IBM Software Group

CPLEX Optimization Modeling using Python

Guang Feng (gfeng@us.ibm.com)
Nikhila Arkalgud (narkalgu@us.ibm.com)
Technical Support Engineers, Level 2
21 October 2014

WebSphere® Support Technical Exchange

ON DEMAND BUSINESS™

Agenda

- Introduction to CPLEX Python API
- Python API Functionalities
- Debugging in Python API
- Some learning based on past PMRs and forums
- Tips for Programming Large Models
- Conclusion

Introduction to CPLEX Python API

Overview of CPLEX Connectors

- CPLEX core routines are coded in C.
- CPLEX has a tool interactive optimizer for quick diagnostics
- CPLEX provides various wrappers and connectors for other programming languages and applications.
  - C++
  - Java
  - .NET
  - MATLAB
  - **Python**
  - Excel

Why Python API?

- Open source scripting language
- Easy to learn and maintain
- Provides procedural, functional and object-oriented paradigms
- Interpreted language
- Provides other mature libraries for visualizations, statistical analysis and other scientific computing
- Combined capabilities of other CPLEX APIs and Interactive CPLEX
  - Can be used instead of interactive CPLEX
  - Has capabilities to do iterative solves, callbacks like other CPLEX APIs

Setting up CPLEX Python API

- Setup CPLEX Python API
  - Default installation
    - use the script **`setup.py`** to invoke the **`distutils`**

```
python setup.py install
```

    - set **`PYTHONPATH`** to

```
yourCPLEXhome/python/PLATFORM
```

  - To customize location of CPLEX Python modules
    - use the script **`setup.py`** to invoke the **`distutils`**

```
python setup.py install --home yourPythonPackageshome/cplex
```

Overview of API structure

- CPLEX Python API is provided under python package **`cplex`**
- Under package **`cplex`**
  - Key class is **`cplex.Cplex`**
  - Module **`cplex._internal`**
  - Module **`cplex.callbacks`**
  - Module **`cplex.exceptions`**
  - Classes **`cplex.SparsePair`** and **`cplex.SparseTriple`**
  - Constant **`cplex.infinity`**
  - Function **`cplex.terminate`**

Small sample to import and solve a model

```
import cplex
import sys

def sample1(filename):

    c = cplex.Cplex(filename)

    try:
        c.solve()
    except CplexSolverError:
        print "Exception raised during solve"
        return

    # solution.get_status() returns an integer code
    status = c.solution.get_status()
    print "Solution status = " , status, ":",
    print c.solution.status[status]

    print "Objective value = " , c.solution.get_objective_value()
```

Python API Functionalities

Modifying and querying model data

- Modify model variables and constraints

```
c.variables.add(names=["new_var"], lb=[1.0])
c.variables.get_index("new_var")
c.variables.set_upper_bounds("new_var",10.0)
```

- Some of the key interface classes
  - **VariablesInterface**
  - **LinearConstraintInterface**
  - **ObjectiveInterface …**
- Access solutions values interactively using Lambda functions

```
>>> print filter(lambda x:x[1]!=0, zip(c.variables.get_names(),c.solution.get_values()))
```

Querying solution values

- Access solutions values through a function

```
def access_solution_values(c):
    for i, x in enumerate(c.solution.get_values()):
        if (x!= 0):
            print "Solution value of ",c.variables.get_names(i), " = ",x
```

- Access solutions values interactively using Lambda functions

```
>>> print filter(lambda x:x[1]!=0, zip(c.variables.get_names(),c.solution.get_values()))
```

- For code reusability, it is advised to use functions

Managing CPLEX Parameters

- Use the `ParameterInterface` under `Cplex` class

- Setting CPLEX parameters

```
c = cplex.Cplex()
c.parameters.lpmethod.set(c.parameters.lpmethod.values.dual)
```

- Query Parameter values

```
c.parameters.lpmethod.get()
c.parameters.simplex.tolerances.markowitz.max()
c.parameters.simplex.tolerances.markowitz.default()
```

Using Callbacks

- Use the `callbacks` module under `cplex` package

- Some of the callback modules available

```
callbacks.SimplexCallback
callbacks.MIPInfoCallback
callbacks.HeuristicCallback …
```

