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

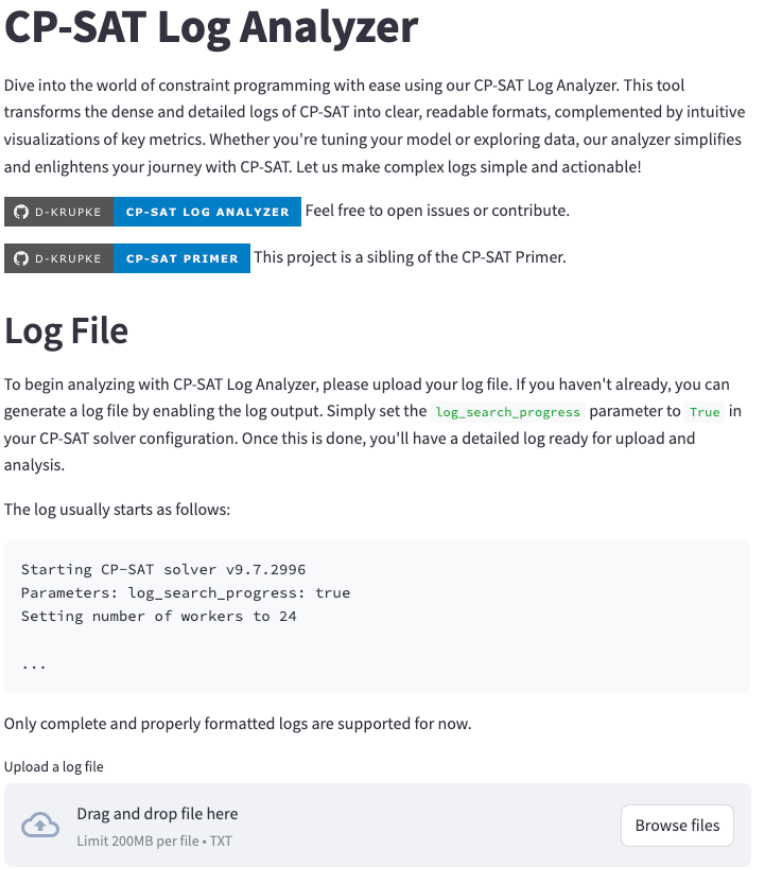
https://github.com/chgogos/INTRO_OR_AI

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 developed by Dominik Krupke



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

CP-SAT: LOGGING – LOG ANALYZER

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

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)

```python
# 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)

```python
from ortools.sat.python import cp_model
from pathlib import Path
import os


def solve():
    # Create the solver
    model = cp_model.CpModel()

    # Load model proto file.
    filename = os.path.join(os.path.dirname(__file__), "nqueens_cpsat_proto_model.pb")
    model.Proto().ParseFromString(Path(filename).read_bytes())
    print("Model proto file loaded!")

    # Solve the model.
    solver = cp_model.CpSolver()
    solver.solve(model)

    # Display proto file of the solution.
    print("\nProto solution")
    solution_proto = solver.ResponseProto()
    print(solution_proto)


if __name__ == "__main__":
    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 a 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

```python
from ortools.sat.python import cp_model

model = cp_model.CpModel()

start_var = model.new_int_var(0, 100, "start")
length_var = model.new_int_var(10, 20, "length")
end_var = model.new_int_var(0, 100, "end")
is_present_var = model.new_bool_var("is_present")

# creating an interval whose length can be influenced by a variable (more expensive)
flexible_interval = model.new_interval_var(
    start=start_var, size=length_var, end=end_var, name="flexible_interval"
)

# creating an interval of fixed length
fixed_interval = model.new_fixed_size_interval_var(
    start=start_var,
    size=10,  # needs to be a constant
    name="fixed_interval",
)

# creating an interval that can be present or not and whose length can be influenced by a variable (most expensive)
optional_interval = model.new_optional_interval_var(
    start=start_var,
    size=length_var,
    end=end_var,
    is_present=is_present_var,
    name="optional_interval",
)

# creating an interval that can be present or not
optional_fixed_interval = model.new_optional_fixed_size_interval_var(
    start=start_var,
    size=10,  # needs to be a constant
    is_present=is_present_var,
    name="optional_fixed_interval",
)
```

