yaposib Documentation

Release 0.3.2

Christophe-Marie Duquesne

February 16, 2014

Yaposib is a python binding to OSI, the Open Solver Interface from COIN-OR. It intends to give access to various solvers through python. Yaposib was created to be integrated with pulp-or, and plays nicely with it.

CHAPTER 1

Manual

1.1 Getting Started

What follows is a guide for installing yaposib very quickly and solve your first linear program using it.

1.1.1 Installing

Recommended method: pip

1. Install pip, python-dev and boost::python. On ubuntu:

```
sudo apt-get install python-pip python-dev libboost-python-dev
```

2. Install osi. If you want support for commercial solvers, relying on your distribution is not recommended. Otherwise, using a package from your distribution is fine. Note that if you modify the Osi installed on your machine, you will have to recompile yaposib.
3. Use pip to install yaposib:

```
sudo pip install yaposib
```

Alternative: development version

1. Follow 1. and 2. from the previous method
2. Clone the repository

```
git clone https://code.google.com/p/yaposib/
```

3. Run setup.py

```
cd yaposib
sudo python setup.py install
```

1.1.2 Checking your installation

The utility *yaposib-config* is a helper script that helps you determine if your installation went fine. Run it without any argument.

```
yaposib-config
```

This tool runs the yaposib test suite on every solvers that you Osi build reportedly supports. Since not all solvers behave equivalently, some tests might fail with some solvers, and succeed with others. A failure does not necessarily means that yaposib is completely unusable with your solver, it might simply mean that it was not tested yet combined with your solver. Please report any failures on the bugtracker.

1.1.3 A quick code snippet

Let's dive into the code. Here is an example program that illustrates some features of yaposib:

```
"""
builds the following problem

0  <= x <= 4
-1 <= y <= 1
0  <= z
0  <= w

minimize obj = x + 4*y + 9*z
such that:
c1: x+y <= 5
c2: x+z >= 10
c3: -y+z == 7
c4: w >= 0
"""
import yaposib

prob = yaposib.Problem("Clp")

obj = prob.obj
obj.name = "MyProblem"
obj.maximize = False

# names
cols = prob.cols
for var_name in ["x", "y", "z", "w"]:
    col = cols.add(yaposib.vec([]))
    col.name = var_name
# lowerbounds
for col in cols:
    col.lowerbound = 0
cols[1].lowerbound = -1
# upperbounds
cols[0].upperbound = 4
cols[1].upperbound = 1
# constraints
rows = prob.rows
rows.add(yaposib.vec([(0,1),(1,1)]))
rows.add(yaposib.vec([(0,1),(2,1)]))
rows.add(yaposib.vec([(1,-1),(2,1)]))
rows.add(yaposib.vec([(3,1)]))
# constraints bounds
rows[0].upperbound = 5
rows[1].lowerbound = 10
rows[2].lowerbound = 7
rows[2].upperbound = 7
```

```
rows[3].lowerbound = 0
# constraints names
for row, name in zip(rows, ["c1", "c2", "c3", "c4"]):
    row.name = name

# obj
prob.obj[0] = 1
prob.obj[1] = 4
prob.obj[2] = 9

prob.solve()

for col in prob.cols:
    print("%s=%s" % (col.name, col.solution))
```

It is also easy to write a generic command line solver in a few lines of code. The following script is part of the yaposib distribution and is shipped as the command line utility *yaposib-solve*

```
import yaposib
import sys

def main():
    """Extra simple command line mps solver"""

    if len(sys.argv) <= 1:
        print("Usage: yaposib-solve <file1.mps> [<file2.mps> ...]")
        sys.exit(0)

    solver = yaposib.available_solvers()[0]

    for filename in sys.argv[1:]:

        problem = yaposib.Problem(solver)

        print("Will now solve %s" % filename)
        err = problem.readMps(filename)
        if not err:
            problem.solve()
            if problem.status == 'optimal':
                print("Optimal value: %f" % problem.obj.value)
                for var in problem.cols:
                    print("\t%s = %f" % (var.name, var.solution))
            else:
                print("No optimal solution could be found.")

if __name__ == "__main__":
    main()
```

Other examples are available in the examples directory.

