Google AI’s OR-Tools CP-SAT Solver

unofficial tutorial

Bobak Pezeshki

General Info

Home Page: https://developers.google.com/optimization

About: https://developers.google.com/optimization/introduction/overview

CP-SAT Solver

Overview Page: https://developers.google.com/optimization/cp

Intro Use Case: https://developers.google.com/optimization/cp/cp_solver

Documentation:
https://developers.google.com/optimization/reference/python/sat/python/cp_model

Simple Example

Problem Description

Consider three numbers.

Each can be either 0, 1, or 2.

The first two numbers are the same.

What are the different possibilities for the three numbers?

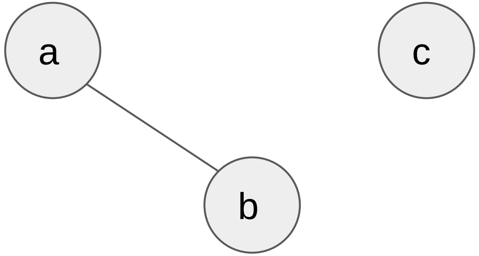
Constraint Model

Variables: $V = \{ a, b, c \}$

Domains: $D = \{ D_a, D_b, D_c \}$, st. $\forall x \in V$, $D_x = \{ 0, 1, 2 \}$

Constraints: a == b

Primal Graph:



“Hello World”

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Import the CP-SAT Package

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Create Model Object

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Add Variables and their Domains to Model

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Add Constraints to Model

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Create CP-SAT Solver Object

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Create A Solution Printer Object *(optional)*

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Solve Problem (and record status)

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Check Status

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Boolean Variables

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

a = model.NewBoolVar('a')
b = model.NewBoolVar('b')
c = model.NewBoolVar('c')

model.Add(a == b)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

AllDiff() Constraint

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.AddAllDifferent([a,b,c])

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

Element Constraints

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

E_ac = [2, 1, 0]
model.AddElement(a, R_ac, c)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

a	c
0	2
1	1
2	0

Relational Constraints

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

R_ac = [(0,2),(1,1),(2,1),(2,2)]
model.AddAllowedAssignments([a,c],R_ac)

solver = cp_model.CpSolver()
printer = cp_model.VarArraySolutionPrinter([a,b,c])
status = solver.SearchForAllSolutions(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "All solutions found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "Some solutions found!" + '\n')
else:
            print('\n' + "No solution could be found!" +
'\n')
```

a	c
0	2
1	1
2	1
2	2

Optimization

```
from ortools.sat.python import cp_model

model = cp_model.CpModel()

num_vals = 3
a = model.NewIntVar(0, num_vals - 1, 'a')
b = model.NewIntVar(0, num_vals - 1, 'b')
c = model.NewIntVar(0, num_vals - 1, 'c')

model.Add(a == b)

model.Maximize( 2*c - a )

solver = cp_model.CpSolver()
printer = cp_model.VarArrayAndObjectiveSolutionPrinter([a,b,c])
status = solver.SolveWithSolutionCallback(model, printer)

if status == cp_model.OPTIMAL:
            print('\n' + "Optimal solution found!" + '\n')
elif status == cp_model.FEASIBLE:
            print('\n' + "A solution found, but may not be optimal." + '\n')
else:
            print('\n' + "No solution found!" + '\n')
```

Extra Practice

Practice Satisfiability Problem

There are three instructors.

Each be teaching one class.

The university only has one room left available with four different time slots.

Model this as a constraint programing problem and print all solutions.

Practice Optimization Problem

Consider the instructor assignment problem from before...

For each professor, the university has also recorded the various time-slot preferences, each of which they assign a preference multiplier to.

	Time-Slot 1	Time-Slot 2	Time-Slot 3	Time-Slot 4
Instr 1	3	2	2	1
Instr 2	1	3	1	2
Instr 3	3	3	1	1

Practice Optimization Problem

Consider the instructor assignment problem from before...

For each professor, the university has also recorded the various time-slot preferences, each of which they assign a preference multiplier to.

Additionally, different professors have different seniority, again corresponding to different multipliers.

	Seniority
Instr 1	5
Instr 2	4
Instr 3	3

Practice Optimization Problem

Consider the instructor assignment problem from before...

For each professor, the university has also recorded the various time-slot preferences, each of which they assign a preference multiplier to.

Additionally, different professors have different seniority, again corresponding to different multipliers.

Using the different multipliers, model this as a constraint programming problem and print the solution that optimizes the sum of product of each professor’s time-slot preference multiplier and seniority multiplier.

Other Notes

Many more kinds of constraints and functions

https://developers.google.com/optimization/reference/python/sat/python/cp_model#top_of_page

You can limit the solver’s time or number of solutions

https://developers.google.com/optimization/cp/cp_tasks

Default Solution Printers use Zero-Based Indexing

```
Solution 0, time = 0.02 s
  a = 2   b = 2   c = 0
Solution 1, time = 0.02 s
  a = 1   b = 1   c = 0
Solution 2, time = 0.05 s
  a = 1   b = 1   c = 1
Solution 3, time = 0.05 s
  a = 2   b = 2   c = 1
Solution 4, time = 0.06 s
  a = 2   b = 2   c = 2
Solution 5, time = 0.09 s
  a = 1   b = 1   c = 2
Solution 6, time = 0.09 s
  a = 0   b = 0   c = 2
Solution 7, time = 0.10 s
  a = 0   b = 0   c = 1
Solution 8, time = 0.10 s
  a = 0   b = 0   c = 0
```

Note there are actually **9** solutions!

You Can Create Your Own Solution Collector Class

Need to be derived from the CpSolverSolutionCallback class.

Ex: (from: https://stackoverflow.com/questions/58934609/obtain-list-of-sat-solutions-from-ortools)

```
class VarArraySolutionCollector(cp_model.CpSolverSolutionCallback):

    def __init__(self, variables):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__variables = variables
        self.solution_list = []

    def on_solution_callback(self):
        self.solution_list.append([self.Value(v) for v in self.__variables])
```