An ODH|CPLEX Python Primer

Robert Ashford
Alkis Vazacopoulos

Optimization Direct Inc.

October 2022

OPTIMIZATION
DIRECT

Business Partner

Summary

- Build a basic model using the docplex API in Python
  - create variables and constraints
  - solve the model
  - read data from a CSV or spreadsheet
  - use call-backs
  - return the results
- Can use CPLEX or ODH|CPLEX

Modelling Environments

Need to set environment variables etc. e.g. under Windows:

```
PYTHON_DIR=c:\Program Files\Python37
PYTHONHASHSEED=234567
PATH=%PYTHON_DIR%;PYTHON_DIR%\Scripts;%PATH%
```

Python

Installation

Install the docplex, CPLEX and ODH|CPLEX python APIs in

%PYTHON_DIR%\Lib\site-packages\docplex

%PYTHON_DIR%\Lib\site-packages\cplex

%PYTHON_DIR%\Lib\site-packages\heuristic

A few words

Run program files by e.g. python myprog.py at the O/S command prompt or can interpret instructions at its own command prompt e.g.
>>>print('Hello world')

All commands are functions. Functions always have zero or more arguments in parentheses '()'. E.g. >>>quit()

Object types are (usually) inferred. E.g. >>>myarr = ['one','two','three'] sets up a *list* (array) of strings.

WARNING – and key to understanding python:

x += 1 does NOT increment the value of x!

Many excellent guides e.g. https://www.w3schools.com/python/

Sudoku Example

- See how to build and solve a simple pure integer program (PIP)
- Use both CPLEX and ODH|CPLEX
- "World's hardest Sudoku"
  Arto Inkala (Finish mathematician, 2012)
- Chose example to illustrate use of docplex and doheuristic

OPTIMIZATION
DIRECT

Business Partner

Problem

Find digits in range 1..9 so each row, column and box contain different digits

8 . .	. . .	. . .
. . 3	6 . .	. . .
. 7 .	. 9 .	2 . .
. 5 .	. . 7	. . .
. . .	. 4 5	7 . .
. . .	1 . .	. 3 .
. . 1	. . .	. 6 8
. . 8	5 . .	. 1 .
. 9 .	. . .	4 . .

OPTIMIZATION DIRECT

Business Partner IBM

Formulation

Label digits by their row i and column j

Let D_{ij} be their value when > 0

and need to find $cell_{ij}$ when $D_{ij} = 0$

	1	2	3	4	5	6	7	8	9
1	8	.	.	.	.	.	.	.	.
2	.	.	3	6	.	.	.	.	.
3	.	7	.	.	9	.	2	.	.
4	.	5	.	.	.	7	.	.	.
5	.	.	.	.	4	5	7	.	.
6	.	.	.	1	.	.	.	3	.
7	.	.	1	.	.	.	.	6	8
8	.	.	8	5	.	.	.	1	.
9	.	9	.	.	.	.	4	.	.

Formulation

D_{ij} are the input data

Introduce 3rd dimension k =1..9 and binary variables x_{ijk}

Think of x_{ijk} being a column of binaries at i,j so $x_{ijk} = 1$ and $x_{ij\ell} = 0$ for $\ell \neq k$ **means** $cell_{ij} = k$

Exactly one x in each col is one: $\Sigma_k\, x_{ijk} = 1,\ \forall\ i, j$ and $x_{ijD_{ij}} = 1,\ \forall\ i, j$ where $D_{ij} > 0$

Can express the rules of Sudoku:

Formulation

The numbers in each row of our table must be different: $\Sigma_j x_{ijk} = 1, \forall i, k$

The numbers in each col of our table must be different: $\Sigma_i x_{ijk} = 1, \forall j, k$

The numbers in each 3 x 3 box must be different: let $\mathcal{I}_1 = \{1,2,3\}$, $\mathcal{I}_2 = \{4,5,6\}$, $\mathcal{I}_3 = \{7,8,9\}$

