



DEVELOPING OPTIMAL SOLUTIONS FOR ORGANIZATIONAL AND BUSINESS NEEDS USING OR (OPERATIONS RESEARCH) AND AI (ARTIFICIAL INTELLIGENCE)

DAY 3: Google's OR-Tools logging of the solve process. More modeling. Communicating results

Prof. Christos G. Gkogkos

University of Ioannina, Dept. of Informatics and Telecommunications

https://github.com/chgogos/INTRO_OR_AI


CP-SAT: LOGGING

- Logging is useful **for monitoring the search progress** and for understanding **which CP-SAT techniques have the greater impact** in solving the problem (e.g., how many variables were directly removed)
- Logging is activated with:
`solver.parameters.log_search_progress = True`
- Use the [CP-SAT Log analyzer tool](https://cpsat-log-analyzer.streamlit.app/) developed by Dominik Krupke

CP-SAT Log Analyzer

Dive into the world of constraint programming with ease using our CP-SAT Log Analyzer. This tool transforms the dense and detailed logs of CP-SAT into clear, readable formats, complemented by intuitive visualizations of key metrics. Whether you're tuning your model or exploring data, our analyzer simplifies and enlightens your journey with CP-SAT. Let us make complex logs simple and actionable!

 D-KRUPKE **CP-SAT LOG ANALYZER** Feel free to open issues or contribute.

 D-KRUPKE **CP-SAT PRIMER** This project is a sibling of the CP-SAT Primer.

Log File

To begin analyzing with CP-SAT Log Analyzer, please upload your log file. If you haven't already, you can generate a log file by enabling the log output. Simply set the `log_search_progress` parameter to `True` in your CP-SAT solver configuration. Once this is done, you'll have a detailed log ready for upload and analysis.

The log usually starts as follows:

```
Starting CP-SAT solver v9.7.2996
Parameters: log_search_progress: true
Setting number of workers to 24

...
```

Only complete and properly formatted logs are supported for now.

Upload a log file



Drag and drop file here
Limit 200MB per file • TXT

Browse files

<https://cpsat-log-analyzer.streamlit.app/>

CP-SAT: LOGGING – LOG ANALYZER

- Walk through example 3 (multi-knapsack problem) of CP-SAT Log Analyzer

Overview



This log originates from a Multi-Knapsack Problem. The presolve-phase replaces a lot of the constraints by `AtMostOne`, but is only able to eliminate a small amount of the variables during presolve. Only during the search, the model gets strongly reduced.

An important observation here is that CP-SAT very quickly reaches near-optimality but struggles to close a tiny gap at the end. This is actually quite common. By simply allowing CP-SAT some slack by setting `relative_gap_limit`, e.g., to 0.01 to allow it to stop with a 1% gap, can drastically speed up CP-SAT. For many problems in practice, the underlying data has some error anyway, such that 1% (or sometimes even 20%) is a negligible error.

CP-SAT Version ⓘ

v9.8.3296

↑ outdated

Number of workers ⓘ

16

CP-SAT was setup with the following parameters:

```
{
  "max_time_in_seconds" : 1800
  "log_search_progress" : true
  "relative_gap_limit" : 0
}
```

You can find more information about the parameters [here](#).

Status ⓘ

OPTIMAL

Time ⓘ

12.744s

Presolve ⓘ

0.170s

Variables ⓘ

9000

Constraints ⓘ

330

Type ⓘ

Optimization

PROTO



- Protocol Buffers (protobufs or proto) is a **language neutral** and **platform neutral** mechanism, developed by Google, for serializing structured data
- In OR-Tools models and solutions are represented internally using protobuf
- A low-level API is provided for manipulating the solver's internal state (e.g., parameters)
- A model can be exported to a proto file (binary or text), then it can be loaded, instead of having to rebuild it
- Plausible scenario: Build a model in a machine, export the proto file, send to multiple other machines, solve model in each machine
 - No need to share the code that builds the model
 - No need to rebuild the model at each machine

