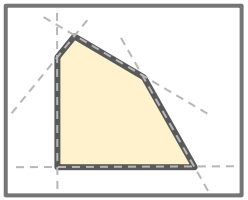
# OR-Tools

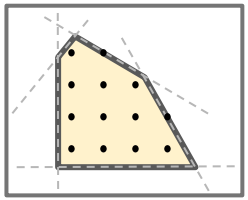
## Open-source from Google

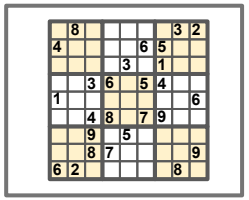
CO@Work 2020, Pawel Lichocki, 25.09.2020

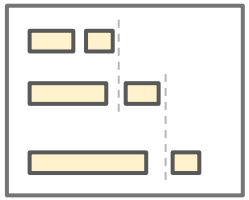
https://developers.google.com/optimization

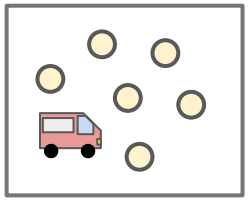
# Combinatorial optimization

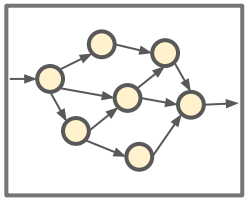




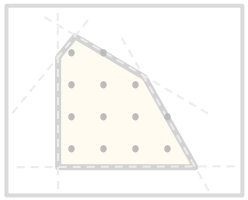
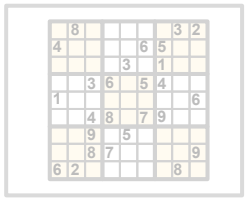


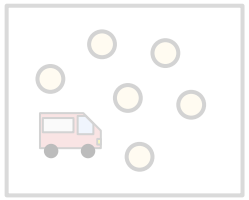







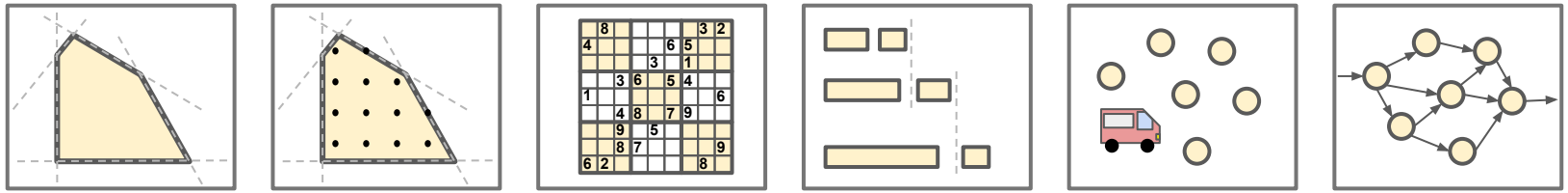
# Solvers

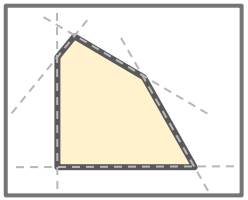
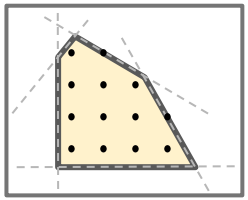
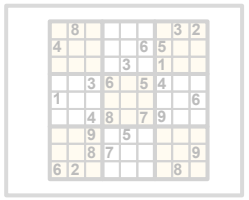



