Learning and Reasoning with Constraint Solving

Prof. Tias Guns
<tias.guns@kuleuven.be>
@TiasGuns

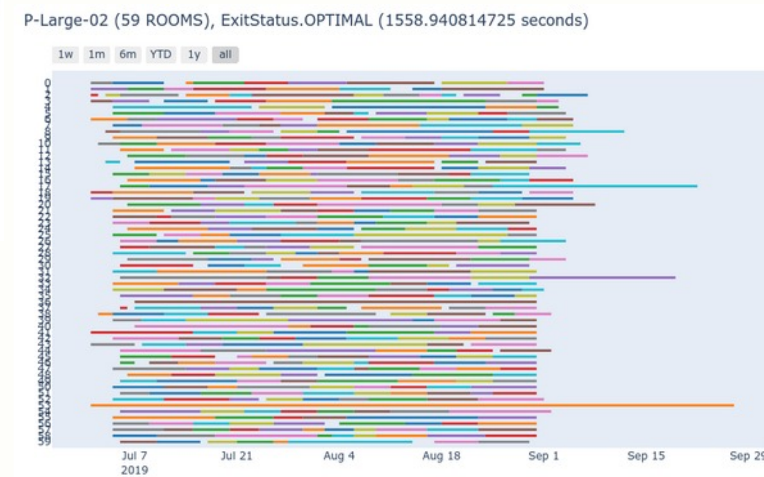
Joint work with team members:
- Rocs Canoy
- Jayanta Mandi
- Maxime Mulamba
- Victor Bucarey Lopez
- Emilio Gamba
- Ignace Bleukx

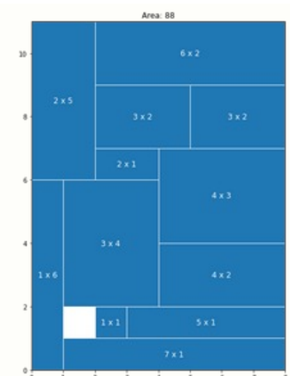
And external colaborators:
- Michelangelo Diligenti (Sienna Uni, It)
- Michele Lombardi (Bologna, It)
- Bart Bogaerts (VUB, Be)

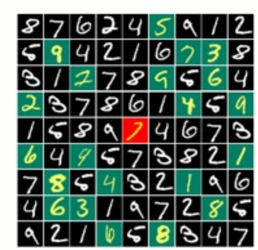
Constraint solving

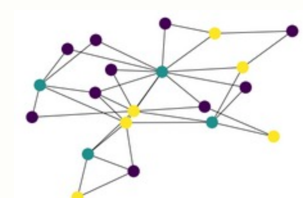
"Solving combinatorial optimisation problems"

- Vehicle Routing
- Scheduling
- Packing
- Other combinatorial problems









[Solved and visualized with the CPMpy constraint solving library]

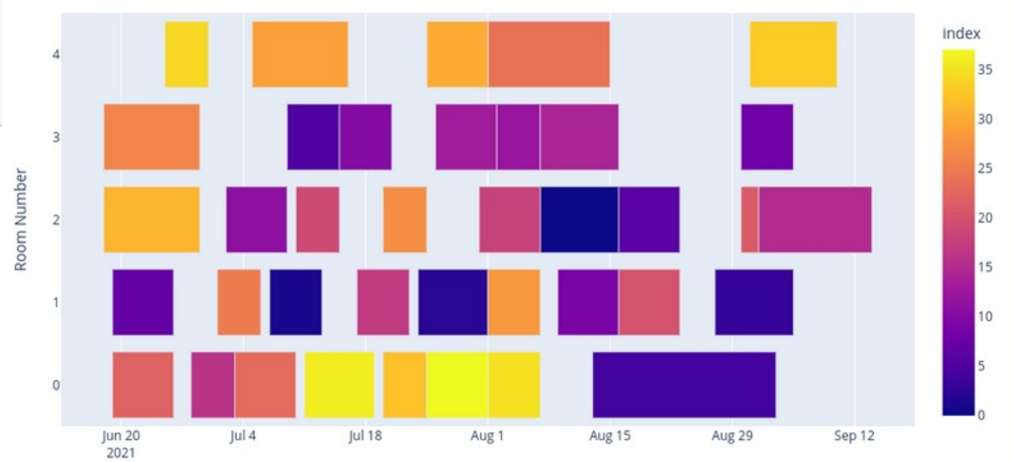
Example: room scheduling

Demo with CPMpy

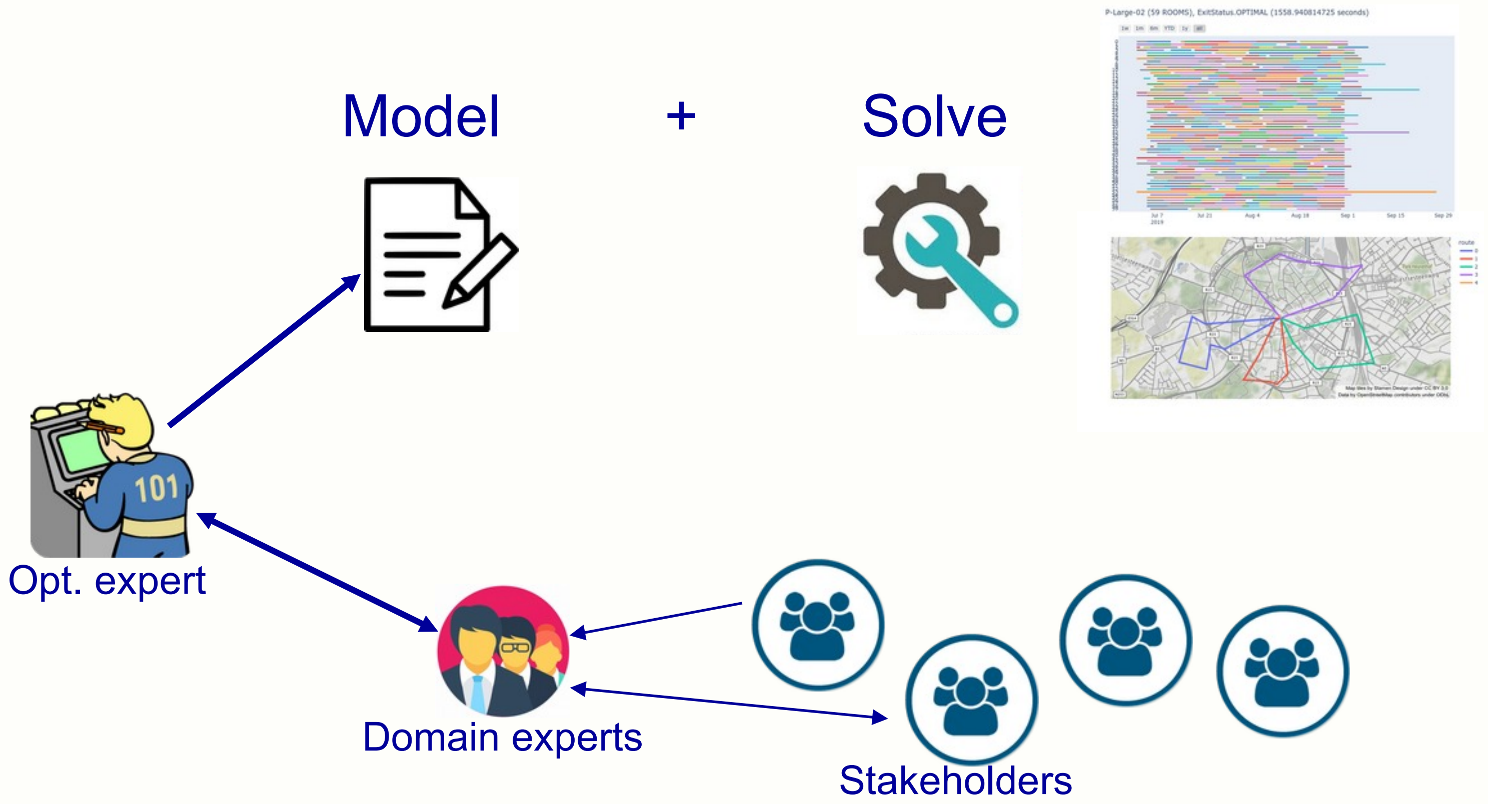
https://github.com/CPMpy/cpmpy/blob/master/examples/room_assignment.ipynb

Example: room scheduling (backup slide)

