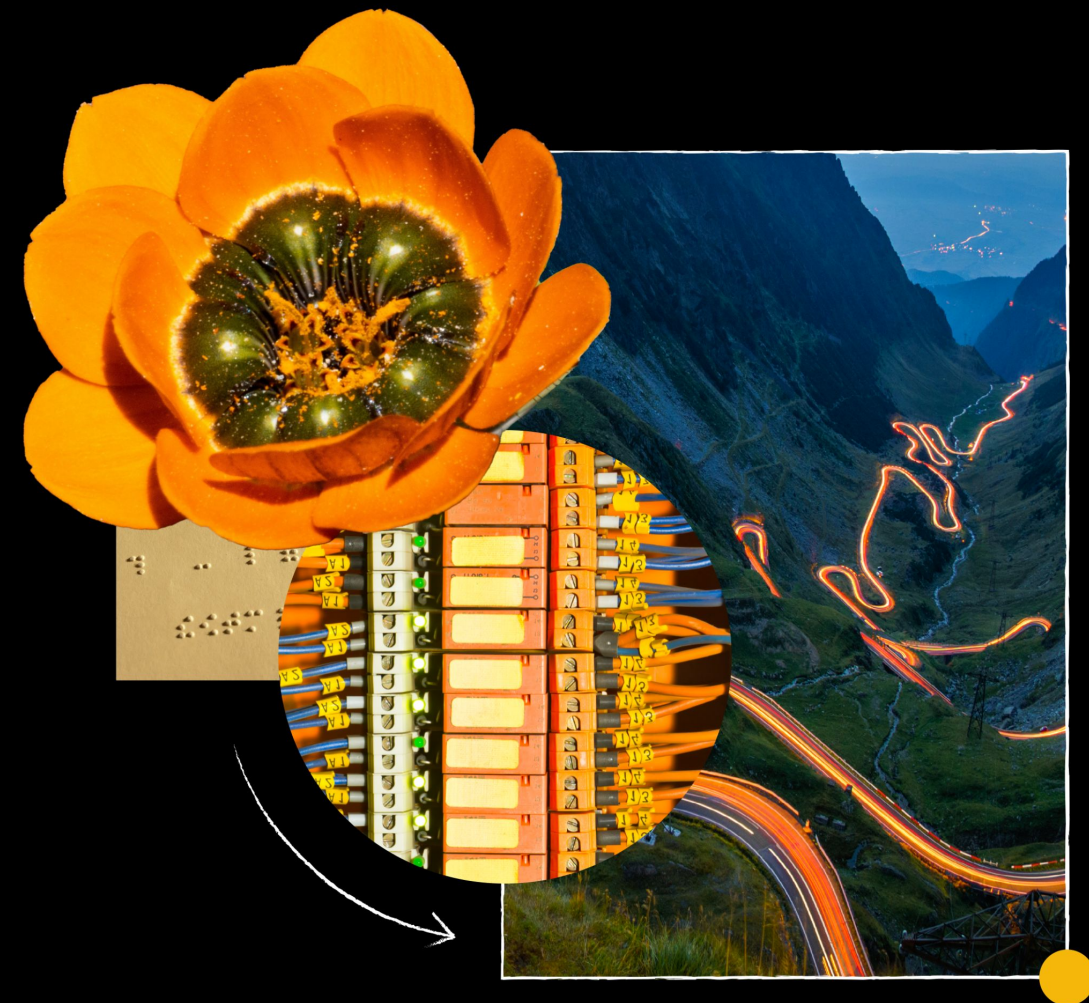
OR-Tools & MathOpt

Different from MathOptInterface.jl!

- MathOpt is part of OR-Tools
- C++ modelling layer
 - Similar to MathOptInterface.jl
- First and foremost: Protocol Buffers interface
 - A kind of binary JSON/XML
- Also a C++ interface
 - Higher performance



Clipart generated by Gemini



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.AddContinuousVariable());
}
for (int j = 0; j < m; ++j) {
    LinearExpression expr(b[0]);
    expr.AddInnerProduct("A[j, :]", x);
}
```

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

Development hurdles

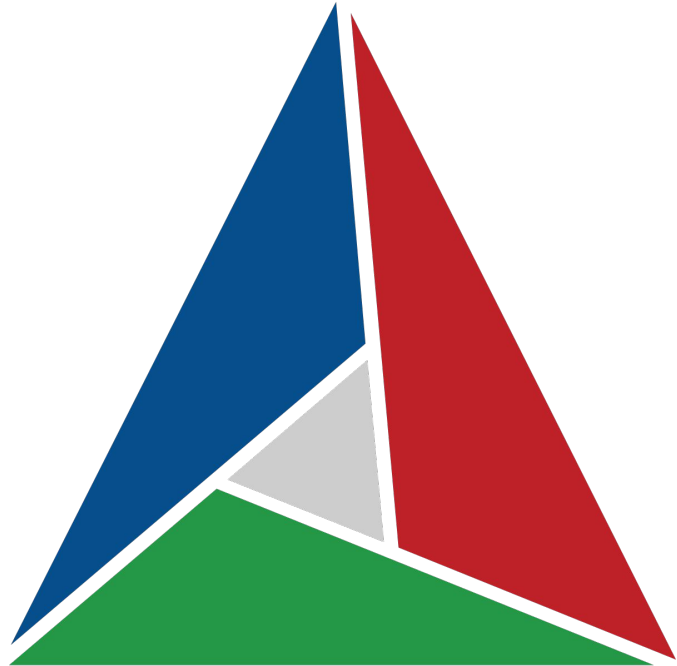
Hence: ORTools.jl!

- 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
 - OR-Tools is easy to build via CMake: Yggdrasil didn't think so!
- Julia
 - 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

- 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
- 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

C++ and Protocol Buffers

- Google uses Protocol Buffers for mostly everything, including 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 ProtoBuf)
 - **Elemental in C:** in development when we started ORTools.jl; now used for the Python interface of MathOpt
- **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



What's next

What's next for OR-Tools.jl!?

- Missing features for existing solvers
 - CP-SAT: no access to the CP part, only MIP models
 - PDLP: limited by ProtoBuf sizes (2 GB only!)
 - Remote solves
- Better/faster implementation
 - Maybe use Elemental for local solves
 - Maybe use the C++ interfaces via libcxxwrap
- More features? **What would you like to see?**
 - Routing solver unavailable
 - Bin-packing solvers unavailable²

WIP

Designed



Clipart generated by Gemini

Thank You