PyCSP^3

– Modeling Combinatorial Constrained Problems –

Christophe Lecoutre

CRIL-CNRS UMR 8188
Universite d'Artois
Lens, France

October 2026

Our 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
- Website: https://xcsp.org/

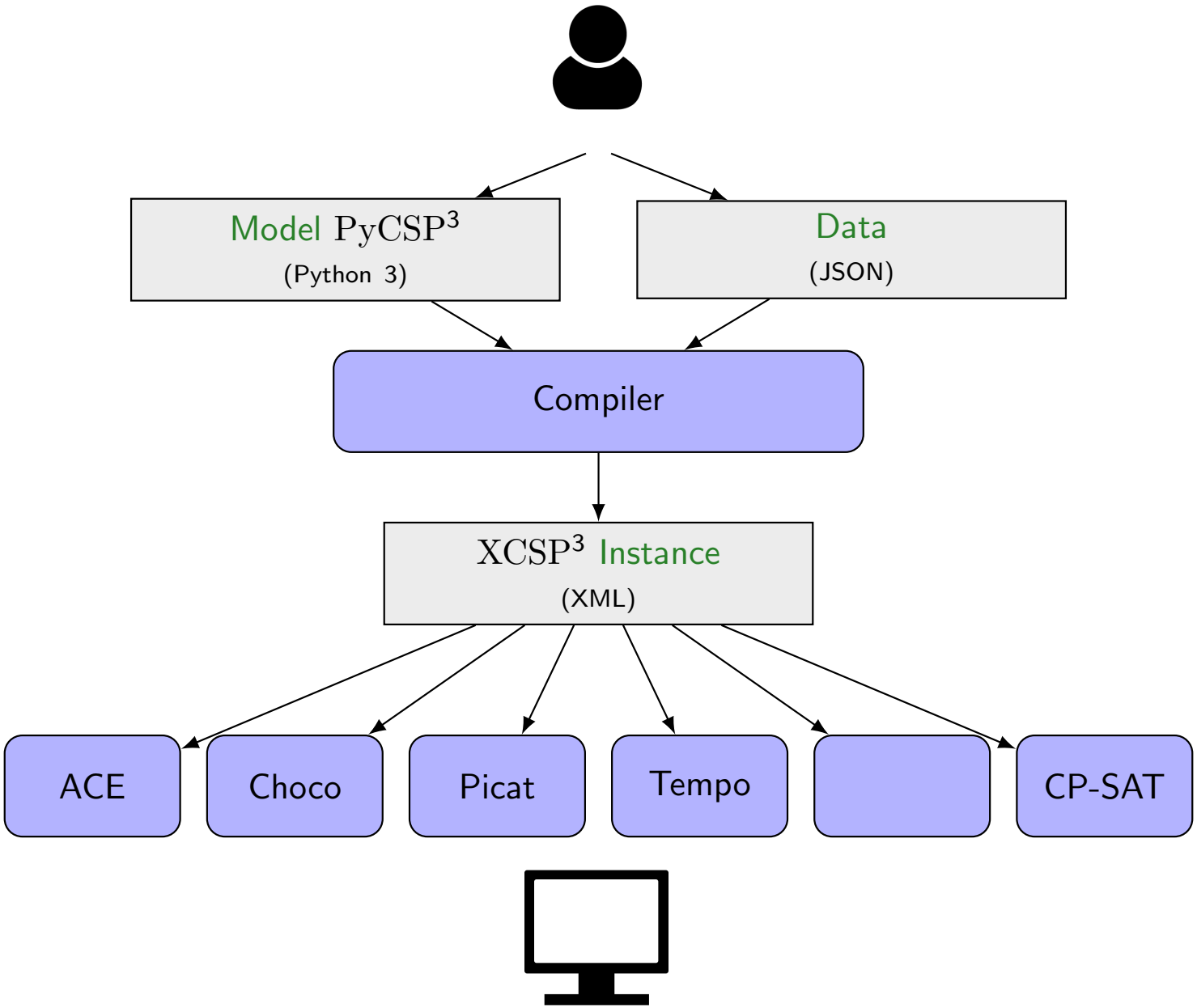
ACE, Generic Constraint Solver — since october 2020

