MIE424 Final Project Report: *Smart, Predict then Optimize* Implementation Project

N. Al Aker, J. Ashworth, J. Chu, and M. Jia

Abstract—Real-world optimization problems are often missing key parameters. These optimization problems are normally solved with estimates for the missing parameters given by a machine learning (ML) prediction model in a *Predict, then Optimize* framework. While most ML loss functions are intended to minimize the prediction error in their estimates, Elmachtoub and Grigas propose an improved framework, "*Smart, Predict then Optimize*", that aims to minimize the decision error in ML models instead of prediction error. Their framework includes a novel loss function, SPO Loss, and a surrogate loss function SPO+ Loss, which can be used to train ML models that estimate parameters for optimization problems. While the SPO paper studies the performance of the SPO and SPO+ Loss functions in minimizing decision error on several optimization problems, we instead study the performance of two different methods that solve SPO+ Loss in optimization problems. We experimented with minimizing SPO+ Loss using two different methods: Stochastic Gradient Descent (SGD), and a Linear Program (LP) reformulation of SPO+ Loss, to empirically determine which problem conditions are most favourable for each method. We compared both methods in solving several instances of the shortest path problem, where we varied the number of data points and features in the data. We ran experiments on 40 different problem sizes 30 times each using Python 3.9.0 on a Windows machine with an i7-10700F CPU. Our results show that SGD is more computationally efficient than the LP approach when minimizing the SPO+ loss on a square grid shortest path problem, except for when the feature and sample size are both very small. However, it found solutions with 8-10% higher SPO loss than the LP method. In conclusion, SGD is the preferred optimization method for large problems given the tradeoff of slightly worse SPO loss versus the immense computation time savings. For small and medium sized problems, an informed decision should be made based on the available computation power to decide if the increased computation time of the LP method is worth the 8-10% improved accuracy.

Code available here: https://github.com/michaelzcjia/smart_predict_optimize

1 Introduction

Real-world optimization problems are often missing key parameters. These optimization problems are normally solved with estimates for the missing parameters given by a machine learning (ML) prediction model in a *Predict, then Optimize* framework. For example, in a vehicle routing problem, we may first need to estimate the cost of taking various routes based on current traffic, weather, holidays, etc. We can then feed these predicted costs into our optimization model which can determine the optimal route based on those predictions being correct. However, most ML models don't account for how their predictions will be used in the downstream optimization problem. Small prediction errors can drastically change the decisions made by the optimization model, which can amplify decision error, which is the regret between the recommended optimal solution and the true optimal solution.

Elmachtoub and Grigas [1] propose a solution to this paradigm called 'Smart, Predict then Optimize' (SPO). The SPO framework includes a novel loss function, *SPO Loss*, which they show minimizes decision error of ML predictions used in optimization problems. The authors also derive a surrogate loss function to SPO Loss, called *SPO+ Loss,* as SPO Loss is non-convex. While that paper studies the performance of models trained with the SPO+ Loss function compared to traditional loss functions like MSE in minimizing decision error on several optimization problems, we instead study the performance of two different methods that train models with SPO+ Loss. Specifically, we experiment with minimizing SPO+ Loss using Stochastic Gradient Descent (SGD) and a Linear Program (LP) reformulation of SPO+ Loss that can be solved using an optimization solver. The authors of the paper claim that the SGD method is better suited to solving large problem instances, while the LP method is better suited to solving smaller problem instances. We conduct a series of computational experiments to assess the validity of these claims by analyzing the runtime and performance of both the SGD and LP methods on different sizes of the shortest path problem. We use these results to analyze the tradeoffs between runtime and performance of the two methods and to empirically determine criteria on when each method should be used in practice.

