Predict-then-Optimize:
a tour of the state-of-the-art using PyEPO

Elias B. Khalil
Department of Mechanical & Industrial Engineering
SCALE AI Research Chair in Data-Driven Algorithms for Modern Supply Chains

GitHub

UNIVERSITY OF TORONTO

Bo Tang

For complete references to related work, please check our arXiv manuscript linked in the GitHub.

Optimization with a linear objective

$$\min_{w} \{ \underbrace{c^\intercal w}_{\text{Linear cost function}} : \underbrace{w \in \mathcal{S}}_{\text{Feasible set}} \}$$

Decision variables

Solution: use appropriate algorithm depending on type of feasible set (MILP, MIQCP, CP, custom algorithms...)

The n cost vectors could be completely unrelated

$$\min_{w} \{ c_1^\intercal w : w \in \mathcal{S} \}$$

$$\min_{w} \{ c_2^\intercal w : w \in \mathcal{S} \}$$

Same Decision variables

Same
Feasible set

$$\min_{w} \{ c_n^\intercal w : w \in \mathcal{S} \}$$

Instance feature vector (Observed)

$$\min_{w}\{c_1^\intercal w : w \in \mathcal{S}\} \qquad x_1 \in \mathbb{R}^p$$

$$\min_{w}\{c_2^\intercal w : w \in \mathcal{S}\} \qquad x_2$$

Same Decision variables

Same Feasible set

$$\min_{w}\{c_n^\intercal w : w \in \mathcal{S}\} \qquad x_n$$

Function g is parametrized by a vector θ.

$$c_i = g(x_i; \theta) \in \mathbb{R}^d$$

Instance feature vector (Observed)

$$\min_w \{c_1^\intercal w : w \in \mathcal{S}\}$$

$$x_1 \in \mathbb{R}^p$$

$$\min_w \{c_2^\intercal w : w \in \mathcal{S}\}$$

$$x_2$$

$\underbrace{w}$ **Same** Decision variables

$\underbrace{\in \mathcal{S}}$ **Same** Feasible set

$$\min_w \{c_n^\intercal w : w \in \mathcal{S}\}$$

$$x_n$$

If you know θ then you can evaluate $g(x_i; \theta)$ to obtain c_i and optimize with your favorite method

$$c_i = g(x_i; \theta) \in \mathbb{R}^d$$

Instance feature vector (Observed)

$$\min_w \{c_1^\intercal w : w \in \mathcal{S}\}$$

$$x_1 \in \mathbb{R}^p$$

$$\min_w \{c_2^\intercal w : w \in \mathcal{S}\}$$

$$x_2$$

$$\min_w \{c_n^\intercal w : w \in \mathcal{S}\}$$

$$x_n$$

Same Decision variables

Same Feasible set

If you know θ then you can evaluate $g(x_i; \theta)$ to obtain c_i and optimize with your favorite method

What if you don't know θ and you only observe x_i?

$$\hat{c}_i = g(x_i; \theta) \in \mathbb{R}^d$$

Instance feature vector (Observed)

$$\min_w \{\hat{c}_1^\intercal w : w \in \mathcal{S}\}$$

$$x_1 \in \mathbb{R}^p$$

$$\min_w \{\hat{c}_2^\intercal w : w \in \mathcal{S}\}$$

$$x_2$$

Same Decision variables

Same Feasible set

$$\min_w \{\hat{c}_n^\intercal w : w \in \mathcal{S}\}$$

$$x_n$$

Assume these n instances are from the past, i.e., "training" instances

$$\hat{c}_i = g(x_i; \theta) \in \mathbb{R}^d$$

$$\min_w \{\hat{c}_1^\top w : w \in \mathcal{S}\}$$

$$\min_w \{\hat{c}_2^\top w : w \in \mathcal{S}\}$$

Same Decision variables

Same Feasible set

$$\min_w \{\hat{c}_n^\top w : w \in \mathcal{S}\}$$

True cost vector (Observed <u>for training instances only</u>)

$$c_1 \in \mathbb{R}^d$$

$$c_2$$

$$c_n$$

Instance feature vector (Observed)

$$x_1 \in \mathbb{R}^p$$

$$x_2$$

$$x_n$$

Assume these n instances are from the past, i.e., "training" instances

Goal: given training set $\mathcal{D} = \{(x_i, c_i)\}_{i=1}^{n}$, learn θ that approximates groundtruth c_i vectors "well"

$$\hat{c}_i = g(x_i; \theta) \in \mathbb{R}^d$$

True cost vector (Observed <u>for training instances only</u>)

Instance feature vector (Observed)

$$\min_w \{\hat{c}_1^\top w : w \in \mathcal{S}\} \qquad c_1 \in \mathbb{R}^d \qquad x_1 \in \mathbb{R}^p$$

$$\min_w \{\hat{c}_2^\top w : w \in \mathcal{S}\} \qquad c_2 \qquad x_2$$

Same Decision variables

Same Feasible set

$$\min_w \{\hat{c}_n^\top w : w \in \mathcal{S}\} \qquad c_n \qquad x_n$$

Predict-then-Optimize

Training

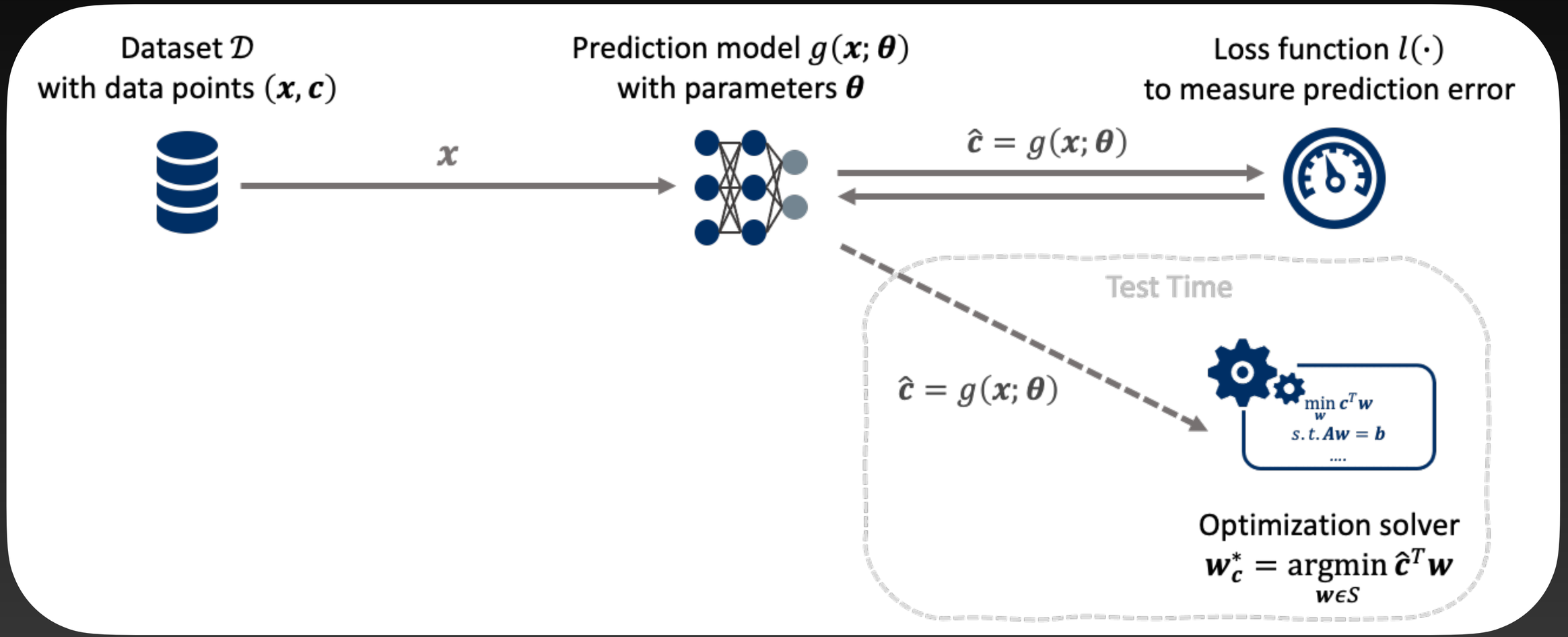
Dataset $\mathcal{D}$
with data points $(\boldsymbol{x}, \boldsymbol{c})$

Prediction model $g(\boldsymbol{x}; \boldsymbol{\theta})$
with parameters $\boldsymbol{\theta}$

Loss function $l(\cdot)$
to measure prediction error

Predict-then-Optimize

Training and Test-time inference



Predict-then-Optimize

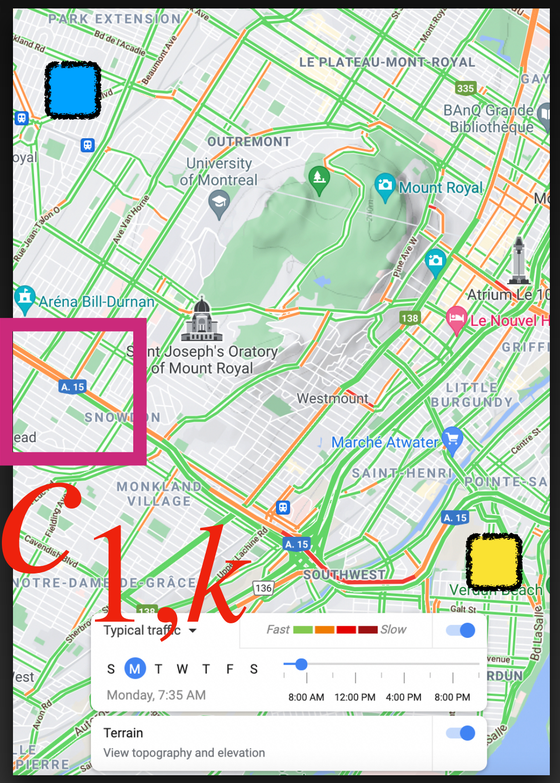
s-t shortest path: training data

x_1 8 am

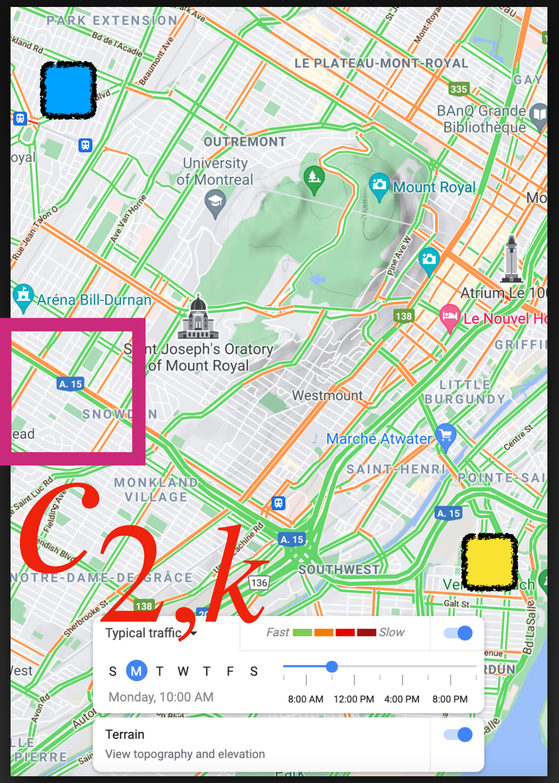
x_2 10 am

x_3 12 pm

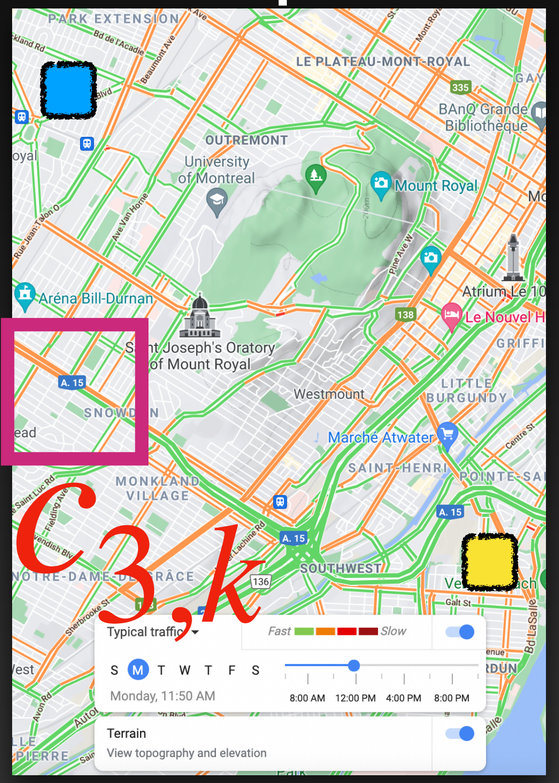
x_4 4 pm



$c_{1,k}$



$c_{2,k}$



$c_{3,k}$

$c_{4,k}$

Google Maps, Montréal, Quebéc, Canada

Predict-then-Optimize

s-t shortest path: full pipeline

Dataset $\mathcal{D}$
with data points $(\boldsymbol{x}, \boldsymbol{c})$

Prediction model $g(\boldsymbol{x}; \boldsymbol{\theta})$
with parameters $\boldsymbol{\theta}$

Loss function $l(\cdot)$
to measure prediction error

Travel time prediction model (regression) for each road segment

Predicted travel times <=> s-t shortest path cost vector

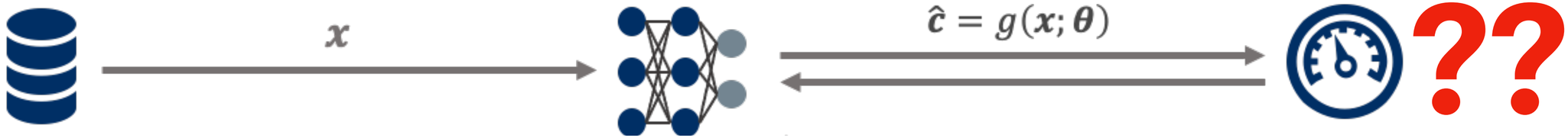
Predict-then-Optimize

s-t shortest path: full pipeline

Dataset $\mathcal{D}$
with data points $(\boldsymbol{x}, \boldsymbol{c})$

Prediction model $g(\boldsymbol{x}; \boldsymbol{\theta})$
with parameters $\boldsymbol{\theta}$

