Python-CPSolver reference manual

Gonzalo Hernandez

September 2024

1 Solving a CSP

$CSP = (X, (D_i)_{i \in X}, C)$

1.1 Package

Python-CPSolver consists of four modules: engine, brancher, propagators, and variables. To use Python-CPSolver, importing the engine module is typically sufficient.

```
from PythonCPSolver.engine import *
```

1.2 Variables and Expressions

PPython-CPSolver uses a generic integer variable type with bounds $-2147483647 \leqslant x \leqslant 2147483647$, where the parameters primarily set the limits.

```
IntVar(min, max, name ) :  IntVar
```

```
x1 = IntVar(1,10,"var")
x2 = IntVar(-5,4)
x3 = IntVar(3)
x4 = IntVar()
```

This example instanciates x_1 labeld as *var*, and x_2, x_3, x_4 without labels, where $(1 \leqslant x_1 \leqslant 10)$, $(-5 \leqslant x_2 \leqslant 4)$, $(3 \leqslant x_3 \leqslant 2147483647)$ and $(-2147483647 \leqslant x_4 \leqslant 2147483647)$.

```
IntVarArray(n, min, max, prefix ) :  IntVar[]
```

```
V = IntVarArray(5,1,10,"var")
W = IntVarArray(3)
```

This example creates two sets of variables, $V = \{v_i \ \mid 0 \leqslant i < 5 \ and \ 1 \leqslant v_i \leqslant 10\}$ labeled as *{var0, var1, .. var4}* and $W = \{w_i \ \mid 0 \leqslant i < 3 \ and \ -2147483647 \leqslant w_i \leqslant 2147483647\}$ without labels.

```
Expression( expr ) :  Expression [Automatic construction]
```

```
ex1 = x1*(x2-6)
ex2 = V[2] >= 4
```

This example shows two mathematical expression useful to filter the searching and to define optimization functions. The operators suported in current version are: $+, -, *, ==, ! =, <, >, <=, >=, \& \ and \ |$.

1.3 Constraints

1.3.1 Specific constraints

`AllDifferent( vars ) : Constraint`

```
c1 = AllDifferent( W )
c2 = AllDifferent( [ x1,x3,V[2] ] )
```

This example implement two constraints. First, Constraint C_1 asserts that *alldifferent*(w_0, w_1, w_2). Second, constraint C_2 asserts that *alldifferent*(x_1, x_3, v_2).

`Linear( vars, total ) : Constraint`

```
c3 = Linear( V, 5 )
c4 = Linear( [x2,x3,x4], W[0] )
```

This example implement two constraints. First, Constraint C_3 asserts that $\Sigma_{v \in V}(v) = 5$. Second, constraint C_4 asserts that $(x_2 + x_3 + v_4) = w_0$. This version only support equalities.

`LinearArgs( args, vars, total ) : Constraint`

```
c5 = LinearArgs( [5,4,2], [x1,x2,x3], 12 )
c6 = LinearArgs( [2,2,2], W, x4 )
```

This example implement two constraints. First, Constraint C_5 asserts that $(5x_1 + 4x_2 + 2x_3) = 12$. Second, constraint C_6 asserts that $(2w_0 + 2w_1 + 2w_2) = x_4$.

1.3.2 Generic constraint

`Constraint( expr ) : Constraint`

```
c7 = Constraint( x1+x4 > ex1 )
c8 = Constraint( x4 == x3+x2 )
```

This example implement two constraints. First, given that $ex1 := x_1 * (x_2 - 6)$ then constraint C_7 asserts that $(x_1 + x_4) > (x_1 * (x_2 - 6))$. Second, constraint C_8 asserts that $x_4 = (x_3 + x_2)$.

1.3.3 Special factions

`count( vars, eqcond ) : expr`

```
c9 = Constraint( count(V,3) < x2 )
```

This example implement a constraint C_9 to assert that $(|\{v \in V | v = 3\}| > x_2)$.

`sum( vars ) : expr`

```
c10 = Constraint( x1*3 == sum(W) )
c11 = Constraint( sum([x1,x2,x3]) > sum(V) )
```

This example implement two constraints. First, The constraint C_10 asserts that $((x_1 * 3) = \Sigma_{w \in W}(w))$. Second, constraint C_{11} asserts that $(\Sigma_{1 \leqslant i \leqslant 3}(x_i) > \Sigma_{v \in V}(v))$.
This example implement a constraint C_9 to assert that $(|\{v \in V | v = 3\}| > x_2)$.