Integrating CP with ML and explanations

The Sudoku Assistant App (and CPMpy)

Prof. Tias Guns <tias.guns@kuleuven.be> @TiasGuns
Maxime Mulamba
Milan Pesa
Ignace Bleukx
Emilio Gamba
Bart Bogaerts
Senne Berden

KU LEUVEN

European Research Council

AI

General-purpose constraint solving

Model + Solve

Decision variables
Constraints
Objective function

Constraint solving paradigm

Model + Solve

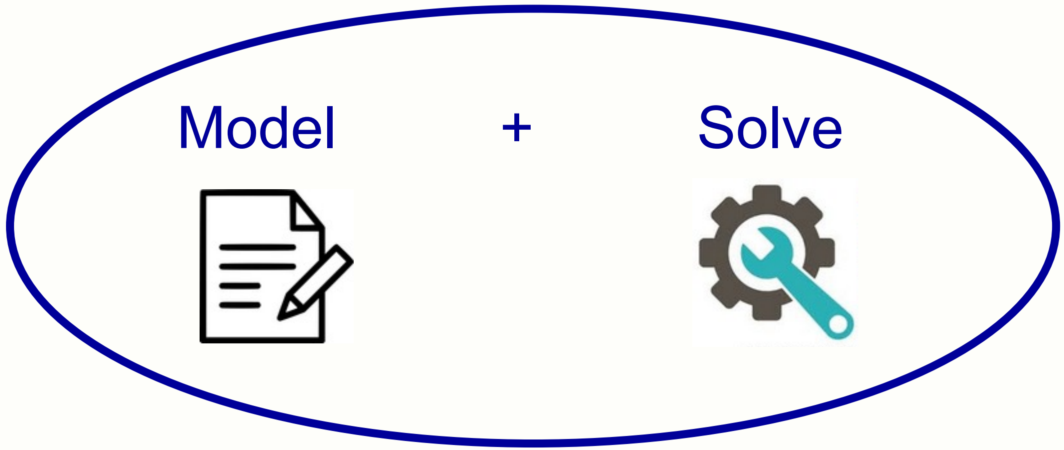
Rich research on
modeling languages, automatic transformations, solver independence, modelling tools

Tools: MiniZinc, Essence', CPMpy

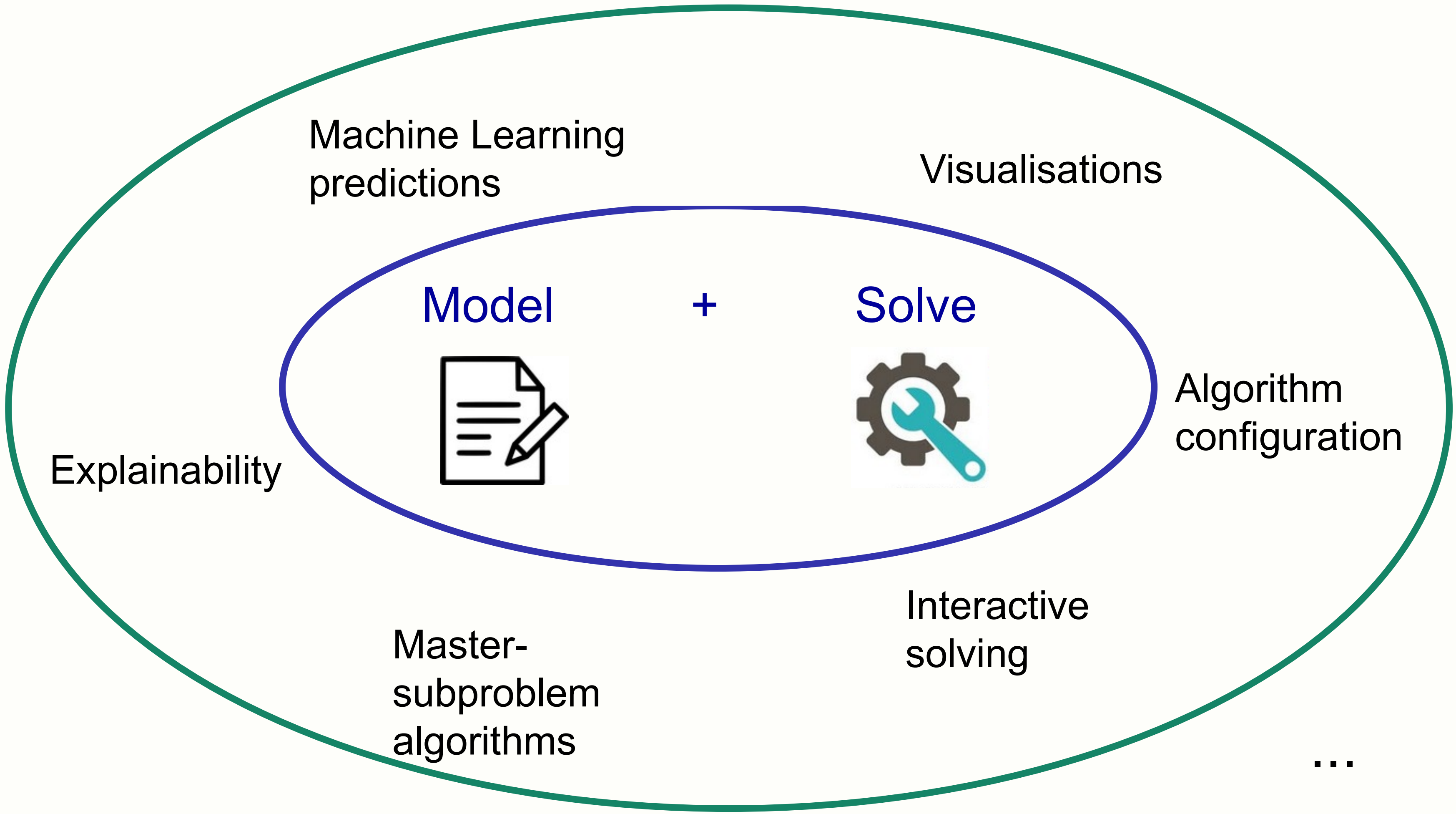
Rich research on
efficient solvers, (global) constraint propagators, automatic search, algorithm configuration, ...

Tools: OrTools, Gecode, Gurobi, Z3, ...

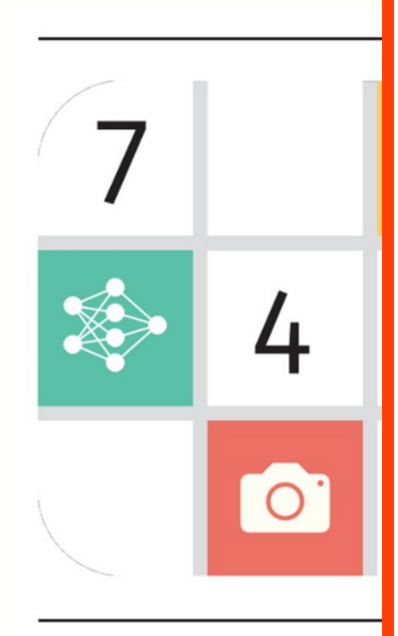
Wider view



Wider view: integration



Percep… …Solving:



BEST TECHNICAL DEMONSTRATION AWARD

FEBRUARY 7-14, 2023

THE ASSOCIATION FOR THE ADVANCEMENT OF ARTIFICIAL INTELLIGENCE

proudly presents

THE AWARD FOR 2023 AAAI BEST TECHNICAL DEMONSTRATION TO

Tias Guns, Emilio Gamba, Maxime Mulamba Ke Tchomba, Ignace Bleukx, Senne Berden, & Milan Pesa

A DEMONSTRATION OF SUDOKU ASSISTANT —
AN AI-POWERED APP TO HELP SOLVE PEN-AND-PAPER SUDOKUS

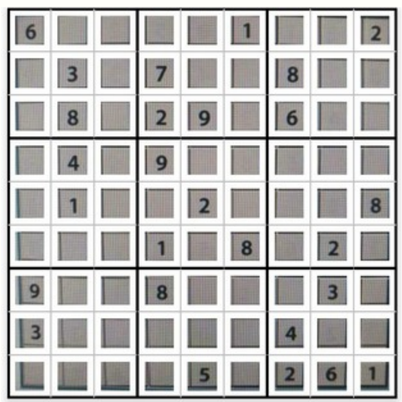
PRESENTED AT THE 37TH AAAI CONFERENCE ON ARTIFICIAL INTELLIGENCE

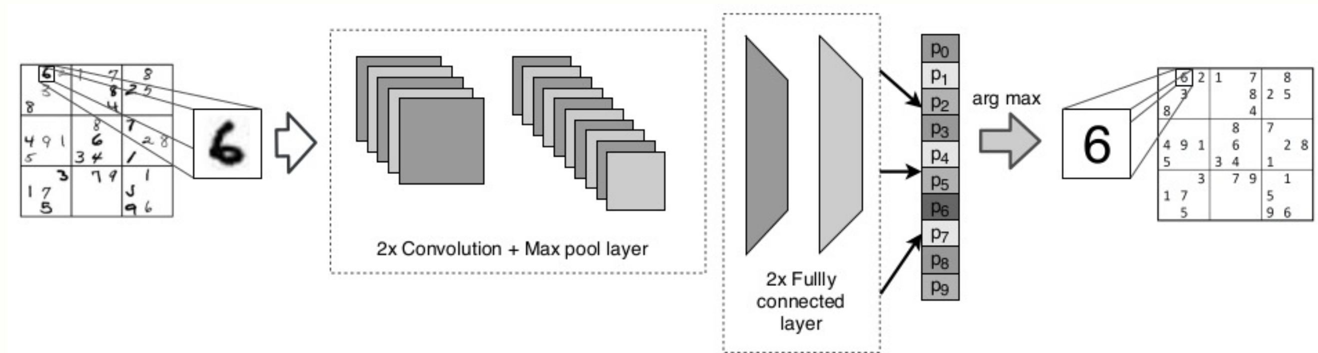
https://s…

GET IT ON Google Play

Sudoku Assistant: usage demo

1) Recognizing the Sudoku digits





- Cut into 81 pieces (introduces additional noise)
- Predict 1-9 or empty (printed and handwritten, robust to borders and markings)
- Custom but standard ML

2) solving the sudoku

Rules of Sudoku (source: sudoku.com)

- **Sudoku Rule № 1: Use Numbers 1-9**

Sudoku is played on a grid of 9 x 9 spaces. Within the rows and columns are 9 “squares” (made up of 3 x 3 spaces). Each row, column and square (9 spaces each) needs to be filled out with the numbers 1-9, without repeating any numbers within the row, column or square. Does it sound complicated? As you can see from the image below of an actual Sudoku grid, each Sudoku grid comes with a few spaces already filled in; the more spaces filled in, the easier the game – the more difficult Sudoku puzzles have very few spaces that are already filled in.