$\Sigma_{\ell \in \mathcal{I}_i\, m \in \mathcal{I}_j} x_{\ell m k} = 1, \forall i \in \{1,2,3\}, j \in \{1,2,3\}, k \in \{1,..,9\}$

Simple! Just want a feasible solution

Python – Sudoku.py

```
# problem data
D = [
  [8, 0, 0,  0, 0, 0,  0, 0, 0],
  [0, 0, 3,  6, 0, 0,  0, 0, 0],
  [0, 7, 0,  0, 9, 0,  2, 0, 0],

  [0, 5, 0,  0, 0, 7,  0, 0, 0],
  [0, 0, 0,  0, 4, 5,  7, 0, 0],
  [0, 0, 0,  1, 0, 0,  0, 3, 0],

  [0, 0, 1,  0, 0, 0,  0, 6, 8],
  [0, 0, 8,  5, 0, 0,  0, 1, 0],
  [0, 9, 0,  0, 0, 0,  4, 0, 0]
]
```

OPTIMIZATION DIRECT

Business Partner IBM

Python – Sudoku.py

```
# set up model in docplex
import sys
from docplex.mp.model import Model
import heuristic

# get a model instance
m = Model("Sudoku")

R = range(1,10) # sequence 1..9

# define variables
x = m.binary_var_dict([(i, j, k) for i in R for j in R for k in R],
                      name="x")
```

Python – Sudoku.py

```
# define the constraints
```

Exactly one x in each col is one: $\Sigma_k x_{ijk} = 1, \forall i, j$

```
# exactly one binary in each column is one:
for i in R:
  for j in R:
    m.add_constraint(
     m.sum(x[i,j,k] for k in R) == 1,'O_%d_%d'%(i,j))
                                                  "

#aliter (more efficient):
m.add_constraints(
 (m.sum(x[i,j,k] for k in R)==1,'O_%d_%d'%(i,j)) for i in R for j in R)
```

and $x_{ijD_{ij}} = 1, \forall i, j$ where $D_{ij} > 0$

```
# fix x's for problem data
m.add_constraints(
 x[i,j,D[i-1][j-1]] == 1 for i in R for j in R if D[i-1][j-1] > 0 )
```

OPTIMIZATION DIRECT

Business Partner

Python – Sudoku.py

The numbers in each row of our table must be different: $\Sigma_j x_{ijk} = 1, \forall i, k$

```
# the nos in each row and col must differ
m.add_constraints(
 m.sum(x[i,j,k] for j in R) == 1 for i in R for k in R )
```

The numbers in each col of our table must be different: $\Sigma_i x_{ijk} = 1, \forall j, k$

```
m.add_constraints(
 m.sum(x[i,j,k] for i in R) == 1 for j in R for k in R )
```

The numbers in each 3 x 3 box must be different:
let $\mathcal{I}_1 = \{1,2,3\}$, $\mathcal{I}_2 = \{4,5,6\}$, $\mathcal{I}_3 = \{7,8,9\}$

```
# the numbers in each 3x3 box must differ
T = [ [1,2,3],[4,5,6],[7,8,9] ]
S = range(3)
```

$\Sigma_{\ell \in \mathcal{I}_i, n \in \mathcal{I}_j} x_{\ell nk} = 1, \forall i \in \{1,2,3\}, j \in \{1,2,3\}, k \in \{1,..,9\}$

```
m.add_constraints(
 m.sum(x[l,n,k] for l in T[i] for n in T[j])==1
  for i in S for j in S for k in R )
```

Python – Sudoku.py

```
# get ODH instance
h = heuristic.doheuristic(m)

# solve
h.setintparam("writesolution",0)
solution = h.opt() # solution = m.solve() for CPLEX

# print solution information
if solution is not None:
  m.print_information()
  m.print_solution()
```