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

Models and Data — since december 2023

- More than 410 models
- Search Engine: https://pycsp.org/models
- Github: https://github.com/xcsp3team/pycsp3-models

A Complete CP Modeling/Solving Toolchain



Outline

1. Practical CP
2. CP Modeling (with Library PyCSP^3)
3. PyCSP^3 for Education
4. CP Solving (with Constraint Solver ACE)
5. CP, a Useful Tool for XAI

Practical CP

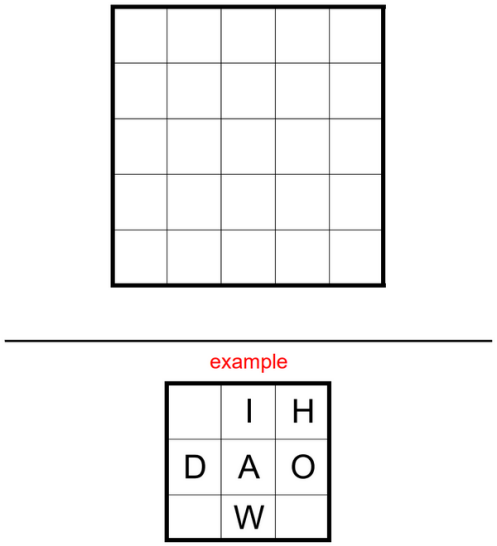


Imagine that:

- your are a puzzle fanatic who wants to show off on the internet (JaneStreet, LinkedIn, ...)
- you are a journalist at *l'equipe 21* who wants to exhibit crazy scenarii about the Champions League
- you are the conference organizer of ROADEF'42, and you have to build the complex programme of the conference
- you are a music composer who wants to deal with Tiling Rhythmic Canons, that are purely rhythmic contrapuntal compositions
- you are a vessel captain who has to load different cargoes to the available tanks of your vessel

Puzzles at JaneStreet

Altered States (https://www.janestreet.com/puzzles)



Enter letters into the 5×5 grid above to achieve the highest score you can. You earn points for each of the 50 U.S. states present in your grid.

- States can be spelled by making King's moves from square to square.
- The score for a state is its length; e.g. OREGON scores 6 points, RHODEISLAND scores 11, etc. The example above scores 13 points1 (for IDAHO, IOWA, and OHIO).
- If a state appears multiple times in your grid, it only scores once.

Puzzles at JaneStreet

```
from pycsp3 import *

...  # TODO

# x[ij] is the letter (index) in cell whose index is ij
x = VarArray(size=n * n, dom=range(26))

# y[k] is 1 if the kth state is present in the matrix
y = VarArray(size=nStates, dom={0, 1})

# z[k][q] is the (flat) cell index in x of the qth letter of the kth state, or 0
z = VarArray(size=[nStates, max(lengths)], dom= range(n * n))

satisfy(
  If(
    y[k] == 0,
    Then=[z[k][q] == 0 for q in range(lengths[k])],  # we force 0 to avoid symmetries
    Else=[
      [x[z[k][q]] == words[k][q] for q in range(lengths[k])]  # ensuring the name is present
      [(z[k][q], z[k][q + 1]) in T for q in range(lengths[k] - 1)]  # ensuring connectedness
    ]
  ) for k in range(nStates)
)

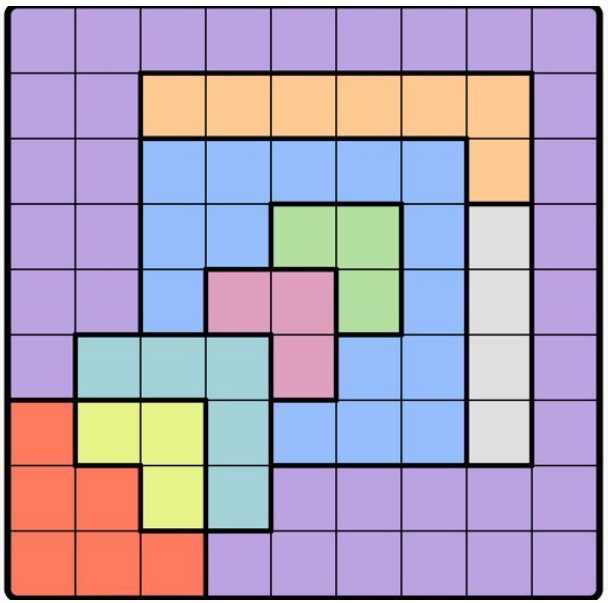
maximize(
  y * lengths
)
```