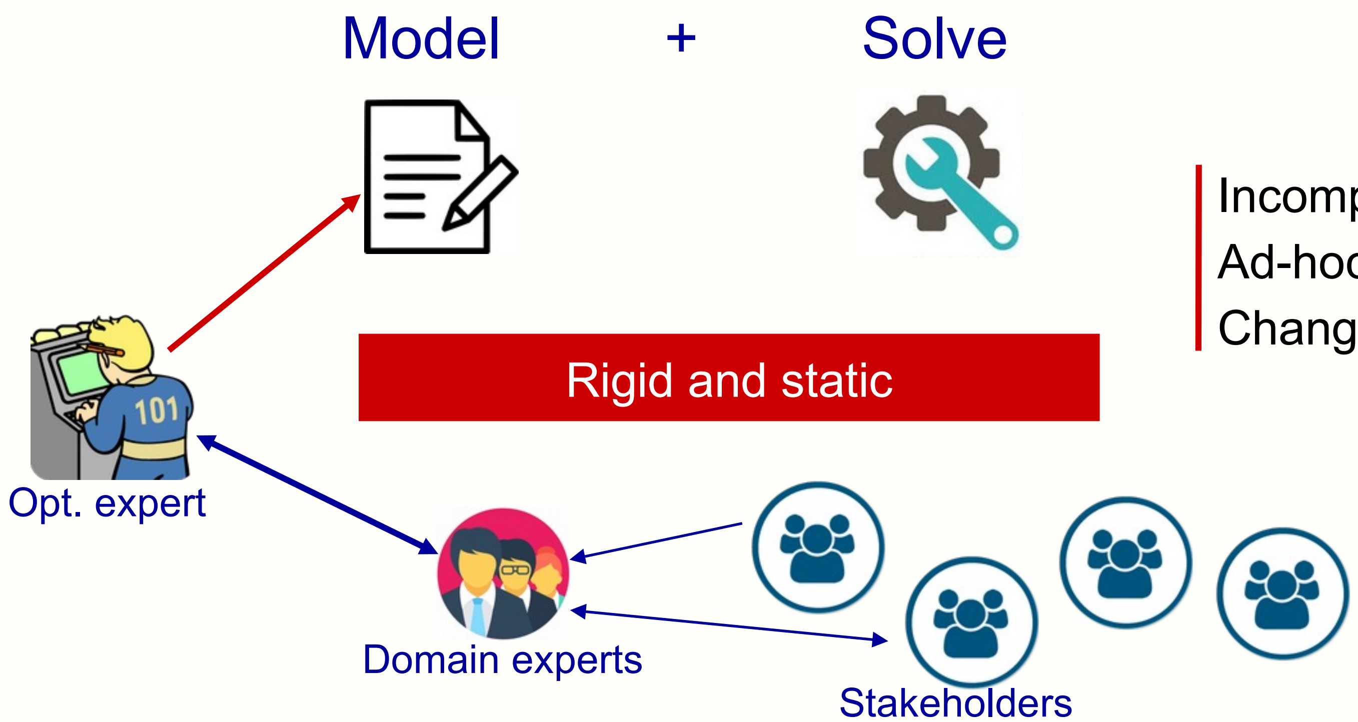
```python
def model_rooms(df, max_rooms, verbose=True):
    n_requests = len(df)

    # All requests must be assigned to one out of the rooms (same room during entire period).
    requestvars = intvar(0, max_rooms-1, shape=(n_requests,))

    m = Model()

    # Some requests already have a room pre-assigned
    for idx, row in df.iterrows():
        if not pd.isna(row['room']):
            m += (requestvars[idx] == int(row['room']))

    # A room can only serve one request at a time.
    # <=> requests on the same day must be in different rooms
    for day in pd.date_range(min(df['start']), max(df['end'])):
        overlapping = df[(df['start'] <= day) & (day < df['end'])]
        if len(overlapping) > 1:
            m += AllDifferent(requestvars[overlapping.index])

    return (m, requestvars)
```



Current combinatorial optimisation practice

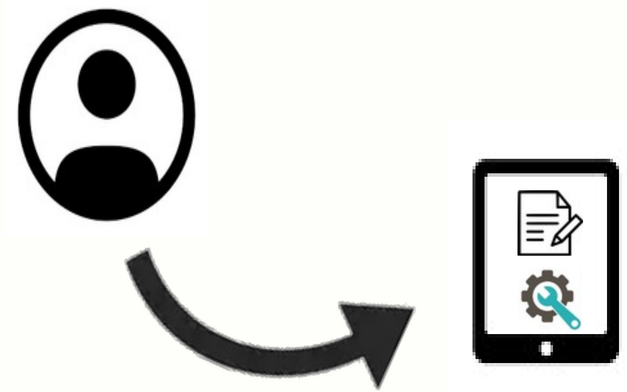


Current combinatorial opt. practice, problem



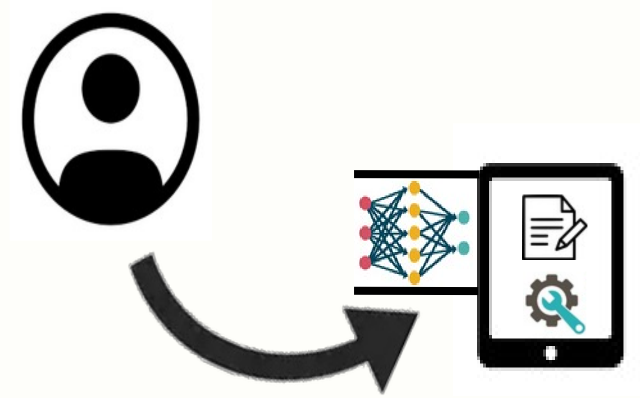
Incomplete information
Ad-hoc modifications
Changing preferences

Vision



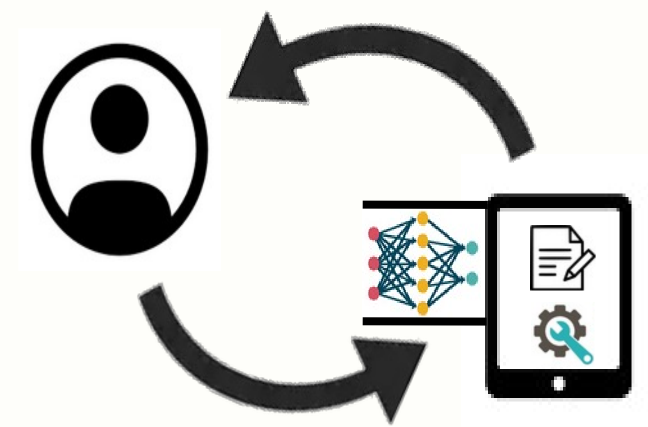
Vision

- Learning implicit user preferences
- Learning from the environment



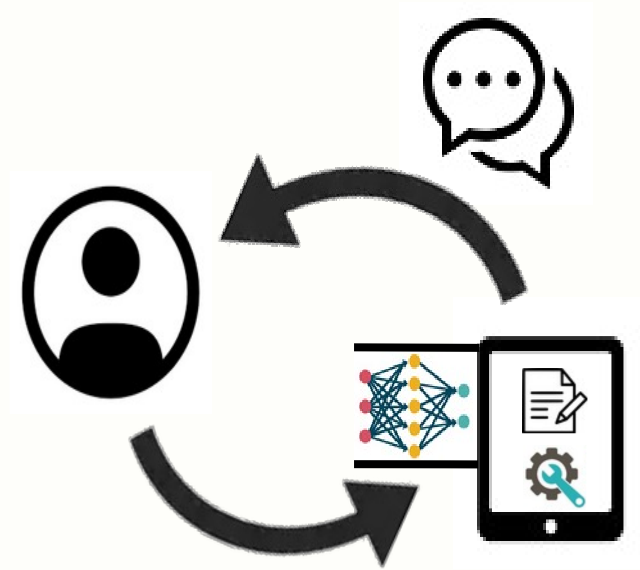
Vision

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



Vision

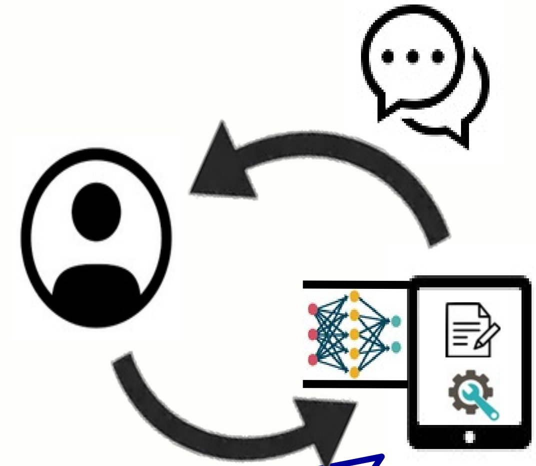
- Learning implicit user preferences
- Learning from the environment
- Explaining constraint solving
- Stateful interaction



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

European Research Council

Established by the European Commission



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

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

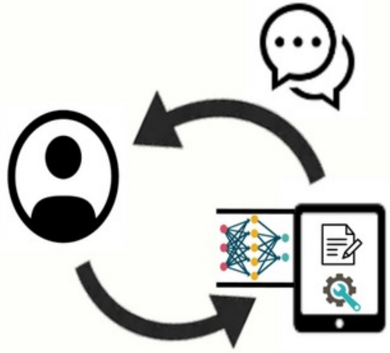
https://people.cs.kuleuven.be/~tias.guns/chat-opt.html

Hiring post-docs!

What would the ideal CP system be?

