Learning from user and environment in combinatorial optimisation

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

Joint work with team members:
- Rocs Canoy
- Jayanta Mandi
- Maxime Mulamba
- Victor Bucarey Lopez
- Ahmed KA Abdullah
- Emilio Gamba

And external collaborators:
- Peter Stuckey (Monash Uni, Au)
- Emir Demirovic (TU Delft, NL)
- Michelangelo Diligenti (Sienna Uni, It)
- Michele Lombardi (Bologna, It)
- Bart Bogaerts (VUB, Be)
My background

4 years PhD @ DTAI lab KU Leuven
   Constraint Programming for Pattern Mining

4 years PostDoc @ DTAI
   Combining Constraint Programming and ML/DM

4 years Assist. Prof. @ VUB (Solvay Buss. School)
   Domain-specific ML (mobility, fintech, transport)
   Prediction + Optimisation

8 months Assoc. Prof @ DTAI
   + ERC CHAT-Opt
Combinatorial optimisation

“Solving constrained optimisation problems”

- Vehicle Routing
- Scheduling
- Configuration
- Graph problems
Current combinatorial optimisation practice

Model + Solve

Opt. expert ➔ Domain experts ➔ Stakeholders

- Domain experts
- Stakeholders
Current combinatorial opt. practice, problem

Model + Solve

Too rigid, too static

Opt. expert

Domain experts

Stakeholders
Research trend

Can we *learn* it instead?

1) learn the constraints
2) learn the objective function
3) learn to solve
Prediction + constraint solving

- Part \textit{explicit} knowledge: in a formal language
- Part \textit{implicit} knowledge: learned from data
Prediction + constraint solving

- Part **explicit** knowledge: in a formal language

- Part **implicit** knowledge: learned from data
  - tacit knowledge (*user preferences, social conventions*)
  - complex environment (*demand, prices, defects*)
  - perception (*vision, natural language, audio*)
Tacit knowledge (user preferences)

“Vehicle routing by learning from historical solutions”

[Rocsildes Canoy and Tias Guns, CP19], Best student paper award

GOAL: Learn preferences, reduce manual effort, adapt to changes over time!
Tacit knowledge (user preferences)

Small data: 6 months = 26 weeks = 130 week days (instances)
Tacit knowledge (user preferences)

For single vehicles, in mobility mining literature:

- Driver turn prediction [Krumm, 2008]
- Prediction of remainder of route early in the trip [Ye et al., 2015]
- Prediction of route given origin and destination [Wang et al., 2015]

Can we use similar techniques (Markov Models) to learn preferences across routings of multiple vehicles?

And can we optimize over them with constraint solving?
Learning and prediction part

1st order Markov approximation:
\[ P([s_1,s_2,s_3,...]) = P(s_1)*P(s_2|s_1)*P(s_3|s_2)*... \]

→ estimate the \( P(s_y|s_x) \) by observing the transitions in the actually driven routes

probability of transition = relative nr of observations in the data

\[ t_{ij} = \frac{f_{ij} + \alpha}{N_i + \alpha d}, \]
Constrained optimisation: what now?

Goal: find maximum likelihood solution:

\[
\text{maximize } P([s_1, s_2, s_3, \ldots]) = P(s_1) * P(s_2|s_1) * P(s_3|s_2)* \ldots \\
\text{s.t. VRP}([s_1, s_2, s_3, \ldots])
\]

Standard probability computation trick: log-likelihood

\[
\max \prod_{(i,j) \in X} \Pr(\text{next stop}=j \mid \text{current stop}=i), \\
= \max \sum_{(i,j) \in A} \log(t_{ij})x_{ij}.
\]

\[
\rightarrow \text{VRP: replace distance matrix by negative log-likelihood matrix!}
\]

\[
\min \sum_{(i,j) \in A} c_{ij} x_{ij} \quad \Rightarrow \quad \min \sum_{(i,j) \in A} -\log(t_{ij})x_{ij}.
\]

Compatible with ALL vrp solvers
Can we do the learning better?