	7	2			4	9		
3		4		8	9	1		
8	1	9			6	2	5	4
7		1					9	5
9					2		7	
			8		7		1	2
4		5			1	6	2	
2	3	7				5		1
				2	5	7		

Model + Solve

Decision variables
Constraints
Objective function

Decision variables
Constraints
Objective function

2) solving the sudoku

Model =

- Variables, with a domain

- grid[i,j] :: {1..9}   for i,j in {1..9}

- Constraints over variables

- alldifferent(grid[i,:])   for i in {1..9}   – rows
  alldifferent(grid[:,j])   for j in {1..9}   – columns
  alldifferent(square(grid, k,l))   for k,l in {1..3}   – squares

grid[i,j] == given[i,j] if given[i,j] not empty   for i,j in {1..9}

- **Sudoku Rule № 1: Use Numbers 1-9**

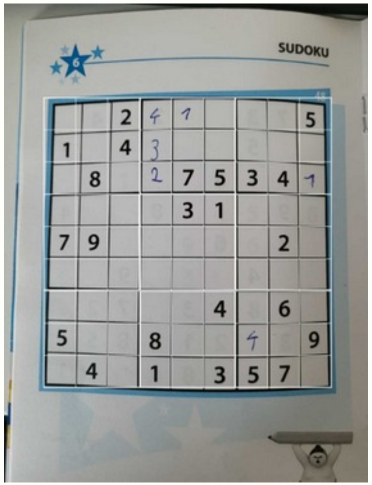
Sudoku is played on a grid of 9 x 9 spaces. Within the rows and columns are 9 “squares” (made up of 3 x 3 spaces). Each row, column and square (9 spaces each) needs to be filled out with the numbers 1-9, without repeating any numbers within the row, column or square. Does it sound complicated? As you can see from the image below of an actual Sudoku grid, each Sudoku grid comes with a few spaces already filled in; the more spaces filled in, the easier the game – the more difficult Sudoku puzzles

Model.solve()

Model

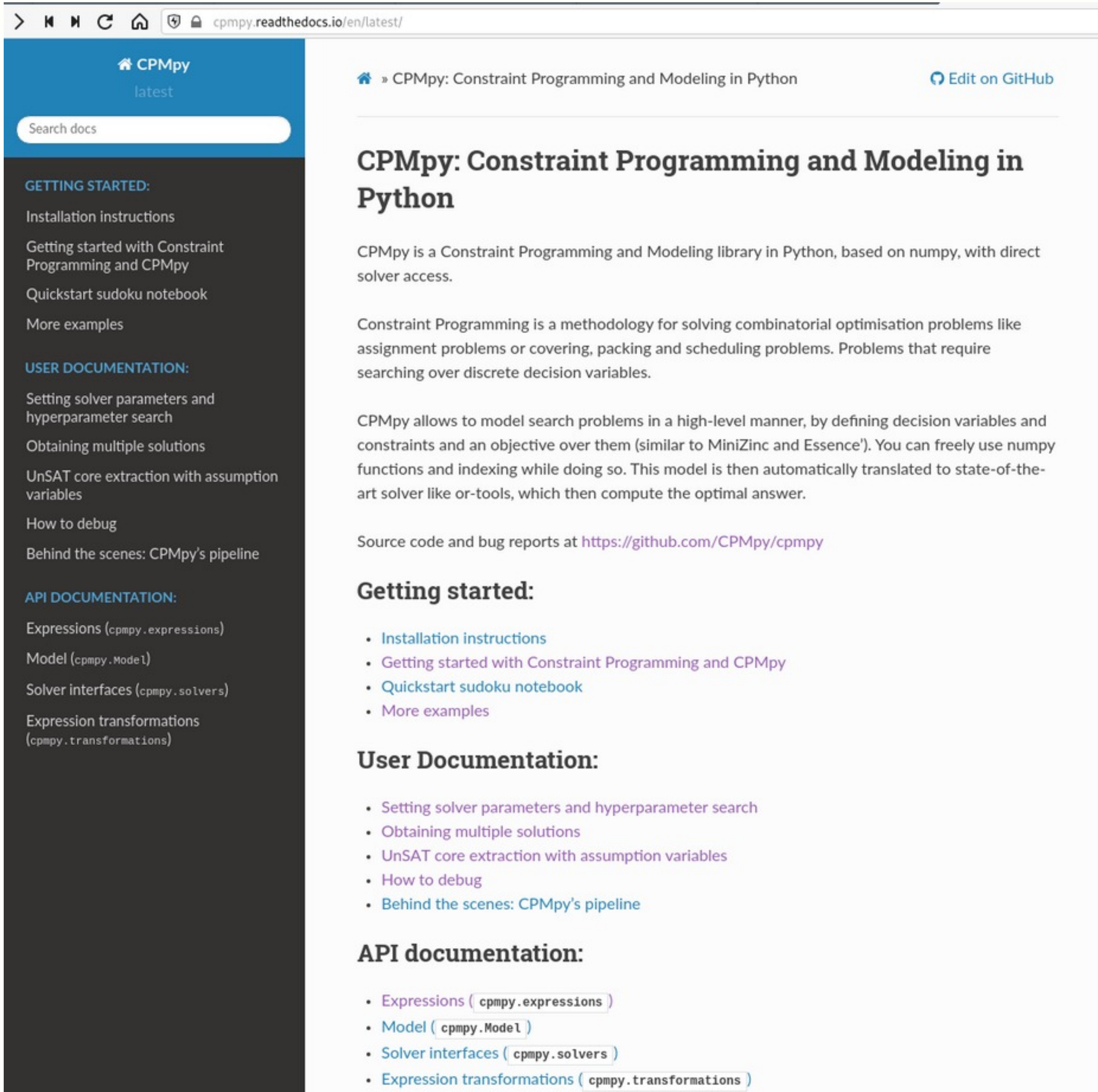
2) solving the sudoku

Decision variables
Constraints
Objective function




```python
e = 0 # value for empty cells
given = np.array([
    [e, e, 2,  4, 1, e,  e, e, 5],
    [1, e, 4,  3, e, e,  e, e, e],
    [e, 8, e,  2, 7, 5,  3, 4, 1],

    [e, e, e,  e, 3, 1,  e, e, e],
    [7, 9, e,  e, e, e,  e, 2, e],
    [e, e, e,  e, e, e,  e, e, e],

    [e, e, e,  e, e, 4,  e, 6, e],
    [5, e, e,  8, e, e,  4, e, 9],
    [e, 4, e,  1, e, 3,  5, 7, e]])
```

```python
model = Model()

# Variables
puzzle = intvar(1, 9, shape=given.shape, name="puzzle")

# Constraints on rows and columns
model += [AllDifferent(row) for row in puzzle]
model += [AllDifferent(col) for col in puzzle.T]

# Constraints on blocks
for i in range(0,9, 3):
    for j in range(0,9, 3):
        model += AllDifferent(puzzle[i:i+3, j:j+3])

# Constraints on values (cells that are not empty)
model += (puzzle[given!=e] == given[given!=e])


model.solve()
```


cpmpy.readthedocs.io/en/latest/

CPMpy
latest
Search docs

GETTING STARTED:
- Installation instructions
- Getting started with Constraint Programming and CPMpy
- Quickstart sudoku notebook
- More examples

USER DOCUMENTATION:
- Setting solver parameters and hyperparameter search
- Obtaining multiple solutions
- UnSAT core extraction with assumption variables
- How to debug
- Behind the scenes: CPMpy's pipeline

API DOCUMENTATION:
- Expressions (`cpmpy.expressions`)
- Model (`cpmpy.Model`)
- Solver interfaces (`cpmpy.solvers`)
- Expression transformations (`cpmpy.transformations`)

» CPMpy: Constraint Programming and Modeling in Python   Edit on GitHub

CPMpy: Constraint Programming and Modeling in Python

CPMpy is a Constraint Programming and Modeling library in Python, based on numpy, with direct solver access.

Constraint Programming is a methodology for solving combinatorial optimisation problems like assignment problems or covering, packing and scheduling problems. Problems that require searching over discrete decision variables.

CPMpy allows to model search problems in a high-level manner, by defining decision variables and constraints and an objective over them (similar to MiniZinc and Essence'). You can freely use numpy functions and indexing while doing so. This model is then automatically translated to state-of-the-art solver like or-tools, which then compute the optimal answer.

Source code and bug reports at https://github.com/CPMpy/cpmpy

Getting started:

- Installation instructions
- Getting started with Constraint Programming and CPMpy
- Quickstart sudoku notebook
- More examples

User Documentation:

- Setting solver parameters and hyperparameter search
- Obtaining multiple solutions
- UnSAT core extraction with assumption variables
- How to debug
- Behind the scenes: CPMpy's pipeline

API documentation:

- Expressions (`cpmpy.expressions`)
- Model (`cpmpy.Model`)
- Solver interfaces (`cpmpy.solvers`)
- Expression transformations (`cpmpy.transformations`)

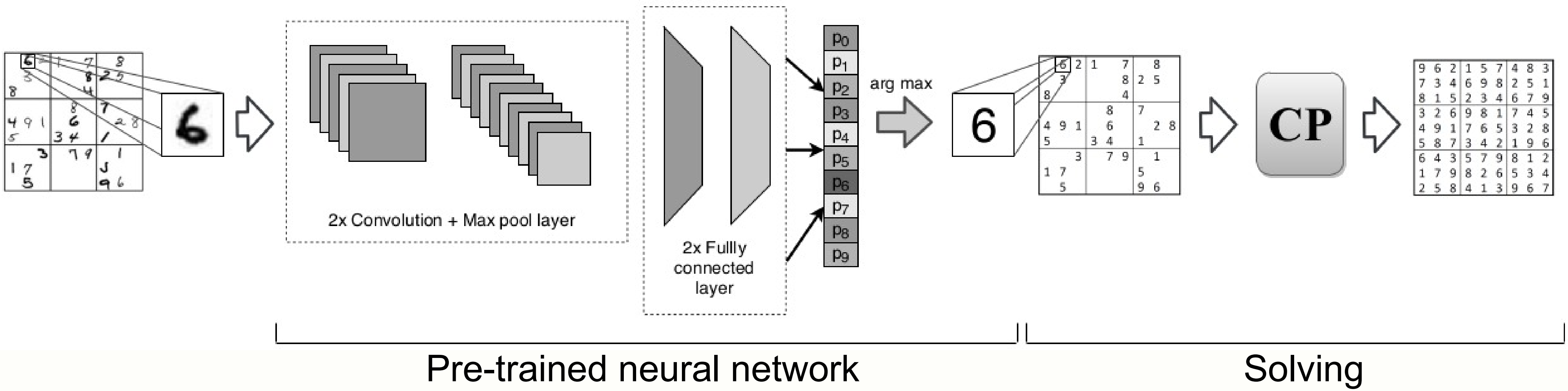
https://github.com/CPMpy/cpmpy

CPMpy:
- Open source
- Python/Numpy based
- Direct solver access

Supported solvers:
- ORTools (CP)
- Gurobi, Exact (MIP)
- Z3 (SMT)
- PySAT (SAT)
- PySDD (knowledge comp)
- More to come… (SCIP, CPOpt)

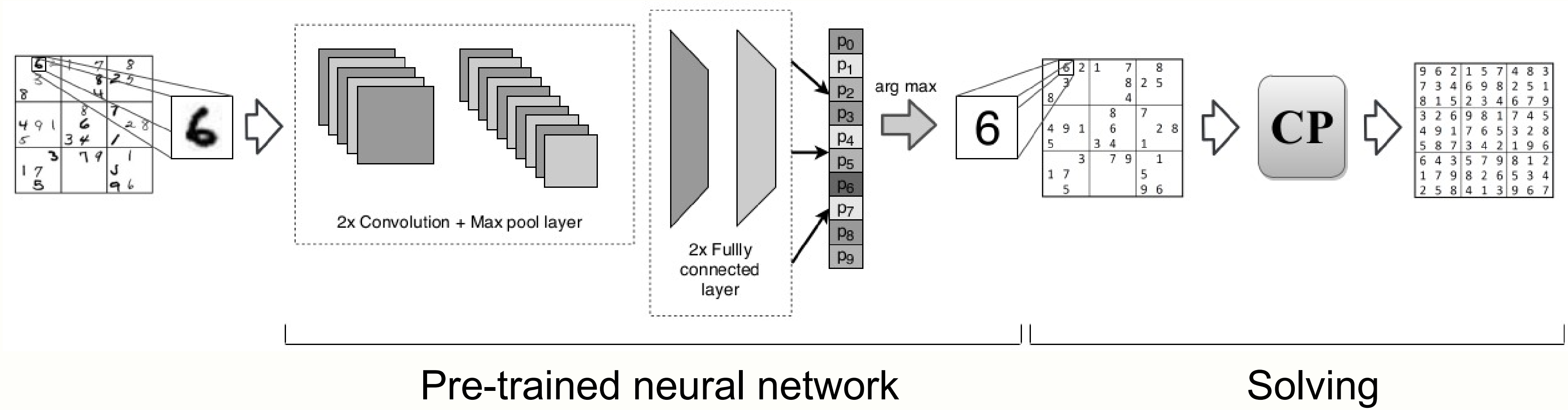
Perception-based constraint solving

Pedagogical instantiation: visual sudoku (naïve)



	accuracy img	cell	grid	failure rate grid	time average (s)
baseline	94.75%	15.51%	14.67%	84.43%	0.01

