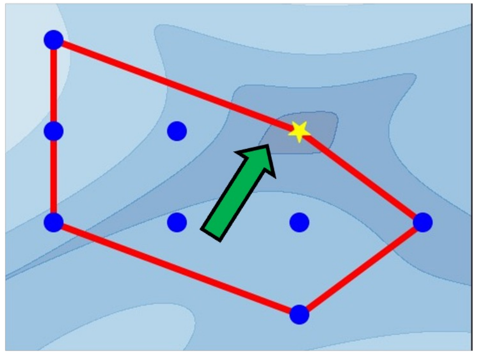


SurCo: Learning Linear Surrogates for Combinatorial Nonlinear Optimization

Aaron Ferber[1], Taoan Huang[1], Daochen Zha[2], Martin Schubert[3], Benoit Steiner[4], Bistra Dilkina[1], Yuandong Tian[4]

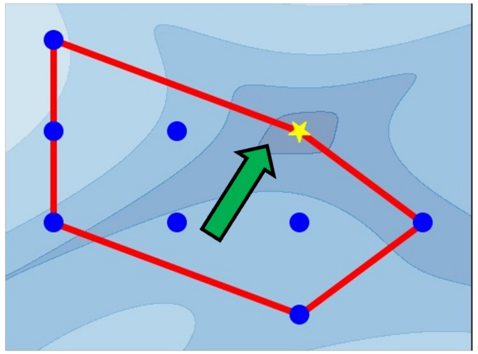
[1]University of Southern California, [2]Rice University, [3]Reality Lab Display, [4]Meta AI (FAIR)

USC

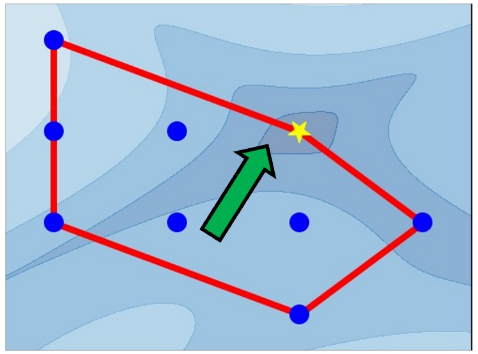
RICE UNIVERSITY School of Engineering Department of Computer Science

Meta AI

Optimizing Nonlinear Functions over Combinatorial Regions



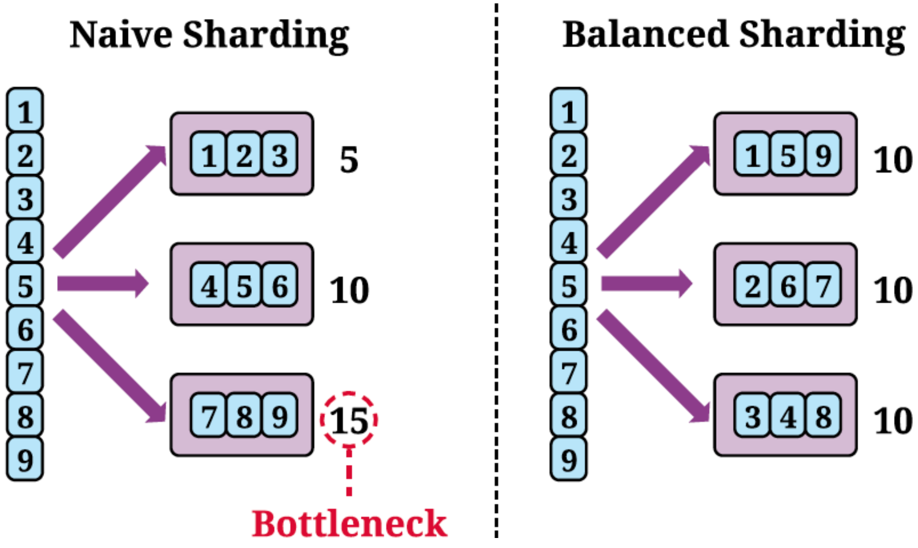
- Nonlinear + differentiable objective
- Combinatorial feasible region
- Real-world domains:
  - Computer system planning
  - Designing photonic devices
  - Throughput optimization
  - Antenna design
  - Energy grid



Example: Embedding Table Placement

Given:

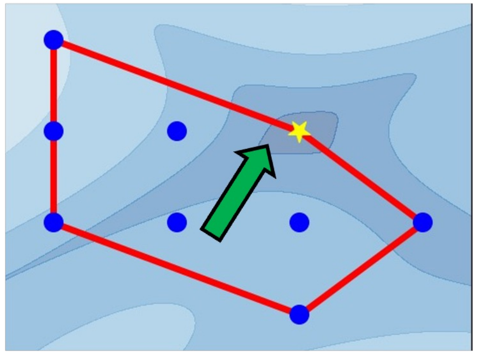
- k tables
- n identical devices
- Table i has memory requirement m_i
- Device memory capacity M



Find

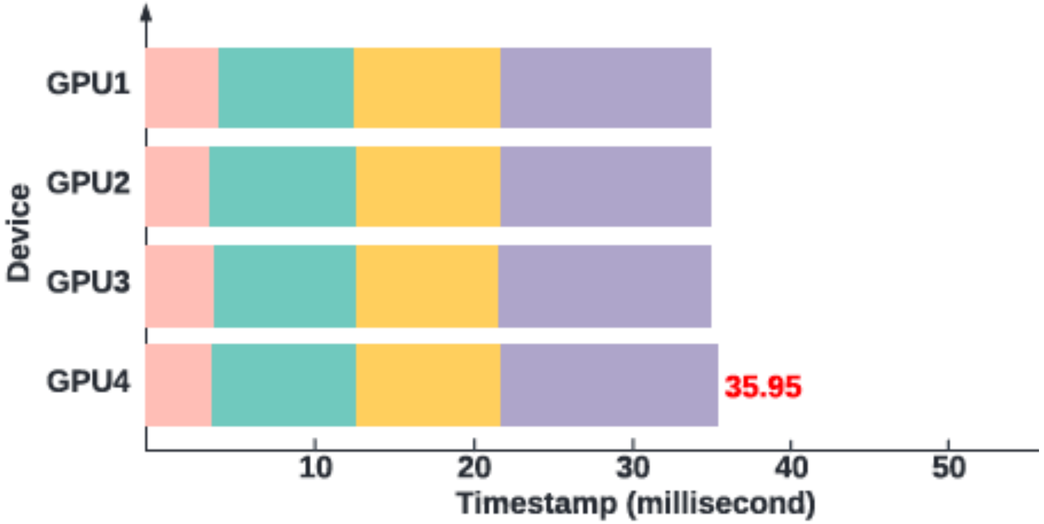
- Allocation of tables to devices observing device memory limits
- Minimize latency which is estimated by a neural network (capturing nonlinear interactions)

Example: Embedding Table Placement



Given:

- k tables
- n identical devices
- Table i has memory requirement m_i
- Device j has memory capacity M_j

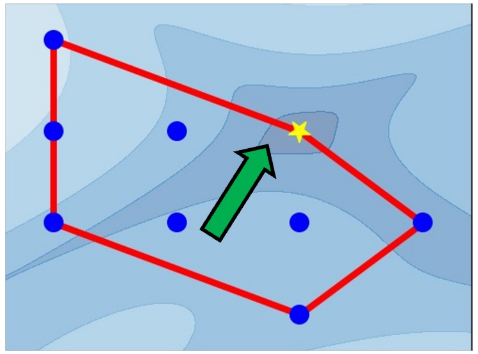


Formulation

$$\text{Min}_x\ \boldsymbol{L}(\{x_{ij}\}) \quad \text{s.t.}\ \textstyle\sum_i x_{ij} m_i \le M_j, \quad \sum_j x_{ij} = 1\,, \qquad x_{ij} \in \{0,1\}$$