2 Related Work

Elmachtoub and Grigas [1] established the SPO framework to minimize decision loss instead of prediction loss in *Predict, then Optimize* analytics problems. Their contributions in this foundational paper include:

1. The formal definition of the new loss function SPO Loss, which measures the decision error of cost vector predictions. The SPO Loss function is non-convex, which implies that training ML models with SPO Loss will be challenging.
2. The development of a surrogate loss function SPO+ Loss that is a convex upper-bound of SPO Loss. The authors derive an SGD method of algorithm for minimizing SPO+ Loss, and a linear optimization reformulation of the problem which is applicable when using linear prediction models.
3. Proof that under a simple instance of the SPO framework, SPO loss becomes 0-1 loss, and SPO+ Loss becomes Hinge Loss, showing that SPO Framework generalizes to a wide family of optimization problems.
4. Proof that the SPO+ Loss function follows the Fisher consistency property, which is an essential

property of any surrogate loss function.

5. Validation of the SPO framework through numerical experiments on the shortest path and portfolio optimization problems, showing that the value of the SPO framework increases as the degree of model misspecification increases.

The authors recognize that aiming to reduce decision error instead of prediction error in Predict, then Optimize type problems is not novel. Kao et. al. [2] is cited as the most closely related work to SPO, whose work also seeks to train an ML model that minimizes loss with respect to a nominal optimization problem. Their work produced a novel Directed Regression algorithm that combines ordinary least squares and empirical optimization. However, their framework is only applicable to unconstrained optimization problems, while the SPO framework generalizes to both unconstrained and constrained problems.

The contributions of the Smart, Predict then Optimize paper has led to several other works that build on this framework. Elmachtoub et. al [3] extended the SPO framework by developing a methodology called SPO Trees, which uses SPO Loss to train decision trees that estimate optimization problem parameters. Their work eliminates the need for the SPO+ Loss surrogate function, which is not guaranteed to produce optimal decisions with respect to SPO Loss, by exploiting the structure of decision trees to train models directly with SPO Loss. The author's claim that, to the best of their knowledge, they have developed the first tractable methodology for training an ML model directly with SPO loss.

Because computing the subgradient of the SPO+ loss for each iteration of SGD requires solving the downstream optimization model, training SPO models for hard combinatorial optimization problems is computationally expensive. Mandy et. al [4] showed that computing the subgradient of combinatorial problems with a relaxation of the discrete constraints leads to similar performance as learning with the full discrete optimization problem while converging much quicker making training SPO models much more tractable for combinatorial problems.

3 Methods

Elmachtoub and Grigas proposed two methods to train linear models with SPO+ loss: direct optimization through a LP, and iterative optimization through SGD [1]. To compare the performance of both approaches, we used a generic shortest path problem formulation. In this section, we present the definitions of SPO loss, SPO+ loss, and the formulations of the LP optimization model, the SGD optimization model, and the shortest path problem.

3.1 SPO and SPO+ Loss

The goal of the 'Smart, Predict then Optimize' framework is to minimize SPO Loss — which quantifies the regret from suboptimal decisions. SPO loss is defined as follows:

$$l_{SPO}(\hat{c}, c) = max_{w \in W*(c)}\{c^T w\} - z^*(c) \tag{1}$$

In (1), c is the ground truth cost vector, and $\hat{c}$ is the predicted cost vector, w is the set of optimized decision variables given a predicted cost vector $\hat{c}$, and $W^*(\hat{c})$ is the optimal set of solutions given $\hat{c}$. $z^*(c)$ is the optimal objective function when solved with the true cost vector c. Note that in most cases $W^*(\hat{c})$ is a singleton, and the $max_{wW^*(c)}$ is only relevant when there is a degenerate solution.

While it would be ideal to optimize using SPO loss directly, this is difficult in practice because the SPO loss function is non-convex. In its place, we use the SPO+ loss function, which is a convex upper bound of SPO loss [1]. SPO+ loss is defined as the following:

$$l_{SPO+}(\hat{c}, c) := max_{w \in S}\{c^T w - 2\hat{c}^T w\} + 2\hat{c}^T w^*(c) - z^*(c) \tag{2}$$

In (2), c, w, and z, share the same definitions as in (1), and S is the set of all feasible solutions to the optimization problem.

