SurCo: Learning Linear SURrogates for COmbinatorial Nonlinear Optimization Problems

Anonymous Author(s)
Affiliation
Address
email

Abstract

Optimization problems with nonlinear cost functions and combinatorial constraints appear in many real-world applications but remain challenging to solve efficiently compared to their linear counterparts. To bridge this gap, we propose ***SurCo*** that learns linear Surrogate costs which can be used in existing Combinatorial solvers to output good solutions to the original nonlinear combinatorial optimization problem. The surrogate costs are learned end-to-end with nonlinear loss by differentiating through the linear surrogate solver, combining the flexibility of gradient-based methods with the structure of linear combinatorial optimization. We propose three SurCo variants: SurCo − zero for individual nonlinear problems, SurCo − prior for problem distributions, and SurCo − hybrid to combine both distribution and problem-specific information. We give theoretical intuition motivating SurCo, and evaluate it empirically. Experiments show that SurCo finds better solutions faster than state-of-the-art and domain expert approaches in real-world optimization problems such as embedding table sharding, inverse photonic design, and nonlinear route planning.

1 Introduction

Combinatorial optimization problems with linear objective functions such as mixed integer linear programming (MILP) (Wolsey, 2007), and occasionally linear programming (LP) (Chvatal et al., 1983), have been extensively studied in operations research (OR). The resulting high-performance solvers like Gurobi (Gurobi Optimization, LLC, 2022) can solve industrial-scale optimization problems with tens of thousands of variables in a few minutes.

However, even with perfect solvers, one issue remains: the cost functions $f(\boldsymbol{x})$ in many practical problems are *nonlinear*, and the highly-optimized solvers mainly handle linear or convex formulations while real-world problems have less constrained objectives. For example, in embedding table sharding (Zha et al., 2022a) one needs to distribute embedding tables to multiple GPUs for the deployment of recommendation systems. Due to the batching behaviors within a single GPU and communication cost among different GPUs, the overall latency (cost function) in this application depends on interactions of multiple tables and thus can be highly nonlinear (Zha et al., 2022a).

To obtain useful solutions to real-world problems, one may choose to directly optimize the nonlinear cost, which can be the black-box output of a simulator (Gosavi et al., 2015; Ye et al., 2019), or the output of a cost estimator learned by machine learning techniques (e.g., deep models) from offline data (Steiner et al., 2021; Koziel et al., 2021; Wang et al., 2021b; Cozad et al., 2014). However, many of these direct optimization approaches either rely on human-defined heuristics (e.g., greedy (Korte & Hausmann, 1978; Reingold & Tarjan, 1981; Wolsey, 1982), local improvement (Voß et al., 2012; Li et al., 2021)), or resort to general nonlinear optimization techniques like gradient descent (Ruder, 2016), reinforcement learning (Mazyavkina et al., 2021), or evolutionary algorithms (Simon, 2013).

Submitted to ICML 2023 Workshop: Sampling and Optimization in Discrete Space. Do not distribute.

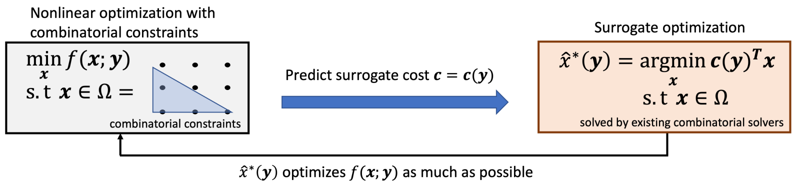
Nonlinear optimization with combinatorial constraints
$\min_x f(x; y)$
s.t $x \in \Omega =$
combinatorial constraints
Predict surrogate cost $c = c(y)$
Surrogate optimization
$\hat{x}^*(y) = \mathrm{argmin}_x\, c(y)^T x$
s.t $x \in \Omega$
solved by existing combinatorial solvers
$\hat{x}^*(y)$ optimizes $f(x; y)$ as much as possible

Figure 1: Overview of our proposed framework SurCo.

While these approaches can work in certain settings, they may lead to a slow optimization process, in particular when the cost function is expensive to evaluate, and they often ignore the combinatorial nature of most real-world applications.

In this work, we propose a systematic framework ***SurCo*** that leverages existing efficient combinatorial solvers to find solutions to nonlinear combinatorial optimization problems arising in real-world scenarios. When only one nonlinear *differentiable* cost $f(\boldsymbol{x})$ needs to be minimized, we propose SurCo-zero that optimizes a *linear surrogate* cost $\hat{\boldsymbol{c}}$ so that the *surrogate optimizer* (SO) $\min_{\boldsymbol{x}\in\Omega} \hat{\boldsymbol{c}}^\top \boldsymbol{x}$ outputs a solution that is expected to be optimal w.r.t. the *original* nonlinear cost $f(\boldsymbol{x})$. Due to its linear nature, SO can be solved efficiently with existing solvers, and the surrogate cost $\hat{\boldsymbol{c}}$ can be optimized in an end-to-end manner by back-propagating *through* the solver via methods proposed in previous work (Pogančić et al., 2019; Niepert et al., 2021; Berthet et al., 2020).

Thus, SurCo is a general-purpose method for solving combinatorial nonlinear optimization. Off-the-shelf nonlinear optimizers are often not directly applicable to these problem domains and often require domain-specific solution methodologies to give high-quality solutions in a reasonable amount of time, and solution prediction methods fail to give combinatorially feasible solutions without problem-specific intervention. Here, learning a linear surrogate problem ensures that the surrogate solver is practically efficient, yields gradient information for offline training, and generates solutions that are combinatorially feasible.

When solving a family of nonlinear differentiable functions $f(\boldsymbol{x};\boldsymbol{y})$ parameterized by instance description $\boldsymbol{y}$, the surrogate *coefficients* $\hat{\boldsymbol{c}}(\boldsymbol{y};\boldsymbol{\theta})$ are learned on a set of optimization instances (called the training set $\{\boldsymbol{y}_i\}$), by optimizing the parameters $\boldsymbol{\theta}$. For an unseen held-out instance $\boldsymbol{y}'$, we propose SurCo-prior that directly optimizes linear SO: $\hat{\boldsymbol{x}}^*(\boldsymbol{y}') := \arg\min_{\boldsymbol{x}\in\Omega(\boldsymbol{y}')} \hat{\boldsymbol{c}}^\top(\boldsymbol{y}';\boldsymbol{\theta})\boldsymbol{x}$ to get the solution, avoiding optimizing the cost $f(\boldsymbol{x};\boldsymbol{y}')$ from scratch. Based on the solution predicted by SurCo-prior, we also propose SurCo-hybrid that fine-tunes the surrogate costs $\hat{\boldsymbol{c}}$ with SurCo-zero to leverage both domain knowledge synthesized offline and information about the specific instance. We provide a comprehensive description of SurCo in Section 3.

We evaluate SurCo in three settings: embedding table sharding (Zha et al., 2022a), photonic inverse design (Schubert et al., 2022), and nonlinear route planning Fan et al. (2005). In the on-the-fly setting, SurCo-zero achieves higher quality solutions in comparable or less runtime, thanks to the help of an efficient combinatorial solver. in SurCo-prior, our method obtains better solutions in held-out problems compared to other methods that require training (e.g., reinforcement learning).

We compare SurCo at a high level with related work integrating learning and optimization at the end of our paper. We additionally present theoretical intuition that helps motivate why training a model to predict surrogate linear coefficients may exhibit better sample complexity than previous approaches that directly predict the optimal solution (Li et al., 2018; Ban & Rudin, 2019).

2 Problem Specification

Our goal is to solve the following nonlinear optimization problem describe by $\boldsymbol{y}$:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x};\boldsymbol{y}) \qquad \text{s.t.} \quad \boldsymbol{x}\in\Omega(\boldsymbol{y}) \tag{1}$$

where $\boldsymbol{x}\in\mathbb{R}^n$ are the n variables to be optimized, $f(\boldsymbol{x};\boldsymbol{y})$ is the nonlinear differentiable cost function to be minimized, $\Omega(\boldsymbol{y})$ is the feasible region, typically specified by linear (in)equalities and integer constraints, and $\boldsymbol{y}\in Y$ are the problem instance parameters drawn from a distribution $\mathcal{D}$ over Y. For example, in the traveling salesman problem, $\boldsymbol{y}$ can be the distance matrix among cities.

Differentiable cost function. The nonlinear cost function $f(\boldsymbol{x};\boldsymbol{y})$ can either be given analytically, or the result of a simulator made differentiable via finite differencing (e.g., JAX (Bradbury et al., 2018)).

Methods	Applicable to nonlinear objective	Objective can be free form	Training Set	Generalize to new instances	Combinatorial constraints
Gradient Descent	Yes	Yes	N/A	No	No
Evolutionary Algorithm	Yes	Yes	N/A	No	No
Nonlinear combinatorial solvers	Yes	No	N/A	No	Yes
Learning direct mapping	Yes	Yes	$\{\boldsymbol{y}_i, \boldsymbol{x}_i^*\}$	Yes	No
Predict-then-optimize	Limited	No	$\{\boldsymbol{y}_i, \boldsymbol{x}_i^*\}$	Yes	Yes
SurCo (proposed)	Yes	Yes	$\{\boldsymbol{y}_i\}$	Yes	Yes

Table 1: Conceptual comparison of optimizers (both traditional and ML-guided). Our approach (SurCo) can handle nonlinear objective without a predefined analytical form, does not require pre-computed optimal solutions in its training set, can handle combinatorial constraints (via commercial solvers it incorporates), and can generalize to unseen instances.

