End-to-end decision focussed learning

with predict+optimize

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

Joint work with team members:
- Jayanta Mandi
- Maxime Mulamba
- Victor Bucarey Lopez

And external colaborators:
- Peter Stuckey (Monash Uni, Au)
- Emir Demirovic (TU Delft, NL)
- Michelangelo Diligenti (Sienna Uni, It)
- Michele Lombardi (Bologna, It)

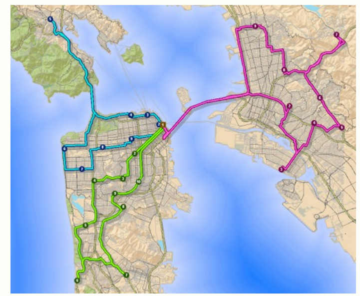
First,

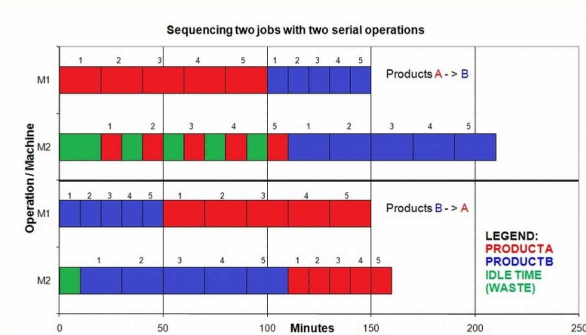
General research theme in my lab...

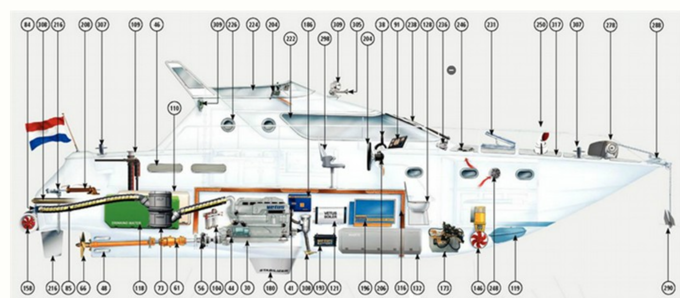
Combinatorial optimisation

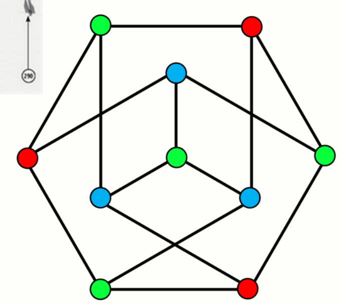
"Solving *constrained* optimisation problems"