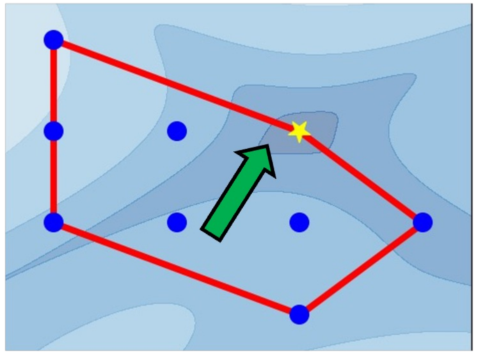
$\boldsymbol{L}$ is nonlinear due to system issues (e.g., batching, communication, etc)

Nonlinear Optimization is Hard



- Specific domains have specialized solvers
- General solvers are often slow (without very careful modeling)
- Genetic algorithms or gradient-based methods may not find feasible solutions

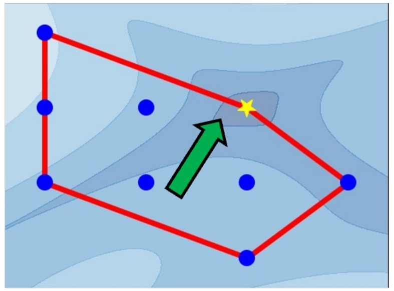
Linear Optimization is Easy(ish)



- MILP solvers (CPLEX, Gurobi, SCIP) easily handle industry-scale problems
- Plus other solvers for linear settings
  - Greedy
  - LP + total unimodularity

Idea: Find a Linear Surrogate

- Learn a MILP objective whose optimal solution x* solves the nonlinear problem

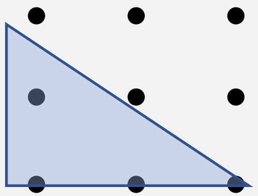


Originally

Nonlinear optimization with combinatorial constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x};\boldsymbol{y})$$

$$\text{s.t } \boldsymbol{x} \in \Omega =$$



combinatorial constraints

Predict surrogate cost $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{y})$

Now

Surrogate optimization

$$\boldsymbol{x}^*(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{y})^{\boldsymbol{T}}\boldsymbol{x}$$

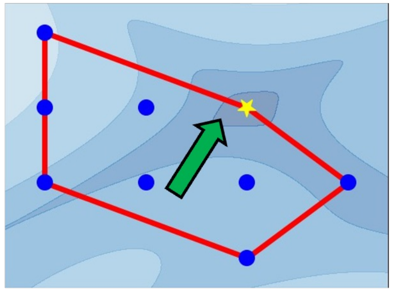
$$\text{s.t } \boldsymbol{x} \in \Omega$$

solved by existing combinatorial solvers

$\boldsymbol{x}^*(\boldsymbol{y})$ optimizes $f(\boldsymbol{x};\boldsymbol{y})$ as much as possible

Idea: Find a Linear Surrogate

- Learn a MILP objective whose optimal solution x* solves the nonlinear problem

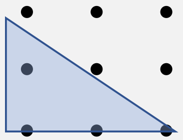


Originally

Nonlinear optimization with combinatorial constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}; \boldsymbol{y})$$

$$\text{s.t } \boldsymbol{x} \in \Omega =$$



Predict surrogate cost $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{y})$

Now

Surrogate optimization

$$\boldsymbol{x}^*(\boldsymbol{y}) = \operatorname{argmin}_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{y})^{\boldsymbol{T}} \boldsymbol{x}$$

$$\text{s.t } \boldsymbol{x} \in \Omega$$

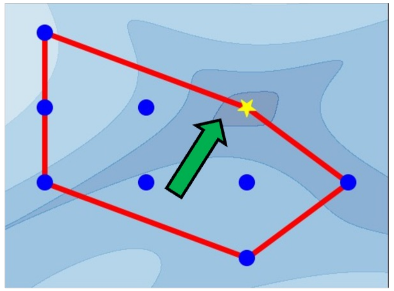
solved by existing combinatorial solvers

$\boldsymbol{x}^*(\boldsymbol{y})$ optimizes $f(\boldsymbol{x}; \boldsymbol{y})$ as much as possible

Challenge: how to find the right objective?

Idea: Find a Linear Surrogate

- Learn a MILP objective whose optimal solution x* solves the nonlinear problem

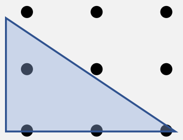


Originally

Nonlinear optimization with combinatorial constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}; \boldsymbol{y})$$

$$\text{s.t } \boldsymbol{x} \in \Omega =$$



combinatorial constraints

Predict surrogate cost $\boldsymbol{c} = \boldsymbol{c}(\boldsymbol{y})$

Now

Surrogate optimization

$$\boldsymbol{x}^*(\boldsymbol{y}) = \operatorname{argmin}_{\boldsymbol{x}} \boldsymbol{c}(\boldsymbol{y})^{\boldsymbol{T}} \boldsymbol{x}$$

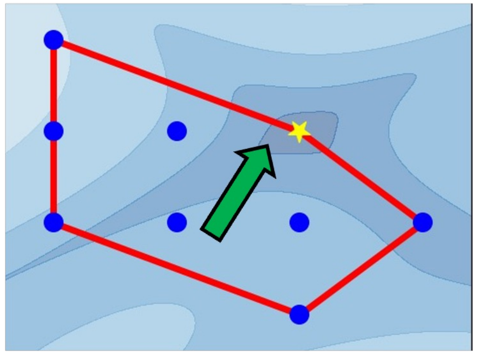
$$\text{s.t } \boldsymbol{x} \in \Omega$$

solved by existing combinatorial solvers

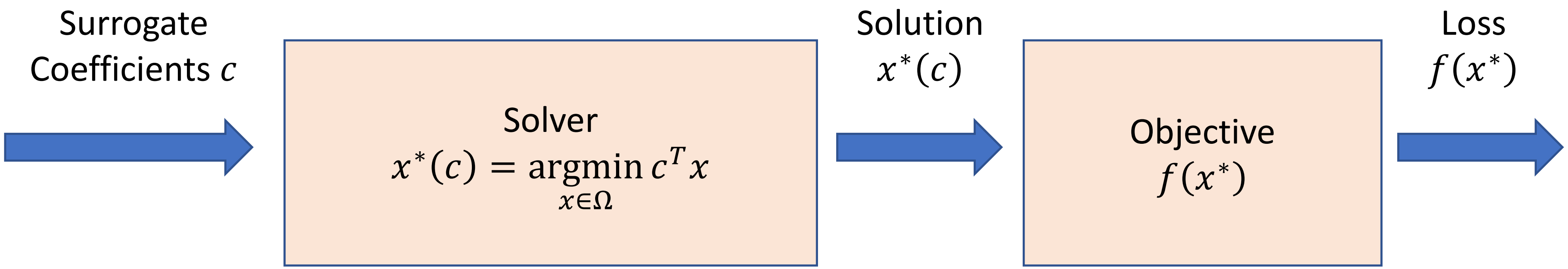
$\boldsymbol{x}^*(\boldsymbol{y})$ optimizes $f(\boldsymbol{x}; \boldsymbol{y})$ as much as possible