If the cost function $f(\boldsymbol{x}; \boldsymbol{y})$ is not differentiable as in one of our experimental settings, we can use a cost model that is learned from an offline dataset, often generated via sampling multiple feasible solutions within $\Omega(\boldsymbol{y})$, and recording their costs. In this work, we assume the following of $f(\boldsymbol{x}; \boldsymbol{y})$:

Assumption 2.1 (Differentiable cost function)**.** During optimization, the cost function $f(\boldsymbol{x}; \boldsymbol{y})$ and its partial derivative $\partial f / \partial \boldsymbol{x}$ are accessible.

Learning a good nonlinear cost model f is non-trivial for practical applications (e.g., AlphaFold (Jumper et al., 2021), Density Functional Theory (Nagai et al., 2020), cost model for embedding tables (Zha et al., 2022a)) and is beyond the scope of this work.

Evaluation Metric. We mainly focus on two aspects: the solution quality evaluated by $f(\hat{\boldsymbol{x}}; \boldsymbol{y})$, and the number of queries of f during optimization to achieve the solution $\hat{\boldsymbol{x}}$. For both, smaller measurements are favorable, i.e., fewer query of f to get solutions closer to global optimum.

When $f(\boldsymbol{x}; \boldsymbol{y})$ is linear w.r.t $\boldsymbol{x}$, and the feasible region $\Omega(\boldsymbol{y})$ can be encoded using mixed integer programs, the problem can be solved using existing scalable optimization solvers. When $f(\boldsymbol{x}; \boldsymbol{y})$ is nonlinear, we propose `SurCo` that learns a surrogate linear objective function, which allow us to leverage these existing scalable optimization solvers, and results in a solution that has high quality with respect to the original hard-to-encode objective function $f(\boldsymbol{x}; \boldsymbol{y})$.

3 `SurCo`: Learning Linear Surrogates

`SurCo-zero`: on-the-fly optimization. We start from the simplest case where we focus on a single instance with $f(\boldsymbol{x}) = f(\boldsymbol{x}; \boldsymbol{y})$ and $\Omega = \Omega(\boldsymbol{y})$. `SurCo-zero` optimizes the following objective:

$$(\texttt{SurCo-zero}): \quad \min_{\boldsymbol{c}} \mathcal{L}_{\texttt{zero}}(\boldsymbol{c}) := f(\boldsymbol{g}_\Omega(\boldsymbol{c})) \tag{2}$$

where the surrogate optimizer $\boldsymbol{g}_\Omega : \mathbb{R}^n \mapsto \mathbb{R}^n$ is the output of certain combinatorial solvers with linear cost weight $\boldsymbol{c} \in \mathbb{R}^n$ and feasible region $\Omega \subseteq \mathbb{R}^n$. For example, $\boldsymbol{g}_\Omega$ can be the following:

$$\boldsymbol{g}_\Omega(\boldsymbol{c}) := \arg\min_{\boldsymbol{x}} \boldsymbol{c}^\top \boldsymbol{x} \quad \text{s.t. } \boldsymbol{x} \in \Omega := \{A\boldsymbol{x} \le \boldsymbol{b}, \boldsymbol{x} \in \mathbb{Z}^n\} \tag{3}$$

which is the output of a MILP solver. Thanks to previous works (Ferber et al., 2020; Pogančić et al., 2019), we can efficiently compute the partial derivative $\partial \boldsymbol{g}_\Omega(\boldsymbol{c}) / \partial \boldsymbol{c}$. Intuitively, this means that $\boldsymbol{g}_\Omega(\boldsymbol{c})$ can be *backpropagated* through. Since f is also differentiable with respect to the solution it is evaluating, we thus can optimize Eqn. 2 in an end-to-end manner using any gradient-based optimizer:

$$\boldsymbol{c}(t+1) = \boldsymbol{c}(t) - \alpha \frac{\partial \boldsymbol{g}_\Omega}{\partial \boldsymbol{c}} \frac{\partial f}{\partial \boldsymbol{x}}, \tag{4}$$

where α is the learning rate. The procedure starts from a randomly initialized $\boldsymbol{c}(0)$ and converges at a local optimal solution of $\boldsymbol{c}$. While Eqn. 2 is still nonlinear optimization and there is no guarantee about the quality of the final solution $\boldsymbol{c}$, we argue that optimizing Eqn. 2 is better than optimizing the original nonlinear cost $\min_{\boldsymbol{x} \in \Omega} f(\boldsymbol{x})$. Furthermore, while we cannot guarantee optimality, we guarantee feasibility by leveraging a linear combinatorial solver.

Intuitively, instead of optimizing directly over the solution space $\boldsymbol{x}$, we optimize over the space of surrogate costs $\boldsymbol{c}$, and delegate the combinatorial feasibility requirements of the nonlinear problem to SoTA combinatorial solvers. Compared to naive approaches that directly optimize $f(\boldsymbol{x})$ via general optimization techniques, our method readily handles complex constraints of the feasible regions, and thus makes the optimization procedure easier. Furthermore, it also helps escape from local minima, thanks to the embedded search component of existing combinatorial solvers (e.g., branch-and-bound (Land & Doig, 2010) in MILP solvers). As we see in the experiments, this is particularly important when the problem becomes large-scale with more local optima. This approach works well when we are optimizing individual instances and may not have access to offline training data or the training time is cost-prohibitive.

Limitation. Note that due to linear surrogate, our approach will always return a vertex in the feasible region, while the solution to the original nonlinear objective may be in the interior. We leave this limitation for future work. In many real-world settings, such as in the three domains we tested, the solutions are indeed on the vertices of feasible regions.

`SurCo-prior`: offline surrogate training. We now consider a more general case where we have N optimization instances, each parameterized by an instance description $\boldsymbol{y}_i$, $i = 1 \ldots N$, and we want to find their solutions to a *collection* of nonlinear loss functions $f(\boldsymbol{x}; \boldsymbol{y}_i)$ simultaneously. Here we write $\mathcal{D}_{\text{train}} := \{\boldsymbol{y}_i\}_{i=1}^N$ as the training set. A naive approach is just to apply `SurCo-zero` N times, which leads to N independent surrogate costs $\{\boldsymbol{c}_i\}_{i=1}^N$. However, this approach does not consider two important characteristics. *First*, it fails to leverage possible relationship between the instance descriptor $\boldsymbol{y}_i$ and its associated surrogate cost $\boldsymbol{c}_i$, since every surrogate cost is independently estimated. *Second*, it fails to learn any useful knowledge from the N instances after optimization. As a result, for an unseen instance, the entire optimization process needs to be conducted again, which is slow. This motivates us to add a surrogate cost *model* $\hat{\boldsymbol{c}}(\boldsymbol{y}; \boldsymbol{\theta})$ into the optimization as a regularizer:

$$(\texttt{SurCo-prior-}\lambda): \quad \min_{\boldsymbol{\theta}, \{\boldsymbol{c}_i\}} \mathcal{L}_{\texttt{prior}}(\boldsymbol{\theta}, \{\boldsymbol{c}_i\}; \lambda) := \sum_{i=1}^N f(\boldsymbol{g}_{\Omega(\boldsymbol{y}_i)}(\boldsymbol{c}_i); \boldsymbol{y}_i) + \lambda \|\boldsymbol{c}_i - \hat{\boldsymbol{c}}(\boldsymbol{y}_i; \boldsymbol{\theta}))\|_2$$

The regressor model $\hat{\boldsymbol{c}}(\boldsymbol{y}; \boldsymbol{\theta})$ directly predicts the surrogate cost from the instance description. The form of the regressor can be a neural network, in which $\boldsymbol{\theta}$ is its parameters. Note that when $\lambda = 0$, it reduces to N independent optimizations, while when $\lambda > 0$, the surrogate costs $\{\boldsymbol{c}_i\}$ interact with each other. With the regressor, we distill knowledge gained from the optimization procedure into $\boldsymbol{\theta}$, which can be used for an unseen instance $\boldsymbol{y}'$. Indeed, we use the learned regressor model to predict the surrogate cost $\boldsymbol{c}' = \hat{\boldsymbol{c}}(\boldsymbol{y}'; \boldsymbol{\theta})$, and directly solve the *surrogate optimization* (SO):

$$\hat{\boldsymbol{x}}^*(\boldsymbol{y}') = \arg\min_{\boldsymbol{x} \in \Omega(\boldsymbol{y})} \hat{\boldsymbol{c}}^\top(\boldsymbol{y}'; \boldsymbol{\theta})\boldsymbol{x} \tag{5}$$

A special case is when $\lambda \to +\infty$, we learn the network parameters $\boldsymbol{\theta}$ instead of surrogate costs:

$$(\texttt{SurCo-prior}): \quad \min_{\boldsymbol{\theta}} \mathcal{L}_{\texttt{prior}}(\boldsymbol{\theta}) := \sum_{i=1}^N f(\boldsymbol{g}_{\Omega(\boldsymbol{y}_i)}(\hat{\boldsymbol{c}}(\boldsymbol{y}_i; \boldsymbol{\theta})); \boldsymbol{y}_i)$$

This approach is useful when the goal is to find high-quality solutions for unseen instances of a problem distribution when the upfront cost of offline training is acceptable but the cost of optimizing on-the-fly is prohibitive. Here, we require access to a distribution of training optimization problems, but at test time only require the feasible region and not the nonlinear objective. Different from predict-then-optimize Elmachtoub & Grigas (2022a); Ferber et al. (2020) or ML optimizers Ban & Rudin (2019), we do not require the optimal solution $\{\boldsymbol{x}_i^*\}_{i=1}^N$ as part of the training set.