Python – Sudoku.py

```
# display the problem

 print( '' )
 print( ' The problem' )
 print( ' ==================================' )
 l = 0
 for i in R:
   for j in R:
     if l%3 == 0:
       print( ' | ', end='' )
     else:
       print( '   ', end='' )
     if D[i-1][j-1] > 0.1:
       print( D[i-1][j-1], end='' )
     else:
       print( '.', end='' )
     l += 1
   print( ' |' )
   if i%3 == 0:
     print( ' ==================================' )
   else:
     print( ' |           |           |           |' )
```

OPTIMIZATION DIRECT

Business Partner IBM

Python – Sudoku.py

```
# tabulate the results

 print( '' )
 print( ' The solution' )
 print( ' ===================================' )
 l = 0
 for i in R:
   for j in R:
     for k in R:
       if x[i,j,k].solution_value > 0.1:
         if l%3 == 0:
           print( ' | ', end='' )
         else:
           print( '   ', end='' )
         print( k, end='' )
         l += 1
   print( ' |' )
   if i%3 == 0:
     print( ' ===================================' )
   else:
     print( ' |           |           |           |' )
```

OPTIMIZATION DIRECT

Business Partner IBM

Python – Sudoku.py

```
ODH Large Scale MIP Heuristic Version 6.0.5 Aug  9 2021 16:18:08
Copyright Optimization Direct 2021
All rights reserved.
*** Expires on Fri Oct 22 01:00:00 2021
*** Developer license for 1024 threads.

CPLEX version 20.1.0.0
Licensed Materials - Property of IBM
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21
Copyright IBM Corporation 1988-2020. All Rights Reserved.
Run on Mon Sep 13 08:48:51 2021

Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz [F:6-0|M:c-3|S:3|X:bfebfbff-7fdafbbf]
There are 8 logical processors and 4 physical processors on a single NUMA node
Level 1 instruction cache size 32768 bytes
Level 1 data cache size 32768 bytes
Level 2 unified cache size 262144 bytes
Level 3 unified cache size 6291456 bytes
Available memory 27903848448 bytes

ODH:
ODH: Processing problem:
ODH:
ODH: Defaulted SUB_CPX_NODELIM to 2048.
ODH: Defaulted PRESOLVE to 1.
ODH: Defaulted PHASE1_CPX_NODELIM to 16384.
ODH: Defaulted PHASE1_CPX_MIPEMPHASIS to 1.
ODH: Using dynamic search for sub-solves.
```

OPTIMIZATION DIRECT

Business Partner

Python – Sudoku.py

```
ODH: Heuristic parameters set:
ODH:  WRITESOLUTION = 0
ODH:
ODH: Intel(R) Core(TM) i7-4710MQ CPU @ 2.50GHz [F:6-0|M:c-3|S:3|X:bfebfbff-7fdafbbf]
ODH: Version: 6.0.5 Aug  9 2021 16:18:08
CPX: Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
CPX: CPXPARAM_Read_DataCheck                            1
CPX: CPXPARAM_Threads                                   6
CPX: CPXPARAM_MIP_Strategy_CallbackReducedLP            0
CPX: CPXPARAM_Read_APIEncoding                          "UTF-8"
CPX: Legacy callback                                    p
CPX: Generic callback                                   0x5c
CPX: Tried aggregator 2 times.
CPX: MIP Presolve eliminated 112 rows and 475 columns.
CPX: Aggregator did 9 substitutions.
CPX: Reduced MIP has 224 rows, 245 columns, and 981 nonzeros.
CPX: Reduced MIP has 245 binaries, 0 generals, 0 SOSs, and 0 indicators.
CPX: Presolve time = 0.00 sec. (1.00 ticks)
CPX: Found incumbent of value 0.000000 after 0.01 sec. (4.60 ticks)
CPX:
CPX: Root node processing (before b&c):
CPX:   Real time             =    0.01 sec. (4.63 ticks)
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) =    0.01 sec. (4.63 ticks)
ODH: CPLEX terminated with Iterations 0 Nodes 0 Best bound 0.000000 Deterministic time 4.79 ticks.
ODH: Large Scale Heuristic terminated with solution from CPLEX of 0.000000 with optimality gap of 0.00% in 0.02
sec.
```