Training data = a sequence (one for every day) of observed routing sequences

→ each routing is over slightly different sets of customers

→ preferences can change over time (concept drift)
Concept drift

When 'counting' the probabilities:

- can include a prior on each historic instance wrt. current day
- e.g. weighing of the instance:
  \[ F = \sum_{t} w_t A^t. \]
  - uniform = unit weight
  - by similarity = how much overlap in clients with current day
  - by time = more recent instances get higher weight
  incl. exponential smoothing
Concept drift, quality AFTER solving

Fig. 7  Route and arc difference during concept drift (drop in number of stops)

Fig. 8  Route and arc difference during concept drift (rise in number of stops)
Tacit knowledge (user preferences)

\[
\max \prod_{(i,j) \in X} \Pr(\text{next stop} = j \mid \text{current stop} = i).
\]

\[
\min \sum_{(i,j) \in A} c_{ij} x_{ij} \implies \min \sum_{(i,j) \in A} -\log(t_{ij}) x_{ij}.
\]

- Solvable with **any VRP solver**, including constraints
- Better than traditional approaches, multiple weighing schemes possible
Prediction + constraint solving

- Part **explicit** knowledge: in a formal language
- Part **implicit** knowledge: learned from data
  - tacit knowledge (*user preferences, social conventions*)
  - perception (*vision, natural language, audio*)
  - complex environment (*demand, prices, defects*)
Perception data and constraint solving

ML view:
Wang, Donti, Wilder, Kolter; ICML19
- can we learn the (pairwise) sudoku constraints?
- test limits of learning for reasoning

Our view part explicit, part implicit:
- know constraints, get predictions
- maximum likelihood problem?
- test limits of reasoning on learning

[Mulamba, Mandi, Canoy, Guns, CPAIOR20]
Perception data and constraint solving

Other application settings:

- Document analysis
- Paper-based configuration problems (tax forms)
- Object-detection based reasoning
- ...
Perception-based constraint solving

Pedagogical instantiation: visual sudoku (naïve)

<table>
<thead>
<tr>
<th></th>
<th>img</th>
<th>accuracy cell</th>
<th>grid</th>
<th>failure rate grid</th>
<th>time average (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>94.75%</td>
<td>15.51%</td>
<td>14.67%</td>
<td>84.43%</td>
<td>0.01</td>
</tr>
</tbody>
</table>
put more into the constraint solving

remove \( \text{arg max} \) of each individual prediction:

\[
\hat{y}_{ij} = f_{\theta}(X_{ij}) = \text{arg max}_{k \in \{0, \ldots, 9\}} P_{\theta}(y_{ij} = k | X_{ij}),
\]

and make part of the CP model!

\[
\min \sum_{(i,j) \in \text{given \{1, \ldots, 9\}}} \sum_{k \in \text{constant}} -\log(P_{\theta}(y_{ij} = k | X_{ij})) \times \mathbb{1}[s_{ij} = k]
\]

s.t. sudoku(s)

= find the maximum likelihood solution!
What does that mean, maximum likelihood solution of a Sudoku?

The *unconstrained* max likelihood solution = argmax prediction of each image
→ sudoku constraint **forbids** certain solutions: find next most likely one...

In ML speak: the solver does 'joint inference' over *all* predictions, takes structure into account
ex. used in Natural Language Processing [Punyakanok, COLING04]
Perception-based constraint solving

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

<table>
<thead>
<tr>
<th></th>
<th>img</th>
<th>accuracy</th>
<th>failure rate</th>
<th>time average (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cell</td>
<td>grid</td>
<td>grid</td>
<td></td>
</tr>
<tr>
<td>baseline</td>
<td>94.75%</td>
<td>15.51%</td>
<td>14.67%</td>
<td>84.43%</td>
</tr>
<tr>
<td>hybrid1</td>
<td>99.69%</td>
<td>99.38%</td>
<td>92.33%</td>
<td>0%</td>
</tr>
<tr>
<td>hybrid2</td>
<td>99.72%</td>
<td>99.44%</td>
<td>92.93%</td>
<td>0%</td>
</tr>
</tbody>
</table>

