FelooPy: An integrated optimization environment for AutoOR in Python

Keivan Tafakkori

October 26, 2022

1 Introduction

FelooPy (FEasible, LOgical and OPtimal + Python) is a hyper-optimization interface that allows operations research scientists to build, develop, and test optimization models with almost all open-source and commercial solvers available. With FelooPy, the users would be able to benchmark the solvers to see if they meet the requirements.

Motivated by the AutoML era, which is ”the process of automating the time-consuming, iterative tasks of machine learning model development,” FelooPy is the first package that is going to provide an AutoOR tool for automating the time-consuming tasks of modeling, implementing, and analyzing an optimization model by providing an integrated (exact and heuristic) optimization environment in Python. Accordingly, FelooPy would also support multiple features such as sensitivity analysis, visualization, and more!

1.1 Advantages

The advantages of FelooPy are as follows:

- An integrated optimization environment.
- Automating operations research model development workflow.
- Straightforward syntax for optimization modeling.
- Using only one syntax for coding.
- Having access to multiple optimization interfaces all at once.
- Having access to multiple exact and heuristic optimization solvers all at once.
- Native support for generating benchmarks of an optimization model.
- Native support for sensitivity analysis.
- Native support for solver interfaces (Coming soon)
- Native support for multi-objective optimization (Coming soon)
- And ....

1.2 Installation

There are multiple ways to install this Python package:

- Using the command `pip install feloopy` in a terminal (note that the Python's `pip` binary package should be in your system PATH).
- Using the command `!pip install feloopy` at the top of your code and implementing it for once.
- Using the following piece of code:

```
import os
os.system("pip install feloopy")
```

- Using the following function:

```
import pip

def install(package):
    if hasattr(pip, 'main'):
        pip.main(['install', package])
    else:
        pip._internal.main(['install', package])

install('feloopy')
```

- Using the package provided in the release section of this link.

2 Usage

2.1 Using exact interfaces and solvers

To start modeling an optimization problem and solving it by an exact solver (algorithm), multiple steps should be taken:
Step 1. Creating the environment.

```
from feloopy import *

model = feloopy([name of the model (string)] , [name of the interface (string)])

..

..

..

..
```

Notably, the model's name is optional but necessary to be provided. The second required argument is the name of the interface, which currently can only be `"gekko"`, `"ortools"`, `"pulp"` or `"pyomo"`.
Step 2. Defining the variables.

```python
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
a = model.pvar([name (string)], [dimension (list of ranges)], [bound (list of two values)])

#A binary variable
b = model.bvar([name (string)], [dimension (list of ranges)], [bound (list of two values)])

#An integer variable
c = model.ivar([name (string)], [dimension (list of ranges)], [bound (list of two values)])

#A free variable
d = model.fvar([name (string)], [dimension (list of ranges)], [bound (list of two values)])
```

Notably, the only important input of the above positive (`pvar`), binary(`bvar`), integer (`ivar` , and free (`fvar`) variable creators is the name of the variable. The others modify the variable according to the characteristics of a mathematical optimization problem. For instance, the second argument is the dimension of the variable which can be defined as `dim=[I,J]` , where both I and J are `range(2)` and `range(3)` . The third argument specifies the bounds of a variable, which if the user desires, should be modified accordingly (e.g., `b=[0,1]`).

Step 3. Defining the objective functions.

```python
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
x = model.pvar('x')

#An integer variable
y = model.ivar('y')

#An objective function
model.obj([a mathematical expression])
```

Notably, according to the interface or solver, the type of the accepted expression as an objective differs. That is because some exact solvers do not accept multiplications between variables or the power of a variable.

Step 4. Defining the constraints.

```
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
x = model.pvar('x')

#An integer variable
y = model.ivar('y')

#An objective function
model.obj(2*x+5*y)

#A Constraint
model.con([an equality or an inequality expression])
```

Notably, similar to the objectives, some kinds of constraints would not be accepted. Moreover, while for exact solvers using notations such as "==', ">=", or "<=" is still supported, it is recommended to use feloopy's specific notations such as "$|e|$", "$|g|$" and "$|l|$".

Step 4. Defining the solver.