Solutions with 127 points in 265 seconds with ACE:

java ace AlteredStates.xml -varh=PickOnDom -dc=symmetry-breaking -aaa

And Puzzles at LinkedIn

For example, Queens game using CP



https://www.linkedin.com/pulse/queens-game-using-cp-alireza-soroudi-2hbke/

By the way, you can avoid buying a quantuum computer by choosing the right model for the n-queens problem. Let as an exercice.

Champions League with 8 Matchdays

Matchday 1



...

Matchday 2



...

Matchday 3



...

...

L'ÉQUIPE 21

As a journalist, you can exhibit some crazy scenarii:

- What is the highest number of points that can be reached by a team without being qualified
  - in the first 8 teams
  - for the first 24 teams
- What is the lowest number of points that can be reached by a team while being qualified
  - in the first 8 teams
  - for the first 24 teams
- ...

```
schedule, position = data  # position is 8 or 24 for example

nWeeks, nMatchesPerWeek, nTeams = len(schedule), len(schedule[0]), len(schedule[0]) * 2
assert nWeeks == 8 and nTeams == 36  # for the moment

WON, DRAWN, LOST = results = range(3)

# x[w][k] is the result of the kth match in the wth week
x = VarArray(size=[nWeeks, nMatchesPerWeek], dom=results)

# y[w][i] is the number of points won by the ith team in the wth week
y = VarArray(size=[nWeeks, nTeams], dom={0, 1, 3})

# z[i] is the number of points won by the ith team
z = VarArray(size=nTeams, dom=range(3 * nWeeks + 1))

# the target team that must be ranked at the specified position
target = Var(dom=range(nTeams))

# the number of teams with a score better than the target team
better_target = Var(dom=range(nTeams))

# the number of teams with a score equal to the target team
equal_target = Var(dom=range(nTeams))

satisfy(
  # computing points won for every match
  [
    (x[w][k], y[w][i], y[w][j]) in {(WON, 3, 0), (DRAWN, 1, 1), (LOST, 0, 3)}
      for w in range(nWeeks) for k, (i, j) in enumerate(schedule[w])
  ],

  # computing the number of points of each team
  [z[i] == Sum(y[:, i]) for i in range(nTeams)],

  ... TODO
)

maximize(
  # maximizing the number of points of the target team
  z[target]
)
```

ROADEF Conferences

The ROADEF conference is :

- the largest French-speaking event bringing people in combinatorial optimization, OR, CP and industry;
- organized annually while welcoming around 600 participants.





Roadef

You goal is to schedule the ROADEF parallel sessions into available time slots while avoiding clashes among research working groups. Papers of each topic must be allocated to such time slots.

Some data:

	2024	2023	2022	2021
Number of topics	40	47	42	27
Number of parallel rooms	7	7	8	11
Paper range per slot	3..6	3..6	3..5	3..5
Number of papers	307	358	311	182
Number of working groups	20	24	24	17

```
Sessions, nSlots, papers_range, nGroups, nSessionPapers, slot_cap, ... = data
NO = nSlots

# pairs (i,j) of sessions shared by a positive number v of working groups
conflicts = [(i, j, v) for i, j in combinations(nSessions, 2)
   if (v := len(set(session_groups[i]).intersection(set(session_groups[j])))) > 0]

T = [(t,v) for t in range(nSlots) for v in range(papers_range[0], slot_cap[t]+1)] + [(NO, 0)]

maxSlots = [next(t if (t + 1) * papers_range[0] > k else t + 1 for t in range(nSlots)
  if sum(slot_cap[:t+1]) >= k) for i in range(nSessions) if (k := nSessionPapers[i],)]

# x[i][k] is the kth slot used for session i (NO if unused slot)
x = VarArray(size=[nSessions, maxSlots], dom=range(nSlots + 1))

# y[i][k] is the number of papers presented in the kth slot used for session i
y = VarArray(size=[nSessions, maxSlots], dom=[0] + papers_range)

satisfy(
  # channeling variables
  [(x[i][k], y[i][k]) in T for i in range(nSessions) for k in range(maxSlots[i])],

  # allocating the right number of papers for each session
  [Sum(y[i]) == nSessionPapers[i] for i in range(nSessions)],

  # not exceeding the possible number of parallel sessions
  ... TODO

  # forbidding some slots
  [x[i][k] != t for (i, t) in forbidden for k in range(maxSlots[i])],

  # ensuring different slots (while ordering them)
  [
    If(
      x[i][k + 1] != NO,
      Then=x[i][k] < x[i][k + 1]
    ) for i in range(nSessions) for k in range(maxSlots[i] - 1)
  ]
)

minimize(
  ... TODO
)
```