[Maxime Mulamba, Jayanta Mandi, Rocsildes Canoy, Tias Guns, CPAIOR20]
Prediction + constraint solving

- Part explicit knowledge: in a formal language

- Part implicit knowledge: learned from data
  - tacit knowledge (user preferences, social conventions)
  - perception (vision, natural language, audio)
  - complex environment (demand, prices, defects)

Time for end-to-end learning!
Complex environment (demand, prices)

Prediction + Optimisation aka decision-focussed learning:

- Optimize task scheduling's energy cost, by predicting energy prices
- Optimize steel plant production waste, by predicting steel defects
- Optimize money transport, by predicting amount of coins at clients

- multi-output prediction
- discrete optimisation, batch (non-sequential)
Prediction + Optimisation, two-step

Pre-trained neural network
Can we do the (deep) learning better?

MSE loss function is not informative enough

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

Why?

- MSE = average of individual errors of the vector
- Joint inference = \textit{joint} error
  \rightarrow some errors worse than others!
Complex environment (demand, prices)

Which errors worse? is combinatorial, need to solve to know

Goal: end-to-end learning with regret as loss

\[
\text{regret}(\hat{v}, c) = f(\hat{v}, c) - f(v^*, c)
\]
\[
\text{with } v^* = \arg\min_{v \in V} f(v, c)
\]
\[
\hat{v} = \arg\min_{v \in V} f(v, \hat{c})
\]

Challenges:
- each regret comp. is NP-hard
- argmin over exponential nr. of outcomes
- discrete & non-differentiable

Problem formulation

Can be seen as a bi-level optimisation problem:

$$\arg \min \frac{1}{N} \sum_{i=1}^{N} f(v_i, c) - f(v_i^*, c)$$

s.t. $v_i^* \in \arg \min_{v \in V} f(v, c_i)$

$\forall i \in \{1, \ldots, N\}$

$\forall i \in \{1, \ldots, N\}$

Challenges:
- $\arg \min f$ is not unique
- $V$ is implicit, exponential size
- $\arg \min f$ may be NP-hard
Bilevel optimisation?

Can be seen as a bi-level optimisation problem:

\[
\text{argmin}_{\omega} \frac{1}{N} \sum_{i=1}^{N} f(v_i, c) - f(v_i^*, c)
\]

\[s.t. \quad v_i^* \in \text{argmin}_{v \in V} f(v, c_i) \quad \forall i \in 1..N\]

\[v_i \in \text{argmin}_{v \in V} f(v, m(x_i; \theta)) \quad \forall i \in 1..N\]

Assume \( f \) is linear and \( V \) is continuous, e.g. \( \text{argmin} f = \) an LP

Solution not unique:

- pessimistic assumption = \( \text{argmin} f \) will return 'worst' regret solution → need to compute all equivalent solutions to find worst, tri-level!

- optimistic assumption = \( \text{argmin} f \) returns 'best' regret solution → ML model can 'cheat' by making ambiguous predictions
SPO+ loss

[Elmachtoub & Grigas, 2017 2021]

Defines an upperbound on pessimistic that is convex:

\[ \ell_{\text{SPO+}}(\hat{c}, c) := \max_{w \in S} \left\{ c^T w - 2\hat{c}^T w \right\} + 2\hat{c}^T w^*(c) - z^*(c). \]

Most importantly: subgradient (for in gradient-descent learning)

subgradients: 2(\(v^* - \text{argmin}_v f(2m(x_i,w) - c^*)\))

True optimal solution \hspace{1cm} \text{Optimal solution under perturbed predicted cost vector}