Perception-based constraint solving



<u>What is going on?</u>

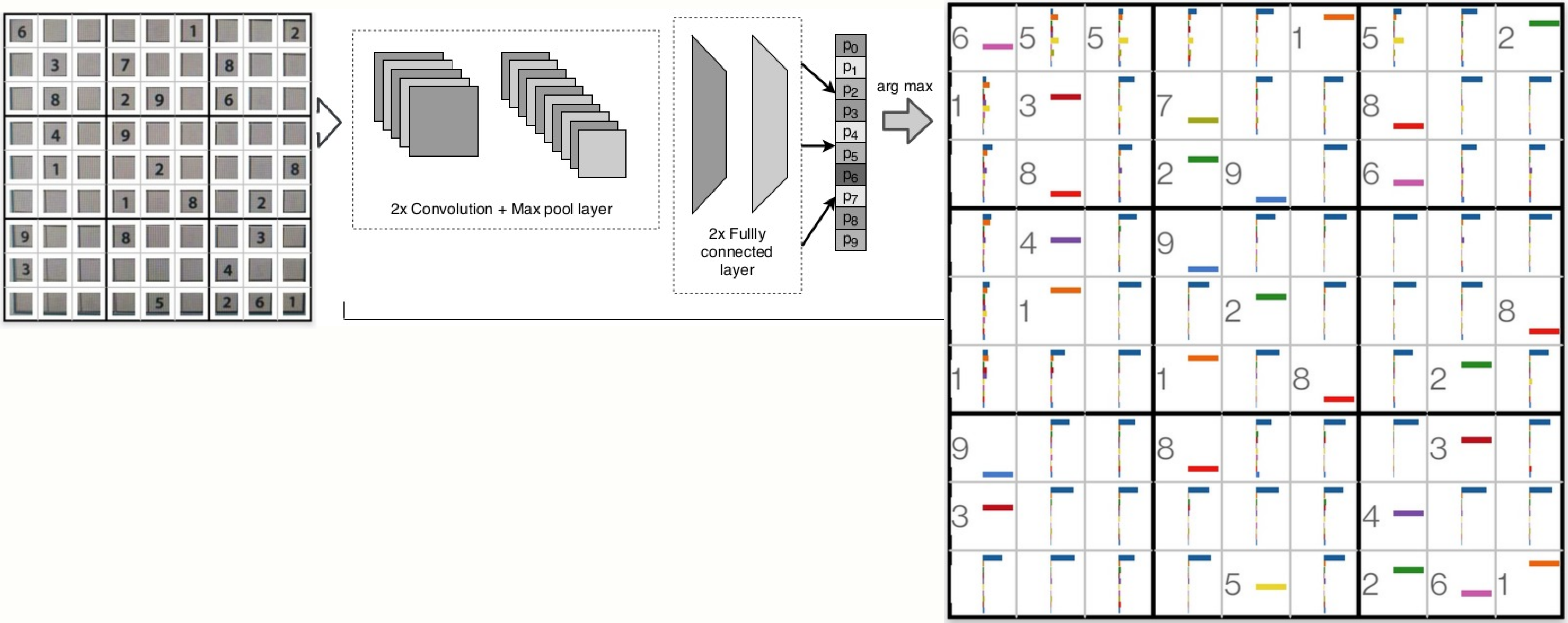
- Each cell predicts the maximum likelihood value:

$$\hat{y}_{ij} = \arg\max P(y_{ij} = k | X_{ij})$$

- But you need all 81 predictions (one for each given cell), it is a multi-output problem: together this is the 'maximum likelihood' interpretation

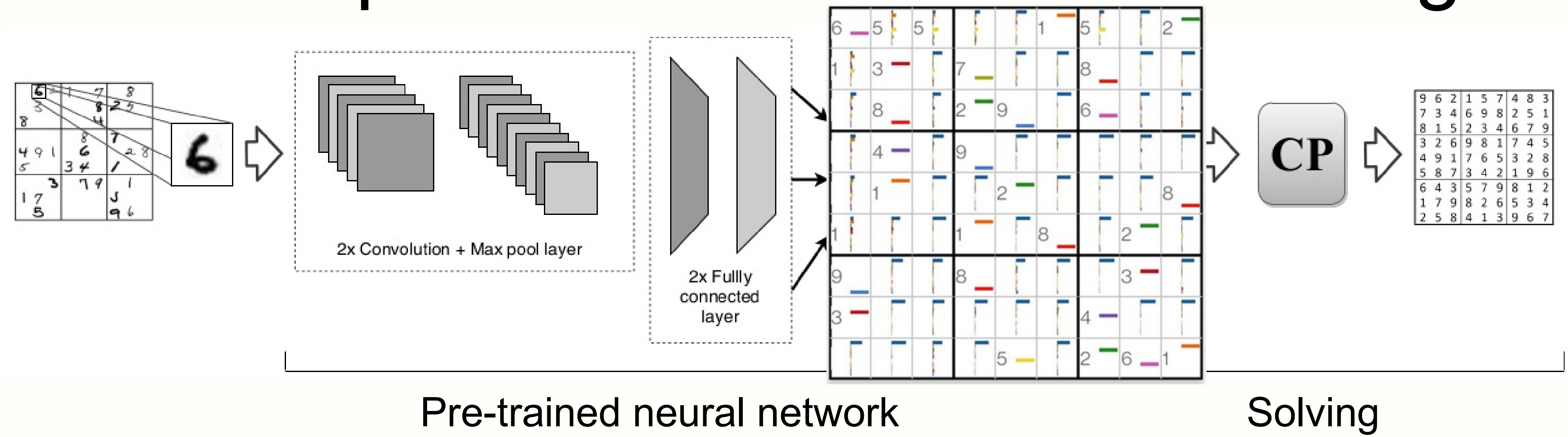
- If $\text{sudoku}(\hat{y})$ = *False*: no solution, interpretation is wrong...

Perception-based constraint solving



What about the *next* most likely interpretation?

Perception-based constraint solving



<u>What about the *next* most likely interpretation?</u>

- Treat prediction as *joint inference* problem:

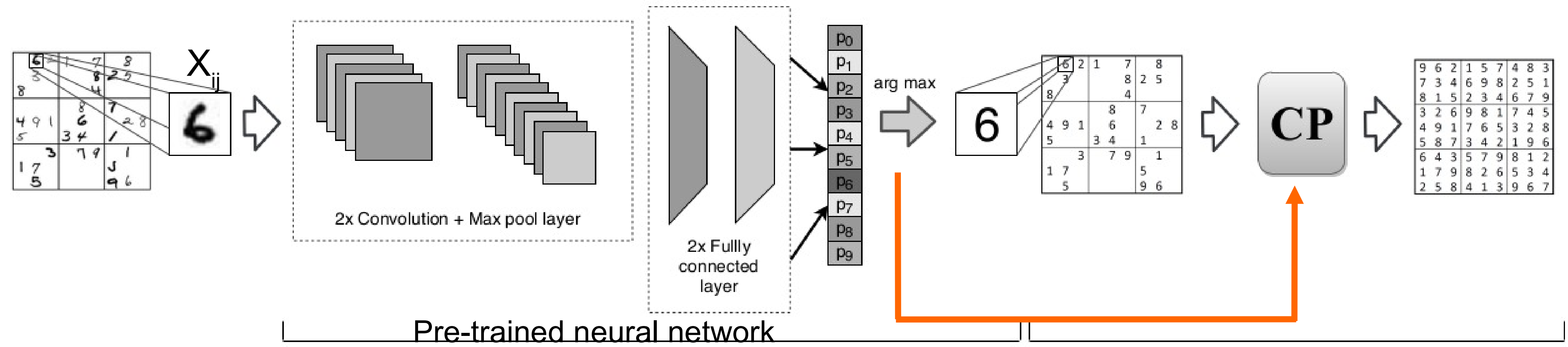
$$\hat{y} = \arg\max \prod_{ij} P(y_{ij} = k | X_{ij}) \quad \text{s.t.} \quad \text{sudoku}(\hat{y})$$

- This is the **constrained** ‘maximum likelihood’ interpretation

=> Structured output prediction

Used e.g. in NLP: [Punyakanok, COLING04]

Perception-based constraint solving



<u>Can we use a constraint solver for that?</u>

$$\hat{y} = \arg\max \prod_{ij} P(y_{ij} = k | X_{ij}) \quad \text{s.t.} \quad \text{sudoku}(\hat{y})$$

- Log-likelihood trick:

$$min \sum_{\substack{(i,j)\in \\ given}} \sum_{\substack{k\in \\ \{1,..,9\}}} \underbrace{-log(P_\theta(y_{ij} = k | X_{ij}))}_{constant} * \mathbb{1}[s_{ij} = k] \quad \text{s.t.} \quad \text{sudoku}(\hat{y})$$

Can do even better!

Are we using all available information?

A sudoku puzzle has to have one unique solution

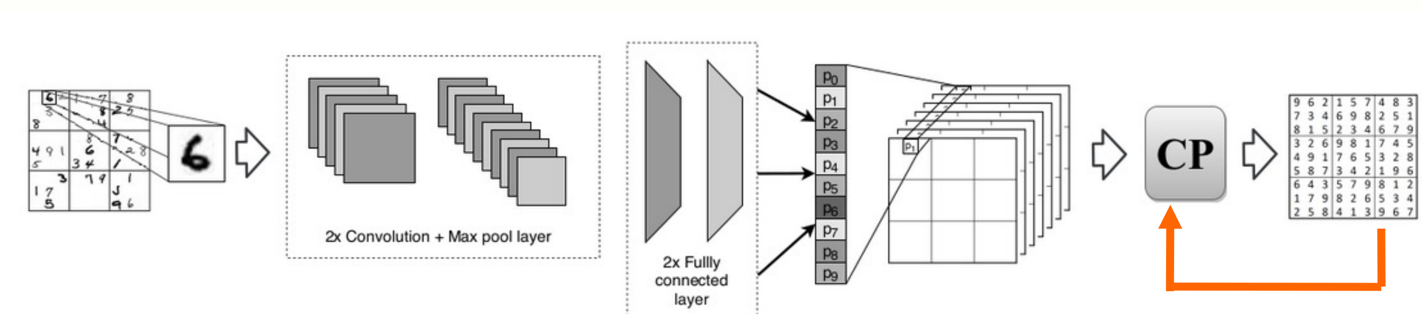
→ not in current constraint model: a 2nd order constraint

$$\begin{array}{ll} \underset{X}{\operatorname{argmin}} & f(X) \\ \text{subject to} & C(X) \\ & \nexists X' : X \neq X', C(X') \end{array}$$

But we can add cutting planes!

if the joint max likelihood image interpretation has multiple solutions:

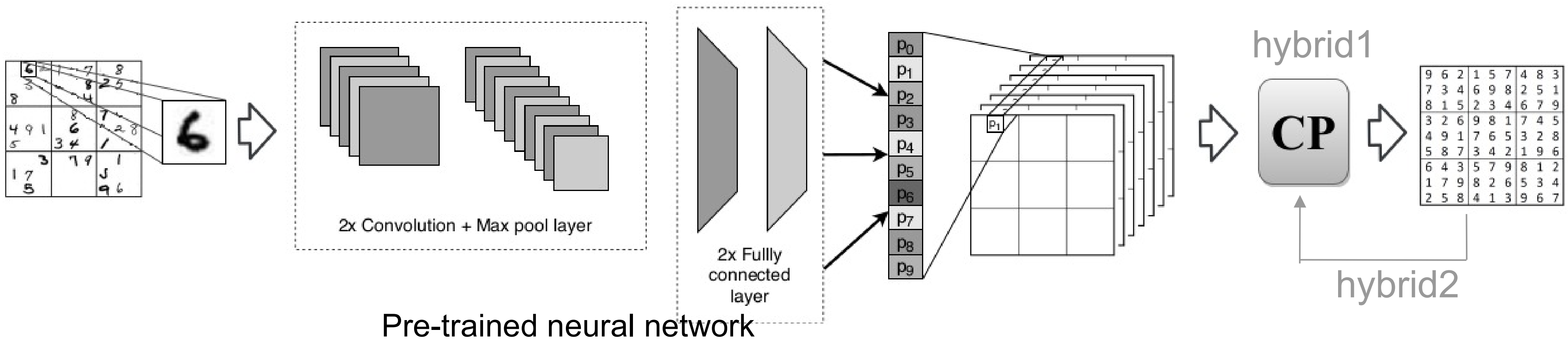
forbid (nogood/cutting plane) and find next most likely one!