It's important to note that while SPO+ loss is non-differentiable, Elmachtoub and Grigas showed that (3) is a subgradient of SPO+ loss and can be used to minimize SPO+ loss with SGD [1].

$$w^*(c) - w^*(2\hat{c} - c) \tag{3}$$

3.2 Linear Programming Optimization Method

When the prediction model to be trained is linear and the constraint space of the optimization problem is also linear Elmachtoub and Grigas [1] showed that SPO+ loss can be minimized directly with an LP as shown in (4). In this LP, the decision variables, B (in matrix form), are the trainable parameters of the linear prediction model.

$$min_{B,p} \quad \frac{1}{n}\sum_{i=1}^{n} \quad [-b^T p_i + 2(w^*(c_i) x_i^T) \cdot B - z^*(c_i)] \tag{4}$$

$$\begin{aligned} s.t \quad & A^T p_i \le 2Bx_i - c_i \quad \forall\, i \\ & p_i \in \mathbb{R}^m, p_i \ge 0 \quad \forall\, i \\ & B \in \mathbb{R}^{d \times p}. \end{aligned}$$

In (4), x_i is a single vector of feature values for sample i out of n data samples, b is a vector representing the right-hand side of each constraint in the downstream optimization problem, A, is a matrix that represents the left-hand side of those constraints, and the p_i are the dual variables of the first term of (2) (see [1] for the derivation of (4) using duality). It should be noted that the first constraint of (4) comes from the dual simplification of the first term of (2) and is an inequality constraint if the decision variables in the downstream optimization problem are non-negative and an equality constraint if those decision variables were unconstrained. The variables w, c, and function z are the same as they were defined in the section 3.1.

3.3 Stochastic Gradient Descent Optimization Method

The other method to train models with SPO+ loss is SGD (see [1] for the SGD algorithm). In this methodology, random samples of *batch_size* data points are chosen each iteration, and SPO+ loss is optimized iteratively by modifying the model parameters, *B*, at each iteration by computing the subgradient of the loss function with respect to the model parameters. Since SPO+ loss is non-differentiable, the subgradient presented in section 3.2 was used instead of a gradient. For consistency, all hyperparameters were chosen based on the recommendations from Elmachtoub and Grigas [1]. The hyper parameters chosen were as follows:

- Learning rate at iteration t: $\gamma_t = \frac{1}{\sqrt{t+1}}$
- Batch size is a fixed constant: 10

As with the LP formulation, a regularization term was not added to the SPO+ loss function.

3.4 Shortest Path Problem

All experiments were tested using a square, grid shaped shortest path problem, where each edge i in the grid has a cost c_i. The goal is to find the lowest cost path from the start node to the goal node. The size of the square grid is *s*-by-*s*, which corresponds to s^2 nodes and *s*(*s*-1)*2 edges. We formulate and solve this problem as a LP with flow constraints: each node in the grid must have equal inflow and outflow, except for the start node and the end node which only has one outflow and one inflow respectively.

4 Datasets

To generate data for the shortest path problem, we followed the synthetic data generation procedure outlined by Elmachtoub and Grigas in the original SPO paper [1]. This is the same process they used for their shortest path computational experiments.

TABLE 1
Synthetic Data Variable Descriptions

Variable	Description
s	Dimension of square shortest path grid
d	Total number of edges in the network computed from s.
p	The number of features to estimate the cost vector
B^*	Parameters of true model of dimension $d \times p$.
x_i	Feature vector of dimension p
c_i	Cost vector of dimension d
$\epsilon_i{}^j$	Multiplicative noise term applied to the cost vector
I_p	Variance of each feature in x_i
deg	The degree to which the features are correlated with the cost vector.