CP-SAT: MODELING USING INTERVALS - CONSTRAINTS

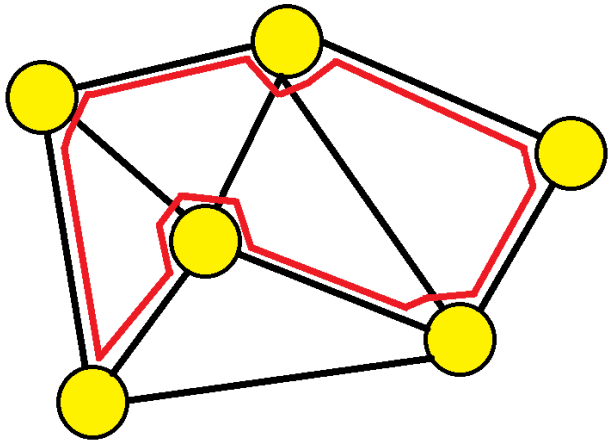
- Three constraints exist that can combined with interval variables:
  - add_no_overlap: prevent overlap of intervals in one dimension (usually time)
  - add_no_overlap_2d: prevent overlap of intervals in two dimensions
  - add_cumulative: model situations where the sum of overlapping intervals must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

```python
from ortools.sat.python import cp_model

# Model
m = cp_model.CpModel()
durs = [3, 2, 4, 1]
horizon = sum(durs)

starts = [m.new_int_var(0, horizon, f's{i}') for i in range(4)]
ends = [m.new_int_var(0, horizon, f'e{i}') for i in range(4)]
intervals = [m.new_interval_var(starts[i], durs[i], ends[i], f'int{i}') for i in range(4)]

m.add_no_overlap(intervals)

# Precedence constraint: Task 0 must finish before Task 2 starts
m.add(ends[0] <= starts[2])

makespan = m.new_int_var(0, horizon, 'makespan')
m.add_max_equality(makespan, ends)
m.minimize(makespan)

# Solve
s = cp_model.CpSolver()