- Vehicle Routing
- Scheduling
- Configuration
- Graph problems

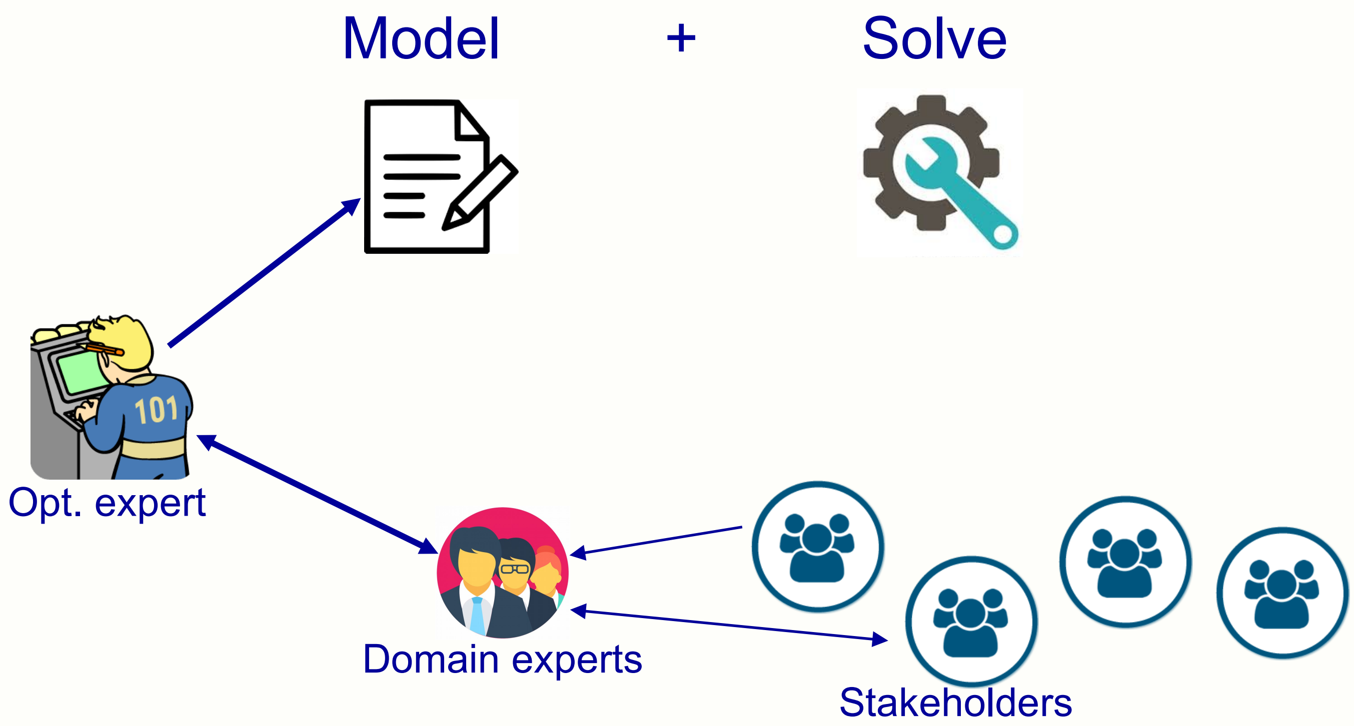




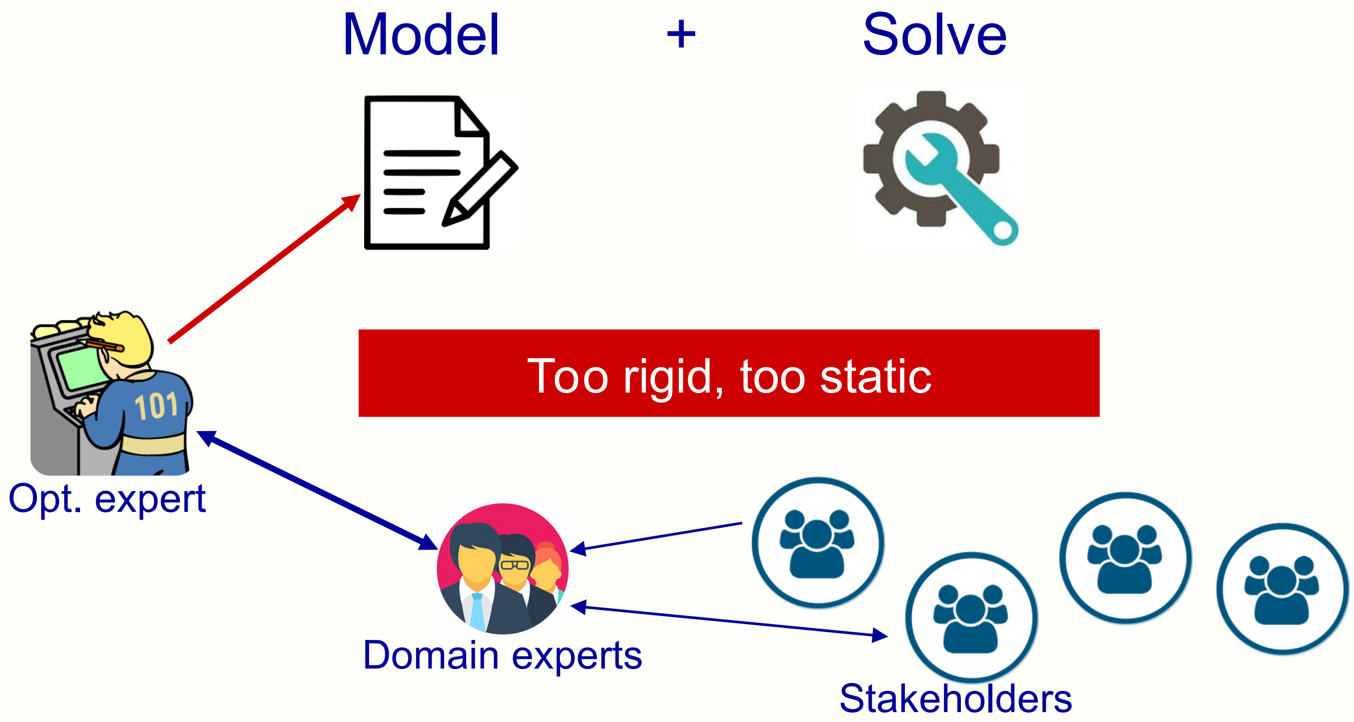




Current combinatorial optimisation practice



Current combinatorial opt. practice, problem



Research trend

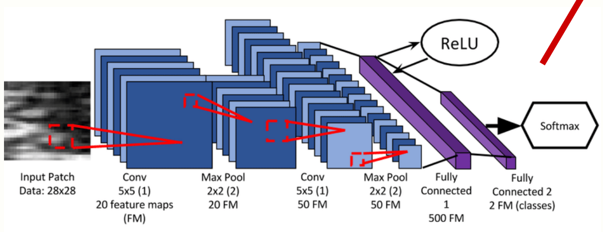
Model + Solve

1) learn the constraints

2) learn the objective function

3) learn to solve

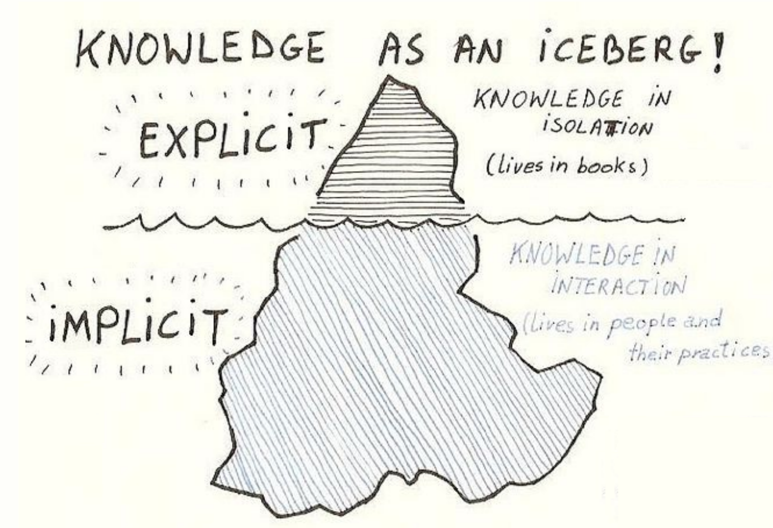
Can we *learn* it instead?



Prediction + constraint solving

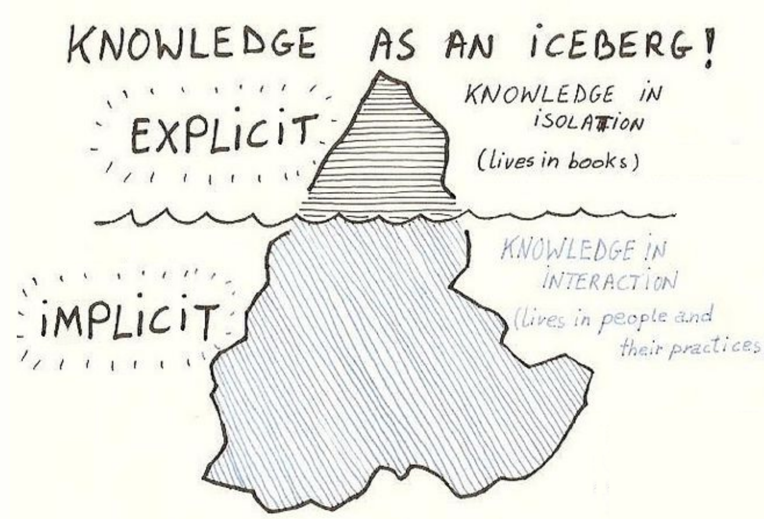
- Part <u>explicit</u> knowledge:
  in a formal language

