Introduction to Google OR-Tools for ENGRI 1101 *

1 Installation

The quickest way to get OR-Tools working for Python is to install the binary version. This can be done by running the following in terminal:

```
python -m pip install --upgrade --user ortools
```

However, to use OR-Tools with a third-party solver (such as Gurobi), it must be compiled from source. Google offers reasonably detailed instructions on doing so for Linux, MacOS, and Windows. However, some additional steps may be necessary. Here, we give a full set of instructions to successfully use Gurobi through OR-Tools on both MacOS and Windows.
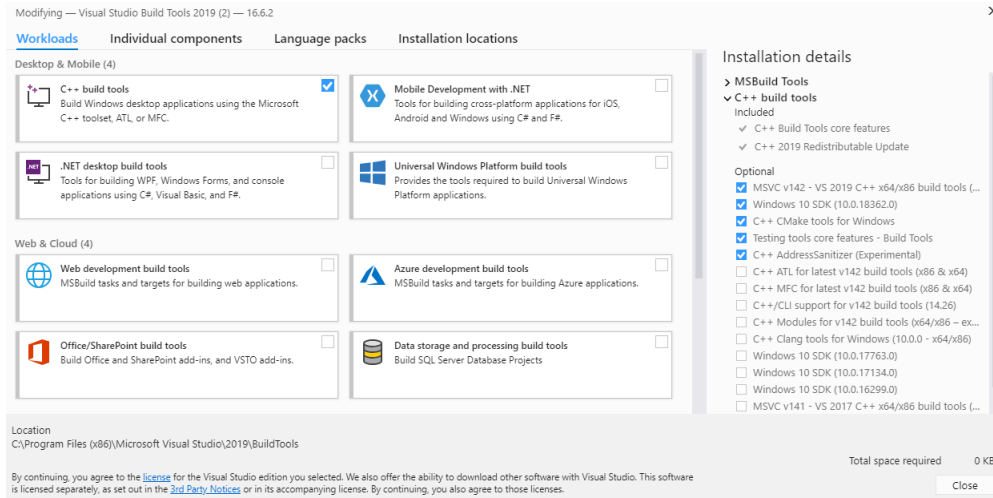
1.1 Windows

These instructions are adapted from those presented by David Williamson and Jody Zhu.

1. Install Visual Studio 2019 Community and Build Tools for Visual Studio 2019. Your Visual Studio Installer should look like this:
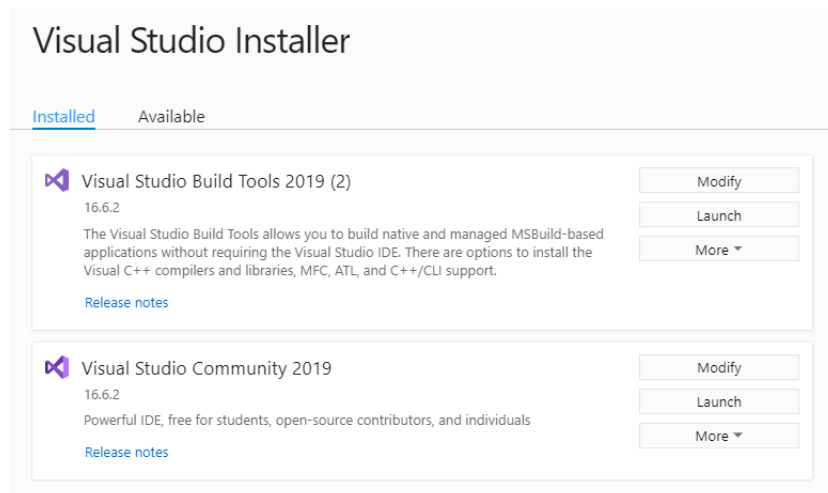

2. Click "Modify" for Build Tools. Then check "C++ build tools."

---

*Currently set up for use by course developers. Should be updated and reworked for 1101 students.

3. Install cmake using "Windows win64-x64 Installer" under Binary distributions.

4. Make sure you have the correct Python and Git.

5. Run `git clone -b master https://github.com/google/or-tools` to get the master branch of the source code (Any command prompt can be used here which is **not** the case for the next step).

6. Open x64 Native Tools Command Prompt that is in the Visual Studio 2019 folder
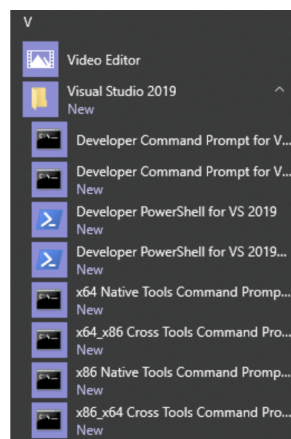
7. Navigate to your "or-tools" folder (cd .. to go up and cd "name" to go down)

8. Run `tools\make third_party`

9. *(Assumes Gurobi has been installed correctly with a license)* Edit the `Makefile.local` file. Note that `WINDOWS_GUROBI_DIR` might be different depending where "gurobi902" is located.