Pre-trained neural network

Perception-based constraint solving

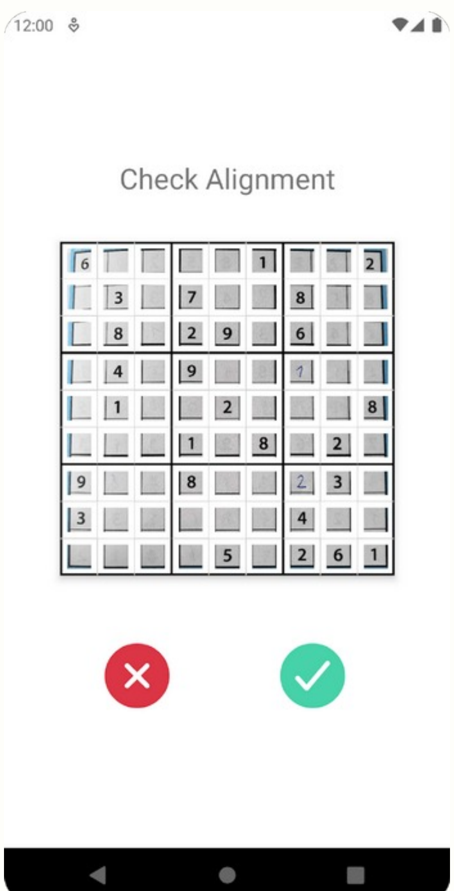
Hybrid: CP solver does *joint inference* over raw probabilities

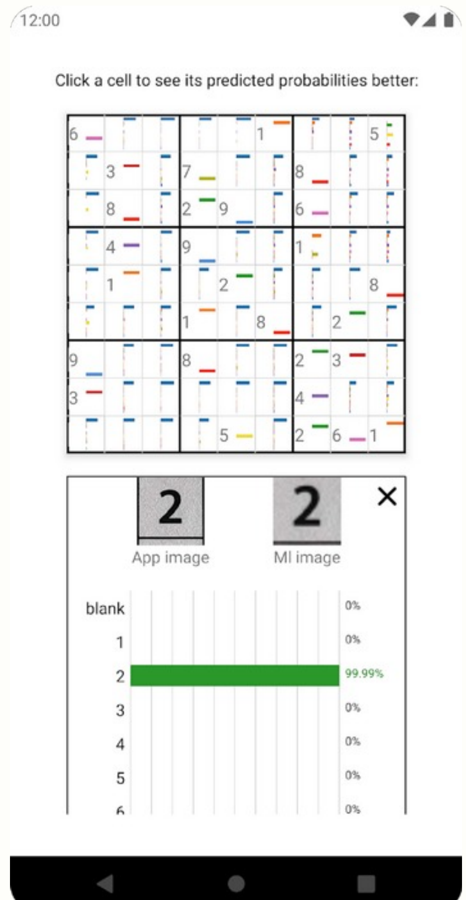


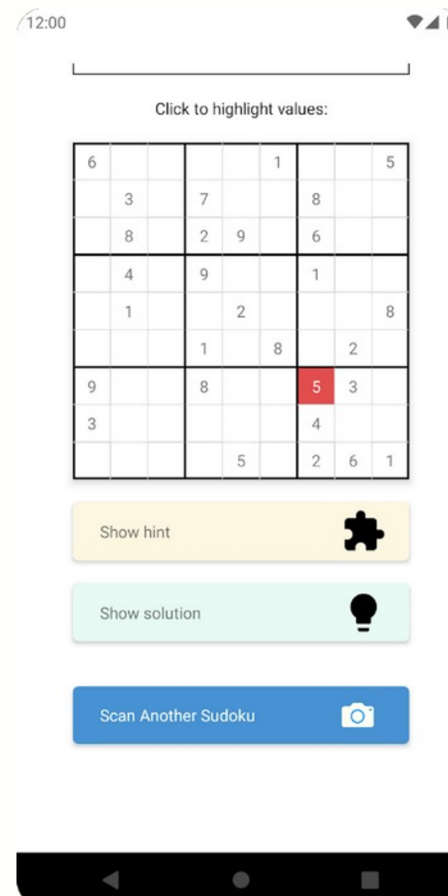
	accuracy img	accuracy cell	accuracy grid	failure rate grid	time average (s)
baseline	94.75%	15.51%	14.67%	84.43%	0.01
hybrid1	99.69%	99.38%	92.33%	0%	0.79
hybrid2	99.72%	99.44%	92.93%	0%	0.83

[Maxime Mulamba, Jayanta Mandi, Rocsildes Canoy, Tias Guns, CPAIOR20]

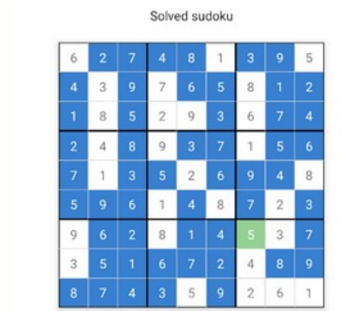
Sudoku Assistant demo, continued







Show solution?



Trivial for CP system (subsecond),

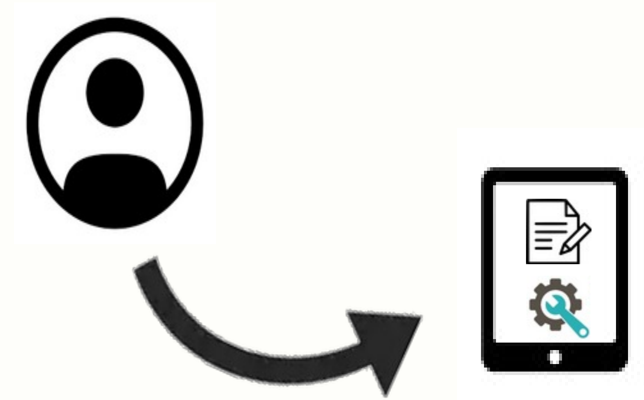
Boring and demotivating for user?

In general: human-aware AI &
AI assistants:

- *Support* users in decision making
- Respect human *agency*
- Provide *explanations* and learning opportunities

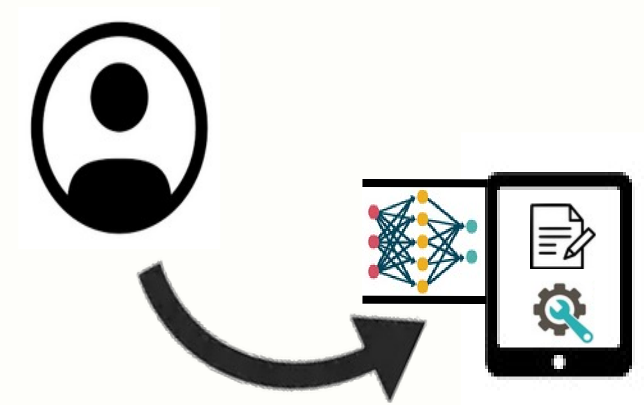
Constraint solving is more than mathematical abstractions...

Bigger picture



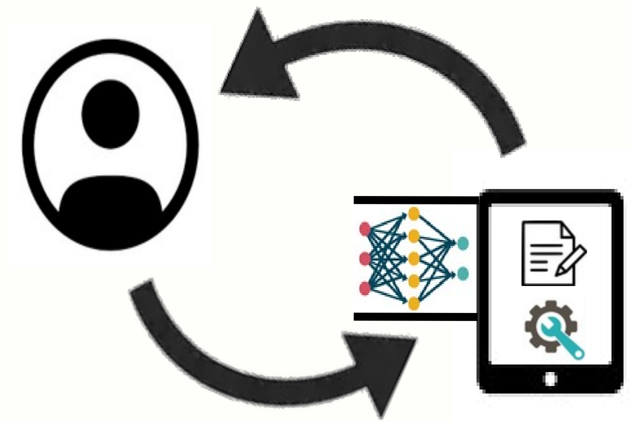
Bigger picture

- Learning implicit user preferences
- Learning from the environment



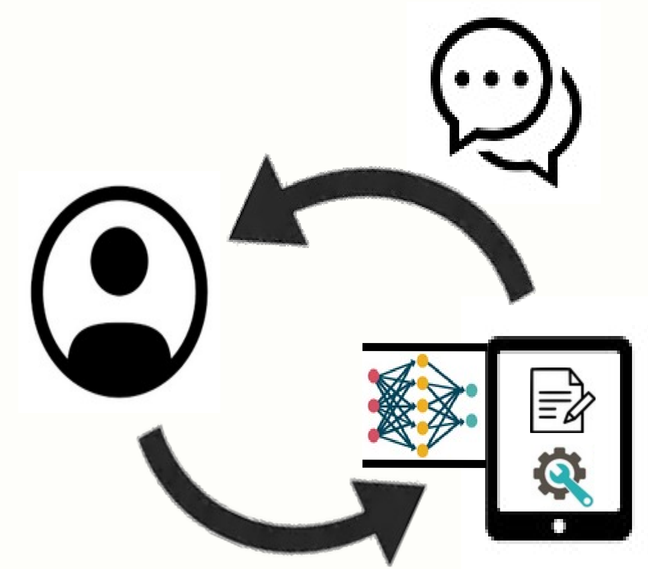
Bigger picture

- Learning implicit user preferences
- Learning from the environment
- Explaining constraint solving

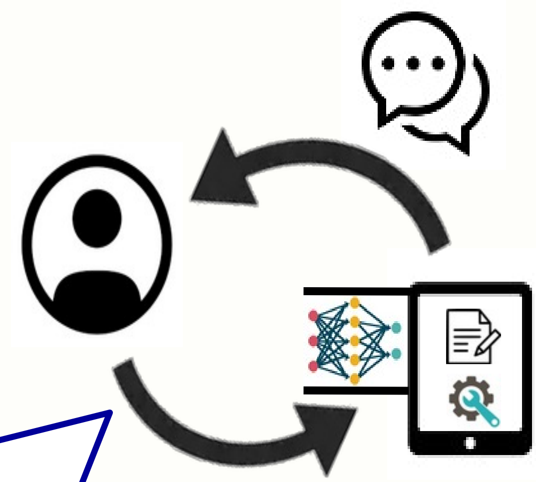


Bigger picture

- Learning implicit user preferences
- Learning from the environment
- Explaining constraint solving
- Stateful interaction



CHAT-Opt:
Conversational **H**uman-**A**ware **T**echnology for **Opt**imisation

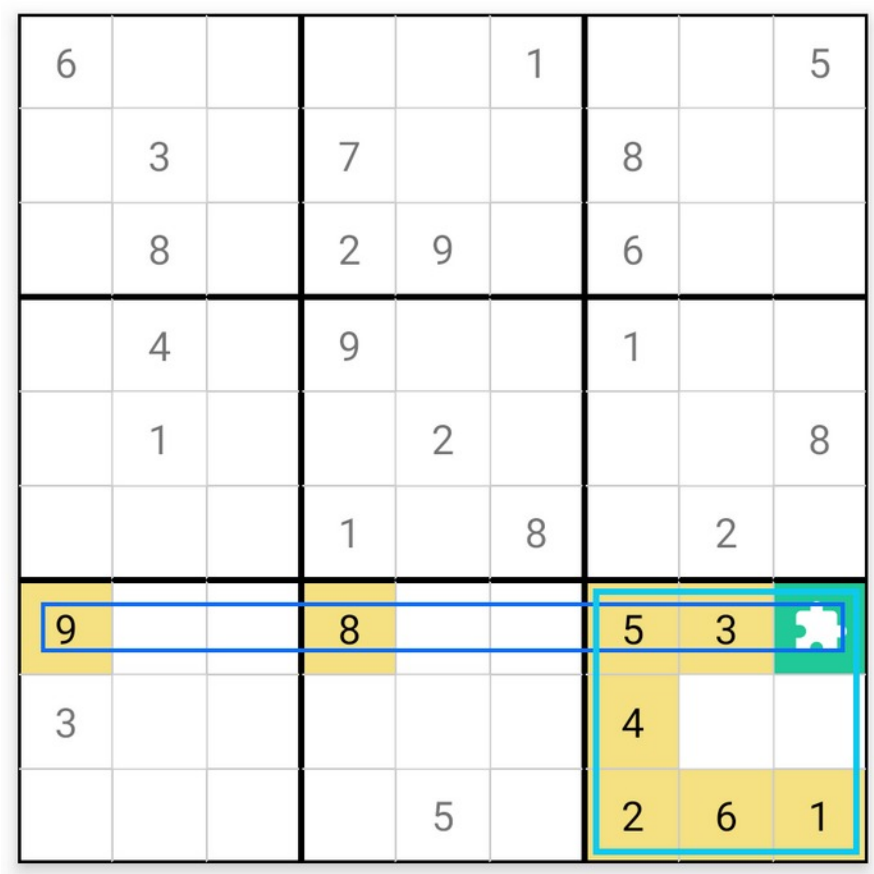


Towards **co-creation** of constraint optimisation solutions

- Solver that learns from user and environment
- Towards conversational: explanations and stateful interaction

https://people.cs.kuleuven.be/~tias.guns/chat-opt.html

Visitors welcome!