1.2 Reference API

1.2.1 Problem

class Problem

Models an LP problem

Main methods

`Problem.markHotStart()`

Makes an internal optimization snapshot of the problem (an internal warmstart object is built)

`Problem.unmarkHotStart()`

Deletes the internal snapshot of the problem (if existing)

`Problem.solve(`*True/False*`)`

Solves the internal problem:

- If an internal snapshot exists, use it.
- If the problem has already been solved, use the internal *ReSolve*.
- If the argument is true, add a branch and bound call.

`Problem.status`

RO Attribute. string describing the solver status: "undefined", "abandoned", "optimal", "infeasible" or "limitreached". You can get more details using the properties:

- isAbandoned
- isProvenOptimal
- isProvenPrimalInfeasible
- isProvenDualInfeasible
- isPrimalObjectiveLimitReached
- isDualObjectiveLimitReached
- isIterationLimitReached

`Problem.writeLp(`*"filename"*`)`

Write the problem in a file (lp format). The argument is appended the extension ".lp"

Objective

`Problem.obj`

Represents the objective of the problem.

`Problem.obj.value`

RO attribute (double). Objective value

`Problem.obj.maximize`

RW attribute (bool) min/max problem

`Problem.obj.name`

RW attribute (string) name

`Problem.obj.__len__`

RO attribute (int) number of columns

`Problem.obj.`**`__iter__`**`()`

Makes iterable

`Problem.obj.`**`__getitem__`**`()`

get the given coef with *Problem.obj[i]*

`Problem.obj.`**`__setitem__`**`()`

set the given coef with *Problem.obj[i] = double*

Rows

`Problem.`**`rows`**

Represents every rows

`Problem.rows.`**`add`**`(`*vec([(1, 2.0), (3, 0.1), ...])*`)`

adds the given row to the problem and returns a Row object

`Problem.rows.`**`__len__`**

the number of rows

`Problem.rows.`**`__iter__`**`()`

Makes iterable

`Problem.rows.`**`__getitem__`**`()`

allows to get the row of index with *Problem.rows[i]*

`Problem.rows.`**`__delitem__`**`()`

deletes the row of given index with del *Problem.rows[i]*

`Problem.rows[i].index`

RO Attribute (int) index in the problem

`Problem.rows[i].name`

RW Attribute (string) name of the row

`Problem.rows[i].lowerbound`

RW Attribute (double) lowerbound of the row

`Problem.rows[i].upperbound`

RW Attribute (double) upperbound of the row

`Problem.rows[i].indices`

RO Attribute (list of int) indices of the columns refered by the row

`Problem.rows[i].values`

RO Attribute (list of double) values of the coefficients for the columns refered by the row

`Problem.rows[i].dual`

RW Attribute (double) dual value of the row

`Problem.rows[i].activity`

RO Attribute (double) activity of the row

Columns

Problem.**cols**

Variables Represent all the columns of the problem

Problem.cols.**add**(*vec([(1, 2.0), (3, 0.1), ...])*)

adds the given column (returns a Col object)

Problem.cols.**__len__**

returns the number of columns

Problem.cols.**__getitem__**()

returns the column at the given index with *Problem.cols[i]*

Problem.cols.**__iter__**()

make iterable

Problem.cols.**__delitem__**()

deletes the column at given index with del *Problem.cols[i]*

Problem.cols[i].index

RO Attribute (int) index in problem

Problem.cols[i].name

RW Attribute (string) name

Problem.cols[i].lowerbound

RW Attribute (double) lowerbound

Problem.cols[i].upperbound

RW Attribute (double) upperbound

Problem.cols[i].indices

RO Attribute (list of int) indices of the row refered by the column

Problem.cols[i].values

RO Attribute (list of double) values of the coefficients for the rows refered by the column

Problem.cols[i].solution

RW Attribute (double) solution

Problem.cols[i].reducedcost

RO Attribute (double) reduced cost

Problem.cols[i].integer

RW Attribute (double) integer variable?