- 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

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

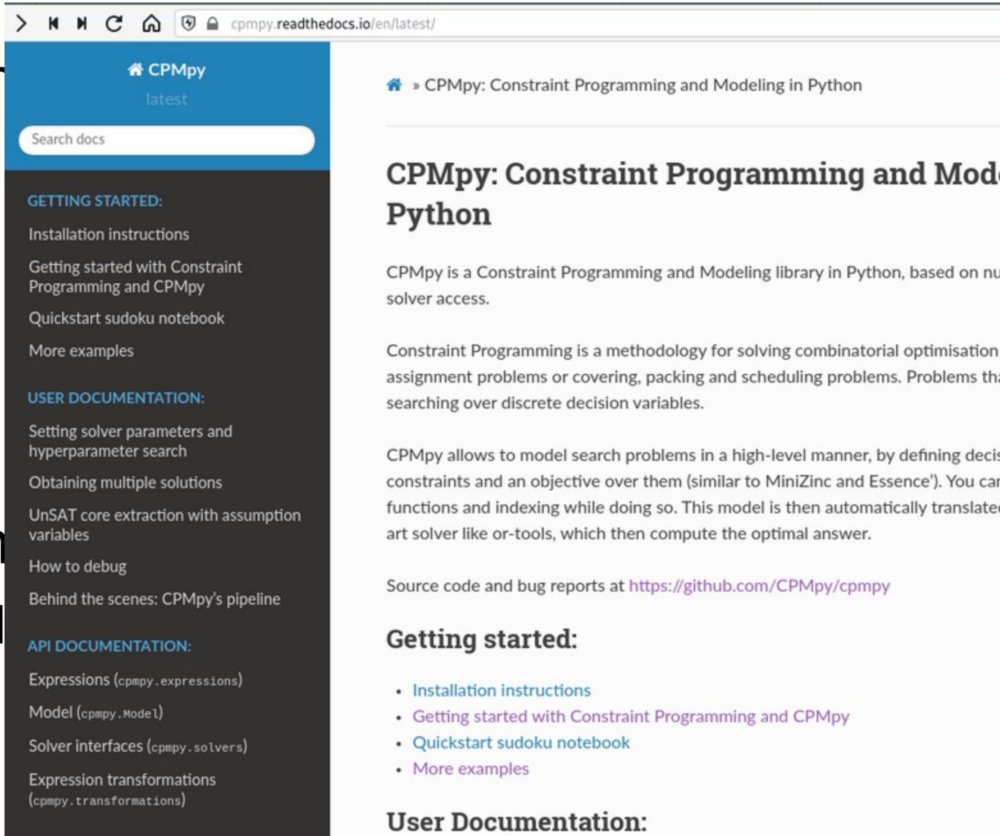


What would the ideal CP system be?

- Easy integration with Machin
  => Python and numpy arrays

- Efficient repeated solving
  => Incremental

- Use CP/SAT/MIP or any con
  => solver independent and mul



https://github.com/CPMpy/cpmpy

This talk

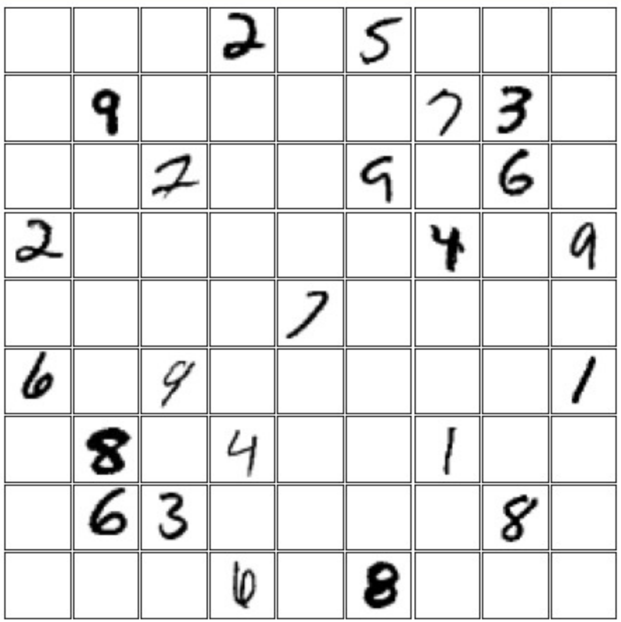
1. Integrating CP with ML *predictions*

2. Integrating CP with ML *learning*

3. CP Explanations *with implicit hitting sets*

+ High-level overview of how CPMpy enables this

Perception-based constraint solving



What if part of the input is in an image?

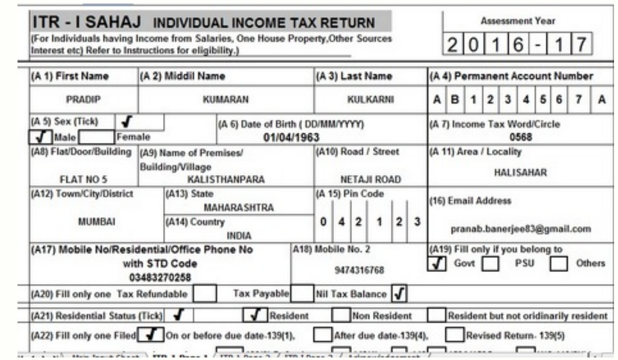
Pedagogical example: Visual Sudoku

- Every image represents a valid sudoku
- Explicitly know: CP constraints
- Need to predict: image labels

=> test limits of reasoning on learning

[Mulamba, Mandi, Canoy, Guns, CPAIOR20]

Perception data and constraint solving



ITR - I SAHAJ INDIVIDUAL INCOME TAX RETURN
(For Individuals having Income from Salaries, One House Property,Other Sources Interest etc) Refer to Instructions for eligibility.)

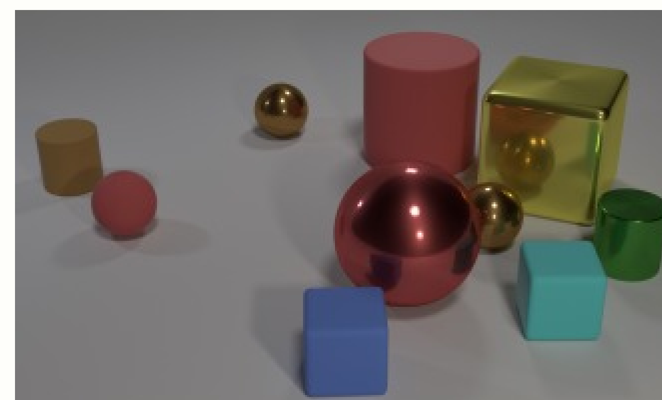
Assessment Year
2016-17

(A 1) First Name	(A 2) Middil Name	(A 3) Last Name	(A 4) Permanent Account Number
PRADIP	KUMARAN	KULKARNI	A B 1 2 3 4 5 6 7 A

(A 5) Sex (Tick) ✓ Male Female
(A 6) Date of Birth (DD/MM/YYYY) 01/04/1963
(A 7) Income Tax Word/Circle 0568
(A8) Flat/Door/Building FLAT NO 5
(A9) Name of Premises/ Building/Village KALISTHANPARA
(A10) Road / Street NETAJI ROAD
(A 11) Area / Locality HALISAHAR
(A12) Town/City/District MUMBAI
(A13) State MAHARASHTRA
(A14) Country INDIA
(A 15) Pin Code 0 4 2 1 2 3
(16) Email Address pranab.banerjee83@gmail.com
(A17) Mobile No/Residential/Office Phone No with STD Code 03483270258
A18) Mobile No. 2 9474316768
(A19) Fill only if you belong to ✓ Govt PSU Others
(A20) Fill only one Tax Refundable Tax Payable Nil Tax Balance ✓
(A21) Residential Status (Tick) ✓ ✓ Resident Non Resident Resident but not oridinarily resident
(A22) Fill only one Filed ✓ On or before due date-139(1), After due date-139(4), Revised Return- 139(5)

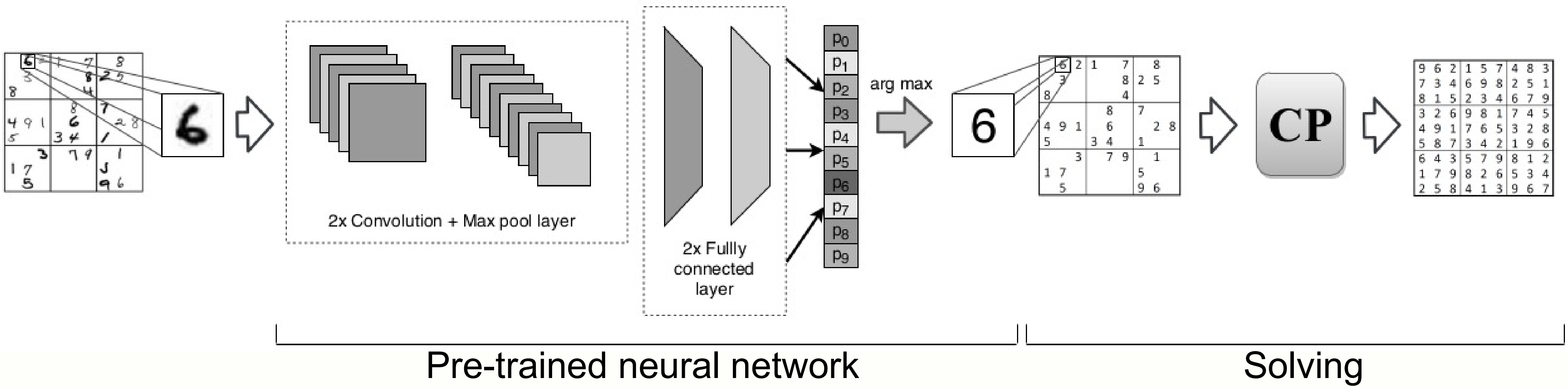
<u>Other application settings:</u>

- Document analysis
- Paper-based configuration problems (tax forms)
- Object-detection based reasoning
- Visual relationship detection
- ...