Stepwise Explanation for Constraint Satisfaction Problems

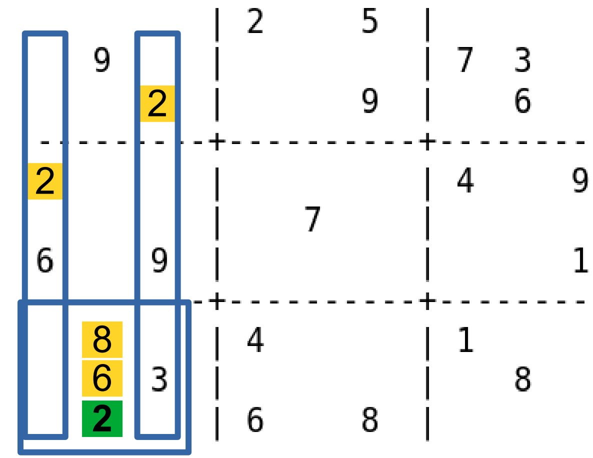
Bogaerts, Bart, Emilio Gamba, and Tias Guns. "A framework for step-wise explaining how to solve constraint satisfaction problems." *Artificial Intelligence* 300 (2021)

Help, I'm stuck:

```
      | 2   5 |
  9   |       | 7 3
    2 |     9 |   6
------+-------+------
2     |       | 4   9
      |   7   |
6   9 |       |     1
------+-------+------
  8   | 4     | 1
  6 3 |       |   8
      | 6   8 |
```

What would a solver do?

- User may not understand all derivations
- Or wants to learn from it



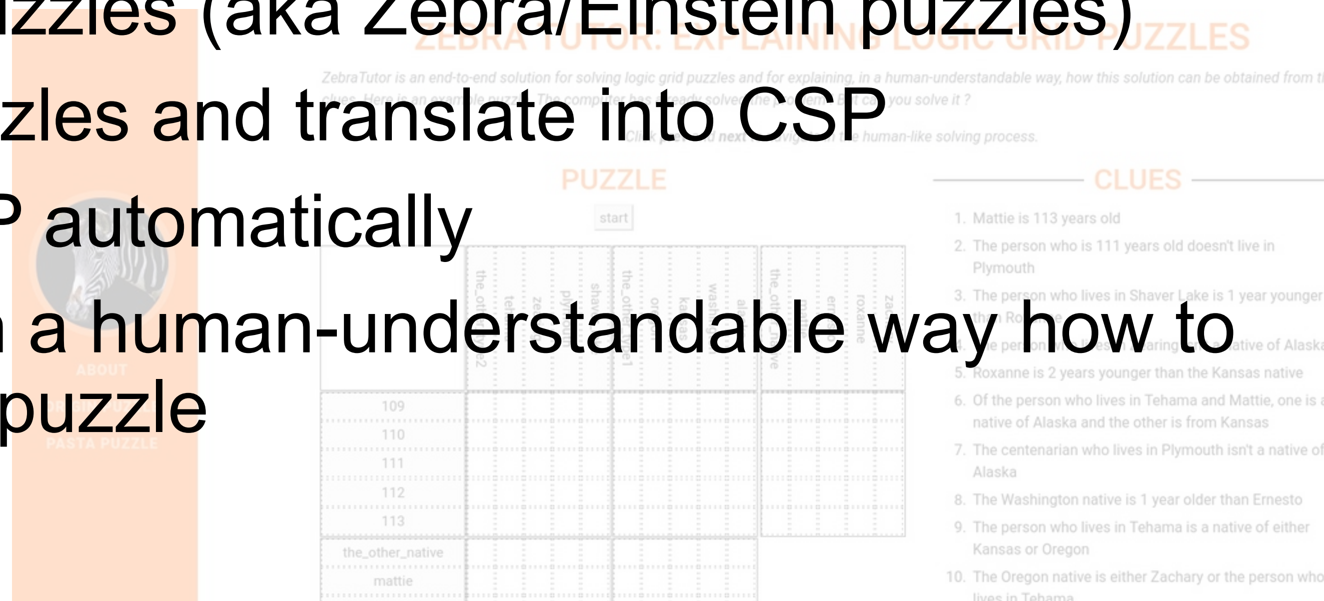
*"**Explain** in a <u>human-understandable</u> way how to solve constraint satisfaction problems"*

Explanations for a SAT problem

Ex. 2019 Holy Grail Challenge (E. Freuder)

Logic Grid Puzzles (aka Zebra/Einstein puzzles)

- Parse puzzles and translate into CSP
- Solve CSP automatically
- **Explain** in a human-understandable way how to solve this puzzle

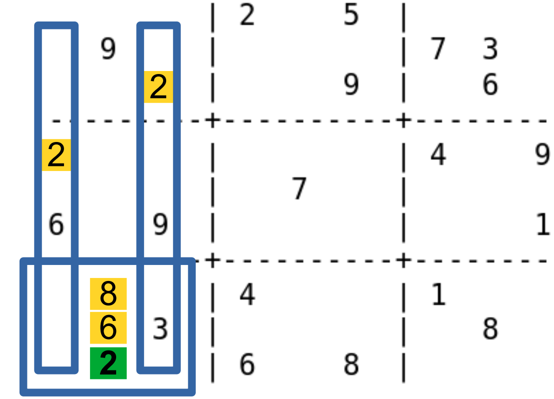


Explain 1 variable from maximal consequence

			2		5			
	9					7	3	
		2			9		6	
2						4		9
				7				
6		9						1
	8		4			1		
	6	3					8	
	2		6		8			

Explanation step

Let *E’* & *S’* => ***n*** be one explanation step.



E’ = a subset of previously derived facts E

(Sudoku) Given and derived digits in the grid

S’ = a minimal subset of constraints S such that E’ & S’ => n

(Sudoku) Alldifferent column, row, box constraints

n = a newly derived fact (from the solution)

How? MUS(¬ n & E & S) is a valid explanation step

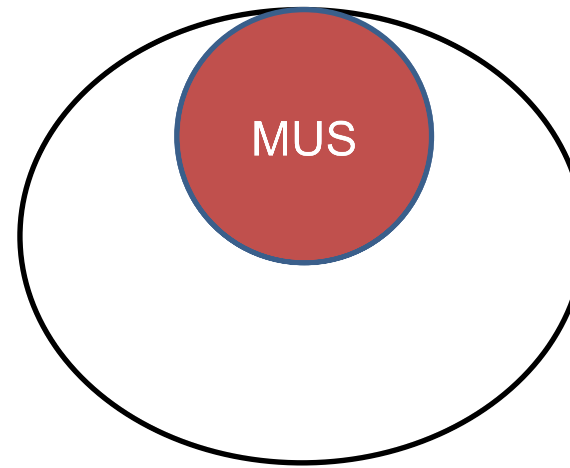
UNSAT set of constraints

= Need for an explanation of UNSAT

1. Identify conflicting constraints as explanation for UNSAT

   → Extract Minimum Unsatisfiable Subset (MUS)

   a.k.a Irreducible Inconsistent Subsystem (IIS)



Constraints

Explaining UNSAT with MUSes

Methods

1. Some solvers provide an implementation for extracting unsatisfiable cores as explanations of UNSAT.
2. **Deletion-based** Minimal unsatisfiable subsets
   - Iterate over constraints
   - Delete constraints if removing them leaves the model UNSAT

```
def mus(constraints):
    m = Model(constraints)
    assert ~m.solve(), "MUS: model must be UNSAT"

    core = m.get_core()  # or all constraints
    i = 0
    while i < len(core):
        subcore = core[:i] + core[i+1:]  # check if all but i makes core SAT

        if Model(subcore).solve():
            i += 1  # removing it makes it SAT, must keep
        else:
            core = subcore  # overwrite core, so core[i] is next one

    return core
```

1

2

Joao Marques-Silva. *Minimal Unsatisfiability: Models, Algorithms and Applications*. ISMVL 2010. pp. 9-14

Example of MUS extraction

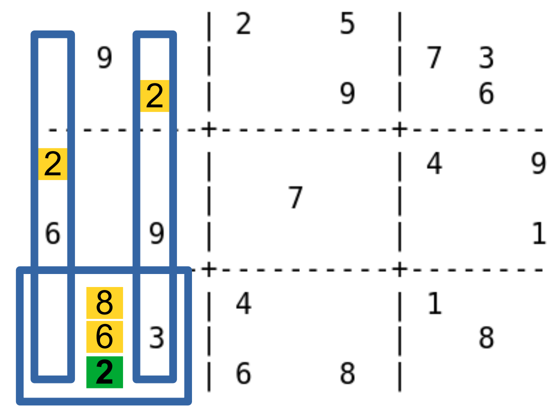
examples/tutorial_ijcai22/3_musx.ipynb

The best/easiest explanation step...

- Let *f(S)* be a *cost function* that quantifies how good (e.g. easy to understand)
- an explanation step is.

Simple MUS-based algo:

```
sol-to-explain = propagate( E & S ) \ E

X_best = None
for n in sol-to-explain:
    X = MUS(~n & E & S )
    if f(X) < f(X_best):
        X_best = X
return X_best
```



MUS gives no guarantees on quality, only subset minimal (SMUS)

The best/easiest explanation step...

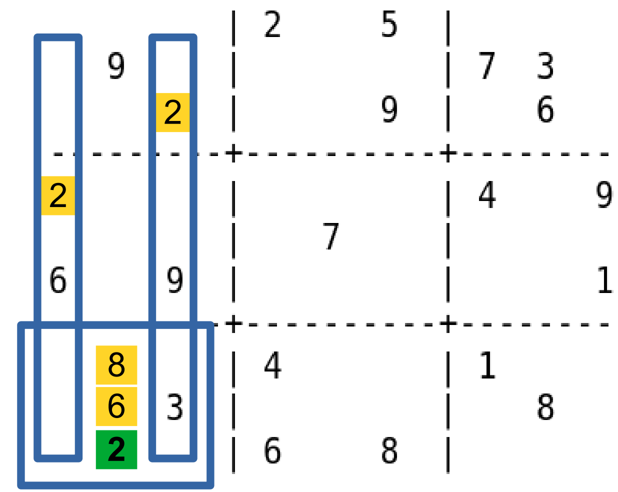
- Let *f(S)* be a *cost function* that quantifies how good (e.g. easy to understand)
- an explanation step is.

