# Analyzing Local Structure in Predict+Optimize Loss Functions

Thomas Kuiper

Delft University of Technology

TU Delft

# Analyzing Local Structure in Predict+Optimize Loss Functions

by

Thomas Kuiper

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Monday December 8, 2025 at 15:30 AM.

| | | |
|---|---|---|
| Student number: | 5086217 | |
| Project duration: | November 1, 2024 – December 8, 2025 | |
| Thesis committee: | Dr. E. Demirović, | TU Delft, head-supervisor |
| | Dr. W. Böhmer, | TU Delft |
| | K. Sidorov, | TU Delft, daily supervisor |

Cover: Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under CC BY-NC 2.0 (Modified)
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

# Preface

*Kindness and honesty are the best problem solvers.*

This report is the culmination of all the work put into my masters thesis. I want to thank my supervisors for guiding me. First and foremost, Konstantin, my daily supervisor. He helped me immensely in completing this work, but also taught me the excitement of research and the occasional life lesson. His guidance was invaluable, and I will carry it with me for the rest of my life. Emir, my head supervisor, played a smaller but important role in my process. He kept the train on the rails, making sure I plan and think things through thoroughly. His feedback often helped me discover problems I had not thought about and brought me in the right direction. Thank you both!

Lastly, I want to thank my friends and family. They were there for me when I was struggling, helping me to keep believing in myself. But I also enjoyed explaining ”what I was actually doing” to them, having discussions or laughs. Thank you for supporting me! I know I can keep counting on you.

*Thomas Kuiper*
*Delft, November 2025*

# Abstract

Predict then optimize (P+O) is an emerging field that uses machine learning to predict variables for a combinatorial optimization (CO) problem. In this, it has to overcome the discontinuous nature of combinatorial problems. Many different solutions have been proposed, like SPO+[6], PFYL[2] and CaVE[18]. What they all have in common is that they give up some information about the problem to make the loss function continuous. To get a better idea of where the strengths and weaknesses lie of P+O losses, we want to find how much of this information on problem structure is retained. This brings us to the main research question: **How much of the local structure of regret do P+O loss functions follow? Are there ways to improve this?**

We found that all P+O methods tested struggled to consistently find locally optimal solutions, often leaving room for improvement in adjacent solutions. When locally optimal solutions were found, they were mostly already globally optimal. We found that P+O methods do not really consider local regret optimality except for the global solution. Since this solution often cannot be found, we reason that regret local search could improve these cases a lot.

We develop Decision Guided Learning (DGL) as a regret local search algorithm and find good improvements for smaller machine learning models and medium to no improvements for larger machine learning models. We reason that when models are large enough to approximate all the true optimal solutions, the need for regret local search becomes less.

# Contents

# 1 Introduction

Machine learning is a well-studied field. With it, a solution can be found for almost any prediction problem. This only holds, though, when viewing the prediction problem in isolation. Predict then Optimize (P+O) is the field that studies machine learning tasks specifically with downstream optimization in mind. Conventional prediction-focused machine learning falls short in this setting. Instead, a decision-focused approach has been shown to produce better results[6]. This involves including the optimization problem in the machine learning pipeline and learning, not based on the prediction quality, but the quality of the decision produced by the final optimization using said prediction.

Including a combinatorial optimization (CO) problem in the learning pipeline also brings new challenges. Training on a loss defined by the value of the decision leaves gradients zero almost everywhere. Since a small change in the prediction may not change the decision, and thus also not the value. And this assumes that we **can** even compute the gradient; this requires back-propagation over the optimization problem itself. Facing these challenges, researchers came up with creative solutions that tackle both of these problems, but in doing so, they have to sacrifice accuracy to both smoothen out the problem and make it differentiable.

Other works analyze P+O methods based on performance measures and computation time [12] [19]. In this work, we characterize P+O methods differently. Instead of measuring performance on different problems, we test the (dis)similarity of P+O loss functions to the fully informed regret landscape. The regret landscape holds all the problem structure, and thus, for CO problems, the discontinuities. By calculating the whole regret landscape, we can analyze where P+O methods give up information and if this might lead to situations where they fail.

There are several advantages of doing analysis on regret landscapes. First, we can analyze the entire shape of the loss function instead of focusing on a single point. In particular, this allows us to explore the local regret structure. Second, since we calculate the full landscape, we know what the optimal regret is. This allows us to measure both model and regret distance to the optimum. Together, these facts allow for reasoning beyond "lower regret is better." We can differentiate P+O methods by their local optimality and ability to find the optimal regret.

Regret landscape analysis also comes with a major downside. Calculating the whole regret landscape is practically infeasible for prediction models with more than a single parameter. Hence, in our research, we are limited to experiments with prediction models of this size.

The analysis is done by finding where P+O losses converge and comparing it with the objectives of the regret landscape. We define the objectives of the regret landscape by using a specialized notion of local optimality we call locality. This allows us to compare whether P+O losses follow the local structure of regret closely, more loosely, or follow something different altogether. This brings us to the main research question:

**How much of the local structure of regret do P+O loss functions follow? Are there ways to exploit this?**

This thesis is split into two parts in accordance with this question. First, the previously discussed local regret analysis. Second, a method to improve local regret in P+O loss functions. Our method, Decision Guided Learning (DGL), is a regret local search algorithm. It guides gradient descent towards regret-improving decisions by searching the local decisions and moving towards them by embedding their normal cones in objective space.

In the first part, we found that P+O losses do not follow the local structure of regret to a very close degree. All our tested P+O losses miss the local structure for around 50% of the tested problem instances. We show that, on average, this leads to a worse solution quality. When the local structure is matched, it is mostly matched around the globally optimal solution. The local regret landscape is mostly missed. We see a lot of potential for improvement, especially in aligning more often with regret locally optimal solutions.

For the second part, we found that our implementation of DGL significantly improved the rate at which the final solution matches local regret. DGL improved one of the methods, CaVE, from finding locally optimal solutions 40% of the time to 70%. We also tested DGL in more practical settings that include larger datasets and model sizes. We found that for smaller models, DGL can give around a 20% reduction in regrets. Increasing the model sizes reduces this improvement to eventually zero. We find that DGL can help augment other P+O methods, especially where models struggle, but when a model already performs well, it does not find much improvement.

The rest of the thesis is structured as follows. In Chapter 2, we define the problem and explain a key technique for the analysis. Chapter 3 describes the main techniques used in P+O methods. The analysis of P+O methods using the regret landscape is shown in Chapter 4. Section 4.1 motivates the approach, Section 4.2 explains the methodology in detail, and we show and analyze the outcomes in Section 4.3. Chapter 5 explains how DGL works, is implemented, and how it performs.

# 2 Preliminaries

## 2.1. Definitions

Borrowing notation from Mandi et al. [12], we define a parameterized Mixed Integer Linear Program (MILP) as follows.
**Definition 2.1.1** (problem description)**.**

$$\boldsymbol{x}^*(\boldsymbol{c}) = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, f(\boldsymbol{x}, \boldsymbol{c}) \tag{2.1}$$

$$s.t.\ \boldsymbol{g}(\boldsymbol{x}) \leq 0 \tag{2.2}$$

$$\boldsymbol{h}(\boldsymbol{x}) = 0 \tag{2.3}$$

The goal of the CO problem is to find an optimal solution $\boldsymbol{x}^*(\boldsymbol{c})$. An optimal solution is a solution $x \in \mathcal{F}$ that minimizes the objective function $f(\boldsymbol{x}, \boldsymbol{c})$, where $\mathcal{F}$ is the feasible space described by constraints $\boldsymbol{g}(\boldsymbol{x})$ and $\boldsymbol{h}(\boldsymbol{x})$.

In the setting of P+O, the objective costs of the optimization problem are unknown. Hence, the problem description is parameterized by the cost $c$. $c$ is generally referred to as the true cost, or the cost under perfect information. P+O models are trained to predict $c_{true}$. A prediction of $c_{true}$ is written as $c_{pred}$.

With these building blocks, we can formalize the objective of P+O: train a machine learning model that predicts $\boldsymbol{c}_{pred}$ such that the objective function $f(x, c_{true})$ is minimized.

Conventional machine learning losses can learn to find a $c_{pred}$ close to $c_{true}$ but can not minimize the objective function directly. For this, we need something different from prediction-focused losses, namely, decision-focused losses. These losses do not compare cost predictions, but instead base their loss value on the decisions $\boldsymbol{x}^*(\boldsymbol{c}_{pred})$ and $\boldsymbol{x}^*(\boldsymbol{c}_{true})$.

A very intuitive prediction-focused loss is regret, also known as SPO loss[6].
**Definition 2.1.2** (Regret)**.**

$$\operatorname{Regret}(c_{pred}, c_{true}) = f(x^*(c_{true}), c_{true}) - f(x^*(c_{pred}), c_{true}) \tag{2.4}$$

Regret is calculated by taking the difference between the optimal decision under perfect information $x^*(c)$ and the optimal decision under predicted information $x^*(\hat{c})$. This difference is calculated in the value of the objective function.

The reason we prefer regret over prediction-based metrics, such as mean squared error (MSE), is to ensure that the right decision is made even if the prediction is not exactly correct. In this sense, DFL techniques can be seen as a risk minimization to general machine learning approaches.

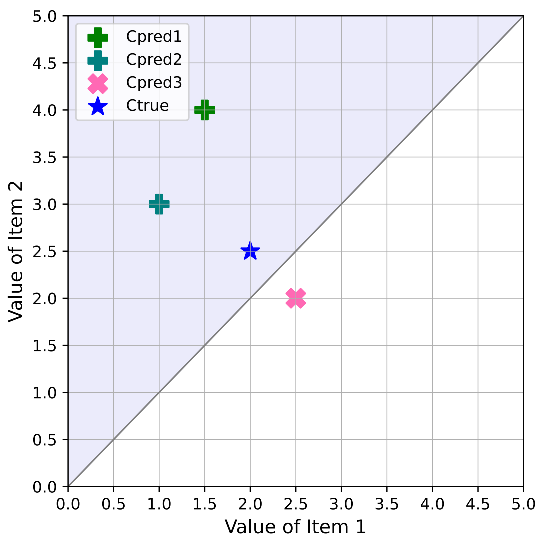


**Figure 2.1:** An example illustrating the difference between regret and prediction-based losses. The figure describes a knapsack problem where one out of two items can be chosen. It shows that something like MSE can become better while the decision made becomes worse (by moving from pred 2 to pred 3).

We can illustrate this difference more clearly with an example. In Figure 2.1, a simple knapsack problem where we maximize the value by picking one of two items. In blue, the true values are given as 2 and 2.5, respectively. Any prediction in the shaded area has a higher value for item 2 and thus leads to it being chosen. We see that predictions 1 and 2 are in this region. Regret sees both of them as already good, but a prediction-based metric wants to get closer to the true prediction. The pink marker is the best one yet according to MSE, even though it would give a worse decision. Scenarios like this show that regret gives up some accuracy in cost prediction to ensure accuracy in decisions.

## 2.2. What makes P+O hard

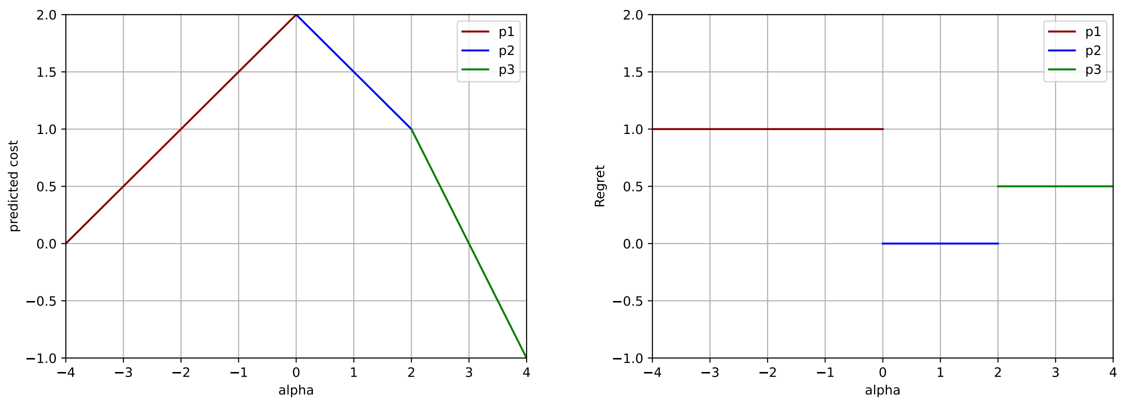
In predict+optimize, we have a prediction model with parameters $\omega$, a CO problem, and a true cost-feature dataset. The model predicts some costs $c_{pred}$ which can be paired with the true cost to calculate the regret. To calculate the gradient of the regret with respect to $\omega$, we apply the chain rule:

$$\frac{d\mathcal{L}(x^*(\hat{c}),c)}{d\omega} = \frac{d\mathcal{L}(x^*(\hat{c}),c)}{dx^*(\hat{c}))}\frac{dx^*(\hat{c})}{d\hat{c}}\frac{d\hat{c}}{d\omega} \tag{2.5}$$

From the right-hand side, the first and the last term can generally be computed. However the second term $\frac{dx^*(\hat{c})}{d\hat{c}}$ can be quite challenging to calculate directly. This term can be described as differentiating the optimization mapping $\hat{c} \to x^*(c)$. This mapping directly depends on the CO problem that we are trying to optimize. Some problems have a well-defined and differentiable optimization mapping, but most interesting optimization problems have points where gradients are undefined, as well as many areas where the gradients are simply zero. Even Linear Programming (LP) problems have zero gradients almost everywhere. Irregular gradients like this make using gradient descent directly impossible on most problems. In chapter 3, P+O methods are discussed in the context of these challenges, and how they overcome them.

## 2.3. Regret landscapes

This research compares P+O loss functions with regret loss. To do this, we want to calculate the entire regret landscape so we can find where the local and global optima are. In this section, it is



**(a)** Predicted costs **(b)** Regret

**Figure 2.2:** Predicted costs (left) and regret (right) for the shortest path example.

explained how the regret landscape is calculated, mathematical properties and assumptions are shown, as well as an example to introduce some intuition.

The regret landscape can be calculated for a single-dimensional prediction model using a dynamic programming approach introduced by Demirović et al. [4]. This method can calculate the regret landscape for any combinatorial optimization problem that can be solved using dynamic programming (DP). In short, instead of the normal numbers you would have in DP, there are linear functions representing the value for all possible model parameters. This is shown in equation 2.6, where the predicted cost for item $i$ is calculated by multiplying feature $n$ with model parameter $\alpha$ and adding feature $e$.

$$\hat{c} = v(\alpha) = n_i \cdot \alpha + e_i \tag{2.6}$$

For the DP algorithm of choice to work with these functions as values, we need to adapt the operators to piecewise linear calculus [4]. With these pieces, we can produce a dynamic programming algorithm that returns a set of intervals $\boldsymbol{I}$ with size equal to the number of different optimal solutions that span the model parameter space.

$$d_i = [\alpha_i^{start}, \alpha_i^{end}] \in \boldsymbol{I} \; forall \; i \leq n \tag{2.7}$$
$$s.t. \; \alpha_1^{start} = -\infty \tag{2.8}$$
$$\alpha_n^{end} = \infty \tag{2.9}$$
$$\alpha_i^{end} = \alpha_{i+1}^{start} \tag{2.10}$$

Each interval $d_i$ represents the optimal solution that would be found with the prediction using any $\alpha$ within it. Calculating the regret of each of these solutions and giving it as the value of the corresponding interval will produce the piecewise-constant function that fully describes regret.

To illustrate this further, an example is given. We have a shortest path problem with a start and end node. Between them are three different edges with the following features: edge one: $(n_1 : 0.5, e_1 : 2, c_1 : 2)$, edge two: $(n_2 : -0.5, e_2 : 2, c_2 : 1)$ and edge three: $(n_3 : -1, e_3 : 3, c_3 : 1.5)$. Applying our value function 2.6 gives three linear functions that describe the predicted cost. The dynamic programming algorithm finds, at each point in the model parameter space, the edge predicted as the shortest. In the top figure of Figure 2.2, the result of this can be observed. We see that in the interval $(-\infty, 0]$, edge one is found to be optimal, because the other two edges will have a higher predicted cost for those model parameters. The key is that with this, we know for the whole model space which edge will be chosen. We can drop the predicted costs and replace them with the value of the chosen decision subtracted by the optimal decision value to find the regret landscape. This is illustrated in the bottom figure of Figure 2.2. We see that the optimal decision would be predicted with an $\alpha$ between $[0, 2]$.