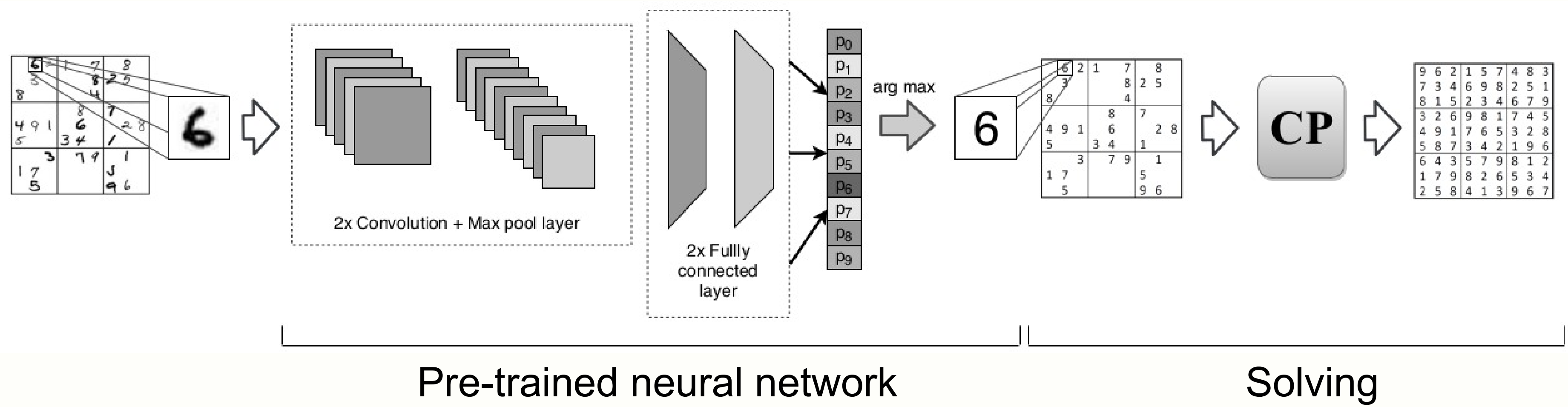
Perception-based constraint solving

Visual sudoku (naïve)



	accuracy			failure rate	time
	img	cell	grid	grid	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:

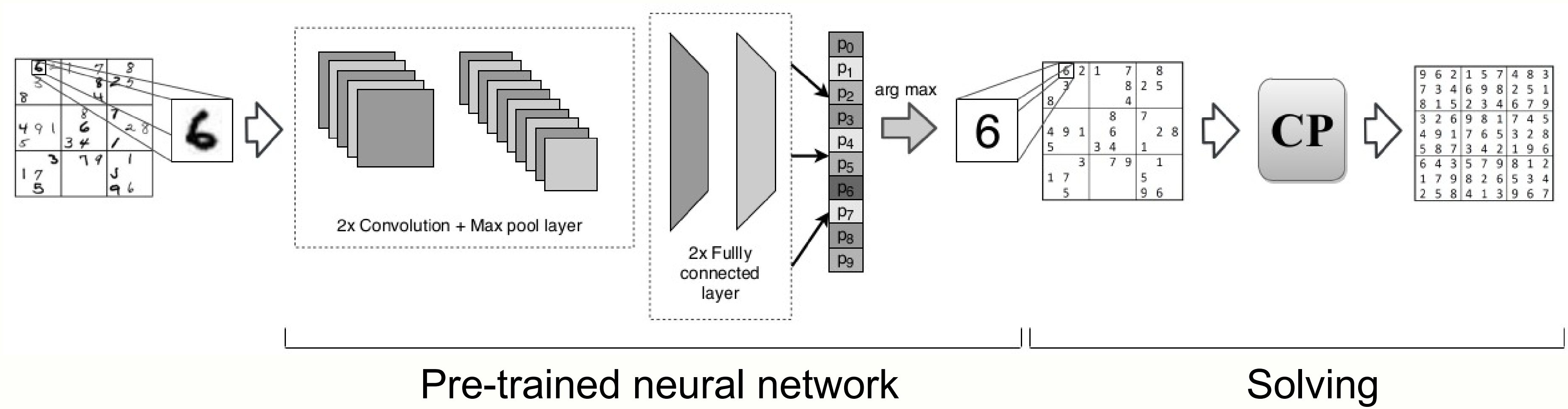
$$\hat{y}_{ij} = \arg\max P(y_{ij} = k | X_{ij})$$

- This is a multi-output problem (one for each given cell):
  together this is the 'maximum likelihood' interpretation

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

What about the *next* most likely interpretation?

Perception-based constraint solving



What about the *next* most likely interpretation?

- Treat prediction as *joint inference* problem:

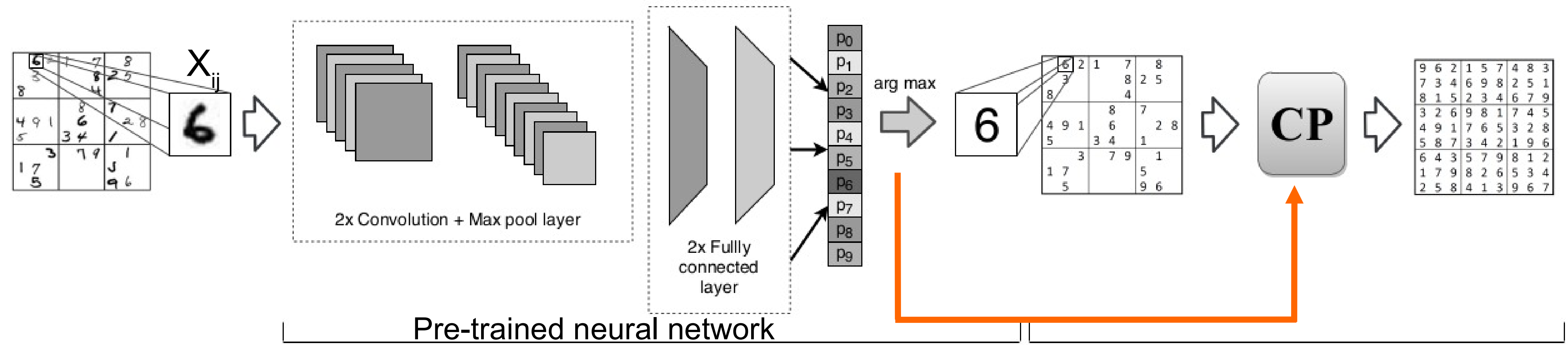
$$\hat{y} = \arg\max \prod_{ij} P(y_{ij} = k | X_{ij}) \quad \text{s.t.} \quad \text{sudoku}(\hat{y})$$

- 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:

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

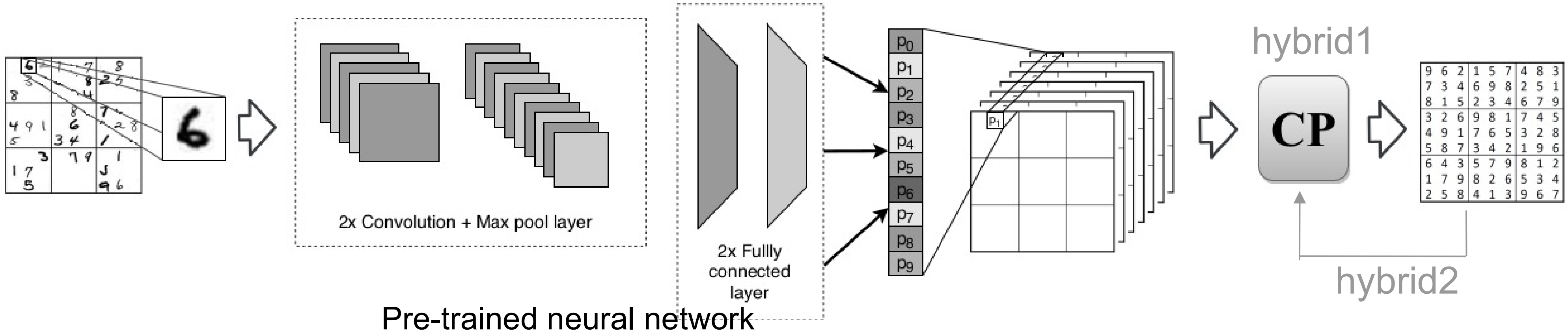
Visual sudoku

Demo with CPMpy

https://github.com/CPMpy/cpmpy/blob/master/examples/advanced/visual_sudoku.ipynb

Perception-based constraint solving

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



		accuracy		failure rate	time
	img	cell	grid	grid	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]

This talk

1. Integrating CP with ML *predictions*
2. **Integrating CP with ML *learning***
3. CP Explanations *with implicit hitting sets*

\+ High-level overview of how CPMpy enables this

Research trend

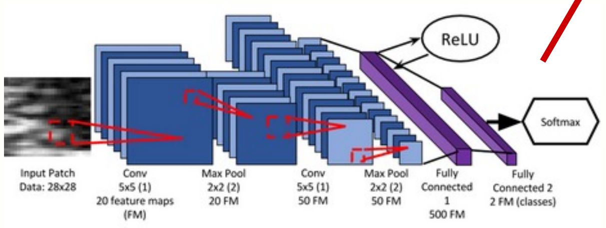
Model + Solve

1) learn the constraints

2) learn the objective function

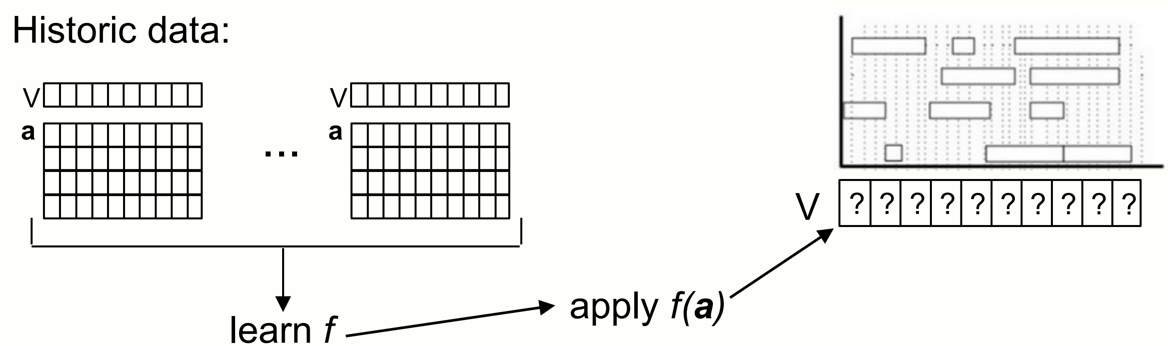
3) learn to solve (faster)

Can we *learn* it instead?