`SurCo-hybrid`: fine-tuning a predicted surrogate. Naturally, we consider `SurCo-hybrid`, a hybrid approach which initializes the coefficients of `SurCo-zero` with the coefficients predicted from `SurCo-prior` which was trained on offline data. This allows `SurCo-hybrid` to start out optimization from an initial prediction that has good performance for the distribution at large but which is then fine-tuned for the specific instance. Formally, we initialize $\boldsymbol{c}(0) = \hat{\boldsymbol{c}}(\boldsymbol{y}_i; \boldsymbol{\theta})$ and then continue optimizing $\boldsymbol{c}$ based on the update from `SurCo-zero`. This approach is geared towards optimizing the nonlinear objective using a high-quality initial prediction that is based on the problem distribution and then fine-tuning the objective coefficients based on the specific problem instance at test time. Here, high performance comes at the runtime cost of both having to train offline on a problem distribution as well as performing fine-tuning steps on-the-fly. However, this additional cost is often worthwhile when the main goal is to find the best possible solutions by leveraging synthesized domain knowledge in combination with individual problem instances as arises in chip design (Mirhoseini et al., 2021) and compiler optimization (Zhou et al., 2020).

4 Surrogate Costs vs Solution Prediction, A Theoretical Analysis

One of the key ingredient of our proposed methods (SurCo-prior and SurCo-hybrid) is to learn a model to predict surrogate cost $\boldsymbol{c}$ from instance description $\boldsymbol{y}$, which is in contrast with previous solution regression approaches that directly learn a mapping from problem description $\boldsymbol{y}$ to the solution $\boldsymbol{x}^*(\boldsymbol{y})$ (Ban & Rudin, 2019). A natural question arise: which one is better?

In this section, we give theoretical intuition to compare the two approaches using a simple 1-nearest-neighbor (1-NN) solution regressor (Fix, 1985). We first relate the number of samples needed to learn any mapping to its *Lipschitz constant* L, and then show that for the direct mapping $\boldsymbol{y} \mapsto \boldsymbol{x}^*(\boldsymbol{y})$, L can be very large. Therefore, there exist fundamental difficulties to learn such a mapping. When this happens, we can still find surrogate cost mapping $\boldsymbol{y} \mapsto \boldsymbol{c}^*(\boldsymbol{y})$ with finite L that leads to the optimal solution $\boldsymbol{x}^*(\boldsymbol{y})$ of the original nonlinear problems.

Lipschitz constant and sample complexity. Formally, consider fitting any mapping $\boldsymbol{\phi} : \mathbb{R}^d \supseteq Y \mapsto \mathbb{R}^m$ with a dataset $\mathcal{C} := \{\boldsymbol{y}_i, \boldsymbol{\phi}_i\}$. Here Y is a compact region with finite volume $\mathrm{vol}(Y)$. The Lipschitz constant L is the smallest number so that $\|\boldsymbol{\phi}(\boldsymbol{y}_1) - \boldsymbol{\phi}(\boldsymbol{y}_2)\|_2 \le L\|\boldsymbol{y}_1 - \boldsymbol{y}_2\|_2$ holds for any $\boldsymbol{y}_1, \boldsymbol{y}_2 \in Y$. The following theorem shows that if the dataset covers the space Y, we could achieve high accuracy prediction: $\|\boldsymbol{\phi}(\boldsymbol{y}) - \hat{\boldsymbol{\phi}}(\boldsymbol{y})\|_2 \le \epsilon$ for any $\boldsymbol{y} \in Y$.

Definition 4.1 (δ-cover)**.** A dataset $\mathcal{C} := \{(\boldsymbol{y}_i, \boldsymbol{\phi}_i)\}_{i=1}^N$ δ-covers the space Y, if for any $\boldsymbol{y} \in Y$, there exists at least one $\boldsymbol{y}_i$ so that $\|\boldsymbol{y} - \boldsymbol{y}_i\|_2 \le \delta$.

Lemma 4.2 (Sufficient condition of prediction with ϵ-accuracy)**.** *If the dataset $\mathcal{C}$ can (ϵ/L)-cover Y, then for any $\boldsymbol{y} \in Y$, a 1-nearest-neighbor regressor $\hat{\boldsymbol{\phi}}$ leads to $\|\hat{\boldsymbol{\phi}}(\boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{y})\|_2 \le \epsilon$.*

Lemma 4.3 (Lower bound of sample complexity for ϵ/L-cover)**.** *To achieve ϵ/L-cover of Y, the size of the dataset set $N \ge N_0(\epsilon) := \frac{\mathrm{vol}(Y)}{\mathrm{vol}_0}\left(\frac{L}{\epsilon}\right)^d$, where vol_0 is the volume of unit ball in d-dimension.*

Please find all proofs in the Appendix. While we do not rule out a more advanced regressor than 1-nearest-neighbor that could lead to better sample complexity, the lemmas demonstrate that the Lipschitz constant L plays an important role in sample complexity.

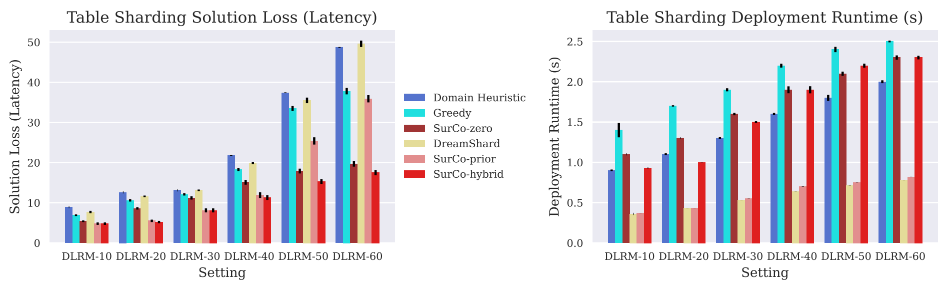

Figure 2: Table placement plan latency (**left**) and solver runtime (**right**). We evaluate SurCo against Dreamshard (Zha et al., 2022b), a SoTA offline RL solver, a domain-heuristic of assigning tables based on dimension, and a greedy heuristic based on the runtime increase. Striped approaches require pre-training.

Difference between Cost and Solution Regression. In the following we will show that in certain cases, the direct prediction $\boldsymbol{y} \mapsto \boldsymbol{x}^*(\boldsymbol{y})$ could have an infinitely large Lipschitz constant L. To show this, let us consider a general mapping $\boldsymbol{\phi} : \mathbb{R}^d \supseteq Y \mapsto \mathbb{R}^m$. Let $\boldsymbol{\phi}(Y)$ be the image of Y under mapping $\boldsymbol{\phi}$ and $\kappa(Y)$ be the number of connected components for region Y.

Theorem 4.4 (A case of infinite Lipschitz constant)**.** *If the minimal distance $d_{\min}$ for different connected components of $\boldsymbol{\phi}(Y)$ is strictly positive, and $\kappa(\boldsymbol{\phi}(Y)) > \kappa(Y)$, then the Lipschitz constant of the mapping $\boldsymbol{\phi}$ is infinite.*

Note that this theorem applies to a wide variety of combinatorial optimization problems. For example, when Y is a connected region and the optimization problem can be formulated as an integer programming, the optimal solution set $\boldsymbol{x}^*(Y) := \{\boldsymbol{x}^*(\boldsymbol{y}) : \boldsymbol{y} \in Y\}$ is a discrete set of integral vertices, so the theorem applies. Combined with analysis in Sec. 4, we know the mapping $\boldsymbol{y} \mapsto \boldsymbol{x}^*(\boldsymbol{y})$ is hard to learn even with a lot of samples.

We can see this more clearly with a concrete example in 2D space. Let the 1D instance description $y \in [0, \pi/2]$, and the feasible region is a convex hull of 3 vertices $\{(0,0),(0,1),(1,0)\}$. The nonlinear objective is simply $f(\boldsymbol{x}; y) := (x_1 \cos(y) + x_2 \sin(y))^2$, in which $\boldsymbol{x} = (x_1, x_2)$ is the 2D solution vector. The direct mapping $y \to \boldsymbol{x}^*$ maps a continuous region of instance descriptions (i.e., $y \in [0, \pi/2]$) into 2 disjoint regions points ($\boldsymbol{x}^* = (0,1)$ and $\boldsymbol{x}^* = (1,0)$), and thus according to Theorem 4.4, its Lipschitz constant must be infinite. In contrast, there exists a surrogate cost mapping $\boldsymbol{c}(y) = [\cos(y), \sin(y)]^\top$, and the mapping $y \to \boldsymbol{c}$ has finite Lipschitz constant (actually $L \le 1$) and can be learned easily.

5 Empirical Evaluation

We evaluate the variants of `SurCo` on three settings, embedding table sharding, inverse photonic design, and nonlinear route planning, with the first two being real-world industrial settings. Each setting consists of a family of problem instances with varying feasible region and nonlinear objective function. Additionally, both table sharding and inverse photonic design lack analytical formulations of the objective function which prevents them from being used by many off-the-shelf nonlinear solvers like SCIP (Achterberg, 2009).

Embedding Table Sharding. The task of sharding embedding tables arises in the deployment of large-scale neural network models which operate over both sparse and dense inputs (e.g., in recommendation systems (Zha et al., 2022a,b, 2023; Sethi et al., 2022)). Given T embedding tables and D homogeneous devices, the goal is to distribute the tables among the devices such that no device's memory limit is exceeded, while the tables are processed efficiently. Formally, let $x_{t,d}$ be the binary variable indicating whether table t is assigned to device d, and $\boldsymbol{x} := \{x_{t,d}\} \in \{0,1\}^{TD}$ be the collection of the variables. The optimization problem is $\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x}; \boldsymbol{y})$ where $\Omega(\boldsymbol{y}) := \{\boldsymbol{x} : \forall t, \sum_t x_{t,d} = 1, \forall d, \sum_t m_t x_{t,d} \le M\}$.