PROTO

Write proto (model)

```
# Write model proto to disk.
fn = os.path.join(os.path.dirname(__file__), "nqueens_cpsat_proto_model.pb")
with open(fn, "wb") as f:
    f.write(model.Proto().SerializeToString())
```

Load proto (model)

```
1  from ortools.sat.python import cp_model
2  from pathlib import Path
3  import os
4
5
6  def solve():
7      # Create the solver
8      model = cp_model.CpModel()
9
10     # Load model proto file.
11     filename = os.path.join(os.path.dirname(__file__), "nqueens_cpsat_proto_model.pb")
12     model.Proto().ParseFromString(Path(filename).read_bytes())
13     print("Model proto file loaded!")
14
15     # Solve the model.
16     solver = cp_model.CpSolver()
17     solver.solve(model)
18
19     # Display proto file of the solution.
20     print("\nProto solution")
21     solution_proto = solver.ResponseProto()
22     print(solution_proto)
23
24
25 if __name__ == "__main__":
26     solve()
```

CP-SAT: MODELING USING INTERVALS – INTERVAL VARIABLES

- An interval variable can be used to model a span of some length that has a start (and an end)
- Types of interval variables:
 - Fixed length interval
 - Flexible length interval
 - Optional fixed length interval
 - Optional flexible length interval
- Usually used together with the `no_overlap` constraint

```
1  from ortools.sat.python import cp_model
2
3  model = cp_model.CpModel()
4
5  start_var = model.new_int_var(0, 100, "start")
6  length_var = model.new_int_var(10, 20, "length")
7  end_var = model.new_int_var(0, 100, "end")
8  is_present_var = model.new_bool_var("is_present")
9
10 # creating an interval whose length can be influenced by a variable (more expensive)
11 flexible_interval = model.new_interval_var(
12     start=start_var, size=length_var, end=end_var, name="flexible_interval"
13 )
14
15 # creating an interval of fixed length
16 fixed_interval = model.new_fixed_size_interval_var(
17     start=start_var,
18     size=10, # needs to be a constant
19     name="fixed_interval",
20 )
21
22 # creating an interval that can be present or not and whose length can be influenced by a variable (most expensive)
23 optional_interval = model.new_optional_interval_var(
24     start=start_var,
25     size=length_var,
26     end=end_var,
27     is_present=is_present_var,
28     name="optional_interval",
29 )
30
31 # creating an interval that can be present or not
32 optional_fixed_interval = model.new_optional_fixed_size_interval_var(
33     start=start_var,
34     size=10, # needs to be a constant
35     is_present=is_present_var,
36     name="optional_fixed_interval",
37 )
```

CP-SAT: MODELING USING INTERVALS - CONSTRAINTS

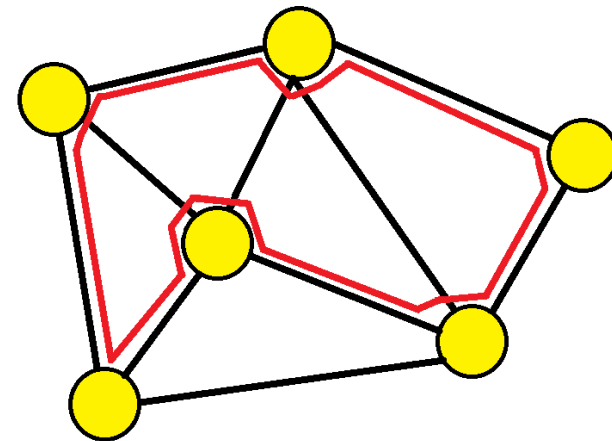
- Three constraints exist that can combined with interval variables:
 - `add_no_overlap`: prevent overlap of intervals in one dimension (usually time)
 - `add_no_overlap_2d`: prevent overlap of intervals in two dimensions
 - `add_cumulative`: model situations where the sum of overlapping intervals must not exceed a given capacity

```
1 from ortools.sat.python import cp_model
2
3 # Model
4 m = cp_model.CpModel()
5 durs = [3, 2, 4, 1]
6 horizon = sum(durs)
7
8 starts = [m.new_int_var(0, horizon, f's{i}') for i in range(4)]
9 ends = [m.new_int_var(0, horizon, f'e{i}') for i in range(4)]
10 intervals = [m.new_interval_var(starts[i], durs[i], ends[i], f'int{i}') for i in range(4)]
11
12 m.add_no_overlap(intervals)
13
14 # Precedence constraint: Task 0 must finish before Task 2 starts
15 m.add(ends[0] <= starts[2])
16
17 makespan = m.new_int_var(0, horizon, 'makespan')
18 m.add_max_equality(makespan, ends)
19 m.minimize(makespan)
20
21 # Solve
22 s = cp_model.CpSolver()
23 s.solve(m)
24
25 for i in range(4):
26     print(f'Task {i}: {s.value(starts[i])} -> {s.value(ends[i])}')
27 print(f'Makespan: {s.value(makespan)}')
```

```
Task 0: 0 -> 3
Task 1: 7 -> 9
Task 2: 3 -> 7
Task 3: 9 -> 10
Makespan: 10
```