Learn the objective function

Prediction + Optimisation (regression of weights)



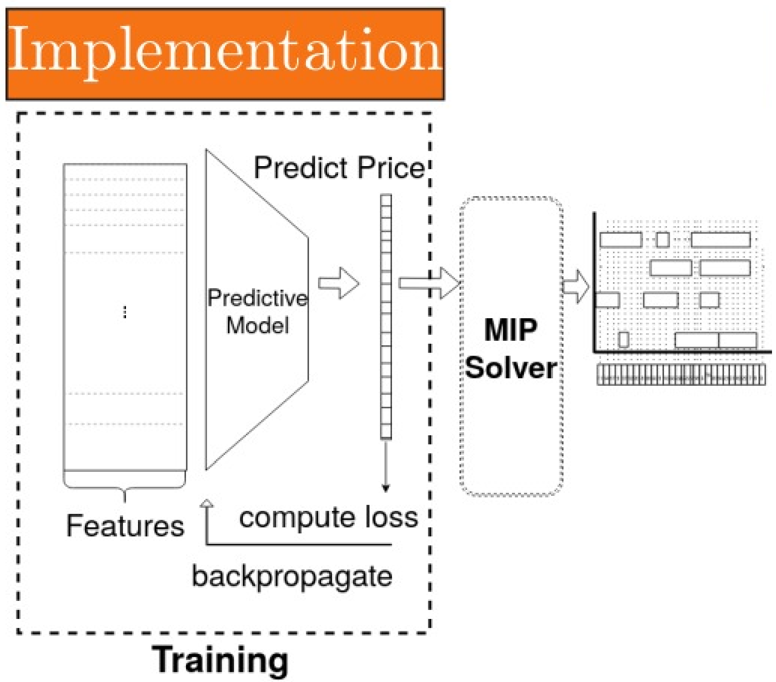
Predict: hourly energy prices + Optimize: energy-aware scheduling

---

Other examples:

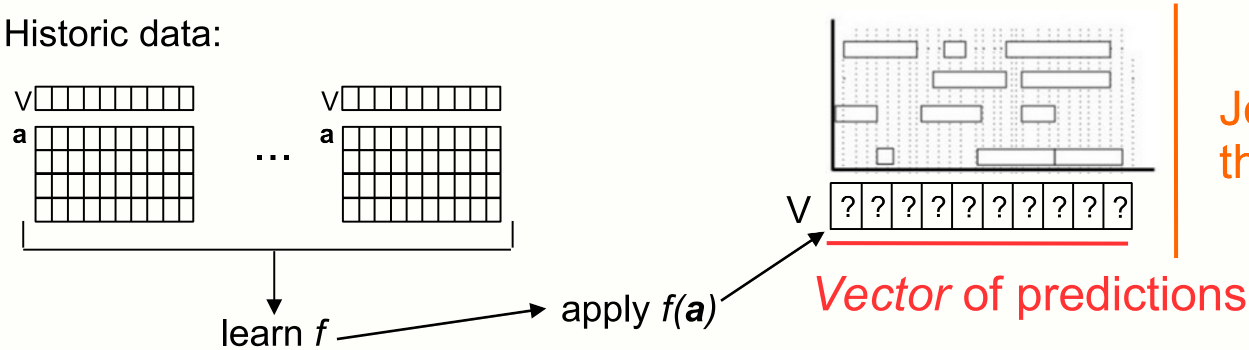
- Optimize steel plant production waste, by predicting steel defects
- Optimize money transport, by predicting value of coins at clients
- ...

prediction-focussed **regression**



Pre-trained neural network

MSE loss not the best proxy for *task* loss....

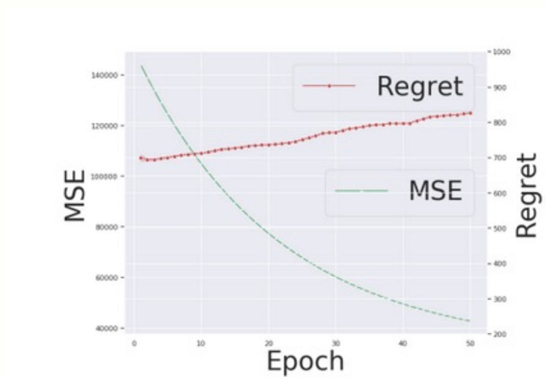


Joint inference: trades off the individual predictions

Why?

- MSE = average of individual errors of the vector
- Joint inference = *joint* error

  → some errors worse than others!



Which errors worse?

is combinatorial, need to *solve* to know

features — true cost vector

$$\underset{\omega}{\operatorname{argmin}}\, \mathbb{E}\left[regret\left(\underbrace{m(x_i;\omega)}_{\text{predicted cost vector}}, c_i\right)\right]$$

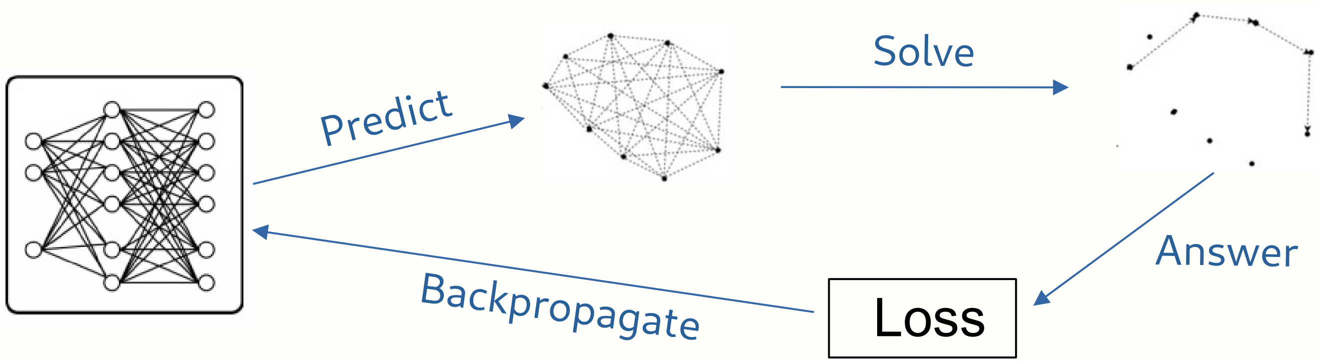
network params

$$\begin{aligned} regret(\hat{c}, c) &= f(\hat{v}, c) - f(v^*, c) \\ with \quad v^* &= argmin_{v \in V} f(v, c) \\ \hat{v} &= argmin_{v \in V} f(v, \hat{c}) \end{aligned}$$

Challenges:
- no explicit gradient
- V is implicit, exponential size
- argmin f may be NP-hard

[Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems, Jayanta Mandi, Emir Demirovic, Peter Stuckey, Tias Guns. AAAI20]

Learning approaches (gradient descent)



Key challenges:

1) suitable loss function? (non-differentiable solver)

2) scalability due to repeated solving:
once per instance per epoch

Related work for discrete optimisation

- Differentiating KKT of a relaxed problem [Wilder, B., Dilkina, B., & Tambe, M. (2019, July)., Ferber, A., Wilder, B., Dilkina, B., & Tambe, M. (2020, April)]
- Differentiating HSD of a relaxed problem [Mandi, J., & Guns, T. (2020)]
- Subgradient of a surrogate loss [Elmachtoub, A. N., & Grigas, P. (2022), Mulamba, M. & Mandi, J. & Diligenti, M. & Lombardi, M. & Bucarey, V. & Guns, T.]
- Differentiation by perturbation [Pogančić, Marin Vlastelica, et al. (2020), Niepert, M., Minervini, P., & Franceschi, L. (2021)]

Decision-focused learning

Suitable loss function?