- Part <u>implicit</u> knowledge:
  learned from data



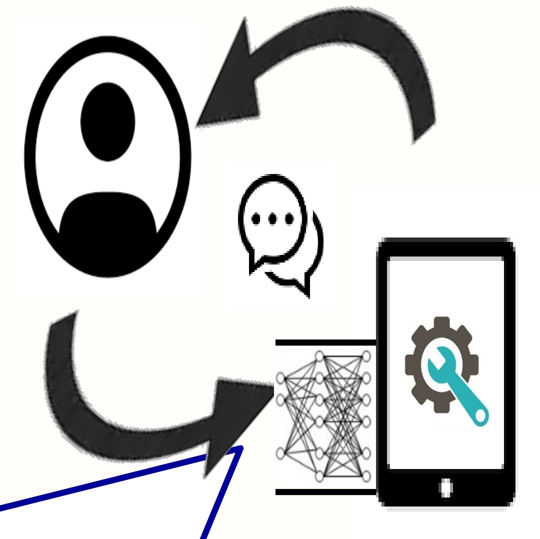
Prediction + constraint solving

- 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)*

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



Towards **co-creation** of combinatorial 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

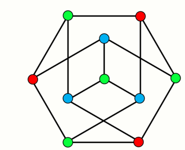
Predict + Optimize for combinatorial opt.

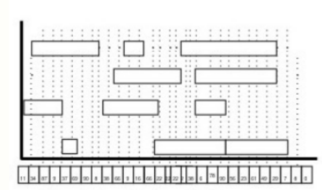
aka decision-focussed learning

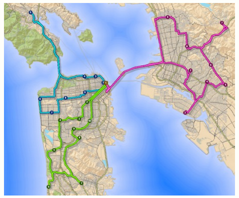
One type of Hybrid AI: learning + reasoning

here, reasoning technology = combinatorial optimisation

» discrete (Boolean/Integer valued choices)
» constraints
» requires search

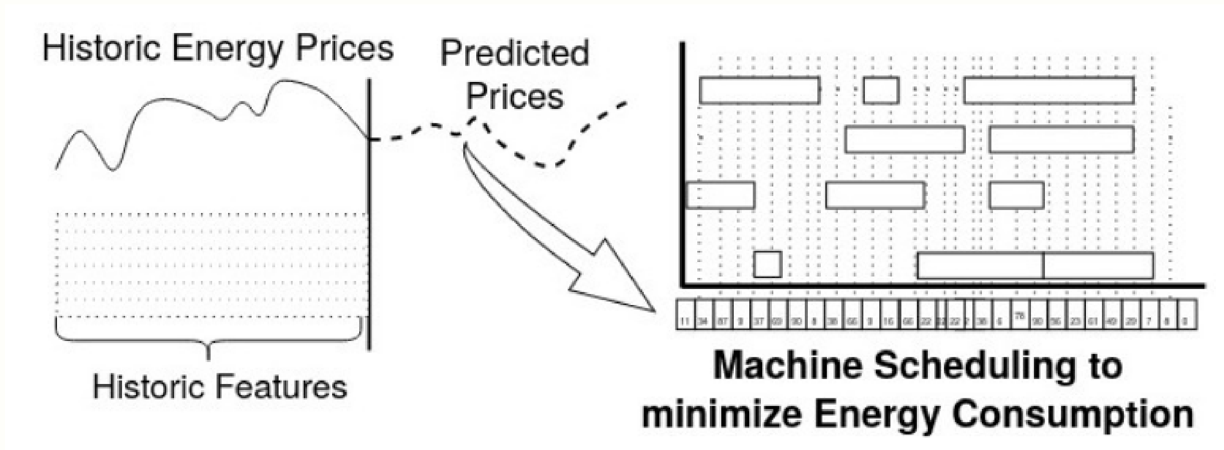






Complex environment (demand, prices)

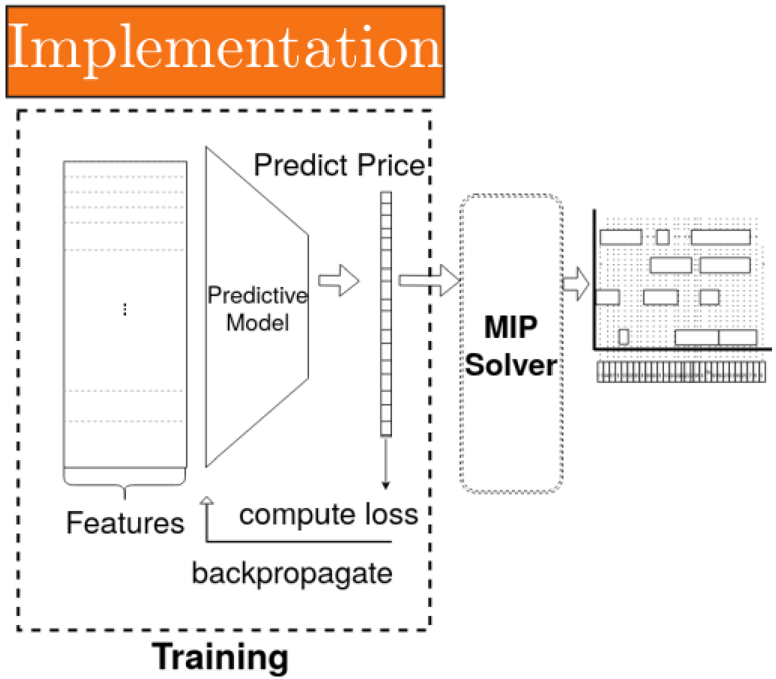
Prediction + Optimisation aka decision-focussed learning:



- multi-output prediction
- discrete optimisation, batch (non-sequential)

- 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

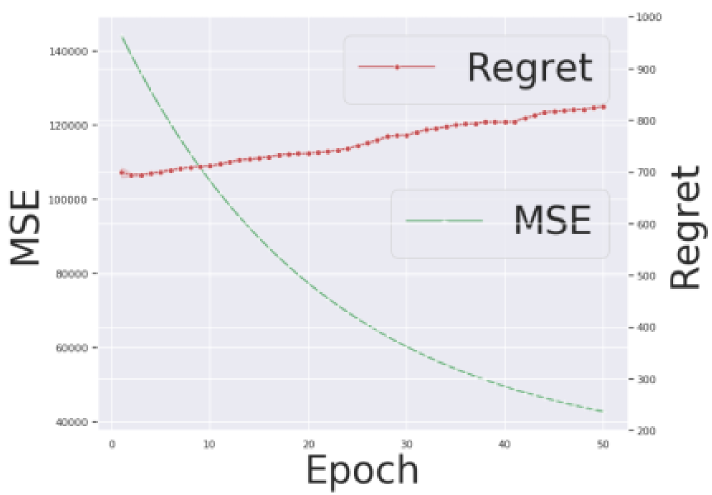
Prediction + Optimisation, two-step



Pre-trained neural network

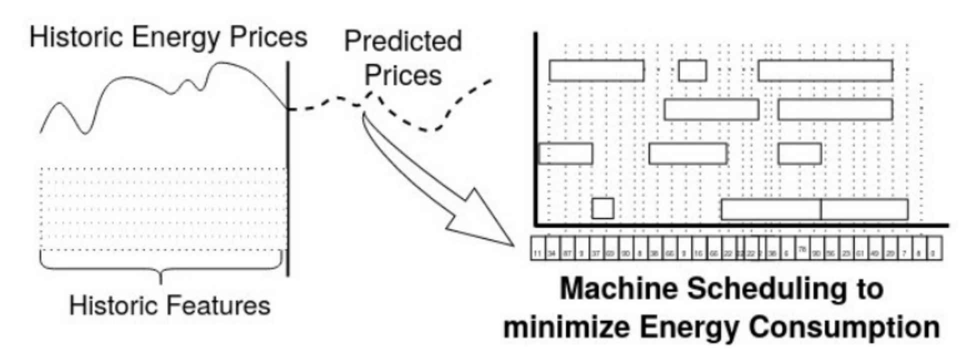
Can we do the (deep) learning better?

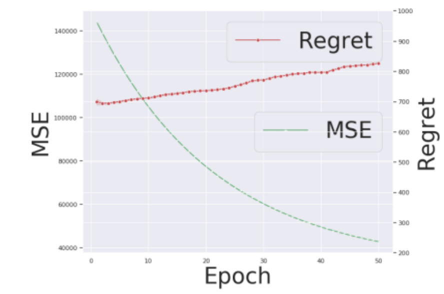
MSE loss function is not informative enough



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

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





Vector of predictions

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!

Complex environment (demand, prices)

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

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

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

Challenges:

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

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

Problem formulation

features true cost vector

$$\underset{\omega}{\operatorname{argmin}}\, \mathbb{E}\left[regret\left(m(x_i;\omega), c_i\right)\right]$$

network params predicted cost vector

Can be seen as a bi-level optimisation problem:

Learning

$$\underset{\omega}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} f\left(v_i, c\right) - f\left(v_i^*, c\right)$$

Reasoning (scheduling, routing)

$$s.t. \quad v_i^* \in argmin_{v \in V} f\left(v, c_i\right) \qquad \forall i \in 1..N$$

$$v_i \in argmin_{v \in V} f\left(v, m(x_i;\omega)\right) \qquad \forall i$$

Challenges:

- argmin f is not unique
- V is implicit, exponential size
- argmin f may be NP-hard

Bilevel optimisation?

Can be seen as a bi-level optimisation problem:

$$\underset{\omega}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} f\left(v_i, c\right) - f\left(v_i^*, c\right)$$
$$\begin{aligned} s.t. \quad & v_i^* \in argmin_{v \in V} f\left(v, c_i\right) && \forall i \in 1..N \\ & v_i \in argmin_{v \in V} f\left(v, m(x_i; \omega)\right) && \forall i \in 1..N \end{aligned}$$

Assume f is linear and V is continuous, e.g. argmin f = an LP

Solution not unique:

- pessimistic assumption = argmin f will return 'worst' regret solution
  → need to compute all equivalent solutions to find worst, tri-level!
- optimistic assumption = 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:

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

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

subgradients: 2(v^*_i – argmin_v f(2$m(x_i,w)$ -c*))

True optimal solution

Optimal solution under perturbed predicted cost vector

Key idea is (imho) perturbation of the predictions,

- solve convex combination of real *c** and predicted *c* values: solve(2c – c*) = solve(c* + 2(c-c*))
- amplifies error of predictions and avoids abusing equivalent solutions

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 + eps) perturbed predictions

[1] Elmachtoub AN, Grigas P. Smart" predict, then optimize" arxiv 2017; Management Science 2021
[2] Pogancic, Marin Vlastelica, et al. "Differentiation of Blackbox Combinatorial Solvers." ICLR. 2020

SPO+: a deeper look at the (deep) learning

Standard:

Algorithm 1: Stochastic gradient descent

Input : training data $\mathcal{D} = \{X, y\}_{i=1}^{n}$, learning rate γ
1 initialize θ (neural network weights)
2 **for** *epochs* **do**
3   **for** *batches* **do**
4   sample batch $(X, y) \sim \mathcal{D}$
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+:

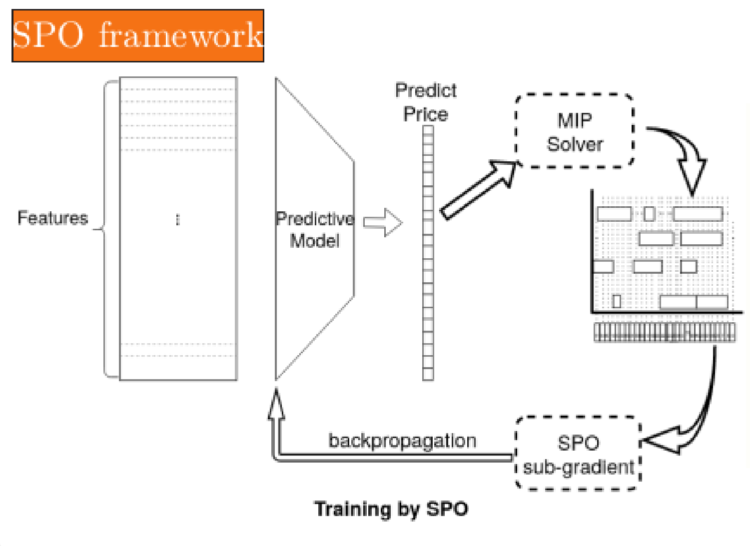
Algorithm 2: Stochastic gradient descent with SPO+ subgradient

Input : training data $\mathcal{D} = \{X, y\}_{i=1}^{n}$, architecture g, learning rate γ
1 initialize θ (neural network weights of g)
2 **for** *epochs* **do**
3   **for** *batches* **do**
4   sample batch $(X, y) \sim \mathcal{D}$
5         $\hat{y} \leftarrow g(X, \theta)$ (forward pass: compute predictions)
6         $\bar{y} = y + 2(\hat{y} - y)$ // SPO+ trick, convex comb. of y and $\hat{y}$
7   Solve $sol = solver(\bar{y})$ // calls external solver
8   Use subgradient $\partial L = solver(y) - sol$
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?



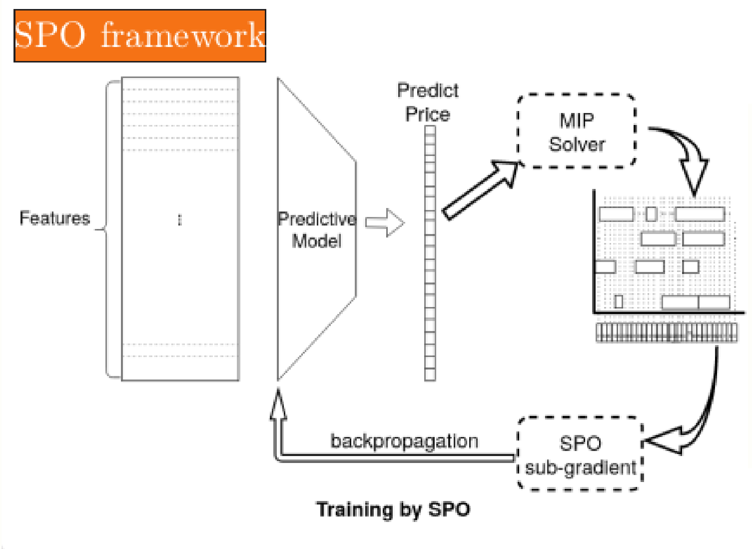
Challenge

To compute subgradient $v^*(2\hat{\theta} - \theta)$ must be solved repeatedly for each training instance

High training time & computation-expensive

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

Can we do the solving better?



Challenge

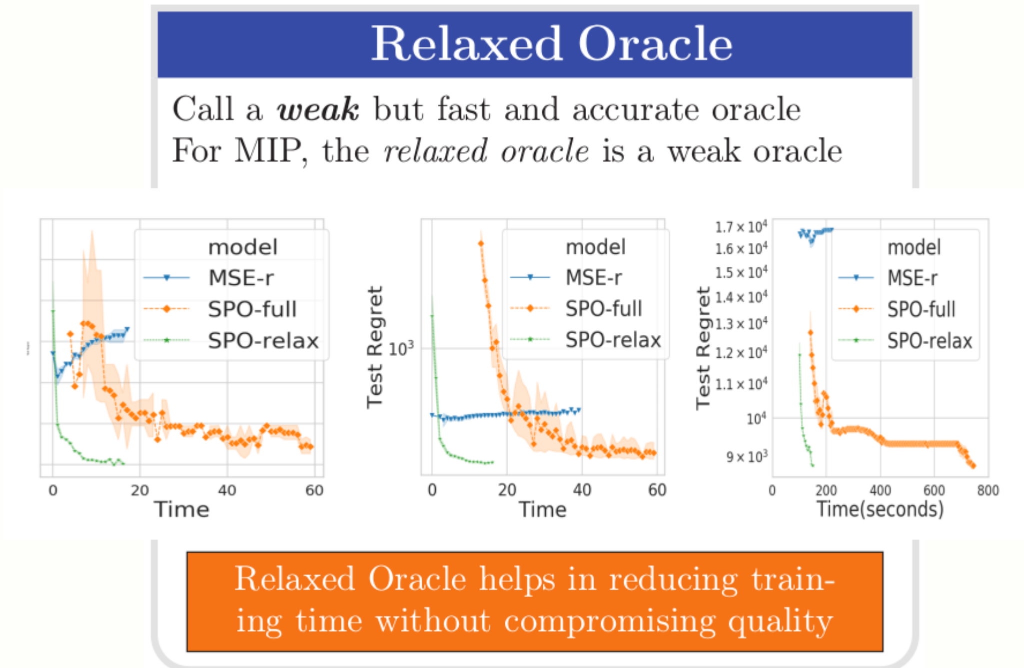
To compute subgradient $v^*(2\hat{\theta} - \theta)$ must be solved repeatedly for each training instance

High training time & computation-expensive

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

- 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?

