Using the ODH-CPLEX Python Interface

Alkis Vazacopoulos
Robert Ashford
Optimization Direct Inc.
November 2018
Summary

• Look at simple CPLEX Python examples
  • ‘in Python’ model
  • file-resident model
• See how to use ODH|CPLEX instead
• Combine ODH|CPLEX and CPLEX
  • Joint run
• Combine ODH heuristic engine and CPLEX
  • Feed CPLEX with ODH engine solution
• Using ODH|CPLEX call-backs
A Tiny CPLEX example

Maximize \[ x_1 + 2x_2 + 3x_3 + x_4 \]
Subject to
\[-x_1 + x_2 + x_3 + 10x_4 \leq 20\]
\[x_1 - 3x_2 + x_3 \leq 30\]
\[x_2 - 3.5x_4 = 0\]
Bounds
\[0 \leq x_1 \leq 40\]
\[0 \leq x_2\]
\[0 \leq x_3\]
\[2 \leq x_4 \leq 3\]
Integers
\[x_4\]
import sys
import cplex
# data common to all populateby functions
my_obj = [1.0, 2.0, 3.0, 1.0]
my_ub = [40.0, cplex.infinity, cplex.infinity, 3.0]
my_lb = [0.0, 0.0, 0.0, 2.0]
my_ctype = "CCCI"
my_colnames = ["x1", "x2", "x3", "x4"]
my_rhs = [20.0, 30.0, 0.0]
my_rownames = ["r1", "r2", "r3"]
my_sense = "LLE"

c = cplex.Cplex()
c.objective.set_sense(c.objective.sense.maximize)
c.variables.add(obj=my_obj, lb=my_lb, ub=my_ub, types=my_ctype,
                 names=my_colnames)
rows = [["x1", "x2", "x3", "x4"], [-1.0, 1.0, 1.0, 10.0]],
       [["x1", "x2", "x3"], [1.0, -3.0, 1.0]],
       [["x2", "x4"], [1.0, -3.5]]
c.linear_constraints.add(lin_expr=rows, senses=my_sense,
                          rhs=my_rhs, names=my_rownames)
A Tiny CPLEX example: ..Python code

c.solve()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()

numcols = c.variables.get_num()
numrows = c.linear_constraints.get_num()

slack = c.solution.get_linear_slacks()
x = c.solution.get_values()

for j in range(numrows):
    print "Row", j, "Slack", slack[j]
for j in range(numcols):
    print "Column", j, "Value", x[j]
A Tiny CPLEX example: Output

Python 2.7.8 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tinymip
CPXPARAM_Read_DataCheck 1
Found incumbent of value 46.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 2 times.
Aggregator did 1 substitutions.
Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.01 ticks)
Tried aggregator 1 time.
Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.00 ticks)
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 8 threads.
Root relaxation solution time = 0.00 sec. (0.00 ticks)
A Tiny CPLEX example: Output

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>Left</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
</tr>
<tr>
<td>*</td>
<td>0+</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Elapsed time = 0.02 sec. (0.03 ticks, tree = 0.01 MB, solutions = 2)

Root node processing (before b&c):

- Real time = 0.02 sec. (0.03 ticks)
- Parallel b&c, 8 threads:
  - Real time = 0.00 sec. (0.00 ticks)
  - Sync time (average) = 0.00 sec.
  - Wait time (average) = 0.00 sec.

Total (root+branch&cut) = 0.02 sec. (0.03 ticks)