CP-SAT: ADD_CIRCUIT CONSTRAINT

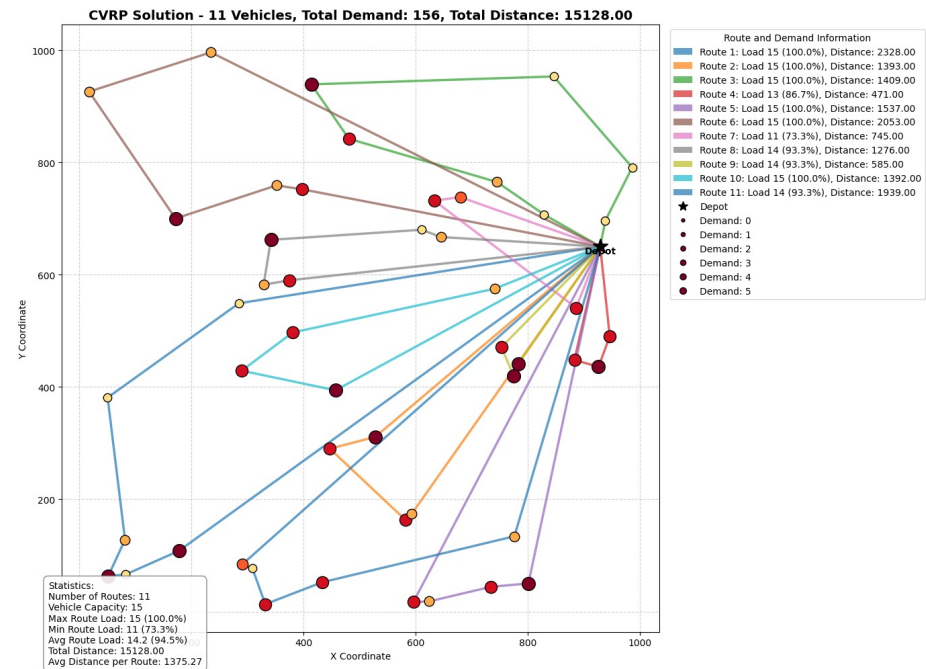
- A circuit is a cycle over all vertices of a graph that starts and ends at the same vertex (there are no repeated vertices except the start = end)
- The add_circuit constraint enforces a circuit to occur in a directed graph
 - It takes a list of triplets (u,v,var) where u and v are the source and target vertices respectively and var is a boolean variable that indicates if the edge $u \rightarrow v$ is included in the circuit
 - It enforces that the edges marked True form a circuit, visiting all vertices



A Hamiltonian cycle (circuit)
over an undirected graph

CP-SAT: ADD_MULTIPLE_CIRCUIT CONSTRAINT

- add_multiple_circuit constraint is used for problems involving multiple trips starting from a depot solving
- Works similarly to add_circuit but allows the depot to be visited many times