Explain 1 step with OCUS

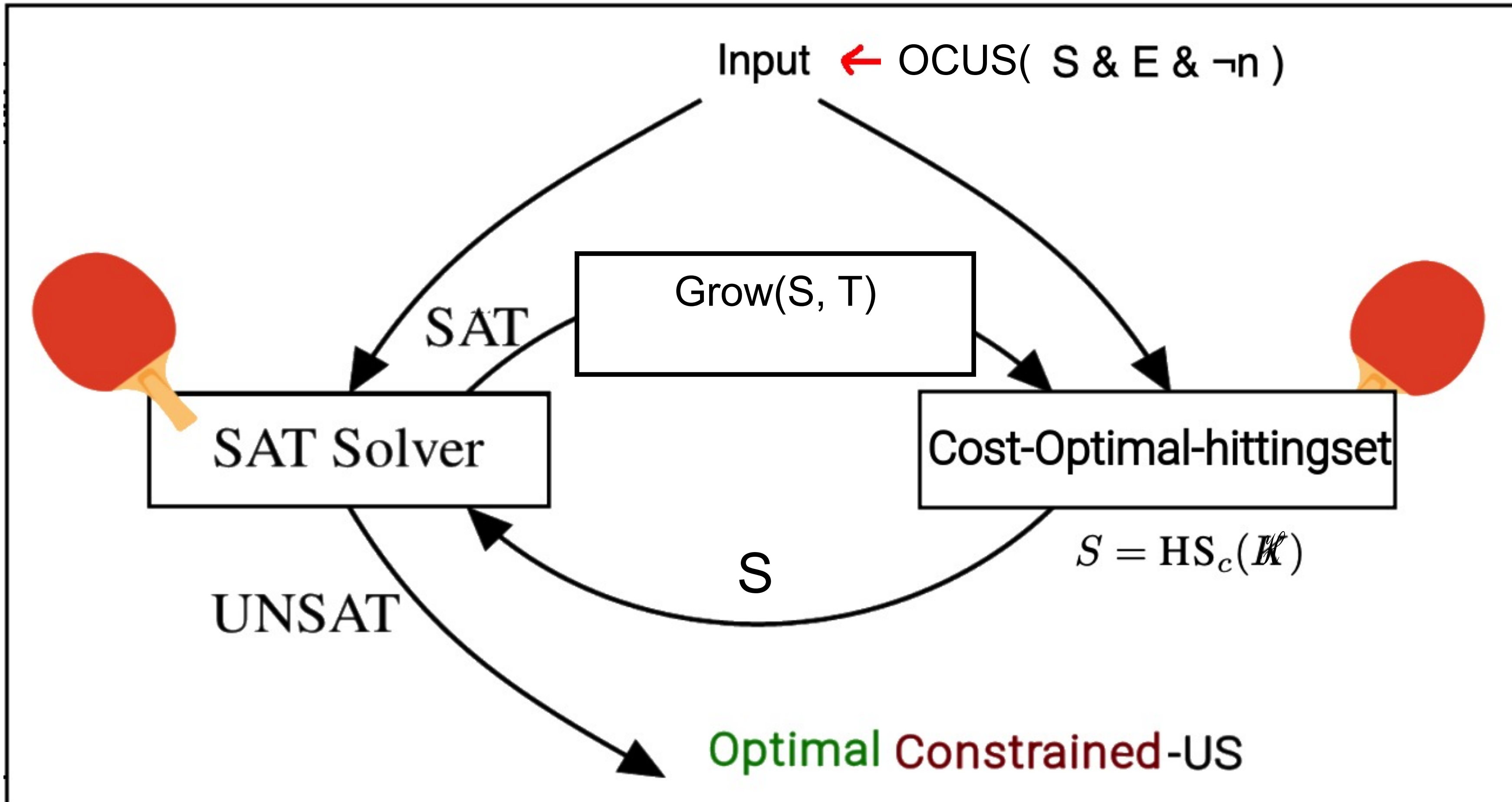
sol-to-explain = `propagate(E & S) \ E`

`c` = exactly-one(`{~n|n∈` *sol-to-explain*`}`),

`return` OCUS(`n|n ∈`sol-to-explain`} & S & E &{~,`**f**`, c)`

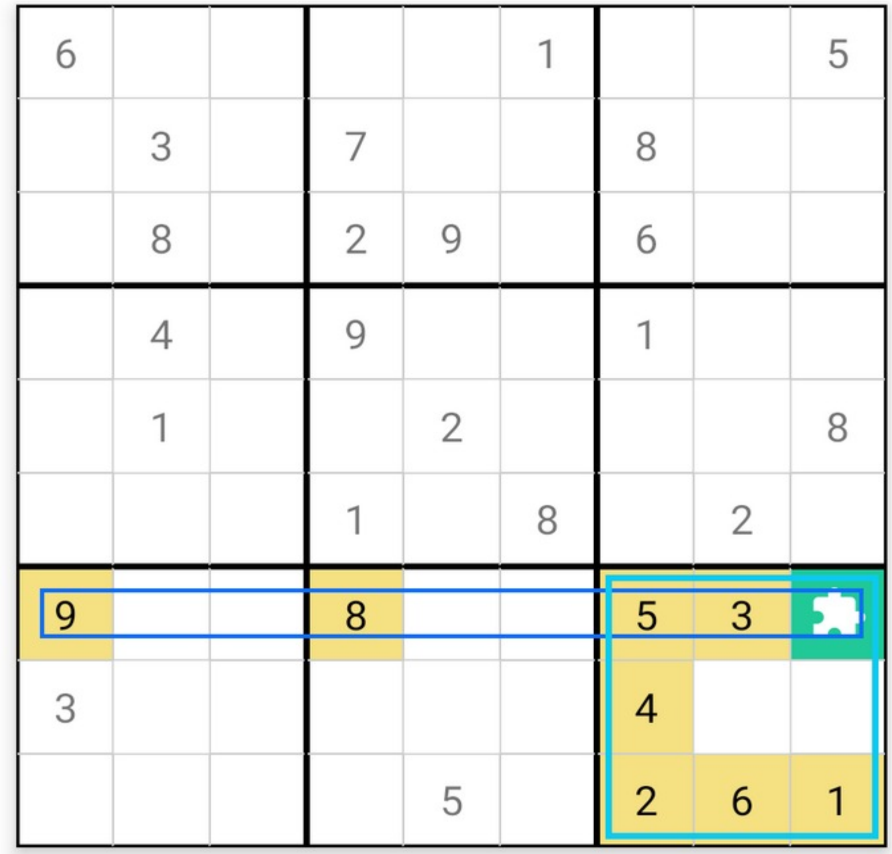


Implicit hitting-set algorithm



OUS extraction

[*examples/tutorial_ijcai22/5_ocus_explanations.ipynb*](examples/tutorial_ijcai22/5_ocus_explanations.ipynb)



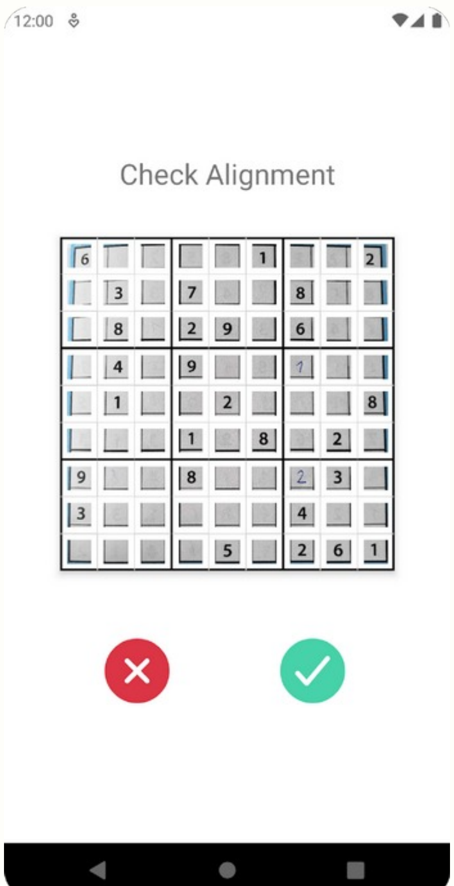
Stepwise Explanation for Constraint Satisfaction Problems

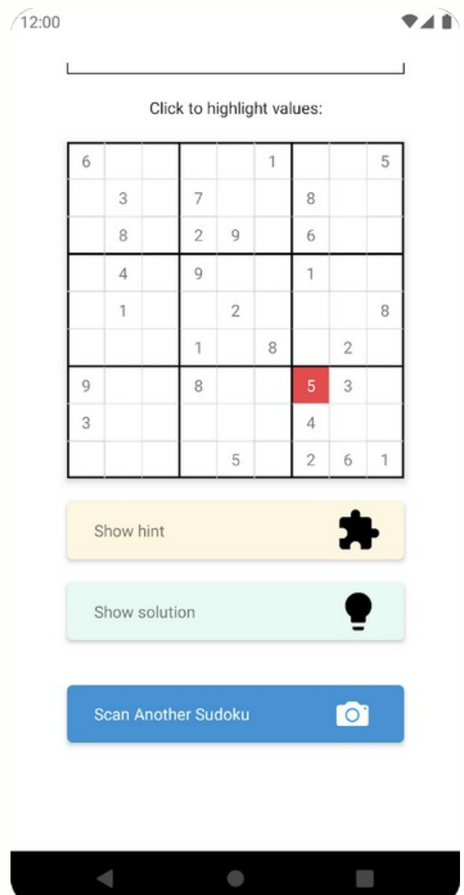
Intelligible hints:

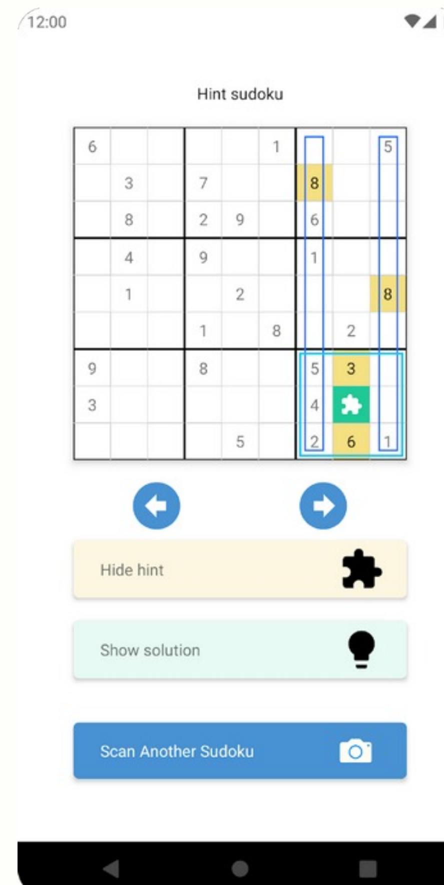
- The Constraint Solver searches for the **Optimal Unsatisfiable Subset** (OUS) for the negation of each value to be assigned.
- Computing this over all empty cells is **computationally challenging**.
- A cost function estimates the complexity of each subset, which allows the app to provide the **easiest** one at each step

Gamba, Emilio. "Efficiently Explaining CSPs with Unsatisfiable Subset Optimization." IJCAI. 2021

Sudoku Assistant demo, continued







The changing role of solvers

Holy Grail: user specifies, solver solves [Freuder,1997]

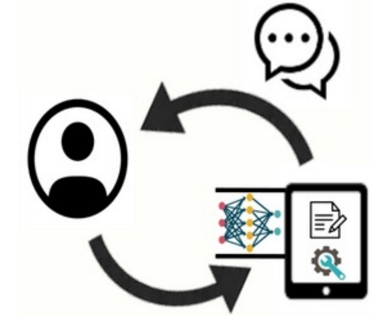
I think we reached it… MiniZinc, Essence

- "Beyond NP" → Constraint Solver as an **oracle**

- Use CP solver to solve subproblem of larger algorithm
- Iteratively build-up and solve a problem until failure
- Integrate neural network predictions (structured output prediction)
- Generate proofs, explanations, or counterfactual examples, ...

[Freuder,1997] Freuder, Eugene C. "In pursuit of the holy grail." *Constraints* 2.1 (1997): 57-61.

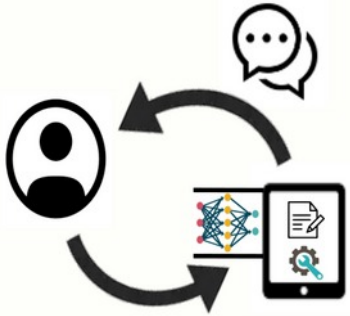
Integrated solving



What would the ideal Constraint Solving system be?

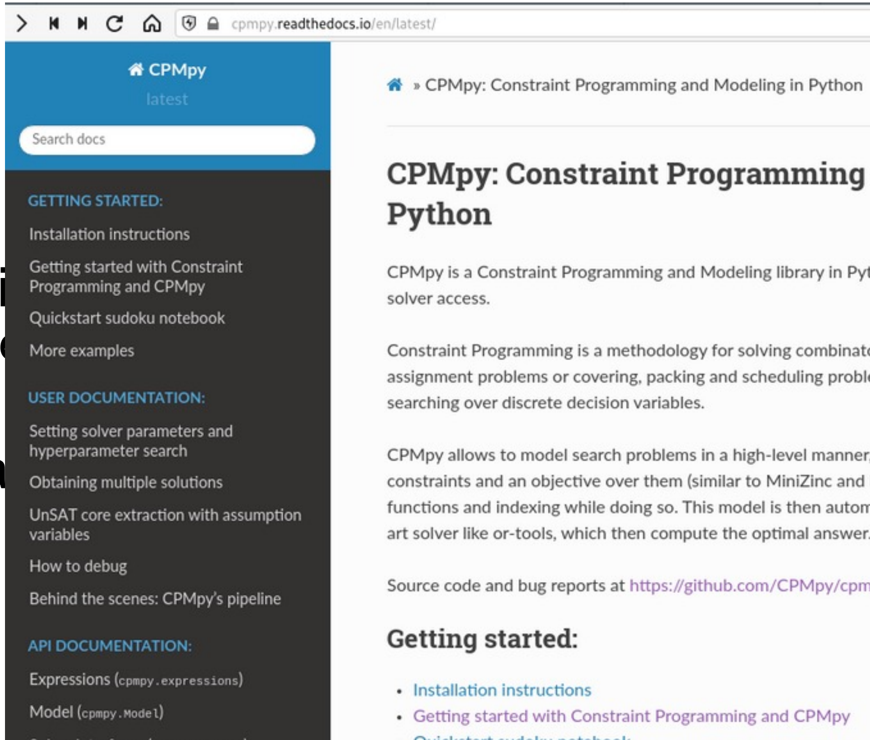
- Efficient repeated solving
  => Incremental
- Use CP/SAT/MIP or any combination
  => solver independent and multi-solver
- Easy integration with Machine Learning libraries
  => Python and numpy arrays