Here the problem description $\boldsymbol{y}$ includes table memory usage $\{m_t\}$, and capacity M of each device. $\sum_d x_{t,d} = 1$ means each table t should be assigned to exactly one device, and $\sum_d m_t x_{t,d} \le M$ means the memory consumption at each device d should not exceed its capacity. The nonlinear cost function $f(\boldsymbol{x}; \boldsymbol{y})$ is the *latency*, i.e., the runtime of the longest-running device. Due to shared computation (e.g., batching) among the group of assigned tables, and communication costs across devices, the objective is highly nonlinear. $f(\boldsymbol{x}; \boldsymbol{y})$ is well-approximated by a sharding plan runtime estimator proposed by Dreamshard (Zha et al., 2022b). Note that here, the runtime is approximated by a differentiable function since the real world deployment runtime isn't differentiable.

`SurCo` learns to predict $T \times D$ surrogate cost $\hat{c}_{t,d}$, one for each potential table-device assignment. During training, the gradients through the combinatorial solver $\partial \boldsymbol{g}/\partial \boldsymbol{c}$ are computed via CVXPY-Layers (Agrawal et al., 2019a), and the integrality constraints are relaxed. In practice, we obtained mostly integral solutions in that only one table on any given device was fractional. At test time, we solve for the integer solution using SCIP (Achterberg, 2009), a branch and bound MILP solver.

Settings. We evaluate `SurCo` on the public Deep Learning Recommendation Model (DLRM) dataset (Naumov et al., 2019). We consider 6 settings placing 10, 20, 30, 40, 50, and 60 tables on 4 devices, with a 5GB memory limit on GPU devices and 100 instances each (50 train, 50 test).

Baselines. `Greedy` allocates tables to devices based on local latency increase f, and the domain-expert algorithm `Domain-Heuristic` balances the aggregate dimension (Zha et al., 2022b). For `SurCo-prior`, we use Dreamshard, the SoTA embedding table sharding RL algorithm.

Results. Fig. 2, `SurCo-zero` finds lower latency sharding plans than the baselines, while it takes slightly longer than `Domain-Heuristic` and DreamShard due to taking optimization steps rather than building a solution with an RL policy. `SurCo-prior` obtains lower latency solutions in about the same time as DreamShard with a slight runtime increase from SCIP. Lastly, `SurCo-hybrid` obtains the best solutions and has runtime comparable to `SurCo-zero`. In smaller instances ($T \le 40$), `SurCo-prior` finds better solutions than its impromptu counterpart, `SurCo-zero`, likely by escaping local optima by training on a variety of examples. For larger instances with more tables available for placement, `SurCo-zero` performs better by optimizing for the test instances as opposed to `SurCo-prior` which only uses training data. Using `SurCo-hybrid`, we obtain the best solutions but incur the upfront pretraining cost and the deployment-time optimization cost.

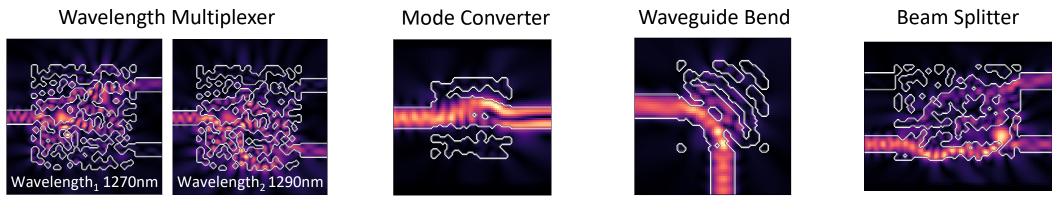

Figure 3: Inverse photonic design settings from the ceviche challenges Schubert et al. (2022) along with `SurCo-zero` solution designs and wavelength intensities. Light is fed in on the left and is routed at desired intensities to the output by designing the intermediate region. In the Wavelength Multiplexer setting, two wavelengths of interest are visualized as they are routed to different locations.

Inverse Photonic Design. Photonic devices play an essential role in high-speed communication (Marpaung et al., 2019), quantum computing (Arrazola et al., 2021), and machine learning hardware acceleration (Wetzstein et al., 2020). The photonic components can be encoded as a binary 2D grid, with each cell being filled or void. There are constraints on which binary patterns are physically manufacturable: only those that can be drawn by a physical brush instrument with a specific cross shape can be manufactured. It remains challenging to find manufacturable designs that satisfy design specifications like splitting beams of light. An example solution developed by `SurCo` is shown in Figure 3: beams are routed from the left to output locations, depending on wavelength. The solution is also manufacturable: a 3-by-3 cross can fit in all filled and void space. Given the design, existing work (Hughes et al., 2019) enables differentiation of the design misspecification cost, evaluated as how far off the transmission intensity of the wavelengths are from the desired output locations, with zero design loss meaning that the specification is satisfied. Researchers also develop the Ceviche Challenges (Schubert et al., 2022) a standard benchmark of inverse photonic design problems. Formally, a feasible design is a rectangle of pixels which are either filled or void where both the filled and void pixels can be expressed as a unions of the brush shape. Please see (Schubert et al., 2022) for an in depth description of the nonlinear objective and feasible region.

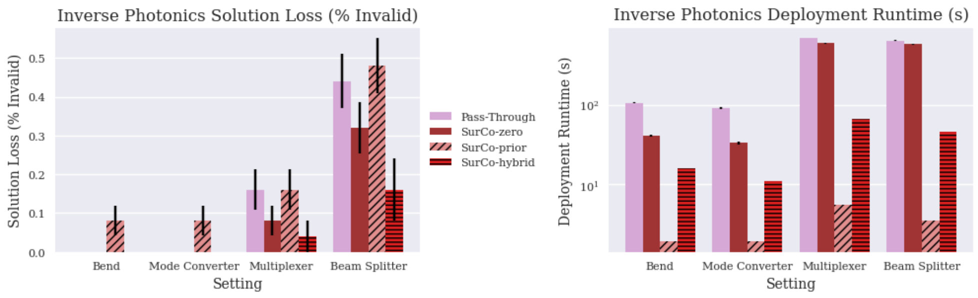

Figure 4: **Left** The solution loss (% of failed instances when the design loss is not 0), and **right** test time solver runtime in log scale. For both, lower is better. We compare against the Pass-Through gradient approach proposed in Schubert et al. (2022). We observe that `SurCo-prior` achieves similar success rates to the previous approach `Pass-Through` with a substantially improved runtime. Additionally, `SurCo-zero` runs comparably or faster, while finding more valid solutions than `Pass-Through`. `SurCo-hybrid` obtains valid solutions most often and is faster than `SurCo-zero` at the expense of pretraining. Striped approaches use pretraining.

Settings. We compare our approaches against the `Pass-Through` method (Schubert et al., 2022) on randomly generated instances of the four types of problems in Schubert et al. (2022): Waveguide Bend, Mode Converter, Wavelengths Division Multiplexer, and Beam Splitter. We generate 50 instances in each setting (25 training/25 test), randomly sampling the location of input and output waveguides, or "pipes" where we are taking in light and desire light to output. We fix the wavelengths themselves and so the problem description $\boldsymbol{y}$ contains an image description of the problem instance, where each pixel is either "fixed" or "designable". Further generation details are in the appendix. We evaluated several algorithms described in the appendix, such as genetic algorithms and derivative-free optimization, which failed to find feasible solutions. We consider two wavelengths (1270nm/1290nm), and optimize at a resolution of 40nm, visualizing the test results in Fig. 4.

Results. Fig. 4, `SurCo-zero` consistently finds as many or more valid devices compared to the `Pass-Through` baseline (Schubert et al., 2022). Additionally, since the on-the-fly solvers stop when they either find a valid solution, or reach a maximum of 200 steps, the runtime of `SurCo-zero` is slightly lower than the `Pass-Through` baseline. `SurCo-prior` obtains similar success rates as `Pass-Through` while taking two orders of magnitude less time as it does not require impromptu optimization. Lastly, `SurCo-hybrid` finds valid solutions more often than the other approaches. It also takes less runtime than the other on-the-fly approaches although it still requires optimization on-the-fly so it takes longer than `SurCo-prior`. In Fig. 5, `SurCo-zero` has smoother and faster convergence than `Pass-Through`.

Nonlinear Route Planning. Nonlinear route planning can arise where one wants to maximize the probability of arrival before a set time in graphs with random edges (Fan et al., 2005; Nikolova et al., 2006; Lim et al., 2013). These problems occur in risk-aware settings where operators need to maximize the probability of arriving before a critical time.

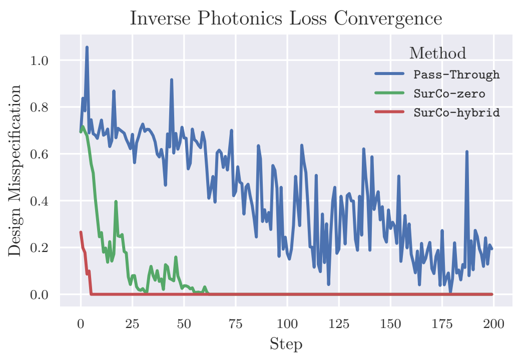

Figure 5: Inverse photonic design convergence. `SurCo-zero` smoothly lowers the loss while the baseline converges noisily. `SurCo-hybrid` fine-tunes an already high-quality solution.