Key idea is (imho) perturbation of the predictions,

- solve convex combination of real \(c^\ast\) and predicted \(c\) values: solve(2c – c\(^\ast\)) = solve(c\(^\ast\) + 2(c-c\(\ast\)))
- amplifies error of predictions and avoids abusing equivalent solutions
**SPO+: a deeper look at the (deep) learning**

Standard:

```
Algorithm 1: Stochastic gradient descent

Input : training data \( D = \{X, y\}_{i=1}^n \), learning rate \( \gamma \)

1. initialize \( \theta \) (neural network weights)

2. for epochs do
3.     for batches do
4.         sample batch \( (X, y) \sim D \)
5.         \( \hat{y} \leftarrow g(z, \theta) \) (forward pass: compute predictions)
6.         Compute loss \( L(y, \hat{y}) \) and gradient \( \frac{\partial L}{\partial \theta} \)
7.         Update \( \theta = \theta - \gamma \frac{\partial L}{\partial \theta} \) through backpropagation (backward pass)
8.     end
9. end
```

with SPO+:

```
Algorithm 2: Stochastic gradient descent with SPO+ subgradient

Input : training data \( D = \{X, y\}_{i=1}^n \), architecture \( g \), learning rate \( \gamma \)

1. initialize \( \theta \) (neural network weights of \( g \))

2. for epochs do
3.     for batches do
4.         sample batch \( (X, y) \sim D \)
5.         \( \hat{y} \leftarrow g(X, \theta) \) (forward pass: compute predictions)
6.         \( \hat{y} = y + 2(\hat{y} - y) \) // SPO+ trick, convex comb. of \( y \) and \( \hat{y} \)
7.         Solve \( \text{sol} = \text{solver}(\hat{y}) \) // calls external solver
8.         Use subgradient \( \partial L = \text{solver}(\hat{y}) - \text{sol} \)
9.         Update \( \theta = \theta - \gamma \frac{\partial L}{\partial \theta} \) through backpropagation (backward pass)
10. end
11. end
```

we need to solve a comb. problem on line 7 for every training example
(typically: 10-50 epochs, of 500 to 5000 samples...)
Can we do the solving better?

Solving MIP = repeatedly solving LP
- Do we need to solve the MIP to optimality? or to a small gap?
- Can we replace the MIP by the LP relaxation?

Solving LP = repeatedly finding improved basis
- Can we warm-start from previous basis's?

Observe: constraints always the same, only cost vector $c$ changes, and we solve it for thousands of $c$ values, each instance having a different true optimal solution.

SPO-relax is scalable

- Really hard instances: (1+ hour for single MIP solution)
- SPO-relax with total time budget:

<table>
<thead>
<tr>
<th>Hard Instances (200 tasks on 10 machines)</th>
<th>Two-stage Approach</th>
<th>SPO-relax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2 epochs</td>
<td>4 epochs</td>
</tr>
<tr>
<td>instance I</td>
<td>90,769</td>
<td>88,952</td>
</tr>
<tr>
<td>instance II</td>
<td>128,067</td>
<td>124,450</td>
</tr>
<tr>
<td>instance III</td>
<td>129,761</td>
<td>128,400</td>
</tr>
<tr>
<td>instance IV</td>
<td>135,398</td>
<td>132,366</td>
</tr>
<tr>
<td>instance V</td>
<td>122,310</td>
<td>120,949</td>
</tr>
</tbody>
</table>

Related work using deep learning (gradient descent)

Differentiable task losses for end-to-end learning:

**Black box (subgradient methods):**
- SPO+[1]: solve with $f(2c - c^*)$ (convex comb of real and predicted values)
- bb[2]: solve with $f(c)$ and $f(c + \text{eps})$ perturbed predictions

**White box (implicit differentiation):**
- QPTL[3]: solve Quadratic Program, differentiate KKT conditions
- Melding[4]: solve tightened LP relaxation as QP
- IntOpt[5]: solve LP with Interior Point, differentiate HSD

[1] Elmachtoub AN, Grigas P. Smart" predict, then optimize" arxiv, 2017
Prediction + Optimisation for MIP

SPO's subgradient is an indirect 'black box' method
→ If we know it is a MIP... can we get better gradients?

