Python OR-tools Notes

Kunlei Lian

2/18/23

Table of contents

Preface

1 Introduction

This book covers the usage of Google OR-Tools to solve optimization problems in Python. There are several major chapters in this book:

In Chapter 2, we explain the steps needed to setup OR-Tools in a Python environment.

In Chapter 3, we use an example to illustrate the modeling capability of OR-Tools to solve linear programming problems.

In Chapter 4, we go through the modeling techniques made available in OR-Tools.

2 Environment Setup

In this chapter, we explain the steps needed to set up Python and Google OR-Tools. All the steps below are based on MacBook Air with M1 chip and macOS Ventura 13.1.

2.1 Install Homebrew

The first tool we need is Homebrew, 'the Missing Package Manager for macOS (or Linux)', and it can be accessed at https://brew.sh/. To install Homebrew, just copy the command below and run it in the Terminal.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install
```

We can then use the `brew --version` command to check the installed version. On my system, it shows the info below.

```
  ~/ brew --version
Homebrew 3.6.20
Homebrew/homebrew-core (git revision 5f1582e4d55; last commit 2023-02-05)
Homebrew/homebrew-cask (git revision fa3b8a669d; last commit 2023-02-05)
```

2.2 Install Anaconda

Since there are several Python versions available for our use and we may end up having multiple Python versions installed on our machine, it is important to use a consistent environment to work on our project in. Anaconda is a package and environment manager for Python and it provides easy-to-use tools to facilitate our data science needs. To install Anaconda, run the below command in the Terminal.

```
  ~/ brew install anaconda
```

After the installation is done, we can use `conda --version` to verify whether it is available on our machine or not.

```
  ~/ conda --version
conda 23.1.0
```

2.3 Create a Conda Environment

Now we will create a Conda environment named 'ortools'. Execute the below command in the Terminal, which effectively creates the required environment with Python version 3.10.

```
  ~/ conda create -n ortools python=3.10
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/homebrew/anaconda3/envs/test

  added / updated specs:
    - python=3.10


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    setuptools-67.4.0          |     pyhd8ed1ab_0         567 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         567 KB

The following NEW packages will be INSTALLED:

  bzip2              conda-forge/osx-arm64::bzip2-1.0.8-h3422bc3_4
  ca-certificates    conda-forge/osx-arm64::ca-certificates-2022.12.7-h4653dfc_0
  libffi             conda-forge/osx-arm64::libffi-3.4.2-h3422bc3_5
  libsqlite          conda-forge/osx-arm64::libsqlite-3.40.0-h76d750c_0
  libzlib            conda-forge/osx-arm64::libzlib-1.2.13-h03a7124_4
  ncurses            conda-forge/osx-arm64::ncurses-6.3-h07bb92c_1
  openssl            conda-forge/osx-arm64::openssl-3.0.8-h03a7124_0
  pip                conda-forge/noarch::pip-23.0.1-pyhd8ed1ab_0
  python             conda-forge/osx-arm64::python-3.10.9-h3ba56d0_0_cpython
```

```
  readline           conda-forge/osx-arm64::readline-8.1.2-h46ed386_0
  setuptools         conda-forge/noarch::setuptools-67.4.0-pyhd8ed1ab_0
  tk                 conda-forge/osx-arm64::tk-8.6.12-he1e0b03_0
  tzdata             conda-forge/noarch::tzdata-2022g-h191b570_0
  wheel              conda-forge/noarch::wheel-0.38.4-pyhd8ed1ab_0
  xz                 conda-forge/osx-arm64::xz-5.2.6-h57fd34a_0


Proceed ([y]/n)?
```

Type 'y' to proceed and Conda will create the environment for us. We can use `cnoda env list` to show all the created environments on our machine:

```
 ~/ conda env list
# conda environments:
#
base                     /opt/homebrew/anaconda3
ortools                   /opt/homebrew/anaconda3/envs/ortools
```

Note that we need to manually activate an environemnt in order to use it: `conda activate ortools`. On my machine, the activated environment `ortools` will appear in the beginning of my prompt.