Given a graph G with edge lengths coming from a random distribution, a pair of source and destination nodes s, t, and a time limit T that we would like to arrive before, we select a feasible $s-t$ path $P_{s,t}$ that maximizes the probability of arriving before the deadline $P[\text{length}(P_{s,t}) \leq T]$. If we assume that edge times are distributed according to a random normal distribution $t_e \sim \mathcal{N}(\mu_e, \sigma_e^2)$, then we could write the objective as maximizing $f(x;y) = \Phi\left((T - \sum_{e \in P_{s,t}} \mu_e)/\sqrt{\sum_{e \in P_{s,t}} \sigma_e^2}\right)$, with Φ being the cumulative distribution function of a standard Gaussian distribution, with the feasible region $\Omega(y)$ being the set of $s-t$ paths in the graph. Explicitly, the problem parameters $\boldsymbol{y}$ are the graph G, source and destination nodes s, t, time limit T, and the edge weight distributions given by means and variances μ_e, σ_e^2. We only consider the zero-shot setting since we need to solve the problem on-the-fly. `SurCo` trains surrogate edge costs $\hat{c}_e$, finds the shortest path using Bellman-Ford (Bellman, 1958), and differentiates using blackbox differentiation (Pogančić et al., 2019).

Settings. We run on a 5x5 grid graph with 25 draws of edge parameters $\mu_e \sim U(0.1, 1)$ and $\sigma_e^2 \sim U(0.1, 0.3) * (1 - \mu_e)$, with $U(a, b)$ being the uniform random distribution between a and b. We have deadline settings based on the length of the least expected time path (LET) which is simply the shortest path using μ_e as weights. We use loose, normal, and tight deadlines of 1.1 LET, 1 LET, and 0.9 LET respectively. The source and destination are oppose corners of the grid graph.

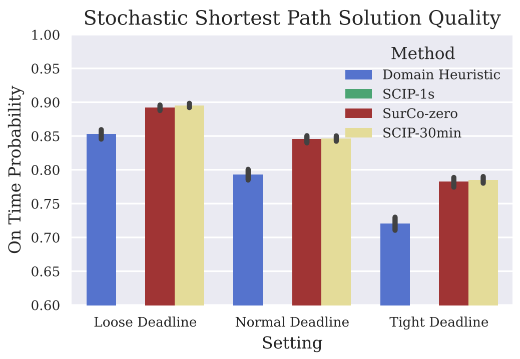

Figure 6: Comparison of nonlinear route planning probability of arriving on time. We compare against a domain heuristic (Nikolova et al., 2006) and SCIP (Achterberg, 2009). `SurCo-zero` outperforms the domain heuristic, and is similar to SCIP using less time. SCIP-1s fails to find feasible solutions.

Results. Fig. 6, we compare `SurCo-zero` against a domain-specific approach that minimizes a linear combination of mean and variance (Nikolova et al., 2006), and SCIP (Achterberg, 2009). In this setting, we focus on the zero-shot performance of `SurCo`, comparing it against two other zero-shot approaches. Furthermore, here we are able to encode the objective analytically into SCIP whereas the objectives of the other settings do not have readily-encodeable formulations, relying on neural networks or physical simulation. Since `SurCo-zero` and the domain approach take much less than 1 second, we use SCIP-1s and find that SCIP cannot find feasible solutions at that time scale. SCIP-30min demonstrates how well a general-purpose method can do given enough time, with SCIP timing out on all instances. We also find that `SurCo-zero` is able to obtain comparable solutions to SCIP-30min. Furthermore, SurCo-zero consistently outperforms the domain heuristic, finding paths that reach the deadline with $4.5\%, 6.5\%, 8.5\%$ times higher success rates in loose, normal, and tight deadlines. Finally, the domain heuristic only beats SurCo-zero in 2 instances.

6 Related Work

Differentiable Optimization. OptNet (Amos & Kolter, 2017) implicitly differentiates through KKT conditions: equations that determine the optimal solution. Followup work differentiated through linear programs (Wilder et al., 2019a), submodular optimization (Djolonga & Krause, 2017; Wilder et al., 2019a; Wang et al., 2020a), cone programs (Agrawal et al., 2019a,b), MaxSAT (Wang et al., 2019), mixed integer linear programs (Ferber et al., 2020; Mandi et al., 2020), integer linear programs (Mandi et al., 2020), dynamic programs Demirovic et al. (2020), blackbox discrete linear optimizers (Pogančić et al., 2019; Rolínek et al., 2020a,b), maximum likelihood estimation (Niepert et al., 2021), kmeans clustering (Wilder et al., 2019b), knapsack (Guler et al., 2022; Demirović et al., 2019), the cross-entropy method (Amos & Yarats, 2020), least squares (Pineda et al., 2022), SVM training (Lee et al., 2019). SurCo can use these surrogates as needed.

Task Based Learning. Task-based learning solves distributions of linear or quadratic optimization problems with the true objective hidden at test time but available for training (Elmachtoub & Grigas, 2022b; Donti et al., 2017; El Balghiti et al., 2019; Liu & Grigas, 2021; Hu et al., 2022). (Donti et al., 2021) predicts solutions for continuous nonlinear optimization. Machine learning can also guide combinatorial algorithms. Several approaches produce combinatorial solutions (Zhang & Dietterich, 1995; Khalil et al., 2017; Kool et al., 2018; Nazari et al., 2018; Zha et al., 2022a,b), but are limited to constructively building solutions for problems like routing, assignment, or covering. However, these approaches fail to handle more complex constraints. Other approaches set parameters that improve solver runtime (Khalil et al., 2016; Bengio et al., 2021). Similarly, a neural diving approach has been proposed for finding fast MILP solutions Nair et al. (2020), but requires iteratively solving subproblems which are nontrivial for nonlinear objectives.

Learning Latent Space for Optimization. We learn latent linear objectives to optimize nonlinear functions while other approaches learn latent embeddings for faster solving. FastMap (Faloutsos & Lin, 1995) learns latent embeddings for efficient search, with variants for graph optimization and shortest path (Cohen et al., 2018; Hu et al., 2022; Li et al., 2019). Wang et al. (2020b, 2021a); Yang et al. (2021); Zhao et al. (2022) use Monte Carlo Tree Search to learn to split the search space.

Mixed Integer Nonlinear Programming (MINLP). SurCo-zero solves some MINLP instances, optimizing nonlinear objectives over discrete linear regions, like some general solvers (Burer & Letchford, 2012; Belotti et al., 2013); however, scalability often requires problem-specific techniques.

7 Conclusion

We introduced SurCo, a method for learning linear surrogates for combinatorial nonlinear optimization problems. At its core, SurCo differentiates through the surrogate solver which maps the predicted coefficients to a combinatorially feasible solution, combining the flexibility of gradient-based optimization with the structure of combinatorial solvers. Our theoretical intuition for SurCo poses promising directions for future work in proving convergence guarantees or generalization bounds. We present three variants of SurCo, SurCo-zero for individual instances, SurCo-prior which trains a coefficient prediction model offline, and SurCo-hybrid which fine-tunes the coefficients predicted by SurCo-prior on individual test instances. We evaluated variants of SurCo against the state-of-the-art approaches on three domains, with two used in industry, obtaining better solutions faster in the embedding table sharding domain, quickly identifying viable photonic devices, and finding successful routes in stochastic path planning. Overall, SurCo trains linear surrogate coefficients to find high-quality solutions to tackle a broad class of combinatorial problems with nonlinear objectives where off-the-shelf solvers fail.

References

Achterberg, T. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019a.

Agrawal, A., Barratt, S., Boyd, S., Busseti, E., and Moursi, W. M. Differentiating through a cone program. *J. Appl. Numer. Optim*, 1(2):107–115, 2019b.

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pp. 136–145. PMLR, 2017.

Amos, B. and Yarats, D. The differentiable cross-entropy method. In *International Conference on Machine Learning*, pp. 291–302. PMLR, 2020.

Arrazola, J. M., Bergholm, V., Brádler, K., Bromley, T. R., Collins, M. J., Dhand, I., Fumagalli, A., Gerrits, T., Goussev, A., Helt, L. G., et al. Quantum circuits with many photons on a programmable nanophotonic chip. *Nature*, 591(7848):54–60, 2021.

Ban, G.-Y. and Rudin, C. The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108, 2019.

Bellman, R. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., and Mahajan, A. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 2013.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

Burer, S. and Letchford, A. N. Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2):97–106, 2012.

Chvatal, V., Chvatal, V., et al. *Linear programming*. Macmillan, 1983.

Cohen, L., Uras, T., Jahangiri, S., Arunasalam, A., Koenig, S., and Kumar, T. S. The fastmap algorithm for shortest path computations. In *IJCAI*, 2018.

Cozad, A., Sahinidis, N. V., and Miller, D. C. Learning surrogate models for simulation-based optimization. *AIChE Journal*, 60(6):2211–2227, 2014.

Demirović, E., J Stuckey, P., Bailey, J., Chan, J., Leckie, C., Ramamohanarao, K., and Guns, T. Predict+ optimise with ranking objectives: Exhaustively learning linear functions. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 1078–1085. International Joint Conferences on Artificial Intelligence, 2019.

Demirovic, E., J Stuckey, P., Guns, T., Bailey, J., Leckie, C., Ramamohanarao, K., and Chan, J. Dynamic programming for predict+ optimise. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 1444–1451. AAAI Press, 2020.

Djolonga, J. and Krause, A. Differentiable learning of submodular models. *Advances in Neural Information Processing Systems*, 30, 2017.

Donti, P., Amos, B., and Kolter, J. Z. Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 30, 2017.

Donti, P. L., Rolnick, D., and Kolter, J. Z. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=V1ZHVxJ6dSS`.

El Balghiti, O., Elmachtoub, A. N., Grigas, P., and Tewari, A. Generalization bounds in the predict-then-optimize framework. *Advances in neural information processing systems*, 32, 2019.