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



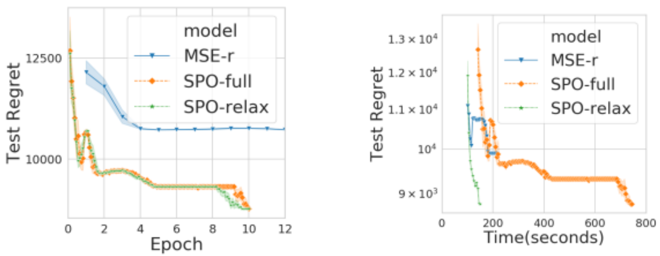
LP relaxations and warmstarts:

- Faster training time = possible to do wider grid search
- Faster training time = possible to scale up to larger problems

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

Relaxed Oracle

Call a ***weak*** but fast and accurate oracle
For MIP, the *relaxed oracle* is a weak oracle



(a) Epoch **(b)** Time

Relaxed Oracle helps in reducing training time without compromising quality

SPO-relax is scalable

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

Hard Instances (200 tasks on 10 machines)	Two-stage Approach				SPO-relax		
	2 epochs	4 epochs	6 epochs	8 epochs	2 hour	4 hour	6 hour
instance I	90,769	88,952	**86,059**	86,464	**72,662**	74,572	79,990
instance II	128,067	124,450	124,280	**123,738**	120,800	**110,944**	114,800
instance III	129,761	128,400	122,956	**119,000**	108,748	**102,203**	112,970
instance IV	135,398	132,366	132,167	**126,755**	109,694	99,657	**97,351**
instance V	122,310	**120,949**	122,116	123,443	118,946	**116,960**	118,460

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

But LP relaxation can be weak?

Solving MIP = repeatedly solving LP

- cutting plane algorithm: solve LP, cut fractional solution
- never cuts integral solutions

→ add Gomory and other cuts to the LP to strengthen it
(e.g. solve only root node of MIP, add those cuts)

→ tighter relaxation, still LP

[MIPaal: MIP as a layer, A. Ferber, B. Wilder, B. Dilkina, M. Tambe, AAAI2020]

Related work using deep learning (gradient descent)

<u>Differentiable task losses</u> 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 + 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
[2] Pogancic, Marin Vlastelica, et al. "Differentiation of Blackbox Combinatorial Solvers." ICLR. 2020
[3] Amos, Brandon, and J. Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks." ICML, 2017
[4] Wilder B, Dilkina B, Tambe M. “Melding the data-decisions pipeline: Decision-focused learning for comb. optimization.” AAA
[5] Mandi, Guns. “Interior Point Solving for LP-based prediction+optimisation.” NeurIPS, 2020

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^{nd} derivative is 0, so not invertible

Can we compute the gradient of a QP?

- » yes, through *implicit differentiation* of the KKT conditions

[B. Amos and Z. Kolter. "Optnet: Differentiable optimization as a layer in neural networks." ICML, 2017]

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

$$\max \theta^T x - \gamma ||x||_2^2 \;\text{ s.t. }\; Ax = b,\; Gx \leq h$$

(with some hyperparameter gamma)

→ can use Amos&Kolter's OptNet!

in case of submodular maximization, closed form special case!

[Wilder B, Dilkina B, Tambe M. “Melding the data-decisions pipeline: Decision-focused learning for comb. optimization.” AAAI, 2020]

Prediction + Optimisation for MIP

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

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

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

Lagrangian relaxation, does not restrict x >= 0:

$$\mathbb{L}(x, y; c) = f(c, x) + y^\top (b - Ax)$$

Interior point solving: adding a <u>logarithmic barrier</u>

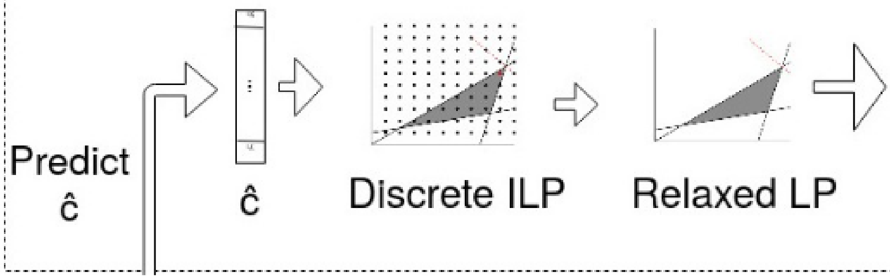
$$f(c, x) := c^\top x - \lambda \Big(\textstyle\sum_{i=1}^{k} ln(x_i) \Big)$$

- twice differentiable
- lambda is *automatically* decreased during barrier solving
- implicitly enforces x >= 0