Proposal: gradient-based optimization

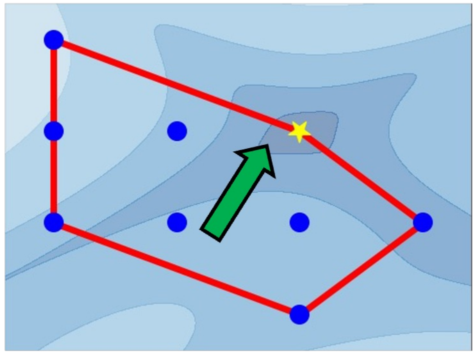
Proposal: surrogate learning



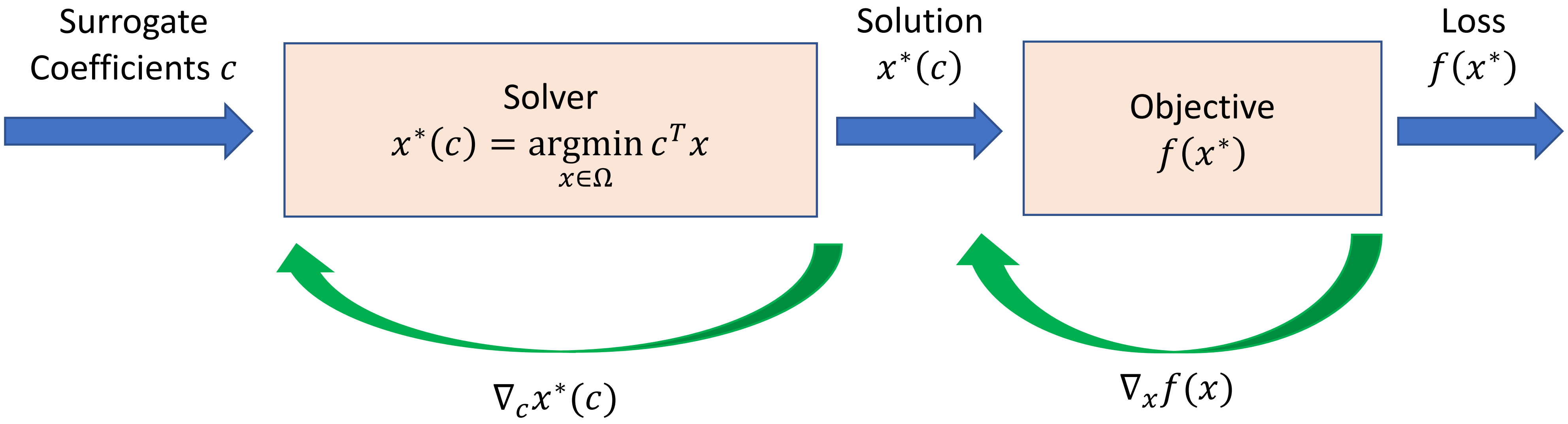
- Use surrogate MILP to solve original problem
- Find linear coefficients c such that $\underset{x \in \Omega}{\operatorname{argmin}} f(x) \approx \underset{x \in \Omega}{\operatorname{argmin}} c^T x$



SurCo-zero: gradient-based optimization



- **Iterative** solver based on linear surrogate guided by **gradient updates**
- Update linear coefficients c such that $x^*(c)$ improves objective $f(x^*(c))$



Recent work on differentiable optimization

Differentation of blackbox optimizers

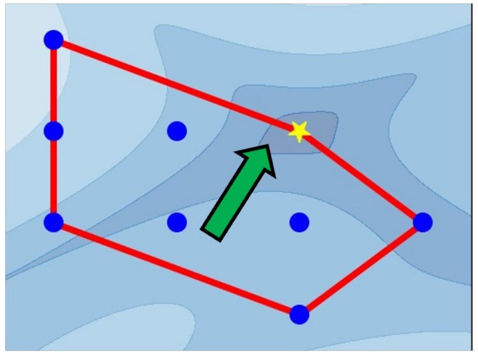
CVXPYLayers

MIPaaL

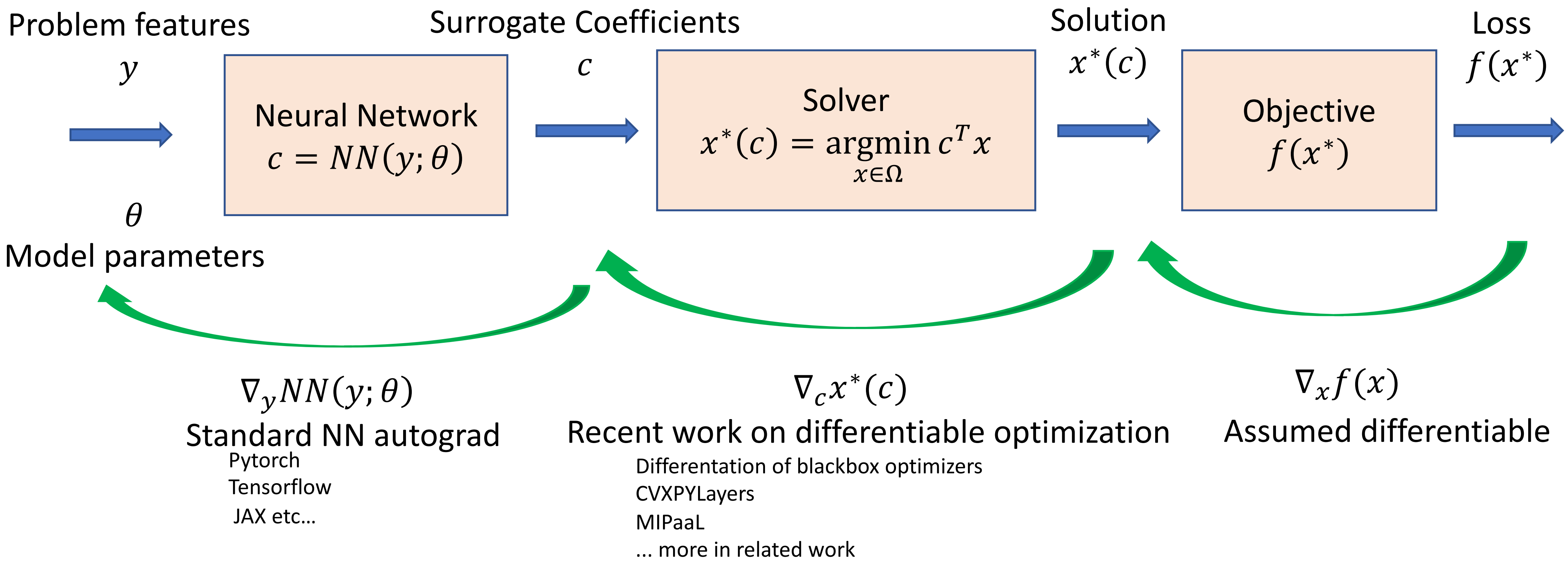
... more in related work

Assumed differentiable

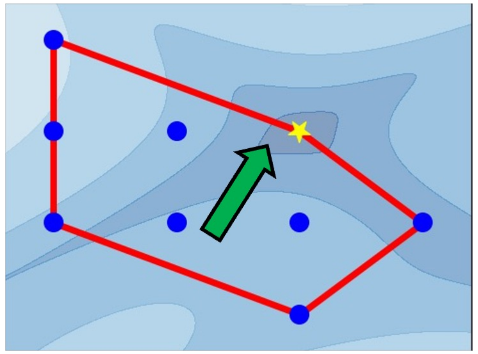
SurCo-prior: distributional learning



- One pass solver based on model **learned offline**
- Use neural model based on **problem features** to predict linear coefficients