Key observation:

“The objective function induces a ranking over feasible solutions”

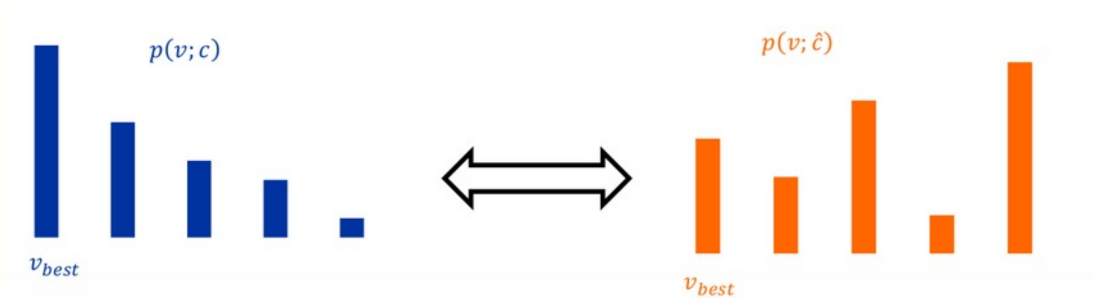
	Obj with true costs	Obj with predicted
Sol 1 [a,c,b,d,a]	12 (rank: 1)	14 (rank: 3)
Sol 2 [a,b,c,d,a]	15 (rank: 2)	10 (rank: 1)
Sol 3 [a,c,d,b,a]	16 (rank: 3)	11 (rank: 2)
Sol 4 [a,d,b,c,a]	23 (rank: 4)	16 (rank: 4)
Sol 5 [a,d,c,b,a]	28 (rank: 5)	18 (rank: 5)

[Mandi, Mulamba, Bucarey, Guns, Decision-focused learning: through the lens of learning to rank, ICML2022]

Decision-focussed learning

Assume a set of feasible solutions S.

“The objective function induces a ranking over feasible solutions”



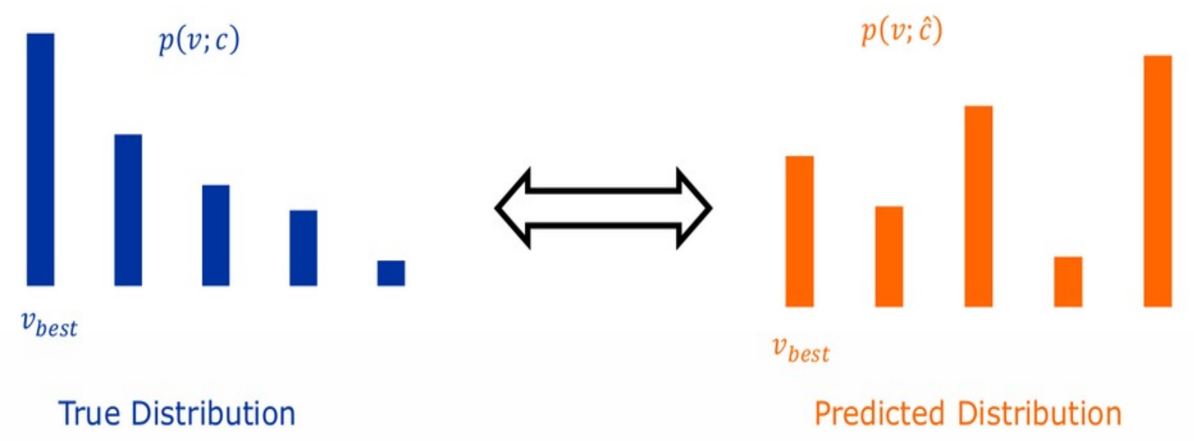
=> We can now use techniques from the much more mature ‘Learning to Rank’ field in ML!

[Mandi, Mulamba, Bucarey, Guns, Decision-focused learning: through the lens of learning to rank, ICML2022]

Listwise Learning 2 Rank for DFL

Discrete exponential distribution in solution space

$$p(v;c) = \begin{cases} \frac{1}{Z} exp(-f(v,c)/\tau) & v \in V \\ 0 & v \notin V \end{cases}$$



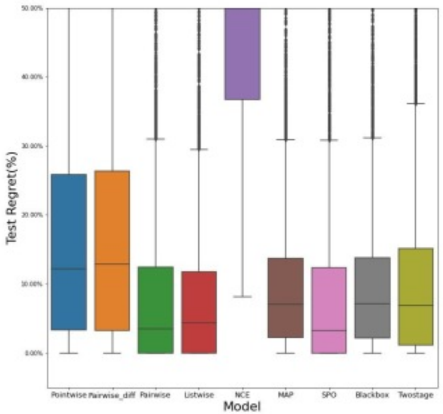
We obtain 2 distributions (one for true costs, one for predicted costs) over a finite sample of feasible solutions S

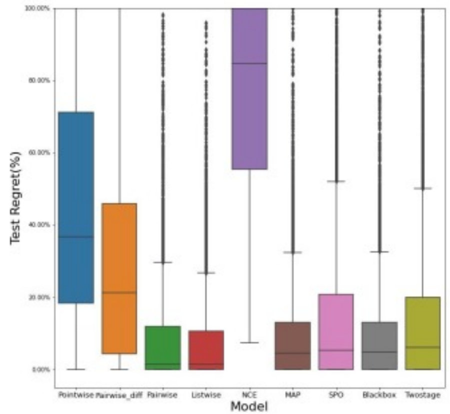
=> Can use the standard Kullback-Leibler Divergence loss!

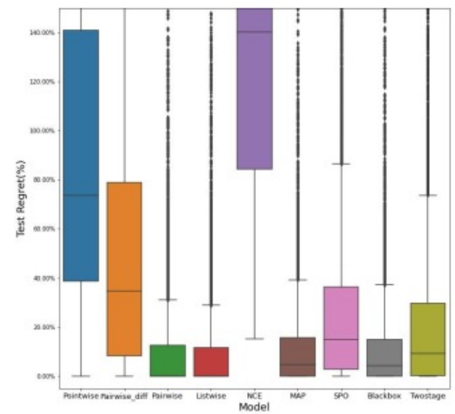
[Mandi, Mulamba, Bucarey, Guns, Decision-focused learning: through the lens of learning to rank, ICML2022]

Results

Shortest Path Problem



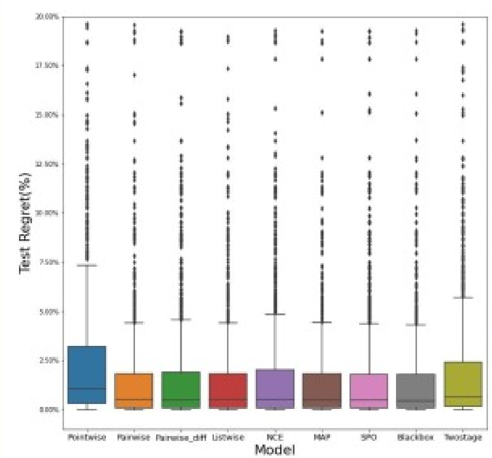


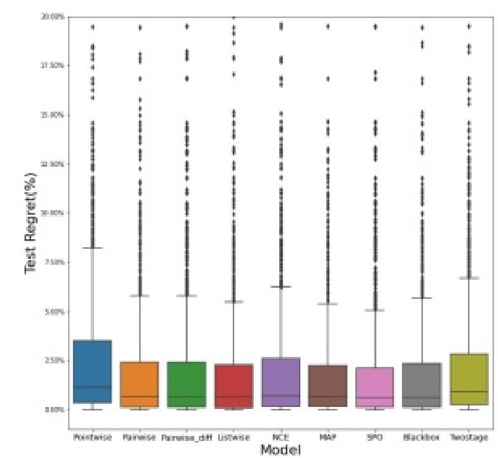


Degree 4

Degree 6

Degree 8

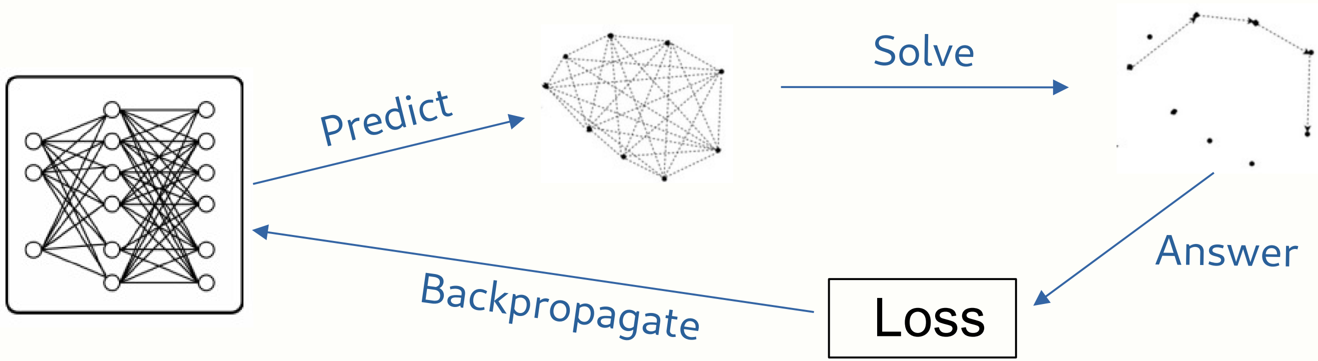




Scheduling Problem

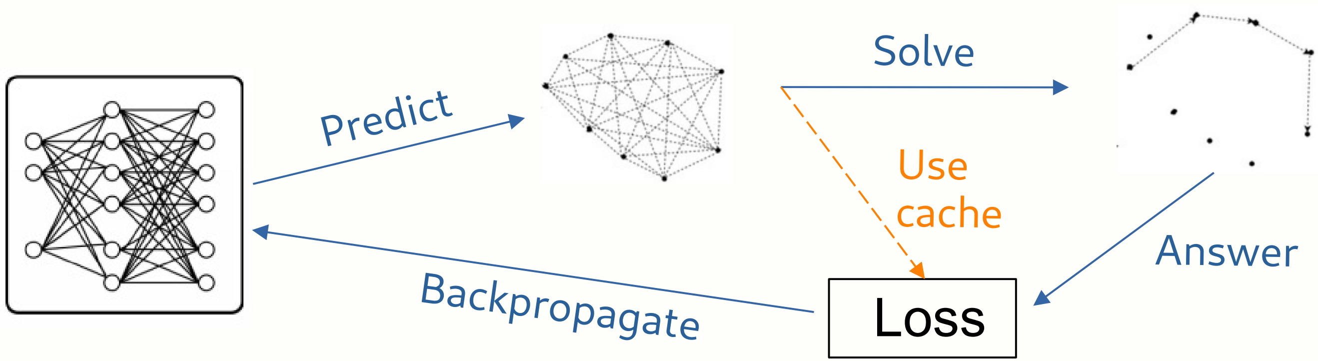
Decision-focused learning with L2R

2nd Key bottleneck: repeatedly calling the solver



Decision-focused learning with L2R

2nd Key bottleneck: repeatedly calling the solver



Can use *cached* solutions as approximate solver!!