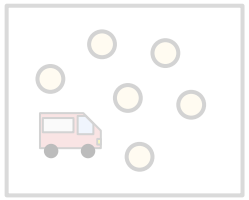

LP MIP SAT CP VRP Graph

# OR-tools




LP MIP SAT CP VRP Graph




8 3 2
4 6 5
3 1
3 6 5 4
1 6
4 8 7 9
9 5
8 7 9
6 2 8



# LP / MIP solver wrapper

```
enum SolverType

 GLOP_LINEAR_PROGRAMMING
 CLP_LINEAR_PROGRAMMING
 GLPK_LINEAR_PROGRAMMING
 GUROBI_LINEAR_PROGRAMMING
 XPRESS_LINEAR_PROGRAMMING
 CPLEX_LINEAR_PROGRAMMING

 SCIP_MIXED_INTEGER_PROGRAMMING
 GLPK_MIXED_INTEGER_PROGRAMMING
 CBC_MIXED_INTEGER_PROGRAMMING
 GUROBI_MIXED_INTEGER_PROGRAMMING
 XPRESS_MIXED_INTEGER_PROGRAMMING
 CPLEX_MIXED_INTEGER_PROGRAMMING

 BOP_INTEGER_PROGRAMMING
 SAT_INTEGER_PROGRAMMING

 KNAPSACK_MIXED_INTEGER_PROGRAMMING
```

# LP / MIP solver wrapper

```
enum SolverType

 GLOP_LINEAR_PROGRAMMING
 CLP_LINEAR_PROGRAMMING
 GLPK_LINEAR_PROGRAMMING
 GUROBI_LINEAR_PROGRAMMING
 XPRESS_LINEAR_PROGRAMMING
 CPLEX_LINEAR_PROGRAMMING

 SCIP_MIXED_INTEGER_PROGRAMMING
 GLPK_MIXED_INTEGER_PROGRAMMING
 CBC_MIXED_INTEGER_PROGRAMMING
 GUROBI_MIXED_INTEGER_PROGRAMMING
 XPRESS_MIXED_INTEGER_PROGRAMMING
 CPLEX_MIXED_INTEGER_PROGRAMMING

 BOP_INTEGER_PROGRAMMING
 SAT_INTEGER_PROGRAMMING

 KNAPSACK_MIXED_INTEGER_PROGRAMMING
```


# ortools/linear_solver/linear_solver.proto

$$\min/\max\ c_0 + c^T x$$
$$lb_{ct} \le Ax \le ub_{ct}$$
$$lb_{var} \le x \le ub_{var}$$
$$x_j \in Z,\ j \in J$$

```
MPModelProto {
  bool maximize
  double objective_offset
  repeated MPVariableProto variable
  repeated MPConstraintProto constraint
}
MPVariableProto {
  double lower_bound
  double upper_bound
  double objective_coefficient
  bool is_integer
}
MPConstraintProto {
  double lower_bound
  double upper_bound
  repeated int32 var_index
  repeated double coefficient
}
```

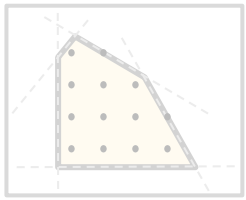
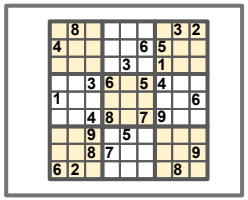
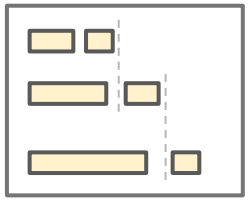
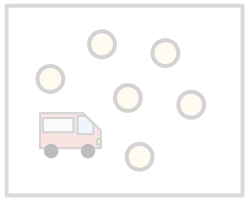
# ortools/linear_solver/linear_solver.proto

$$\max 2x_0 + x_1$$
$$x_0 + x_1 = 1$$
$$x_0 \in \{0, 1\}$$
$$x_1 \in \{0, 1\}$$

```
maximize: true
variable {
  lower_bound: 0.0 upper_bound: 1.0
  objective_coefficient: 2.0
  is_integer: true
}
variable {
  lower_bound: 0.0 upper_bound: 1.0
  objective_coefficient: 1.0
  is_integer: true
}
constraint {
   lower_bound: 1.0 upper_bound: 1.0
   var_index: 0 coefficient: 1.0
   var_index: 1 coefficient: 1.0
}
```

# ortools/linear_solver/linear_solver.proto

| Constraints | Objective |
| --- | --- |
| indicator | quadratic |
| SOS | |
| quadratic | |
| abs | |
| and, or | |
| min, max | |