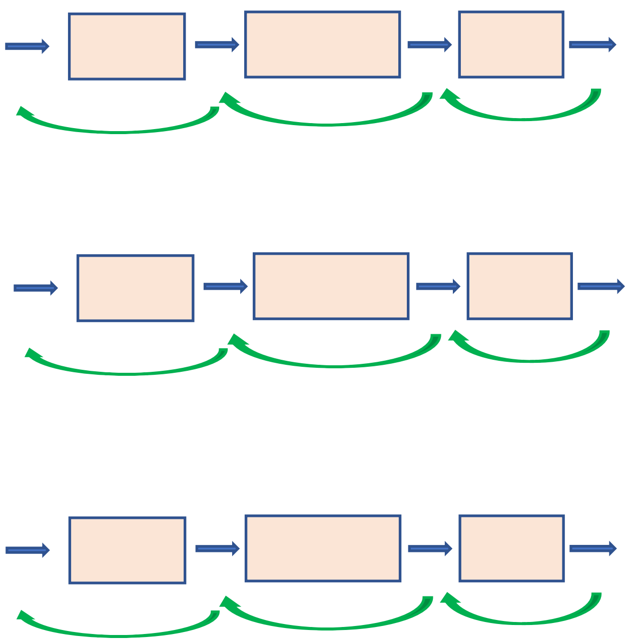
SurCo-prior: distributional learning



- Update neural network parameters from training dataset

$c_i = NN(y_i;\theta)$

Train Model parameters $\boldsymbol{\theta}$



Surrogate Coefficients

$c_{\text{test}} = NN(y_{\text{test}};\theta)$

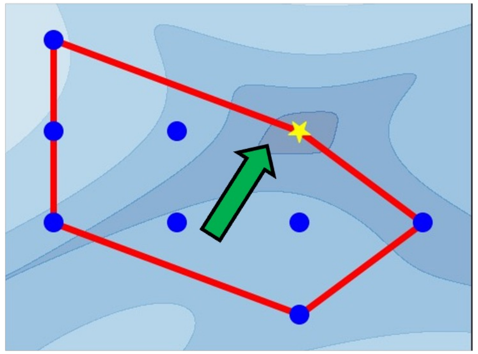
Solver

$x^*(c) = \underset{x\in\Omega}{\operatorname{argmin}}\, c^T x$

Solution

$x^*(c_{\text{test}})$

SurCo-hybrid: fine-tuning from trained model

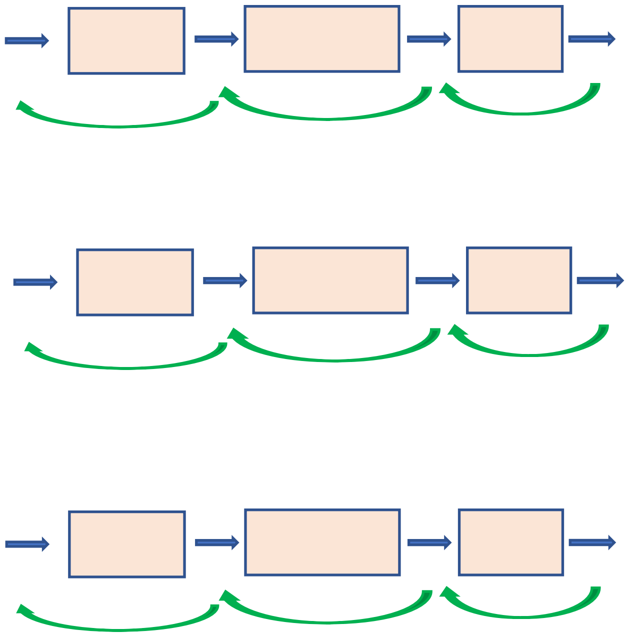


Update neural network parameters from training dataset

Fine-tune surrogate on-the-fly

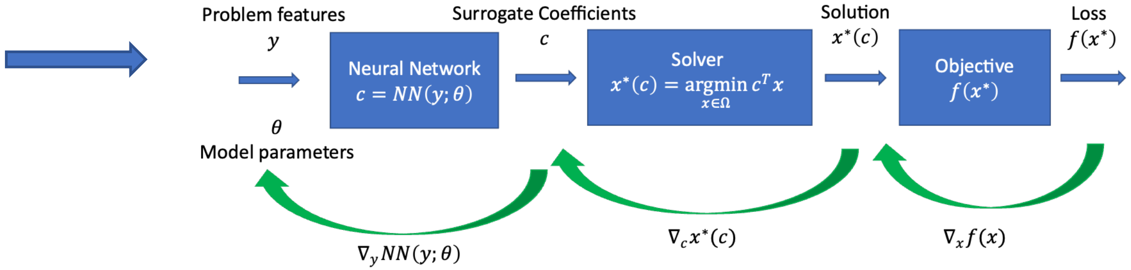
$$c_i = NN(y_i; \theta)$$

Train Model parameters $\boldsymbol{\theta}$

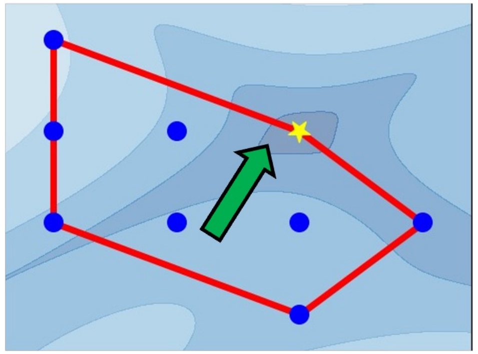


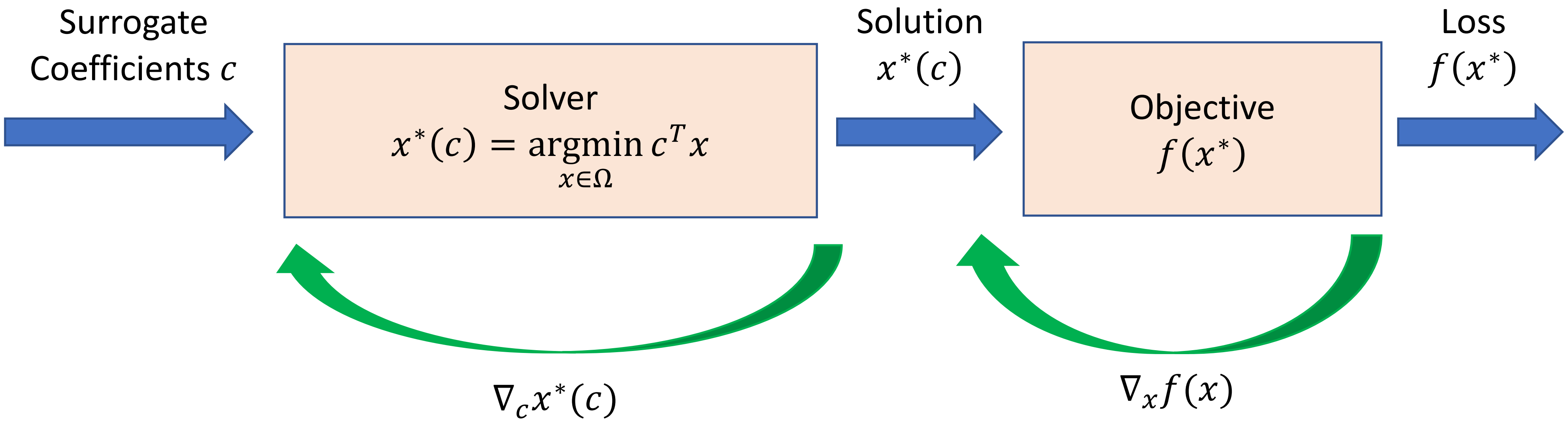
Initial Surrogate Coefficients

$$c_0 = NN(y_{\text{test}}; \theta)$$



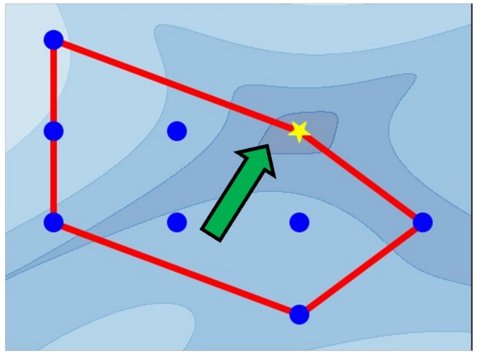
SurCo-zero



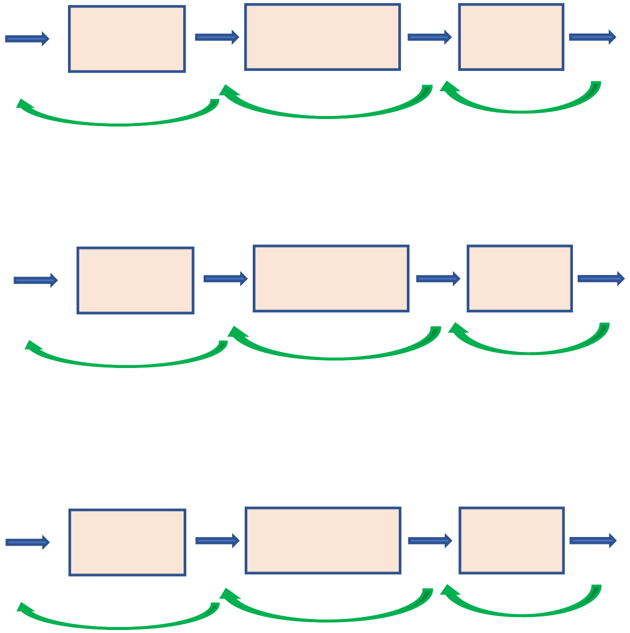


No offline training data, just solve a single problem instance on-the-fly

SurCo-prior



$c_i = NN(y_i; \theta)$



Train Model parameters $\boldsymbol{\theta}$

Surrogate Coefficients

$c_{\text{test}} = NN(y_{\text{test}}; \theta)$

Solver

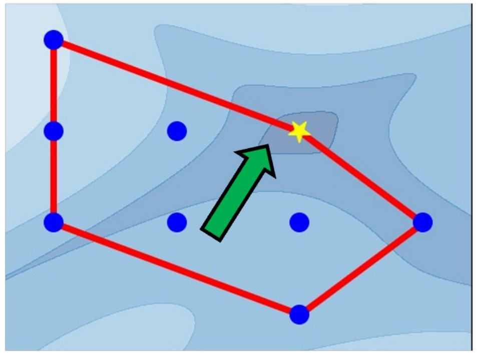
$$x^*(c) = \underset{x \in \Omega}{\operatorname{argmin}}\, c^T x$$

Solution

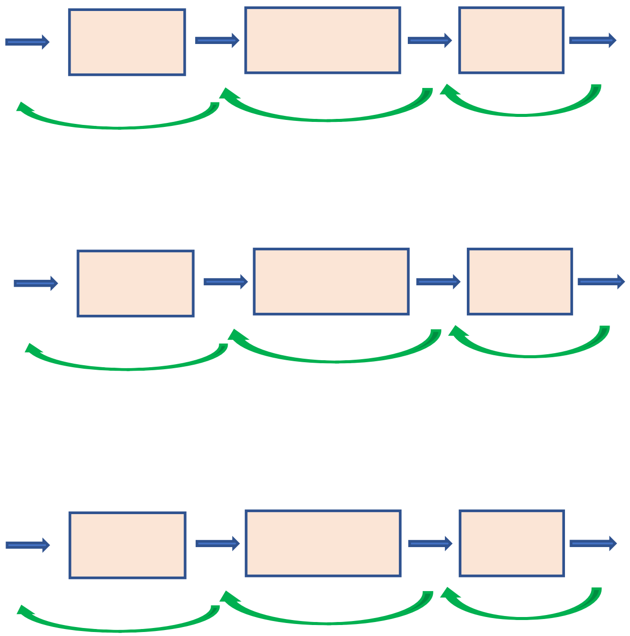
$x^*(c_{\text{test}})$

Uses offline training data to quickly solve problems at test time with just one solver call

SurCo-hybrid



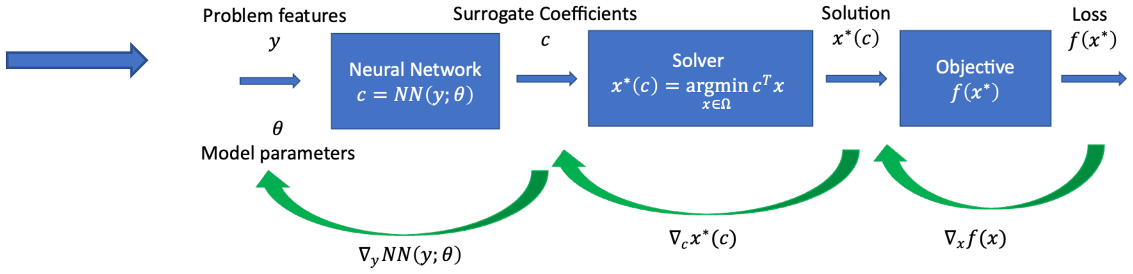