COMMUNICATING RESULTS



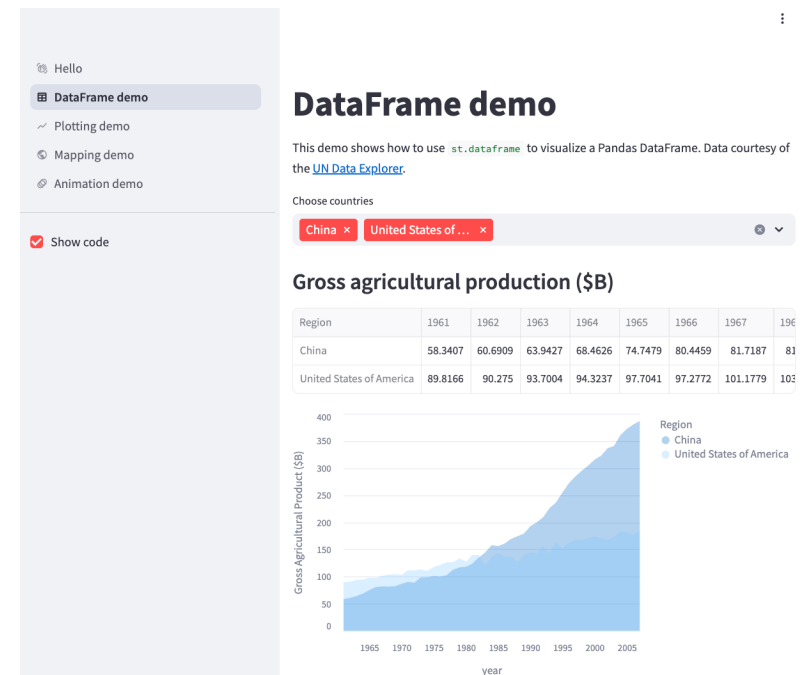
- Streamlit is a great way to demo a working code

```
$ uv add streamlit
$ streamlit hello
Welcome to Streamlit. Check out our demo in your browser.

Local URL: http://localhost:8502
Network URL: http://192.168.1.83:8502

Ready to create your own Python apps super quickly?
Head over to https://docs.streamlit.io

May you create awesome apps!
```



VISUALIZATION OF A SIMPLE OPTIMIZATION PROBLEM (PRODUCT MIX)

```
1 import streamlit as st
2 import matplotlib.pyplot as plt
3 from ortools.sat.python import cp_model
4
5 > def solve():-
6
7     st.title("A product mix problem")
8     st.write("Our company produces desks and tables")
9     st.write("Production of one desk requires 3 units of wood, 1 hour of labor and 50 minutes of machine time")
10    st.write("We have 3600 units of wood, 1600 labour hours and 48000 minutes of machine time")
11    desk_price = st.slider('Desks unit price', min_value=500, max_value=1500, step=10, value=700)
12    table_price = st.slider('Tables unit price', min_value=500, max_value=1500, step=10, value=900)
13    st.write(f"desk price={desk_price} \u20AC, table price={table_price} \u20AC")
14
15    objective, desk_items, table_items = solve()
16
17    st.markdown("----")
18    st.markdown("## Results")
19    st.write(f"Total profit = {objective:.0f} \u20AC")
20    st.write(f"Number of desks = **{desk_items}**, Number of tables = **{table_items}**")
21
22    st.markdown("----")
23    fig, ax = plt.subplots()
24    ax.bar(["Desks", "Tables"], [desk_items, table_items], color=["blue", "orange"])
25    ax.set_ylabel("Number of items")
26    ax.set_title("Production Mix")
27    for i, v in enumerate([desk_items, table_items]):
28        ax.text(i, v + 0.5, str(v), ha='center', va='bottom')
29    st.pyplot(fig)
30
31 def solve():
32     # create model
33     model = cp_model.CpModel()
34
35     # decision variables
36     x1 = model.new_int_var(0,10_000, "x1")
37     x2 = model.new_int_var(0,10_000, "x2")
38
39     model.add(3*x1 + 5*x2 <= 3600) # wood
40     model.add(x1 + 2*x2 <= 1600) # labor (hours)
41     model.add(50*x1 + 20*x2 <= 48000) # machine time (minutes)
42
43     model.maximize(desk_price*x1+table_price*x2)
44
45     solver = cp_model.CpSolver()
46     status = solver.solve(model)
47
48     if status == cp_model.OPTIMAL:
49         return solver.objective_value, solver.value(x1), solver.value(x2)
```