Loss function $l(\cdot)$
to measure prediction error



Travel time prediction model (regression) for each road segment

Predicted travel times <=> s-t shortest path cost vector

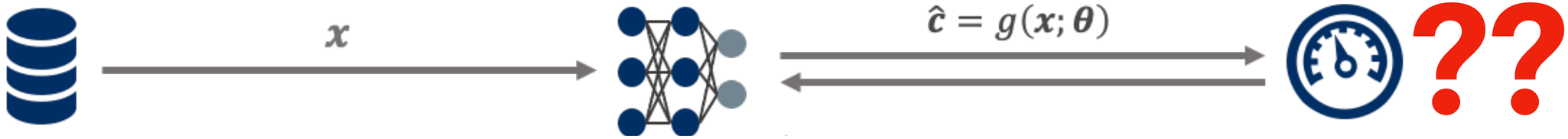
Predict-then-Optimize

s-t shortest path: full pipeline

Dataset $\mathcal{D}$
with data points $(\boldsymbol{x}, \boldsymbol{c})$

Prediction model $g(\boldsymbol{x}; \boldsymbol{\theta})$
with parameters $\boldsymbol{\theta}$

Loss function $l(\cdot)$
to measure prediction error



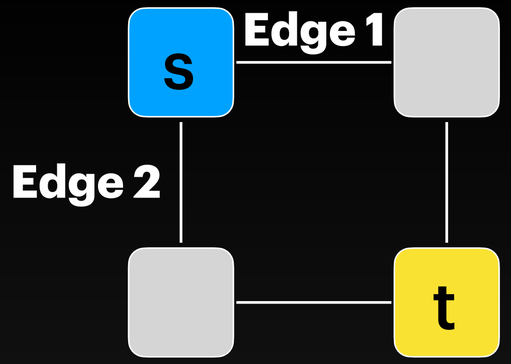
Travel time prediction model (regression) for each road segment

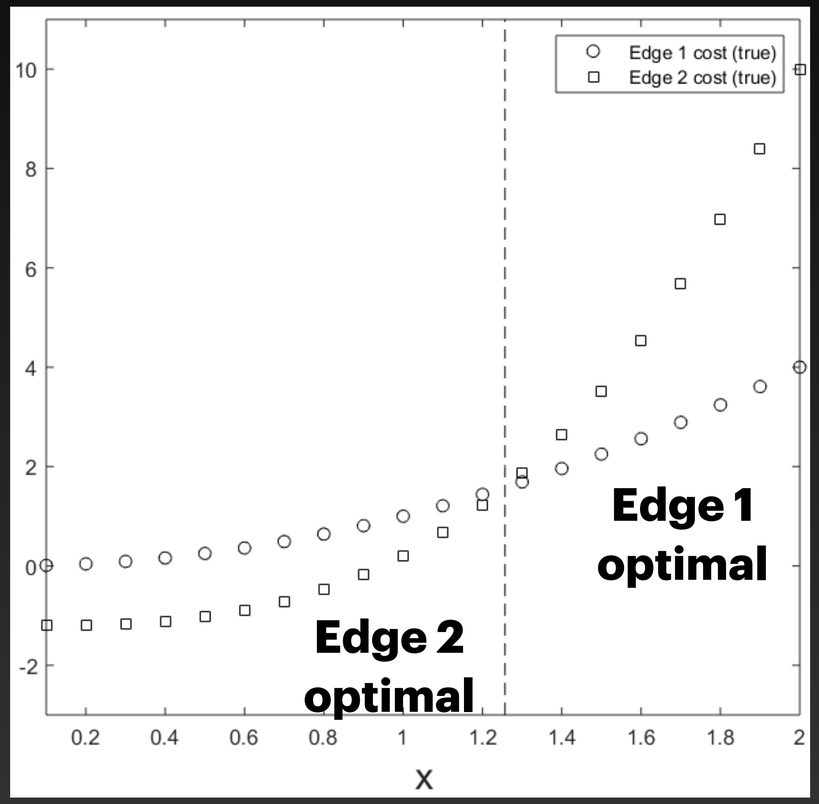
Predicted travel times <=> s-t shortest path cost vector

Squared error between c_i **and** $\hat{c}_i$

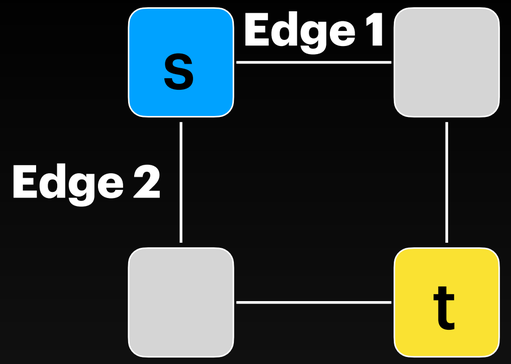
Why regression on cost coefficients may fail

Vertical axis: edge cost. Horizontal axis: feature x





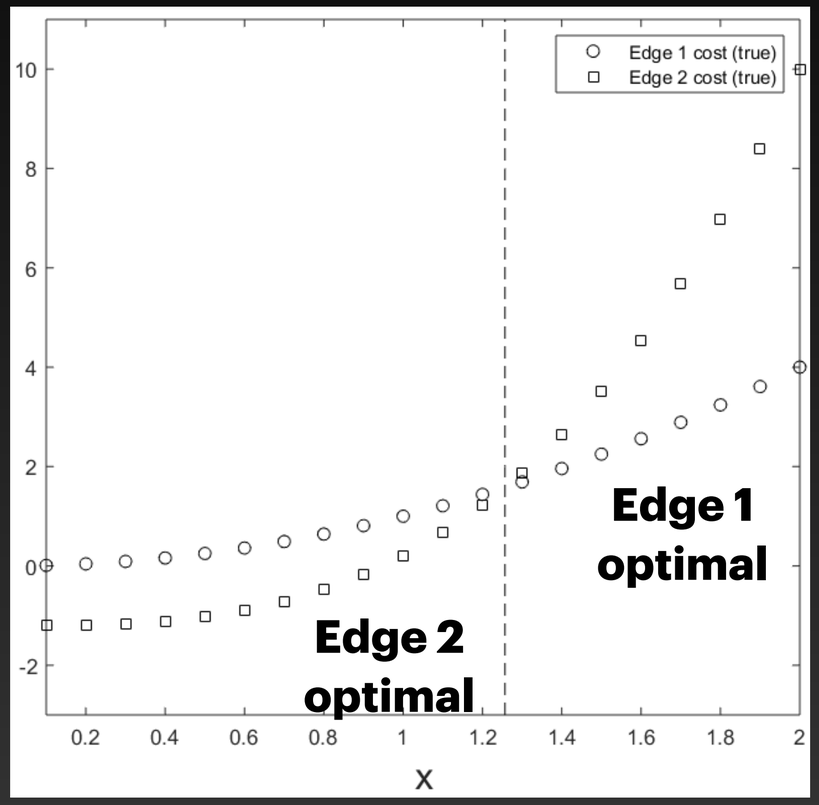
Figures from Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

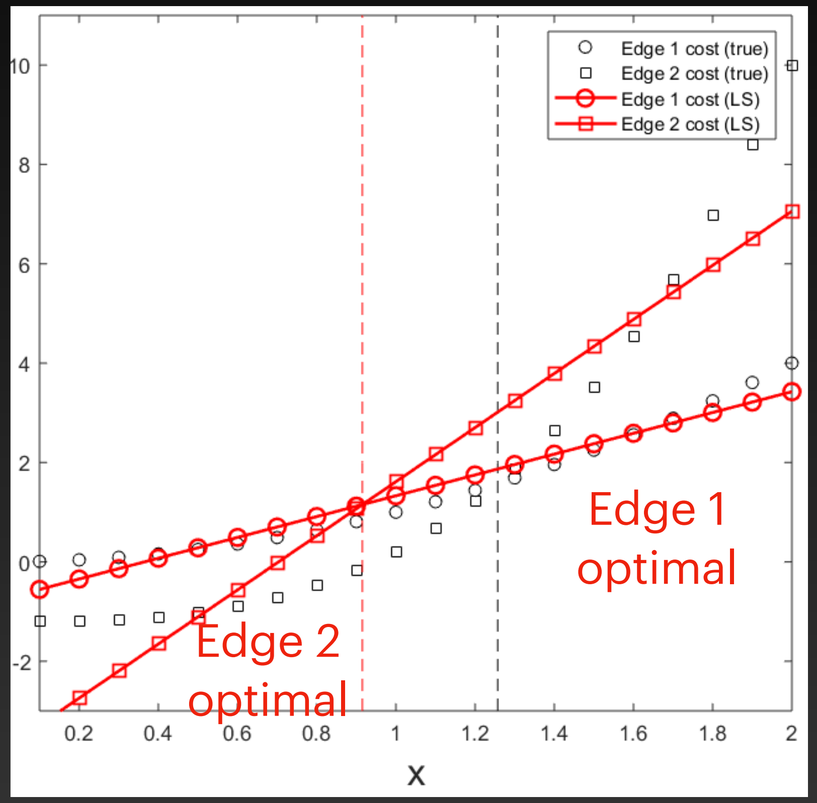


Why regression on cost coefficients may fail

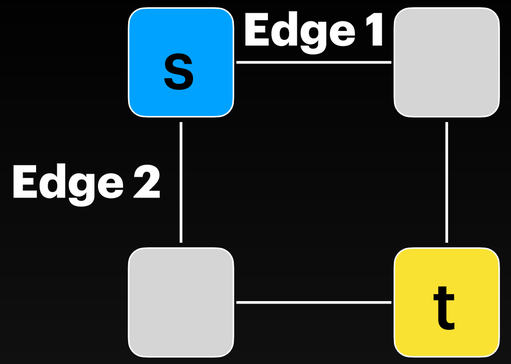
Vertical axis: edge cost. Horizontal axis: feature x

Least-squares regression on dataset of (x, c) pairs



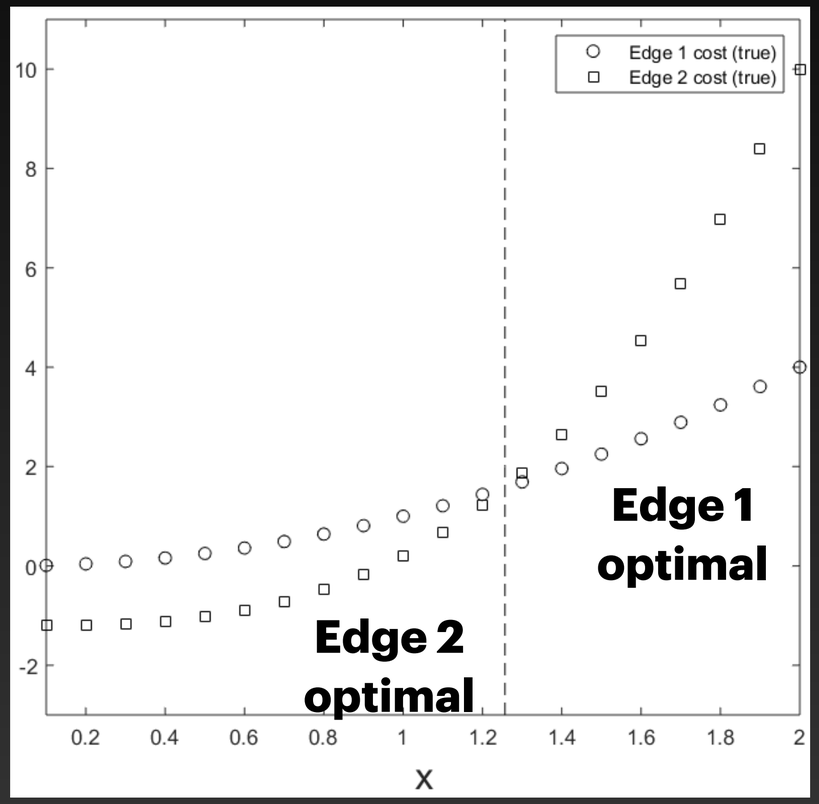


Figures from Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

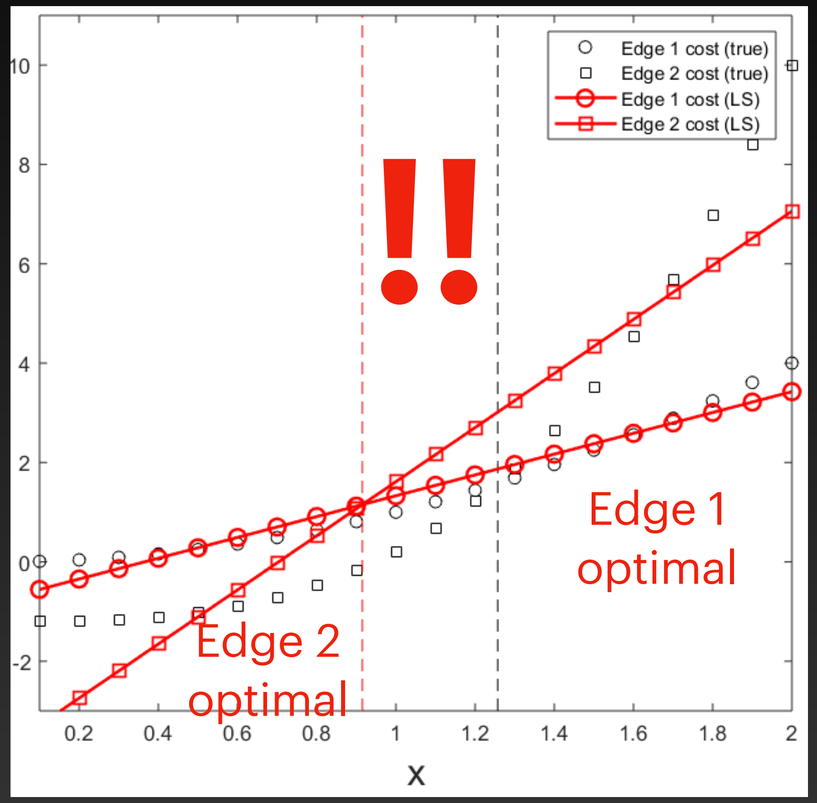


Why regression on cost coefficients may fail

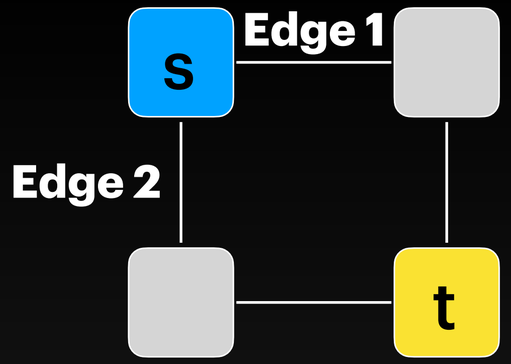
Vertical axis: edge cost. Horizontal axis: feature x