The synthetic data was generated via the following procedure (see Table 1 for input variables):

1. Define the number of edges, d, based on the desired grid size (s), where
   d = *s**(*s*-1)*2
2. Generate a random matrix $B^* \in \mathbb{R}^{d\times p}$ that encodes the parameters of the true model where p is a given number of features. Each entry of B is a Bernoulli random variable that is equal to 1 with probability 0.5.
3. Generate the feature vector $x_i \in \mathbb{R}^p$ by sampling a multivariate Gaussian distribution with i.i.d. standard normal entries i.e., $x_i \sim N(0, I_p)$
4. Generate the noise vector ϵ_{ij} which is a multiplicative noise term that is sampled from the random uniform distribution on $[1-\xi, 1+\xi]$, where $\xi \geq 0$.
5. Generate the cost vector according to $c_{ij} = [\frac{1}{\sqrt{p}}(B^* x_i)_j + 3)^{deg} + 1] \cdot \epsilon_i{}^j$. Here, c_{ij} denotes the jth component of c_i, and $(B^* x_i)_j$ denotes the *jth* component of $B^* x_i$, where *j* = *1,…,d*.

5 Experiments

Experiments were run to compare the performance of fitting linear SPO models with the direct LP approach versus the SGD approach. The experiments were designed to answer the following three questions:

1. At what size of problem does the SGD approach become more computationally efficient in terms of runtime than the LP approach?
2. By how much does the LP approach outperform the SGD in terms of SPO+ loss and does this advantage also translate to better performance in true SPO loss?
3. How does the size of the prediction model and the size of the data sample affect the performance of the two approaches?[1]

Answering these three questions will aid future researchers in deciding which approach to use for their problem by knowing the tradeoffs of the two approaches in terms of runtime and loss.

Experiments were run using the shortest path problem formulation and data generation procedure discussed in Sections 3 and 4 respectively. The parameters used for the data generation procedure are shown in Table 2. The sample size and feature vector length were varied to investigate the effects of data sample size and prediction model size. All other parameters were held constant and

[1]The effect of the size of the optimization problem was not investigated due to computational limits.

drawn from the original shortest path experiments run by Elmachtoub and Grigas [1]. 30 sample problems were generated from each set of parameters and each sample problem was solved with both the direct LP and SGD approaches.

TABLE 2
PARAMETER VALUES USED

Parameter	Values
Sample size (*n*)	100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
p	5, 10, 15, 20
s	5
$\epsilon_i{}^j$	0.25
deg	3
I_p	0.2

All experiments were run using Python 3.9.0 on a Windows machine with an i7-10700F CPU. LPs for the shortest path problem and the direct solution approach were defined with the cvxpy package and solved with the ECOS solver. All other computation to fit the SPO models was done with numpy or native Python.

5.1 Runtime Results

Figure 1 shows the runtime of the SGD and LP methods to solving shortest path problems with varying sample sizes and feature vector sizes. Consistent with the claims made by Elmachtoub and Grigas [1], the LP approach is faster for small problems, but becomes less computationally efficient for larger problems. Quantifying this claim with our experiments on the shortest path problem with a grid size of 5 shows the LP approach is only faster than the SGD approach when the feature vector is small (p = 5) and the sample size is small (n < 375). For both the LP and SGD approach, the computation time increased with increasing sample size. However, the runtime for the LP approach increased at a much faster rate than the runtime of the SGD approach. This difference can be explained by the nature of the two approaches. SGD is an iterative method with a fixed batch size, so adding samples will only increase the runtime by delaying the stopping condition and increasing the number of iterations needed for convergence. These added iterations will have the same computational complexity as previous iterations. In contrast, every sample added to the LP approach increases the size of the LP by adding a decision variable, a term to the objective function, and a constraint the size of the cost vector of the nominal optimization problem (number of edges for the shortest path problem). Enlarging the LP like this greatly increases its complexity. Another observation from Figure 1 is that the variance is consistent across changes of n and p for the SGD method, but for the LP solution it generally increases with sample size.

Figure 1 also shows that the feature size (p) has a larger impact on runtime than the sample size (n) for both methods. For example, a doubling of the feature size from 5 to 10 approximately doubles the runtime of the SGD approach, while a doubling of the sample size for a fixed feature size has a much smaller impact on the SGD runtime. This can be explained by the feature size increasing the number of free parameters in the prediction model, while the sample size does not. Additionally, for the SGD method, feature size directly impacts the complexity of each iteration, while sample size only affects the number of iterations.