$$c_i = NN(y_i; \theta)$$



Train Model parameters $\boldsymbol{\theta}$

Initial Surrogate Coefficients

$$c_0 = NN(y_{\text{test}}; \theta)$$



Offline train + on-the-fly fine-tuning the surrogate

Related Work

Differentiable optimization: backprop through solvers

Amos et al. OptNet: Differentiable optimization as a layer in neural networks. ICML 2017

Agrawal et al. Differentiable Convex Optimization Layers. NeurIPS 2019

Berthet et al. Learning with Differentiable Perturbed Optimizers. NeurIPS 2020

Demirović et al. Predict+Optimise with Ranking Objectives: Exhaustively Learning Linear Functions. IJCAI 2019

Demirović et al. Dynamic Programming for Predict + Optimise. AAAI 2020

Djolonga et al. Differentiable Learning of Submodular Models. NeurIPS 2017

Donti et al. Task-Based End-to-End Model Learning in Stochastic Optimization. NeurIPS 2017

Elmachtoub et al. Smart "Predict, then Optimize". Management Science 2022

Ferber et al. MIPaaL: Mixed Integer Program as a Layer. AAAI 2020

Lee et al. Meta-Learning with Differentiable Convex Optimization. CVPR 2019

Mandi et al. Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems. AAAI 2020

Niepert et al. Implicit MLE: Backpropagating Through Discrete Exponential Family Distributions. NeurIPS 2021

Valstelica et al. Differentiation of Blackbox Combinatorial Solvers. ICLR 2019

Rolnínek et al. Optimizing Rank-Based Metrics with Blackbox Differentiation. CVPR 2020

Wang et al. Automatically Learning Compact Quality-Aware Surrogates for Optimization Problems. NeurIPS 2020