10. Run `tools\make python`

11. This stopped with an error for me, so I edited `Makefile.win.mk` on the line for STATIC_SCIP_LNK to change `scip.lib` to `libscip.lib`. Then I reran `tools\make python`, and this worked.

12. I need to install a package wheel, so ran `pip install wheel`

13. Run `tools\make install_python`

14. If you previously installed ortools from binary, run `pip uninstall ortools`

1.2   MacOS

These instructions are adapted from those presented by David Williamson

1. Run `xcode-select --install`

2. Verify `xcode-select -p` returns `/Applications/Xcode.app/Contents/Developer`

3. Install Homebrew:
   `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

4. Update Homebrew: `brew update`

5. Verify `brew --version` returns
   `Homebrew 1.6.9-8-g25542d7`
   `Homebrew/homebrew-core (git revision 0e0c84; last commit 2018-06-20)`

6. Install the remaining prerequisites:

   (a) Run `brew install cmake wget pkg-config`

   (b) Run `brew install swig`

   (c) Run `brew install python`

   (d) Run `python3 -m pip install -U --user wheel six`

7. Run `git clone -b master https://github.com/google/or-tools` to get the source master branch

8. Navigate to installation location

9. Run `make third_party`

10. *(Assumes Gurobi has been installed correctly with a license)* Edit the `Makefile.local` file. Note that UNIX_GUROBI_DIR might be different depending where "gurobi902" is located.

```
# Define UNIX_GUROBI_DIR and GUROBI_LIB_VERSION to use Gurobi
UNIX_GUROBI_DIR = /Library/gurobi902/
GUROBI_LIB_VERSION = 90
```

11. Run `make python`

12. Run `make install_python`

13. You may have to edit a line in in `makefiles\Makefile.python.mx` by adding `--prefix=` at the end

```
install_python: pypi_archive
    cd "$(PYPI_ARCHIVE_TEMP_DIR)$Sortools" && "$(PYTHON_EXECUTABLE)" setup.\
py install --user --prefix=
```

14. If you previously installed ortools from binary, run `pip uninstall ortools`

2 Documentation

Google offers decent documentation. Here, we introduce the subset of functionality that 1101 students will use. Furthermore, some standard style guidelines are established[1].

2.1 Imports

All Jupyter Notebooks should begin with the following imports named accordingly. Both `numpy` and `math` are standard imports. `pandas` is used for manipulating CSV files which act as AMPL dat files. `itertools` is used for taking the cartesian product of two or more sets. Lastly, `pywraplp` is the package used for creating and solving LPs.

```python
#imports
import numpy as np
import math as math
import pandas as pd
import itertools as it
from ortools.linear_solver import pywraplp as OR
```

2.2 Model

To solve an LP or MIP, we first create a `Solver` similar to an AMPL model.

```
m = OR.Solver(<name>, <problem_type>)
```

instantiates a solver where `problem_type` declares which solver is used to solve the model. Assuming OR-Tools was complied from source for use with Gurobi, the available options for `problem_type` are

- `GLOP_LINEAR_PROGRAMMING`: An open-source LP solver (by Google)

- `CBC_MIXED_INTEGER_PROGRAMMING`: An open-sorce MIP solver

- `GUROBI_MIXED_INTEGER_PROGRAMMING`: A leading commercial MIP solver

For example, we can define a model named "ex" that uses Gurobi to solve mixed integer programs:

```python
m = OR.Solver('ex', OR.Solver.GUROBI_MIXED_INTEGER_PROGRAMMING)
```

Use `m.EnableOutput()` to see solver output. (This goes to terminal right now. Looking into a better way)

---

[1]Style guidelines are adapted from those written by Jody Zhu

2.3 Sets and Parameters

Similar to AMPL, sets should be named in uppercase. In Python, they are represented by a list. Some helpful commands are `SET.append(x)` which adds x to the SET and `SET1.extend(SET2)` which adds elements of SET2 to SET1. Furthermore, `list(it.product(SET1,SET2))` returns the cartesian product of SET1 and SET2. Parameters should be named in lowercase. It is good practice to keep both set and parameter names no more than 6 characters. Parameters that have values for every element of a set should be stored as a Python dictionary. Here is an example for a max flow formulation (which will serve as a running example).

```
NODES = ['S',2,3,4,5,6,'T']
EDGES = [('S',2), ('S',3), (2,3), (2,4), (2,5), (3,5),
         (3,6), (4,'T'), (5,4), (5,6), (5,'T'),(6,'T')]
s = 'S'
t = 'T'
cap = {('S',2):1, ('S',3):7, (2,3):5, (2,4):6, (2,5):2, (3,5):2,
              (3,6):4, (4,'T'):5, (5,4):1, (5,6):4, (5,'T'):4, (6,'T'):3}
```