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

Forward Pass

LP solving with barrier: Int. Point method

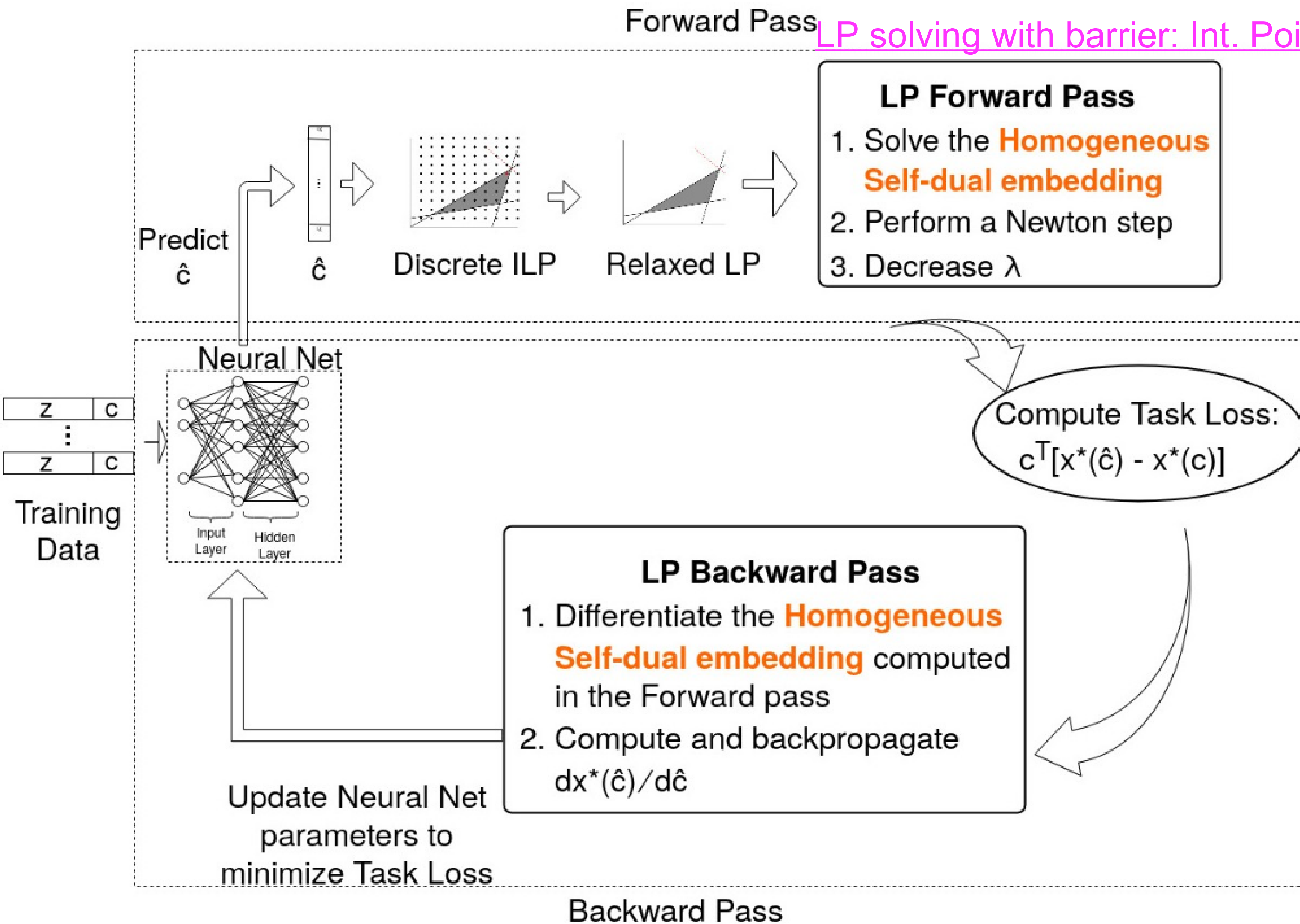


LP Forward Pass

1. Solve the **Homogeneous Self-dual embedding**
2. Perform a Newton step
3. Decrease λ

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

LP solving with barrier: Int. Point method



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

Interior Point Solving for LP-based prediction + optimisation

KKT vs HSD

λ / λ-cut-off	KKT, log barrier 10^{-1}	KKT, log barrier 10^{-3}	KKT, log barrier 10^{-10}	HSD, log barrier 10^{-1}	HSD, log barrier 10^{-3}	HSD, log barrier 10^{-10}
Regret	*14365*	14958	21258	**10774**	14620	21594

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

Compariosn with the state of the art

	Two-stage		QPTL		SPO		HSD,log barrier	
	0-layer	1-layer	0-layer	1-layer	0-layer	1-layer	0-layer	1-layer
MSE-loss	**745 (7)**	796 (5)	3516 (56)	2×10^9 (4×10^7)	3327 (485)	3955 (300)	2975 (620)	1.6×10^7 (1×10^7)
Regret	13322 (1458)	13590 (2021)	13652 (325)	13590 (288)	11073 (895)	12342 (1335)	**10774 (1715)**	11406 (1238)

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

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

Problem formulation

features   true cost vector

$$\underset{\omega}{\operatorname{argmin}}\ \mathbb{E}\left[regret\left(m(x_i;\omega), c_i\right)\right]$$

network params   predicted cost vector

Can be seen as a bi-level optimisation problem:

$$\underset{\omega}{\operatorname{argmin}} \frac{1}{N}\sum_{i=1}^{N} f\left(v_i, c\right) - f\left(v_i^*, c\right)$$

$$s.t. \quad v_i^* \in argmin_{v\in V} f\left(v, c_i\right) \qquad \forall i$$

$$v_i \in argmin_{v \in V} f\left(v, m(x_i;\omega)\right) \qquad \forall i$$

Challenges:

- argmin f is not unique
- V is implicit, exponential size
- argmin 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 >> 1
turn n-ary argmax into n-1 *pairwise* argmaxs!
(then subsample some)

For decision-focussed learning: $v^*(c) = \underset{v \in V}{\operatorname{argmin}} f(v, c)$

- define exponential distribution over V: $p(v|m(\omega, x)) = \frac{1}{Z} \exp\Big(- f(v, m(\omega, x))\Big)$
- contrastive loss for S subset V:
- partition function Z cancels out!!

$$\underset{\omega}{\operatorname{argmax}} \log \prod_i \prod_{v^s \in S} \frac{p\Big(v_i^\star | m(\omega, x_i)\Big)}{p\Big(v^s | m(\omega, x_i)\Big)} =$$

$$\mathcal{L}_{\text{NCE}} = \sum_i \sum_{v^s \in S} \Big(f\big(v_i^\star, m(\omega, x_i)\big) - f\big(v^s, m(\omega, x_i)\big)\Big)$$

Prediction + Optimisation for MIP and more

All current method 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 <u>inner approximation</u>?

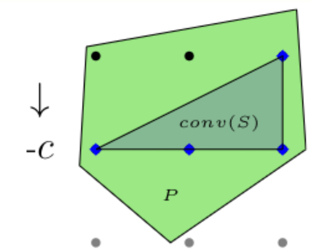


Figure 1: Representation of a solution cache (blue points) and the continuous relaxation (green area) of V.

[“Discrete solution pools and noise-contrastive estimation for predict-and-optimize” Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, Tias Guns, IJCAI 2021]

Prediction + Optimisation for MIP and more

All current method use a 'continuous approximation' 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 <u>inner approximation</u>?

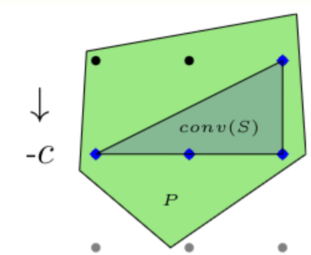


Figure 1: Representation of a solution cache (blue points) and the continuous relaxation (green area) of V.