Wang et al. SATNet: Bridging Deep Learning and Logical Reasoning Using a Differentiable Satisfiability Solver. ICML 2019

Wilder et al. Melding the Data-Decisions Pipeline: Decision-focused Learning for Combinatorial Optimization. AAAI 2019

Wilder et al. End to End Learning and Optimization on Graphs. NeurIPS 2019

Mixed Integer Nonlinear Optimization: general-purpose solvers

Burer et al. Non-Convex Mixed Integer Nonlinear Programming: A Survey. ORMS 2012

Belotti et al. Mixed Integer Nonlinear Optimization. Acta Numerica 2013

General-purpose heuristic optimizers: combinatorial constraints are hard

Gad et al. Pygad: An Intuitive Genetic Algorithm Python Library. 2021

Rapin et al. Nevergrad – A Gradient-Free Optimization Platform. 2018

Wang et al. Learning Search Space Partition for Black-Box Optimization Using Monte Carlo Tree Search. NeurIPS 2020

Wang et al. Sample Efficient Neural Architecture Search by Learning Actions for Monte Carlo Tree Search. PAMI 2021

RL for combinatorial optimization: combinatorial constraints are hard

Khalil et al. Learning Combinatorial Optimization Algorithms Over Graphs. NeurIPS 2017

Kool et al. Attention, Learn to Solve Routing Problems! ICLR 2018

Mazyavkina et al. Reinforcement Learning for Combinatorial Optimization: A Survey. COR 2021

Nazari et al. Reinforcement Learning for Solving the Vehicle Routing Problem. NeurIPS 2018

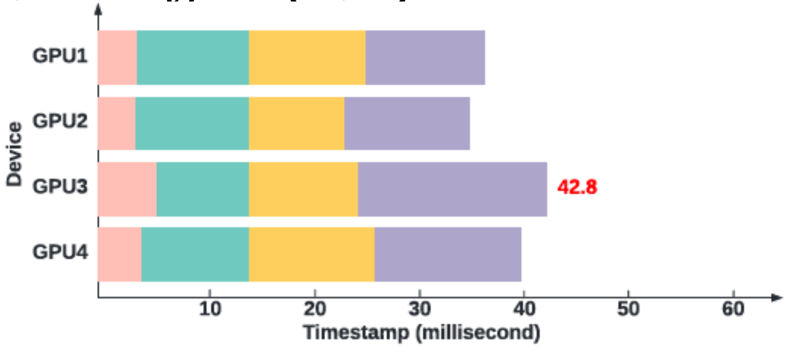
Zhang et al. A Reinforcement Learning Approach to Job-Shop Scheduling. IJCAI 1995

Embedding Table Sharding

Used in large-scale deep learning systems: recommendation systems, knowledge graph

Place N "tables" (with known memory need m_i) on K devices ($x_{ij} = 1$: table i assigned to device j)

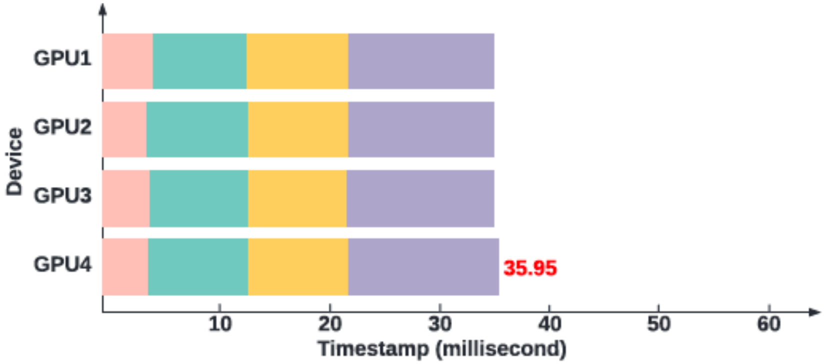
$$\mathrm{Min}_x\, L(\{x_{ij}\}) \quad \text{s.t.} \quad \textstyle\sum_i x_{ij} m_i \le M_j, \quad \sum_j x_{ij} = 1\,, \quad x_{ij} \in \{0,1\}$$



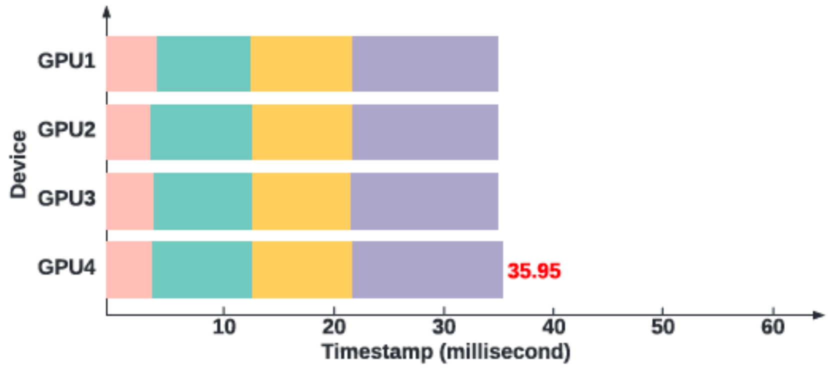
L : Runtime bottleneck f(x) estimated by NN (longest-running device)

L is nonlinear due to system issues
(e.g., batching, communication, etc.)

$\mathrm{c}(y;\theta)$ gives surrogate "per-table cost" c_{ij}
(and $\sum_{ij} c_{ij} x_{ij}$ is the surrogate latency objective)



Embedding Table Sharding

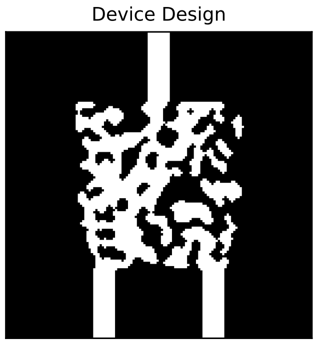


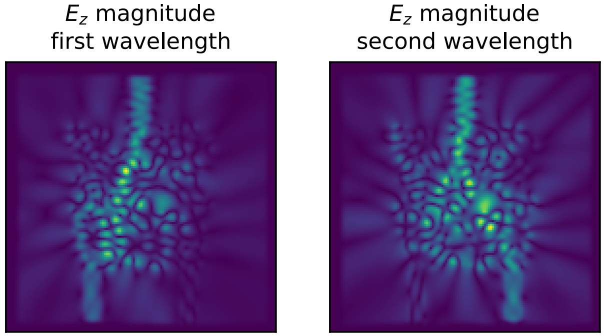
- Public **D**eep **L**earning **R**ecommendation **M**odel (DLRM dataset) placing between 10 to 60 tables on 4 GPUs

- Baseline: Greedy
- SoTA: RL approach Dreamshard[1]
- SurCo: Surrogate NN model learned via CVXPYLayers (differentiable LP Solver)

[1] Zha et al. NeurIPS 2022
Dataset: https://github.com/facebookresearch/dlrm_datasets

Inverse Photonic Design

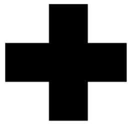


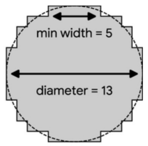


- Design physically-viable devices that take light waves and routes different wavelengths to correct locations

$$\mathcal{L}(S) = \left(\left\| \operatorname{softplus}\left(g \frac{|S|^2 - |S_{\text{cutoff}}|^2}{\min(w_{\text{valid}})} \right) \right\|_2 \right)^2$$