# 3 Related Work

Approaches in P+O can be split into two categories: gradient-based and gradient-free. The latter are not analysed directly in this work; however, they do serve as the basis for our analysis, as explained in Section 2.3.

Demirović et al. show how to fully calculate the regret landscape for a single-variable model [4]. They propose a top-down approach that leverages information from the combinatorial problem side to inform what parameter the model should have. They show that using coordinate descent, they can achieve regret scores on par with or even outperforming state-of-the-art methods like SPO+[6]. However, the method is limited to CO problems that can be solved using dynamic programming (DP).

Guler et al. extend this method by employing a divide-and-conquer approach to significantly speed up the training [9]. They also generalize the problem space to include all problems with a bilinear objective, which is much broader than only DP-solvable problems. This improvement is at the cost of only being able to map a predefined finite area of the model space, as opposed to the DP variant, which calculates it entirely.

For gradient-based methods, the challenge lies in overcoming the discontinuities that are often present due to the combinatorial nature of the problems in the space. In particular, calculating the gradient over the optimization mapping $\boldsymbol{c} \to \boldsymbol{x}^*(\boldsymbol{c})$ is the problem, because it is often zero for most points or even undefined. For example, linear programs (LPs) have gradients that are zero almost everywhere. In the rest of this section, we go over the different ways that have been found to overcome this problem.

Mandi et al. give an overview of the state of the P+O field and also compare gradient-based methods thoroughly on several problems [12]. Some of the main takeaways include:

1. SPO+ shows robust performance across all problems
2. QPTL's performance depends heavily on the quality of the relaxation, but can outperform other methods when the relaxation is good.
3. Both LTR losses and NCE can be trained really fast, this is crucial for real-world large-scale problems and thus makes them suitable. Especially since NCE showed performance that did not fall far below SPO+ on most test problems.

Continuing to follow Mandi et al. [12], we identified three distinctive classes: (a) analytical differentiation of specific problems and finding ways to change or augment different problems into those specific forms, (b) using perturbations to instead reason over averages, (c) defining surrogate loss functions based on heuristics. For each of these classes, the most notable works are discussed along with general properties that apply to them.

## 3.1. Analytical differentiation and smoothing

A natural place to start is to see which problems we can differentiate directly. For unconstrained optimization, Gould et al. show that the argmin of a smooth and convex function can be differentiated analytically [8]. They then used this method to differentiate through some constrained problems by introducing a log barrier function in place of inequality constraints, as originally proposed by Boyd and Vandenberghe [3]. To get good performance from this method, hyperparameter tuning is needed to define the weight(s) for the log barrier function.

Mandi and Guns used a similar approach to drop constraints while also exploiting several properties of LPs [11]. In particular, they used an interior point method, making it more efficient while limiting the use to LP problems.

Amos and Kotler singled out quadratic programs (QPs) as problems that can be differentiated while constrained [1]. By differentiating through its Karush-Kuhn-Tucker (KKT) conditions, the gradient of a convex QP can be found. The quadratic nature of the objective function ensures a continuous value resulting in a non-zero gradient. Wilder et al. propose adding a Euclidean norm of the decision variables to the objective function. They then make use of Amos and Kotler's *OptNet* to differentiate the optimization mapping[22][1]. They call this combination of techniques *quadratic programming task loss* (QPTL).

Something all of the mentioned methods have in common is that the problems they can solve need to be convex and continuous. Unfortunately, these properties do not hold for most CO problems. In this case, continuous relaxation can still be a good option. Ferber et al. show an alternative to this [7]. They propose using cutting planes before dropping the integrality constraints to greatly decrease the integrality gap, at the cost of computation speed.

**Summary:** If the objective function is a smooth and convex function, the optimization mapping can be differentiated by KKT conditions. However, for most CO problems the objective function is not smooth and convex. Methods in this section use some means to ensure these properties by for example, adding a euclidean norm of the decision variables to the objective function.

## 3.2. Smoothing by random perturbations

The methods in this section were built on a probabilistic view of the "predict then optimize" problem. Instead we view the optimization mapping as $c$ onto a probability distribution over feasible region $\mathcal{F}$. This can be written as the conditional distribution $p(x^*(c)|c)$.

To see why this is a good idea, we look at Domke's work [5]. They found that if you have a conditional distribution $p(\theta_1|\theta_2)$ with a loss $\mathcal{L}$ defined on $\theta_1$, the gradient of $\mathcal{L}$ with respect to $\theta_2$ can be approximated. This is significant considering that when this is applied to $p(x^*(\hat{c})|\hat{c})$ the gradient $\frac{d\mathcal{L}(x^*(\hat{c}))}{d\hat{c}}$ is calculated. With this gradient, we do not need to differentiate the optimization mapping anymore, solving the problem by avoiding it entirely.

$$\frac{dL}{d\theta_2} \approx \frac{1}{\delta}(\mathbb{E}[\theta_1|(\theta_2 + \delta\frac{d}{d\theta_1}(\mathcal{L}(\mathbb{E}[\theta_1|\theta_2]))] - \mathbb{E}[\theta_1|\theta_2]) \tag{3.1}$$

The key observation here is the perturbation of $\theta_2$. Many authors used this idea in their work.

Given that $\mathbb{E}_{x\sim p(x|c)}[x|c] = x^*(c)$ Vlastelica et al. saw that (3.1) reduces to a linear interpolation between $\hat{c}$ and $\hat{c} + \delta\frac{d\mathcal{L}(x^*(\hat{c}))}{dx*(\hat{c})}$ [21][5]. Using this, they created the differentiation of black box combinatorial solvers (DBB). The name is because it treats the solver as a black box, allowing it to be any kind of solver. This fact is true for all methods described in this section.

We can also try to calculate (3.1) directly, however for this we need the distribution $p(x|c)$. To calculate this we would need to go over every point $x \in \mathcal{F}$. Papandreou and Yuille circumvent this by instead estimating the distribution by calculating many perturbations of $\hat{c}$[17]. Their Perturb-and-MAP approach calculates $\hat{c}' = \hat{c} + \eta$ where $\eta \sim Gumbel(0, \epsilon)$. With the distribution, the expected values inside (3.1) can be calculated using Monte Carlo simulation.

Berthet et al. make use of this perturb-and-MAP framework to define their differentiable perturbed optimizers (DPO)[2]. They adapt it to use the reparameterization trick to draw samples from $p(x|c)$. It uses a temperature coefficient $\epsilon$, which gives a tradeoff between accuracy and smoothness. Berthet et al. further improve it by constructing a Fenchel-Young loss to go with their differentiation method[2].

**Summary:** By reformulating the P+O problem to one of probabilistic inference, we can formulate an equation that gives the gradient we need. However, to calculate it, we need the distribution $p(x|c)$. It was proposed to estimate this distribution by perturbing $\hat{c}$ multiple times and calculating the corresponding solutions $x$. Using this idea, the gradient can be calculated regardless of the solver used.

## 3.3. Differentiating surrogate loss functions

The methods discussed above described some way to change the model or define the gradient so that the optimization mapping can be differentiated. Works in this section define specific task losses that come with gradients or subgradients that are well defined.

Elmachtoub and Grigas find a convex surrogate upperbound of regret that they call SPO+ loss[6]. This loss function has a subgradient defined for any $\hat{c}$:

$$x^*(c) - x^*(2\hat{c} - c) \in \delta\mathcal{L}_{SPO_+}$$

With this subgradient, neural networks can be trained to predict any parameters inside a CO problem with a linear objective. Elmachtoub and Grigas also show that their surrogate is Fisher consistent [6]. This means that when minimizing SPO+ loss, the expectation of regret is also minimized.

Noise Contrastive Estimation (NCE) is a methodology that has been shown to have great performance over various machine learning tasks such as language modeling[15] and image classification[10]. The idea behind it is to have a model learn to discriminate between the true distribution and a noise distribution. Mulamba et al. adapt the concept of NCE to P+O by defining a model that learns to distinguish between the ground-truth solution $x^*(c)$ and different feasible solutions $x' \in S$. So the model will learn to predict $\hat{c}$ that perform well with the solution found by $x^*(c)$ and worse for solutions $x'$ [16]. The key to making this work is to construct a good set $S$, referred to as “solution cache.” Mulamba et al. suggest to first add all optimal solutions in the training data, then during training time, new predicted costs are added. Mulamba et al also showed that for any DFL technique the solver call can be replaced by a solution cache lookup[16].

Mandi et al. transform the DFL objective into a learning-to-rank (LTR) problem [13]. Essentially, this turns the objective into trying to predict $\hat{c}$ such that the order of solutions is similar to that given by the true cost $c$. Similarly to the NCE approach, they make use of a solution cache $S$. The loss is defined such that for each solution pair $x^*(c)$ and $x' \in S$, it is minimized if the predicted $\hat{c}$ produces the same ordering as $c$. In addition to defining the previously described pairwise learning-to-rank loss, Mandi et al. also define a pairwise difference and listwise learning-to-rank loss function[13]. Both LTR and NCE approaches have an advantage of not needing to solve the CO problem itself during training of the model.

Tang and Khalil bypass the CO optimizer bottleneck in a different way, using their Cone-aligned Vector Estimation (CaVE) [18]. CaVE uses the true cost $c$ to find the optimal decision and then creates a cone around the true cost vector where the optimal decision is taken. This cone can be made with the binding constraints at the time of solving for the true cost. During training, the predicted costs $\hat{c}$ are projected onto the conic space. The loss is defined as the angle between the optimal cone and $\hat{c}$. Minimizing this loss will ensure that $\hat{c}$ aligns with the optimal cone constructed from the ground truth solution $x^*(c)$.

**Summary:** Surrogate losses are designed to work specifically for combinatorial optimization problems and come with well-defined gradients or subgradients. These methods also treat the solver as a black box and can hence use any solver. Many surrogate losses minimize the amount of calls to the solver or even forego the need for it entirely. This makes these methods very time efficient and usable on more complex problems.

# 4 Local Structure of P+O Losses

In this chapter, we propose a new way to analyze P+O loss functions. We start by motivating our approach in section 4.1. In section 4.2, it is explained how we compare P+O loss functions to the regret landscape in an experiment. Finally, in section 4.3, the results of the experiment are analyzed and discussed.

## 4.1. Motivation

The objective of P+O is represented by regret, but as we saw before, regret is not fit for gradient descent. P+O losses circumvent this by representing a similar objective to regret while also remaining viable for gradient descent. But how much do they actually represent regret as an objective?

P+O losses can still be evaluated in regret score, however, as we show in Figure 4.1, it does not tell the full story. By instead looking at the regret landscape, we can gain more intuition behind why loss functions fail because we can see which parts of the landscape are followed.

To compare P+O losses with the regret landscape, we need to define when a loss represents the same objective as regret. An intuitive definition is when the shape is similar, when it goes up and down in the same places. But, while this describes the ideal case of approximation, it is unrealistic and takes attention away from the important places of the regret landscape. Instead, we want to know if it defines the same places as good. If the local optima of the P+O loss line up with good places in regret, then we can guarantee that gradient descent on it leads to good solutions.

It is hard to say whether the shape of CaVE or SPO+ is closer to the regret landscape. But, if we first define the left and rightmost segments as regret objectives, we can immediately tell that SPO+ follows the regret landscape a lot better, since its objective agrees with the regret objective.

This brings us to the research question for this section. *To what extent do P+O local optima align with the regret objectives?*

## 4.2. Methodology

In this section, we define how P+O losses are compared to the regret landscape, what metrics are tracked and how they are calculated, and the problems used to test the loss functions.

### 4.2.1. Locality experiment

To start, we need to define which places in the regret landscape are important. To do this, we use the notion of locality. Locality is an intuitive way to define local optimality in one-dimensional regret landscapes.

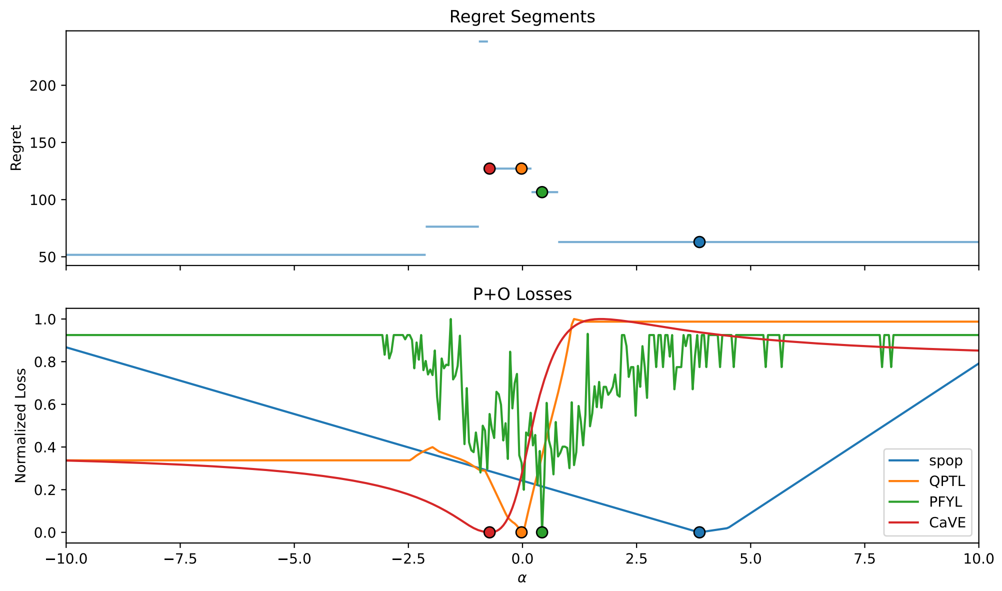


**Figure 4.1:** Regret landscape with normalized P+O loss functions. SPO+ selects a good segment far on the right, while CaVE, QPTL and PFYL select a lot worse segments near the middle. Regret score would tell you SPO+ is the best while the other 3 are a lot worse. However with the regret landscape we can see that none of the loss functions converge to the optimal regret segment and that SPO+ is actually the furthest away from it.

First, we introduce some notation: Given a problem $P$, we generate features and costs for each objective parameter. Equation 2.6 describes our prediction model where $\alpha$ is the learned parameter. For this model, we can fully calculate the regret landscape described by the intervals $d_i \in \boldsymbol{I}$ and the regret values $r_i \in \boldsymbol{R}$. With these variables, we can start describing the experiment.
**Definition 4.2.1** (Locality). Given a segment $d_i$, it can have three different localities: global, local and random. If $d_i$ has the lowest regret in the landscape, then it is a 'global' segment. We call $d_i$ 'local' if it does not have improving adjacent segments. If it does have an improving adjacent segment, it is 'random', i.e., not locally optimal. More formally, we describe these conditions as the following equations:

$$d_i \in \text{global} \iff r_i \leq r_j \; \forall \, j \tag{4.1}$$

$$d_i \in \text{local} \iff r_{i-1} > r_i < r_{i+1} \quad \& \quad d_i \notin \text{global} \tag{4.2}$$

$$else \; d_i \in \text{random} \tag{4.3}$$

The global and local segments are the local optima of the regret landscape and which we thus use as the objectives. We do not view any random segments as objectives even though they can be better regret-wise than local. This is because, intuitively, we do not want a learning procedure to land in a random segment, as it can be improved by a straightforward procedure: take a few steps to the left, then do the same to the right.

Now that we can partition the regret landscape into local, global, and random segments, we want to know to which of these segments the differentiable P+O losses converge. We considered using gradient descent but settled on doing optimistic analysis. Which means recording all stationary points of each loss function instead of one gradient descent point.