Solution status = 101 : MIP_optimal

Solution value = 122.5
Column 0 Value 40.0
Column 1 Value 10.5
Column 2 Value 19.5
Column 3 Value 3.0
A Tiny ODH example: Python code

```python
import sys
import heuristic
import cplex

# data common to all populateby functions
my_obj = [1.0, 2.0, 3.0, 1.0]
my_ub = [40.0, cplex.infinity, cplex.infinity, 3.0]
my_lb = [0.0, 0.0, 0.0, 2.0]
my_ctype = "CCCI"
my_colnames = ["x1", "x2", "x3", "x4"]
my_rhs = [20.0, 30.0, 0.0]
my_rownames = ["r1", "r2", "r3"]
my_sense = "LLE"

c = cplex.Cplex()
h = heuristic.Heuristic(c)
c.objective.set_sense(c.objective.sense.maximize)
c.variables.add(obj=my_obj, lb=my_lb, ub=my_ub, types=my_ctype,
                 names=my_colnames)
rows = [["x1", "x2", "x3", "x4"], [-1.0, 1.0, 1.0, 10.0]],
       [["x1", "x2", "x3"], [1.0, -3.0, 1.0]],
       [["x2", "x4"], [1.0, -3.5]]
c.linear_constraints.add(lin_expr=rows, senses=my_sense,
                         rhs=my_rhs, names=my_rownames)
```
A Tiny ODH example: Python code

```python
h.opt()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()

numcols = c.variables.get_num()
numrows = c.linear_constraints.get_num()

slack = c.solution.get_linear_slacks()
x = c.solution.get_values()

for j in range(numrows):
    print "Row", j, "Slack", slack[j]
for j in range(numcols):
    print "Column", j, "Value", x[j]
```
A Tiny ODH example: Output

Python 2.7.8 (default, Jun 30 2014, 16:08:48) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tinyodh

ODH Large Scale MIP Heurisitic Version 3.3.7 Apr 11 2018 17:16:28
Copyright Optimization Direct 2018
All rights reserved.

CPLEX version 12.8.0.0
Licensed Materials - Property of IBM
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corporation 1988-2017. All Rights Reserved.
Run on Wed Apr 11 17:49:23 2018

System has 8 logical processors and 4 physical processors on a single NUMA node
Level 2 instruction cache size 32768 bytes
Level 2 data cache size 32768 bytes
Level 3 unified cache size 262144 bytes
Level 4 unified cache size 6291456 bytes
Available memory 30106947584 bytes
ODH: Processing problem:
ODH: Defaulted SUB_CPX_NODELIM to 2048.
ODH: Defaulted SUB_CPX_STARTALG to 1.
ODH: Defaulted PRESOLVE to 1.
ODH: Defaulted PHASE1_CPX_NODELIM to 16384.
ODH: Defaulted PHASE1_CPX_MIPEMPHASIS to 1.
ODH: Using dynamic search.
ODH: No Heuristic parameters set.
ODH:
CPX: CPXPARAM_Read_DataCheck
CPXPARAM_Read_DataCheck
1
CPX: CPXPARAM_Threads
CPXPARAM_Threads
6
ODH: Original problem size: 3 rows, 4 columns, 9 nonzeros, 1 integers.
ODH: CPXPARAM_Threads
CPXPARAM_Threads
2
ODH: Found incumbent of value 46.000000 after 0.00 sec. (0.00 ticks)
CPX: Found incumbent of value 46.000000 after 0.00 sec. (0.00 ticks)
ODH: Tried aggregator 2 times.
CPX: Tried aggregator 2 times.
ODH: Aggregator did 1 substitutions.
CPX: Aggregator did 1 substitutions.
ODH: Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
CPX: Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
CPX: Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
ODH: Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
CPX: Presolve time = 0.00 sec. (0.01 ticks)
ODH: Presolve time = 0.00 sec. (0.01 ticks)
ODH: Tried aggregator 1 time.
CPX: Tried aggregator 1 time.
ODH: Tried aggregator 1 time.
CPX: Tried aggregator 1 time.
ODH: Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
CPX: Reduced MIP has 2 rows, 3 columns, and 6 nonzeros.
ODH: Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
CPX: Reduced MIP has 0 binaries, 1 generals, 0 SOSs, and 0 indicators.
ODH: Presolve time = 0.00 sec. (0.00 ticks)
CPX: Presolve time = 0.00 sec. (0.00 ticks)
ODH: MIP emphasis: balance optimality and feasibility.
ODH: Parallel mode: deterministic, using up to 2 threads.
ODH: Root relaxation solution time = 0.00 sec. (0.00 ticks)
CPX: MIP emphasis: balance optimality and feasibility.
ODH:

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Cuts/</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODH:</td>
<td>Node Left Objective IInf Best Integer Best Bound ItCnt Gap</td>
</tr>
<tr>
<td>CPX:</td>
<td>Parallel mode: deterministic, using up to 6 threads.</td>
</tr>
<tr>
<td>ODH:</td>
<td>Node Left Objective IInf Best Integer Best Bound ItCnt Gap</td>
</tr>
</tbody>
</table>

ODH: Root node processing (before b&c):
CPX: Root node processing (before b&c):

ODH:  *
CPX:  *

ODH: Real time = 0.00 sec. (0.02 ticks)
CPX: Real time = 0.00 sec. (0.02 ticks)
ODH: Parallel b&c, 2 threads:

ODH: Real time = 0.00 sec. (0.00 ticks)
ODH: Sync time (average) = 0.00 sec.
ODH: Wait time (average) = 0.00 sec.

CPX: * 0+ 0 46.0000 163.0000 254.35%

ODH: Total (root+branch&cut) = 0.00 sec. (0.02 ticks)

CPX: 0 0 125.2083 1 46.0000 125.2083 3 172.19%

ODH: Presolved problem size: 2 rows, 3 columns, 6 nonzeros, 1 integers.

CPX: * 0+ 0 122.5000 125.2083 2.21%

ODH: No index key specified.
ODH: Using automatic decomposition.
ODH: There are 3 keys (0 keys were dropped) with 3 values.
ODH: Decomposition score 100.00%, graph score 6/6.
ODH: Time taken 0.06 sec. (0.00 ticks)

ODH: Solution improvement heuristic

CPX: 0 0 cutoff 122.5000 3 ---

ODH: Started 2 threads in deterministic mode.

ODH: Nodes

ODH: Node/ Left/ Objective/ Best Integer/ Cuts/
Thread Divsr Sum Infeas IInf Inf Solution Best Bound ItCnt Gap
Time WorkRate

CPX: Elapsed time = 0.11 sec. (0.03 ticks, tree = 0.01 MB, solutions = 2)

CPX:
ODH:
CPX: Root node processing (before b&c):
ODH:       0     4        0.0000     0      122.5000      122.5000       14    0.00%
CPX:       0.12     2.30
CPX:   Real time             =    0.11 sec. (0.03 ticks)
ODH: Solution improvement heuristic terminated with a solution of 122.500000 in 0.13 sec. (0.05 ticks)
ODH: Sync time: CPLEX 0.00% ODH (average) 1.04%, Wait time: CPLEX 0.00% ODH (average) 0.00%
CPX: Parallel b&c, 6 threads:
CPX:   Real time             =    0.00 sec. (0.00 ticks)
CPX:   Sync time (average)   =    0.00 sec.
CPX:   Wait time (average)   =    0.00 sec.
CPX:                           ------------
CPX: Total (root+branch&cut) =    0.11 sec. (0.03 ticks)
ODH: CPLEX terminated with Iterations 3 Nodes 0 Best bound 122.500000 Deterministic time 0.03 ticks.
ODH: Large Scale Heuristic terminated with solution from ODH of 122.500000 with optimality gap of 0.00% in 0.15 sec.
ODH: Written solution to file .sol.
Solution status = 101 : MIP_optimal
Solution value =  122.5
Row 0 Slack 0.0
Row 1 Slack 2.0
Row 2 Slack 0.0
Column 0 Value 40.0
Column 1 Value 10.5
Column 2 Value 19.5
Column 3 Value 3.0
>>> quit()
A Tiny Heuristic example: ..Python code

```python
h.solve()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()

numcols = c.variables.get_num()
numrows = c.linear_constraints.get_num()

slack = c.solution.get_linear_slacks()
x = c.solution.get_values()

for j in range(numrows):
    print "Row", j, "Slack", slack[j]
for j in range(numcols):
    print "Column", j, "Value", x[j]
```
CPLEX example: Python code

"""Demonstrates reading MIP model from file, solving it, and displaying solution."""