8 3 2
4 6 5
3 1
3 6 5 4
1 6
4 8 7 9
9 5
8 7 9
6 2 8



# CP-SAT solver

**OR-tools CP-SAT solver at MiniZinc Challenge**

2016: 1 gold

2017: 1 gold + 1 silver

2018: 4 golds

2019: 4 golds

# CP-SAT solver

**OR-tools CP-SAT solver at MiniZinc Challenge**

- 2016: 1 gold
- 2017: 1 gold + 1 silver
- 2018: 4 golds
- 2019: 4 golds

**Closed open MIPLIB 2017 problems**

- amaze22012-07-04i (31s)
- neos-3209462-rhin (87s)
- l2p2i (16s)
- neos-3214367-sovi (341s, vs. solved in 21 days with ParaXpress)
- stoch-vrpvrp-s5v2c8vrp-v2c8i (30s)

# ortools/sat/cp_model.proto

**Constraints**

NoOverlapConstraint

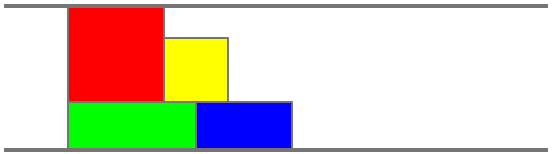
Intervals cannot overlap

# ortools/sat/cp_model.proto

## Constraints

NoOverlapConstraint

CumulativeConstraint

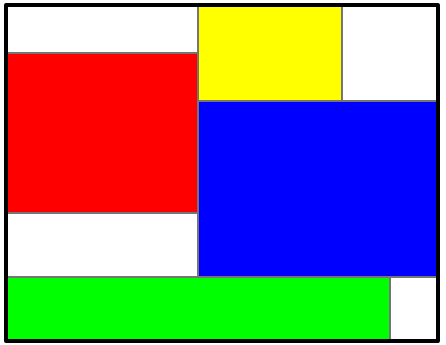


Intervals have heights, resource has capacity. Interval can overlap without overloading capacity

# ortools/sat/cp_model.proto

**Constraints**

- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D



2D boxes cannot overlap

# ortools/sat/cp_model.proto

**Constraints**

- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D
- Boolean constraints

A ∨ B ∨ -C
A => B
A ⇔ -B
A ∧ -B => C

# ortools/sat/cp_model.proto

## Constraints

- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D
- Boolean constraints
- LinearConstraint

3 <= x + 2y + z <= 19

# ortools/sat/cp_model.proto

**Constraints**

- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D
- Boolean constraints
- LinearConstraint (with enforcement)

$3 \leq x + 2y + z \leq 19$

$B \Rightarrow (x \leq 2)$

# ortools/sat/cp_model.proto

## Constraints

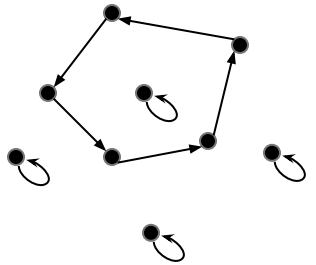
- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D
- Boolean constraints
- LinearConstraint (with enforcement)
- ElementConstraint

target = variables[index]

# ortools/sat/cp_model.proto

**Constraints**

- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D
- Boolean constraints
- LinearConstraint (with enforcement)
- ElementConstraint
- CircuitConstraint



Graph + boolean variables for arcs
Boolean variables must form a (sub) circuit

# ortools/sat/cp_model.proto

## Constraints

- NoOverlapConstraint
- CumulativeConstraint
- NoOverlap2D
- Boolean constraints
- LinearConstraint (with enforcement)
- ElementConstraint
- CircuitConstraint
- TableConstraint

(x, y, z) must be in

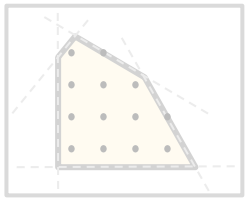
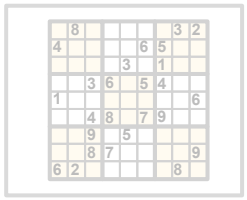
1 1 2
1 3 1
1 3 4
2 1 3
2 4 5

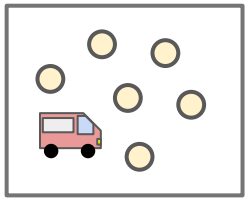
# ortools/sat/cp_model.proto

**Constraints**

NoOverlapConstraint
CumulativeConstraint
NoOverlap2D
Boolean constraints
LinearConstraint (with enforcement)
ElementConstraint
CircuitConstraint
TableConstraint

...