In practice, this means that we calculate the loss and gradients over the finite part of the regret landscape by varying the single machine learning parameter with regular steps. We do this within the interval $[\alpha_1^{end} - \delta, \alpha_n^{start} + \delta]$. Looking at the finite end of the outer segments and adding a factor $\delta$ to it also to account for the outer segments themselves. The loss and gradient values are calculated for every $\alpha$ in this interval in steps of 0.05.

When looking at the sequence of gradient values, we can find the P+O defined local optima by finding where the gradient is zero, or more likely where it crosses zero. For each sequential pair of gradient values, if their multiplication $\Delta_i \cdot \Delta_{i+1} \leq 0$, then somewhere in between, zero has been crossed. When this holds true for discontinuous gradients, we cannot be sure that it crossed zero, it could have jumped over it. Functionally however, it would still act as a point of convergence, so it still achieves the intended goal. We record the alpha value in the middle of the points we are comparing, as a local optimum of the loss function. For a convex loss function like SPO+, we can simplify this process and take the $\alpha$ with the lowest loss value.

The last step is to find within which regret landscape interval $d_i$ the optimal $\alpha$ of the loss lies.

$$d_{opt} = \alpha_i^{start} \geq \alpha_{opt} > \alpha_i^{end} \tag{4.4}$$

At this point, we have computed the regret landscape, which intervals are global, local and random, and the stationary points of each loss function. With this, we can record on which segment each stationary point lies and thus also the locality associated with it.

For the losses with multiple local optima, we assign the best locality found among them. This makes their results comparable to the convex loss functions in terms of number. It represents their potential, but the results do become exaggerated. In section 4.3, we explore this difference more using precision and recall analysis.
**Definition 4.2.2** (Precision and Recall)**.** Precision indicates how likely a segment to which P+O loss converges is local or global. This is calculated by taking all the local optima, then calculating the number of locally optimal segments found and dividing this by the total number of segments spanned by the P+O local optima. If we refer to global/local as 'good' segments, we can formalize it as follows:

$$\text{Precision} = \frac{\#\text{good segments found}}{\#\text{segments found}} \tag{4.5}$$

Recall is percentage of local and global segments in the regret landscape are found by P+O losses. With it, we can reason about the extent of the regret landscape that is modeled by the P+O losses. In our case, we compute it by dividing the number of local and global segments found by the total number of local and global segments in the landscape.

$$\text{Recall} = \frac{\#\text{good segments found}}{\#\text{good segments total}} \tag{4.6}$$

For random and local segments, there can be quite a lot of variance. For example, one random segment can be next to global while another can be far from anything good. In addition to this, some random/local segments can have a competitive regret, while others represent really bad solutions. To explore these differences, we have additional metrics, distance and depth.
**Definition 4.2.3** (Distance)**.**

$$\text{Distance}(\alpha_{opt}, d_{global}) = min(abs(\alpha_{opt} - \alpha_{global}^{start}), abs(\alpha_{opt} - \alpha_{global}^{end})) \tag{4.7}$$

This equation describes the distance from the optimal $\alpha$ found by a loss function to the segment which is globally optimal in the regret landscape.
**Definition 4.2.4** (Depth)**.**

$$\text{Depth}(r_{opt}, r_{global}) = \frac{r_{opt} - r_{global}}{r_{max} - r_{global}} \tag{4.8}$$

which measures the relative regret of the optimal $\alpha$ compared to the difference between the highest regret segment and the lowest. In this definition, a 1 would mean that it has the worst possible regret value, while a 0 would mean that it has the globally optimal regret value.

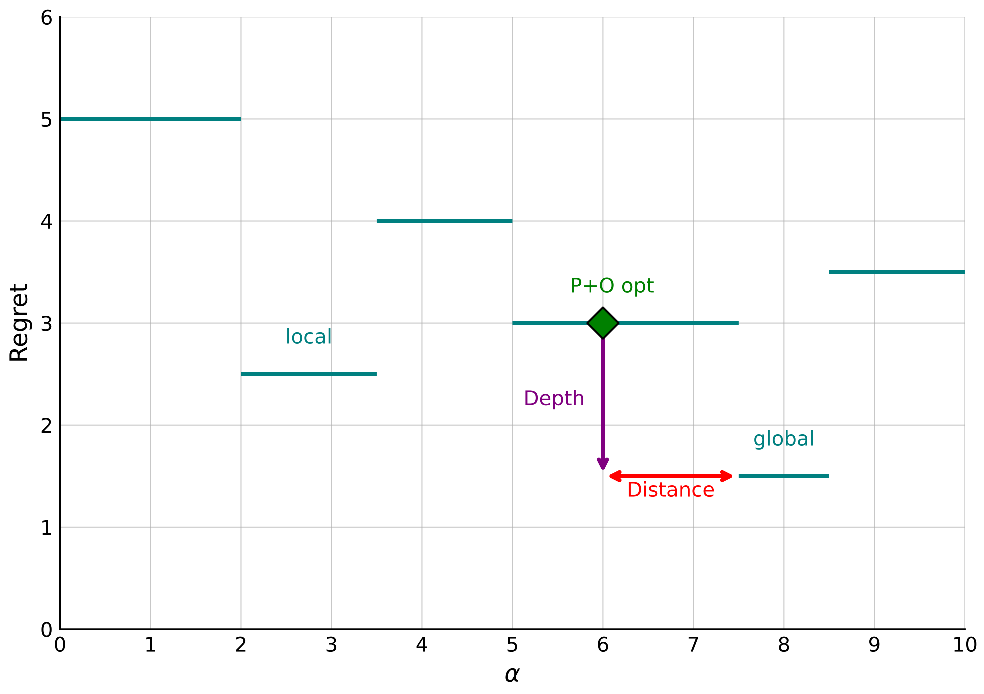


**Figure 4.2:** Regret landscape illustrating distance and depth metrics. Depth to global: difference in regret of a P+O stationary point and global. Distance to global: difference in model parameter $\alpha$ between P+O opt and global.

In our definitions, we use the (most used) example of distance/depth to global. However, these metrics can also be applied in different ways, such as distance/depth to the nearest local, for example.

Figure 4.2 shows the concepts of distance and depth in a visual example. Along with an example of a local and global segment.

### 4.2.2. Data generation

So far, we have described the experiment on a single problem instance. To properly perform it, we do it for many problem instances divided among three problem types: knapsack, shortest path (SP), and traveling salesperson problem (TSP). In this section, it is shown how the features and costs for each problem are generated. All data generation methods are adapted from Tang and Khalil's PyEPO [19].

**Knapsack** The knapsack problem is to find the subset of items that is within the capacity that has the highest value. Capacity is set to 20. For each item $i$ we generate a weight $w_i$, two features $n_i, e_i$ and a true cost $c_i$. The weight is drawn from a uniform distribution $U(3, 8)$ and the two features are drawn from a normal distribution $N(0, 1)$. We use a random transformation $B$ to calculate the true costs from the features. $B$ consists of two multipliers drawn from normal distribution $N(0, 2)$.

$$\boldsymbol{c} = (B \cdot [\boldsymbol{n}, \boldsymbol{e}])^2 \tag{4.9}$$

**Shortest Path** The shortest path formulation from PyEPO sets up a grid with edges between adjacent spaces, where the goal is to find the shortest path between the top left space and the bottom right. For each edge, two features $n_i, e_i$ are sampled from a normal distribution $N(0, 1)$ once again. The true cost for each edge is calculated like in equation 4.9.

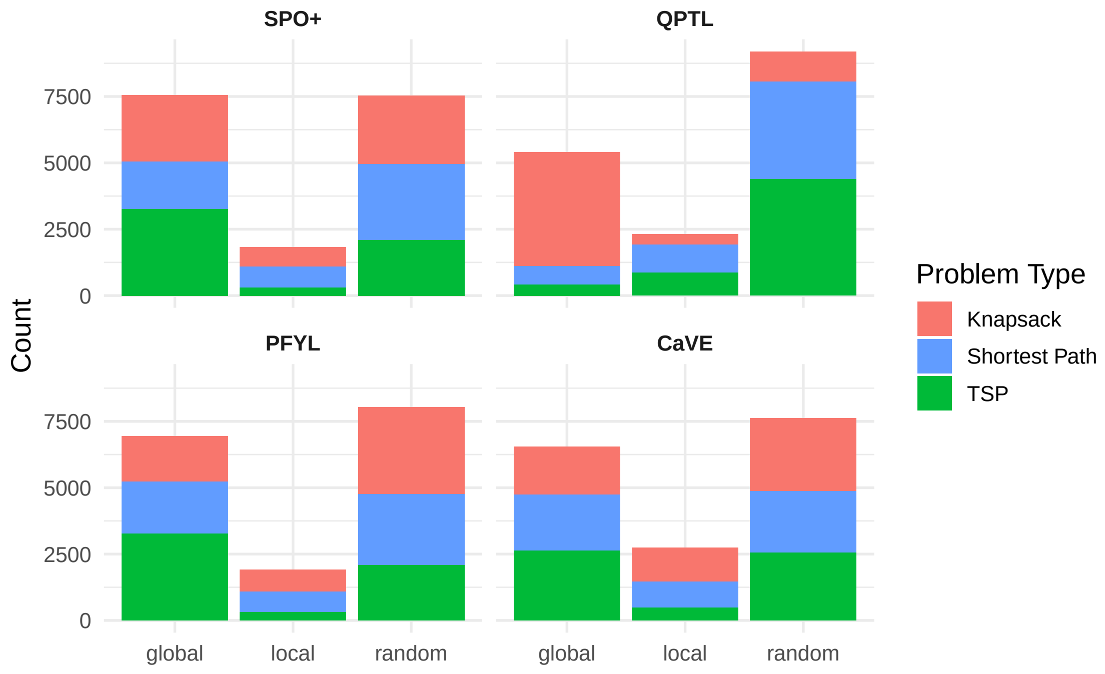


**Figure 4.3:** Distribution of loss functions having a stationary point on a globally/locally optimal or neither (random).

**Traveling salesperson Problem** The traveling salesperson problem is to find the shortest route that passes through all the nodes on a graph once. In our formulation, we use a fully connected graph. Each node is drawn with initial coordinates from $(U(-2,2), N(0,1))$. Each edge once again has two features $n_i, e_i$ drawn from a normal distribution $N(0,1)$. The true costs of each edge is the sum of the Euclidean distance and the transformed features, like in equation 4.9.

### 4.2.3. P+O losses

We perform the described experiment on a group of P+O losses. To represent every type of P+O loss, we use the following methods:

- **QPTL** (quadratic programming task loss) as an analytical smoothing technique [22].
- **PFYL** (perturbed Fenchel-Young loss): as a smoothing by perturbation method [2].
- **SPO+** (smart predict then optimize+): a surrogate loss [6].
- **CaVE** (Cone-aligned vector estimation): another surrogate loss [18].

## 4.3. Results

In this section, the results of comparing P+O losses to the regret landscape are shown. We first look at the general distribution of locality. Then we take a closer look at local and random instances. Next, we discuss what happens when we take all the stationary points of the non-convex losses into account, after which we look at the impact of problem size on locality. Lastly, we talk about the potential of combining loss functions.

### 4.3.1. Locality

To start the results section, we look at what types of localities are found by inspecting the distribution of locality in Figure 4.3. In it, we observe at what frequencies each P+O loss function aligns with global, local, and random segments.

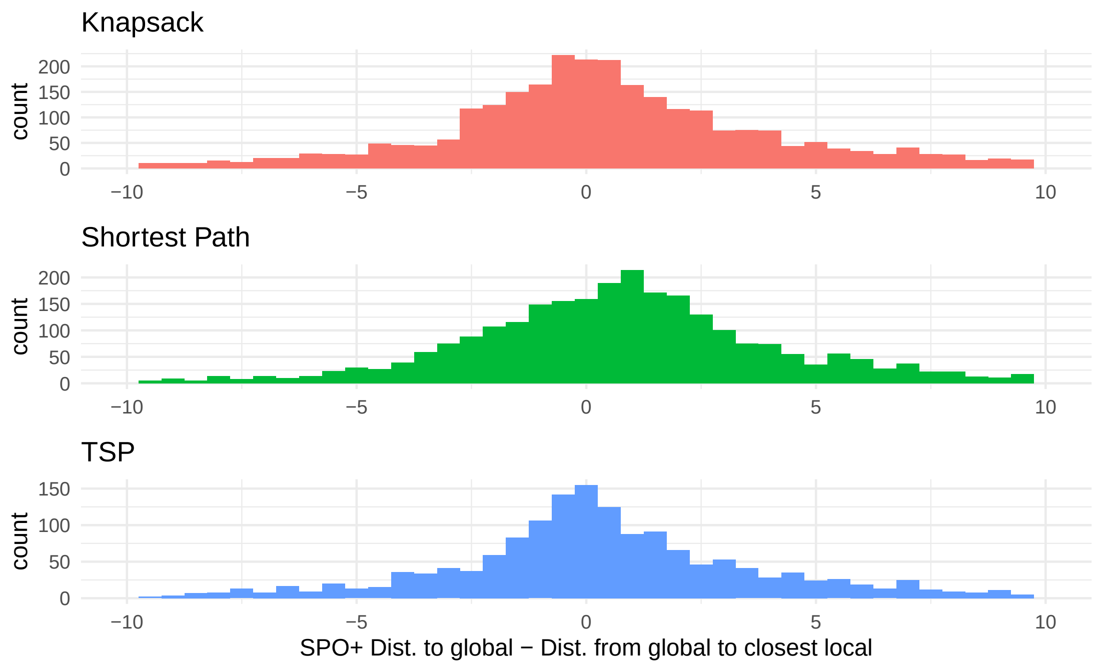


**Figure 4.4:** SPO+ distribution of the difference between the distance to the global optimum and the distance to the closest non-global local optimum.

First, we observe that all the distributions are very similar. Although we see some differences, like QPTL doing better on knapsack problems and worse on others, all the distributions are shaped very similarly. They are all dominated by a large amount of random segments, with a decent amount of global and some local. Even though these methods all work differently, they produce very similar locality.

Two things are interesting here. First, *every loss function consistently finds a lot of random segments*, i.e., not aligned with any regret local optima. This is surprising and shows that there is still considerable room for improvement in the accuracy of these losses. Secondly, when these losses align with local optima, it is more often a global one.

With these facts from Figure 4.3, we hypothesize that P+O losses do not consider the local regret landscape closely. If they would close approximate regret, then Instead, it looks like we have an often faulty approximation of only the optimal regret area. In the following sections, we explore this hypothesis more closely.

### 4.3.2. Random Locality

In this section, we look at the random points found by P+O losses. We hypothesize that, even though these points are random, they are often very close to a regret local optimum. If this is true, the losses in question follow the local regret better than indicated by the locality distributions. However, if it is false, it supports our theory that P+O losses do not follow local regret closely.

To find this out, we look at distances from the global and local optima. Specifically, we want to know whether these points are close to local or global optima, or if the random locality is appropriate, meaning they do not approach either.