OPTIMIZATION DIRECT

Business Partner IBM

Python – Sudoku.py

```
Model: Sudoku
 - number of variables: 729
   - binary=729, integer=0, continuous=0
 - number of constraints: 345
   - linear=345
 - parameters: defaults
 - objective: none
 - problem type is: MILP
x_1_1_8=1
x_1_2_1=1
x_1_3_2=1
x_1_4_7=1
x_1_5_5=1
x_1_6_3=1
:
:
```

Python – Sudoku.py

```
The problem
=====================================
| 8   .   .  | .   .   .  | .   .   . |
|            |            |           |
| .   .   3  | 6   .   .  | .   .   . |
|            |            |           |
| .   7   .  | .   9   .  | 2   .   . |
=====================================
| .   5   .  | .   .   7  | .   .   . |
|            |            |           |
| .   .   .  | .   4   5  | 7   .   . |
|            |            |           |
| .   .   .  | 1   .   .  | .   3   . |
=====================================
| .   .   1  | .   .   .  | .   6   8 |
|            |            |           |
| .   .   8  | 5   .   .  | .   1   . |
|            |            |           |
| .   9   .  | .   .   .  | 4   .   . |
=====================================
```

OPTIMIZATION DIRECT

Business Partner IBM

Python – Sudoku.py

```
The solution
=====================================
| 8    1    2 | 7    5    3 | 6    4    9 |
|              |              |              |
| 9    4    3 | 6    8    2 | 1    7    5 |
|              |              |              |
| 6    7    5 | 4    9    1 | 2    8    3 |
=====================================
| 1    5    4 | 2    3    7 | 8    9    6 |
|              |              |              |
| 3    6    9 | 8    4    5 | 7    2    1 |
|              |              |              |
| 2    8    7 | 1    6    9 | 5    3    4 |
=====================================
| 5    2    1 | 9    7    4 | 3    6    8 |
|              |              |              |
| 4    3    8 | 5    2    6 | 9    1    7 |
|              |              |              |
| 7    9    6 | 3    1    8 | 4    5    2 |
=====================================
```

OPTIMIZATION DIRECT

Business Partner IBM

Python – Reading data from CSV file

Could replace in-line data:

```
D = [
  [8, 0, 0,  0, 0, 0,  0, 0, 0],
  [0, 0, 3,  6, 0, 0,  0, 0, 0],
  [0, 7, 0,  0, 9, 0,  2, 0, 0],

  [0, 5, 0,  0, 0, 7,  0, 0, 0],
  [0, 0, 0,  0, 4, 5,  7, 0, 0],
  [0, 0, 0,  1, 0, 0,  0, 3, 0],

  [0, 0, 1,  0, 0, 0,  0, 6, 8],
  [0, 0, 8,  5, 0, 0,  0, 1, 0],
  [0, 9, 0,  0, 0, 0,  4, 0, 0]
]
```

OPTIMIZATION DIRECT

Business Partner

Python – Reading data from CSV file

with:

```
from pandas import *
L = pandas.read_csv('problem.csv',header=None)
D = L.values
```

Where problem.csv is:

```
8, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 3, 6, 0, 0, 0, 0, 0
0, 7, 0, 0, 9, 0, 2, 0, 0
0, 5, 0, 0, 0, 7, 0, 0, 0
0, 0, 0, 0, 4, 5, 7, 0, 0
0, 0, 0, 1, 0, 0, 0, 3, 0
0, 0, 1, 0, 0, 0, 0, 6, 8
0, 0, 8, 5, 0, 0, 0, 1, 0
0, 9, 0, 0, 0, 0, 4, 0, 0
```