Small sample using Callbacks

```
import cplex
from cplex.callbacks import SolveCallback
import sys

class MySolve(SolveCallback):
    def __call__(self):
        self.times_called += 1
        if self.get_num_nodes() < 1:
            self.solve(self.method.primal)
        else:
            self.solve(self.method.dual)
        status = self.get_cplex_status()
        self.use_solution()

def sample3(filename):
    c = cplex.Cplex(filename)
    solve_instance  = c.register_callback(MySolve)
    solve_instance.times_called = 0
    try:
        c.solve()
    except CplexSolverError:
        print "Exception raised during solve"
        return
    print "Objective value = " , c.solution.get_objective_value()
```

Debugging in Python API

Log Message handling

- CPLEX specifies the below output streams:
  - **log** and **results** streams are set to `stdout`
  - **warning** and **error** stream to `stderr`
- Redirect to a specific logfile:

```
f = "/path/to/your/logfile.txt"
c = cplex.Cplex()
c.set_results_stream(f)
c.set_warning_stream(f)
c.set_error_stream(f)
c.set_log_stream(f)
```

- You can disable the output with: `set_xxx_stream(None)`

Direct access to histogram of non zero counts

- Formatted histogram reports available
- Access through python session interactively or through a script
- Two report types:
  - Constraints (rows) based:

```
c.linear_constraints.get_histogram()
 Nonzero Count:    2    3    4    5   10   37
Number of Rows:   36    1    9    1    4    1
```

  - Variables (columns) based:

```
c.variables.get_histogram()
    Nonzero Count:    1    2    4   11
Number of Columns:    1    2   36    4
```

Data consistency check and cleanup methods

- Data consistency check parameter
  - In Python API data check is turned **ON** by default

    ```
    c.parameters.read.datacheck.default()
    ```
  - Helps track bogus data

- Data cleanup method
  - Useful to zero out small values
  - Helps in handling numerically unstable models

    ```
    c.cleanup(epsilon)
    ```

High precision display of non zero values

```
for i, x in enumerate(c.solution.get_values()):
        if (x!= 0):
            print "Solution value of ",c.variables.get_names(i), " = ", " %+18.16e" %x
```

```
Solution value of  cost  =   +4.9900000000000000e+02
Solution value of  fixed  =   +2.8000000000000000e+02
Solution value of  transport  =   +2.1900000000000000e+02
Solution value of  x1  =   +1.0000000000000000e+00
Solution value of  x2  =   +1.0000000000000000e+00
```

Invoking the Tuning Tool

- To tune a given CPLEX model:

```
c.parameters.tune_problem()

c.parameters.tune_problem([(c.parameters.lpmethod, 0)])
```

- To tune a set of CPLEX models:

```
c.parameters.tune_problem_set(["lpex.mps“,
  "example.mps"])

c.parameters.tune_problem_set(["lpex.mps",
  "example.mps"],
  fixed_parameters_and_values=[(c.parameters.lpmethod,
  0)])
```

Some learning based on past PMRs

Some learning based on past PMRs

- Performance drop when using control callbacks in Python API
  - In Python parallel callbacks end up running sequentially
  - CPython uses GIL (Global Interpreter Lock) to prevent multiple native threads from executing Python bytecodes at once
  - Compared to other APIs you may see some performance drop when using parallel callbacks with Python APIs
- Duplicate names for variables
  - Unlike Concert APIs, there is no automatic merging of duplicate variables in a constraint
  - Use data check parameter to ensure no duplicate variables are present
- For faster access, reference variables using indexes instead of constraint names

Tips for Programming Large Models

Tips for Programming Large Models

- Some concert best practices programming conventions still applies
  - Batching preferred

- Manage variables/constraints by indices

- Program in Python style

- Python has a built-in profiler

Concert Best Practices

- Batching preferred

```
//slower
for i in range(range1):
    for j in range(range2):
        c.linear_constraints.add(lin_expr = [ ... ], senses = ["E"],
  rhs = [1])


//faster
c.linear_constraints.add(lin_expr = [ ... for i in range(range1)],
                         senses   = ["E"] * range1,
                         rhs      = [1.0] * range1)
```

Manage variables/constraints by indices

- With names, variable/constraint creation can be much slower.

```
//slower
c.variables.add(obj = ..., lb = ..., ub = ..., types = ..., names =
   ...)
//faster
c.variables.add(obj = ..., lb = ..., ub = ..., types = ...)
```

- Names can be added later.

```
c.variables.set_names([(2, "third"), (1, "second")])
```

- Referencing variables/constraints by indices is also faster.
  - Also reduce confusion, as CPLEX Python API won’t merge variables with same names.