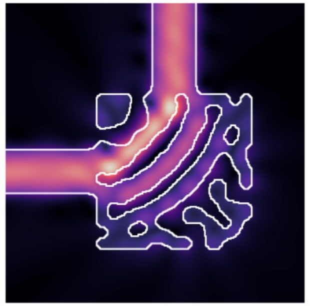
- Device design misspecification loss f(x) computed by differentiable electromagnetic simulator
- Feasible solution: the design must be the union of brush pattern
  - x = binary_opening(x, brush)
  - x = ~binary_opening(~x, brush)

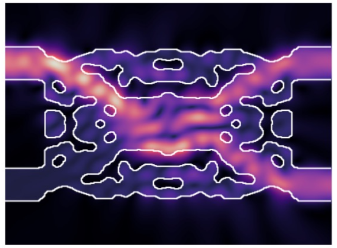


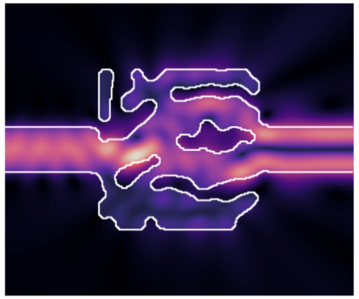


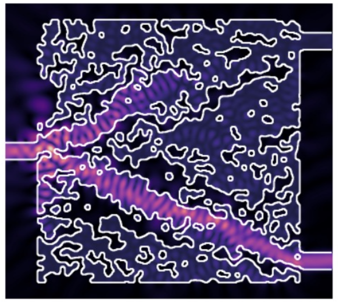
Inverse Photonic Design

- Dataset: Ceviche Challenges[1]
- Most baselines don’t work here due to combinatorial constraints
- SoTA: Brush-based algorithm [1]
- SurCo: Surrogate learned via blackbox differentiation [2] of brush solver