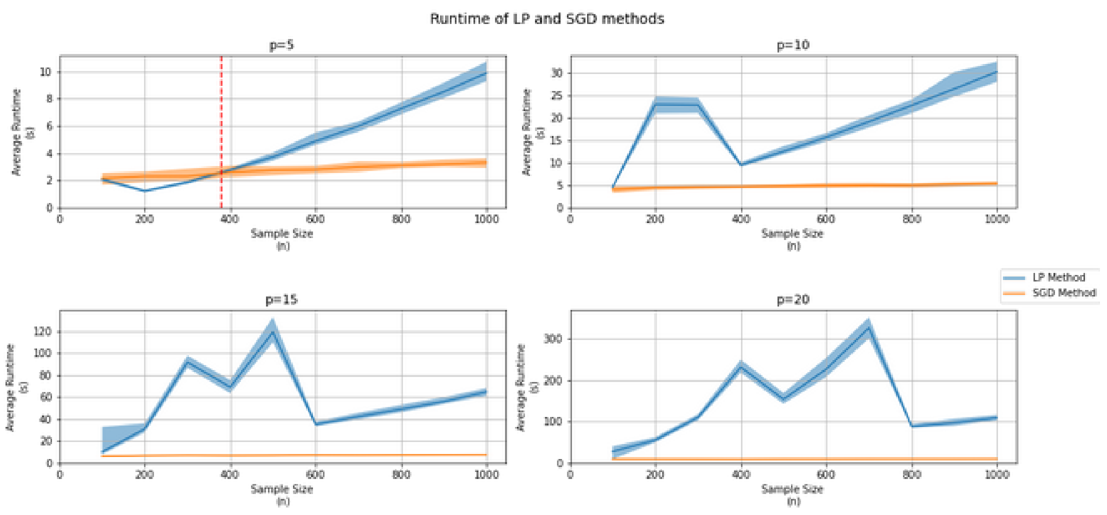

Figure 1. The runtime of the LP and SGD methods for varying sample size (n) and feature size (p). The solid lines denote the mean and the shading denotes a 95% confidence interval for the mean. For p=5, the LP method is faster when the sample size is less than 400. For all other parameter values the SGD method is faster. The runtime for both methods increases with increasing sample size and feature size, but the rate of increase is much higher for the LP method.

Another interesting observation from Figure 1 is that the runtime for the LP solution dips when the sample size is equal to the number of free parameters in the prediction model. Note that this number is less than the amount of decision variables in the LP because of the slack variables added by the dual problem in the direct solution formulation. When the feature size is 5, 10, 15, and 20 the number of free parameters in the linear prediction model is 200, 400, 600, and 800 respectively corresponding to decreases in runtime at those values of sample size in Figure 1. For a feature size of 15 and 20 there is an earlier dip when the sample size is 400 and 500 respectively for which the reason is unclear. We hypothesize that similar dips would occur for feature sizes of 5 and 10 if we had run experiments for sample sizes less than 100. Despite these interesting observations that the runtime of the LP approach can dip with increasing sample size at critical points, even with these dips the runtime for the LP approach is magnitudes higher than the runtime for the SGD approach, so from an algorithm comparison perspective these dips are irrelevant. The SGD approach is more computationally efficient than the LP approach except when the feature size and sample size are both very small.

5.2 Loss Results

Recall that the SPO+ loss is the tractable surrogate loss function used for training models, while SPO loss is the decision regret that is more relevant for measuring the effectiveness of the trained models. Figure 2 shows that SPO+ loss always upper bounds SPO loss as it should given its definition [1]. Additionally, Figure 2 shows the direct LP solution always has a lower mean SPO and SPO+ loss than the SGD approach. This mean is taken from 30 instances generated for each problem size, and for each specific instance the LP solution had a lower SPO+ loss as it should because it is a global minimizer of SPO+ loss. The better SPO+ loss usually translated to superior performance on SPO loss except for a couple instances where the SGD solution actually had a better SPO loss.