Tiling Rhytmic Canons

Tiling Rhythmic Canons are purely rhythmic contrapuntal compositions. It is well-known that all canons are periodic: for a fixed period n, a tiling rhythmic canon is then represented by a couple of sets A, B subsets of $\{0, 1, 2, ..., n-1\}$ where:

- A defines the sequence of beats played by every voice,
- B the offsets at which the voices start to play,
- and such that at every beat there is exactly one voice playing.

If one of the sets, say A, is given, it is well-known that the problem of finding a complement B has in general no unique solution.
From a mathematical point of view, the most interesting canons are those in which both sets A and B are aperiodic (also known as Vuza Canons).

Tank Allocation for Liquid Bulk Vessels

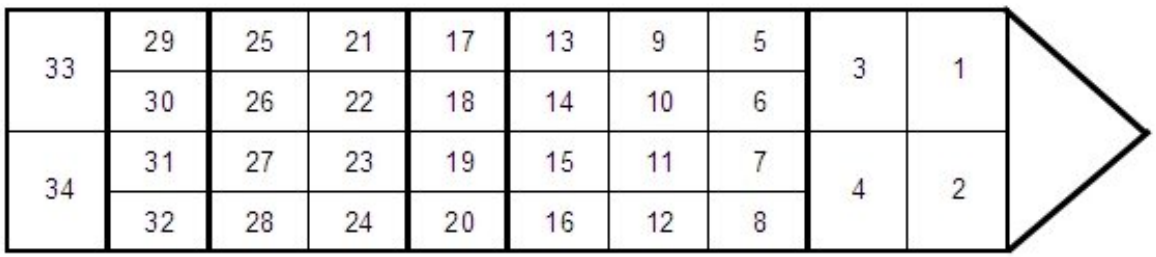


You have to put different cargoes (volumes of chemical products to be shipped by the vessel) to the available tanks of the vessel. You have to:

- prevent chemicals from being loaded into certain types of tanks;
- prevent some pairs of cargoes to be placed next to each other.

To minimize the inconvenience of tank cleaning, an ideal loading plan should maximize the total volume of unused tanks (i.e. free space).

Layout of the Vessel



The characteristics of this (real) instance (coming from a major chemical tanker company):

- there are 20 cargoes with volumes ranging from 381 to 1527 tons;
- the vessel has 34 tanks with capacities from 316 to 1017 tons;
- there are 5 pairs of cargoes that cannot be placed into adjacent tanks;
- each tank has between 1 to 3 cargoes that cannot be assigned to it.

Model for Tank Allocation

```
volumes, conflicts, tanks = data
T = conflicts + [(v, u) for u, v in conflicts]
capacities, impossible_cargos, neighbours = zip(*tanks)
nCargos, nTanks = len(volumes), len(tanks)

DUMMY_CARGO = nCargos

# x[i] is the cargo (type) of the ith tank (DUMMY_CARGO, if empty)
x = VarArray(size=nTanks, dom=range(nCargos + 1))

satisfy(
  # allocating a compatible cargo to each tank
  [x[i] not in impossible_cargos[i] for i in range(nTanks)],

  # ensuring no adjacent tanks containing incompatible cargo
  [(x[i], x[j]) not in T for i in range(nTanks) for j in neighbours[i]]

  # ensuring each cargo is shipped
  [
    Sum(capacities[i] * (x[i] == cargo) for i in range(nTanks)) >= volumes[cargo]
      for cargo in range(nCargos)
  ]
)

maximize(
  # maximizing free space
  Sum(capacities[i] * (x[i] == DUMMY_CARGO) for i in range(nTanks))
)
```

