

# OR-Tools' Vehicle Routing Solver:

A Generic Constraint-Programming Solver with Heuristic Search for Routing Problems (VRPs)

Thibaut Cuvelier  
June 9th, 2026

Google Research



# Agenda

- 01 Operations research @ Google
- 02 VRP modelling
- 03 VRP solving
- 04 Academic vs. industrial VRPs

01

# 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, 10+ gold medals at the [MiniZinc competition](#)
- PDLP: LP solver (first order)
- MathOpt: modelling layer

**One open-source product:** [OR-Tools](#)

- Written in C++, accessible in many languages: Python, Java, C#, Julia

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

<https://github.com/google/or-tools/>



Google Research

02

# Vehicle-routing modelling

Vehicle routing (VRP): given a fleet of vehicles, find routes to serve clients

- Generalises the travelling salesperson problem
- Applies to deliveries, pickup-and-delivery, field-service management, carsharing, etc.
- Many complex constraints: time windows (hard/soft), vehicle capacities, visit alternatives...

## Modelling concepts

- OR-Tools offers a few orthogonal concepts for modelling:
  - Weighted graph (of which one or more depots)
  - Vehicles
  - Dimensions: anything that accumulates or dissipates on a route:  
distance, time, weight, etc.

When going from  $i$  to  $j$ :

$$\text{cumul}(j) = \text{cumul}(i) + \text{transit}(i, j) + \text{slack}(i)$$

- If that's not enough: the model is built atop a CP model

# How to model a TSP?

Travelling salesperson problem:

- Weighted graph
- One vehicle
- No capacities
- No time

Model:

- Index manager: maps indices between nodes and internal variables
- Actual model
- Callback for distances

```
RoutingIndexManager manager(num_nodes,  
    /*num_vehicles=*/1, /*depot=*/1);  
RoutingModel routing(manager);
```

```
const int dist_callback_index =  
    routing.RegisterTransitCallback(  
        [&dist, &manager](int from, int to) {  
            return dist(  
                manager.IndexToNode(from).value(),  
                manager.IndexToNode(to).value());  
        });  
routing.SetArcCostEvaluatorOfAllVehicles(  
    dist_callback_index);  
auto solution = routing.Solve();
```

# How to model a CVRP?

Capacitated vehicle-routing problem:

- Weighted graph
- Vehicles with capacities

Model:

- **Callback for demand values**
- **New dimension: demand, for capacities**

No slack: hard constraint

```
RoutingIndexManager manager(num_nodes,  
    num_vehicles, /*depot=*/1);  
RoutingModel routing(manager);
```

```
const int node_demand_callback_index =  
    model->RegisterUnaryTransitCallback(  
        [&demand, &manager](int i) {  
            return demand[  
                manager->IndexToNode(i).value()];  
        });  
model->AddDimension(  
    node_demand_callback_index,  
    /*slack_max=*/0,  
    /*capacity=*/max_capacity,  
    /*fix_start_cumul_to_zero=*/false,  
    /*name=*/'Demand');
```

```
auto solution = routing.Solve();
```

# How to model a VRPTW?

Vehicle-routing problem with time windows:

- Weighted graph
- Vehicles with no capacity
- Visits with time windows:

Model:

- **Callback for arc lengths**
- **New dimension: time**  
Slack: waiting time
- **Variable bounds: time windows**

```
const int max_end_time = ...;
const int arc_length =
    model->RegisterTransitCallback(...);
model->AddDimension(
    arc_length,
    /*slack_max=*/max_end_time,
    /*capacity=*/max_end_time,
    /*fix_start_cumul_to_zero=*/false,
    "Time");
```

```
RoutingDimension* time_dimension =
    model->GetMutableDimension("Time");
```

```
for (NodeIndex node(0);
     node < manager->num_nodes(); ++node) {
    const int index =
        manager->NodeToIndex(node);
    IntVar* const cumul =
        time_dimension->CumulVar(index);
    cumul->SetMin(time_start(node));
    cumul->SetMax(time_end(node));
}
```

Google Research

## 03

# Vehicle-routing solving

Solving is done in three steps:

- First-solution heuristics
- Local search and metaheuristics
- Feasibility check

# First-solution heuristics

- Goal: generate a first set of routes
  - Hopefully feasible
  - Possibly, not many visits performed
- Many techniques, including:
  - Incremental route creation: cheapest arc, most constrained arc
  - Savings, sweep, Christofides

# Local search and metaheuristics

- Local-search operators to improve a solution:
  - Intra-route neighbourhoods (like a TSP)
  - Inter-route neighbourhoods (between routes)
  - Large neighbourhoods (LNS)
- Metaheuristic to guide the search:
  - Guided local search (most effective, usually)
  - Tabu search, simulated annealing

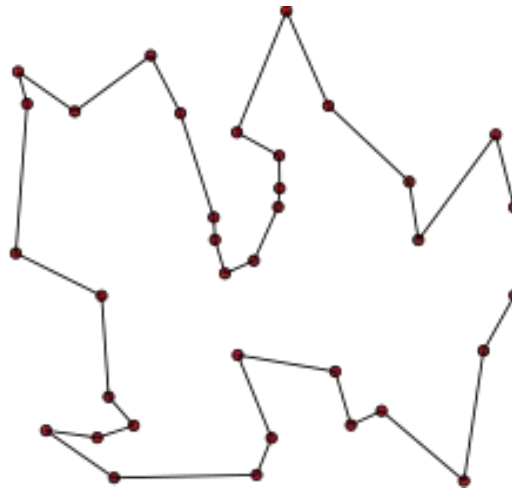
# Constraint satisfaction

- Local-search operators ignore constraints!
  - Extremely quick filters reject infeasible neighbours
  - Very few false positives, but extremely quick
- After the search: CP engine to check for constraint satisfaction
- If you want: tree exploration to prove optimality

# 04

## Academics vs industry users

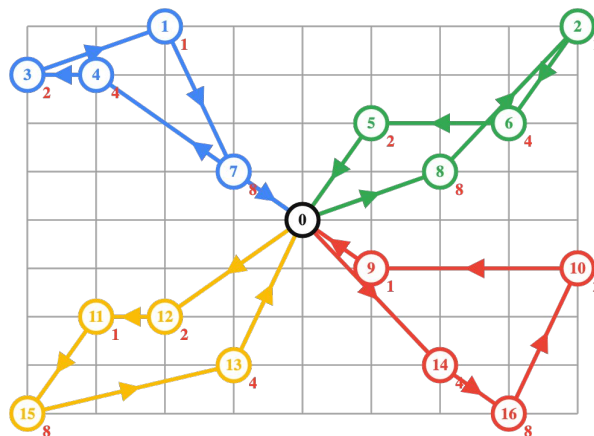
- Academics focus on “pure” problems:
  - Traveling salesperson problem (TSP)



# 04

## Academics vs industry users

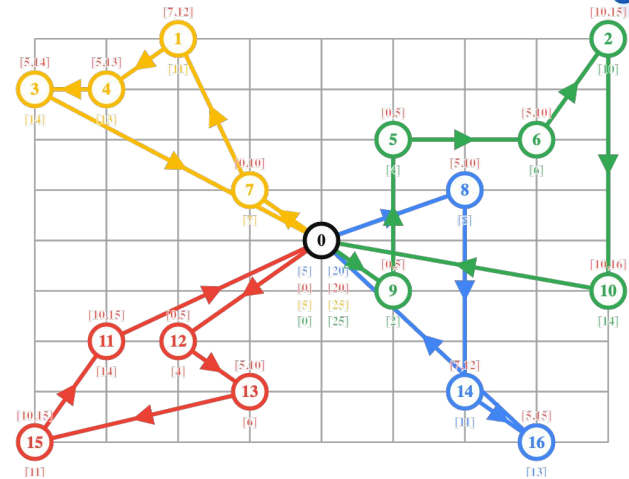
- Academics focus on “pure” problems:
  - Traveling salesperson problem (TSP)
  - Capacitated vehicle routing (CVRP)