Before determining a solver, the interested user can use FelooPy to see which solver is provided by the used interface. This can be done using the following command:

```
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
x = model.pvar('x')

#An integer variable
y = model.ivar('y')

#An objective function
model.obj(2*x+5*y)

#A constraint
model.con(5*x + 3*y |l| 10)

#Another constraint
model.con(2*x + 7*y |l| 10)

#Getting to know the available solvers
model.ava()
```

Which generates:

```
-------------------------
Available LOCAL:
-------------------------
dict_keys(['clp', 'cbc', 'scip', 'glop', 'bop', 'sat', 'gurobi_', 'gurobi', 'cplex_', 'cplex'])
```

Notably, ortools supports more solvers, and a summary is provided herein. You can select the solver based on your knowledge or refer to this link and the corresponding interface website as a reference. Once the desired solver is selected and also installed on your machine (while being added to PATH), then the solve command should be provided as follows:

```python
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
x = model.pvar('x')

#An integer variable
y = model.ivar('y')

#An objective function
model.obj(2*x+5*y)

#A constraint
model.con(5*x + 3*y |l| 10)

#Another constraint
model.con(2*x + 7*y |l| 10)

#Getting to know the available solvers
model.ava()

#Solving the provided model
model.sol([direction of the objective (string)], [solver name (string)])
```

Step 5. Getting to know the overall features of the optimization problem.

```python
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
x = model.pvar('x')

#An integer variable
y = model.ivar('y')

#An objective function
model.obj(2*x+5*y)

#A constraint
model.con(5*x + 3*y |l| 10)

#Another constraint
model.con(2*x + 7*y |l| 10)

#Getting to know the available solvers
model.ava()

#Solving the provided model
model.sol('max', 'scip')

#Getting to know the available solvers
model.inf()
```

which generates the following markdown table:

```
----------------------------------------------------------
   FelooPy (Version 0.1.1) - Released: October 26, 2022
----------------------------------------------------------



PROBLEM FEATURES
 --------
| info      | detail      | variable   | count (cat,tot)    | other      |   count (tot) |
|-----------|-------------|------------|--------------------|------------|---------------|
| model     | simple_milp | positive   | (1, 1)             | objective  |             1 |
| interface | ortools     | binary     | (0, 0)             | constraint |             2 |
| solver    | scip        | integer    | (1, 1)             |            |               |
| direction | max         | free       | (0, 0)             |            |               |
|           |             | tot        | (2, 2)             |            |               |
```

To access benchmark results, the following command should be used:

```python
model.ben('cpt')
```

which generates the following output:

```
BENCHMARK: TIME
 --------
| cpt        | unit       |
|----------|-----------|
| 1472       | micro sec |
| 00:00:00 | h:m:s      |
```

Step 5. Displaying/getting the optimal variable/objective values or the status.

```python
from feloopy import *

model = feloopy('simple_milp' , 'ortools')

#A positive variable
x = model.pvar('x')

#An integer variable
y = model.ivar('y')

#An objective function
model.obj(2*x+5*y)

#A constraint
model.con(5*x + 3*y |l| 10)

#Another constraint
model.con(2*x + 7*y |l| 10)

#Getting to know the available solvers
model.ava()

#Solving the provided model
model.sol('max', 'scip')

#Getting to know the available solvers
model.inf()

#Displaying objective value, problem status, and variables

# model.dis([var1],[var2],...,[varn])

model.dis(x,y)
```

which generates the following:

```
------------------
status: optimal
obj: 7.0
------------------
x: 1.0
y: 1.0
```

However, to get values of the variables, objective and the status of the poblem, the following commands should be used:

```
#Getting objective value, problem status, and variables

x = model.get(x)
y = model.get(y)
obj = model.get_obj()
status = model.get_stat()
print(x,y,obj,status)
```

Step 6. Doing a sensitivity analysis.

