Modeling and Solving Combinatorial Constrained Problems with PyCSP^3

Christophe Lecoutre
with Gilles Audemard and Nicolas Szczepanski

CRIL, Université d'Artois and CNRS, Lens, France

Journée industrielle GDR ROD et RADIA

Lyon – December 14, 2023

Constraint Programming (CP) Open-Source Tools

PyCSP^3, Modeling Library — since december 2019

- MIT Licence
- Github: https://github.com/xcsp3team/pycsp3
- Website: https://pycsp.org/

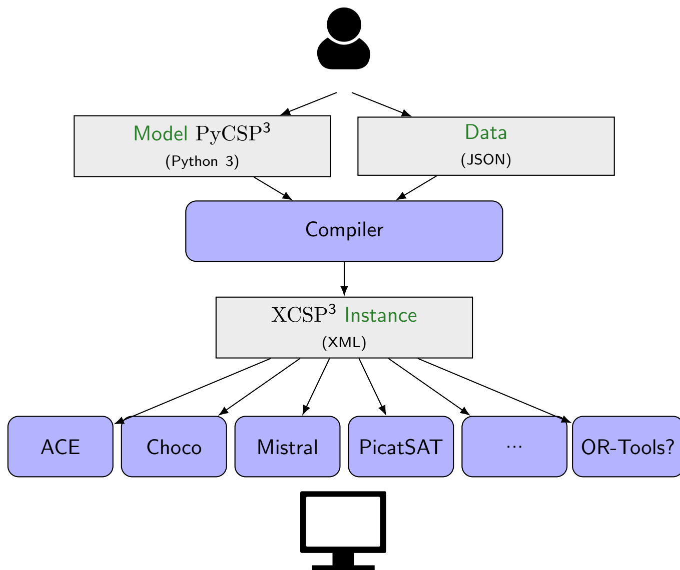
XCSP^3, Integrated Format — since november 2016

- Creative Commons (CC BY-SA 4.0) for specifications
- MIT Licence for any derived piece of software and benchmarks
- Website: https://xcsp.org/

ACE, Generic Constraint Solver — since october 2020 (2000)

- MIT Licence
- Github: https://github.com/xcsp3team/ace

A Complete CP Modeling/Solving Toolchain



Outline

1. CP Modeling with Library PyCSP^3
2. PyCSP^3 for Education
3. PyCSP^3 for Industry
4. CP Solving
5. Conclusion

PyCSP^3

PyCSP^3, Version 2.2 (December 2023)

- Github: https://github.com/xcsp3team/pycsp3
- Guide: https://arxiv.org/abs/2009.00326
- Available as a PyPi package

New functionalities of Version 2.2:

- new (control) structures 'If' and 'Match'
- new derivated constraint forms (Hamming, Exist, NotExist, AllHold)
- new functions 'both' and 'either'
- auto-adjustment of array indexing
- new predefined named tuple 'Task'
- new embedded versions of solvers ACE and Choco

PyCSP^3 influences are XCSP^3 and Numberjack.

Warehouse Location Problem

Problem 034 on CSPLib.

"A company considers opening warehouses at some candidate locations in order to supply its existing stores. Each possible warehouse has the same maintenance cost, and a capacity designating the maximum number of stores that it can supply. Each store must be supplied by exactly one open warehouse."

"The supply cost to a store depends on the warehouse. The objective is to determine which warehouses to open, and which of these warehouses should supply the various stores, such that the sum of the maintenance and supply costs is minimized."

Data (in JSON)

```
{
  "fixedCost": 30,
  "warehouseCapacities": [1,4,2,1,3],
  "storeSupplyCosts":
    [[100,24,11,25,30],[28,27,82,83,74],[74,97,71,96,70],
     [2,55,73,69,61],[46,96,59,83,4],[42,22,29,67,59],
     [1,5,73,59,56],[10,73,13,43,96],[93,35,63,85,46],[47,65,55,71,95]]

}
```

Note that:

- *warehouseCapacities*[*i*] indicates the maximum number of stores that can be supplied by the *ith* warehouse
- *storeSupplyCosts*[*i*][*j*] indicates the cost of supplying the *ith* store with the *jth* warehouse

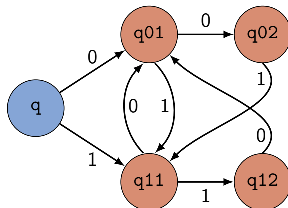
Note that we can use tuple unpacking (**“one line can suffice for handling data”**), and NumPy-like notations.

