Constraint Programming With Python Using Google OR-Tools

Alan Velasco
June 2018

Setting Up the Environment

Setting Up with Python 3.6 on your laptop

```
$ python -m venv ENV
$ source ENV/bin/activate
(ENV) $ pip install py3-ortools
... Success!
```

This is known to work on recent MacOS and Linux.

Setup using PythonAnywhere (works for all!)

- Create a free *Beginner Account* at PythonAnywhere.com
- Once you've created the account, click on a new Bash Console
- *Hint*: you can use the bash shell to edit files (vi, pico, etc) or use the *Files* app in your PythonAnywhere dashboard.
- Once in, create your virtualenv and install the library.

```
$ mkvirtualenv myvirtualenv --python=/usr/bin/python3.6
$ pip install py3-ortools
```

Introduction to Constraint Programming

- Constraint Programming is a paradigm where relations between variables are defined in the form constraints.
- Constraints are stated in a declarative way, instead of providing step-by-step instructions.
- Constraints can take many forms, like boolean satisfaction or linear inequalities:
  - This or that must happen for my solution to be valid.
  - $X < Y$ and $X + Y = Z$
- Two search strategies: Refinement and Perturbation
- The focus of Constraint Programming is to find feasible solutions in a complex solution domain

Problem 1
The Ranch Corral

Problem Description

Suppose we are visiting a ranch, and in one of the corrals we see 56 legs and 20 heads total of both cows and chickens.

How many cows and how many chickens are we seeing?

Any ideas?

How would a procedural solution look like?

Procedural Solution

```
def success(cows, chickens):
    return (2 * chickens + 4 * cows == 56) and (cows + chickens == 20)

solutions = []
for cows in range(21):
    for chickens in range(21):
        if success(cows, chickens):
            solutions.append((cows, chickens))

for (cows, chickens) in solutions:
    print("{} cows, {} chickens".format(cows, chickens))
```

Introduction to the Google OR-Tools

Google OR-Tools

The Google Operation Research Tools (OR-Tools) is an open source project that includes several optimization algorithms and solvers. For example:

- A general purpose Constraint Programming Solver
- An interface to Linear, Mixed and Integer Programming Solvers (GBC, CLP, GLOP, GLPK, SCIP, etc)
- Graph Algorithms
- Algorithms for the TSP and VRP problems

The library is written in C++, but it also offers wrappers for Python, Java, and C

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

https://github.com/google/or-tools

Solving Problems

1 Understand the Problem

Identify the type of problem

Identify the variables and constraints

Choose the target variable to optimize

2 Model the Problem

Translate variables and constraints to code

Implement support variables and constraints that may not exist in the original scenario

3 Feed the Model to the Solver

Give the CP Solver all the information it needs to find a solution to the problem

Specify which values should be kept after the search.

4 Solve the Problem!

Add phases to the search the Solver will perform

Parametrize the solver instance for optimal performance and quality of solutions

5 Extract & Report the Solution

Extract and decode the values of the obtained solution.

Present the solution in a way the end-user can understand.

Problem 1.1
The Ranch Corral: Take 2

Solving Problems

1

Understand the Problem

Identify the type of problem

Identify the variables and constraints

Choose the target variable to optimize

2

Model the Problem

Translate variables and constraints to code

Implement support variables and constraints that may not exist in the original scenario

3

Feed the Model to the Solver

Give the CP Solver all the information it needs to find a solution to the problem

Specify which values should be kept after the search.

4

Solve the Problem!

Add phases to the search the Solver will perform

Parametrize the solver instance for optimal performance and quality of solutions

5

Extract & Report the Solution

Extract and decode the values of the obtained solution.

Present the solution in a way the end-user can understand.

Problem 2
Sudoku

Hint: Use AllDifferent()

Solving Problems

1 Understand the Problem

Identify the type of problem

Identify the variables and constraints

Choose the target variable to optimize

2 Model the Problem

Translate variables and constraints to code

Implement support variables and constraints that may not exist in the original scenario

3 Feed the Model to the Solver

Give the CP Solver all the information it needs to find a solution to the problem

Specify which values should be kept after the search.

4 Solve the Problem!

Add phases to the search the Solver will perform

Parametrize the solver instance for optimal performance and quality of solutions

5 Extract & Report the Solution

Extract and decode the values of the obtained solution.

Present the solution in a way the end-user can understand.

Problem 3
Einstein’s Riddle

Einstein’s Riddle

1. There are 5 houses in five different colors.
2. In each house lives a person with a different nationality.
3. These five owners drink a certain type of beverage, smoke a certain brand of cigar and keep a certain pet.
4. No owners have the same pet, smoke the same brand of cigar or drink the same beverage.

Einstein’s Riddle

- The Brit lives in the red house
- The Swede keeps dogs as pets
- The Dane drinks tea
- The green house is on the left of the white house
- The green house's owner drinks coffee
- The person who smokes Pall Mall rears birds
- The owner of the yellow house smokes Dunhill
- The man who smokes blend has a neighbor who drinks water
- The man living in the center house drinks milk
- The Norwegian lives in the first house
- The man who smokes blends lives next to The one who keeps cats
- The man who keeps horses lives next to the man who smokes Dunhill
- The owner who smokes BlueMaster drinks beer
- The German smokes Prince
- The Norwegian lives next to the blue house

Solving Problems

1 Understand the Problem

Identify the type of problem

Identify the variables and constraints

Choose the target variable to optimize

2 Model the Problem

Translate variables and constraints to code

Implement support variables and constraints that may not exist in the original scenario

3 Feed the Model to the Solver

Give the CP Solver all the information it needs to find a solution to the problem

Specify which values should be kept after the search.

4 Solve the Problem!

Add phases to the search the Solver will perform

Parametrize the solver instance for optimal performance and quality of solutions

5 Extract & Report the Solution

Extract and decode the values of the obtained solution.

Present the solution in a way the end-user can understand.

Alan Velasco

@alanbato