The pychoco Python library for Constraint Programming

Learn more about Choco, pychoco, and Constraint Programming at https://choco-solver.org/

pychoco

The **pychoco** library provides Python bindings to the Choco Constraint Programming solver. It relies on a native build of the original Java Choco library and is available through PyPI.

Installing pychoco

```
$ pip install pychoco          # in a terminal
```

Loading pychoco

```
>>> from pychoco import *    # in Python
```

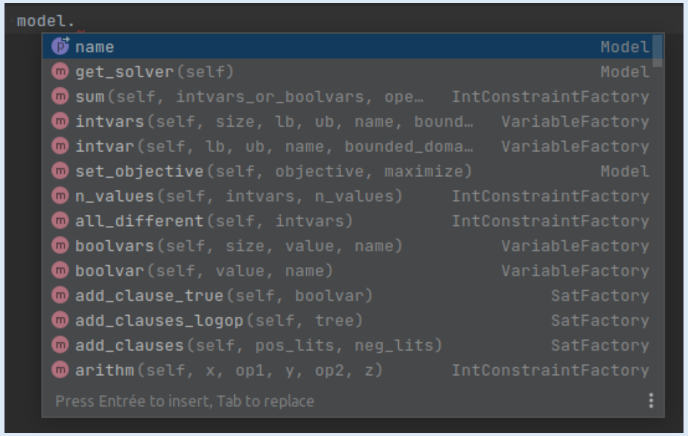
The Model object

The `Model` object is the key component of Choco's and pychoco's API. It provides access to variables, constraints, and to the Solver.

Instantiating a Model

```
>>> m = Model() # default constructor
>>> m2 = Model("my model") # named
```

You can see the API using autocompletion from the Python console or your IDE.



Variables

pychoco supports four main types of variables: *boolvars* (*taking values in {0, 1}*), *intvars* (*taking values in a set of integers, enumerated or bounded*), *setvars* (*taking values in a set interval*), and graphvars (*taking values in a graph interval, directed or undirected*).

Declaring *intvars*

```
# Declaring single intvars
>>> v0 = m.intvar(42, name="v0") # constant
>>> v1 = m.intvar(1, 3) # values in {1,2,3}
>>> v2 = m.intvar([1,3,4,5]) # values in {1,3,4,5}
# Declaring an array of 5 intvars in [-2,2]
>>> vs = m.intvars(5,-2,2)
# Declaring a 5x6 matrix of intvars in [-1,1]
>>> ws = [m.intvars(6,-1,1) for i in range(5)]
```

Declaring *boolvars*

```
>>> b = m.boolvar(name="b")
>>> t = m.boolvar(True) # boolvar fixed to True
```

Declaring *setvars*

```
# Constant setvar equal to {2,3,12}
>>> s1 = m.setvar([2,3,12], name="s1")
>>> s2 = m.setvar({2,3,12}) # using a Python set
# setvar representing a subset of {1,2,3,5,12}
# -> possible values: {}, {2}, {1,3,5}, ...
>>> y = m.setvar({}, {1,2,3,5,12}, name="y")
# superset of {2,3} and subset of {1,2,3,5,12}
# -> possible values: {2,3}, {2,3,5}, {1,2,3}, ...
>>> z = m.setvar({2,3}, {1,2,3,5,12})
```

Declaring *graphvars*

```
### Directed graphs ###
from pychoco.objects.graphs.directed_graph import *
# lower and upper bound of a directed graphvar
>>> n = 3 # maximum number of nodes
>>> lb = create_directed_graph(m, n, [], [])
>>> ub = create_directed_graph(m, n,
          [0,1,2], # nodes
          [[0,1], [1,2],[2,0]]) # edges
# declare the directed graphvar
>>> g = m.digraphvar(lb, ub, "g")

### Undirected graphs ###
from pychoco.objects.graphs.undirected_graph import *
# undirected graphvar with complete graph as ub
>>> lb = create_undirected_graph(m, n, [], [])
>>> ub = m.create_complete_undirected_graph(m, 3)
>>> g = m.graphvar(lb, ub, "g")
```

Constraints

Constraints are logic formulas defining allowed combinations of values for a set of variables, i.e. restrictions that must be respected by feasible solutions. Constraints can be declared in extension, by specifying the valid/invalid tuples, or in intention, by defining a relation between the variables. Constraints can be unary (involving one variable), binary, ternary, ..., or global (unfixed number of variables).

Posting constraints

```
>>> m.all_different(vars).post()
>>> m.arithm(v1, "+", v2, "<=", v0).post()
>>> m.table([v1,v2], [[1,3],[2,5],[3,1]]).post()
```

Reifying constraints

```
>>> x = m.intvar(0, 5)
>>> y = m.intvar(0, 5)
>>> cs = m.arithm(x, "<", y) # cs is NOT posted
>>> b = constraint.reify()
# b = True only if x < y, otherwise b = False
```

Solving and retrieving solutions

Once your model is ready, you can launch the solver through the `Solver` object associated to the `Model` object. The Solver object also offers methods to configure the search.

Finding one solution

```
>>> solver = m.get_solver()
>>> solver.show_statistics()
>>> solver.set_dom_over_w_deg_search([v0,v1,v2])
>>> solution = solver.find_solution()
>>> opt = solver.find_optimal_solution(
                 objective=v0,
                 maximize=True)
>>> v0val = solution.get_int_val(v0)
```

Enumerating several solutions

```
>>> solver = m.get_solver()
>>> solver.show_statistics()
>>> solution = solver.find_all_solutions(
                 solution_limit=10,
                 time_limit="10s")
>>> optimals = solver.find_all_optimal_solutions(
                 v0, True)
```