OPTIMIZATION DIRECT

Business Partner

Python – Reading data from CSV file

If we're being picky, D is a pandas data frame
To put the data in an 'ordinary' array we need:

```
from pandas import *
L = pandas.read_csv('problem.csv',header=None)
D = []
for i in range(len(L)):
  D.append(i)
  D[i] = []
  for j in range(len(L.values[i])):
    D[i].append(L.values[i][j])
```

Python – Writing data to CSV file

Easy to write CSV files in python. To write the results:

```
f = open('Results.csv','w')
 for i in R:
   for j in R:
     for k in R:
       if x[i,j,k].solution_value > 0.1:
         f.write(str(k))
         f.write(',')
   f.write('\n')
 f.close()
```

Call-backs

Solver can call into python model when solving

- Monitor progress
- Control solution process
- Do so using *model* objects (not index values)

CPLEX call-backs

- MIP progress, incumbent, add cuts, etc.

ODH call-backs

- Solution, decomposition

Call-back Example

ODH index key call-back

Use it to specify our own model decomposition

```
h = heuristic.doheuristic(m)
h.setindexkeycallback( myindexkeycallback )
# solve
solution = h.opt()
```

Make our Sudoku example more interesting

reduce number of cells initially specified and minimize the sum of cells on the diagonal:

```
m.minimize(m.sum(k*x[i,i,k] for i in R for k in R))
```

Call-back Example

```
def myindexkeycb( heur, cpx, ncol, numvar, varind, keyvals ):
  # use the _index attribute of each variable to get the
  # CPLEX variable index value to identify the variable to
  # the ODH heuristic engine through argument varind[]
  v = n = 0
  for s in S:
    for t in S:
      v += 1
      for i in T[s]:
        for j in T[t]:
          for k in R:
            varind[n] = x[i,j,k]._index
            keyvals[n] = v
            n += 1
  numvar[0] = n;
  print('***  Set',v,'keys on',n,'variables')
  return 0
```

`h.setindexkeycallback( myindexkeycb )`

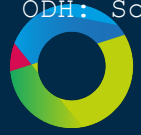
Call-back Example

```
def myindexkeycb( heur, cpx, ncol, numvar, varind, keyvals ):
  # use the _index attribute of each variable to get the
  # CPLEX variable index value to identify the variable to
  # the ODH heuristic engine through argument varind[]
  v = n = 0
  for s in S:
    for t in S:
      v += 1
      for i in T[s]:
        for j in T[t]:
          for k in R:
            varind[n] = x[i,j,k]._index
            keyvals[n] = v
            n += 1
  numvar[0] = n;
  print('***  Set',v,'keys on',n,'variables')
  return 0
```

h.setindexkeycallback(myindexkeycb)

Call-back Example

```
ODH: No index key specified.
ODH: Using automatic decomposition.
ODH: There are 502 keys (0 keys were dropped) with 502 values.
ODH: Decomposition score 100.00%, graph score 10900/10900.
ODH: Time taken 0.00 sec. (0.21 ticks)
ODH:
ODH: Solution improvement heuristic
ODH: Started 2 threads in deterministic mode.
ODH:          Nodes
ODH:    Node/ Left/    Objective/       Best Integer/       Cuts/
ODH:   Thread Divsr    Sum Infeas  IInf  Inf Solution    Best Bound    ItCnt     Gap       Time WorkRate
CPX: Root relaxation solution time = 0.02 sec. (8.69 ticks)
CPX:
CPX:          Nodes                                          Cuts/
CPX:     Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap
CPX: *     0+    0                           52.0000        1.0000           98.08%
CPX: *     0+    0                           37.0000        1.0000           97.30%
CPX:       0     0       18.0000   203       37.0000       18.0000      482   51.35%
CPX: *     0+    0                           27.0000       18.0000           33.33%
ODH:       0    13        0.0000     0       30.0000        1.0000       32   96.67%      0.07   207.91
ODH:       0    14        0.0000     0       28.0000        1.0000       39   96.43%      0.08   213.69
ODH:       0    14        0.0000     0       25.0000        1.0000       45   96.00%      0.13   160.73
ODH:       0    17        0.0000     0       24.0000        1.0000       49   95.83%      0.15   169.54
ODH:       0    17        0.0000     0       20.0000       18.0000       51   10.00%      0.17   238.34
CPX: *     0+    0                           18.0000       18.0000            0.00%
CPX:       0     0        cutoff             18.0000       18.0000      482   0.00%
CPX: Elapsed time = 0.17 sec. (54.73 ticks, tree = 0.01 MB, solutions = 4)
CPX:
:
ODH:       0    17        0.0000     0       18.0000       18.0000       53   0.00%      0.22   345.51
ODH: Solution improvement heuristic terminated with a solution of 18.000000 in 0.22 sec. (76.40 ticks))
```