File `Warehouse.py`

```
from pycsp3 import *

cost, capacities, costs = data
nWarehouses, nStores = len(capacities), len(costs)

# w[i] is the warehouse supplying the ith store
w = VarArray(size=nStores, dom=range(nWarehouses))

satisfy(
  # capacities of warehouses must not be exceeded
  Count(w, value=j) <= capacities[j] for j in range(nWarehouses)
)

minimize(
  # minimizing the overall cost
  Sum(costs[i][w[i]] for i in range(nStores)) + NValues(w) * cost
)
```

Regular Constraints for TTP

- No team can play more than two consecutive home/away games
- Global constraint `regular`

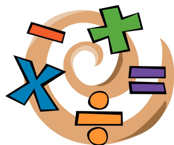



```
q, q01, q02, q11, q12 = states = "q", "q01", "q02", "q11", "q12"
t = [(q, 0, q01), (q, 1, q11), ..., (q02, 1, q11), (q12, 0, q01)]
A = Automaton(start=q, final=states[1:], transitions=t)


satisfy(
  # at most 2 consecutive games at home, or consecutive games away
  h[i] in A for i in range(nTeams)
)
```

Control Structure Match

Arithmetic Target (from Minizinc Challenge 2022)

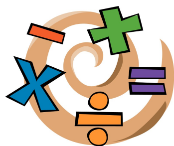
```
...
# computing values associated with all elements
[
    Match(
        x[i],
        Cases={
            VAL: z1[i] == numbers[index[i]],
            ADD: z1[i] == z1[left[i]] + z1[right[i]],
            SUB: z1[i] == z1[left[i]] - z1[right[i]],
            MUL: z1[i] == z1[left[i]] * z1[right[i]],
            DIV: z1[i] * z1[right[i]] == z1[left[i]],
            NO: z1[i] == 0
        }
    ) for i in M
],
...
```



Control Structure If

Arithmetic Target (from Minizinc Challenge 2022)

```
...
# distributivity of multiplication
[
  If(
    x[i] in {ADD, SUB}, x[left[i]] == MUL, x[right[i]] == MUL,
    Then=[
       z1[left[left[i]]] != z1[left[right[i]]],
       z1[left[left[i]]] != z1[right[right[i]]],
       z1[right[left[i]]] != z1[left[right[i]]],
       z1[right[left[i]]] != z1[right[right[i]]]]
  ) for i in M
],
...
```



Stable Marriage

```
w_rankings, m_rankings = data  # ranking by women and men
n = len(w_rankings)
Men, Women = range(n), range(n)

# wf[m] is the wife of the man m
wf = VarArray(size=n, dom=Women)

# hb[w] is the husband of the woman w
hb = VarArray(size=n, dom=Men)

satisfy(
  # spouses must match
  Channel(wf, hb),

  # if m prefers another woman o to his wife, o prefers her husband to m
  [
    If(
      m_rankings[m][o] < m_rankings[m][wf[m]],
      Then=w_rankings[o][hb[o]] < w_rankings[o][m]
    ) for m in Men for o in Women
  ],

  # if w prefers another man o to her husband, o prefers his wife to w
  [
    If(
      w_rankings[w][o] < w_rankings[w, hb[w]],
      Then=m_rankings[o][wf[o]] < m_rankings[o][w]
    ) for w in Women for o in Men
  ]
)
```

Outline

1. CP Modeling with Library PyCSP^3
2. PyCSP^3 for Education
3. PyCSP^3 for Industry
4. CP Solving
5. Conclusion

AllDifferent	AllDifferentMatrix	AllEqual
Cardinality	Channel	Circuit
Count	Cumulative	Decreasing
Element	ElementMatrix	Extension
Increasing	Intension	LexDecreasing
LexIncreasing	MDD	Maximum
Minimum	NValues	NoOverlap
Regular	Sum	Knapsack*
BinPacking*	Hybrid*	Precedence*

See, e.g., https://pycsp.org/documentation/constraints/element/

*: New Jupyter Notebooks

AllInterval	BIBD	BoardColoration
CryptoPuzzle	FlowShopScheduling	GolombRuler
LabeledDice	MagicSequence	Queens
RectanglePacking	SubgraphIsomorphism	Sudoku
TrafficLights	Warehouse	
BACP	Blackhole	Layout
Mario	Nonogram	Quasigroup
RCPSP	SocialGolfers	SportScheduling
StableMarriage	Vellino	
Amaze	Diagnosis	OpenStacks
PizzaVoucher	RackConfiguration	TravelingTournament

See, e.g., https://pycsp.org/documentation/models/COP/PizzaVoucher/

It is possible in Python to

- run a solver (ACE or Choco) and get solutions
- perform some form of incremental solving

See, e.g., https://pycsp.org/documentation/solving/IncrementalSolving/

A Jupyter Notebook is about to be posted about how to set solving options.