import sys
import cplex
c = cplex.Cplex()
c.read("air04.mps.gz")
c.solve()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()
umcols = c.variables.get_num()
umrows = c.linear_constraints.get_num()
slack = c.solution.get_linear_slacks()
x = c.solution.get_values()

for j in range(min(numcols,10)):
    if( x[j] > 0.0001 ):
        print "Column", j, "Value", x[j]
import sys
import heuristic
import cplex
c = cplex.Cplex()
h = heuristic.Heuristic(c)
c.read("air04.mps.gz")
h.opt()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()
numcols = c.variables.get_num()
numrows = c.linear_constraints.get_num()
slack = c.solution.get_linear_slacks()
x = c.solution.get_values()

for j in range(min(numcols,10)):
    if( x[j] > 0.0001 ):
        print "Column", j, "Value", x[j]
ODH + CPLEX example

- Idea is to run ODH
  - Get a good solution and small-ish gap
  - Resources used by both CPLEX and ODH heuristic engine
    - Processors
    - Bus bandwidth
- Then run CPLEX on its own
  - Releasing resources used by ODH heuristic engine
import sys
import heuristic
import cplex
c = cplex.Cplex()
h = heuristic.Heuristic(c)
c.read("example.mps")

h.setdblparam("timelimit",25)
h.opt()
c.parameters.mip.tolerances.mipgap.set(3.4e-2)
c.solve()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value  = ", c.solution.get_objective_value()
numcols = c.variables.get_num()
numrows = c.linear_constraints.get_num()
slack = c.solution.get_linear_slacks()
x = c.solution.get_values()
for j in range(min(numcols,10)):
    if( x[j] > 0.0001 ):
        print "Column", j, "Value", x[j]
Heuristic + CPLEX example

- Idea is to run ODH heuristic engine only
  - Get a good solution
  - Resources used only by ODH heuristic engine
    - Processors
    - Bus bandwidth
- Then run CPLEX on its own
  - Releasing resources used by ODH heuristic engine
  - Getting solution of proven quality
Heuristic + CPLEX example: Python code

```python
import sys
import heuristic
import cplex

c = cplex.Cplex()
h = heuristic.Heuristic(c)
c.read("example.mps")

h.setdblparam("objtarget",70000)
h.solve()

c.parameters.timelimit.set(50.0)
c.solve()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()
numcols = c.variables.get_num()
numrows = c.linear_constraints.get_num()
slack = c.solution.get_linear_slacks()
x = c.solution.get_values()
for j in range(min(numcols,10)):
    if( x[j] > 0.0001 ):
        print "Column", j, "Value", x[j]
```

OPTIMIZATION DIRECT

Business Partner IBM
ODH Python call-backs

Call-backs provided for

- Inspecting solutions when they are found
  - Can stop the search if it satisfies some criterion (is good enough, has required property, etc.)
- Handling ‘screen’ output
- Providing user specified decomposition
import sys
import heuristic
import cplex

def mysolutioncallback( heur, cpx, objvalue ):
    print("In mysolutioncallback: objvalue = " + str(objvalue))
    x = heur.solution()
    print("there are " + str(len(x)) + " variable values and the 1st 5 are:")
    for i in range(5):
        print(cpx.variables.get_names(i) + " = " + str(x[i]))
    if objvalue < 54500.0:
        print("Solution is good enough. Stopping")
        return 1
    return 0

c = cplex.Cplex()
h = heuristic.Heuristic(c)
c.read("example.mps")
h.setsolutioncallback( mysolutioncallback )
h.opt()

print "Solution status = ", c.solution.get_status(), ":",
print c.solution.status[c.solution.get_status()]
print "Solution value = ", c.solution.get_objective_value()
Conclusions

- Python increasingly popular platform
- Easy to use both CPLEX and ODH in Python
- Build models and analyze results in Python
- Easy to deploy call-backs
- Access to data science tools makes Python a powerful model development environment
Benchmarking and Evaluation

• If you think that ODHeuristics and/or ODH-CPLEX might work for you:
  • send us your difficult matrices and we will send you the results
  • request an evaluation copy
Thanks for listening

Robert Ashford
rwa@optimizationdirect.com
www.optimizationdirect.com