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022a.

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022b.

Faloutsos, C. and Lin, K.-I. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pp. 163–174, New York, NY, USA, 1995. Association for Computing Machinery. ISBN 0897917316. doi: 10.1145/223784.223812. URL `https://doi.org/10.1145/223784.223812`.

Fan, Y., Kalaba, R. E., and Moore, J. E. Arriving on time. *Journal of Optimization Theory and Applications*, 127:497–513, 2005.

Ferber, A., Wilder, B., Dilkina, B., and Tambe, M. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 1504–1511, 2020.

Fix, E. *Discriminatory analysis: nonparametric discrimination, consistency properties*, volume 1. USAF school of Aviation Medicine, 1985.

Gad, A. F. Pygad: An intuitive genetic algorithm python library, 2021.

Gosavi, A. et al. *Simulation-based optimization*. Springer, 2015.

Guler, A. U., Demirović, E., Chan, J., Bailey, J., Leckie, C., and Stuckey, P. J. A divide and conquer algorithm for predict+ optimize with non-convex problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 3749–3757, 2022.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

Hu, Y., Kallus, N., and Mao, X. Fast rates for contextual linear optimization. *Management Science*, 2022.

Hughes, T. W., Williamson, I. A., Minkov, M., and Fan, S. Forward-mode differentiation of maxwell's equations. *ACS Photonics*, 6(11):3010–3016, 2019.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

Korte, B. and Hausmann, D. An analysis of the greedy heuristic for independence systems. In *Annals of Discrete Mathematics*, volume 2, pp. 65–74. Elsevier, 1978.

Koziel, S., Çalık, N., Mahouti, P., and Belen, M. A. Accurate modeling of antenna structures by means of domain confinement and pyramidal deep neural networks. *IEEE Transactions on Antennas and Propagation*, 70(3):2174–2188, 2021.

Land, A. H. and Doig, A. G. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pp. 105–132. Springer, 2010.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10657–10665, 2019.

Li, J., Felner, A., Koenig, S., and Kumar, T. S. Using fastmap to solve graph problems in a euclidean space. In *Proceedings of the international conference on automated planning and scheduling*, volume 29, pp. 273–278, 2019.

Li, S., Yan, Z., and Wu, C. Learning to delegate for large-scale vehicle routing. *Advances in Neural Information Processing Systems*, 34:26198–26211, 2021.

Li, Z., Chen, Q., and Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in neural information processing systems*, 31, 2018.

Lim, S., Sommer, C., Nikolova, E., and Rus, D. Practical route planning under delay uncertainty: Stochastic shortest path queries. In *Robotics: Science and Systems*, volume 8, pp. 249–256. United States, 2013.

Liu, H. and Grigas, P. Risk bounds and calibration for a smart predict-then-optimize method. *Advances in Neural Information Processing Systems*, 34:22083–22094, 2021.

Liuzzi, G., Lucidi, S., and Rinaldi, F. Derivative-free methods for mixed-integer constrained optimization problems. *Journal of Optimization Theory and Applications*, 164(3):933–965, 2015.

Mandi, J., Stuckey, P. J., Guns, T., et al. Smart predict-and-optimize for hard combinatorial optimization problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 1603–1610, 2020.

Marpaung, D., Yao, J., and Capmany, J. Integrated microwave photonics. *Nature photonics*, 13(2): 80–90, 2019.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. A graph placement methodology for fast chip design. *Nature*, 594 (7862):207–212, 2021.

Nagai, R., Akashi, R., and Sugino, O. Completing density functional theory by machine learning hidden messages from molecules. *npj Computational Materials*, 6(1):1–8, 2020.

Nair, V., Bartunov, S., Gimeno, F., Von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Naumov, M., Mudigere, D., Shi, H. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C., Azzolini, A. G., Dzhulgakov, D., Mallevich, A., Cherniavskii, I., Lu, Y., Krishnamoorthi, R., Yu, A., Kondratenko, V., Pereira, S., Chen, X., Chen, W., Rao, V., Jia, B., Xiong, L., and Smelyanskiy, M. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019. URL `https://arxiv.org/abs/1906.00091`.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. Reinforcement learning for solving the vehicle routing problem. *Advances in neural information processing systems*, 31, 2018.

Niepert, M., Minervini, P., and Franceschi, L. Implicit mle: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34:14567–14579, 2021.

Nikolova, E., Kelner, J. A., Brand, M., and Mitzenmacher, M. Stochastic shortest paths via quasi-convex maximization. In *European Symposium on Algorithms*, pp. 552–563. Springer, 2006.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pineda, L., Fan, T., Monge, M., Venkataraman, S., Sodhi, P., Chen, R. T., Ortiz, J., DeTone, D., Wang, A., Anderson, S., et al. Theseus: A library for differentiable nonlinear optimization. *Advances in Neural Information Processing Systems*, 35:3801–3818, 2022.

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

Rapin, J. and Teytaud, O. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

Reingold, E. M. and Tarjan, R. E. On a greedy heuristic for complete matching. *SIAM Journal on Computing*, 10(4):676–681, 1981.

Rolínek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., and Martius, G. Optimizing rank-based metrics with blackbox differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7620–7630, 2020a.

Rolínek, M., Swoboda, P., Zietlow, D., Paulus, A., Musil, V., and Martius, G. Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision*, pp. 407–424. Springer, 2020b.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Schubert, M. F., Cheung, A. K. C., Williamson, I. A. D., Spyra, A., and Alexander, D. H. Inverse design of photonic devices with strict foundry fabrication constraints. *ACS Photonics*, 9(7): 2327–2336, 2022. doi: 10.1021/acsphotonics.2c00313.

Sethi, G., Acun, B., Agarwal, N., Kozyrakis, C., Trippel, C., and Wu, C.-J. Recshard: statistical feature-based memory optimization for industry-scale neural recommendation. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 344–358, 2022.

Simon, D. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

Steiner, B., Cummins, C., He, H., and Leather, H. Value learning for throughput optimization of deep learning workloads. In Smola, A., Dimakis, A., and Stoica, I. (eds.), *Proceedings of Machine Learning and Systems*, volume 3, pp. 323–334, 2021. URL `https://proceedings.mlsys.org/paper/2021/file/73278a4a86960eeb576a8fd4c9ec6997-Paper.pdf`.

Van Rossum, G. and Drake, F. L. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Voß, S., Martello, S., Osman, I. H., and Roucairol, C. *Meta-heuristics: Advances and trends in local search paradigms for optimization*. Springer Science & Business Media, 2012.

Wang, K., Wilder, B., Perrault, A., and Tambe, M. Automatically learning compact quality-aware surrogates for optimization problems. *Advances in Neural Information Processing Systems*, 33: 9586–9596, 2020a.

Wang, L., Fonseca, R., and Tian, Y. Learning search space partition for black-box optimization using monte carlo tree search. *Advances in Neural Information Processing Systems*, 33:19511–19522, 2020b.

Wang, L., Xie, S., Li, T., Fonseca, R., and Tian, Y. Sample-efficient neural architecture search by learning actions for monte carlo tree search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021a.

Wang, P.-W., Donti, P., Wilder, B., and Kolter, Z. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pp. 6545–6554. PMLR, 2019.

Wang, X., Liu, Y., Zhao, J., Liu, C., Liu, J., and Yan, J. Surrogate model enabled deep reinforcement learning for hybrid energy community operation. *Applied Energy*, 289:116722, 2021b.

Wetzstein, G., Ozcan, A., Gigan, S., Fan, S., Englund, D., Soljačić, M., Denz, C., Miller, D. A., and Psaltis, D. Inference in artificial intelligence with deep optics and photonics. *Nature*, 588(7836): 39–47, 2020.

Wilder, B., Dilkina, B., and Tambe, M. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1658–1665, 2019a.

Wilder, B., Ewing, E., Dilkina, B., and Tambe, M. End to end learning and optimization on graphs. *Advances in Neural Information Processing Systems*, 32, 2019b.

Wolsey, L. A. An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica*, 2(4):385–393, 1982.

Wolsey, L. A. Mixed integer programming. *Wiley Encyclopedia of Computer Science and Engineering*, pp. 1–10, 2007.

Yang, K., Zhang, T., Cummins, C., Cui, B., Steiner, B., Wang, L., Gonzalez, J. E., Klein, D., and Tian, Y. Learning space partitions for path planning. *Advances in Neural Information Processing Systems*, 34:378–391, 2021.

Ye, Y., Zhang, X., and Sun, J. Automated vehicle's behavior decision making using deep reinforcement learning and high-fidelity simulation environment. *Transportation Research Part C: Emerging Technologies*, 107:155–170, 2019.

Zha, D., Feng, L., Bhushanam, B., Choudhary, D., Nie, J., Tian, Y., Chae, J., Ma, Y., Kejariwal, A., and Hu, X. Autoshard: Automated embedding table sharding for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4461–4471, 2022a.

Zha, D., Feng, L., Tan, Q., Liu, Z., Lai, K.-H., Bhargav, B., Tian, Y., Kejariwal, A., and Hu, X. Dreamshard: Generalizable embedding table placement for recommender systems. In *Advances in Neural Information Processing Systems*, 2022b.

Zha, D., Feng, L., Luo, L., Bhushanam, B., Liu, Z., Hu, Y., Nie, J., Huang, Y., Tian, Y., Kejariwal, A., et al. Pre-train and search: Efficient embedding table sharding with pre-trained neural cost models. In *Sixth Conference on Machine Learning and Systems*, 2023.

Zhang, W. and Dietterich, T. G. A reinforcement learning approach to job-shop scheduling. In *IJCAI*, volume 95, pp. 1114–1120. Citeseer, 1995.