In Figure 4.4, we see the difference between the distance to the global optimum and the distance to the closest non-global local optimum is plotted three times for each of the different problems. Less than zero means closer to the global optimum, while above zero means closer to a local

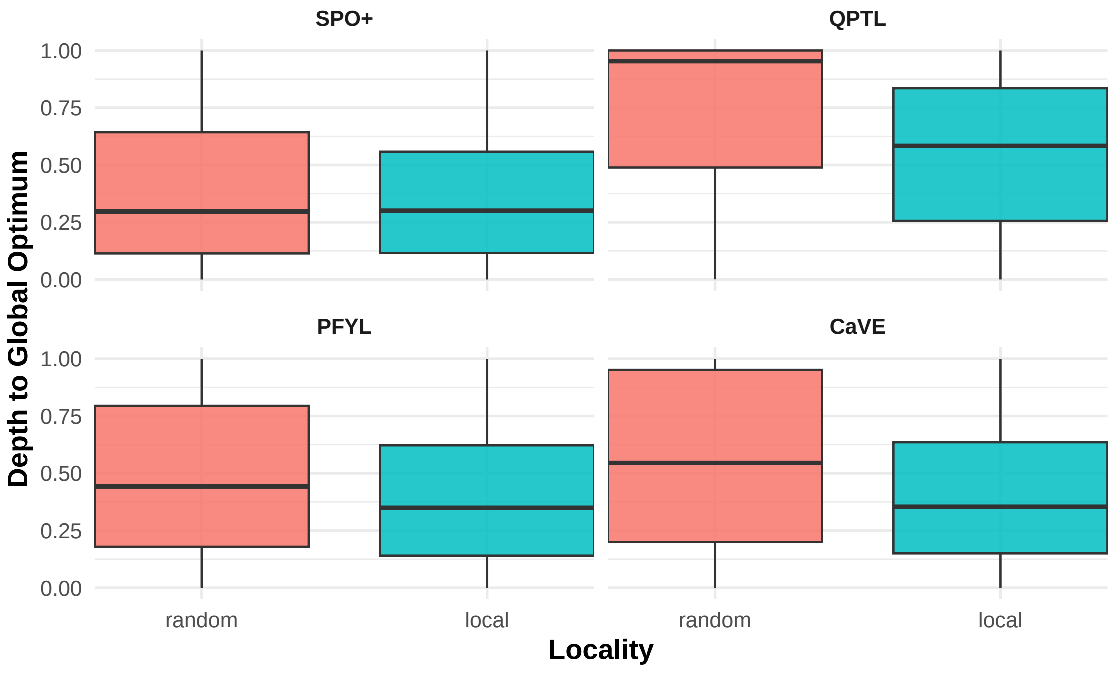


**Figure 4.5:** Depth of random and local points from SPO+. Depth is a measure of relative regret compared to the best and worst regret in the landscape.

optimum.

Following our hypothesis, we would expect two high-density areas. One below zero and one above zero, since this would account for the points that are approaching both global and local points. However, we see something very different. For every problem, it centers around zero. So our hypothesis was wrong.

For the knapsack and shortest path problem, it is evenly distributed, showing that on average, SPO+ leads its models between local and global optima. This leads us to the conclusion that when it aligns with random locality, most of the time it does not 'miss' the local or global segment, but instead *something more fundamental is misaligned in the approximation of regret*. For the other losses, we see similar patterns and draw the same conclusions. Their results can be found in Appendix A.

### 4.3.3. Local vs Random

We have found that the random segments are often aligned with by P+O loss functions. Even though these points do not exactly follow the regret landscape's local structure, it could be that the regret value is still good. In this section, we hypothesize that the quality of solutions of random is worse than local. If true, we show that converging to a local solution would generally give better results.

In Figure 4.5 we see that there is a significant difference between the depth (relative regret) of random and local points found. What this shows is that, even though random points can be close to the global optimum, they still give a lot worse solutions on average, compared to local ones, confirming our hypothesis.

With the low number of times a local optima are aligned with, it is reasonable to think that this is a product of randomness. But looking at Figure 4.6, we see that there is a clear separation in distance to the global optimum between random points and local points. P+O losses align with

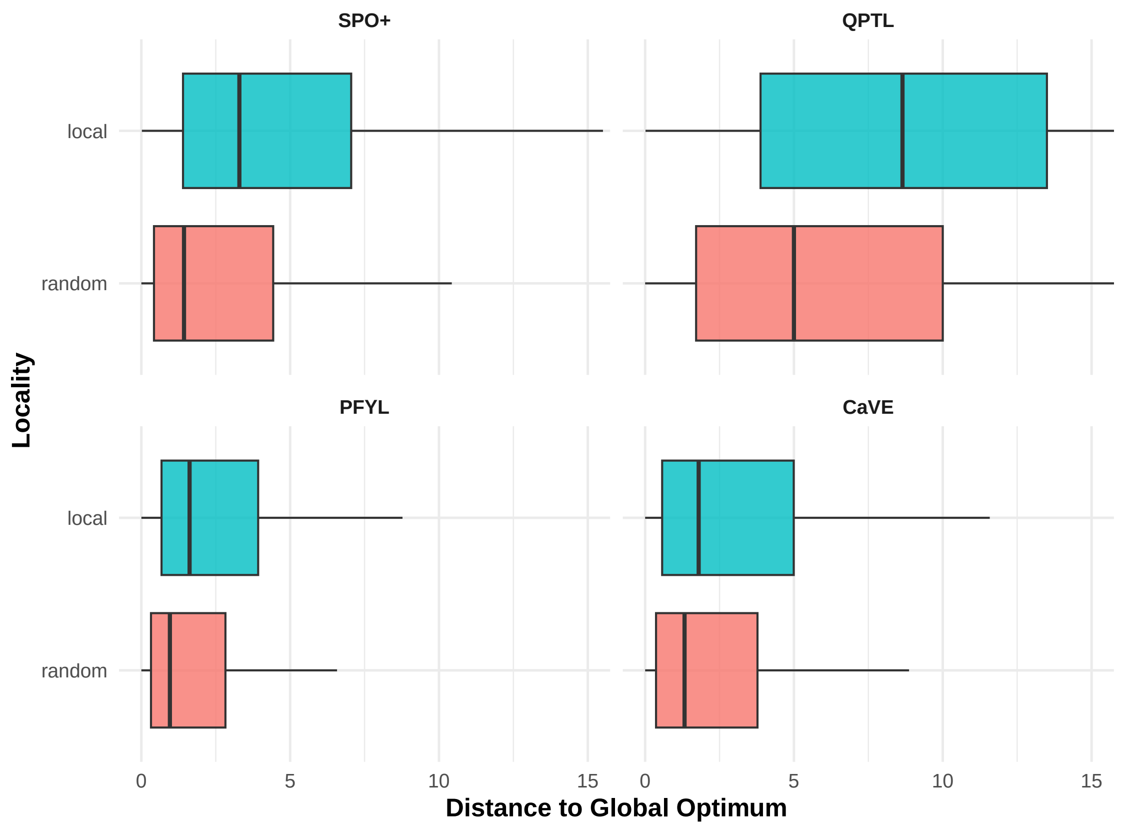


**Figure 4.6:** Distribution of distance to the global optimum of stationary points found by P+O losses, separated by the locality of random and local.

local more often when the distance increases. An explanation for this could be as follows: when the loss is calculated sufficiently far from the global optimum, the pull of the global optimum on the loss is low, now it can approximate the local structure better and align with a local optimum.

Figure 4.7 shows that for random samples, the distance to the global optimum can be a good predictor of solution quality. Being close often means a lot better regret. So even though local solutions are found more often when the distance from global increases, the general solution quality gets worse.

When comparing the boxplots of the close random points with the local boxplots from Figure 4.5, they are very similar. Random points close to global get similar regret to local segments. This suggests that approaching the global optimum is also a good strategy. But we also see the other 50% that is far from global, which drops significantly in quality. Because P+O loss functions are unaware of the regret landscape, they cannot consistently get close, especially when the true optimum does not align with the regret optimum.

Most importantly though, we showed that local regret is on average better than regret of random segments. This shows that there is a good incentive to try to improve these methods to find locally optimal segments more often. Ideally, the search is close to the global optimum since it has showed to be a good predictor for lower regret.

### 4.3.4. Multiple local optima

Both PFYL and CaVE produce many stationary points. So far in the results, we have simply used the best solution found. Here, we hypothesize that PFYL and CaVE model the local regret

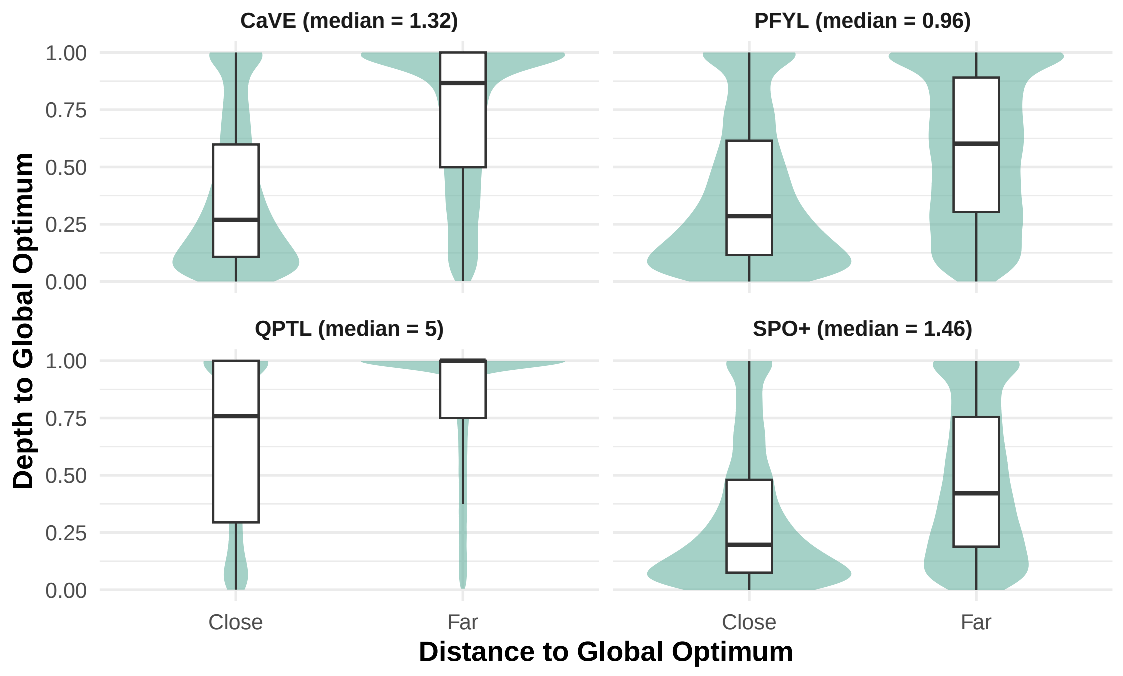


**Figure 4.7:** Depth of random points found by P+O losses, both close to the global optima and far from it. Close if below the median distance to the global optima, far if above it.

| Optima found | PFYL | Mean precision | Mean recall | CaVE | Mean precision | Mean recall |
|---|---|---|---|---|---|---|
| 0 | 3279 | 0% | 0% | 3258 | 0% | 0% |
| 1 | 3329 | 78.1% | 38.7% | 2934 | 67.7% | 40.3% |
| 2 | 109 | 70.4% | 47.1% | 480 | 72.7% | 62.8% |
| Total | | 39.9% | 20.0% | | 35.3% | 22.6% |

**Table 4.1:** Precision and recall analysis for non-convex loss functions PFYL and CaVE, grouped by the number of optima found. Finding more than 2 local optima did not happen for more than 50 cases, so it was left out of the table.

landscape more closely because they can model multiple local optima. We also look at how realistic our previously made assumption is to always find the best local optima.

In Table 4.1, several statistics are shown about PFYL and CaVE. First, when an optimum is found, it is almost always the only one that is found. Finding multiple local optima is very rare, showing once again that these loss functions do not follow the local regret landscape closely, but more globally.

For this reason, we also see that recall is very low, only about 20% on average. Meaning that PFYL and CaVE only align with 20% of local+global segments that are in the regret landscape. This also shows that our hypothesis does not hold, or at least the stationary points beyond the best one rarely match with other regret local optima.

Regarding precision, we see that PFYL especially does well, with a precision of 78.1% on problems where it finds an optimum. The local and global counts of PFYL in Figure 4.3 are 78% likely, so our assumption of using the best result holds decently well. The total precision values for PFYL and CaVE remain very low at 39.9% and 35.3%, respectively. If you were to take a random one of the used problems and run PFYL on it, you would have about this chance to train a model into a regret local optimum. This puts it well below SPO+, for example, which has a mean precision of 48.3%.

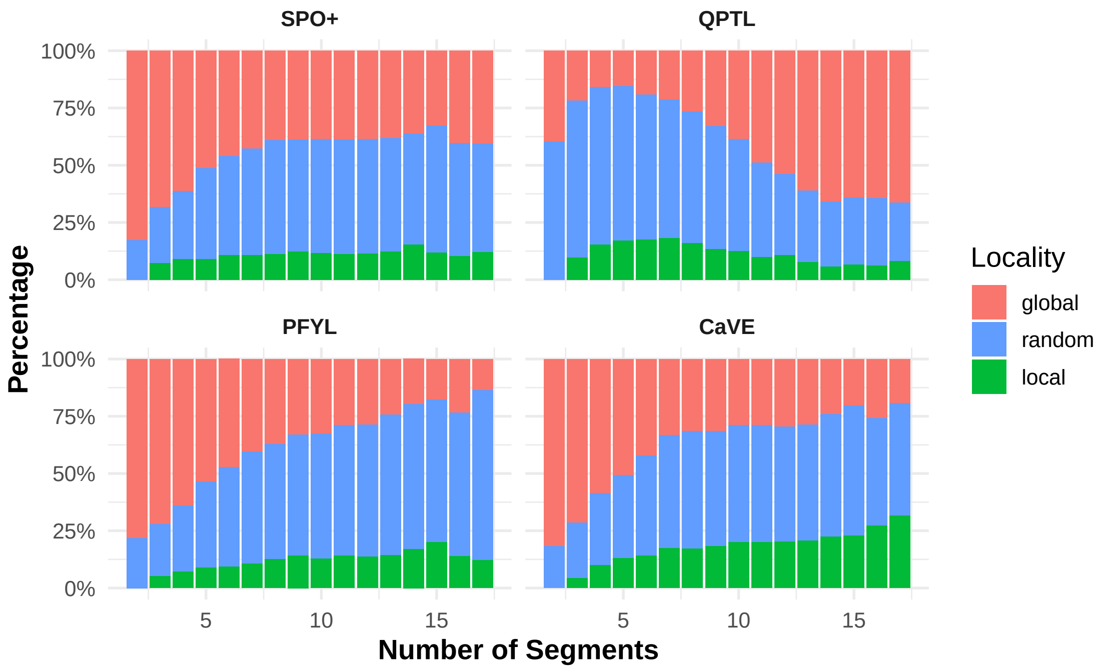


**Figure 4.8:** Locality distribution as the number of segments in the regret landscape goes up.

### 4.3.5. Impact of problem size on locality

Intuitively, it would make sense that finding good solutions is easier for simple optimization problems and harder for complicated ones. But since we previously found that P+O losses follow regret very loosely, it could be that locality is invariant of the complexity of the regret landscape.

In this section, we hypothesize that when problems become more complex, the P+O losses' locality becomes worse. To find this, we will connect the size of the problem to the complexity of the regret landscape and see how the locality of the solutions varies when the complexity of the landscape increases.

For each of the tested problem types, we varied one of the problem-defining parameters. For knapsack, we varied the number of items from 10-50, for shortest path, the grid size was varied from 3x3 to 8x8, and for TSP, the number of nodes varied from 5-15. We made two observations from this:

1. For knapsack, the locality distribution remained consistent when the problem parameter increased.
2. For both SP and TSP, as the problem parameter increased, the global locality share went down while random locality went up.

To explain this difference between knapsack and the other problems, we introduce complexity of the regret landscape as the number of segments in the landscape. This represents the number of different possible optimal solutions depending on the objective values. A complex problem will naturally have more of these.

We find that for SP and TSP we see big increases in the number of segments, from 4-12 and 5-13, while for knapsack it changes very little, from 10-15. This could be an explanation for why the performance of our loss functions remains consistent across our tested knapsack instances and becomes worse for SP and TSP.

When we plot the distribution of locality over the number of regret landscape segments in Figure 4.8, we see the expected pattern emerge. For smaller problems, global locality is common, but as complexity increases, it gets replaced by random.

Local fluctuates a little bit, but remains a very small part of the total picture. CaVE seems to find the most local segments; the contribution makes a small rise as global goes down. This is a promising pattern, when the global segment cannot be found, settle for a local one.

QPTL looks like a bit of an anomaly, but it can be explained very easily. QPTL only performs well on knapsack problems. Due to the relaxation of the knapsack problem being good, there is a spike in global locality around 10-15, where knapsack problems lie mostly.

Generally though, we see that as the regret landscape complexity increases, the global locality goes down significantly, making it clear that our hypothesis holds. This shows that, especially on harder problems, there is a lot of room for improvement still. Ideally, when problems become too hard to align with global, P+O methods would align with local instead, like CaVE already does a little bit. However, this requires these losses to be aware of the local regret landscape.