A product mix problem

Our company produces desks and tables

Production of one desk requires 3 units of wood, 1 hour of labor and 50 minutes of machine time

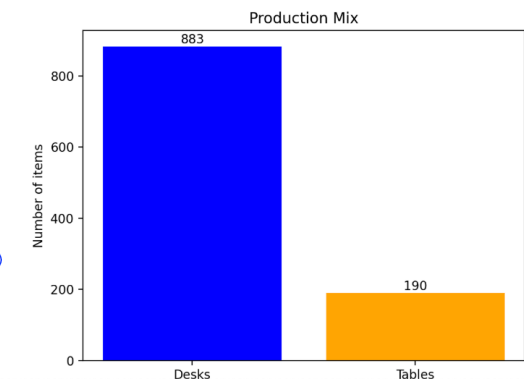
We have 3600 units of wood, 1600 labour hours and 48000 minutes of machine time



Results

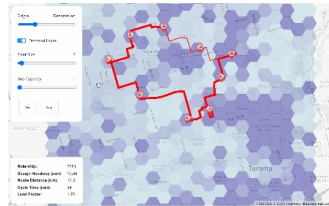
Total profit = 789100 €

Number of desks = 883, Number of tables = 190



EXAMPLES OF COMMUNICATING RESULTS

- <https://interactive-or.net/>



NETWORK DESIGN

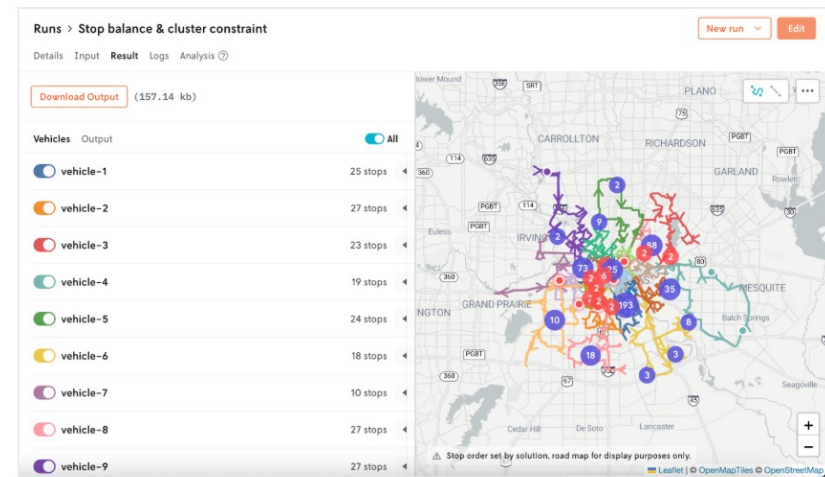
1. **4-Step Model:** Implements trip generation, distribution, and assignment.
2. **GTFS Integration:** Supports editing of routes, blocks, and trips.
3. **Adjustable Parameters:** Allows changes to stop locations and headways.
4. **Congestion Visualization:** Line thickness reflects network load.

REAL-TIME MONITORING

1. **Real-Time Animation:** Predicts bus positions between GTFS-RT updates for smooth display.
2. **Bunching Detection:** Identifies bunching using static and real-time GTFS data.
3. **Heatmap Display:** Shows bunching intensity over 5–30 minute intervals.
4. **Upcoming Feature:** Incorporation of crowding data for service adjustments.