Zhao, Y., Wang, L., Yang, K., Zhang, T., Guo, T., and Tian, Y. Multi-objective optimization by learning space partition. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=FlwzVjfMryn`.

Zhou, Y., Roy, S., Abdolrashidi, A., Wong, D., Ma, P., Xu, Q., Liu, H., Phothilimtha, P., Wang, S., Goldie, A., et al. Transferable graph optimizers for ml compilers. *Advances in Neural Information Processing Systems*, 33:13844–13855, 2020.

A Proofs

Lemma A.1 (Sufficient condition of prediction with ϵ-accuracy). *If the dataset $\mathcal{C}$ can (ϵ/L)-cover Y, then for any $\boldsymbol{y} \in Y$, a 1-nearest-neighbor regressor $\hat{\boldsymbol{\phi}}$ leads to $\|\hat{\boldsymbol{\phi}}(\boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{y})\|_2 \leq \epsilon$.*

Proof. Since the dataset is a ϵ/L-cover, for any $\boldsymbol{y} \in Y$, there exists at least one $\boldsymbol{y}_i$ so that $\|\boldsymbol{y}-\boldsymbol{y}_i\|_2 \leq \epsilon/L$. Let $\boldsymbol{y}_{\mathrm{nn}}$ be the nearest neighbor of $\boldsymbol{y}$, and we have:

$$\|\boldsymbol{y} - \boldsymbol{y}_{\mathrm{nn}}\|_2 \leq \|\boldsymbol{y} - \boldsymbol{y}_i\|_2 \leq \epsilon/L \tag{6}$$

From the Lipschitz condition and the definition of 1-nearest-neighbor classifier ($\hat{\boldsymbol{\phi}}(\boldsymbol{y}) = \boldsymbol{\phi}(\boldsymbol{y}_{\mathrm{nn}})$), we know that

$$\|\boldsymbol{\phi}(\boldsymbol{y}) - \hat{\boldsymbol{\phi}}(\boldsymbol{y})\|_2 = \|\boldsymbol{\phi}(\boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{y}_{\mathrm{nn}})\|_2 \leq L\|\boldsymbol{y} - \boldsymbol{y}_{\mathrm{nn}}\|_2 \leq \epsilon \tag{7}$$

□

Lemma A.2 (Lower bound of sample complexity for ϵ/L-cover). *To achieve ϵ/L-cover of Y, the size of the dataset set $N \geq N_0(\epsilon) := \frac{\mathrm{vol}(Y)}{\mathrm{vol}_0}\left(\frac{L}{\epsilon}\right)^d$, where vol_0 is the volume of unit ball in d-dimension.*

Proof. We prove by contradiction. If $N < N_0(\epsilon)$, then for each training sample $(\boldsymbol{y}_i, \boldsymbol{\phi}_i)$, we create a ball $B_i := B(\boldsymbol{y}_i, \epsilon/L)$. Since

$$\mathrm{vol}\left(\bigcup_{i=1}^{N} B_i \cap Y\right) \leq \mathrm{vol}\left(\bigcup_{i=1}^{N} B_i\right) \leq \sum_{i=1}^{N} \mathrm{vol}(B_i) = N\mathrm{vol}_0\left(\frac{\epsilon}{L}\right)^d < \mathrm{vol}(Y) \tag{8}$$

Therefore, there exists at least one $\boldsymbol{y} \in Y$ so that $\boldsymbol{y} \notin B_i$ for any $1 \leq i \leq N$. This means that $\boldsymbol{y}$ is not ϵ/L-covered. □

Theorem 4.4 (A case of infinite Lipschitz constant). *If the minimal distance $d_{\min}$ for different connected components of $\boldsymbol{\phi}(Y)$ is strictly positive, and $\kappa(\boldsymbol{\phi}(Y)) > \kappa(Y)$, then the Lipschitz constant of the mapping $\boldsymbol{\phi}$ is infinite.*

Proof. Let $R_1, R_2, \ldots, R_K$ be the $K = \kappa(\boldsymbol{\phi}(Y))$ connected components of $\boldsymbol{\phi}(Y)$, and $Y_1, Y_2, \ldots, Y_J$ be the $J = \kappa(Y)$ connected components of Y. From the condition, we know that $\min_{k\neq k'} \mathrm{dist}(R_k, R_{k'}) = d_{\min} > 0$.

We have $R_k \cap R_{k'} = \emptyset$ for $k \neq k'$. Each R_k has a pre-image $S_k := \boldsymbol{\phi}^{-1}(R_k) \subseteq Y$. These pre-images $\{S_k\}_{k=1}^K$ form a partition of Y since

- $S_k \cap S_{k'} = \emptyset$ for $k \neq k'$ since any $\boldsymbol{y} \in Y$ cannot be mapped to more than one connected components;
- $\bigcup_{k=1}^K S_k = \bigcup_{k=1}^K \boldsymbol{\phi}^{-1}(R_k) = \boldsymbol{\phi}^{-1}\left(\bigcup_{k=1}^K R_k\right) = \boldsymbol{\phi}^{-1}(\boldsymbol{\phi}(S)) = S.$

Since $K = \kappa(\boldsymbol{\phi}(Y)) > \kappa(Y)$, by pigeonhole principle, there exists one Y_j that contains at least part of the two pre-images S_k and $S_{k'}$ with $k \neq k'$. This means that

$$S_k \cap Y_j \neq \emptyset, \quad S_{k'} \cap Y_j \neq \emptyset \tag{9}$$

Then we pick $\boldsymbol{y} \in S_k \cap Y_j$ and $\boldsymbol{y}' \in S_{k'} \cap Y_j$. Since $\boldsymbol{y}, \boldsymbol{y}' \in Y_j$ and Y_j is a connected component, there exists a continuous path $\gamma : [0,1] \mapsto Y_j$ so that $\gamma(0) = \boldsymbol{y}$ and $\gamma(1) = \boldsymbol{y}'$. Therefore, we have $\boldsymbol{\phi}(\gamma(0)) \in R_k$ and $\boldsymbol{\phi}(\gamma(1)) \in R_{k'}$. Let $t_0 := \sup\{t : t \in [0,1], \boldsymbol{\phi}(\gamma(t)) \in R_k\}$, then $0 \leq t_0 < 1$. For any sufficiently small $\epsilon > 0$, we have:

- By the definition of sup, we know there exists $t_0 - \epsilon \leq t' \leq t_0$ so that $\boldsymbol{\phi}(\gamma(t')) \in R_k$.
- Picking $t'' = t_0 + \epsilon < 1$, then $\boldsymbol{\phi}(\gamma(t'')) \in R_{k''}$ with some $k'' \neq k$.

On the other hand, by continuity of the curve γ, there exists a constant $C(t_0)$ so that $\|\gamma(t') - \gamma(t'')\|_2 \leq C(t_0)\|t' - t''\|_2 \leq 2C(t_0)\epsilon$. Then we have

$$L = \max_{\boldsymbol{y},\boldsymbol{y}'\in Y} \frac{\|\boldsymbol{\phi}(\boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{y}')\|_2}{\|\boldsymbol{y} - \boldsymbol{y}'\|_2} \geq \frac{\|\boldsymbol{\phi}(\gamma(t')) - \boldsymbol{\phi}(\gamma(t''))\|_2}{\|\gamma(t') - \gamma(t'')\|_2} \geq \frac{d_{\min}}{2C(t_0)\epsilon} \to +\infty \tag{10}$$

□

Task	Randomization
mode converter	randomize the right and left waveguide width
bend setting	randomize the waveguide width and length
beam splitter	randomize the waveguide separation, width and length
wavelength division multiplexer	randomize the input and output waveguide locations

Table 2: Task randomization of 4 different tasks in inverse photonic design.

B Experiment Details

B.1 Setups

Experiments are performed on a cluster of identical machines, each with 4 Nvidia A100 GPUs and 32 CPU cores, with 1T of RAM and 40GB of GPU memory. Additionally, we perform all operations in Python (Van Rossum & Drake, 2009) using Pytorch (Paszke et al., 2019). For embedding table placement, the nonlinear cost estimator is trained for 200 iterations and the offline-trained models of Dreamshard and `SurCo-prior` are trained against the pretrained cost estimator for 200 iterations. The DLRM Dataset Naumov et al. (2019) is available at `https://github.com/facebookresearch/dlrm_datasets`, and the dreamshard (Zha et al., 2022b) code is available at `https://github.com/daochenzha/dreamshard`. Additional details on dreamshard's model architecture and features can be obtained in the paper and codebase. Training time for the networks used in `SurCo-prior` and `SurCo-hybrid` are on average 8 hours for the inverse photonic design settings and 6, 21, 39, 44, 50, 63 minutes for DLRM 10, 20, 30, 40, 50, 60 settings respectively.

B.2 Network Architectures

B.2.1 Embedding Table Sharding

The table features are the same used in Zha et al. (2022b), and sinusoidal positional encoding Vaswani et al. (2017) is used as device features so that the learning model is able to break symmetries between the different tables and effectively group them onto homogeneous devices. The table and device features are concatenated and then fed into Dreamshard's initial fully-connected table encoding module to obtain scalar predictions $\hat{c}_{t,d}$ for each desired objective coefficient. The architecture is trained with the Adam optimizer with learning rate 0.0005. Here, we use the dreamshard backbone to predict coefficients for each table-device pair. We add more output dimensions to the dreamshard backbone, ensuring that we output the desired number of coefficients.

B.2.2 Inverse Photonic Design

Network architectures. The input design specification (a 2D image) is passed through a 3 layer convolutional neural network with ReLU activations and a final layer composed of filtering with the known brush shape. Then a tanh activation is used to obtain surrogate coefficients $\hat{\boldsymbol{c}}$, one component for each binary input variable. The architecture is trained with the Adam optimizer with learning rate 0.001.