To do a sensitivity analysis on a parameter called a the defined model should go inside a function, and the following command should be used:

```
#Doing a sensitivity analysis

def instance(a):
    model = feloopy('simple_milp' , 'ortools')
    x = model.pvar('x')
    y = model.ivar('y')
    model.obj(2*x+5*y)
    model.con(a[0]*x + 3*y |l| 10)
    model.con(a[1]*x + 7*y |l| 9)
    model.sol('max','scip')
    return model

a = [5,2]

sensitivity(instance, a, [-6,6], stepofchange=1, table=True, plot=False)
```

The generated result is:

```
SENSITIVITY ANALYSIS
 --------
|   % change |   objective value |
|------------|-------------------|
|         -6 |           7.12766 |
|         -5 |           7.10526 |
|         -4 |           7.08333 |
|         -3 |           7.06186 |
|         -2 |           7.04082 |
|         -1 |           7.0202  |
|          0 |           7       |
|          1 |           6.9802  |
|          2 |           6.96078 |
|          3 |           6.94175 |
|          4 |           6.92308 |
|          5 |           6.90476 |
|          6 |           6.88679 |
```

If the figure argument is set to true, a figure can also be generated.

2.2 Using heuristic interfaces and solvers

FelooPy currently supports the genetic algorithm for single-objective optimization. While the steps of the model creation (i.e., adding variable, constraints, and objectives) is similar to before, the environment looks a bit different. For instance, the above example should be defined as follows:

```
def instance(agent, active):
    model = feloopy('simple_milp' , 'ga', agent, active)
    x = model.pvar('x')
    y = model.ivar('y')
    model.obj(2*x+5*y)
    model.con(a[0]*x + 3*y |l| 10)
    model.con(a[1]*x + 7*y |l| 9)
    model.sol('max','ga',{'T': 10, 'S': 10, 'Mu': 0.1, 'Cr': 0.5  })
    return model[active]

model = implement(instance)

model.sol()

model.inf()

model.dis(['x',(0,)], ['y',(0,)])

model.ben(['cpt',10])
```

Which should generate the following results:

```
----------------------------------------------------------
   FelooPy (Version 0.1.1) - Released: October 26, 2022
----------------------------------------------------------



PROBLEM FEATURES
 --------
| info      | detail      | variable   | count (cat,tot)    | other      |   count (tot) |
|-----------|-------------|------------|--------------------|------------|---------------|
| model     | simple_milp | positive   | (1, 1)             | objective  |             1 |
| interface | ga          | binary     | (0, 0)             | constraint |             2 |
| solver    | ga          | integer    | (1, 1)             |            |               |
| direction | max         | free       | (0, 0)             |            |               |
|           |             | tot        | (2, 2)             |            |               |

------------------
status:  near optimal
obj: 6.800858139449942
------------------
x(0,):  0.9004290697249709
y(0,):  1.0

BENCHMARK: TIME
 --------
| cpt (ave)    | cpt (std)          | unit      |
|--------------|--------------------|-----------|
| 58049.1      | 23377.52744603244  | micro sec |
| 00:00:00     | 00:00:00           | h:m:s     |

BENCHMARK: OBJ
 --------

|      obj | unit               |
|----------|--------------------|
| 6.97353  | max                |
| 6.81322  | average            |
| 0.122983 | standard deviation |
| 6.56389  | min                |
```

Note the use of different syntax to display variables. Also, `model.ben('cpt',10)` runs the model 10 times and provides benchmarks such as time and obj.

3 Conclusion

FelooPy, an integrated optimization environment for AuthoOR in Python, is a hyper-optimization interface that provides access to multiple exact and heuristic interfaces/solvers or algorithms. Through FelooPy, not only operations research scientists, but also machine learning engineers, simulation engineers, industrial engineers, electrical engineers, chemical engineers, and other experts in other fields of science can have easier access to multiple interfaces and solvers to model and analyze a real-world problem. By supporting the heuristic algorithms in an integrated manner, FelooPy can be considered the first world-class scalable interface in Python to solve more challenging real-world problems that usually belong to the computationally complex NP-hard class. The second version of FelooPy has been releasd on October 26, 2022, by the main contributor Keivan Tafakkori under the MIT license and the copy right notice as: `Copyright (c) 2022 Keivan Tafakkori & FELOOP (https://ktafakkori.github.io/)`. This project can currently be cited using following reference: Tafakkori, K. (2022). Feloopy: An integrated optimization environment for AutoOR in Python (Version 0.1.1) [Computer software]. `https://github.com/ktafakkori/feloopy/`