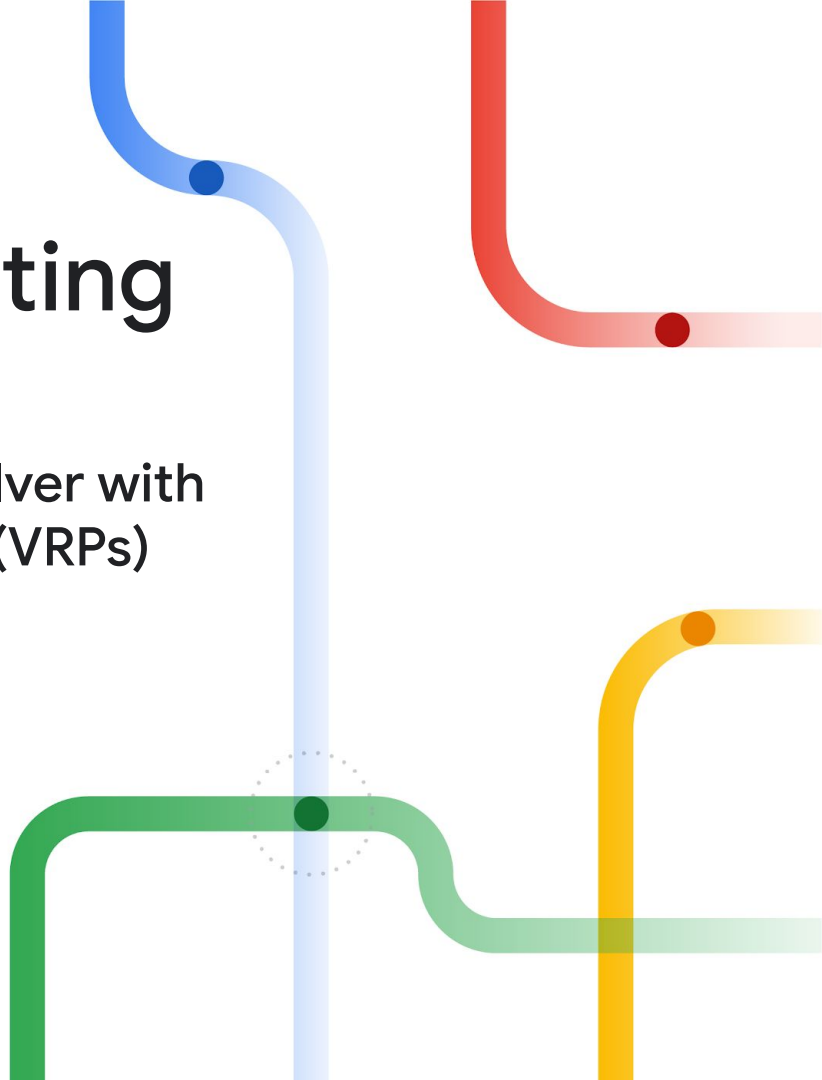
# OR-Tools' Vehicle Routing Solver:

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

Thibaut Cuvelier
February 23, 2023

Google Research



# Agenda

01 Operations research @ Google

02 VRP modelling

03 VRP solving

Google Research

01

# Operations research @ Google

- Team based in Paris (FR) and Cambridge (US)
- Started in 2008
- One mission: **optimise Google**
  - Initially: route vehicles to create the Street View data base efficiently
  - Now: also datacenter design, robotics
  - Vehicle routing is also available on Google Cloud

https://cloud.google.com/optimization

Google Research

# OR-Tools

- Core solver technology: OR-Tools
  - In-house CP and linear solvers — CP-SAT has won many MiniZinc gold medals!
  - Generic modelling layer, access to MIP solvers
  - Routing modelling layer
- Open-source software since 2010 (Apache 2.0 license)
- Written in C++, accessible from Python, Java, C#

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`:

    ```
    cumul(j) = cumul(i) + transit(i, j) + slack(i)
    ```
- If that's not enough: the model is built atop a CP model

Google Research

# 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();
```

Google Research

https://developers.google.com/optimization/routing/tsp

# 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();
```

Google Research

https://developers.google.com/optimization/routing/cvrp

# 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

https://developers.google.com/optimization/routing/vrptw

03

# Vehicle-routing solving

Solving is done in three steps:

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

Google Research

# 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

# 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 Cloud's Cloud Fleet Routing
  - Major differences: integration with Google Maps for distances, automatic decomposition for large instances

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

Google Research