We plan to permit a dialog with the solver while letting it alive.

Outline

1. CP Modeling with Library PyCSP^3
2. PyCSP^3 for Education
3. PyCSP^3 for Industry
4. CP Solving
5. Conclusion

Managing Airport Issues

From 25 types of (global) constraints, you can write models for (almost) any kind of integer problems. For example, in airport contexts;

- Airport Check-in Counter Allocation Problem (ACCAP) with fixed opening/closing times
  See ACCAP on GitHub (pycsp3-models)
- Deviation from target landing times
  See AirCraftLanding on GitHub (pycsp3-models)
- Airport Resources at Paris Charles de Gaulle International Airport (contract with CRIL)
  - Check-in Desk Scheduling Optimisation
  - Parking Scheduling Optimisation

Various Realistic/Industrial Problems

On GitHub pycsp3-models:

- RCPSP and its variants
  Cyclic RCPSP, MRCPSP, RCPSP/MAX, RCPSP/WET, ...
- VRP and its variants
  CVRP, CVRP-TW, VRP, VRP-LC, ...
- Hoist Scheduling
- Benzenoide Generation
- Nurse Rostering
- Cargo Assembly in a coal supply chain
- Kidney Exchange
- ...

Aircraft Assembly Line (ONERA, with data from Airbus)

```
...

# x[i] is the starting time of the ith task
x = VarArray(size=nTasks, dom=range(takt * nStations + 1))

# z[j] is the number of operators at the jth station
z = VarArray(size=nStations, dom=lambda i: range(stationMaxOperators[i] +

satisfy(
  # respecting the final deadline
  [x[i] + durations[i] <= takt * nStations for i in range(nTasks)],

  # respecting limit capacities of areas
  [
    Cumulative(
      Task(origin=x[t], length=durations[t], height=usedAreaRooms[t][i])
        for t in areaTasks[i]
    ) <= areaCapacities[i] for i in range(nAreas)
  ],

  # avoiding tasks using the same machine to overlap
  [NoOverlap(tasks=[(x[j], durations[j]) for j in tasksPerMachine[i]])
    for i in range(nMachines)]
)

minimize(
    # minimizing the number of operators
    Sum(z)
)
```

Education for Industry

"Certificat Chef de projet IA at Dauphine
(en partenariat avec MINES & PR[AI]RIE)"

Course about CP and Optimization with PyCSP^3:

- Public: Car industry engineers
- Interactive course
  - Live coding
  - Use of Jupyter Notebooks
- Nice Interaction/dialog wrt problems encountered by various participants (in their companies)

Outline

1. CP Modeling with Library PyCSP^3
2. PyCSP^3 for Education
3. PyCSP^3 for Industry
4. CP Solving
5. Conclusion

Successful Generic Techniques

Generic search-related techniques proven to be successful in many constraint solvers, and in particular in ACE.

- restarts (2000)
- dom/wdeg (2004), and its variant wdeg^{cacd} (2019)
- nogood recording from restarts (2007)
- last-conflict reasoning (2009)
- solution-saving (2017, J. Vion@JFPC)
- BIVS (2017)
- frba/dom (2021), and pick/dom (2023)
- (existential) SAC at preprocessing?

ACE

Constraint Solver:

- ACE (AbsCon Essence)
- Written in Java (25,000 lines of code)
- Version 2.2 on GitHub in December 2023 (https://github.com/xcsp3team/ace)

Rather efficient (competitor to Picat)

- search ingredients mentioned earlier
- efficient propagators for some global constraints

But room for improvement on some items (large domains, certain constraints, learning)

Successful Generic Techniques

Problem VRP (to be minimized) at XCSP^3 Competition 2019

```
java ace Vrp-A-n38-k5.xml
```

Results:

- 757 in 2 minutes
- 1614 in 2 minutes without solution-saving

while:

- 754 in 2 minutes with the new heuristic pick/dom
- 770 with the VBS of XCSP^3 Competition 2019