<u>Inner approximation = cache of *known* solutions</u>

→ can replace 'argmin()' by 'linear pass' over *finite* nr of solutions! (any blackbox)

→ can use this cache as subsample 'S' in contrastive loss!

[“Discrete solution pools and noise-contrastive estimation for predict-and-optimize” Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, Tias Guns, arxiv 2020]

Prediction + Optimisation for MIP and more

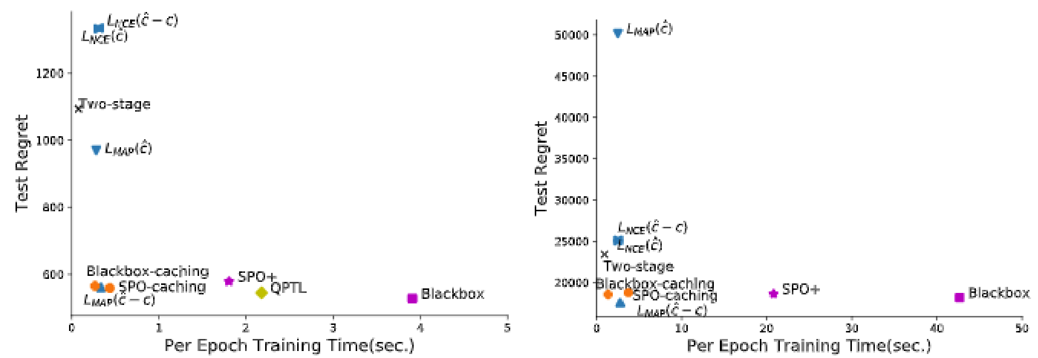
<u>Inner approximation = pool of known solutions</u>

→ can replace 'solver()' by 'linear pass' over finite solutions! (SPO+,BB)

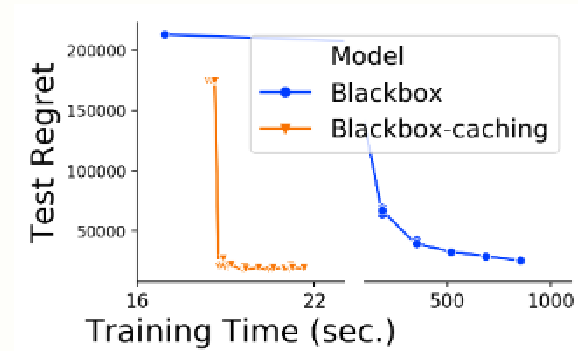
→ can use this cache as subsample 'S' in contrastive loss!

Main advantage: do not have to call a solver for each training instance!

Can 'grow' solution cache **FAST and GOOD**



(a) Knapsack-120 (b) Energy-3



[“Discrete solution pools and noise-contrastive estimation for predict-and-optimize” Maxime Mulamba, Jayanta Mandi, Michelangelo Diligenti, Michele Lombardi, Victor Bucarey, Tias Guns, arxiv 2020]

Related work using deep learning (gradient descent)

<u>Differentiable task losses</u> 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 + eps) perturbed predictions
- NCE[6]: contrastive loss function

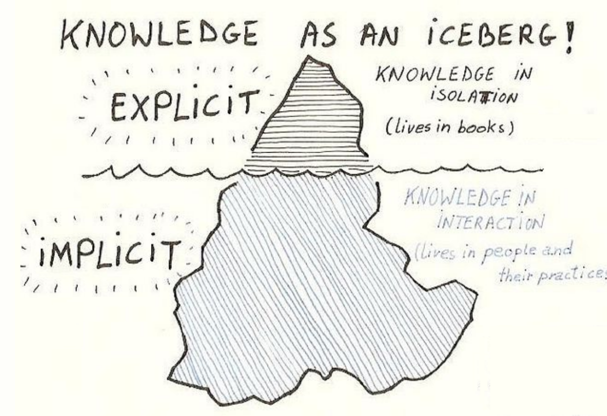
=> all these: inner approximation/solution caching for efficiency gain [6]

White box:

- 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
[2] Pogancic, Marin Vlastelica, et al. "Differentiation of Blackbox Combinatorial Solvers." ICLR. 2020
[3] Amos, Brandon, and J. Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks." ICML, 2017
[4] Wilder B, Dilkina B, Tambe M. “Melding the data-decisions pipeline: Decision-focused learning for comb. optimization.” AAAI, 2020
[5] Mandi, Guns. “Interior Point Solving for LP-based prediction+optimisation.” NeurIPS, 2020
[6] M. Mulamba, J. Mandi, M. Lombardi, M. Diligenti, V. Bucarey, T. Guns “Contrastive losses and solution caching for predict-and-optimize” IJCAI, 2021 to appear

Key take-aways:



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

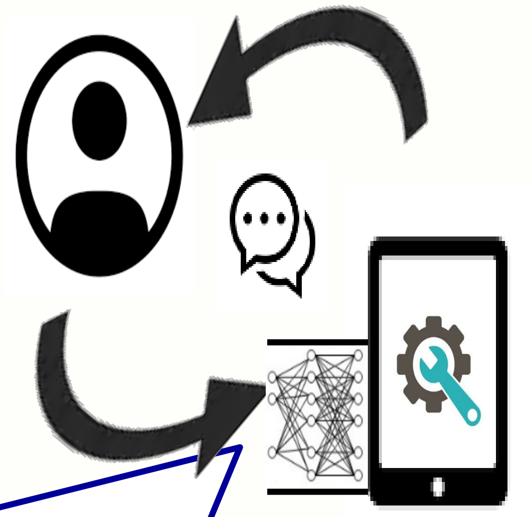
- Comb. optimisation inside neural loss becoming actually feasible
  → end-to-end hybrid prediction and optimisation
- dig into ML-side and Opt-side equally profoundly

Future Work

- Complexity of learned models vs. complexity of CP solving
- Scalability vs accuracy trade-off
- Interactive preference learning, multi-agent
- Other perception data (language, voice, camera)

- Wide range of applications (Industry 4.0, transport & more)

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
@TiasGuns

Hiring post-docs!