Knowing that our hypothesis for this section holds, we can also explain our results in a different light, namely, based on the limitations. In our experiments, we used a single-parameter model. This model is unable to approximate the entire solution space. For smaller problems, this effect is likely limited. But for the larger problems, it means that the best, i.e. zero-regret, solution can often not be found. P+O losses approximate regret precisely by trying to find this zero-regret solution; when it cannot be found due to model limitations, it will converge to the place closest to that solution, which turns out to be random quite often.

### 4.3.6. Combined locality

A potential way to improve the locality of P+O loss functions is to combine them. As shown in Figure 4.9, a significantly better locality is found when taking the best out of the four tested loss functions. When comparing this with their individual performances in Figure 4.3, it can be observed that a large part of the random points have shifted to the global and local locality, with global now having by far the most points.

Combining losses also has potential in terms of computation speed since the most computationally expensive step, running a solver, is only needed once for each step. However, combining these methods in an optimization setting that trains on a large dataset is not as easy as just choosing the best one for each problem. When to use which loss function and how to combine multiple loss functions to produce a single prediction model are problems without clear answers.

## 4.4. Summary

To conclude, we have seen that all of the tested P+O methods perform quite poorly on our experiment. All of them do not align with any local or global optima around 50% of the time. In the rest of the results, we looked at why this happens and if it is even a problem at all.

First, we saw that random localities are often found in the middle between local and global segments, showing that these results do not seem to be pulled closer towards either global or local segments but instead align with something else.

In a similar trend, we also saw that when SPO+ aligns further away from global localities, it finds local segments at an increased rate. This is an indicator that when the pull of global on the loss gets weaker, it can more easily align with local segments.*-+

We found that local segments have better depth than random ones, showing that, generally, it is better to align with local segments when the global optimum cannot be found.

Problem size is very relevant for locality; we have seen that on small problems, P+O losses find global segments fairly consistently. As the regret landscape grows larger, however, locality of P+O losses quickly disappears.

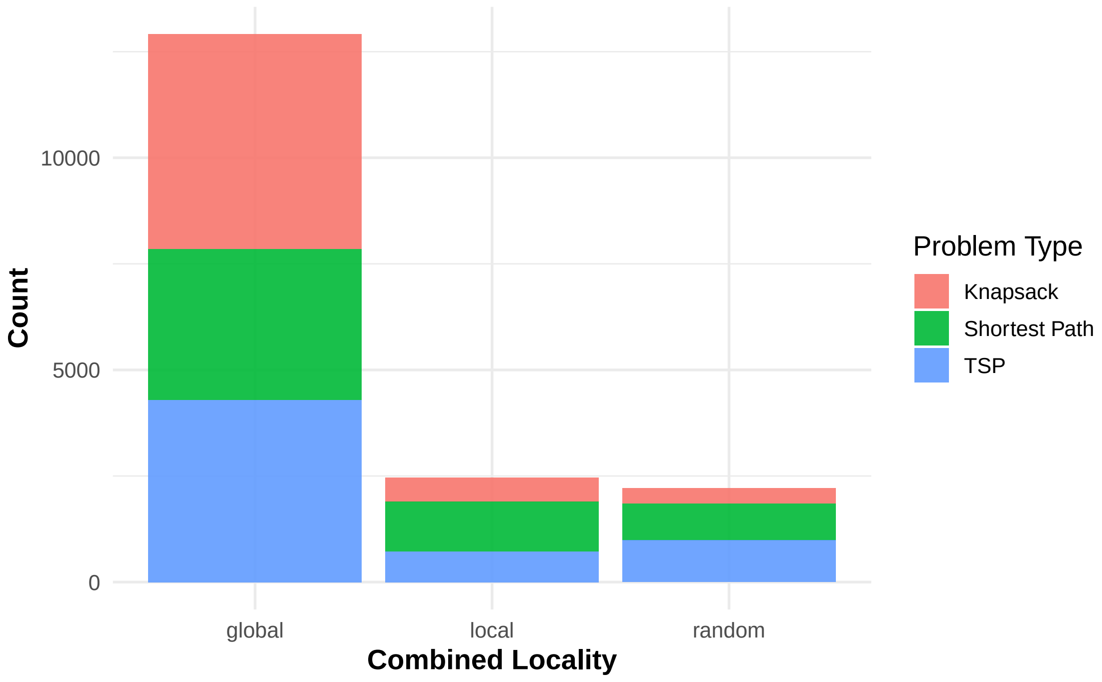


**Figure 4.9:** Locality distribution by problem type for the maximum locality among loss functions SPO+, QPTL, PFYL, and CaVE.

Putting all of this together, we can answer the research question for this section: *At best, P+O losses consider locality globally, but in many cases, not at all*. If loss functions could consider locality more, it could move the large amount of random points into local segments and make a significant improvement.

We have already seen a potential way to make this improvement, namely by combining the strengths of several of these losses. This seemingly leads to very consistently finding a local or mostly global segment. This shows potential, but would still need a lot of research to come up with a way to combine multiple losses effectively. In chapter 5 we propose a different method to make this improvement, by adjusting gradient descent to be more aware of the solution side and thus not moving from a local segment to a random one if this would make it worse.

# 5 Decision Guided Learning

In chapter 4 we found that the current P+O loss functions follow regret local optima to a limited degree. We have established that even on simple problems, commonly used P+O methods struggle to find locally optimal solutions. It was shown that many solutions with the locality 'random' were consistently selected, meaning that there is an adjacent improving solution. If we can search for and select an improving adjacent solution, then we can further improve current methods.

Searching the local regret landscape is not a simple task. When machine learning models become larger, the model search space becomes hard to explore. Instead, we approach the search from the decision side. Assuming an LP optimization problem, we can limit the number of 'smart' decisions to be made. We can explore them and find improving decisions. Then, we adjust the learning model such that it produces this decision.

In this chapter, we propose a different P+O framework to better take into account locality. A first version is implemented and tested.

## 5.1. Preliminaries

In this section, we introduce the key concepts to formulate our method. We first explain how model space, decision space, and objective space interact in polyhedral geometry. Then we explain the basic concepts of the simplex method.

These two sections will form the basis for each of the two phases that define DGL, the Adjacent Decision Finder and the Model-Decision Aligner. The adjacent decision finder uses simplex pivots to find adjacent vertices on the feasible polytope, while the model-decision aligner uses normal cones to form a loss function that has a gradient towards a certain decision.

### 5.1.1. Polyhedral Geometry in P+O

In this section, we discuss the basic concepts of polyhedral geometry and their relationship to the P+O problem.

**Polyhedra and vertices.**
**Definition 5.1.1** (Polyhedron and faces)**.** The feasible region of a linear program is a *polyhedron*: the intersection of finitely many half-spaces defined by linear inequalities. Formally, a polyhedron $P \subseteq \mathbb{R}^n$ can be written as

$$P = \{x \in \mathbb{R}^n \mid Ax \leq b\}, \tag{5.1}$$

for some matrix $A \in \mathbb{R}^{m \times n}$ and vector $b \in \mathbb{R}^m$.

A *face* of $P$ is obtained by turning some of these inequalities into equalities; faces of dimension $0$ are called *vertices*, dimension $1$ faces are *edges*, and so on [23].

**Theorem 5.1.1** (Fundamental theorem of linear programming)**.** Whenever $P$ is bounded, an optimal solution of a linear program

$$\max\{c^\top x \mid x \in P\} \tag{5.2}$$

is attained at a vertex of $P$ [20].

**Example 5.1.1** (Finding a vertex in a polygon)**.** The constraint system in Equation 5.3 defines a polyhedron in $\mathbb{R}^2$, shown in Figure 5.1.

$$\begin{aligned} \text{maximize} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1 \geq 0,\ x_2 \geq 0, \\ & 3x_1 + 2x_2 \leq 8, \\ & x_1 \leq 2, \\ & x_2 \leq 2. \end{aligned} \tag{5.3}$$

Each inequality corresponds to one of the bounding lines; for example, $x_2 \geq 0$ defines the horizontal axis as a boundary. The feasible region is the polygon enclosed by all such constraints. The corner points of this polygon are the vertices. One of the vertices of the feasible polygon is at $(\frac{4}{3}, 2)$. To see why, note that a vertex is obtained when a set of constraints are simultaneously active, i.e., satisfied as equalities. At $(2, 0)$ the following inequalities are tight:

$$3x_1 + 2x_2 \leq 8 \quad \Rightarrow \quad 3x_1 + 2x_2 = 8,$$

$$x_2 \leq 2 \quad \Rightarrow \quad x_2 = 2.$$

The intersection of these two boundary lines is the point $(2, 0)$. This point lies inside all the other constraints, so it is a feasible solution. Because it is the intersection of two independent boundary constraints in $\mathbb{R}^2$, the feasible set has been "shrunk down" to a zero-dimensional object: a single point. Hence, $(\frac{4}{3}, 2)$ is a vertex of the polyhedron.

**Definition 5.1.2** (Normal cone)**.** For a given vertex $v \in P$, not all objective vectors $c$ select it as optimal. The set of all such $c$ forms a convex cone, called the *normal cone* of $v$. This cone is generated by the normals of the active constraints at $v$ [23]. More formally, if $a_1, \ldots, a_k$ are the normals of the active constraints at $v$, the cone they generate is

$$N_P(v) = \Big\{ \sum_{i=1}^{k} \lambda_i a_i \ \Big|\ \lambda_i \geq 0 \Big\}. \tag{5.4}$$

Geometrically, the normal cone partitions the objective space. Each cone corresponds to the set of directions in which the vertex remains optimal: if $c \in N_P(v)$, then $v$ is an optimal solution of the linear program

**Example 5.1.2** (Normal cone)**.** In Figure 5.1, each vertex has a normal cone drawn from it. We can see that the vertex $(\frac{4}{3}, 2)$ contains the true objective and is thus the optimal solution for the problem defined by that objective.

The cone is defined by the two active constraints in that vertex, i.e., the constraints that can be replaced with equalities. In this case: $3x_1 + 2x_2 \leq 8$ becomes $(3, 2)$ and $x_2 \leq 2$ becomes $(0, 1)$.

Hence the normal cone at $(\frac{4}{3}, 2)$ is generated by the vectors $(3, 2)$ and $(0, 1)$. To see that the true objective $c = (1, 1)$ lies in this cone, observe that it can be written as a conic combination:

$$(1, 1) = \tfrac{1}{3}(3, 2) + \tfrac{1}{3}(0, 1).$$

Since $c$ lies inside the cone, the vertex $(\frac{4}{3}, 2)$ is indeed optimal for this objective.

In P+O, the objective is unknown; we define it as such:

$$\text{maximize} \quad c_1x_1 + c_2x_2$$

Figure 5.2 shows the objective space of Equation 5.3 when the objective is unknown. The cones from Figure 5.1 are the same ones, they can additionally be identified by the vertex labels in Figure 5.2. The whole space is divided into the normal cones from the decision space. Depending on the prediction of $c_1$ and $c_2$, a different cone, thus vertex, is defined as optimal.

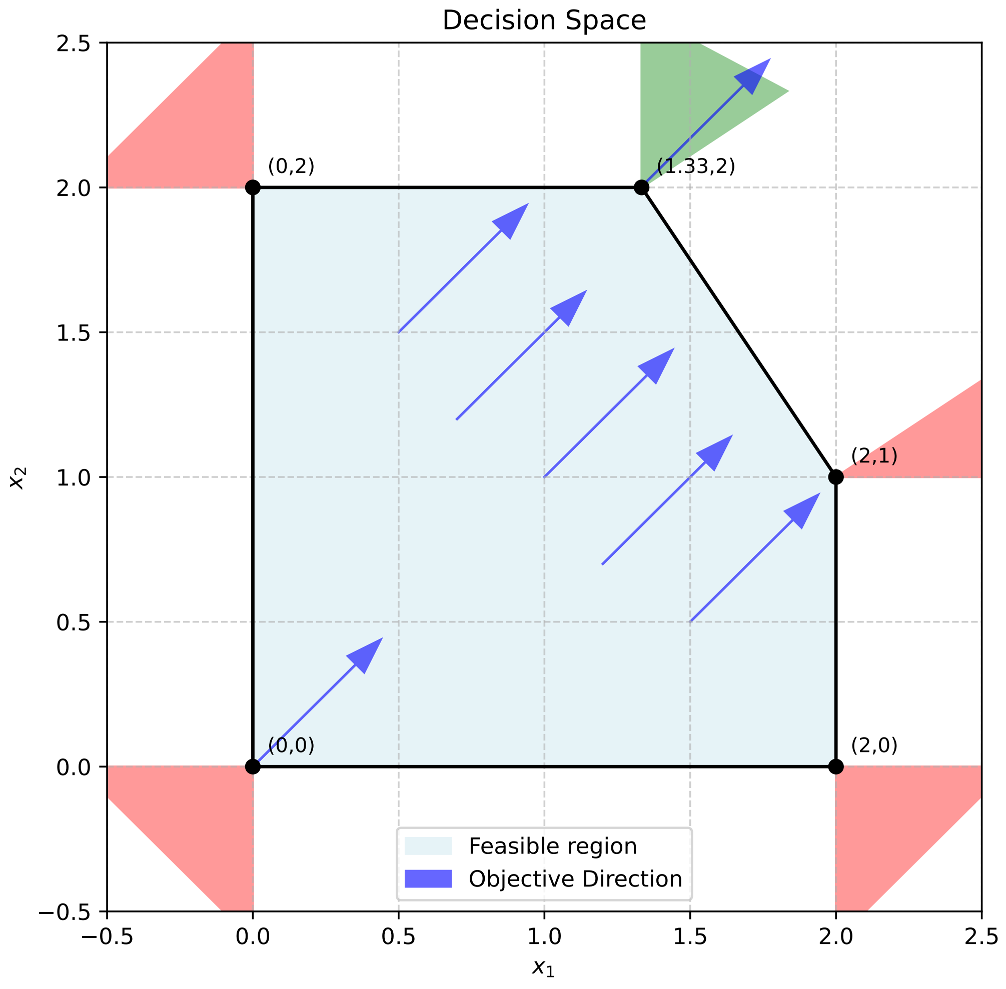


**Figure 5.1:** Decision space of a constraint problem with normal cones.

A P+O model predicts coefficients $c_1$ and $c_2$, We represent the output of our model in the objective space. Given certain features, the prediction of the model could look like the purple line in Figure 5.2. We see that, depending on the parameter $\alpha$, the model could end up in many different cones, thus giving many different decisions.

Note that any line could depict a model, which means that the model cannot always reach all cones, including the optimal cone.