2.4 Variables

To add an additional variable to a model, we use the method `m.NumVar(<lb>, <ub>, <name>)` for continuous variables and `m.IntVar(<lb>, <ub>, <name>)` for integer variables. In both cases, lb and ub refer to the lower and upper bound for that variables respectively. Note that `m.infinity()` and `-m.infinity()` can be used when there is no upper or lower bound. The decision variables for the max flow formulation are instantiated as follows.

```
f = {}
for i,j in EDGES:
    f[i,j] = m.NumVar(0, m.infinity(), ('(%s, %s)' % (i,j)))
```

2.5 Objective Function

The objective function is specified by passing an expression in the variables to `m.Maximize(<expression>)` or `Minimize(<expression>)` depending on the direction of optimization. Often, expressions consist of a summation. In Python, `sum(x[i] for i in SET if <boolean expression>)` is useful for summing over elements. The objective of max flow is

```
m.Maximize(sum(f[i,j] for i,j in EDGES if j == sink))
```

2.6 Constraints

To add a constraint to the model, we use `m.Add(<boolean expression>)` where the constraint is represented by some boolean expression in the variables. The only comparison operators that should be used are <=, >=, or ==. It is best practice to have some comment describing what each constraint does as seen below.

```
# subject to: edge capacity
for i,j in EDGES:
    m.Add(f[i,j] <= capacity[i,j])

# subject to: flow conservation
for k in VERTICES:
    if k != sink and k != source:
        flowIn = sum(f[i,j] for i,j in EDGES if j == k)
        flowOut = sum(f[i,j] for i,j in EDGES if i == k)
        m.Add(flowIn == flowOut)
```

2.7   Solve

To solve the model, we use `m.Solve()`. The optimal value is then retrieved using `m.Objective().Value()`. By iterating through `m.variables()`, one can examine the value of each variable using method `.solution_value()`. An example solution output is:

```
m.Solve()
print('Solution:')
print('Objective value =', m.Objective().Value())
for i,j in EDGES:
    print(f[i,j].name(),':', f[i,j].solution_value())
```

```
Solution:
Objective value = 6.0
(S, 2) : 1.0
(S, 3) : 5.0
(2, 3) : 0.0
(2, 4) : 1.0
(2, 5) : 0.0
(3, 5) : 2.0
(3, 6) : 3.0
(4, T) : 1.0
(5, 4) : 0.0
(5, 6) : 0.0
(5, T) : 2.0
(6, T) : 3.0
```

2.8   Early Termination

One way to terminate early is by setting a time limit. This can be done using `m.SetTimeLimit(<time>)` where `<time>` is the time limit in milliseconds. For MIPs, one can terminate by the relative MIP gap defined as

$$\frac{|\texttt{best\_bound} - \texttt{incumbent}|}{|\texttt{incumbent}|}.$$

This is done as follows:

```
param = OR.MPSolverParameters()
param.SetDoubleParam(param.RELATIVE_MIP_GAP, <gap>)
```

where `<gap>` is the gap to terminate at (say 0.05 for a solution within 5% of optimal). Note, when running `m.Solve()`, you must pass these parameters using `m.Solve(param)`. If one terminates by time limit, the MIP gap can be ascertained through

```python
incumbent = m.Objective().Value()
best_bound = m.Objective().BestBound()
if not incumbent == 0:
    gap = abs(best_bound - incumbent) / abs(incumbent)
else:
    gap = -1
```

If the time limit is set too small, the solver may not reach a feasible solution. In this case, every variable is set to zero. This often results in an objective value of zero. Hence, to prevent a divide by zero error when dividing by the incumbent, the gap is set to -1 (may be better ways to handle this...)

2.9   Solver-Specific Parameters

Gurobi offers an extensive set of parameters that "control the operation of the Gurobi solver." All of these parameters can be set through the OR-Tools interface using `m.SetSolverSpecificParametersAsString(<param_file>)` where `param_file` is a PRM file. Below is an example of some possible termination parameters that could be set.

```python
m = OR.Solver('ex', OR.Solver.GUROBI_MIXED_INTEGER_PROGRAMMING)
m.SetSolverSpecificParametersAsString('''TimeLimit 3000 \n
                                         NodeLimit 100 \n
                                         MIPGap 0.05''')
```

(To my knowledge, one can not set general parameters for CBC yet)