Least-squares regression on dataset of (x, c) pairs



Figures from Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

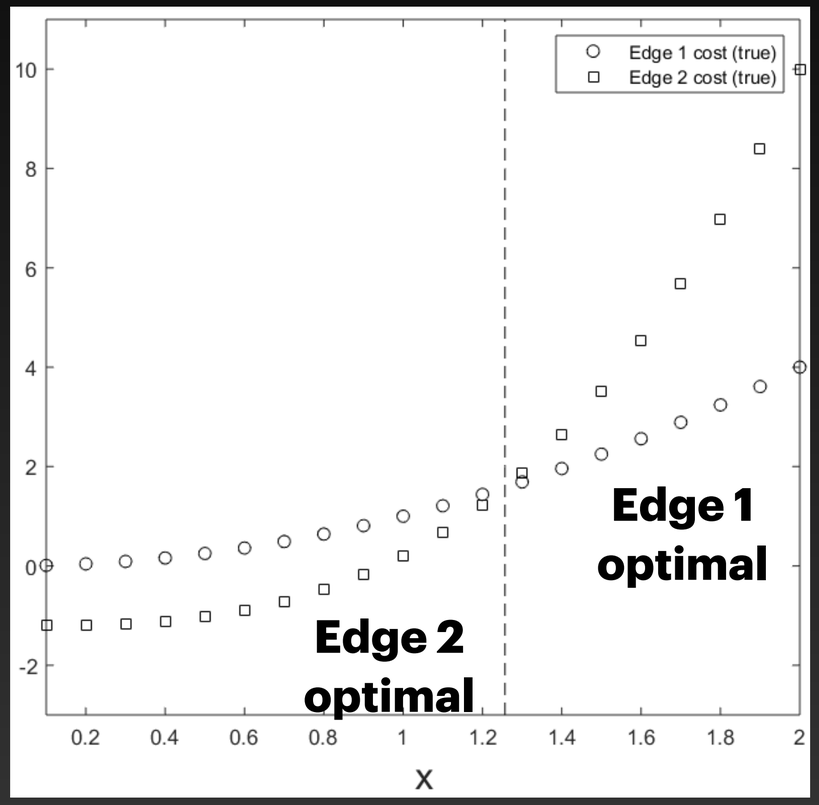


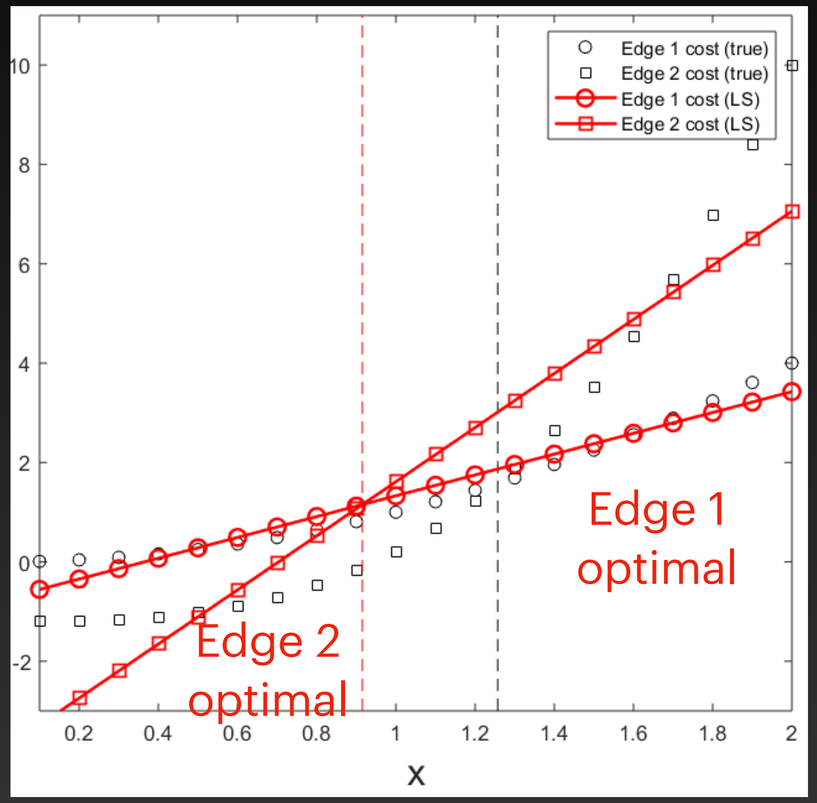
Why regression on cost coefficients may fail

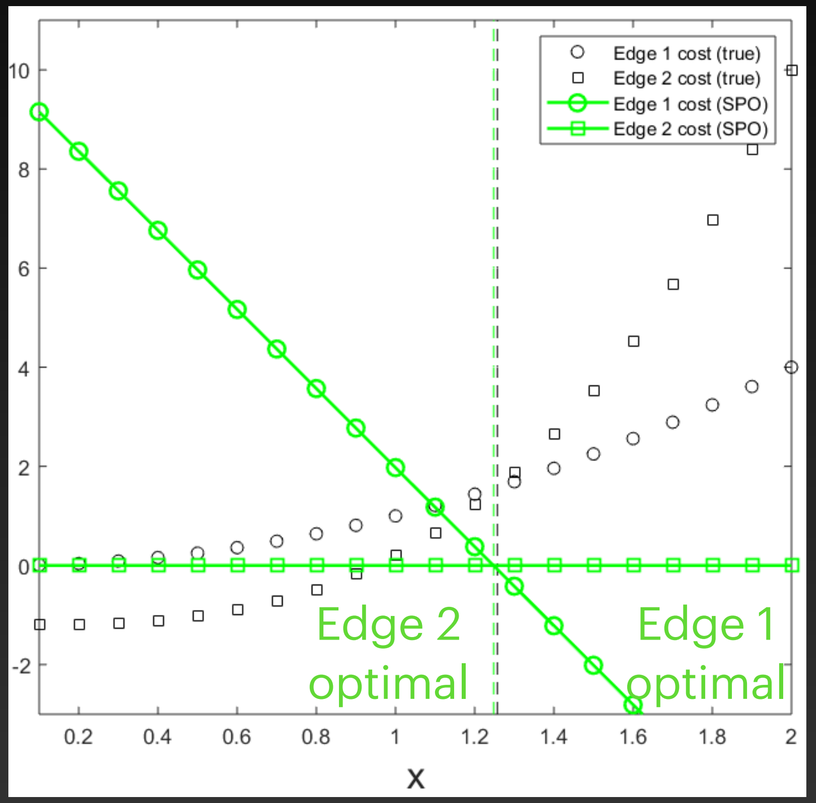
Vertical axis: edge cost. Horizontal axis: feature x

Least-squares regression on dataset of (x, c) pairs

"Smart Predict then Optimize"



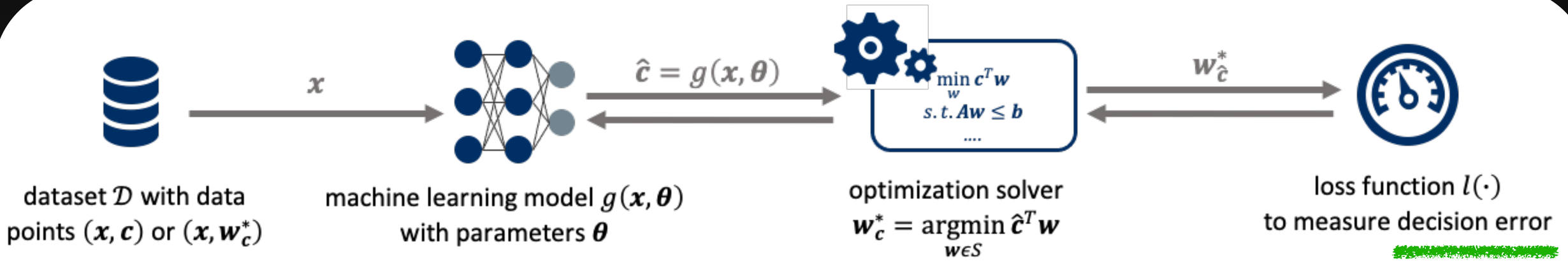




Figures from Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

End-to-End Predict-then-Optimize

Training

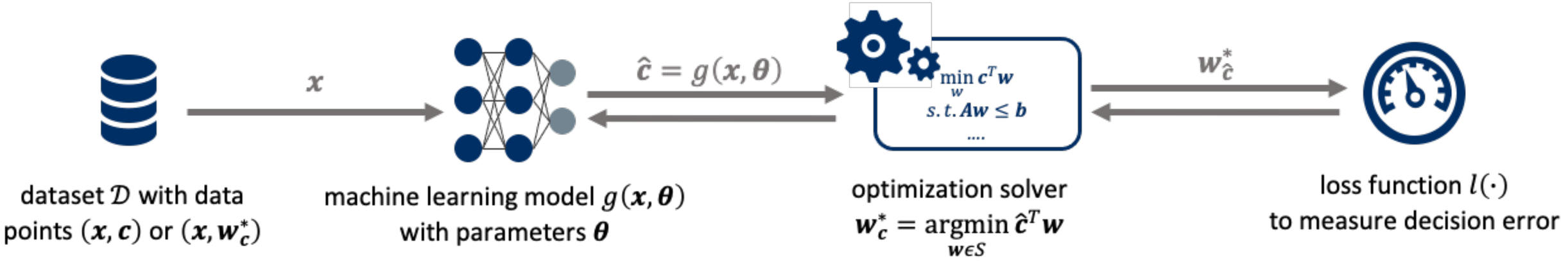


Decision error

$$l_{\text{Regret}}(\hat{c}, c) = c^\intercal \underbrace{w^*(\hat{c})}_{} - \underbrace{z^*(c)}_{}$$

Optimal solution if you optimize with $\hat{c}$

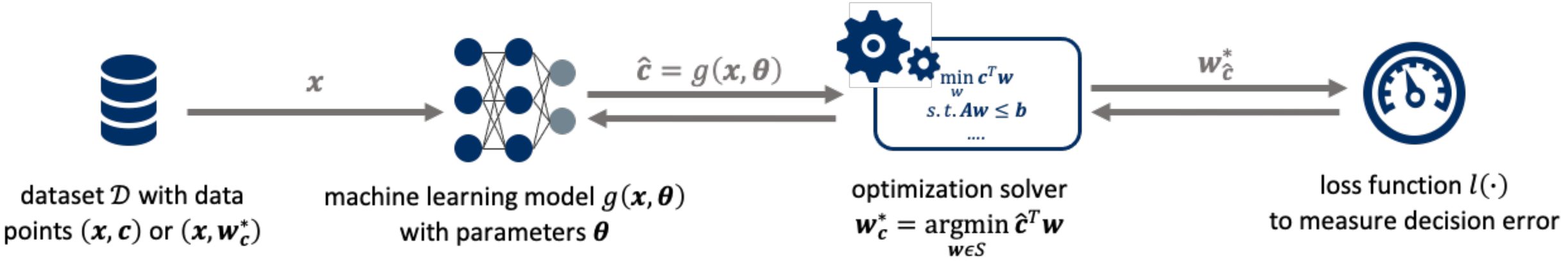
Optimal value with true costs c



Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \operatorname{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
  end for
end for



Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \operatorname{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
  end for
end for

??

$$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

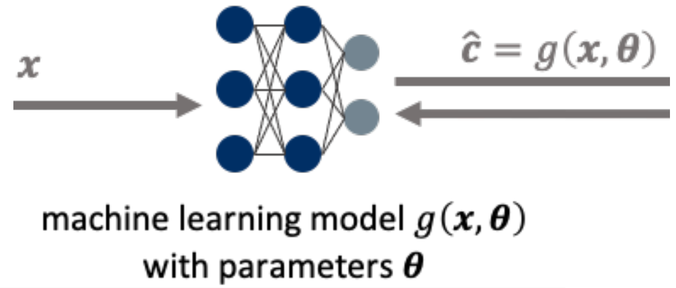
Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient ??
  end for
end for

Easy: gradient of predicted costs to model parameters!

$$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$



Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
  end for
end for

??

Trickier: how does the decision loss vary with the predicted costs?