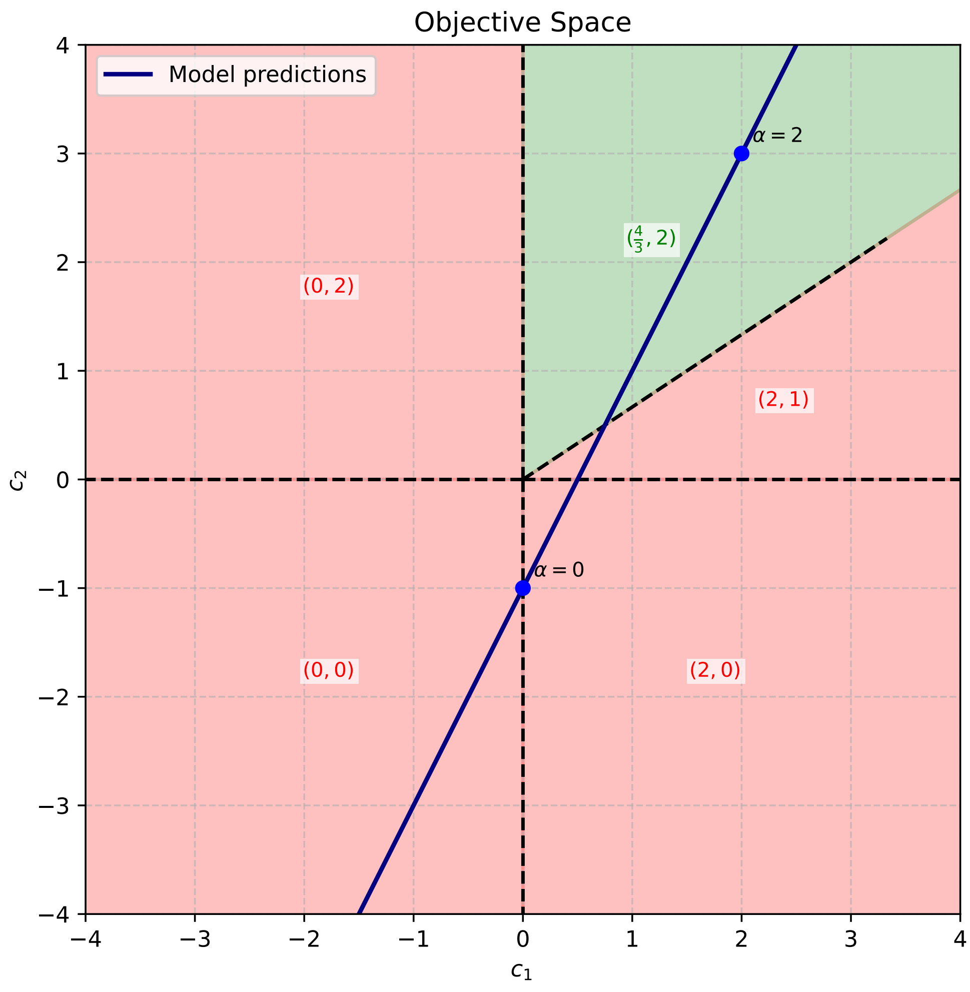


**Figure 5.2:** Objective space of a constraint problem with normal cones.

### 5.1.2. Simplex Algorithm

The **simplex method** is an algorithm for solving linear programming (LP) problems. It works by moving from vertex to vertex, exploiting the convexity to make repeated improvements until the optimum is reached.

- A **Basic Feasible Solution (BFS)** corresponds to a vertex. It is obtained by setting a set of variables (the *non-basic variables*) to zero and solving for the remaining variables (the *basic variables*). Each BFS is a candidate solution to the optimization problem.
- A **pivot** is the operation that moves the algorithm from one BFS to an adjacent one. To do this one non-basic variable that can improve the objective is selected to enter the basis and one basic variable is selected to leave. This changes the basis and shifts the solution to another vertex of the feasible region.

By repeating pivots, the simplex method systematically improves the objective value until it reaches the optimal BFS, defined by the fact that there are no more improving pivots.

## 5.2. Methodology

Decision Guided Learning (DGL) approaches the learning problem starting from the decision side. In this section, we first explain the general DGL procedure defined by its two phases. After which, we look at each phase in detail and discuss the implementation.

### 5.2.1. DGL procedure

DGL is a regret local search algorithm. It does this by alternating between two phases: finding adjacent improving decisions and aligning the model with those decisions. In Algorithm 1, we recognize these two phases by the function names in lines 4 and 9. The first takes the current decisions made and finds adjacent candidate ones that improve the true objective. Then, in the second phase, the model is aligned with the candidate decisions, and if regret is improved, the new model is accepted. This is repeated until all the problems have optimal decisions, or if none of the candidate decisions improve regret.

**Algorithm 1** Iterative Model-Decision Alignment

```
 1: Initialize ML model with parameters $\omega$
 2: Compute initial set of current decisions $\boldsymbol{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\}$ for all samples
 3: while true do
 4:     $\boldsymbol{X'} \leftarrow$ AdjacentDecisionFinder($\boldsymbol{X}$)
 5:     if $\boldsymbol{X'} = \emptyset$ then
 6:         stop (optimum reached)
 7:     end if
 8:     for all $\boldsymbol{x'} \in \boldsymbol{X'}$ do
 9:         $\omega' \leftarrow$ ModelDecisionAligner($\omega, \boldsymbol{x'}$)
10:         if Regret($\omega'$) $\leq$ Regret($\omega$) then
11:             replace corresponding $\boldsymbol{x}$ with $\boldsymbol{x'}$ in $\boldsymbol{X}$
12:             $\omega \leftarrow \omega'$
13:         end if
14:     end for
15:     if No improvement found then
16:         stop (Local optimum)
17:     end if
18: end while
```

This procedure enumerates all the adjacent decisions that are improving on the single problem level. This does not yet guarantee that it is also improving on the entire dataset level; hence, a post-alignment regret check is needed. In addition, if the alignment fails, the regret check will also ensure the failure is not accepted.

If the first phase can find all the adjacent decisions and the second phase is able to align with all of them, the DGL procedure gives the guarantees of a local search and converges to a local optimum.

### 5.2.2. Adjacent Decision Finder

This component takes the current decision to output an adjacent and improved decision. The challenges are in the adjacent and improving conditions. Adjacent means that in the regret landscape, the new decision's regret region is geometrically adjacent to the current regret region. To meet these requirements, we use the simplex algorithm to find pivots as new decisions. The pivots are always adjacent, because only one variable is swapped, and improving by definition.

Simplex requires the problems to be in LP form, making a relaxation of the problem necessary in most cases. However, past research from Mandi et al. showed that only solving the relaxation is often sufficient [14]. They found that SPO+ does not fall far behind in solving only the relaxations compared to the full problems. Thus, we expect the negative effect on the performance to be small.

**Implementation details:** To initialize simplex, we need to get the BFS of the current decision. To find this, we take the predicted costs of the current model and set them as the objective for the problem. Optimizing this problem with simplex will give the BFS for the decision that the current model would produce. With the current BFS, we can initialize the problem with the true costs. For bigger problems, there can be many improving pivots, enumerating all of them can be too time-consuming. Instead, we use $k$ as a hyperparameter that determines how many pivots are returned. We also do not sort the pivots in any way; $k$ random, improving pivots are returned.

### 5.2.3. Model-Decision Aligner

In section 5.1, we showed that a decision can be translated to a cone in objective space. CaVE trains the model to move into the cone of the true objective [18]. We use the same method, but expand it to move into any cone instead of just the optimal one. The model can be moved towards a cone by minimizing the angle between the predicted objective and the target cone [18]. First, $c_{pred}$ is projected onto the cone. We calculate the cosine similarity between the projection and $c_{pred}$ and use it as a loss.

When we train on many dataset samples at the same time, each one functions as its own problem and has a separate cone. The pivot is found for one problem while all the others remain the same. They are combined by minimizing the average angle of each problem's objective and cone.

## 5.3. Experiments and Results

To test our implementation of DGL we perform two experiments. First, we test the locality of DGL following the same steps as 4.2. Second, we test the viability on larger datasets and machine learning models to find the limits of DGL. In the experiments, we first let CaVE optimize a machine learning model, which DGL uses as a starting point. We then measure the improvement of DGL over the original model found by CaVE.

### 5.3.1. Experimental setup

Since DGL is using a large part of CaVE's methodology, we compare it with CaVE to see how much the regret local search component can add to just training towards the optimal solution with this method.

DGL is tested by letting CaVE optimize a model and then seeing how much further DGL can improve it. First, we perform gradient descent with CaVE and save the final model achieved with this. Then this model is used as a starting point for DGL to do local search on.

**Locality** For the locality experiment, we follow the same methodology as in section 4.2. We do a smaller experiment only on knapsack problems. The data is generated like in subsection 4.2.2, varying the number of items from 10 to 40 in steps of 5.

To show results, measures of locality and depth are used as defined in section 4.2.

Since we work with single problems, we are able to fully explore all the adjacent decisions without it costing too much computation time. Therefore for the locality experiment we set the maximum number of pivots to consider $k$ to $\infty$.

**Regret** The locality experiments can only cover limited ground due to the restrictions of single-parameter models. That is why we also perform regret measurements, done over varying model sizes and dataset sizes.

We perform this experiment on two different problems: knapsack according to PyEPO [19] and capacitated vehicle routing problem (CVRP) like in [18]. We use the following problem parameters:

Knapsack with both 20 items and a capacity of 20. CVRP with 10 nodes and 5 vehicles each with a capacity of 30.

We use a simple linear model that scales in size based on the input size and output size. The outputs are kept constant, so we vary the input, i.e. number of features. They are varied from 2-20, meaning that the number of model parameters is scaled from 40-400 for knapsack and 20-200 for CVRP. The dataset sizes are varied from 20-80.

Each combination of number of features and dataset size gets 20 different seeds, of which the results are averaged.

When the number of data samples increases, all the computations become slower while the number of available pivots becomes much larger. Together, this can make for a very long run time of hours for a single result. To keep the amount of computation given to DGL fair, we set a time limit of 5 minutes. To make sure it does not spend all this time looking for pivots on only a small subset of the data samples, we set $k$, the max number of decisions returned by the adjacent decision finder, to $10$.

The results are shown in two measurements: First relative regret [6], as shown in Equation 5.5.

$$\text{Relative Regret}(\hat{x}, c) \;=\; \frac{c^\top \hat{x} - c^\top x^*}{c^\top x^*}. \tag{5.5}$$

Relative regret equals $0$ when the prediction is optimal, a value of $0.2$ means the cost is $20\%$ higher than optimal, and a value of $1$ means the prediction is twice as costly as the optimal decision.

The second is relative improvement as shown in Equation 5.6:

$$\text{Improvement Ratio} \;=\; \frac{\text{Regret}_{\text{CaVE}}}{\text{Regret}_{\text{CaVE+DGL}}}. \tag{5.6}$$

An improvement ratio of $1$ means both methods perform equally, a value of $2$ means CaVE+DGL achieves half the regret of CaVE, values below $1$ mean regret became worse. We aggregate the improvement ratios using the geometric mean.

### 5.3.2. Locality experiment

DGL is designed to improve locality, so in this section, we show that this is the case.

In Figure 5.3 we observe the results of the experiment. On the left, the locality of CaVE after a gradient descent procedure. On the right, the locality of those same models after being further optimized by DGL. In between, the flow from CaVE to DGL locality is shown.

First, we observe that the share of global locality has increased drastically while random has gone from almost 66% to 30%. The flow in between shows how these changes came to be. Over half of the random localities are cut down. We see that the majority moves into global, with a small part moving to local. DGL can drastically improve locality, making non-locally optimal solutions a minority.

Second, we see that the flow from local is also very interesting. We see that the majority remains in the local optimum, as you would expect from a local search algorithm. However, we also see a decent amount move into global and (better regret) random segments. This shows that while simplex pivots provide a decent approximation of the local regret structure, they can also extend outside of the directly adjacent segments. The opposite is also true, for some random points, no better solutions are found, even though in the regret landscape, they are adjacent. This can be caused by either the aligner not getting to the decision or the decision finder not considering it, due to, for example, the relaxation of the problem or having many degenerate solutions.

Figure 5.4 shows the distribution of depth for local and random solutions. This shows that even when DGL does not find the global, the average solution quality is still brought up a lot.