Conversational **H**uman-**A**ware **T**echnology for **Opt**imisation

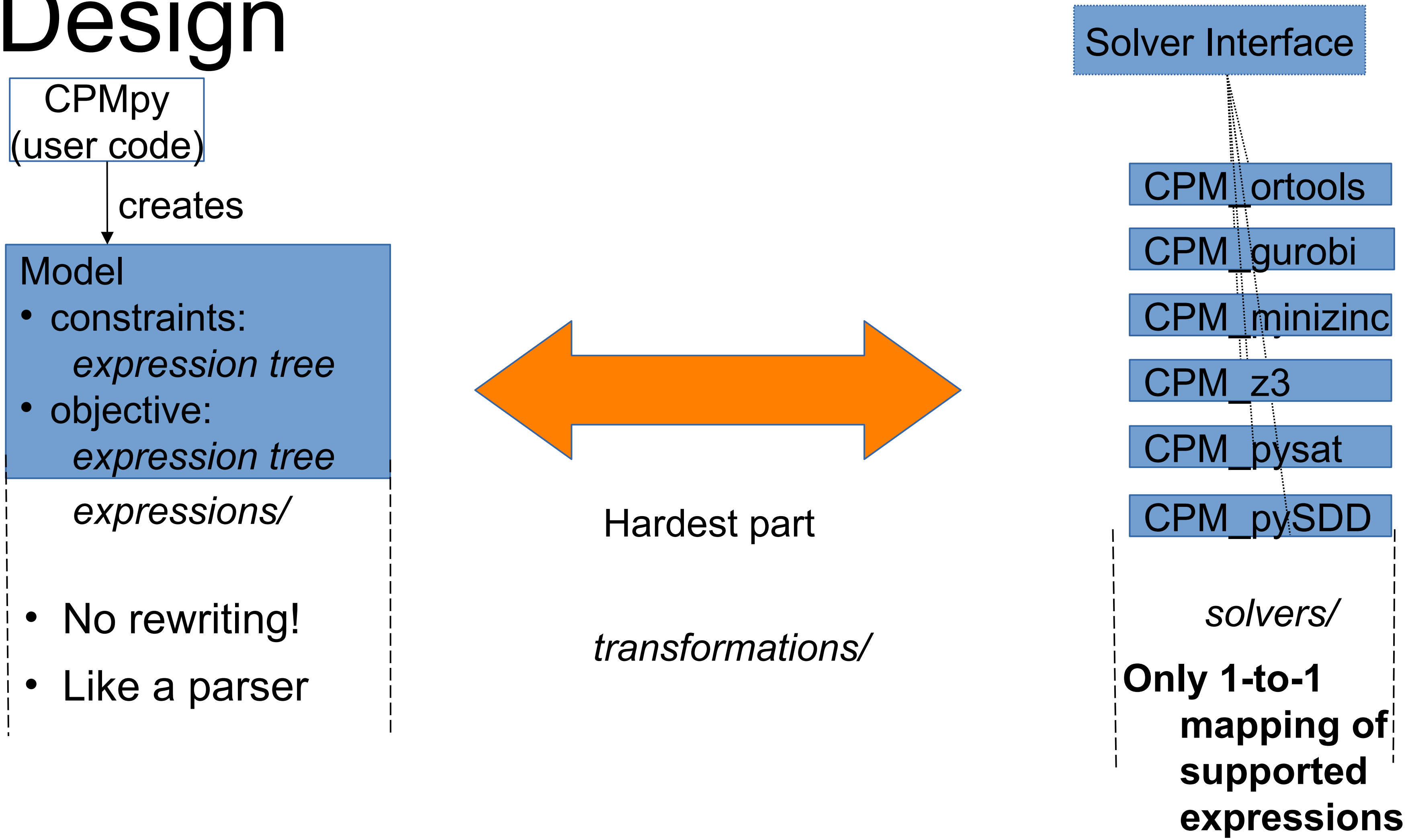


What would the ideal Constraint Solving system be?

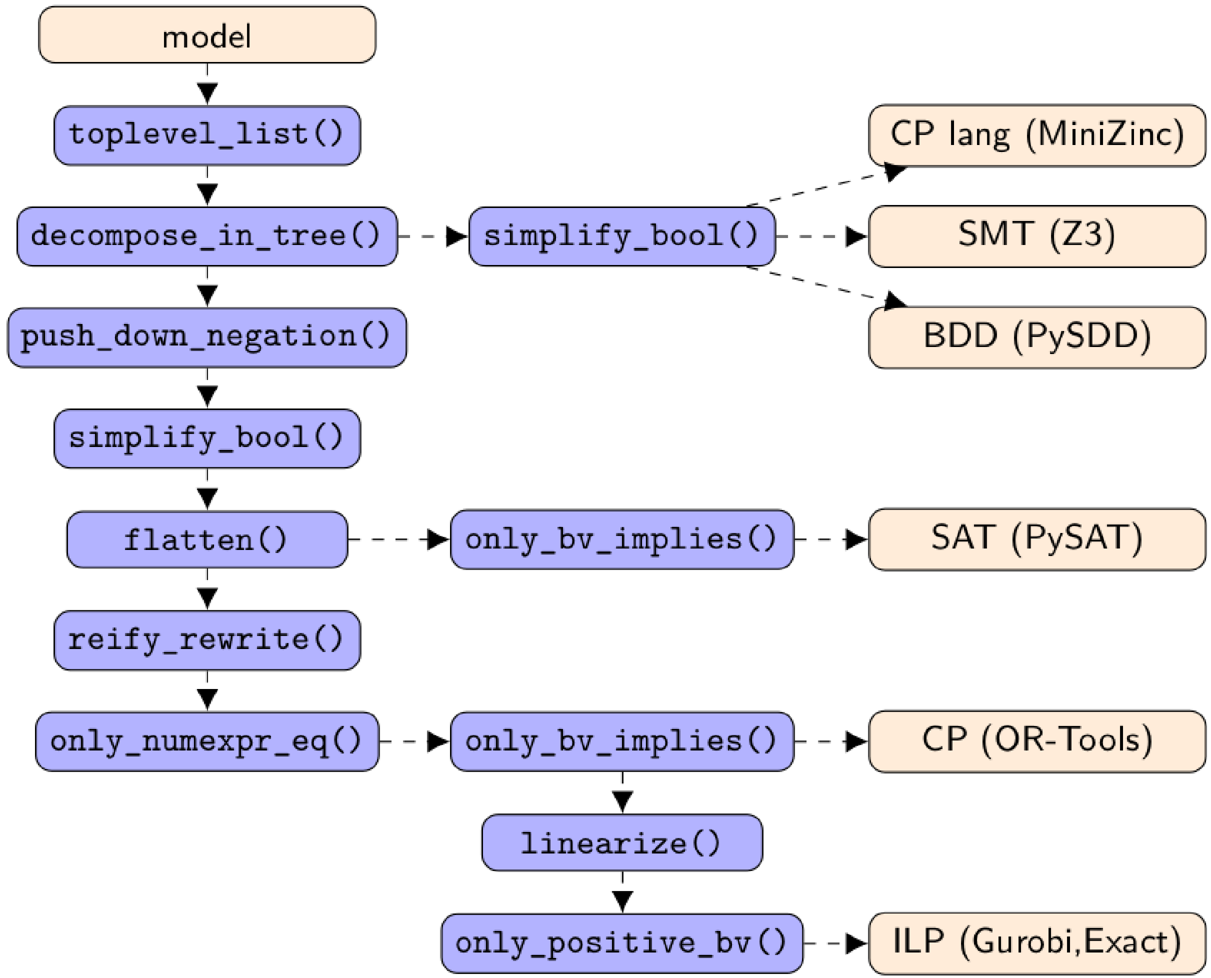
- Efficient repeated solving
  - => Incremental
- Use CP/SAT/MIP or any combinati
  => solver independent and multi-solv
- Easy integration with Machine Lea
  => Python and numpy arrays



Design


CPMpy (user code)
creates
Model
• constraints: expression tree
• objective: expression tree
expressions/
• No rewriting!
• Like a parser
Hardest part
transformations/
Solver Interface
CPM_ortools
CPM_gurobi
CPM_minizinc
CPM_z3
CPM_pysat
CPM_pySDD
solvers/
Only 1-to-1 mapping of supported expressions

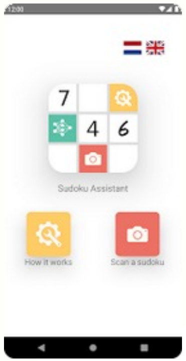
Transformations (overview)



Implementation: integration

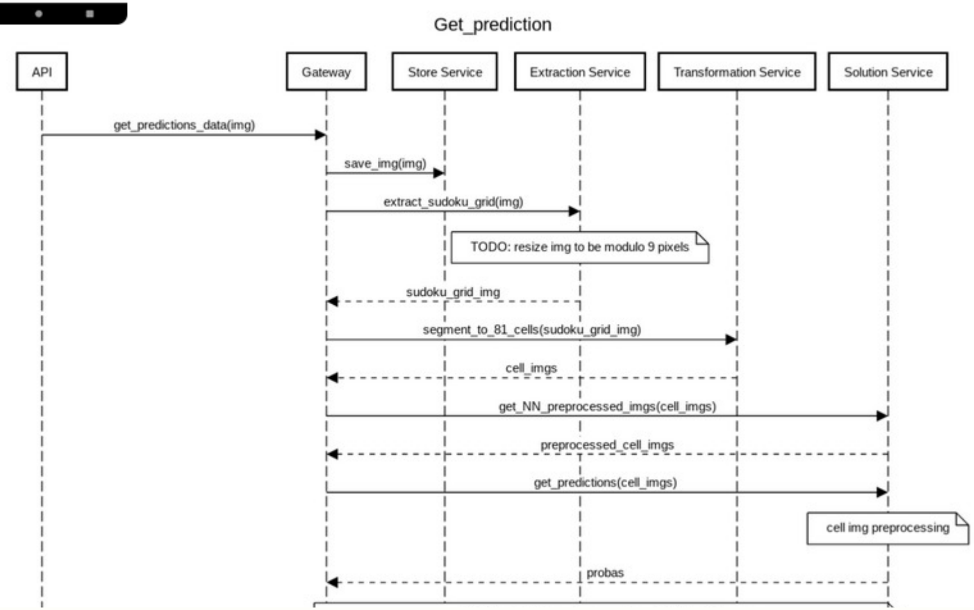
Frontend:

- React-native
- Only displays results

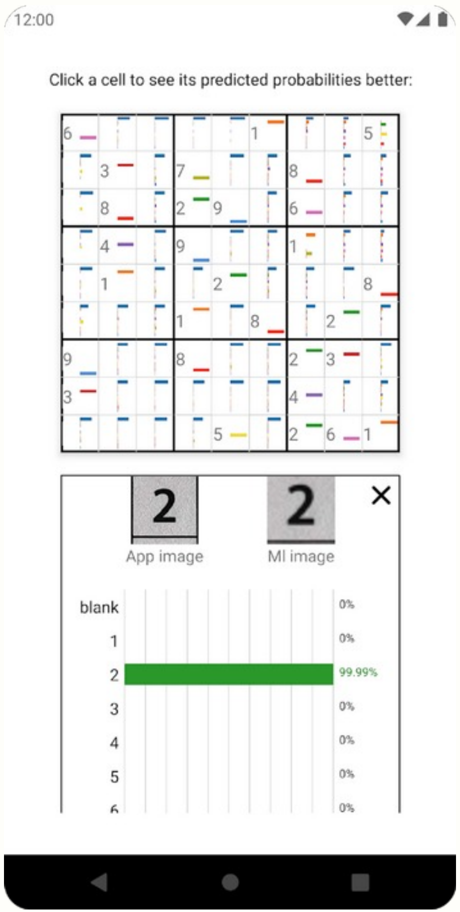


Backend:

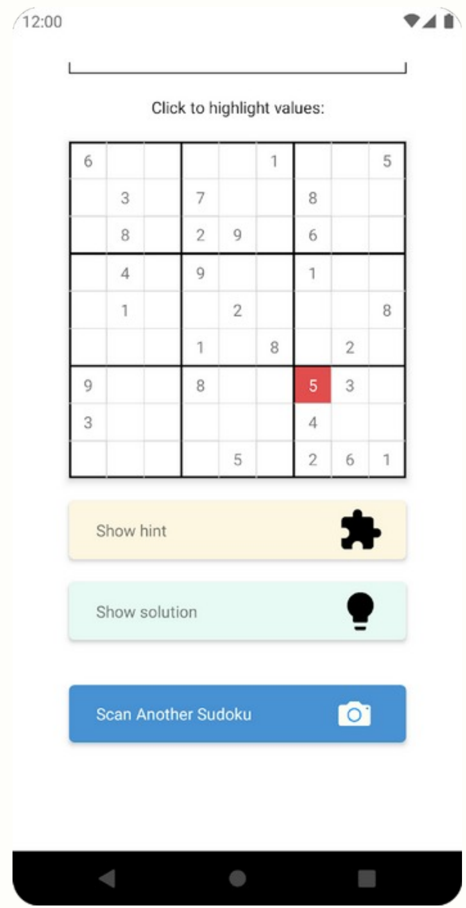
- FastAPI (Python)
- NN Service (PyTorch)
- Solver Service (CPMpy)
- Preloading, caching, hyperparameter optimisation...