8 3 2
4 6 5
3 1
3 6 5 4
1 6
4 8 7 9
9 5
8 7 9
6 2 8

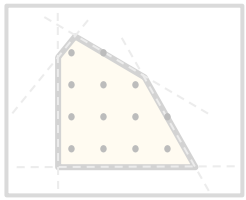
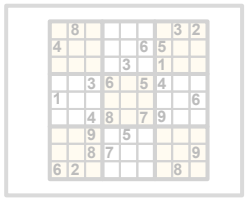


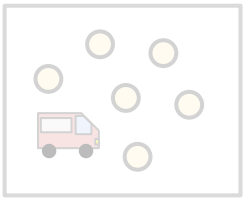
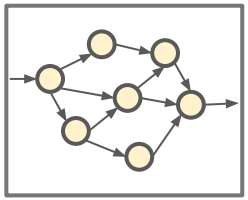
# ortools/constraint_solver/routing.h

```
from ortools.constraint_solver import pywrapcp

def distance(from_index, to_index):
  return from_index + to_index

indexes = pywrapcp.RoutingIndexManager(num_nodes=10, num_vehicles=1, num_depots=0)
routing = pywrapcp.RoutingModel(indexes)
transit = routing.RegisterTransitCallback(distance)
routing.SetArcCostEvaluatorOfAllVehicles(transit)
solution = routing.Solve()
```



8 3 2
4 6 5
3 1
3 6 5 4
1 6
4 8 7 9
9 5
8 7 9
6 2 8

# ortools/graph/min_cost_flow.h

```
from ortools.graph import pywrapgraph

... # Initialize input data.

min_cost_flow = pywrapgraph.SimpleMinCostFlow()

for arc in num_arcs:
  min_cost_flow.AddArcWithCapacityAndUnitCost(from_node[arc], to_node[arc],
                                              capacities[arc], unit_costs[arc])
for node in num_nodes:
  min_cost_flow.SetNodeSupply(node, supplies[node])

if min_cost_flow.Solve() == min_cost_flow.OPTIMAL:
  print('Minimum cost:', min_cost_flow.OptimalCost())
```


LP
MIP
SAT
CP
VRP
Graph

LP solver = primal/dual simplex *(GLOP)*

LP solver = primal/dual simplex *(GLOP)*

CP solver = propagation
VRP solver = CP + heuristics

LP solver = primal/dual simplex *(GLOP)*

CP solver = propagation
VRP solver = CP + heuristics

SAT solver = conflict-driven clause learning
BP solver = heuristics + SAT *(BOP)*
CP-SAT solver = propagation + (lazy) SAT + LP
IP solver = CP-SAT + cuts *(CP-SAT-MIP)*

LP solver = primal/dual simplex *(GLOP)*

CP solver = propagation
VRP solver = CP + heuristics

SAT solver = conflict-driven clause learning
BP solver = heuristics + SAT *(BOP)*
CP-SAT solver = propagation + (lazy) SAT + LP
IP solver = CP-SAT + cuts *(CP-SAT-MIP)*

Graph =
- linear assignment
- max flow
- min cost flow
- matching
- components
- cliques
- ...

LP solver = primal/dual simplex *(GLOP)*

CP solver = propagation
VRP solver = CP + heuristics

SAT solver = conflict-driven clause learning
BP solver = heuristics + SAT *(BOP)*
CP-SAT solver = propagation + (lazy) SAT + LP
IP solver = CP-SAT + cuts *(CP-SAT-MIP)*

Graph =
linear assignment
max flow
min cost flow
matching
components
cliques
...

# Thank you!

https://developers.google.com/optimization