Easy: gradient of predicted costs to model parameters!

$$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient ??
  end for
end for

Trickier: how does the decision loss vary with the predicted costs?

$$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

$$l_{\text{Regret}}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = \boldsymbol{c}^\intercal \boldsymbol{w}^*(\hat{\boldsymbol{c}}) - z^*(\boldsymbol{c})$$

Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient ??
  end for
end for

Trickier: how does the decision loss vary with the predicted costs?

Crux: how does the optimum change with the predicted costs?

$$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

$$l_{\text{Regret}}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = \boldsymbol{c}^{\intercal} \boldsymbol{w}^*(\hat{\boldsymbol{c}}) - z^*(\boldsymbol{c})$$

Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

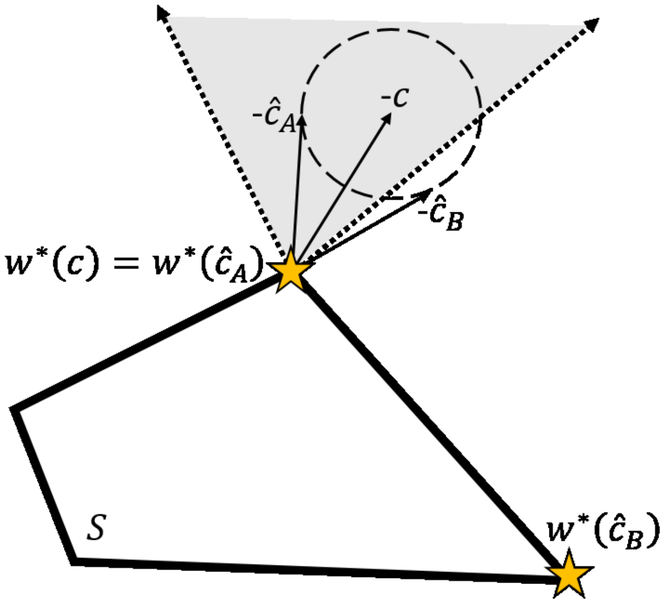
Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^{\intercal} \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
  end for
end for

SPO+: a principled and effective method

Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

$$l_{\text{Regret}}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = \boldsymbol{c}^{\intercal}\boldsymbol{w}^*(\hat{\boldsymbol{c}}) - z^*(\boldsymbol{c})$$

is not differentiable w.r.t. $\hat{c}$



Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^{\intercal}\boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
  end for
end for

SPO+: a principled and effective method

Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

$$l_{\text{Regret}}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = \boldsymbol{c}^\intercal \boldsymbol{w}^*(\hat{\boldsymbol{c}}) - z^*(\boldsymbol{c})$$

is not differentiable w.r.t. $\hat{c}$

$$l_{SPO+}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = -\min_{\boldsymbol{w} \in S}\{(2\hat{\boldsymbol{c}} - \boldsymbol{c})^\intercal \boldsymbol{w}\} + 2\hat{\boldsymbol{c}}^\intercal \boldsymbol{w}^*(\boldsymbol{c}) - z^*(\boldsymbol{c})$$

is a convex upper bound on regret and has subgradients

Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$

Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
for epochs **do**
  for each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
    Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
    Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
    Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
    Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
    Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
  end for
end for

SPO+: a principled and effective method

Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

$$l_{\text{Regret}}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = \boldsymbol{c}^\intercal \boldsymbol{w}^*(\hat{\boldsymbol{c}}) - z^*(\boldsymbol{c})$$

is not differentiable w.r.t. $\hat{c}$

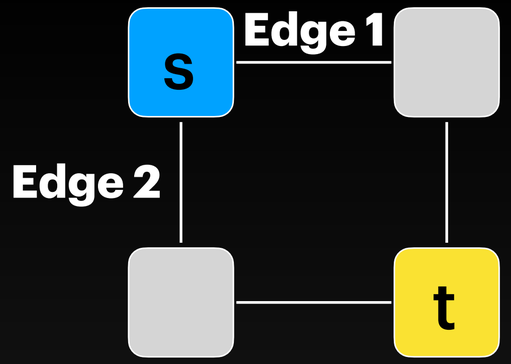
$$l_{SPO+}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = -\min_{\boldsymbol{w} \in S}\{(2\hat{\boldsymbol{c}} - \boldsymbol{c})^\intercal \boldsymbol{w}\} + 2\hat{\boldsymbol{c}}^\intercal \boldsymbol{w}^*(\boldsymbol{c}) - z^*(\boldsymbol{c})$$

is a convex upper bound on regret and has subgradients

Algorithm 1 End-to-end Gradient Descent

Require: coefficient matrix $\boldsymbol{A}$, right-hand side $\boldsymbol{b}$, data $\mathcal{D}$
1: Initialize predictor parameters $\boldsymbol{\theta}$ for predictor $g(\boldsymbol{x}; \boldsymbol{\theta})$
2: **for** epochs **do**
3: **for** each batch of training data $(\boldsymbol{x}, \boldsymbol{c})$ **do**
4: Sample batch of the cost vectors $\boldsymbol{c}$ with the corresponding features $\boldsymbol{x}$
5: Predict cost using predictor $\hat{\boldsymbol{c}} := g(\boldsymbol{x}; \boldsymbol{\theta})$
6: Forward pass to compute optimal solution $\boldsymbol{w}^*(\hat{\boldsymbol{c}}) := \text{argmin}_{\boldsymbol{w} \in S} \hat{\boldsymbol{c}}^\intercal \boldsymbol{w}$
7: Forward pass to compute decision loss $l(\hat{\boldsymbol{c}}, \cdot)$
8: Backward pass from loss $l(\hat{\boldsymbol{c}}, \cdot)$ to update parameters $\boldsymbol{\theta}$ with gradient
9: **end for**
10: **end for**

Computational overhead: To compute SPO+ loss, we need to solve an optimization problem in the forward pass; this is shared with other methods.

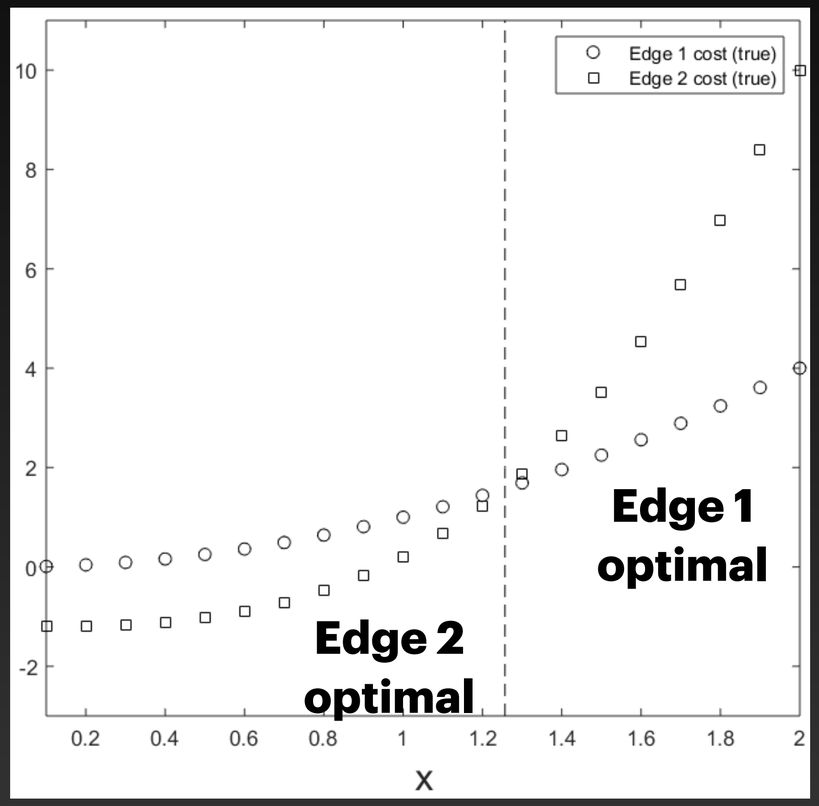


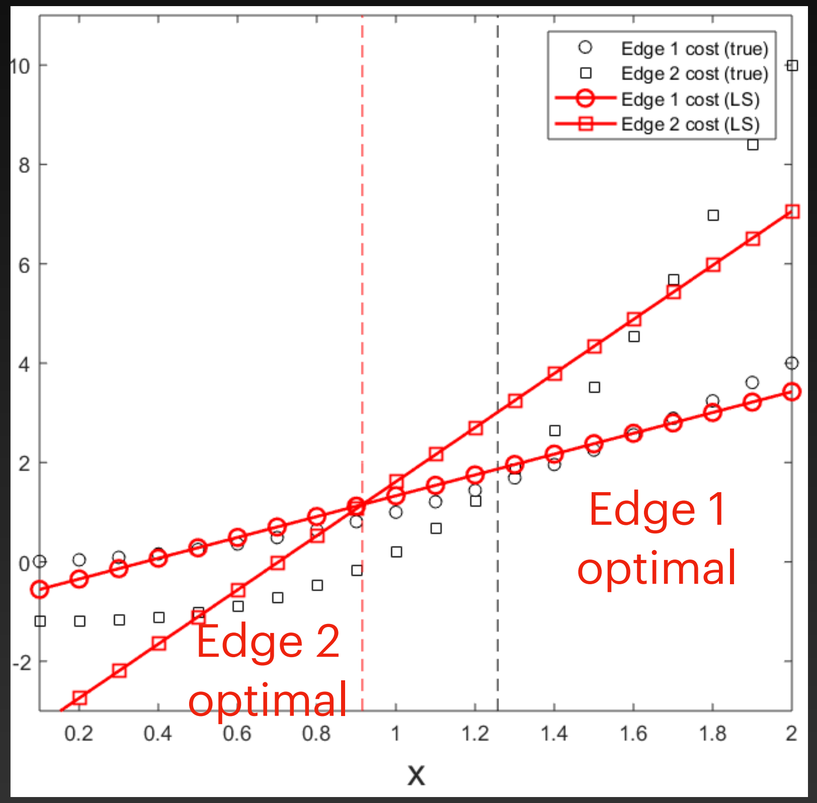
Why regression on cost coefficients may fail

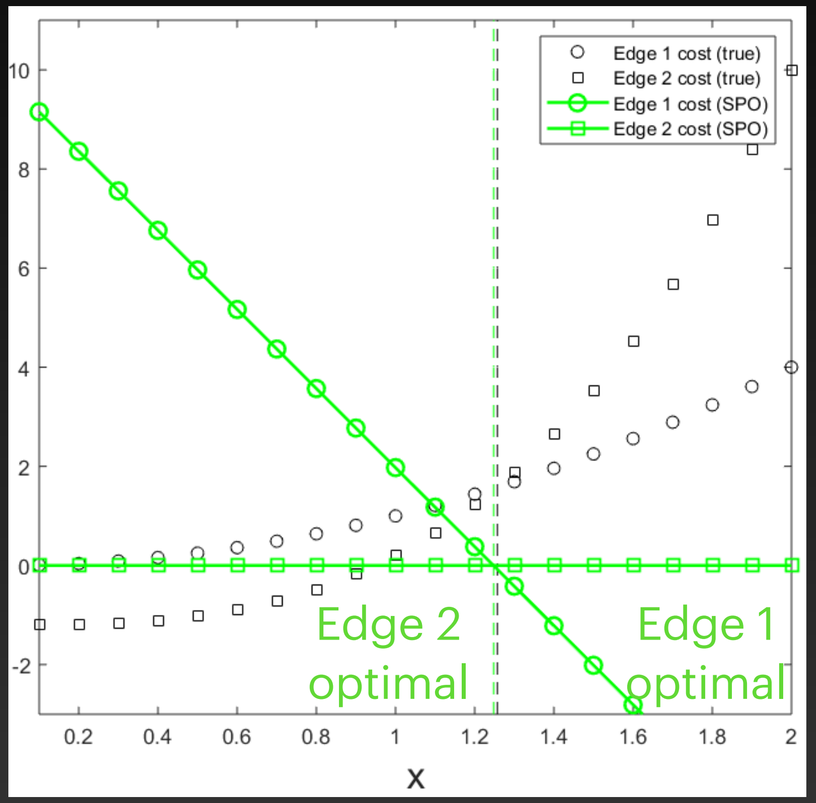
Vertical axis: edge cost. Horizontal axis: feature x

Least-squares regression on dataset of (x, c) pairs

SPO+







Figures from Elmachtoub, Adam N., and Paul Grigas. "Smart "predict, then optimize"." *Management Science* 68.1 (2022): 9-26.