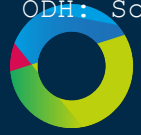


OPTIMIZATION DIRECT

Business Partner IBM

Call-back Example

```
ODH: No index key specified.
ODH: Using user call-back for matrix decomposition.
***  Set 9 keys on 729 variables
ODH: There are 9 keys (0 keys were dropped) with 502 values.
ODH: Decomposition score 18.42%, graph score 36/10900.
ODH: Time taken 0.00 sec. (0.06 ticks)
ODH:
ODH: Solution improvement heuristic
ODH: Started 2 threads in deterministic mode.
ODH:         Nodes
ODH:   Node/ Left/    Objective/       Best Integer/       Cuts/
ODH:  Thread Divsr    Sum Infeas  IInf  Inf Solution    Best Bound    ItCnt     Gap       Time WorkRate
CPX: Root relaxation solution time = 0.00 sec. (8.69 ticks)
CPX:
CPX:         Nodes                                          Cuts/
CPX:    Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap
CPX: *     0+    0                           52.0000        1.0000           98.08%
CPX: *     0+    0                           37.0000        1.0000           97.30%
CPX:       0     0       18.0000   203       37.0000       18.0000      482   51.35%
CPX: *     0+    0                           27.0000       18.0000           33.33%
ODH:       0     5        0.0000     0       44.0000                      16                0.12     83.16
ODH:       0     6        0.0000     0       34.0000        0.0000        19  100.00%       0.12     87.12
ODH:       0     6        0.0000     0       33.0000        0.0000        22  100.00%       0.13     92.17
ODH:       0     6        0.0000     0       30.0000        1.0000        25   96.67%       0.15    107.33
ODH:       0     6        0.0000     0       28.0000        1.0000        28   96.43%       0.16    114.23
ODH:       0     6        0.0000     0       25.0000        1.0000        28   96.00%       0.17    116.36
ODH:       0     6        0.0000     0       21.0000        1.0000        30   95.24%       0.18    115.70
ODH:       0     7        0.0000     0       20.0000        1.0000        36   95.00%       0.21    123.13
ODH:       1     8        0.0000     0       19.0000       18.0000        38    5.26%       0.23     92.28
ODH:       1     8        0.0000     0       18.0000       18.0000        39    0.00%       0.23     93.60
ODH: Solution improvement heuristic terminated with a solution of 18.000000 in 0.20 sec. (32.08 ticks)
```



OPTIMIZATION DIRECT

Business Partner IBM

Redistricting Models

- Discussed in earlier talk
- Used for redistricting States and Cities
- ~900 lines of python (~10 x size of Sudoku)
- docplex/doheuristic to build and solve
- Use pandas for data input
- Use ODH callbacks for
  - Decomposition
  - Monitoring solutions and their KPIs

OPTIMIZATION
DIRECT

Business Partner

Thanks for listening

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

OPTIMIZATION DIRECT

Business Partner