Outline

1. Practical CP
2. CP Modeling (with Library PyCSP^3)
3. PyCSP^3 for Education
4. CP Solving (with Constraint Solver ACE)
5. CP, a Useful Tool for XAI

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)

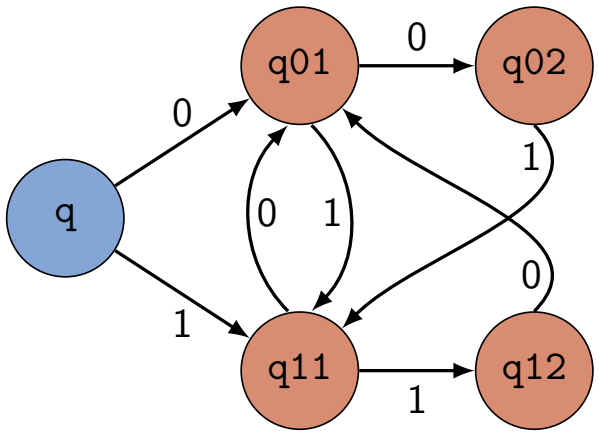
# w[i] is the warehouse supplying the ith store
w = VarArray(size=nStores, dom=range(nWarehouses))

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)
)
```

Cardinality Constraints for TTP

Each team must play exactly two times against each opponent:

```
satisfy(
  # each team must play exactly two times against each other team
  Cardinality(
    within=o[i],
    occurrences={j: 2 for j in range(nTeams) if j != i}
  ) for i in range(nTeams)
)
```



Outline

1. Practical CP
2. CP Modeling (with Library PyCSP^3)
3. PyCSP^3 for Education
4. CP Solving (with Constraint Solver ACE)
5. CP, a Useful Tool for XAI

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/

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/

PyCSP^3 for Education

Zero Installation with Google Colab

- in two minutes, you can model and solve your first problem
- embedded solvers (ACE and Choco) are directly available

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/

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

Outline

1. Practical CP
2. CP Modeling (with Library PyCSP^3)
3. PyCSP^3 for Education
4. CP Solving (with Constraint Solver ACE)
5. CP, a Useful Tool for XAI

Successful Generic Techniques

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

- restarts (2000)
- variable ordering heuristics
  - wdeg/dom (2004), and the variant wdeg^{cacd} (2019)
  - frba/dom (2021)
  - pick/dom (2023)
- nogood recording from restarts (2007)
- last-conflict reasoning (2009)
- solution-saving (2017, J. Vion@JFPC)
- BIVS (2017)

Guiding Search From Conflicts

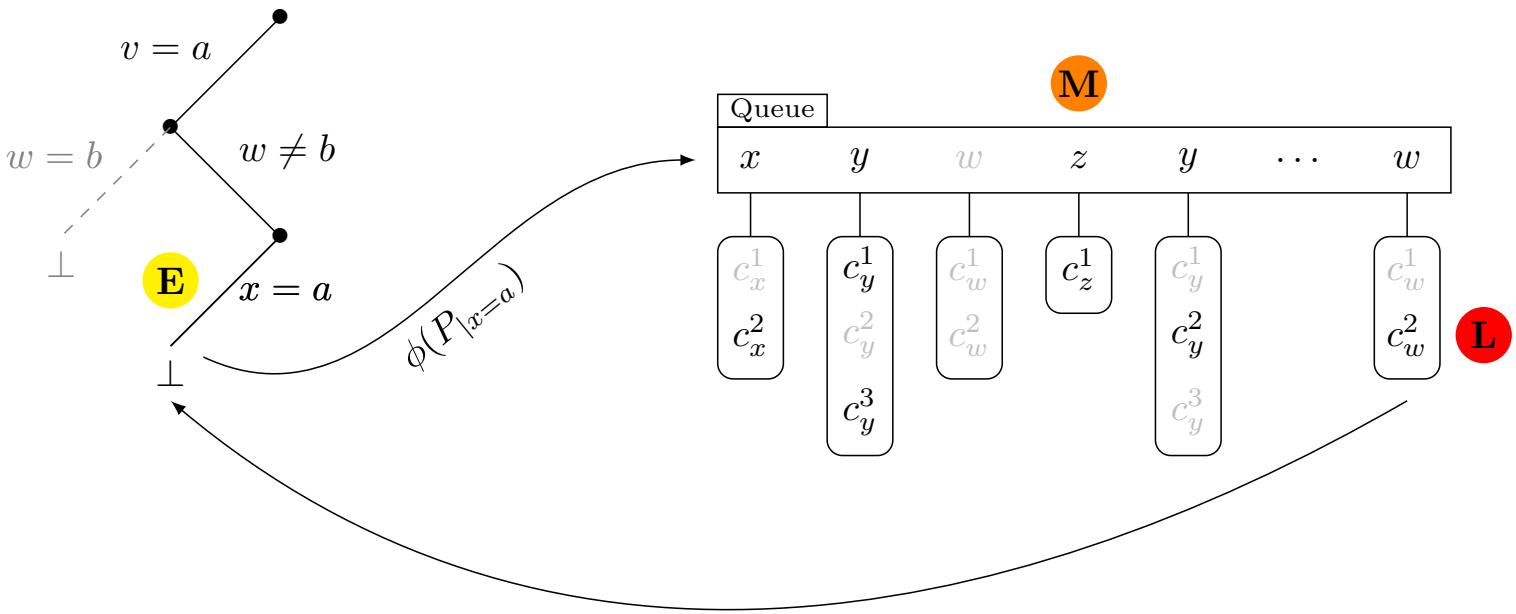


Illustration of pivotal moments for collecting information about conflicts. This correspond to

- early (E)
- midway (M)
- late (L)

processing of conflicts

ACE



Constraint Solver:

- ACE
- Written in Java (25,000 lines of code)
- Version 2.3 on GitHub in March 2024 (https://github.com/xcsp3team/ace)

Efficiency (most robust solver for optimization in XCSP^3 2022 and 2023 Competitions)

- search ingredients mentioned earlier
- efficient propagators

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 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 pick/dom
- 2963 with the VBS of MZN Challenge 2021 (6 solvers out of 36)

Outline

1. Practical CP
2. CP Modeling (with Library PyCSP^3)
3. PyCSP^3 for Education
4. CP Solving (with Constraint Solver ACE)
5. CP, a Useful Tool for XAI

Intelligible/User-friendly Tools (PyCSP^3 and ACE)

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

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

- Educational purpose (but can be useful for industry too)
- Blackbox model: focus on modeling (and then, independent solving)
- Embedded solvers (a unique command to compile and run):
- Zero installation (from Google Colab)
- Incremental solving
- Website for analysing results
  e.g., see https://www.cril.univ-artois.fr/XCSP23

CP: a useful tool for XAI

CP is a useful technology:

- very declarative (and clean separation modeling/solving)
- very flexible
- very efficient on some problems

CP is an historic field of symbolic AI (reasoning with knowledge):

- logic and constraint reasoning
- generic tool for complex calculations

CP is an ingredient of the library PyXAI that:

- bring explanations of various forms from classifiers resulting of ML techniques (Decision Trees, Random Forests, Boosted Trees)
- such explanations being abductive (why?) or contrastive (why not?)

CP: a useful Tool for XAI

Great invited talk by Frédéric Koriche at JFPC'24 about "Comment utiliser l'optimisation de contraintes pour expliquer n'importe quel classifieur ?" from:

> *Frederic Koriche, Jean-Marie Lagniez, Stefan Mengel, Chi Tran*
> *Learning Model Agnostic Explanations via Constraint Programming.*
> *In proceedings of ECML/PKDD'24.*

Abstract:

> *One of the major challenges of XAI is to explain, in terms understandable to humans, predictions made by complex classifiers, such as ensemble-based models or neural networks. When such models are viewed as black boxes, the aim is to identify a small set of interpretable attributes that jointly determine the response made by the black box with minimal error. However, inferring such model-agnostic explanations is computationally very expensive: even for Boolean classifiers, the problem is complete for the class NP power PP, which implies that finding an optimal explanation is beyond the reach of current solvers. Our aim is then to seek to approach the optimum using both sampling and constraint programming. From a theoretical point of view, such an approach provides bounds that are probably approximately correct and, from an experimental point of view, it offers a compromise between accuracy and computation time.*