Some benchmarks on model generation

Model Size	Default	Batching	Batching and w/o Name
7500	22	13	0.24
15000	85	51	0.49
20000	150	93	0.70
30000	349	207	1.04

Program in Python style

- Python has some unique features and syntaxes not available in other programming languages
  - Lambda expressions

`lambda x: 0 if abs(x)<=1e-12 else x`

  - List processing

`map(lambda x: 0 if abs(x)<=1e-12 else x, coefs)`

  - Generator: no list population. Generate one value each time.
    - `range` vs `xrange`
    - `yield` keyword
  - Functions/packages provide convenience
- Sometimes for performance, sometimes for writing more compact/intuitive codes.

Python Profiler – cProfile, pstats

- Built-in Profiler
  - Command line:

```
python -m cProfile [-o profile.log]script.py
```

  - Within code:
    - Function `enable()` to start collecting data
    - Function `disable()` to stop collecting data
- Try to read the screen outputs and find the lines with significant numbers, or
- Export logs and use package `pstats` or others to analyze logs.

Python Profiler – cProfile, pstats

```
C:\>python -m pstats profile.log
Welcome to the profile statistics browser.
profile.log% sort cumulative
profile.log% stats 20
Thu Oct 16 13:39:42 2014    profile.log

         18765875 function calls (18765546 primitive calls) in 365.513 seconds

   Ordered by: cumulative time
   List reduced from 536 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.303    0.303  365.513  365.513 example_cplex.py:2(<module>)
        1    1.280    1.280  364.317  364.317 example_cplex.py:39(example)
    30010    0.579    0.000  345.762    0.012 C:\Python27\lib\site-
packages\cplex\_internal\_subinterfaces.py:1127(add)
    30010    0.150    0.000  343.763    0.011 C:\Python27\lib\site-
packages\cplex\_internal\_matrices.py:66(__init__)
    30010    0.055    0.000  343.599    0.011 C:\Python27\lib\site-
packages\cplex\_internal\_procedural.py:76(Pylolmat_to_CHBmat)
    30010  338.273    0.011  343.544    0.011
{cplex._internal.py27_cplex1260.Pylolmat_to_CHBmat}
        1    0.000    0.000    9.702    9.702 C:\Python27\lib\site-
packages\cplex\__init__.py:927(solve)
        1    0.000    0.000    9.673    9.673 C:\Python27\lib\site-
packages\cplex\_internal\_procedural.py:422(mipopt)
```

Conclusion

- We introduced CPLEX Python connector API
- We discussed how to use Python API to perform some common tasks, especially for debugging purpose.
- We discussed some learning from past PMRs
- We also discussed how to program large models in an efficient way.

Further Readings

- CPLEX Python API Reference Manual
  http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r6/topic/ilog.odms.cplex.help/refpythoncplex/html/help.html

- Concert best practices programming conventions
  http://www.ibm.com/support/docview.wss?uid=swg21400056

- Presentations of IBM ILOG CPLEX Optimization Studio and IBM Decision Optimization Center/ODM Enterprise
  http://www.ibm.com/support/docview.wss?uid=swg21647915

- IBM RFE Community
  http://www.ibm.com/developerworks/rfe/

Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at: http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at: http://www.ibm.com/developerworks/websphere/community/
- Join the Global WebSphere Community: http://www.websphereusergroup.org
- Access key product show-me demos and tutorials by visiting IBM Education Assistant: http://www.ibm.com/software/info/education/assistant
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically: http://www.ibm.com/software/websphere/support/d2w.html
- Sign up to receive weekly technical My Notifications emails: http://www.ibm.com/software/support/einfo.html

Connect with us!

1. **Get notified on upcoming webcasts**
   Send an e-mail to wsehelp@us.ibm.com with subject line "wste subscribe" to get a list of mailing lists and to subscribe

2. **Tell us what you want to learn**
   Send us suggestions for future topics or improvements about our webcasts to wsehelp@us.ibm.com

3. **Be connected!**
   Connect with us on Facebook
   Connect with us on Twitter

Questions and Answers

This Support Technical Exchange session will be recorded and a replay will be available on IBM.COM sites and possibly social media sites such as YouTube. When speaking, **do not state any confidential information, your name, company name or any information you do not want shared publicly in the replay**. By speaking in during this presentation, you assume liability for your comments.