Problem Tuning

Problem.**maxNumIterations**

RW attribute (int) The maximum number of iterations (whatever that means for the given solver) the solver can execute before terminating (When solving/resolving)

`Problem.`**`maxNumIterationsHotStart`**

RW attribute (int) The maximum number of iterations (whatever that means for the given solver) the solver can execute when hot starting before terminating.

`Problem.`**`dualObjectiveLimit`**

RW attribute (double) Dual objective limit. This is to be used as a termination criteria in methods where the dual objective monotonically changes (e.g., dual simplex, the volume algorithm)

`Problem.`**`primalObjectiveLimit`**

RW attribute (double) Primal objective limit. This is to be used as a termination criteria in methods where the primal objective monotonically changes (e.g., primal simplex)

`Problem.`**`dualTolerance`**

RW attribute (double) The maximum amount the dual constraints can be violated and still be considered feasible.

`Problem.`**`primalTolerance`**

RW attribute (double) The maximum amount the primal constraints can be violated and still be considered feasible.

`Problem.`**`objOffset`**

RW attribute (double) The value of any constant term in the objective function.

`Problem.`**`doPreSolveInInitial`**

RW attribute (bool) Whether to do a presolve in initialSolve.

`Problem.`**`doDualInInitial`**

RW attribute (bool) Whether to use a dual algorithm in initialSolve. The reverse is to use a primal algorithm

`Problem.`**`doPreSolveInReSolve`**

RW attribute (bool) Whether to do a presolve in resolve

`Problem.`**`doDualInResolve`**

RW attribute (bool) Whether to use a dual algorithm in resolve. The reverse is to use a primal algorithm

`Problem.`**`doScale`**

RW attribute (bool) Whether to scale problem

`Problem.`**`doCrash`**

RW attribute (bool) Whether to create a non-slack basis (only in initialSolve)

`Problem.`**`doInBranchAndCut`**

RW attribute (bool) Whether we are in branch and cut - so can modify behavior

`Problem.`**`iterationCount`**

RO attribute (int) Get the number of iterations it took to solve the problem (whatever iteration means to the solver).

`Problem.`**`integerTolerance`**

RO attribute (double) Get the integer tolerance of the solver

`Problem.`**`isAbandoned`**

RO attribute (bool) Are there numerical difficulties?

`Problem.`**`isProvenOptimal`**

RO attribute (bool) Is optimality proven?

`Problem.`**`isProvenPrimalInfeasible`**

RO attribute (bool) Is primal infeasiblity proven?

`Problem.`**`isProvenDualInfeasible`**

RO attribute (bool) Is dual infeasiblity proven?

`Problem.`**`isPrimalObjectiveLimitReached`**

RO attribute (bool) Is the given primal objective limit reached?

`Problem.`**`isDualObjectiveLimitReached`**

RO attribute (bool) Is the given dual objective limit reached?

`Problem.`**`isIterationLimitReached`**

RO attribute (bool) Iteration limit reached?

1.2.2 Helper

`vec`([*(0, 0.1)*, *(1, 2.3)*])

Helper function that returns a internal type of sparse vector. See OSI's *CoinPackedVector*. Write only.

1.3 FAQ

Is it possible to modify in place a row/column? No. OSI Does not allow such a thing. You can add and remove rows/columns to a problem, but once it's done, it is impossible to modify them.

I can't add colums/rows Columns that you add must refer to existing rows (and vice-versa). That means you first have to add empty rows if you want to add your column. If you absolutely want to be able to add a column that refers to non existing rows, it should be fairly easy: write a function that counts the maximum row refered by the column you add, and add as many empty rows as needed in your problem. Same goes for the rows.

How efficiently does yaposib accesses to solvers memory? yaposib's design has been driven by memory access efficiency. It is built on the top of the C++ OsiSolverInterface class of COIN-OSI. You can thus manipulate and modify the rows/columns of the same problem as fast as you would be able to do it with OSI using the class OsiSolverInterface.

CHAPTER 2

Various Infos

The repository is hosted on googlecode.

```
git clone https://code.google.com/p/yaposib/
```

A mirror is also maintained on github, and can be useful in a number of situations, like when you only have svn:

```
svn checkout https://github.com/chmduquesne/yaposib
```

The license is EPL.

CHAPTER 3

Indices and tables

- *genindex*
- *search*