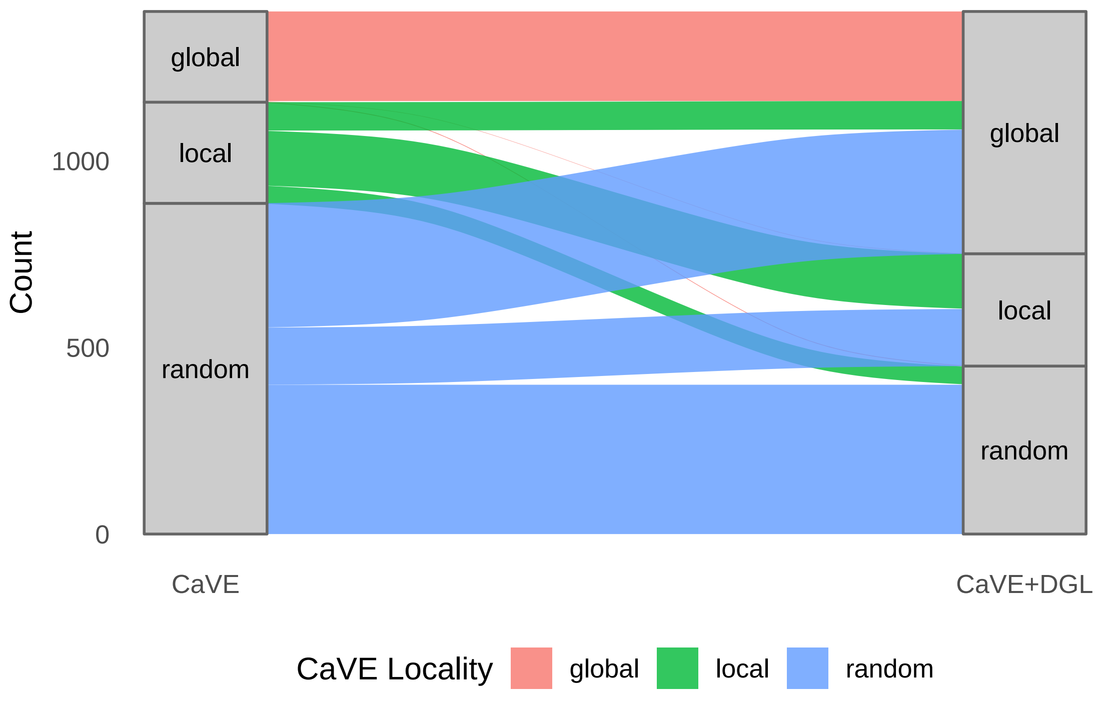


**Figure 5.3:** Change in distribution of locality from CaVE to CaVE+DGL.

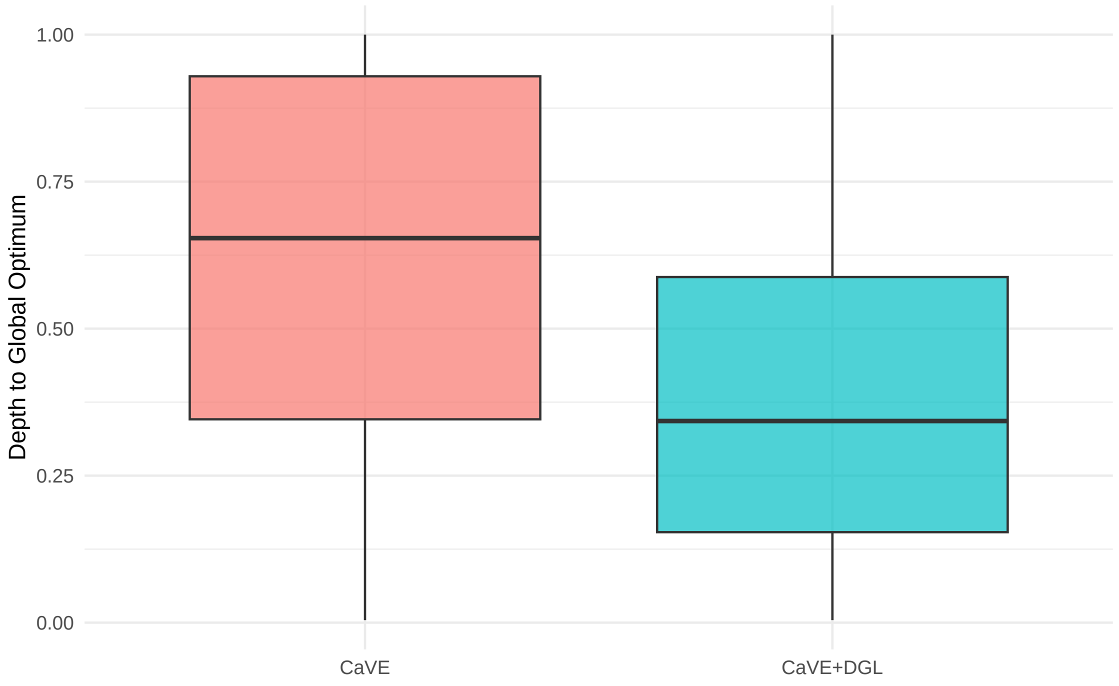


**Figure 5.4:** Distribution of Depth of CaVE vs CaVE+DGL for all local and random points.

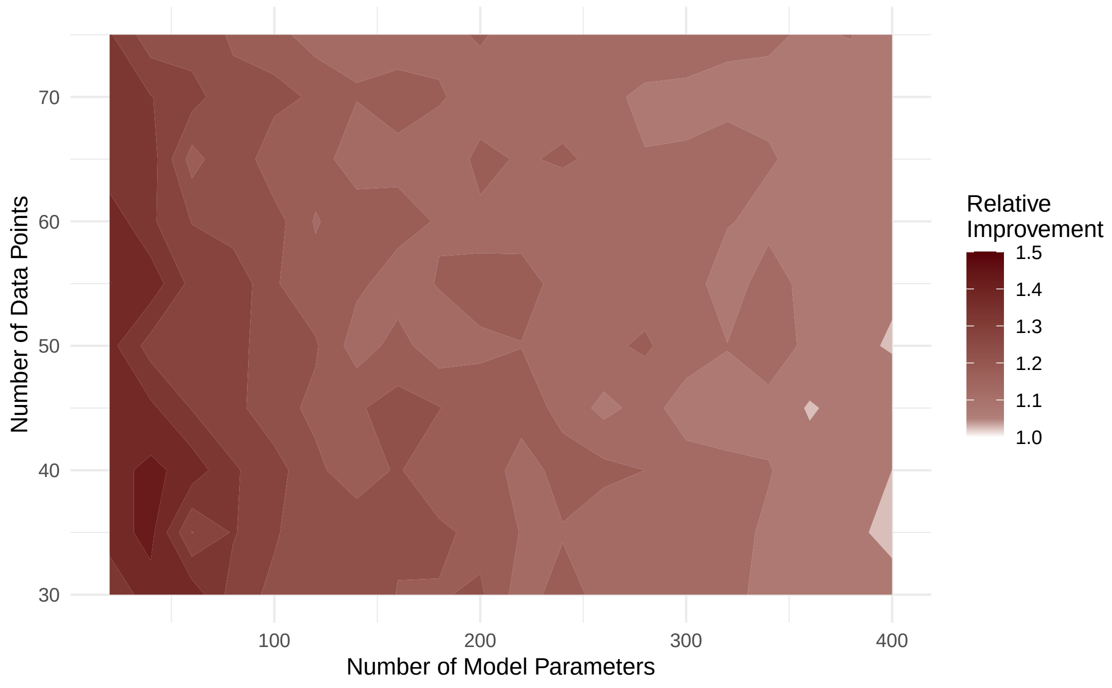


**Figure 5.5:** Relative training regret improvement of DGL over CaVE calculated on the Knapsack problem. The color determines the degree of improvement, while the axes show results for different number of data points and model parameters.

### 5.3.3. Regret experiments

There are two dimensions in which the problems are scaled up in the previously mentioned experiment settings, the number of samples (i.e. the size of the dataset) and model size. From the locality experiment we know that DGL works when both these factors are one. So we perform a grid search among which we scale up both of these parameters to find what the limiting factor is for DGL.

#### Knapsack Problem

In Figure 5.5, the relative improvement DGL finds over cave is shown for train regret. A very clear pattern can be observed where, as the model size increases, the relative improvement decreases. We can also see a slight trend that when data increases, the improvements get a little bit less, but this is very weak in comparison. The first pattern has two factors that explain it. First, the Curse of dimensionality; aligning and finding better solutions becomes more difficult as the dimensionality of the model increases. Secondly, when the model size increases, CaVE can find better models by itself, which leaves less room for DGL to find improvements. The latter is shown by CaVE training regret in Figure 5.6, where we see that the worst regret areas largely correspond with the biggest improvements.

The effect that when the solution is already good, there is less room for improvement seems really intuitive and obvious, but it can also be explained by one of the limitations of this implementation of DGL. When the regret becomes small, it often also means that the model has learned a pattern that spans all data samples. In this case DGL still tries to improve it by looking at one data sample at a time, finding pivots for it that would improve that problem, but then breaking the larger pattern that was discovered by the model.

Figure 5.7 shows the improvement in test regret of DGL. This once again highlights the improvements for smaller models. It seems that 100 model parameters is a threshold where DGL starts to have a smaller impact. But below that, DGL consistently improves regret with margins between

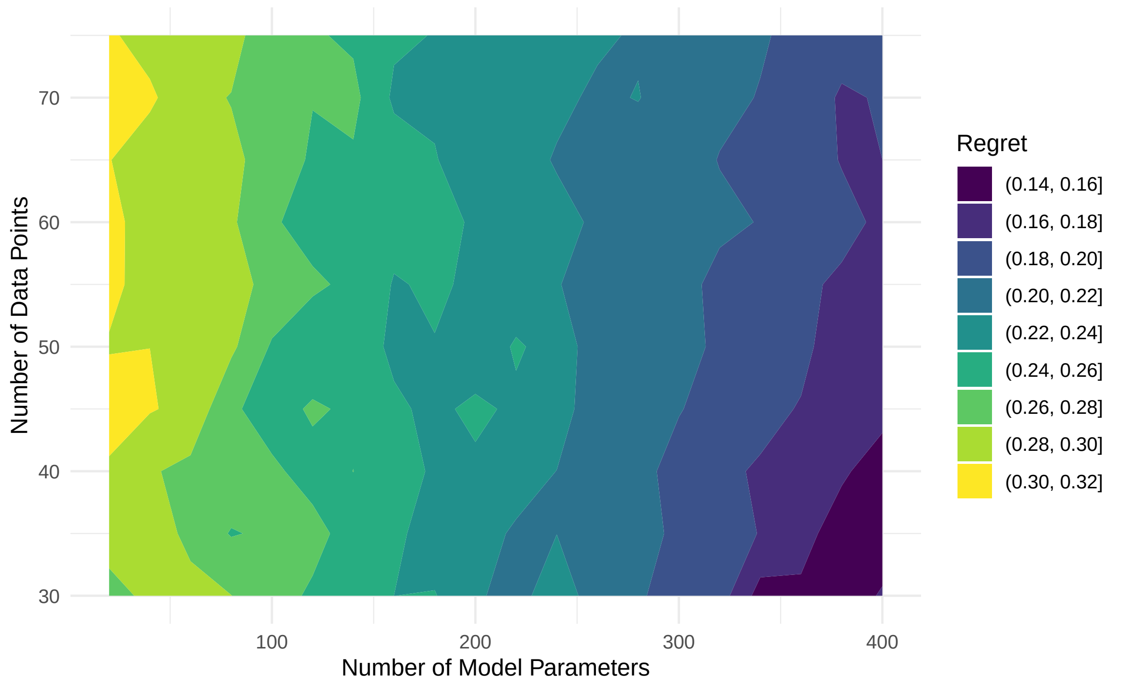


**Figure 5.6:** The training regret of the model trained using CaVE for the Knapsack problem . The color determines the regret value, while the axes differentiate between the number of data points and model parameters.

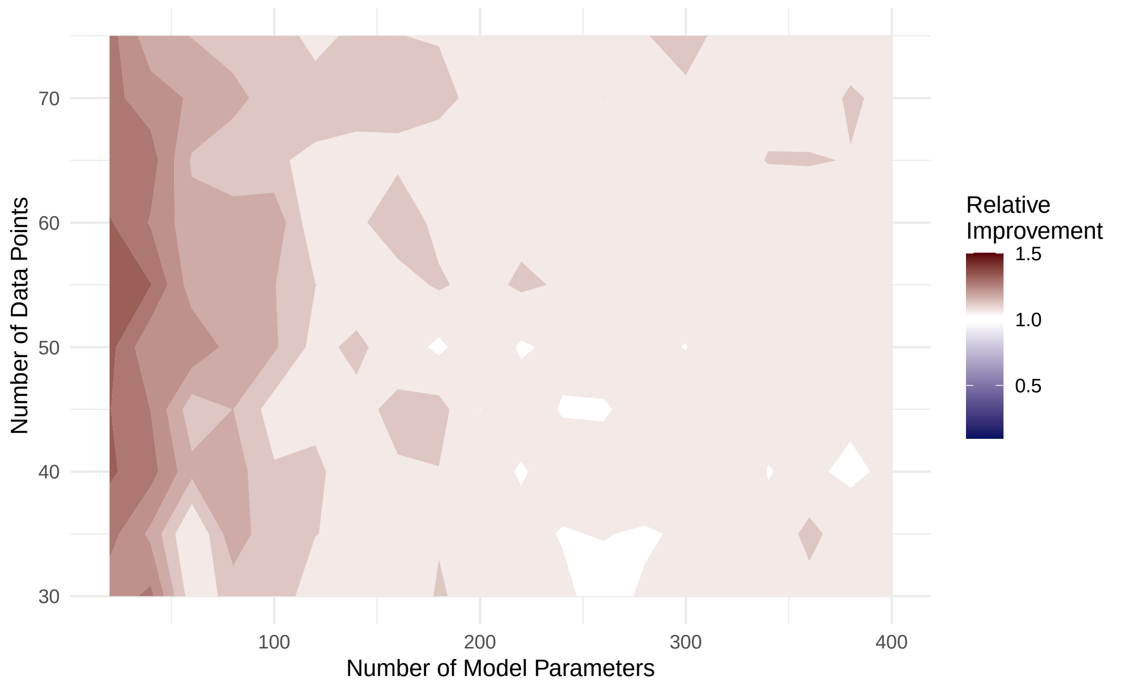


**Figure 5.7:** Relative regret improvement of DGL over CaVE calculated on the Knapsack test set. The color determines the degree of improvement, while the axes show results for different number of data points and model parameters.

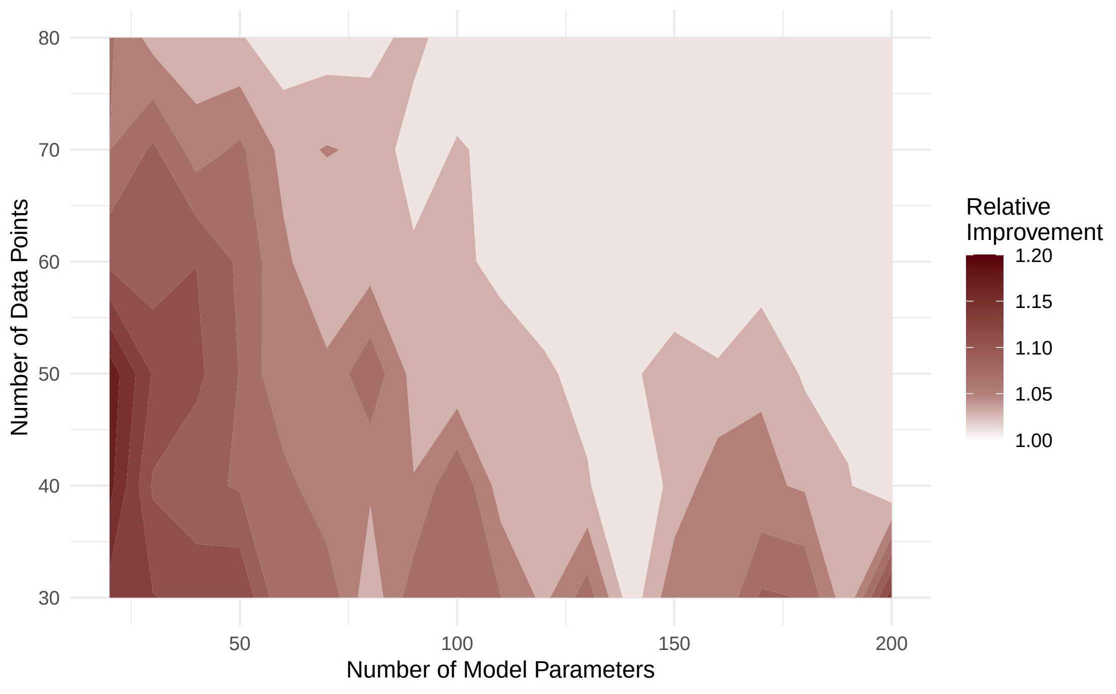


**Figure 5.8:** Relative regret improvement of DGL over CaVE calculated on the CVRP train set. The color determines the degree of improvement, while the axes show results for different number of data points and model parameters.

20-40%.

The number of data points seems largely inconsequential, though at very small numbers, DGL finds less strong improvements. So once we have 30-40 data points, the strong improvement for smaller models is stable.

We can also observe that DGL does not increase overfitting, which could have been expected. On average, there are slight improvements everywhere.

### Capacitated Vehicle Routing Problem

CVRP is what authors of CaVE used to showcase its strengths [18]. On this problem, we know that CaVE already performs really well, so we see if DGL is still able to provide value.

In Figure 5.8 we observe the train improvement. Relatively large improvements are found for smaller models, while for larger models, especially with many data points, almost no improvements are found. The drop off is a lot faster for CVRP, but this pattern is consistent with the knapsack problem. The best improvements are found for smaller models.

The improvement for test regret however, looks very different. In Figure 5.9 we see a large blue area where the solutions become worse, a lot of white, and a red area for larger models. Although this seems very contradictory to the findings for knapsack, we can explain this in a consistent way. For it, we need to look at the CaVE test regret in Figure 5.10. Interestingly, we see that the test regret is relatively low in the exact area where DGL finds improvements. Same as for the knapsack problem, DGL picks up the slack where CaVE performs the worst. But since this improvement is found for larger models, only small improvements can be found indicated by the light color in Figure 5.9. Similarly, we see some slight improvements for smaller models, but because such good test regret is already found in those areas, there is not much to improve for DGL.

The large blue area shows that when a model already does really well on a problem, DGL can also cause overfitting. Although the decrease is small, it shows that using DGL carelessly can

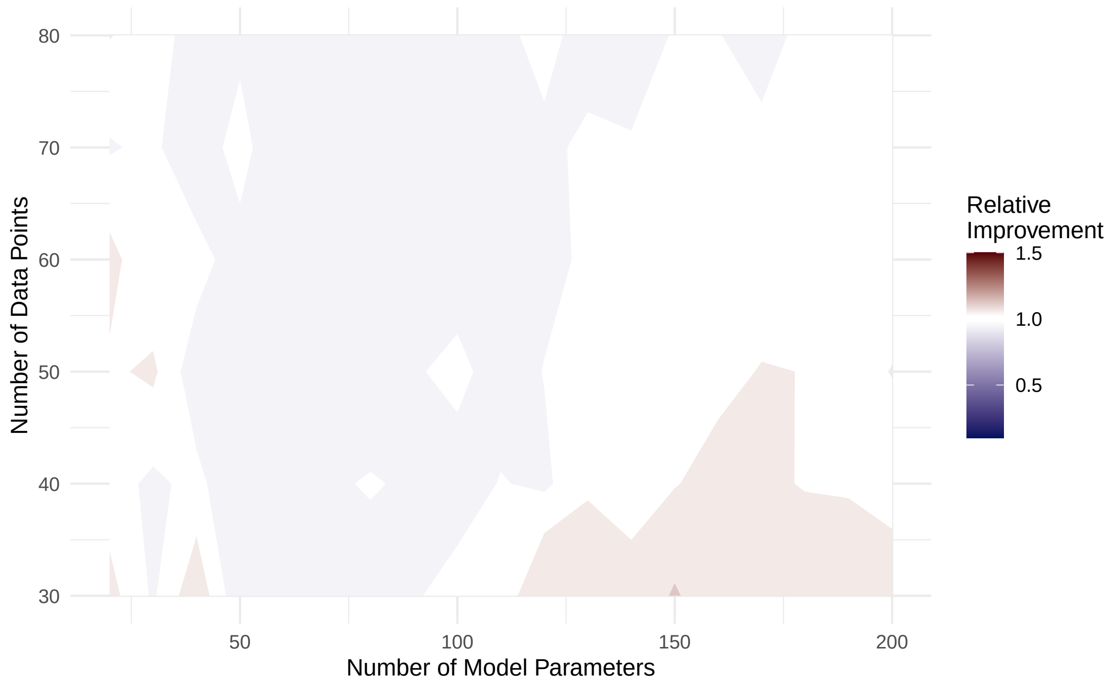


**Figure 5.9:** Relative regret improvement of DGL over CaVE calculated on the CVRP test set. The color determines the degree of improvement, while the axes show results for different number of data points and model parameters.