Predict-then-Optimize

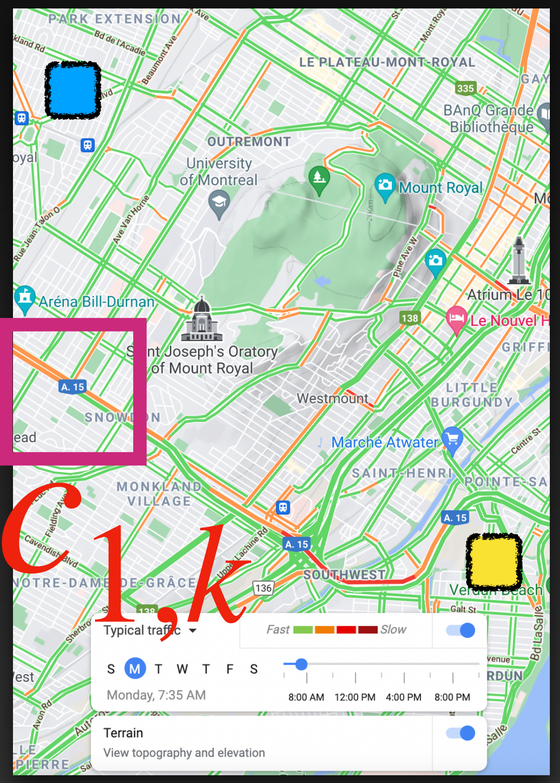
s-t shortest path: training data

x_1 8 am

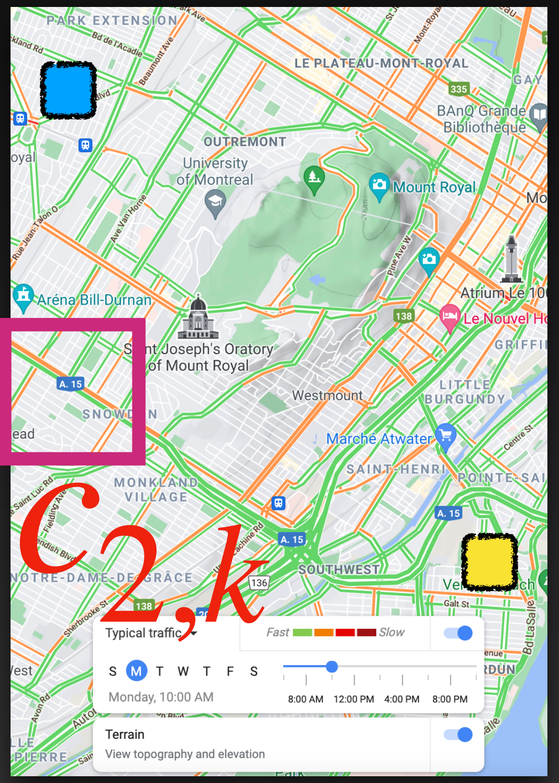
x_2 10 am

x_3 12 pm

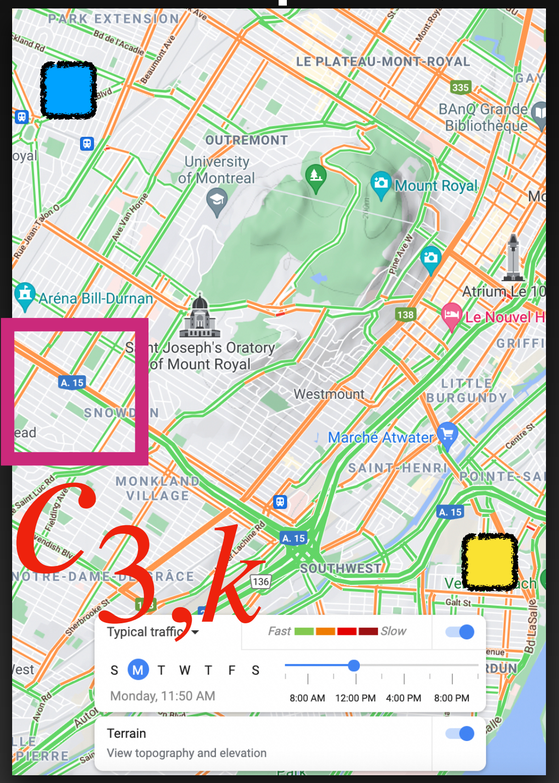
x_4 4 pm



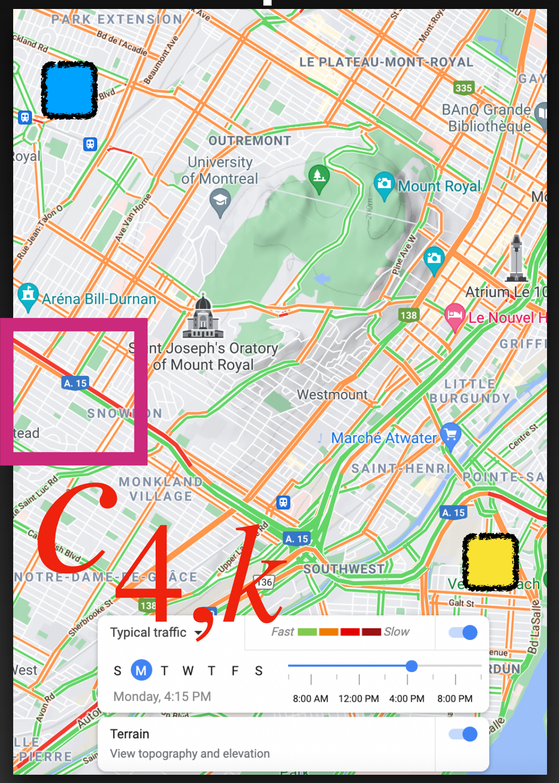
$c_{1,k}$



$c_{2,k}$



$c_{3,k}$



$c_{4,k}$

Google Maps, Montréal, Quebéc, Canada

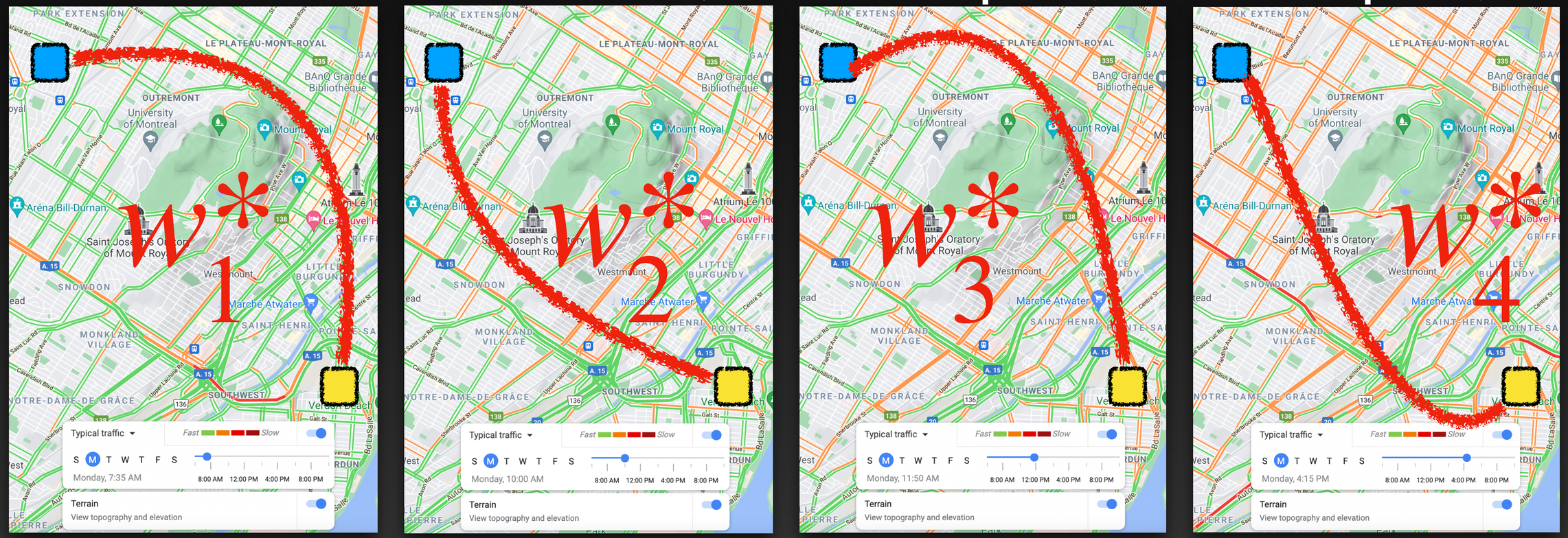
Predict-then-Optimize

s-t shortest path: training data

What if the historical cost vectors were unavailable, but **the optimal solutions** were?

$$\mathcal{D} = \{(x_i, w_i^*)\}_{i=1}^n$$

x_1 8 am x_2 10 am x_3 12 pm x_4 4 pm



Google Maps, Montréal, Quebéc, Canada

PFYL: Perturbed Fenchel-Young Loss

Berthet, Quentin, et al. "Learning with differentiable perturbed optimizers." NeurIPS 2020.

Recall the required gradient: $$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

A random perturbation $(\sigma\xi_k)$ **of predicted costs** $\hat{c}$

$$\frac{\partial l_{\mathrm{FY}}(\hat{\boldsymbol{c}}, \boldsymbol{w}^*(\boldsymbol{c}))}{\partial \hat{\boldsymbol{c}}} \approx \boldsymbol{w}^*(\boldsymbol{c}) - \frac{1}{K} \sum_{\kappa}^{K} \underset{\boldsymbol{w} \in S}{\operatorname{argmin}} \{(\overbrace{\hat{\boldsymbol{c}} + \sigma \boldsymbol{\xi}_\kappa}^{})^\intercal \boldsymbol{w}\}$$

Average optimal solution over K randomly perturbed costs

PFYL: Perturbed Fenchel-Young Loss

Berthet, Quentin, et al. "Learning with differentiable perturbed optimizers." NeurIPS 2020.

Recall the required gradient: $$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

A random perturbation $(\sigma\xi_k)$ of predicted costs $\hat{c}$

$$\frac{\partial l_{\mathrm{FY}}(\hat{\boldsymbol{c}}, \boldsymbol{w}^*(\boldsymbol{c}))}{\partial \hat{\boldsymbol{c}}} \approx \boldsymbol{w}^*(\boldsymbol{c}) - \frac{1}{K} \sum_{\kappa}^{K} \underset{\boldsymbol{w} \in S}{\operatorname{argmin}} \{(\overbrace{\hat{\boldsymbol{c}} + \sigma \boldsymbol{\xi}_\kappa}^{})^\intercal \boldsymbol{w}\}$$

Notice the true costs c do not appear here.

Average optimal solution over K randomly perturbed costs

PFYL: Perturbed Fenchel-Young Loss

Berthet, Quentin, et al. "Learning with differentiable perturbed optimizers." NeurIPS 2020.

Recall the required gradient: $$\frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \boldsymbol{\theta}} = \frac{\partial l(\hat{\boldsymbol{c}}, \cdot)}{\partial \hat{\boldsymbol{c}}} \frac{\partial \hat{\boldsymbol{c}}}{\partial \boldsymbol{\theta}}$$

A random perturbation $(\sigma\xi_k)$ **of predicted costs** $\hat{c}$

$$\frac{\partial l_{\mathrm{FY}}(\hat{\boldsymbol{c}}, \boldsymbol{w}^*(\boldsymbol{c}))}{\partial \hat{\boldsymbol{c}}} \approx \boldsymbol{w}^*(\boldsymbol{c}) - \frac{1}{K} \sum_{\kappa}^{K} \underset{\boldsymbol{w} \in S}{\operatorname{argmin}} \{(\overbrace{\hat{\boldsymbol{c}} + \sigma \boldsymbol{\xi}_\kappa}^{})^\intercal \boldsymbol{w}\}$$

Notice the true costs c do not appear here.

Computational overhead: To compute the PFYL gradient, we need to solve K optimization problems.

Average optimal solution over K randomly perturbed costs

Avoiding solver calls by Learning to Rank

Mandi, Jayanta, et al. "Decision-focused learning: through the lens of learning to rank." ICML, 2022.
Mulamba, Maxime, et al. "Contrastive Losses and Solution Caching for Predict-and-Optimize." IJCAI, 2021.

Recall that both SPO+ and PFYL made one or more calls to the solver in each forward pass!

	w1	w2	w3	Objective using prediction A	Objective using prediction B	True Objective
Sol 1	0	1	0	1	3	1
Sol 2	1	0	0	3	2	2
Sol 3	1	1	1	2	1	3

Avoiding solver calls by Learning to Rank

Mandi, Jayanta, et al. "Decision-focused learning: through the lens of learning to rank." ICML, 2022.
Mulamba, Maxime, et al. "Contrastive Losses and Solution Caching for Predict-and-Optimize." IJCAI, 2021.

Somewhat better...

Terrible solution ranking!

	w1	w2	w3	Objective using prediction A	Objective using prediction B	True Objective
Sol 1	0	1	0	1	3	1
Sol 2	1	0	0	3	2	2
Sol 3	1	1	1	2	1	3

Avoiding solver calls by Learning to Rank

Mandi, Jayanta, et al. "Decision-focused learning: through the lens of learning to rank." ICML, 2022.
Mulamba, Maxime, et al. "Contrastive Losses and Solution Caching for Predict-and-Optimize." IJCAI, 2021.

Somewhat better...

Terrible solution ranking!

	w1	w2	w3	Objective using prediction	Objective using prediction	True Objective
Sol 1						1
Sol 2						2
Sol 3	1	1	1	2	1	3

Key Idea: Learn to predict coefficients that lead to good rankings of a set of collected solutions!

Predict-then-Optimize:
a tour of the state-of-the-art using PyEPO

Manuscript describing PyEPO, the literature, and extensive experiments: https://arxiv.org/abs/2206.14234

GitHub

Bo Tang

Welcome to PyEPO's documentation!

This is the documentation of `PyEPO` (PyTorch-based End-to-End Predict-then-Optimize Library Tool), which aims to provide end-to-end methods for predict-then-optimize tasks.