This is motivated by previous work (Schubert et al., 2022) that also uses the fixed brush shape filter and tanh operation to transform the latent parameters into a continuous solution that is projected onto the space of physically feasible solutions.

In each setting, optimization is done on a binary grid of different sizes to meet fabrication constraints, namely that a 3 by 3 cross must fit inside each fixed and void location. In the beam splitter the design is an 80×60 grid, in mode converter it is a 40×40 grid, in waveguide bend it is a 40×40 grid, in wavelength division multiplexer it is an 80×80 grid.

Previous work formulated the projection as finding a discrete solution that minimized the dot product of the input continuous solution and proposed discrete solution. The authors then updated the continuous solution by computing gradients of the loss with respect to the discrete solution and using pass-through gradients to update the continuous solution. By comparison, our approach treats the projection as an optimization problem and updates the objective coefficients so that the resulting projected solution moves in the direction of the desired gradient.

To compute the gradient of this blackbox projection solver, we leverage the approach suggested by Pogančić et al. (2019) which calls the solver twice, once with the original coefficients, and again with coefficients that are perturbed in the direction of the incoming solution gradient as being an "improved solution". The gradient with respect to the input coefficients are then the difference between the "improved solution" and the solution for the current objective coefficients.

C Pseudocode

Here is the pseudocode for the different variants of our algorithm. Each of these leverage a differentiable optimization solver to differentiate through the surrogate optimization problem.

Algorithm 1 `SurCo-zero`

```
Input: feasible region $\Omega$, data $\boldsymbol{y}$, objective $f$
$\boldsymbol{c} \leftarrow \text{init\_surrogate\_coefs}(\boldsymbol{y})$
while not converged do
    $\boldsymbol{x} \leftarrow \arg\min_{\boldsymbol{x}\in\Omega(\boldsymbol{y})} \boldsymbol{c}^\top \boldsymbol{x}$
    loss $\leftarrow f(\boldsymbol{x}; \boldsymbol{y})$
    $\boldsymbol{c} \leftarrow \text{grad\_update}(\boldsymbol{c}, \nabla_{\boldsymbol{c}}\text{loss})$
end while
Return $\boldsymbol{x}$
```

Algorithm 2 `SurCo-prior` Training

```
Input: feasible region $\Omega$, data $\mathcal{D}_{\text{train}} = \{\boldsymbol{y}_i\}_{i=1}^N$, objective $f$
$\theta \leftarrow \text{init\_surrogate\_model}()$
while not converged do
    Sample batch $B = \{\boldsymbol{y}_i\}_i^k \sim \mathcal{D}_{\text{train}}$
    for $\boldsymbol{y} \in B$ do
        $\hat{\boldsymbol{c}} \leftarrow \hat{\boldsymbol{c}}(\boldsymbol{y}; \theta)$
        $\boldsymbol{x} \leftarrow \arg\min_{\boldsymbol{x}\in\Omega(\boldsymbol{y})} \boldsymbol{c}^\top \boldsymbol{x}$
        loss $+= f(\boldsymbol{x}; \boldsymbol{y})$
    end for
    $\theta \leftarrow \text{grad\_update}(\theta, \nabla_\theta\text{loss})$
end while
Return $\theta$
```

Algorithm 3 `SurCo-prior` Deployment

```
1: Input: feasible region $\Omega$, data $\mathcal{D}_{\text{train}} = \{\boldsymbol{y}_i\}_{i=1}^N$, objective $f$, test instance $\boldsymbol{y}_{\text{test}}$
2: $\theta \leftarrow$ train SurCo-prior$(\Omega, \mathcal{D}_{\text{train}}, f)$
3: $\boldsymbol{c} \leftarrow \hat{\boldsymbol{c}}(\boldsymbol{y}; \theta)$
4: $\boldsymbol{x} \leftarrow \arg\min_{\boldsymbol{x}\in\Omega(\boldsymbol{y})} \boldsymbol{c}^\top \boldsymbol{x}$
5: Return $\boldsymbol{x}$
```

Algorithm 4 `SurCo-hybrid`

```
1: Input: feasible region $\Omega$, data $\mathcal{D}_{\text{train}} = \{\boldsymbol{y}_i\}_{i=1}^N$, objective $f$, test instance $\boldsymbol{y}_{\text{test}}$
2: $\theta \leftarrow$ train SurCo-prior$(\Omega, \mathcal{D}_{\text{train}}, f)$
3: $\boldsymbol{c} \leftarrow \hat{\boldsymbol{c}}(\boldsymbol{y}; \theta)$
4: while not converged do
5:     $\boldsymbol{x} \leftarrow \arg\min_{\boldsymbol{x} \in \Omega(\boldsymbol{y})} \boldsymbol{c}^\top \boldsymbol{x}$
6:     loss $\leftarrow f(\boldsymbol{x}; \boldsymbol{y})$
7:     $\boldsymbol{c} \leftarrow$grad_update$(\boldsymbol{c}, \nabla_{\boldsymbol{c}}$loss$)$
8: end while
9: Return $\boldsymbol{x}$
```

D Additional Failed Baselines

SOGA - Single Objective Genetic Algorithm Using PyGAD (Gad, 2021), we attempted several approaches for both table sharding and inverse photonics settings. While we were able to obtain feasible table sharding solutions, they underperformed the greedy baseline by 20%. Additionally, they were unable to find physically feasible inverse photonics solutions. We varied between random, swap, inversion, and scramble mutations and used all parent selection methods but were unable to find viable solutions.

DFL - A Derivative-Free Library We could not easily integrate DFLGEN (Liuzzi et al., 2015) into our pipelines since it operates in fortran and we needed to specify the feasible region with python in the ceviche challenges. DFLINT works in python but took more than 24 hours to run on individual instances which reached a timeout limit. We found that the much longer runtime made this inapplicable for the domains of interest.

Nevergrad We enforced integrality in Nevergrad (Rapin & Teytaud, 2018) using choice variables which selected between 0 and 1. This approach was unable to find feasible solutions for inverse photonics in less than 10 hours. For table sharding we obtained solutions by using a choice variable for each table, selecting one of the available devices. This approach was not able to outperform the greedy baseline and took longer time so it was strictly dominated by the greedy approach.

Solution Prediction We made several attempts at training solution predictors for each of our domains. We label each problem instance with the best-known solution obtained (including those obtained via SurCo). Note that predicting feasible solutions to combinatorial optimization problems is nontrivial for general settings.

We evaluate solution prediction architectures in each setting. The models here match the architecture of SurCo-prior but the output is fed through a sigmoid transformation to get predictions in [0,1]. In nonlinear shortest path we use a GCN architecture and predict [0,1] whether edges are in the shortest s-t path. Not surprisingly, we found that predicting solutions to combinatorial problems is a nontrivial problem, further motivating the use of SurCo which ensures combinatorial feasibility of the generated solution.

Note that the solutions predicted by the networks may not be binary (and thus not feasible). We then round the individual decision variables to get binary predictions. Empirically, we found that our predictions are very close to binary, indicating that rounding is more a numerical exactness operation than an algorithmic decision, with the largest distance from any original to rounded value being 0.0008 for inverse photonics, 0.0001 for nonlinear shortest path, and 0.0007 for the assignment problem of table sharding.

We evaluate the results on unseen test instances in Table 3 and find that these solution prediction approaches don't yield combinatorially feasible solutions. We present machin learning performance in the table below to verify that the predictive models perform "well" in terms of standard machine learning evaluation even though they fail to generate feasible solutions.

We also iterate on table sharding to produce two more domain-specific approaches. We evaluate a model variant which assigns each table into one of the 4 devices using softmax, which empirically fails to yield feasible solutions that meet device memory limits for any of our instances. We further

Setting	Decision Variable Accuracy Average	Solution Accuracy	Solution Feasibility Rate
Inverse Photonics - Sigmoid	87%	0%	0%
Nonlinear Shortest Path - Sigmoid	95%	0%	0%
Table Sharding - Sigmoid	92%	0%	0%
Table Sharding - Softmax	88%	0%	0%
Table Sharding - Softmax + Iterative	70%	0%	100%

Table 3: Solution prediction results, most methods give infeasible solutions.

Setting	% Latency Increase vs Domain Heuristic (worst baseline)
DLRM-10	6%
DLRM-20	5%
DLRM-30	9%
DLRM-40	7%
DLRM-50	3%
DLRM-60	11%

Table 4: Comparison of only feasible solution prediction method against worst baseline.

develop a method called Softmax + Iterative which iteratively assigns the most likely table-device assignment as long as the device has enough memory to hold the device. Luckily, this Softmax + Iterative method empirically yields feasible solutions in this setting but we note that this approach is not guaranteed to terminate in feasible solutions, unlike SurCo. To see why Softmax + Iterative does not necessarily guarantee feasible termination, consider assigning 3 tables (2 small and 1 large) to 2 devices each with memory limit of 2, the small tables have memory 1 and the large table has memory 2. If the model's highest assignment probability is on the small tables being evenly distributed across devices, the algorithm will first assign the small tables to devices 1 and 2 but stall because it is unable to assign the large table since neither device has enough remaining capacity. We present results for this Softmax + Iterative approach compared to our domain heuristic which is the worst performing baseline in Table 4.

For each setting, we evaluate the three metrics:

- **Decision Variable Accuracy Average**, is the average percent of variables which are correctly predicted.
- **The solution accuracy**, is the rate of predicting the full solution correctly (all decision variables predicted correctly).
- **The solution feasibility rate**, is the percent of instances for which the predicted solution satisfies the constraints.