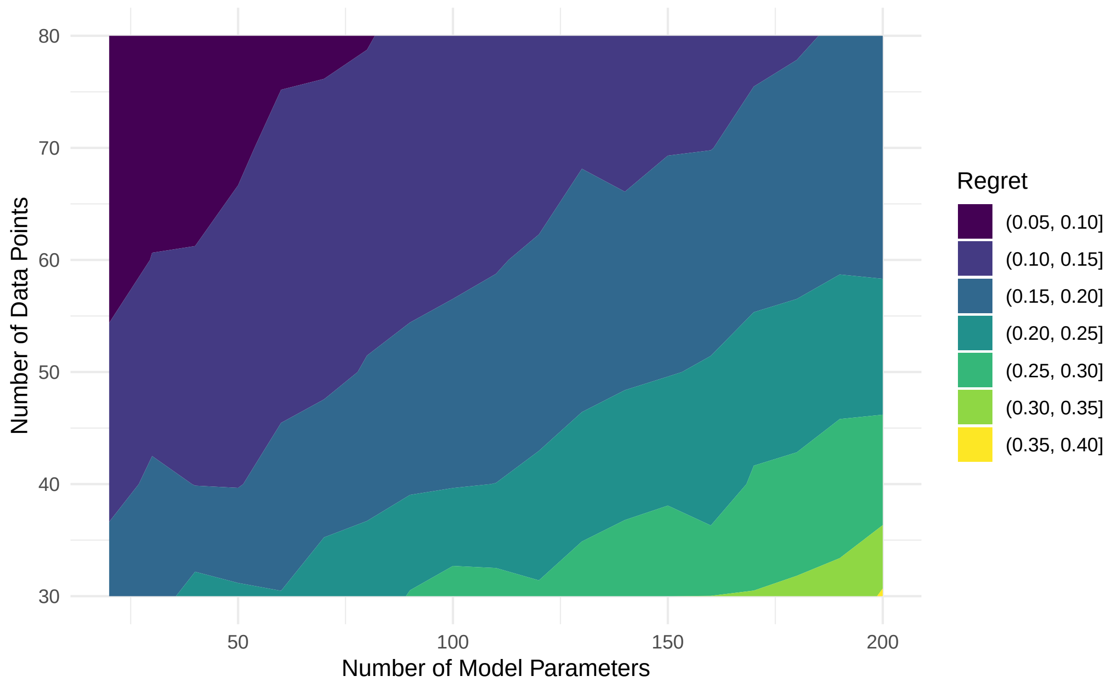


**Figure 5.10:** The regret of the model trained using CaVE for the CVRP test set. The color determines the regret value, while the axes differentiate between the number of data points and model parameters.

also be harmful.

## 5.4. Summary

In short, to improve local regret, DGL combines both gradient-based and gradient-free techniques in a novel way. By identifying adjacent points on the feasible polytope, we discover directions to improve regret where gradients fail to do so. Then, using gradients, we build a bridge between the current and adjacent points using normal cones for the model to follow. This two-step process makes learning from regret a possibility.

We tested DGL on locality and found significant improvements. Random localities are cut down, letting global segments have the majority and improving average solution quality drastically.

We also tested DGL in a normal regret setting, but with a varying dataset and model size. Here, we found that DGL mainly finds good improvements for smaller models. It is limited by two factors: The model size and the current solution quality. When model sizes get big, the curse of dimensionality makes it increasingly hard to explore the space. When the solution quality gets good, there is not much room for DGL to find significant improvements in the local regret space.

# 6 Conclusion

In this work, we question to what extent the gradient-based P+O methods that are approximations and surrogates of regret manage to approach true regret, and how we can improve them.

By comparing P+O loss landscapes with the true regret landscape, we are able to see if they define the same objectives in their local landscapes. We found that P+O losses define stationary points in regret locally optimal places around 50% of the time. In the other half of the instances, they would lead their learning models towards non-locally optimal points in the regret landscape.

In these unoptimized points, we were not able to find a pattern that links them to the regret landscape. Instead, it seems the objective is linked to something else, causing it to misalign with regret. In general, we found that P+O losses can align with the global optimum but inconsistently and show no awareness of the finer details of the regret landscape. From this, we conclude that the P+O losses we tested follow the local structure of regret only globally and inconsistently.

To improve the locality of P+O losses, we develop a regret local search algorithm: Decision Guided Learning (DGL). It searches the regret landscape from the decision side, reducing the local search space to a manageable size.

We performed the same experiment now with DGL and found a drastic increase in the ability to align with regret local optima.

To compare with the true regret, we have to use a heavily simplified setting, so we also tested DGL on more classical regret settings. We found that DGL is mainly able to find improvements for smaller models, and when it cannot find very good results for itself. Once the models get bigger and/or the initial regret score is already better, the improvement DGL can bring diminishes.

To conclude, we found that P+O loss functions do not align with regret very well. We developed a method to improve this, and it was able to provide good improvements for smaller learning models, while it struggled for larger models.

**Future Work** Here is a list of some things we are interested in but were not able to dive deeper into:

1. Having to calculate the true regret landscape poses restrictions on the model. We used a simple one-parameter model, and that way, we were able to adapt an existing method to perform this task. Calculating a higher-dimensional regret landscape is possible, but a new method has to be developed for this task. Seeing how these results generalize to higher-dimensional models would give great insight into how this research connects to more commonly used larger models.

2. In chapter 4 we found that combining loss functions has great potential to improve regret. Looking more in-depth at how much potential this brings and researching how to practically combine loss functions looks like a promising research direction.
3. In chapter 5 we found that when the model is already well optimized the local search has a diminished effect. We hypothesized that this was because looking to improve one problem at a time breaks larger patterns and thus worsens overall regret over the dataset. If we can expand DGL to be able to look for moves in larger neighborhoods, it might be able to find good improvements, even for already well-optimized models.

# References

[1] Brandon Amos and J. Zico Kolter. “OptNet: Differentiable Optimization as a Layer in Neural Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. June 2017, pp. 136–145. url: `https://proceedings.mlr.press/v70/amos17a.html`.

[2] Quentin Berthet et al. *Learning with Differentiable Perturbed Optimizers*. June 9, 2020. doi: `10.48550/arXiv.2002.08676`.

[3] Lieven Boyd. *Stephen and Vandenberghe, Convex optimization theory*. Cambridge university press, 2004. isbn: 978-0521833783.

[4] Emir Demirović et al. “Dynamic Programming for Predict+Optimise”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (Apr. 3, 2020), pp. 1444–1451. doi: `10.1609/aaai.v34i02.5502`.

[5] Justin Domke. “Implicit Differentiation by Perturbation”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010.

[6] Adam N. Elmachtoub and Paul Grigas. *Smart ”Predict, then Optimize”*. Nov. 19, 2020. doi: `10.48550/arXiv.1710.08005`.

[7] Aaron Ferber et al. *MIPaaL: Mixed Integer Program as a Layer*. July 17, 2019. doi: `10.48550/arXiv.1907.05912`.

[8] Stephen Gould et al. *On Differentiating Parameterized Argmin and Argmax Problems with Application to Bi-level Optimization*. July 20, 2016. doi: `10.48550/arXiv.1607.05447`.

[9] Ali Ugur Guler et al. “A Divide and Conquer Algorithm for Predict+Optimize with Non-convex Problems”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.4 (June 2022), pp. 3749–3757. doi: `10.1609/aaai.v36i4.20289`.

[10] Michael U Gutmann and Aapo Hyvärinen. “Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics.” In: *Journal of machine learning research* 13.2 (2012). issn: 1532-4435.

[11] Jayanta Mandi and Tias Guns. “Interior Point Solving for LP-based prediction+optimisation”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 7272–7282. doi: `10.48550/arXiv.2010.13943`.

[12] Jayanta Mandi et al. *Decision-Focused Learning: Foundations, State of the Art, Benchmark and Future Opportunities*. Aug. 16, 2023. doi: `10.48550/arXiv.2307.13565`.

[13] Jayanta Mandi et al. *Decision-Focused Learning: Through the Lens of Learning to Rank*. 2022. arXiv: `2112.03609 [cs.LG]`. url: `https://arxiv.org/abs/2112.03609`.

[14] Jayanta Mandi et al. “Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.02 (Apr. 2020), pp. 1603–1610. doi: `10.1609/aaai.v34i02.5521`.

[15] Andriy Mnih and Koray Kavukcuoglu. “Learning word embeddings efficiently with noise-contrastive estimation”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 2265–2273.

[16] Maxime Mulamba et al. “Contrastive Losses and Solution Caching for Predict-and-Optimize”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*. Aug. 2021, pp. 2833–2840. isbn: 978-0-9992411-9-6. doi: `10.24963/ijcai.2021/390`.

[17] George Papandreou and Alan L. Yuille. “Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models”. In: *2011 International Conference on Computer Vision*. 2011, pp. 193–200. doi: 10.1109/ICCV.2011.6126242.

[18] Bo Tang and Elias B. Khalil. *CaVE: A Cone-Aligned Approach for Fast Predict-then-optimize with Binary Linear Programs*. Mar. 15, 2024. doi: 10.48550/arXiv.2312.07718.

[19] Bo Tang and Elias B. Khalil. *PyEPO: A PyTorch-based End-to-End Predict-then-Optimize Library for Linear and Integer Programming*. 2023. arXiv: 2206.14234 [math.OC].

[20] Robert J Vanderbei. “Linear programming”. In: *Encyclopedia of Applied and Computational Mathematics*. Springer, 2015, pp. 796–800. isbn: 978-3-540-70528-4.

[21] Marin Vlastelica et al. *Differentiation of Blackbox Combinatorial Solvers*. Feb. 16, 2020. doi: 10.48550/arXiv.1912.02175.

[22] Bryan Wilder, Bistra Dilkina, and Milind Tambe. *Melding the Data-Decisions Pipeline: Decision-Focused Learning for Combinatorial Optimization*. Nov. 20, 2018. doi: 10.48550/arXiv.1809.05504.

[23] Günter M Ziegler. *Lectures on polytopes*. Vol. 152. Springer Science & Business Media, 2012.

# A

# Plots

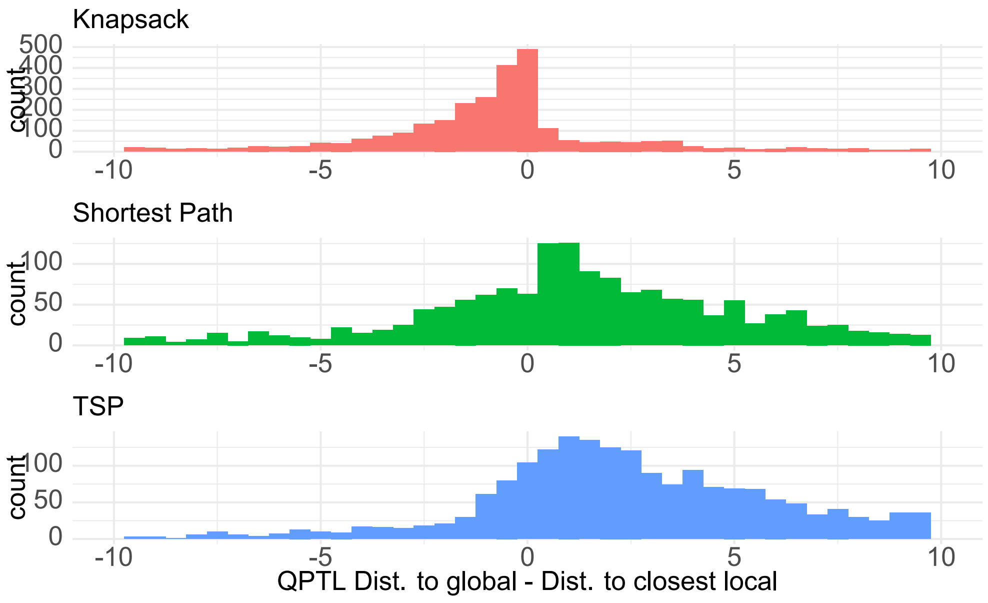


**Figure A.1:** QPTL distribution of the difference between the distance to the global optimum and the distance to the closest non-global local optimum.

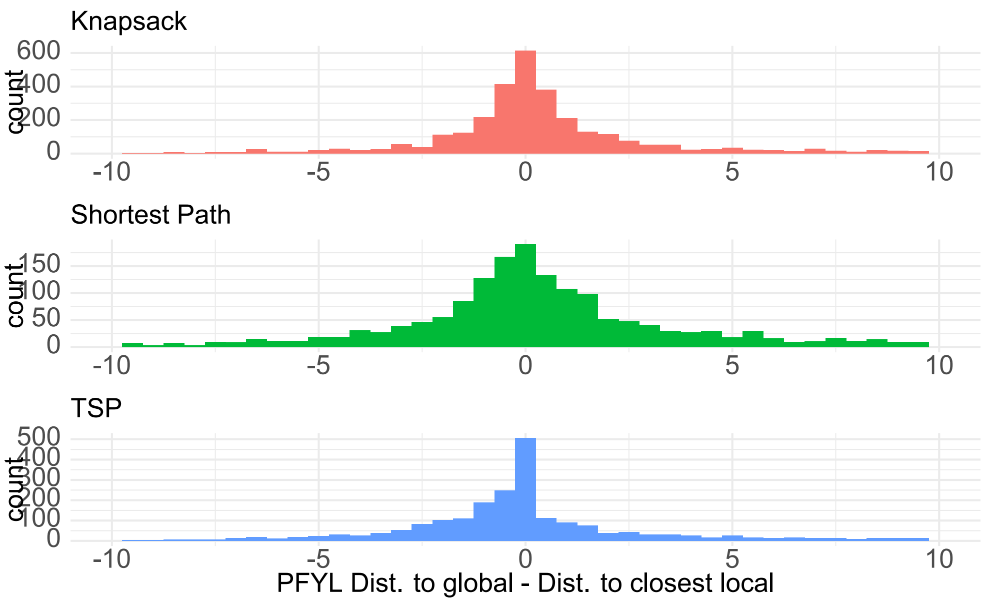


**Figure A.2:** PFYL distribution of the difference between the distance to the global optimum and the distance to the closest non-global local optimum.

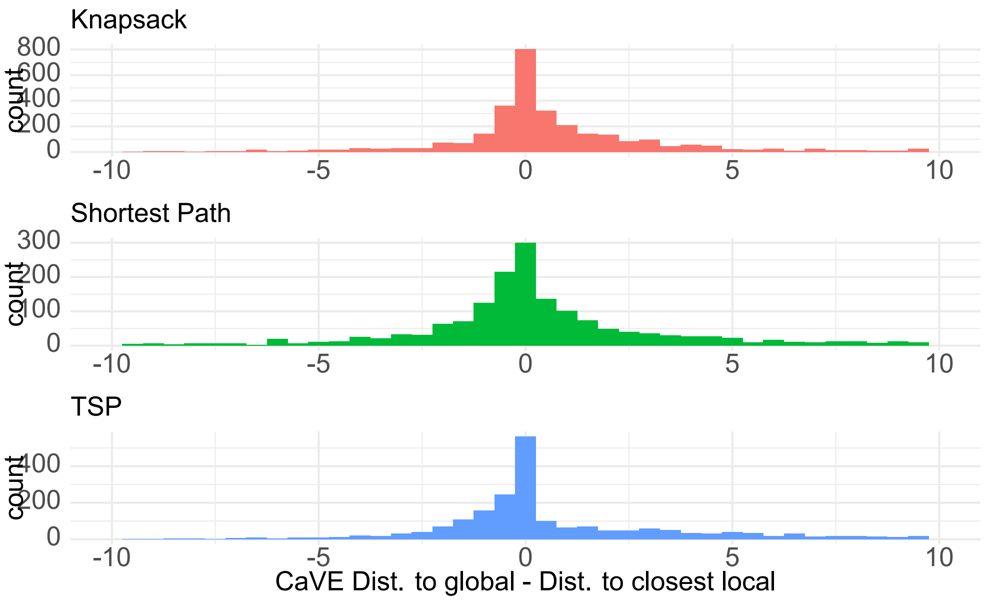


**Figure A.3:** CaVE distribution of the difference between the distance to the global optimum and the distance to the closest non-global local optimum.