Sample Code

```
import random

import gurobipy as gp
from gurobipy import GRB
```

PyEPO

Benchmarks

Shortest path

Knapsack

TSP

Optimization modeling

PYOMO

Gurobi

Any custom solver

ML modeling

PyTorch

scikit learn

Training algorithms

SPO+

SPO+ with relaxations

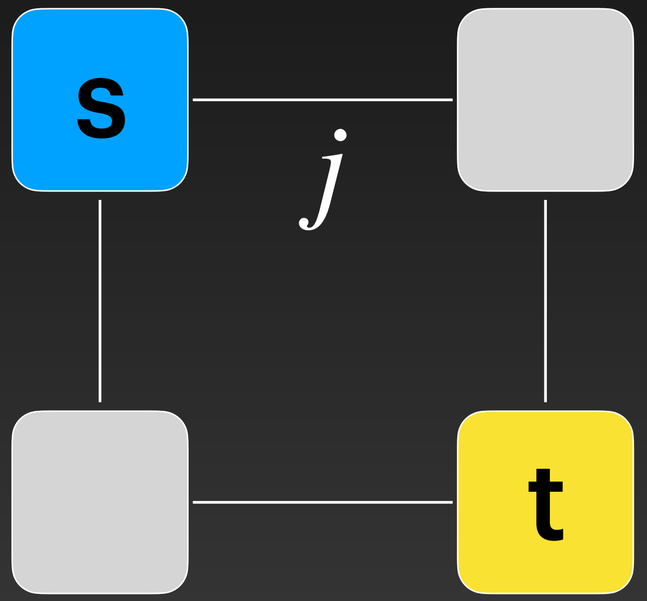
PFYL/**DPO**

DBB
(Pogancic et al. [2019])

Benchmark generation

Based on Elmachtoub & Grigas

$$c_{ij} = \alpha\left((\mathcal{B}x_i)_j + 3\right)^{deg} \cdot \epsilon_{i,j}$$



$x_i \in \mathbb{R}^p$ — i-th instance's feature vector (Observed)

deg — Integer in [1,6] regulating how non-linear the mapping is

$\mathcal{B} \in \{0,1\}^{d \times n}$

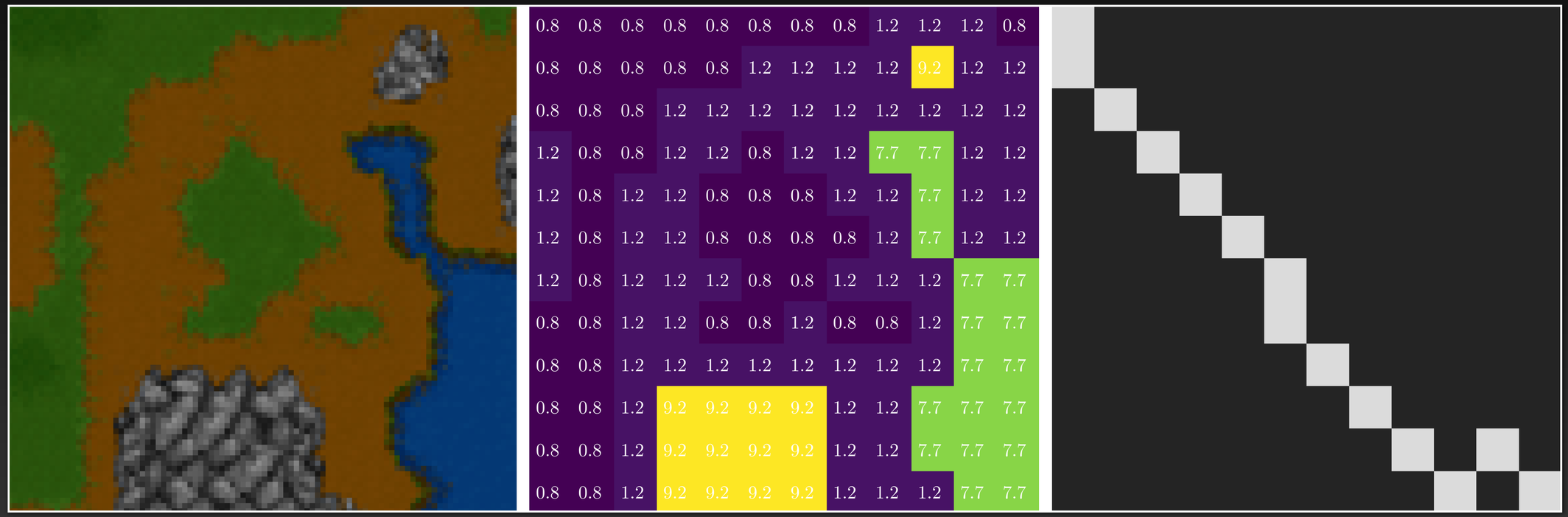
$\mathcal{B}_{p,q} \sim \text{Bernoulli}(0.5)$

$\epsilon_{i,j}$ — Uniform random noise

Warcraft Shortest-Path Benchmark

Based on Pogančić, Marin Vlastelica, et al. "Differentiation of blackbox combinatorial solvers." ICLR 2020.

Image inputs (RGB features) *True costs* *Optimal solution (NW -> SE)*



Optimization model (in Gurobi)

```python
import gurobipy as gp
from gurobipy import GRB
from pyepo.model.grb import optGrbModel

class myModel(optGrbModel):
    def _getModel(self):
        # ceate a model
        m = gp.Model()
        # varibles
        x = m.addVars(5, name="x", vtype=GRB.BINARY)
        # sense (must be minimize)
        m.modelSense = GRB.MAXIMIZE
        # constraints
        m.addConstr(3*x[0]+4*x[1]+3*x[2]+6*x[3]+4*x[4]<=12)
        m.addConstr(4*x[0]+5*x[1]+2*x[2]+3*x[3]+5*x[4]<=10)
        m.addConstr(5*x[0]+4*x[1]+6*x[2]+2*x[3]+3*x[4]<=15)
        return m, x

optmodel = myModel()
```

$$
\begin{aligned}
\max_{x} \quad & \sum_{i=0}^{4} c_i x_i \\
s.t. \quad & 3x_0 + 4x_1 + 3x_2 + 6x_3 + 4x_4 \leq 12 \\
& 4x_0 + 5x_1 + 2x_2 + 3x_3 + 5x_4 \leq 10 \\
& 5x_0 + 4x_1 + 6x_2 + 2x_3 + 3x_4 \leq 15 \\
& \forall x_i \in \{0, 1\}
\end{aligned}
$$

Creating a dataset based on features and true costs

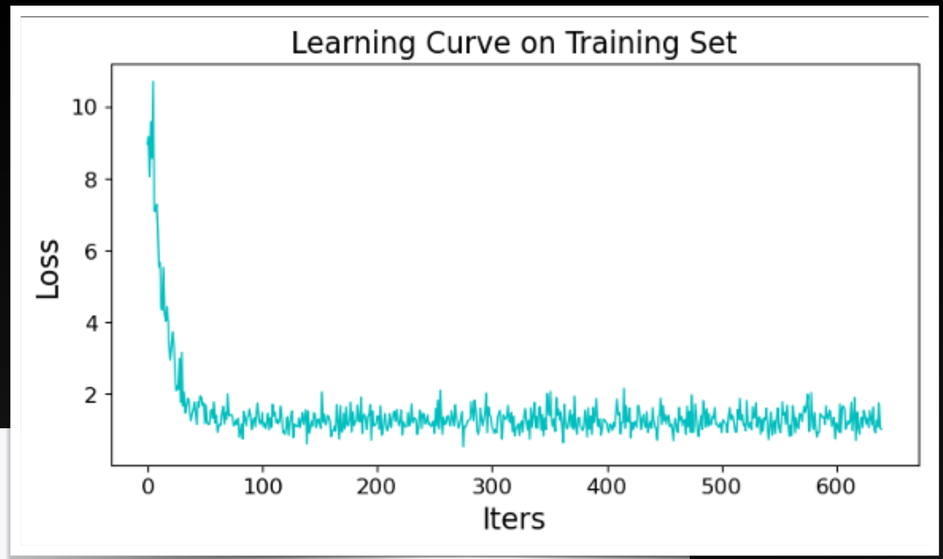
```
import pyepo
from torch.utils.data import DataLoader

# build dataset
dataset = pyepo.data.dataset.optDataset(optmodel, feats, costs)

# get data loader
dataloader = DataLoader(dataset, batch_size=32, shuffle=True)
```

```
for x, c, w, z in dataloader:
    # a batch of features
    print(x)
    # a batch of true costs
    print(c)
    # a batch of true optimal solutions
    print(w)
    # a batch of true optimal objective values
    print(z)
```

Creating an ML model with PyTorch

```
from torch import nn

# construct linear model
class LinearRegression(nn.Module):
    def __init__(self):
        super(LinearRegression, self).__init__()
        # size of input and output is the size of feature and cost
        self.linear = nn.Linear(num_feat, len_cost)
    def forward(self, x):
        out = self.linear(x)
        return out

# init model
predmodel = LinearRegression()
```

End-to-end training!

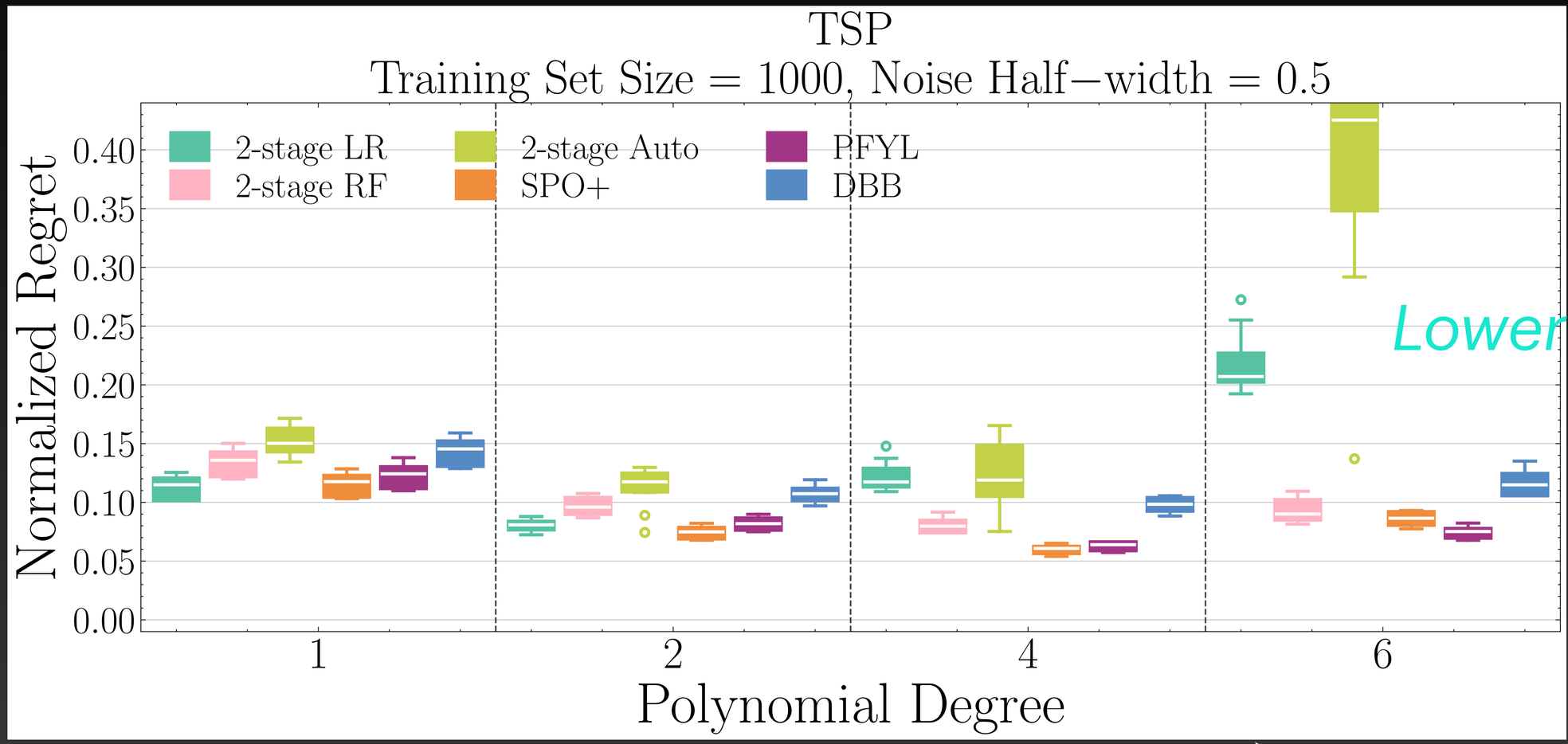



```python
import torch

# set SGD optimizer
optimizer = torch.optim.SGD(predmodel.parameters(), lr=1e-3)

# training
for epoch in range(num_epochs):
    # iterare features, costs, solutions, and objective values
    for x, c, w, z in dataloader:
        # forward pass
        cp = predmodel(x) # predict costs
        loss = spop(cp, c, w, z).mean() # calculate SPO+ loss
        # backward pass
        optimizer.zero_grad() # reset gradients to 0
        loss.backward() # compute gradients
        optimizer.step() # update model parameters
```

Experiments with TSP20

Vertical axis: average regret w.r.t. true OPT on unseen test instances

2-stage methods: regress on true costs (no end-to-end training)