s.solve(m)

for i in range(4):
    print(f'Task {i}: {s.value(starts[i])} -> {s.value(ends[i])}')
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

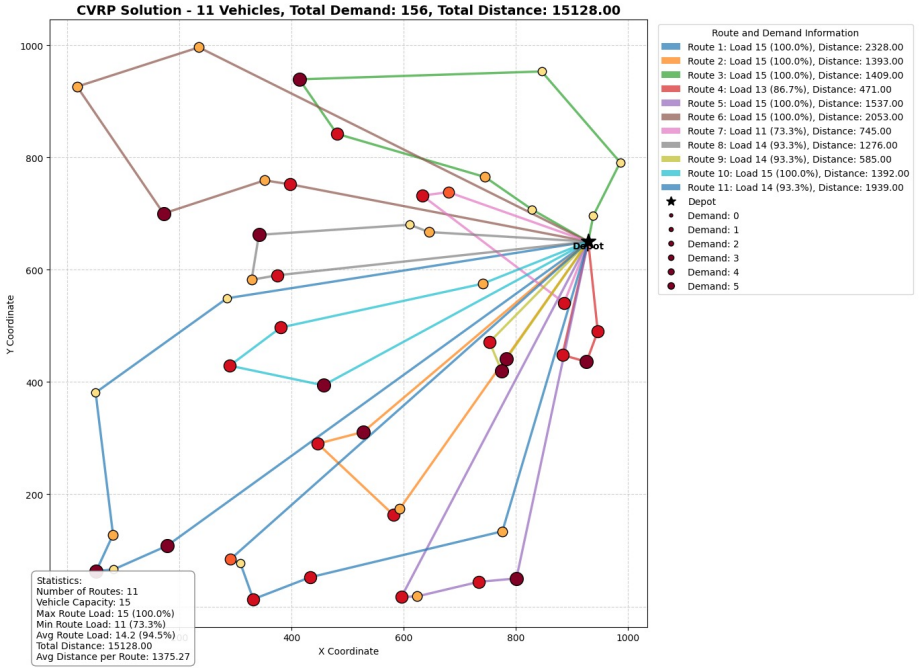
- A circuit is a cycle over all vertices of a graph that starts and ends at the same vertex (there are no repeated vertices except the start = end)
- The add_circuit constraint enforces a circuit to occur in a directed graph
  - It takes a list of triplets (u,v,var) where u and v are the source and target vertices respectively and var is a boolean variable that indicates if the edge u→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 mutiple trips starting from a depot solving
- Works similarly to add_circuit but allows the depot to be visited many times

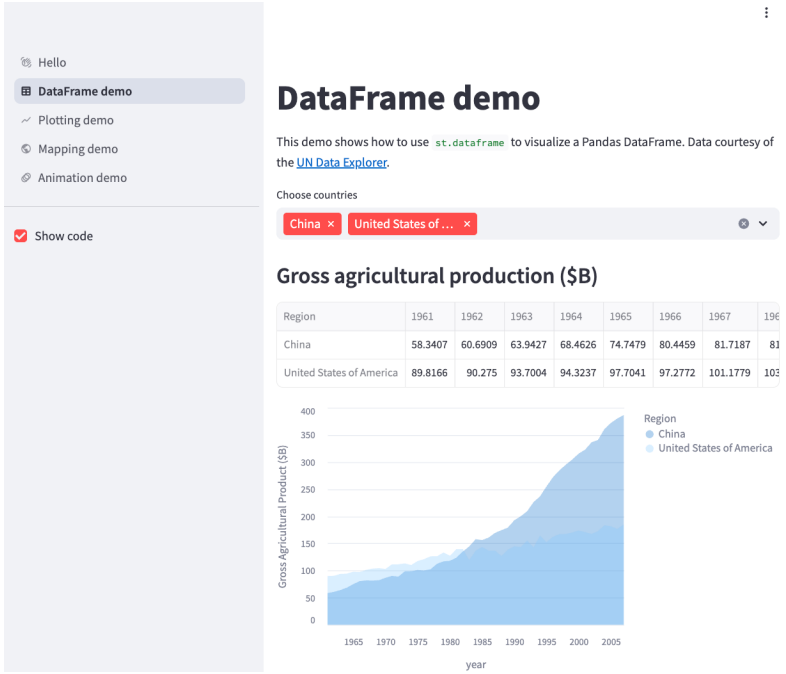


COMMUNICATING RESULTS

Streamlit

- Streamlit is a great way to demo a working code

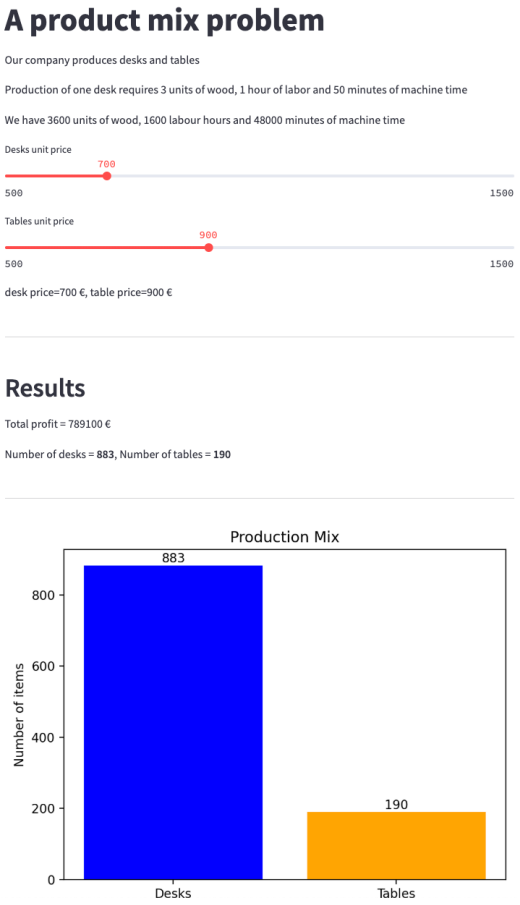
```
$ uv add streamlit
$ streamlit hello
Welcome to Streamlit. Check out our demo in your browser.