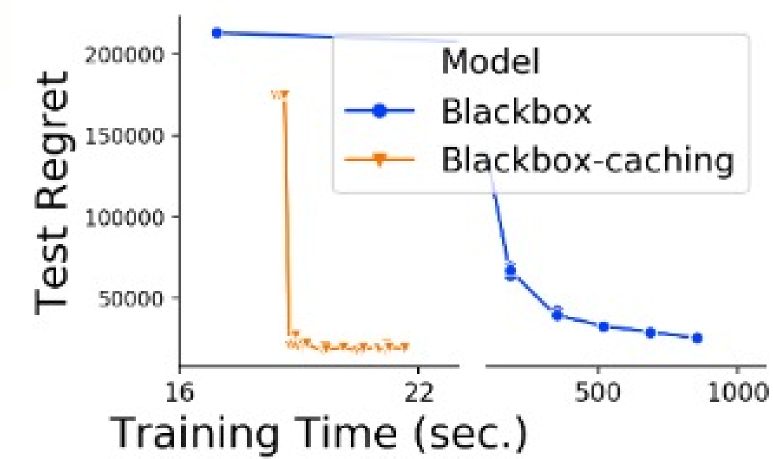
These cached solutions are the feasible set S

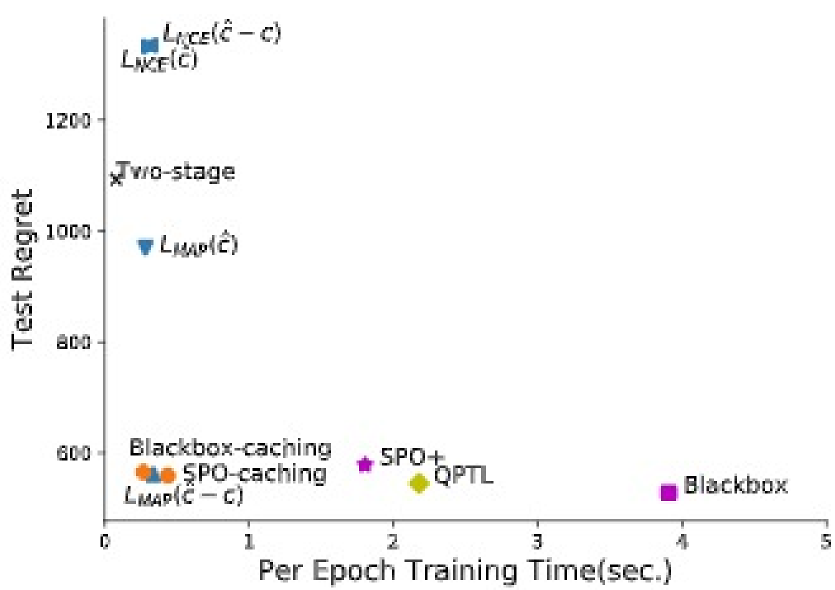
(also: sampling schemes: call the solver only 10% of the times)

[Mulamba, Mandi, Bucarey, Guns, Contrastive Losses and Solution Caching for Predict-and-Optimize, IJCAI2021]

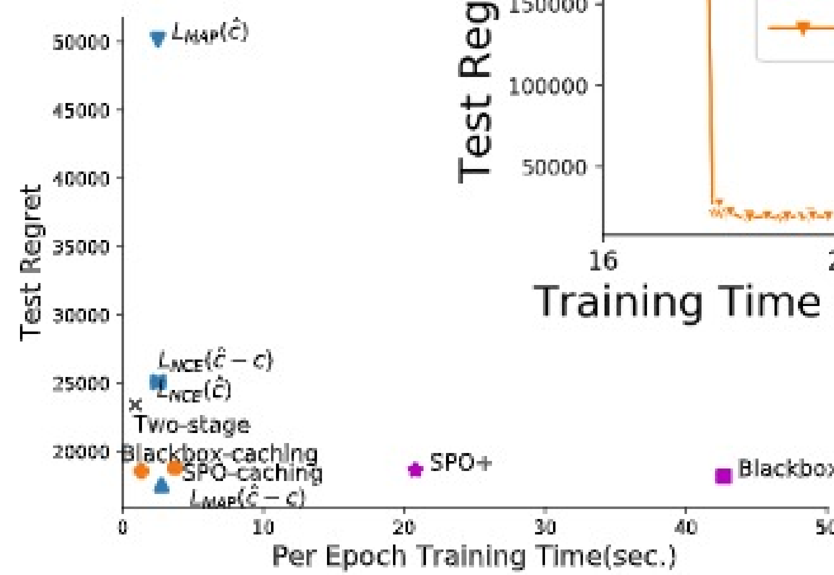
Results

Caching scheme compatible with **all** methods that call a blackbox solver *(call the cache instead, 90% of time)*





(a) Knapsack-120



(b) Energy-3

Implementation in gradient descent loop

Standard:

Algorithm 1: Stochastic gradient descent

Input : training data $\mathcal{D} = \{X, y\}_{i=1}^{n}$, learning rate γ

initialize θ (neural network weights)
for *epochs* **do**
  for *batches* **do**
    sample batch $(X, y) \sim \mathcal{D}$
    $\hat{y} \leftarrow g(z, \theta)$ (forward pass: compute predictions)
    Compute loss $L(y, \hat{y})$ and gradient $\frac{\partial L}{\partial \theta}$
    Update $\theta = \theta - \gamma \frac{\partial L}{\partial \theta}$ through backpropagation (backward pass)
  end
end

with Listwise ranking:

Algorithm 3: Stochastic gradient descent with KL on solutions

Input : training data $\mathcal{D} = \{X, y\}_{i=1}^{n}$, architecture g, learning rate γ, sample rate r

initialize θ (neural network weights of g)
sols $\leftarrow \{solver(y) \mid (X, y) \in \mathcal{D}\}$ (initialize with true solutions)
for *epochs* **do**
  for *batches* **do**
    sample batch $(X, y) \sim \mathcal{D}$
    $\hat{y} \leftarrow g(X, \theta)$ (forward pass: compute predictions)
    if *random()* $\leq r$ **then**
      sols $\leftarrow$ sols $\cup \{solver(\hat{y})\}$
    end
    Compute loss L $=$ KL$(distr(y, sols), distr(\hat{y}, sols))$ and grad. $\frac{\partial L}{\partial \theta}$
    Update $\theta = \theta - \gamma \frac{\partial L}{\partial \theta}$ through backpropagation (backward pass)
  end
end

Decision-focused learning

Demo with CPMpy (older method)

https://github.com/CPMpy/cpmpy/blob/master/examples/advanced/predict_plus_optimize.ipynb

This talk

1. Integrating CP with ML *predictions*

2. Integrating CP with ML *learning*

3. **CP Explanations *with implicit hitting sets***

\+ High-level overview of how CPMpy enables this

Debugging a model

Solver says UNSAT, what now?

→ compute Minimal Unsatisfiable Subset (MUS)

```
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
```

(faster if the solver supports unsat core extraction and assumptions)