Lower is better

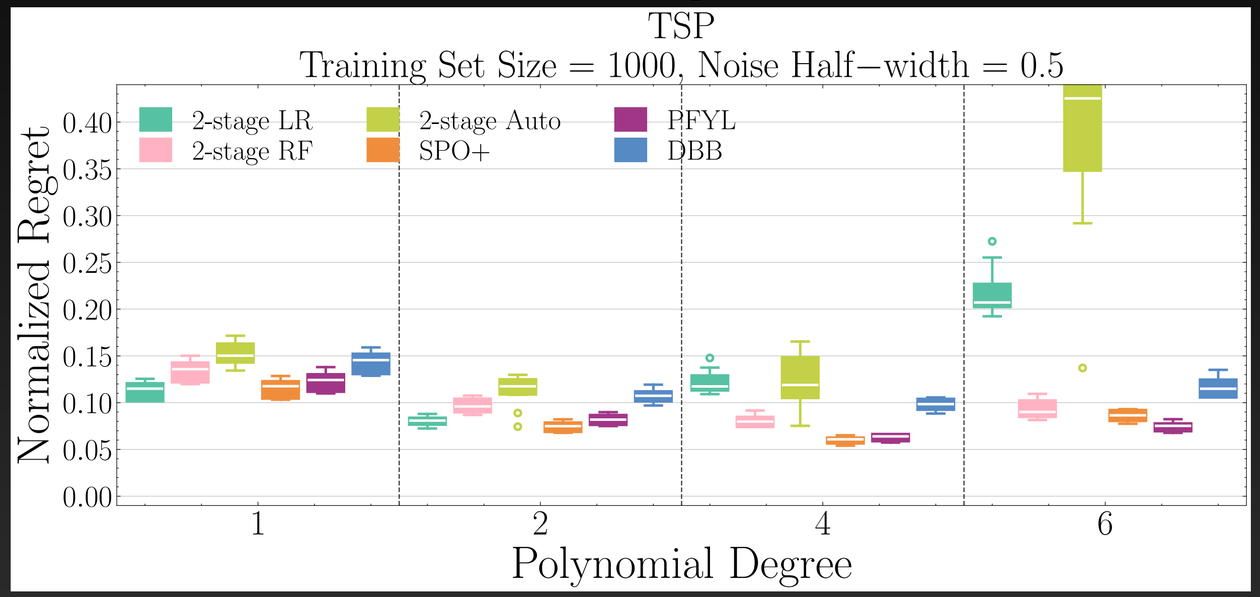
Harder learning task

Experiments with TSP20

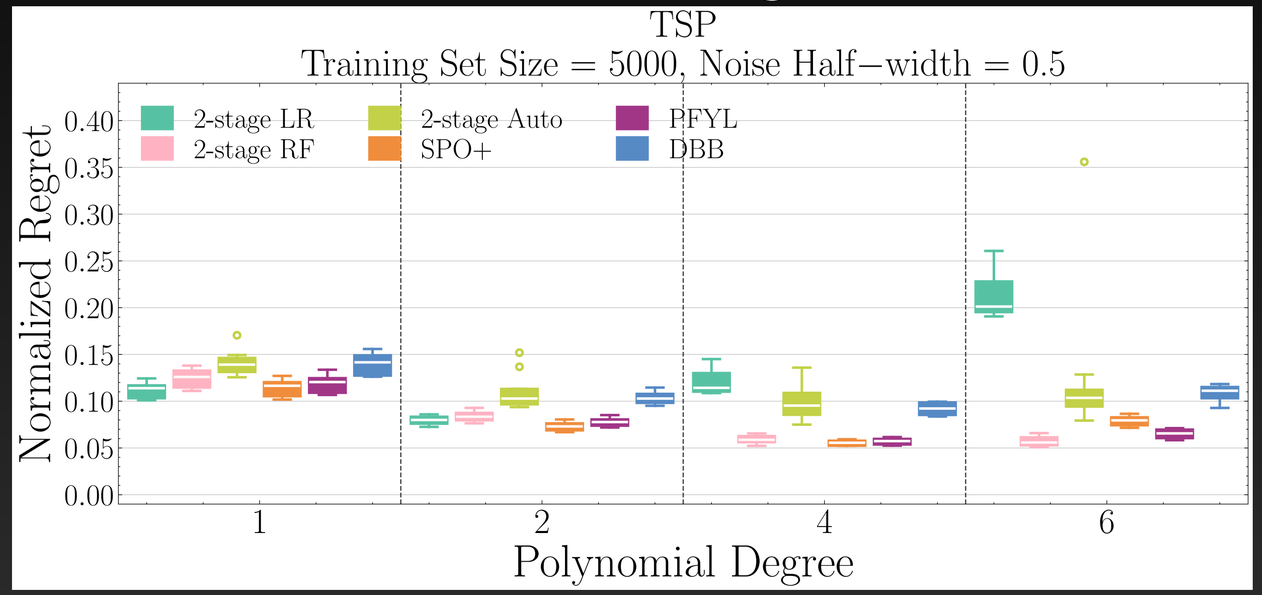
Vertical axis: average regret w.r.t. true OPT on unseen test instances

2-stage methods: regress on true costs (no end-to-end training)

n=1000 training instances



n=5000 training instances



SPO+ and **PFYL**, **both using a linear model** perform the best.

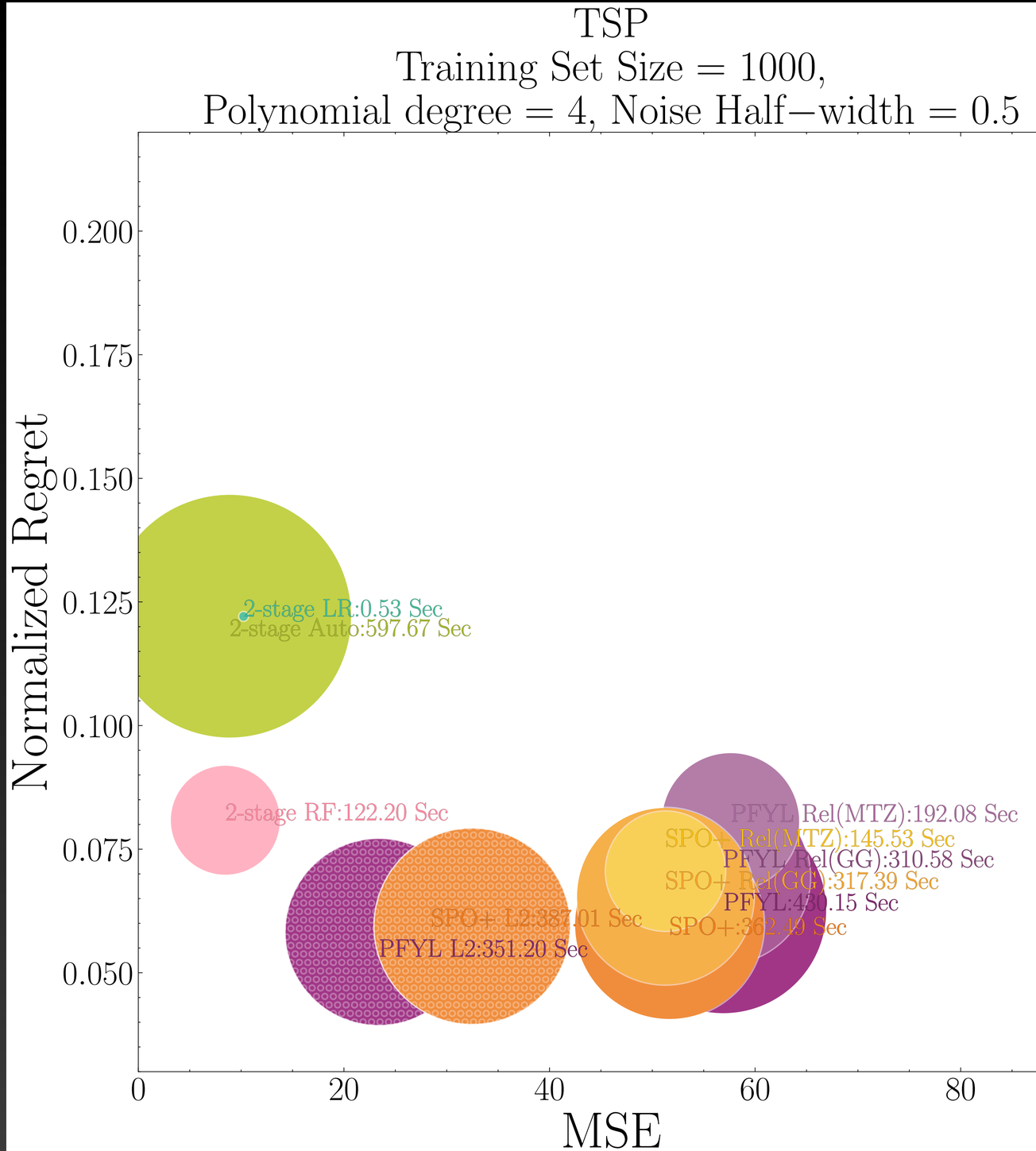
With more data, **2-stage Random Forest** is competitive!

Regret-accuracy tradeoff

Vertical axis: average regret w.r.t. true OPT on unseen test instances

Horizontal axis: Mean-Squared Error on true costs

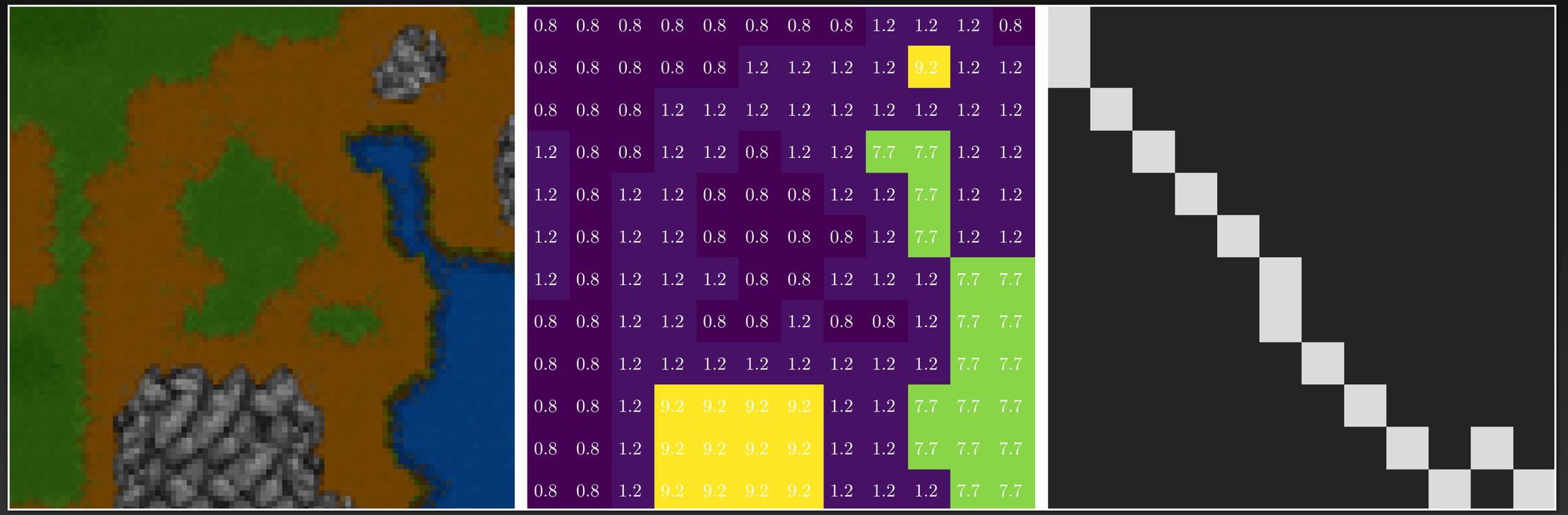
$$l_{\text{MSE}}(\hat{\boldsymbol{c}}, \boldsymbol{c}) = \frac{1}{2n} \sum_i^n \|\hat{\boldsymbol{c}}_i - \boldsymbol{c}_i\|_2^2$$



Warcraft Shortest-Path Benchmark

Based on Pogančić, Marin Vlastelica, et al. "Differentiation of blackbox combinatorial solvers." ICLR 2020.

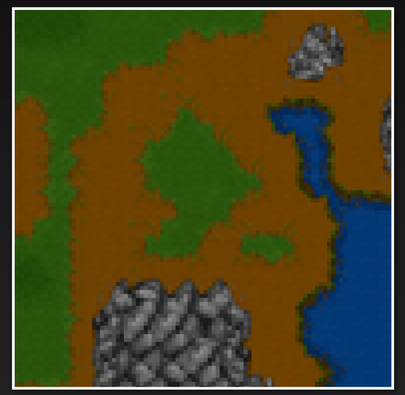
Image inputs (RGB features) *True costs* *Optimal solution (NW -> SE)*



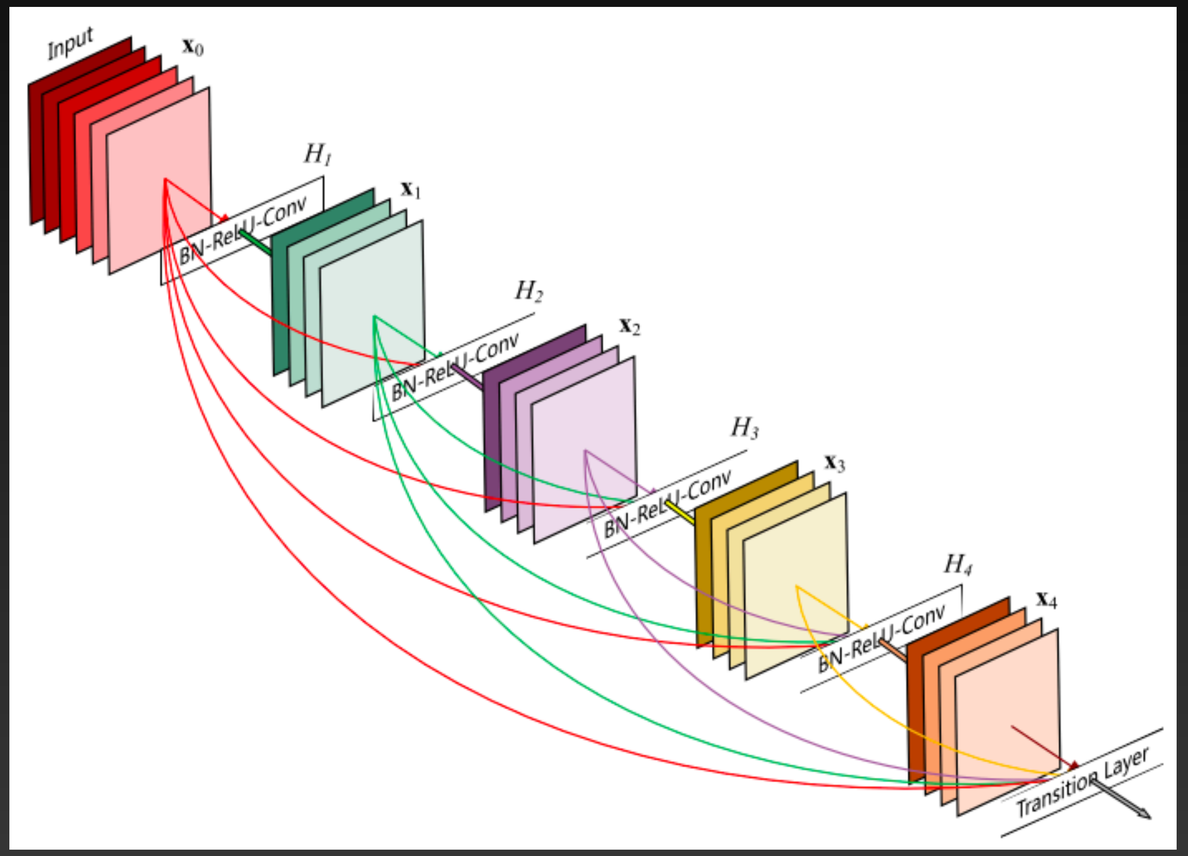
Warcraft Shortest-Path Benchmark

Based on Pogančić, Marin Vlastelica, et al. "Differentiation of blackbox combinatorial solvers." ICLR 2020.