  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)

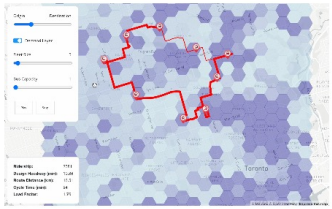
```python
import streamlit as st
import matplotlib.pylab as plt
from ortools.sat.python import cp_model

def solve(): ⋯

st.title("A product mix problem")
st.write("Our company produces desks and tables")
st.write("Production of one desk requires 3 units of wood, 1 hour of labor and 50 minutes of machine time")
st.write("We have 3600 units of wood, 1600 labour hours and 48000 minutes of machine time")
desk_price = st.slider('Desks unit price', min_value=500, max_value=1500, step=10, value=700)
table_price = st.slider('Tables unit price', min_value=500, max_value=1500, step=10, value=900)
st.write(f"desk price={desk_price} \u20AC, table price={table_price} \u20AC")

objective, desk_items, table_items = solve()

st.markdown("---")
st.markdown("## Results")
st.write(f"Total profit = {objective:.0f} \u20AC")
st.write(f"Number of desks = **{desk_items}**, Number of tables = **{table_items}**")

st.markdown("---")
fig, ax = plt.subplots()
ax.bar(["Desks", "Tables"], [desk_items, table_items], color=["blue", "orange"])
ax.set_ylabel("Number of items")
ax.set_title("Production Mix")
for i, v in enumerate([desk_items, table_items]):
    ax.text(i, v + 0.5, str(v), ha='center', va='bottom')
st.pyplot(fig)
```

```python
def solve():
    # create model
    model = cp_model.CpModel()

    # decision variables
    x1 = model.new_int_var(0,10_000, "x1")
    x2 = model.new_int_var(0,10_000, "x2")

    model.add(3*x1 + 5*x2 <= 3600) # wood
    model.add(x1 + 2*x2 <= 1600) # labor (hours)
    model.add(50*x1 + 20*x2 <= 48000) # machine time (minutes)

    model.maximize(desk_price*x1+table_price*x2)

    solver = cp_model.CpSolver()
    status = solver.solve(model)

    if status == cp_model.OPTIMAL:
        return solver.objective_value, solver.value(x1), solver.value(x2)
```



EXAMPLES OF COMMUNICATING RESULTS

- https://interactive-or.net/

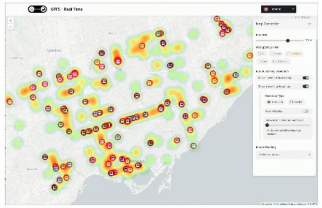


NETWORK DESIGN

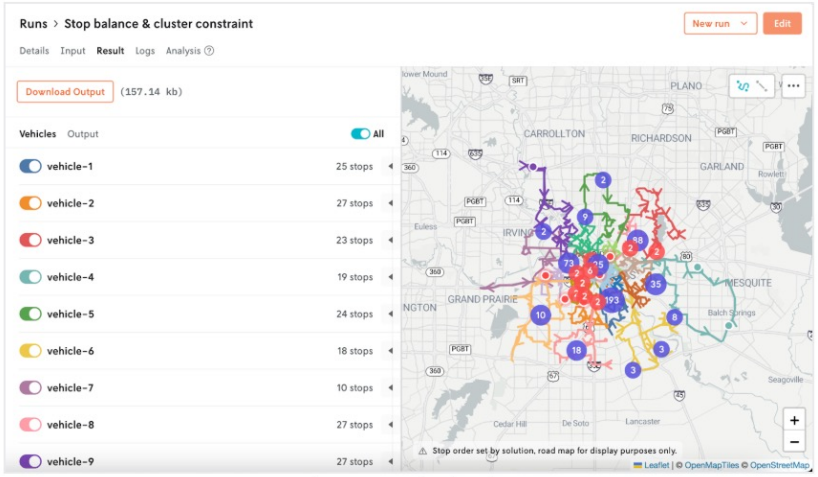
1. **4-Step Model**: Implements trip generation, distribution, and assignment.
2. **GTFS Integration**: Supports editing of routes, blocks, and trips.
3. **Adjustable Parameters**: Allows changes to stop locations and headways.
4. **Congestion Visualization**: Line thickness reflects network load.