- <https://www.nextmv.io/>



Nextmv UI showing routes and unplanned stop on a map

HANDS-ON ACTIVITIES



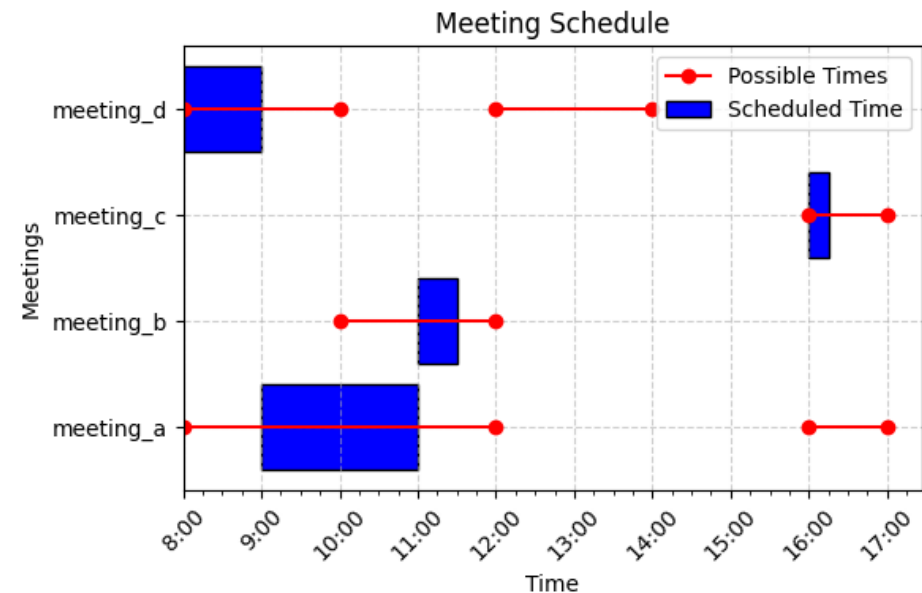
Meeting Schedule



Traveling Salesman Problem

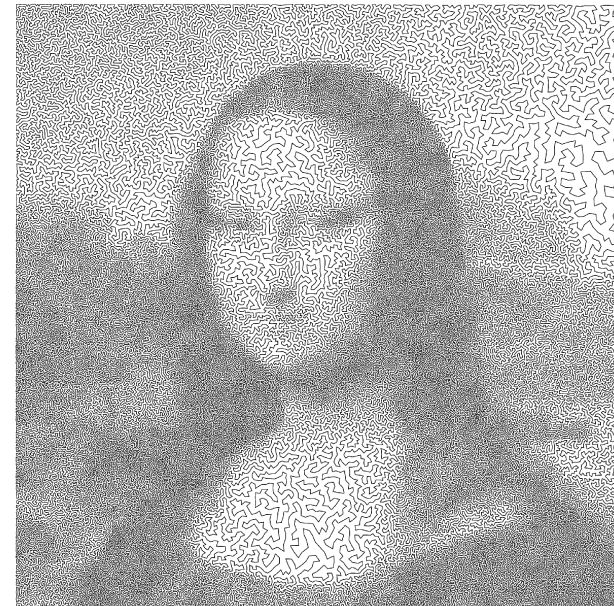
HANDS-ON - MEETING SCHEDULE

- In the “Advanced modeling” section of the online book “CP-SAT Primer”, locate the section “Scheduling for a Conference Room with intervals”
- Find, at the github repository of the book, the corresponding code that generates the chart displayed at the right side of this slide
- Add an extra constraint that forces meeting_d to be scheduled after meeting_b
- Add an extra meeting named “meeting_e” with duration 40 minutes, which should be scheduled between 10:00 and 11:00



HANDS-ON - TRAVELING SALESMAN PROBLEM

- The Traveling Salesman has a simple description:
"Given a list of cities and the distances between each pair, what is the shortest possible route that visits each city exactly once and returns to the origin city?"
- The problem has a long history, see <https://www.math.uwaterloo.ca/tsp/index.html>
- Add the missing `add_circuit` constraint to `day3_ho2_template.py`, so as to be able to find an optimal tour for a 50 cities problem, see https://d-krupke.github.io/cpsat-primer/04B_advanced_modelling.html
- Activate the log and analyze it using the [CP-SAT Log analyzer](#)



mona-lisa 100K.tsp

Tour: 5,757,191 **Bound:** 5,757,084 **Gap:** 107 (0.0019%)

REFERENCES

- Google OR-Tools CP-SAT Python Reference:
https://developers.google.com/optimization/reference/python/sat/python/cp_model
- CP-SAT Primer, <https://d-krupke.github.io/cpsat-primer/>