# 04

## Academics vs industry users

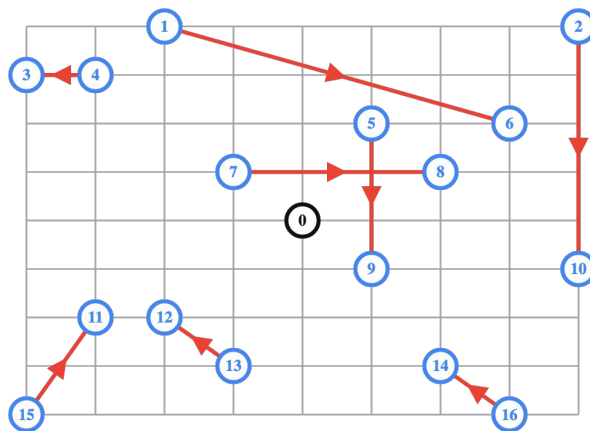
- Academics focus on “pure” problems:
  - Traveling salesperson problem (TSP)
  - Capacitated vehicle routing (CVRP)
  - ... with time windows (CVRPTW)



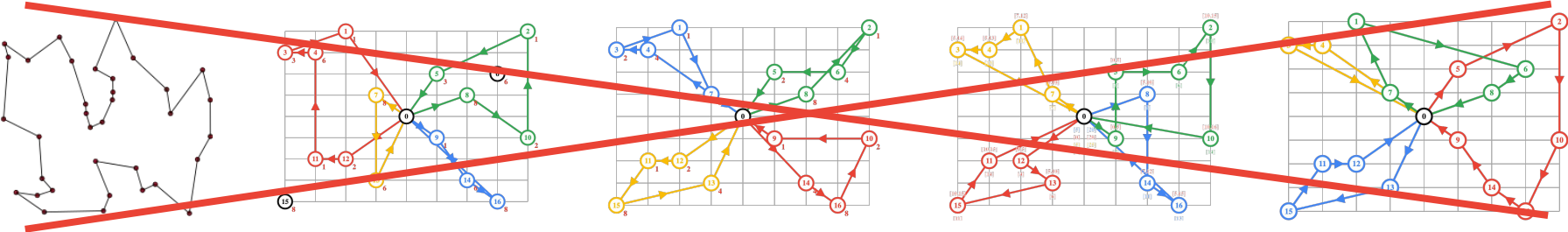
# 04

## Academics vs industry users

- Academics focus on “pure” problems:
  - Traveling salesperson problem (TSP)
  - Capacitated vehicle routing (CVRP)
  - ... with time windows (CVRPTW)
  - Pickup and delivery (PDPTW)



This is not what we see in the real world...



# Structural differences: time

- Time isn't just time windows for deliveries
  - Business opening hours, individuals are more flexible
- Legal constraints:
  - A driver cannot work more than X hours per day
  - A driver must take breaks during the day
- Time is typically the most constraining factor!

# Structural differences: objectives

What's a good solution?

- Academic: minimise the number of vehicles, the total length
- Industrial users: “give me good routes”
  - Execs: minimise the number of vehicles, minimise the number of worked hours
  - Dispatchers: robustness, consistency — avoid stressful replanning
  - Drivers: fairness (hours, number of deliveries)

# Structural differences

- Costs: typically arbitrary
  - Useful for shaping the routes for exact preferences
  - Beta feature: [objectives](#)
- Electric vehicles: avoid going up a hill with full load
  - Beta feature: [load costs](#)
- Explainability: “Why couldn’t you do X instead?”
  - Typically, “why was this shipment skipped?”
- Overconstrained instances: actual constraint or preference?

05

## Final words

- OR-Tools' Vehicle Routing Solver is a complete vehicle-routing library
  - Focus on rich problems: many constraints
  - High-level modelling interface, access to the underlying CP model if you need
  - Apache 2.0 license: <https://github.com/google/or-tools>
- Base of Google Maps Platform Route Optimization
  - Major differences: integration with Google Maps for distances, automatic decomposition for large instances, advanced features



<https://cloud.google.com/blog/products/ai-machine-learning/google-cloud-optimization-ai-cloud-fleet-routing-api>

Google Research

# Thank You

**Thibaut Cuvelier**  
Software engineer