Successful Generic Techniques

Problem CommunityDetection (to be maximized) at MZN Challenge 2021

```
java ace CommunityDetection-rnd_n100_e50_s50_d30_c9_p30.xml
```

Results:

- 2937 in around 120 seconds
- 2139 in around 120 seconds without solution-saving

while:

- 2963 in 5s with the new heuristic pick/dom
- 2963 with the VBS of MZN Challenge 2021 (6 solvers out of 36)

Successful Generic Techniques

Problem MultiAgentPathFinding at Minizinc Challenge 2017

Observations:

1. The model used for the competition introduces many Boolean variables. Consequently:
   - PyCSP^3 is very long for generating similar instances
   - Choco (and other CP solvers) obtains poor results at the competition
   - ACE is even worse (on similar XCSP^3 instances)
2. A natural CP equivalent model offers:
   - no difficulty at all for generating the instances
   - ACE becoming a strong competitor

Search: Three Different Ways of Processing Conflicts

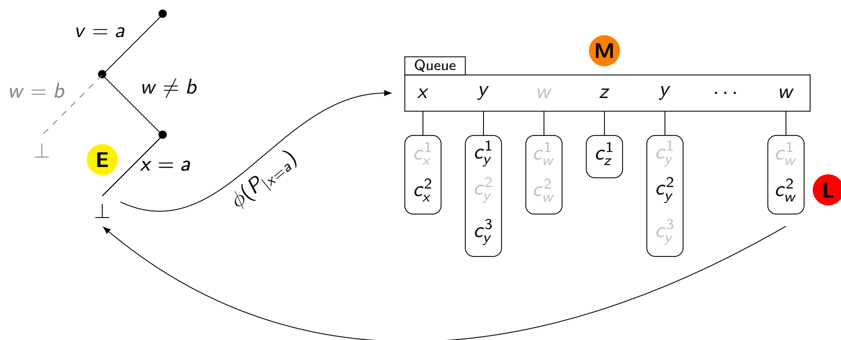


Figure: Illustration of pivotal moments for collecting information about conflicts: this correspond to early (E), midway (M) and late (L) processing of conflicts

Comparison with the state-of-the-art

Search is not dead!

For optimization, look at the results of solver ACE:

- outperformed by Picat when search can be closed (in limited time)
- but serious competitor when long paths towards optimal solutions

However, this is only one important ingredient. And OR-Tools handle three of them:

- search (CP approach)
- clause learning (SAT approach)
- relaxation (LP approach)

Remark: a XCSP^3 Python parser planned to be developped soon $\rightarrow$ a first step towards using OR-Tools from PyCSP^3

Outline

1. CP Modeling with Library PyCSP^3
2. PyCSP^3 for Education
3. PyCSP^3 for Industry
4. CP Solving
5. Conclusion

Simplicity of PyCSP^3

- HiFi compilation preserving the structure (through XCSP^3)
- Easy handling of data: "one line can suffice"

```
nPlanes, times, costs = data
```

- Educational purpose
- Useful for industry too
- Blackbox model: focus on modeling (and then, independent solving)
- Embedding several solvers (a unique command to compile and run):
  - ACE and Choco (from a recurrent dialog with C. Prud'homme)
  - ...
- Incremental solving
- Very large pool of models (and data)

Intelligibility of PyCSP^3

Digital use is more and more subject to legal expectations. In this respect, and in its domain, PyCSP^3 contributes to a general answer to the pressure on digital technology.

Indeed, PyCSP^3 has a high level of intelligibility:

- data are easy to check (due to JSON Technology)
- models are easily understandable (due to the way the library has been conceived)
- problem instances can be easily checked (coarse-grained readable XML structures)
- solutions can be easily checked/visualized/interpreted, by means of:
  - Python interfaces
  - Python tools about explainable constraint solving

And a Rich Ecosystem

And also:

- More than 60 Jupyter notebooks for gently learning CP
- 60 PyCSP^3 models posted on CSPLib;
  see https://www.csplib.org/Languages/PyCSP3/models/
- New repository, https://github.com/xcsp3team/pycsp3-models, with more than 300 models from:
  - scientific papers
  - XCSP3 Competitions
  - CSPLib
  - Minizinc Challenges
- Website for analysing results:
  - https://www.cril.univ-artois.fr/XCSP22
  - https://www.cril.univ-artois.fr/XCSP23

And new forthcoming boosted versions of academic solvers ACE and Choco :-)