```
 ~/ conda activate ortools
(ortools)  ~/
```

2.4 Install Google OR-Tools

As of this writing, the latest version of Google OR-Tools is 9.5.2237, and we can install it in our newly created environment using the command `pip install ortools==9.5.2237`. We can use `conda list` to verify whether it is available in our environment.

```
(ortools)  ~/ conda list
# packages in environment at /opt/homebrew/anaconda3/envs/ortools:
#
# Name                    Version                   Build  Channel
absl-py                   1.4.0                     pypi_0    pypi
bzip2                     1.0.8                h3422bc3_4    conda-forge
ca-certificates           2022.12.7            h4653dfc_0    conda-forge
libffi                    3.4.2                h3422bc3_5    conda-forge
```

```
libsqlite                 3.40.0               h76d750c_0    conda-forge
libzlib                   1.2.13               h03a7124_4    conda-forge
ncurses                   6.3                  h07bb92c_1    conda-forge
numpy                     1.24.2                   pypi_0    pypi
openssl                   3.0.8                h03a7124_0    conda-forge
ortools                   9.5.2237                 pypi_0    pypi
pip                       23.0.1             pyhd8ed1ab_0    conda-forge
protobuf                  4.22.0                   pypi_0    pypi
python                    3.10.9        h3ba56d0_0_cpython    conda-forge
readline                  8.1.2                h46ed386_0    conda-forge
setuptools                67.4.0             pyhd8ed1ab_0    conda-forge
tk                        8.6.12               he1e0b03_0    conda-forge
tzdata                    2022g                h191b570_0    conda-forge
wheel                     0.38.4             pyhd8ed1ab_0    conda-forge
xz                        5.2.6                h57fd34a_0    conda-forge
```

Now we have Python and Google OR-Tools ready, we can start our next journey.

3 Linear Programming

In this chapter, we first go through the modeling capabilities provided by Google OR-Tools to solve linear programming problems. Then we get our hands dirty by solving some linear programming problems.

3.1 Modeling Capabilities

There are three components in a mathematical model, namely, decision varaibles, constraints and objective, for which we will go over in the following sections.

3.1.1 Solver

In Google OR-Tools, a `Solver` instance must be created first so that variables, constraints and objective can be added to it. The `Solver` class is defined in the `ortools.linear_solver.pywraplp` module and it requires a solver id to instantiate an object. In the code snippet below, the required module is imported first and a `solver` object is created with `GLOP`, Google's own optimization engine for solving linear programming problems. It is good practice to verify whether the desired solver is indeed created successfully or not.

```
from ortools.linear_solver import pywraplp

solver = pywraplp.Solver.CreateSolver("GLOP")

if solver:
    print("solver creation success!")
else:
    print("solver creation failure!")
```

```
success
```

3.1.2 Decision Variables

The `Solver` class defines a number of ways to create decision variables:

- `Var(lb, ub, integer, name)`
- `NumVar(lb, ub, name)`
- `IntVar(lb, ub, name)`
- `BoolVar(name)`

3.1.2.1 Function `Var()`

The `Var()` method is the most flexible way to define variables, as it can be used to create numerical, integral and boolean variables. In the following code, a numerical variable named 'var1' is created with bound (0.0, 1.0). Note that the parameter `integer` is set to `False` in the call to function `Var()`.

```
var1 = solver.Var(lb=0, ub=1.0, integer=False, name="var1")
```

We could create an integer variable using the same function:

```
var2 = solver.Var(lb=0, ub=1.0, integer=True, name="var2")
```

3.1.2.2 Function `NumVar()`

`var1` could be creatd alternatively using the specialized function `NumVar()`:

```
var1 = solver.NumVar(lb=0, ub=1.0, name='var1')
```

3.1.2.3 Function `IntVar()`

Similarly, `var2` could be created alternatively using the specialized function `IntVar()`:

```
var2 = solver.IntVar(lb=0, ub=1.0, name='var2')
```

3.1.2.4 Function `BoolVar()`

A boolean variable could be created using the `BoolVar()` function:

```
var3 = solver.BoolVar(name='var3')
```

3.1.3 Constraints

3.1.4 Objective

3.2 Applications

3.2.1 An Artificial Example

3.2.2 The Stigler Diet Problem

4 Integer Programming

5 Column Generation

6 Summary

In summary, this book has no content whatsoever.

References