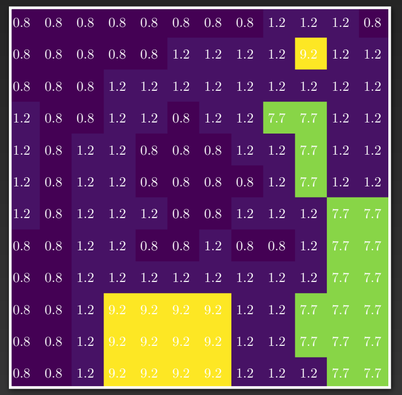
Image inputs



ResNet-18

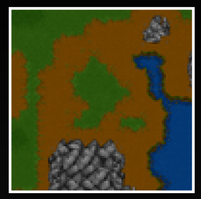


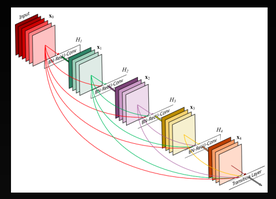
Cost predictions

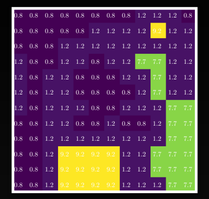


Warcraft Shortest-Path Benchmark

Based on Pogančić, Marin Vlastelica, et al. "Differentiation of blackbox combinatorial solvers." ICLR-20.

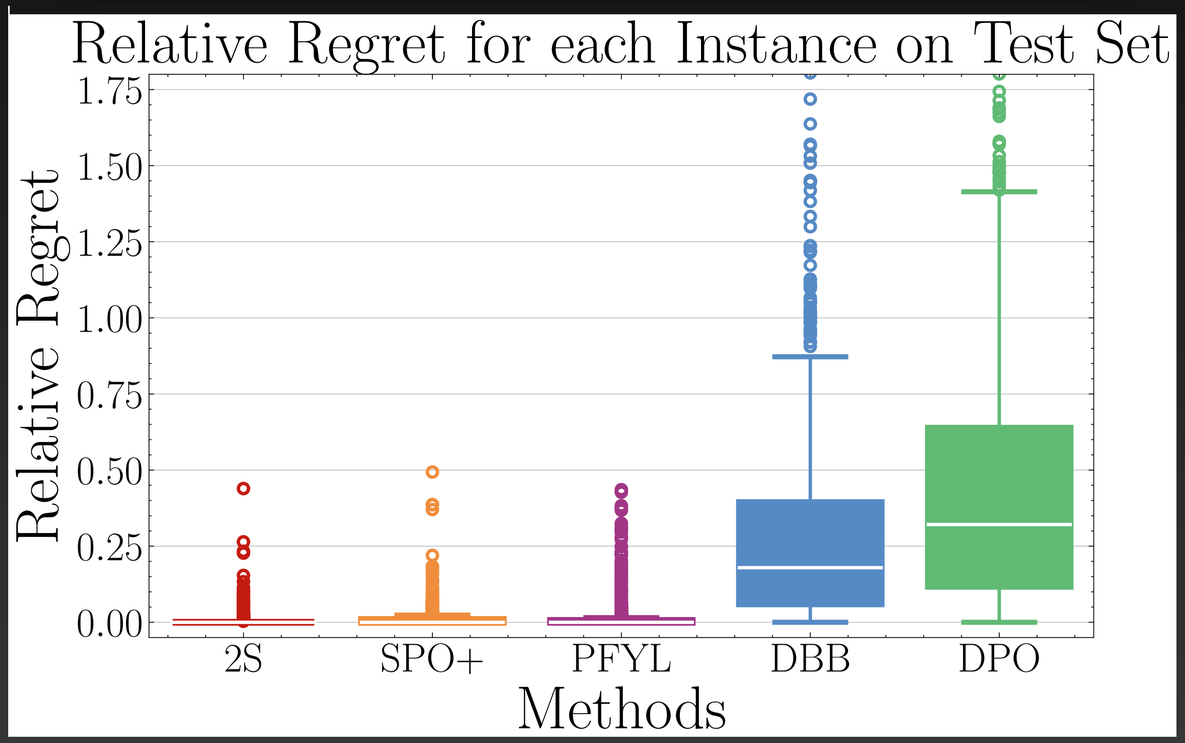


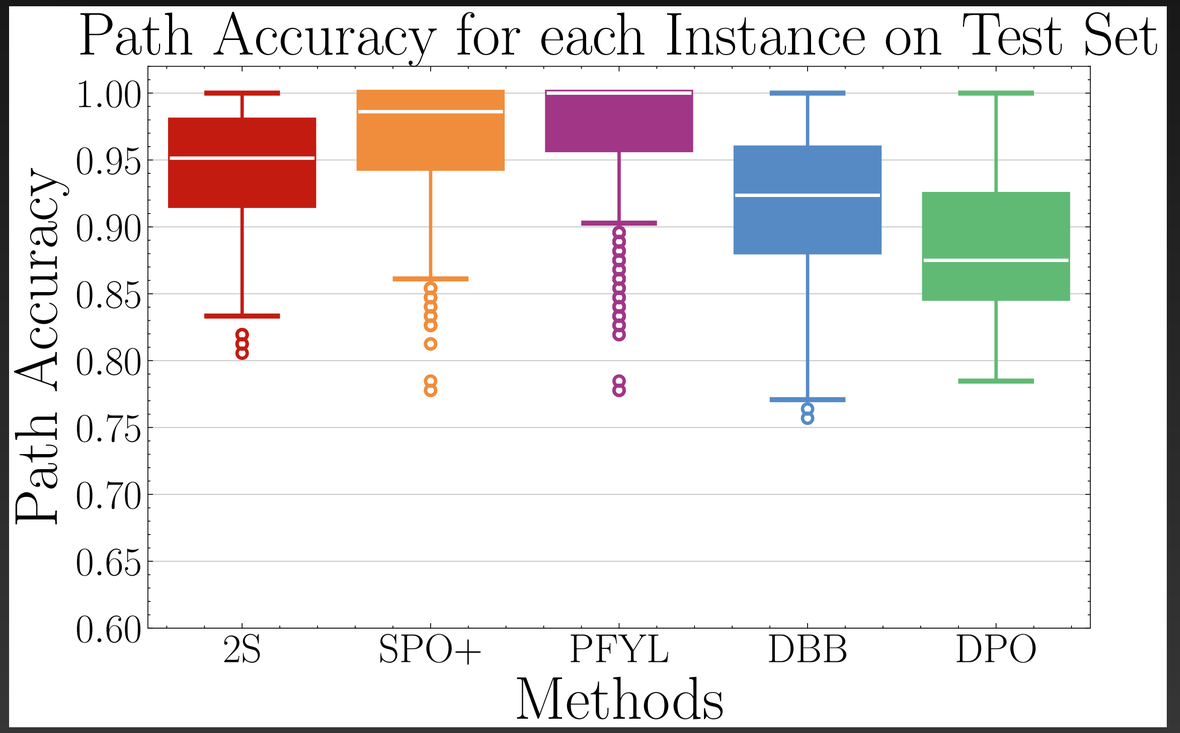




2-Stage, **SPO+**, and **PFYL**, **all using a truncated ResNet-18**, perform the best. **DBB**, originally benchmarked on this dataset, is far worse.

SPO+ and **PFYL** match true optimal paths better than other methods, including **2-Stage**.





Summary

Manuscript: https://arxiv.org/abs/2206.14234

PyEPO

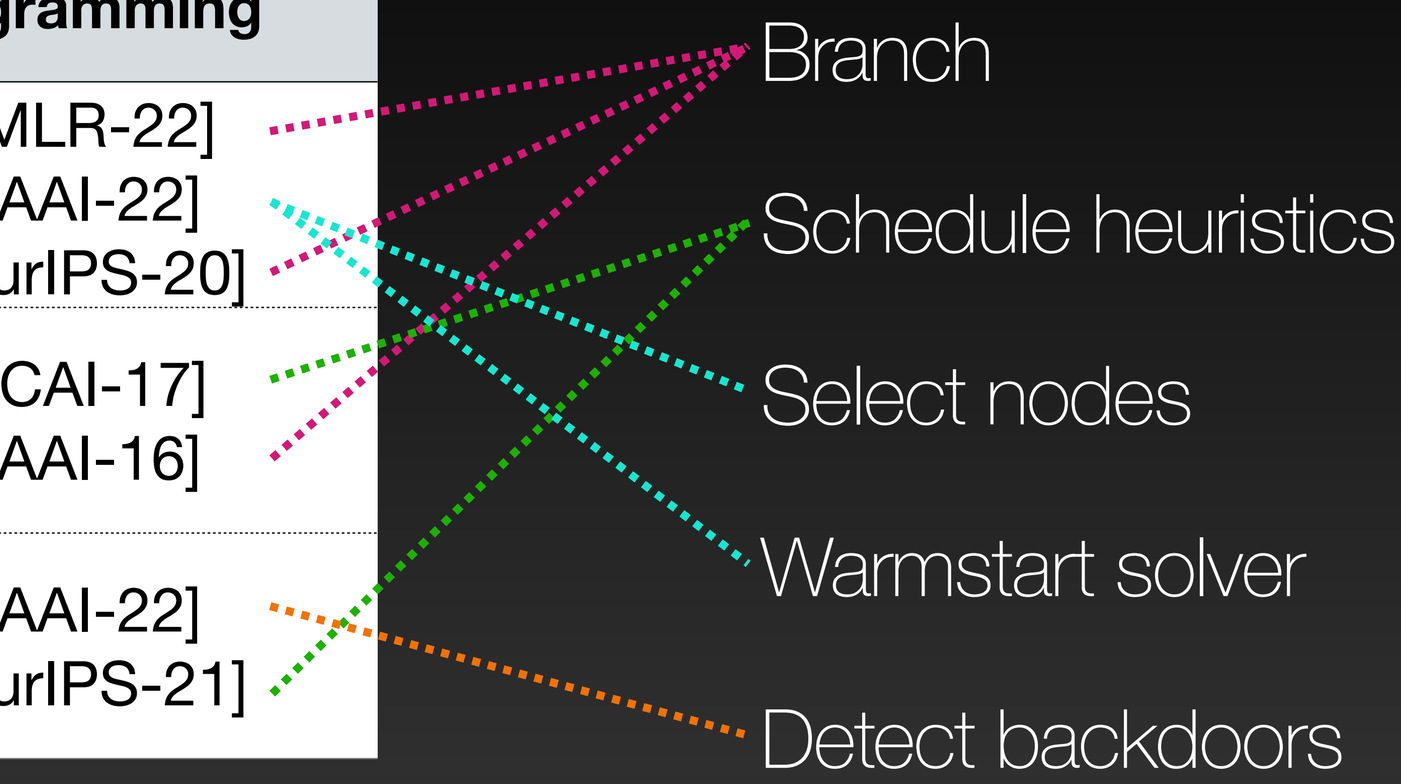
- Predict-then-Optimize is a **highly practical paradigm**.
- **SPO+ and PFYL are very effective** end-to-end learning methods.
- The “naive” **2-stage approach is sufficient training set is large**.
- **Open questions:**
  - Predictions in the constraints; see Hu, Xinyi, Jasper CH Lee, and Jimmy HM Lee. "Branch & Learn with Post-hoc Correction for Predict+ Optimize with Unknown Parameters in Constraints." CPAIOR 2023.
  - Reducing training time; see work by Tias Guns and collaborators.
  - More applications; see work by B. Dilkina, M. Tambe, B. Wilder, H. Bastani

Bo Tang

GitHub

Machine Learning for Integer Programming

	Mixed-Integer Linear Programming
SL + GNN	[TMLR-22] [AAAI-22] [NeurIPS-20]
SL + Simple models	[IJCAI-17] [AAAI-16]
Custom ML	[AAAI-22] [NeurIPS-21]



SL: Supervised Learning
RL: Reinforcement Learning
GNN: Graph Neural Networks

Machine Learning for Discrete Optimization

Survey on GNN for CombOpt [JMLR 2023]

	MILP	Column Generation	Stochastic Programming	Graph Optimization	Multiobjective optimization
RL + GNN		[NeurIPS-22]		[TMLR-22] [NeurIPS-17]	
SL + GNN	[TMLR-22] [AAAI-22] [NeurIPS-20]				
SL + Simple models	[IJCAI-17] [AAAI-16]		[NeurIPS-22]		https://arxiv.org/abs/2307.03171
Custom ML	[AAAI-22] [NeurIPS-21]				

Lab Colab:

https://tinyurl.com/ACP23-PredictAndOptimize

PyEPO Github/Docs:

https://github.com/khalil-research/PyEPO