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

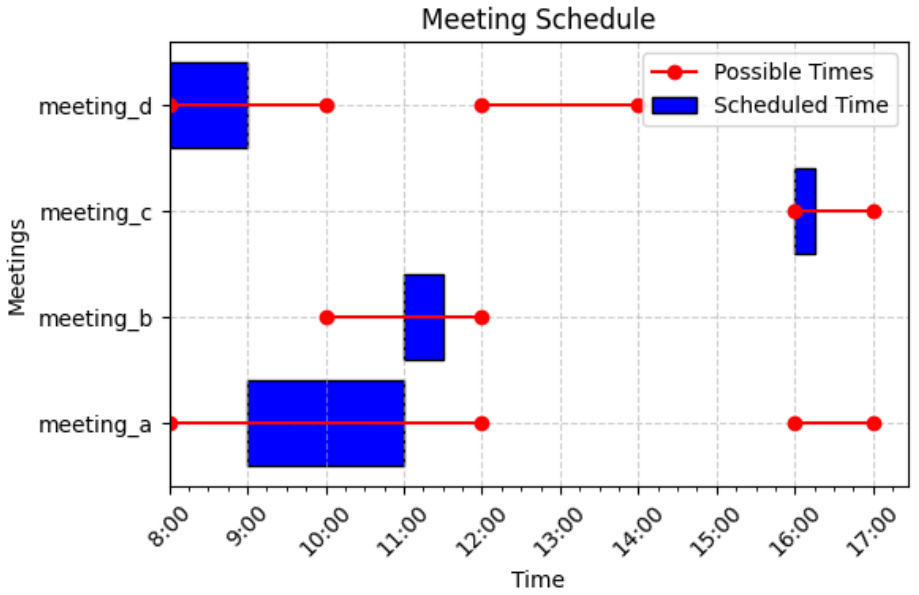
HANDS-ON ACTIVITIES

Meeting Schedule

Traveling Salesman Problem

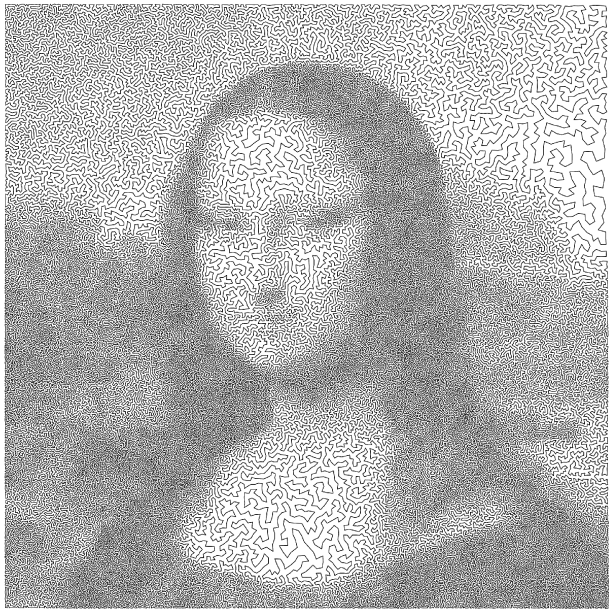
HANDS-ON - MEETING SCHEDULE

- 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/