Can we compute the gradient of a MIP?
  » Discrete so non-differentiable

Can we compute the gradient of an LP?
  » Linear objective, so 2\textsuperscript{nd} derivative is 0, so not invertible

Can we compute the gradient of a QP?
  » yes, through \textit{implicit differentiation} [Amos and Kolter, ICML2017]
Prediction + Optimisation for MIP

Can the QP results be used for LPs?

\[
\max \theta^T x \ \text{s.t.} \ Ax = b, \ Gx \leq h
\]

→ make LP a QP by adding quadratic \( ||v||^2 \) term

\[
\max \theta^T x - \gamma ||x||^2 \ \text{s.t.} \ Ax = b, \ Gx \leq h
\]

(with some hyperparameter gamma)

→ can use QP techniques! [Amos and Kolter, ICML2017]

Prediction + Optimisation for MIP

But wait... why an arbitrary gamma*||x||^2?

→ Interior Point solvers have been computing gradients of LPs for years?

\[
\begin{align*}
\min & \quad c^\top x \\
\text{subject to} & \quad Ax = b; \\
& \quad x \geq 0; \text{ some or all } x_i \text{ integer}
\end{align*}
\]

Interior point solving: adding a logarithmic barrier to the objective

\[
f(c, x) := c^\top x - \lambda \left( \sum_{i=1}^{k} \ln(x_i) \right)
\]

- twice differentiable KKT conditions
- lambda is \textit{automatically} decreased during barrier solving
- implicitly enforces x >= 0

["Interior Point Solving for LP-based prediction + optimisation", Jayanta Mandi, Tias Guns. NeurIPS20]
**LP Forward Pass**
1. Solve the **Homogeneous Self-dual embedding**
2. Perform a Newton step
3. Decrease $\lambda$

**LP Backward Pass**
1. Differentiate the **Homogeneous Self-dual embedding** computed in the Forward pass
2. Compute and backpropagate $dx^*(\hat{c})/d\hat{c}$

**Forward Pass**
- **LP solving with barrier: Int. Point method**
- Predict $\hat{c}$
- Discrete ILP
- Relaxed LP

**Training Data**
- $z$ $c$
- $z$ $c$

**Compute Task Loss:**
$$c^T[x^*(\hat{c}) - x^*(c)]$$

**Update Neural Net parameters to minimize Task Loss**

["Interior Point Solving for LP-based prediction + optimisation", Jayanta Mandi, Tias Guns. NeurIPS20]
Interior Point Solving for LP-based prediction + optimisation

**KKT vs HSD**

<table>
<thead>
<tr>
<th></th>
<th>KKT, log barrier</th>
<th>HSD, log barrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda / \lambda$-cut-off</td>
<td>$10^{-1}$</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>Regret</td>
<td>14365</td>
<td>14958</td>
</tr>
</tbody>
</table>

Table 1: Differentiating the HSD formulation is more efficient than differentiating the KKT condition

**Comparison with the state of the art**

<table>
<thead>
<tr>
<th></th>
<th>Two-stage</th>
<th>QPTL</th>
<th>SPO</th>
<th>HSD, log barrier</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0-layer</td>
<td>1-layer</td>
<td>0-layer</td>
<td>1-layer</td>
</tr>
<tr>
<td>MSE-loss</td>
<td>745</td>
<td>(7)</td>
<td>796 (5)</td>
<td>3516 (56)</td>
</tr>
<tr>
<td>Regret</td>
<td>13322</td>
<td>(1458) (2021)</td>
<td>13652 (325)</td>
<td>13590 (288)</td>
</tr>
</tbody>
</table>

Table 2: Our approach is able to outperform the state of the art

Problem formulation

Can be seen as a bi-level optimisation problem:

\[
\arg\min_{\omega} \mathbb{E}[\text{regret} \ (m(x_i; \omega), c_i)]
\]

Challenges:
- \( \arg\min f \) is not unique
- \( V \) is implicit, exponential size
- \( \arg\min f \) may be NP-hard
Contrastive loss

Gradient over exponential-sized argmin/argmax?