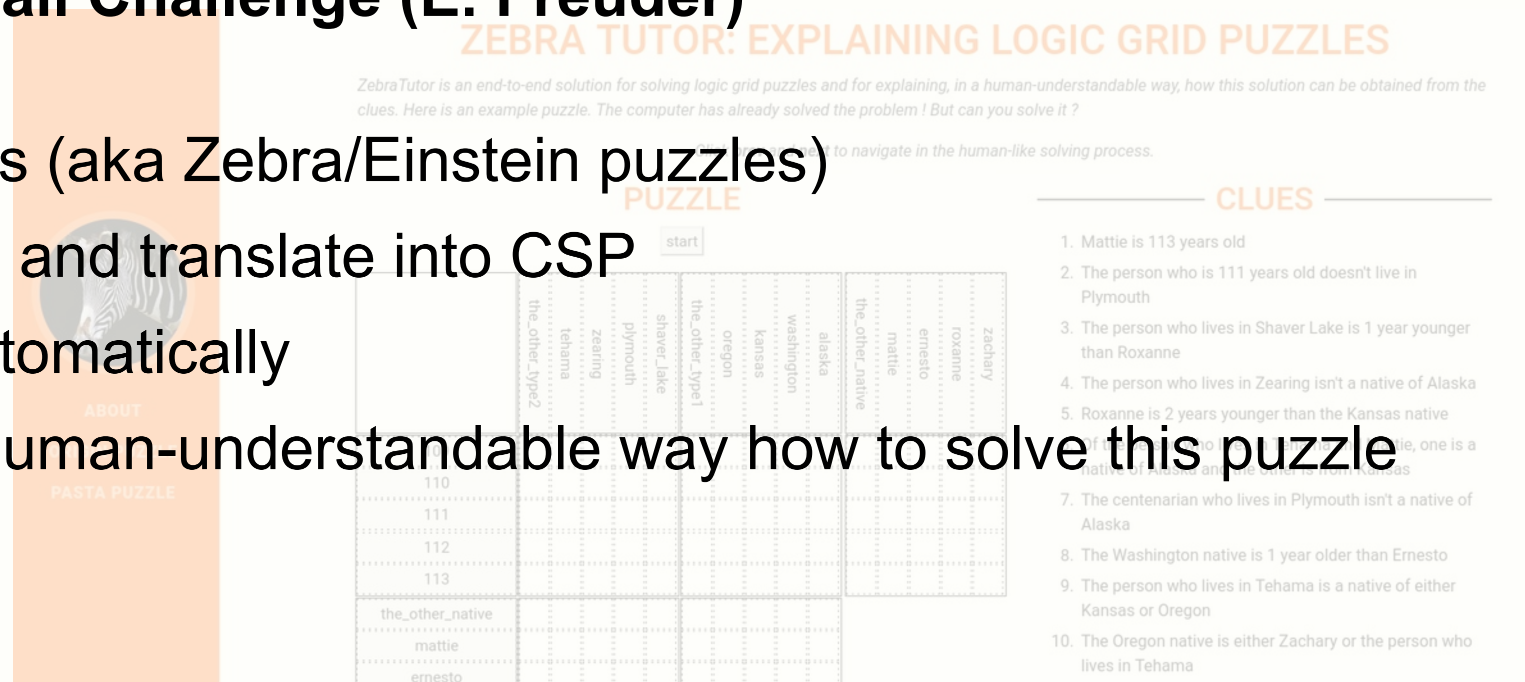
What if a model is SAT?

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

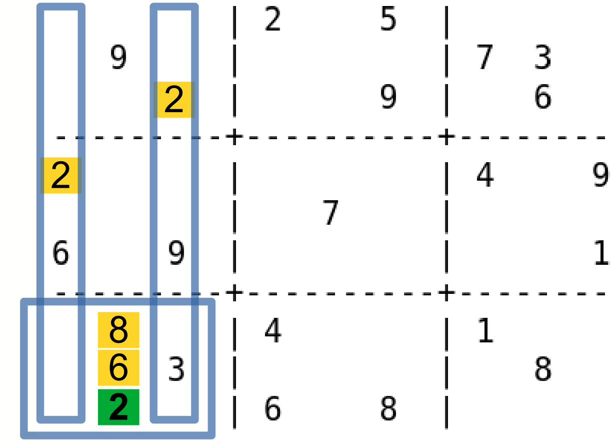
Ex. 2019 Holy Grail Challenge (E. Freuder)

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



Explanation steps



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

E’ = a subset of previously derived facts E

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

n = a newly derived fact

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

The best/easiest explanation step...

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

Simple MUS-based algo:

```
X_best = None
For n in optimal_propagate(constraints):
    X = MUS(~n & E & S)
    If f(X) < f(X_best):
        X_best = X
return X_best
```

But MUS gives no guarantees on quality, only subset minimal

Optimal unsatisfiable subsets

O(C)US: use an implicit hitting set algorithm (like MaxHS)

Algorithm 4: OCUS(T, f, p)

$\mathcal{H} \leftarrow \emptyset$
while true **do**
  $\mathcal{S} \leftarrow$ COST-OPTIMAL-HITTINGSET$(\mathcal{H}, f, p)$
  if $\neg$SAT$(\mathcal{S})$ **then**
    return $\mathcal{S}$
  end
  $\mathcal{S} \leftarrow$ GROW$(\mathcal{S}, T)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{T \setminus \mathcal{S}\}$
end

MIP solver

SAT/CP solver

Big efficiency gains
If incremental...

[E. Gamba, B. Bogaerts, T. Guns, Efficiently Explaining CSPs with Unsatisfiable Subset Optimization, IJCAI 2021]

Step-wise explanations

Demo with CPMpy

https://github.com/CPMpy/cpmpy/blob/master/examples/advanced/explain_stepwise_csp.ipynb

This talk

1. Integrating CP with ML *predictions*
2. Integrating CP with ML *learning*
3. CP Explanations *with implicit hitting sets*

+ High-level overview of how CPMpy enables this

Design

Design principle:

Aim to be a thin layer on top of solver API

Central concept: CPMpy expression

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

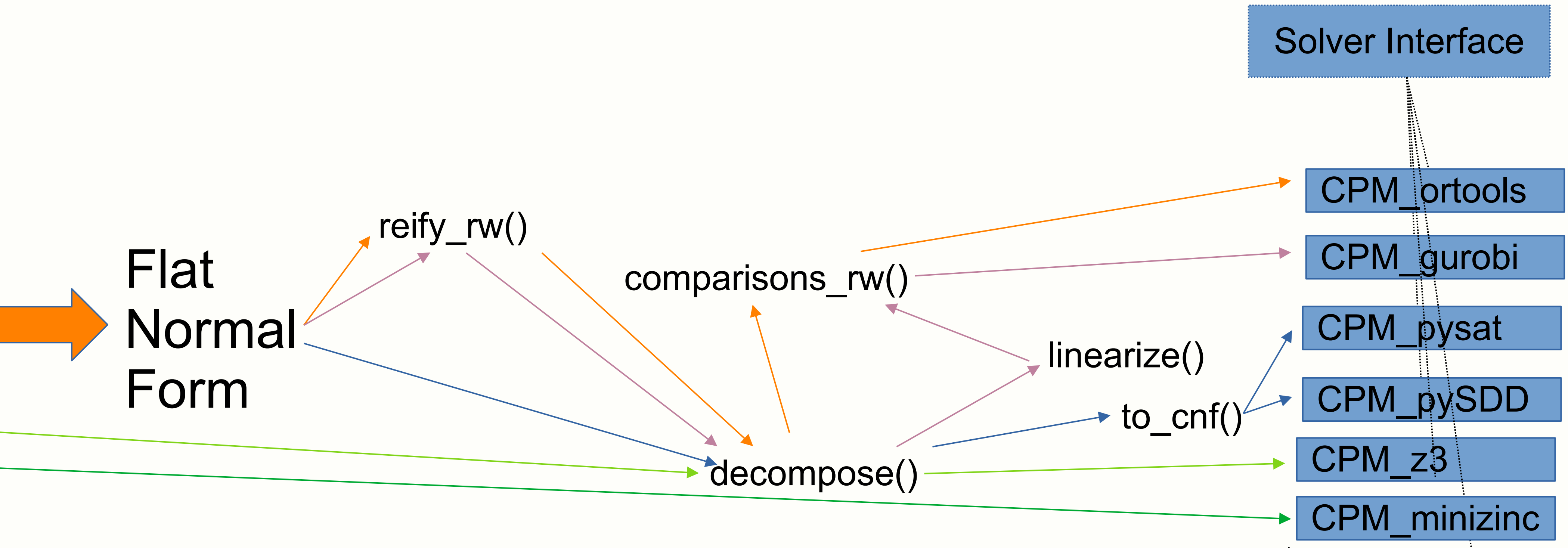
CPM_pysat

CPM_pySDD

solvers/

Only 1-to-1 mapping of supported expressions

Transformations in a nutshell



solvers/

Only 1-to-1 mapping of supported expressions

transformations/

Transformations are functions

- Flat Normal Form:
  - Removes nested expressions (except reification)
  - All subsequent transformations can assume input is ‘normalized’
- All transformations are pure *functions:*
  - Can call them in any order, and indep. of solver objects
  - State can be passed as an argument, but not required

  => they are **incremental**
- Great for debugging too

Solvers

We only interface to Python APIs
(unfortunately, no Common CP solver API : (

Key principle: solver can implement any subset of expressions!

Solvers can also choose to:

- Support assumptions or not
- Be incremental or not
- Expose own solver parameters

Currently:
- ortools
- pysat
- minizinc
- gurobi
- pySDD

Near future: ExactSolver, Z3
Wishlist: Mistral2, Geas, Gecode

This talk

0. Data/Visualisations

1. Integrating CP with ML *predictions*

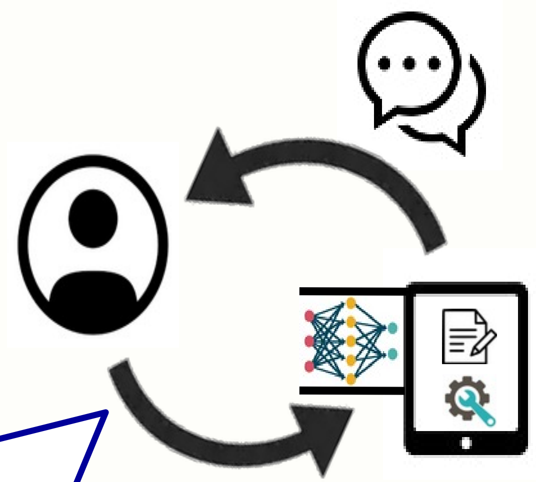
2. Integrating CP with ML *learning*

3. CP Explanations *with implicit hitting sets*

4. How does CPMpy enable this?

https://github.com/CPMpy/cpmpy

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

Hiring post-docs!