Waveguide bend

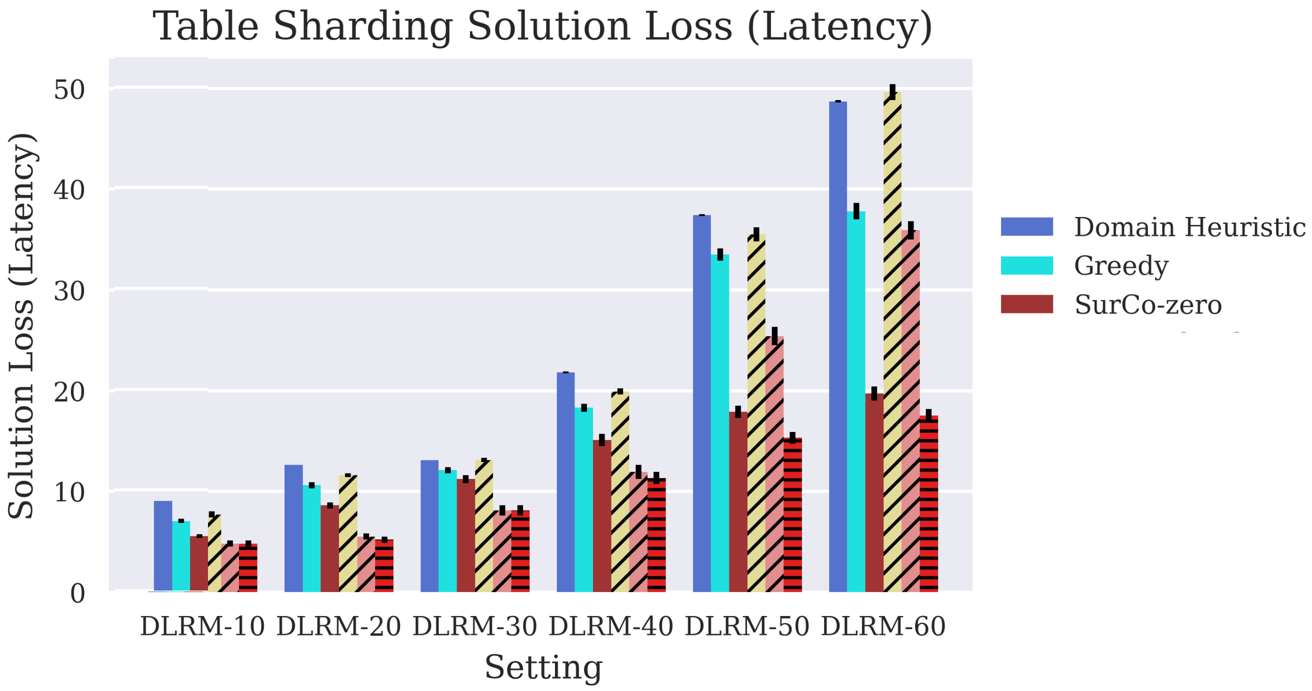

Beam splitter

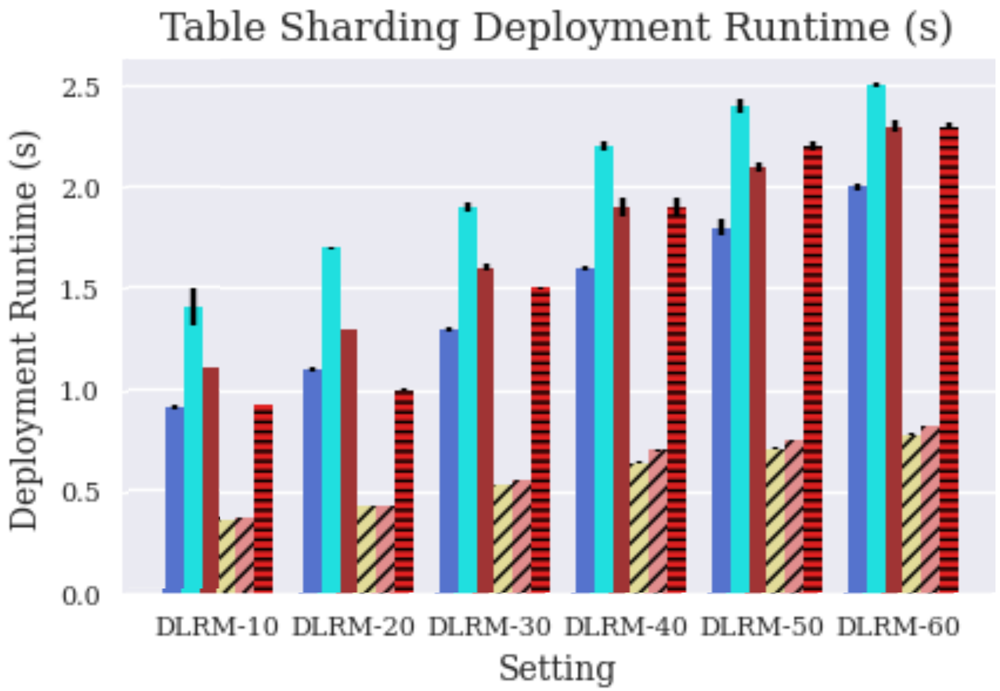

Mode converter


Wavelength division multiplexer

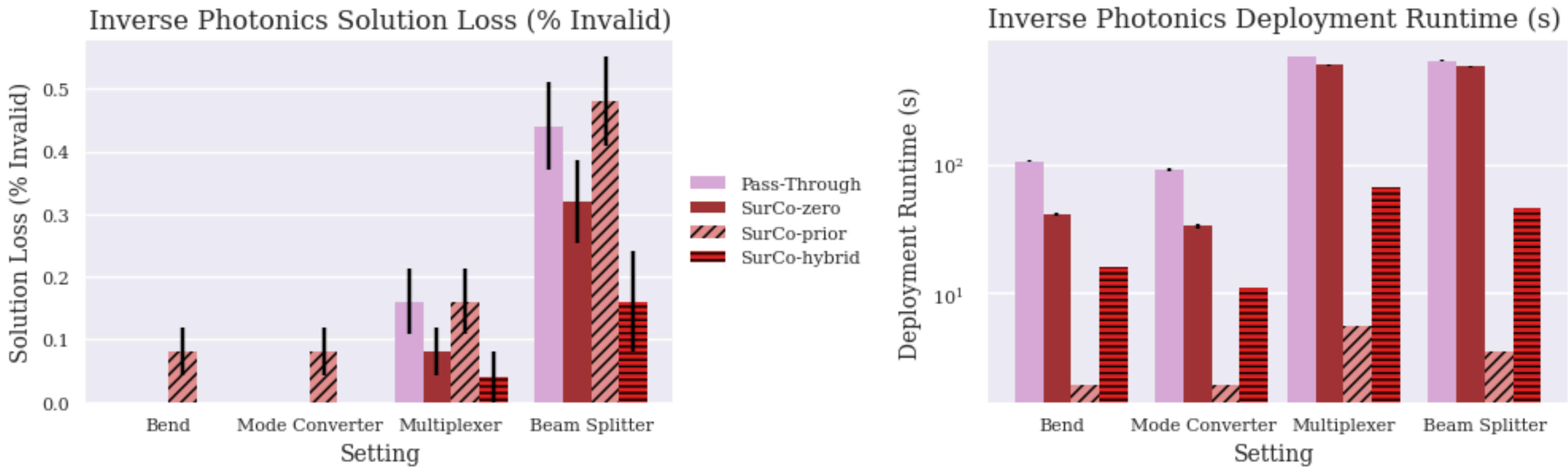
[1]Schubert et al. ACS Photonics 2022
[2]Vlastelica et al. ICLR 2019
Dataset: https://github.com/google/ceviche-challenges

Results – Table Sharding

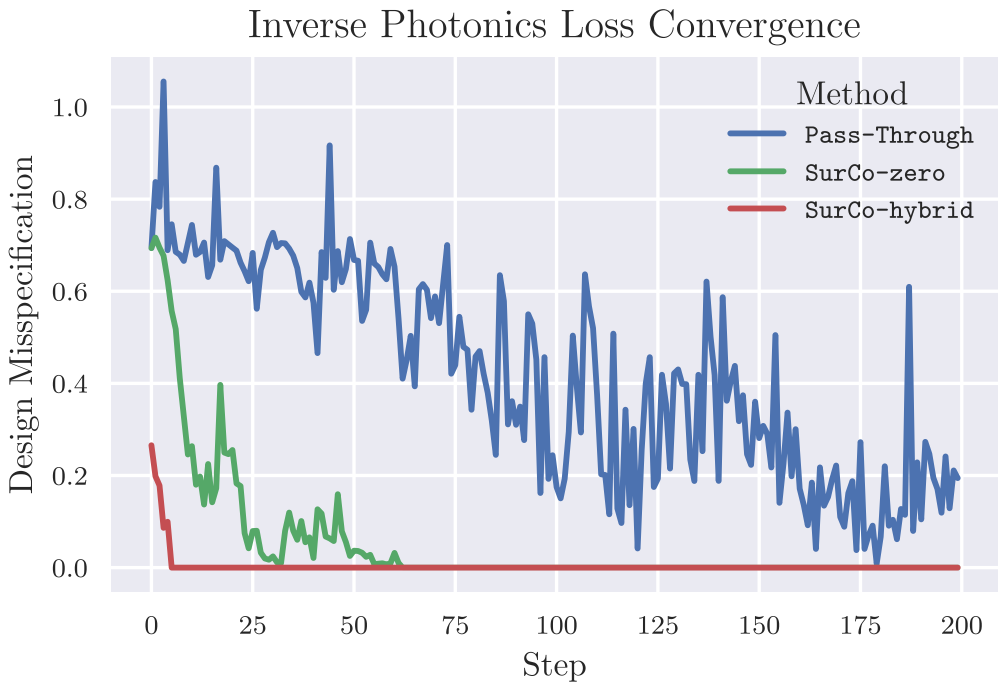




Results – Inverse Photonics

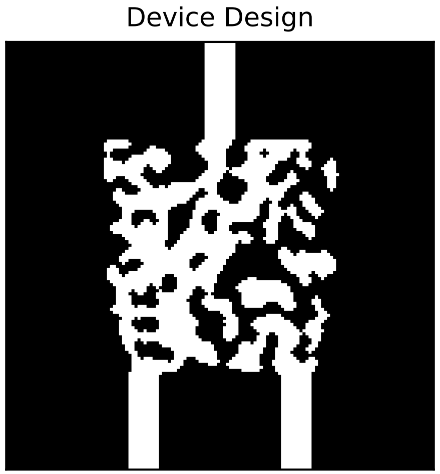


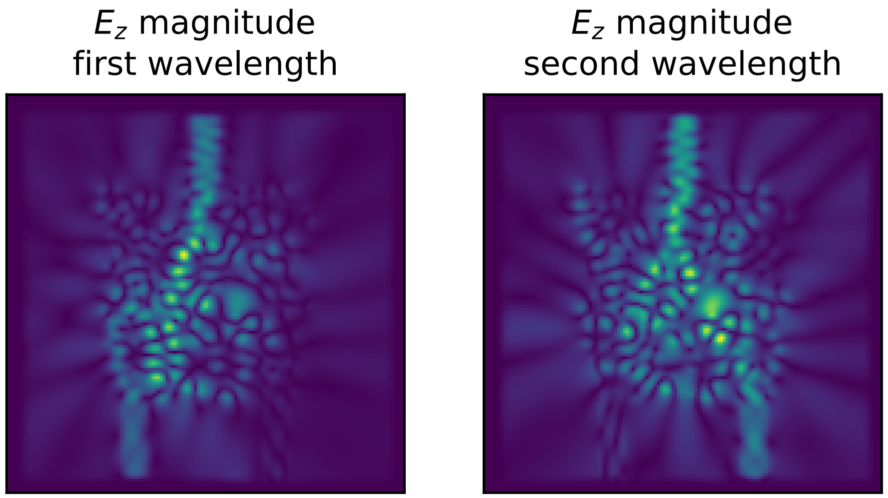
Inverse photonics Convergence comparison + Solution example



Takeaways:

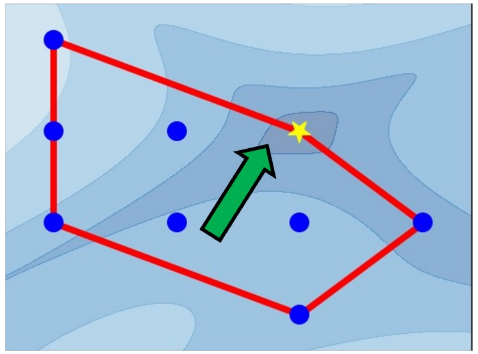
- SurCo-Zero finds loss-0 solutions quickly
- SurCo-Hybrid uses offline training data to get a head start





Wavelength division multiplexer

Conclusion



- Handle industrial applications with differentiable optimization
- High-quality solutions to combinatorial nonlinear optimization by finding linear surrogates
  - Sometimes we can find "easier" surrogate problems that solve much more difficult instances
- SurCo works in several data settings
  - Zero-shot vs Offline training
  - One step inference vs fine-tuning

Thanks!