→ **Contrastive loss**: for \( n >> 1 \)
  turn n-ary argmax into n-1 *pairwise* argmaxs!
  (then subsample some)
Contrastive loss

Gradient over exponential-sized argmin/argmax?

→ **Contrastive loss**: for \( n \gg 1 \)
  
  turn \( n \)-ary argmax into \( n-1 \) *pairwise* argmaxs!
  
  (then subsample some)

For decision-focussed learning:

\[
\mathcal{L}_{\text{NCE}} = \sum_i \sum_{v^* \in S} \left( f(v^*_i, m(\omega, x_i)) - f(v^*, m(\omega, x_i)) \right)
\]

for some **subset** of solutions \( S \)

Prediction + Optimisation for MIP and more

All current methods use a 'continuous relaxation' to make it non-discrete and hence (almost) differentiable.

Observation: constraints always stay the same, so the polytope is always the same.

→ Can we also use an inner approximation?

Figure 1: Representation of a solution cache (blue points) and the continuous relaxation (green area) of $V$. 

Prediction + Optimisation for MIP and more

Inner approximation = pool of known solutions
→ can replace 'solver()' by 'linear pass' over finite solutions! (SPO+,BB)
→ can use this cache as subsample 'S' in contrastive loss!

Main advantage: do not have to call a solver for each training instance!
Can 'grow' solution cache FAST and GOOD

Key take-aways:

- **Explicit** knowledge: use solver
- **Implicit** knowledge: do learning

- Joint inference / collective classification: maximize log likelihood!

- Keep revisiting the solving AND the learning, hybridize and use properties of one in the other!

- Comb. optimisation inside neural loss becoming actually feasible → end-to-end hybrid prediction and optimisation
Future Work

- Complexity of learned models vs. complexity of CP solving
- Faster (runtime), more accurate learning
- Interactive preference learning, multi-agent
- Other perception data (language, voice, camera)

- Wide range of applications (Industry 4.0, transport & more)
It's OK to dream

- Learning from the environment
- Learning implicit user preferences
It's OK to dream

- Learning from the environment
- Learning implicit user preferences
- Explaining constraint solving?
“From human-level problem specification, to human-level solving and explanations.”

Step-Wise Explanations for CSPs

I₀ = I₀ (initial information) + N₀ (new information in the grid)
C constraints (alldifferent, George did not take pasta, …)

How did we get there?
1. Using one or a subset of the constraints (S ⊆ C)
2. Part of the already derived Facts (E ∈ I)

EXPLANATION
Explanation is an implication of the form: Eᵢ ∧ Sᵢ ⇒ Nᵢ

EXPLANATION SEQUENCE

\[ I₀ \xrightarrow{+N₀} S₀ \xrightarrow{I₁} S₁ \xrightarrow{+N₁} I₂ \xrightarrow{+N₂} \ldots \xrightarrow{+Nₙ} I_{\text{end}} \]

Based on MUS (Minimal Unsatisfiable Subsets)

Now: cost-optimal MUSs and constrained
[accepted, IJCAI21]

MUS-based explanations also used in explainable ML [Ignatiev, Narodytska, Marques-Silva]
Conversational Human-Aware Technology for Optimisation

NLP rapidly advancing, but optimisation is NOT ready.

→ Need a **learning** optimisation system: user preferences and environment

→ Need a query-based interface

Would be **paradigm shift** in use of constraint optimisation! *(scheduling, routing, configuration, ...)*

+ Needs right combination of **data science** and **optimisation**
CHAT-Opt: **Conversational Human-Aware Technology for Optimisation**

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 @TiasGuns

**Hiring post-docs!**