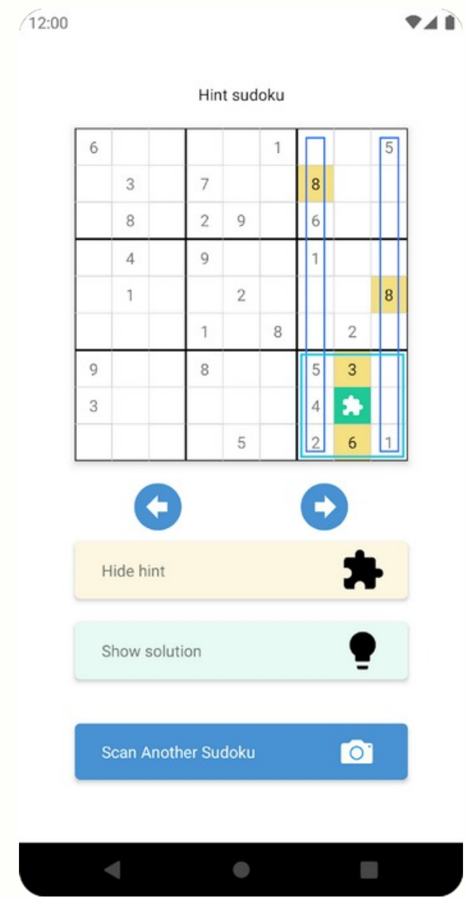
Responsiveness?



Avg ~0.1 s



Avg ~1.6 s (dev 3.2s)



Avg ~0.9 s (dev 1.2s)

Algorithm Configuration

Motivation

Constraint solvers support many hyper-parameters:

- settings for heuristics, pre-solve parameters...

Assuming similar parameters work well across instances of similar problems,

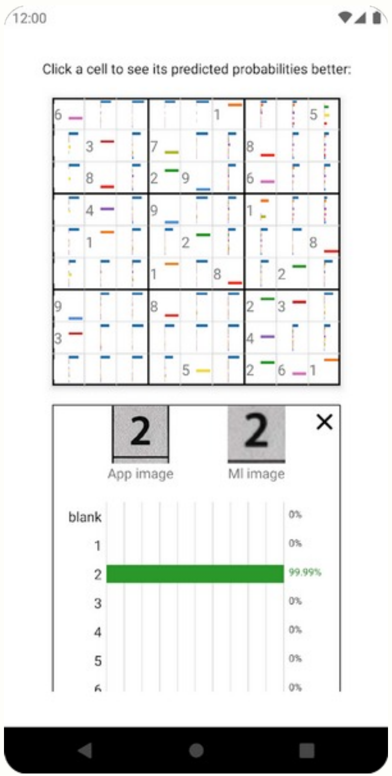
- Tune constraint solver on one instance and re-use configuration

Very easy to do in CPMpy because of direct solver access (checkout our examples!)

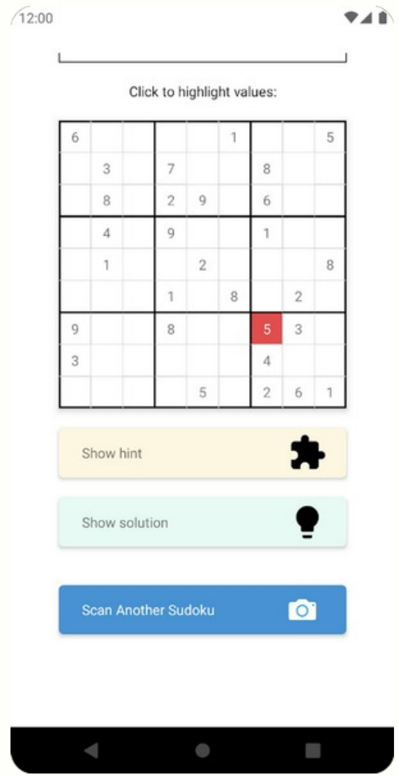
```
model.solve(
    cp_model_probing_level = 2,
    preferred_variable_orde = 1
    symmetry_level = 2
    search_branching = 5,
    use_erwa_heuristic = True
)
```

Naive approach: full grid search on entire hyper-parameter space

Responsiveness?

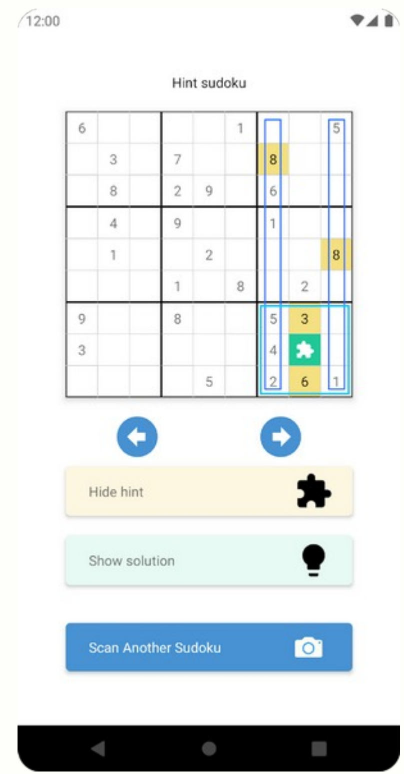


Avg ~0.1 s



Avg ~1.6 s (dev 3.2s)

NOT TUNED



Avg ~0.9 s (dev 1.2s)

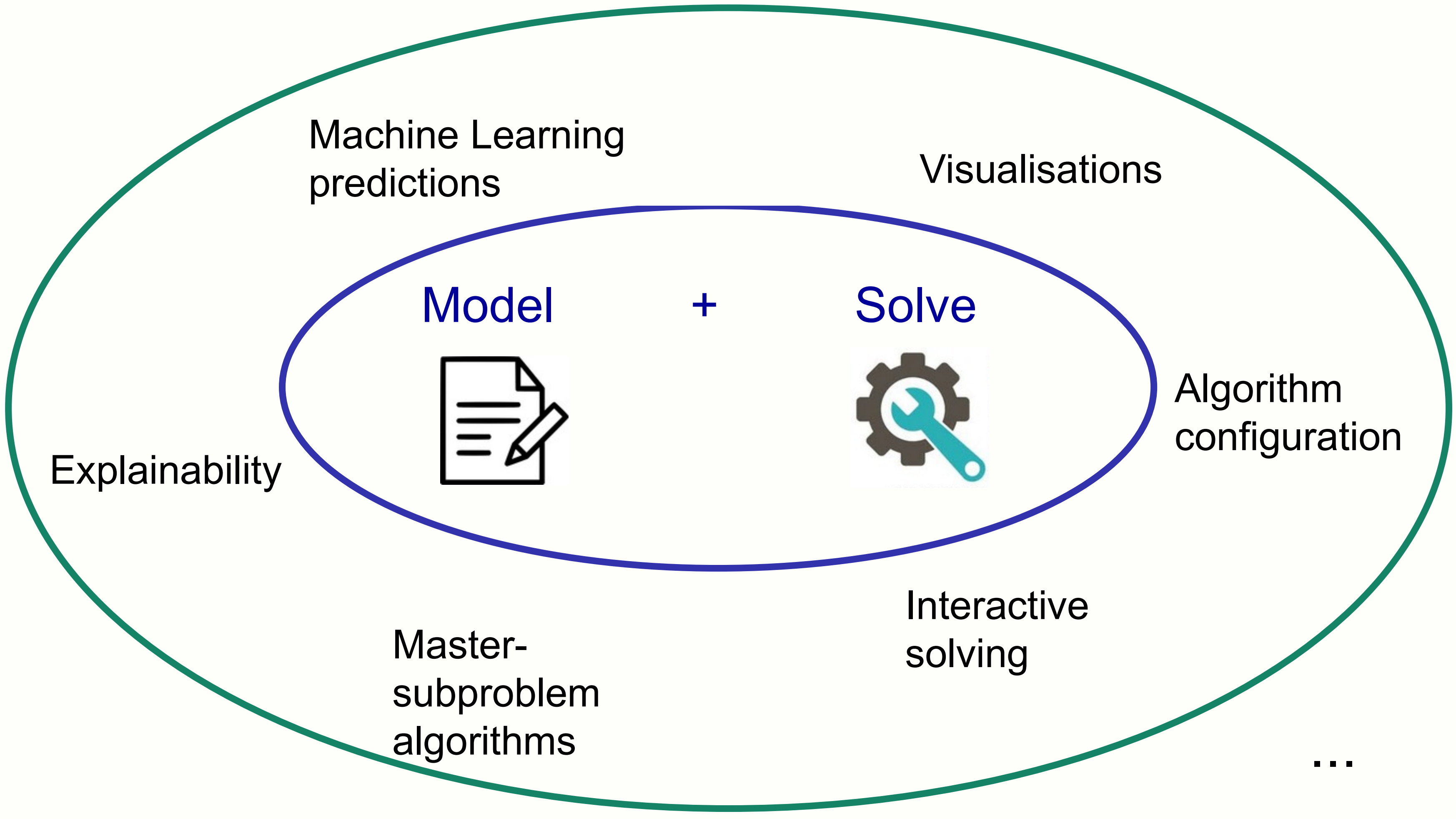
TUNED
(was much more)

Other relevant topics:

- Can we integrate instance-specific algorithm configuration?
- When to use which solver/transformations?
- Can we learn explanation preferences?
- Can we learn the constraints from data?
- Can we train an ML model based on the quality after solving (decision-focussed learning)?
- Can we explain across the CP & ML model?
- ...

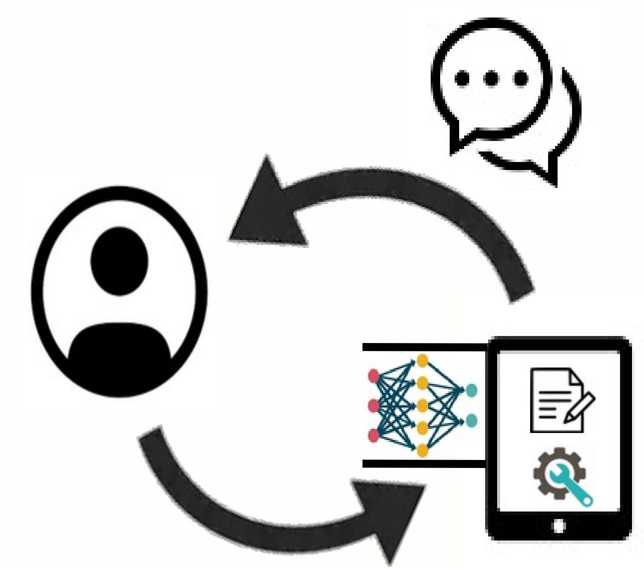
Conclusion

Wider view: integration

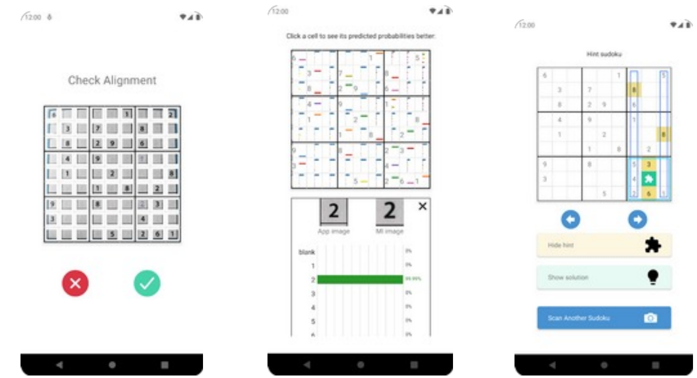


Bigger picture

- Learning implicit user preferences
- Learning from the environment
- Explaining constraint solving
- Stateful interaction



Sudoku Assistant
as integration example



<u>Needed all of:</u>

- Easy integration with Machine Learning libraries
  => Python and numpy arrays
- Efficient repeated solving
  => Incremental
- Use CP/SAT/MIP or any combination
  => solver independent and multi-solver
- Also parameter tuning, visualisations, web service deployment, etc