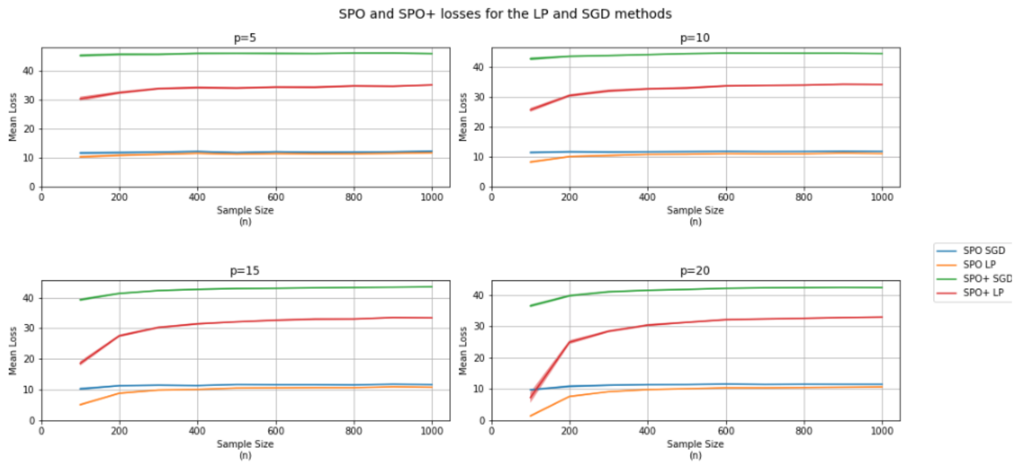

Figure 2. The absolute values of SPO and SPO+ loss functions for LP and SGD methods for varying sample size (n) and feature size (p). As expected, SPO+ loss upper-bounds SPO loss for each experiment. Additionally, the LP method always achieves a lower loss than the SGD method, which is expected as it is directly optimizing SPO+ loss.

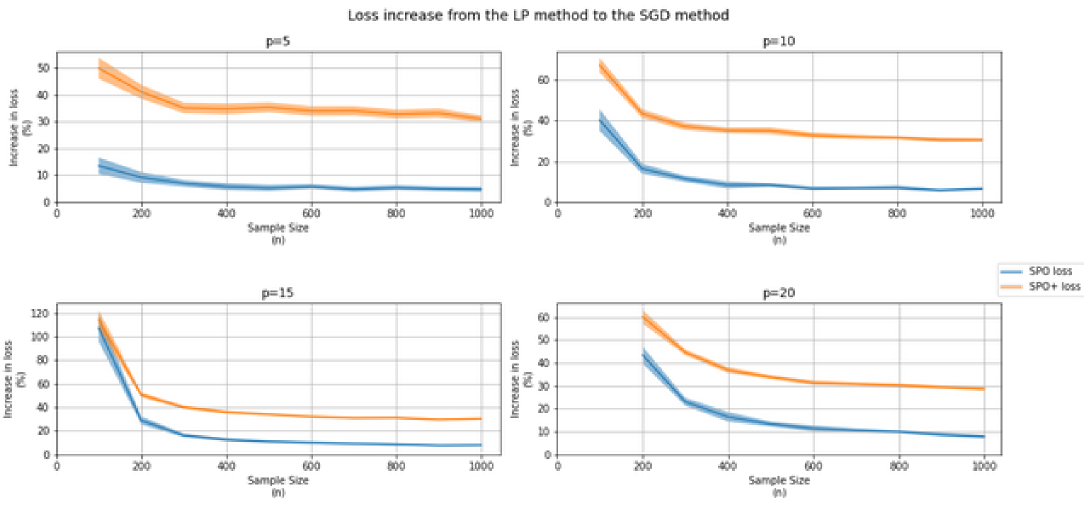

Figure 3. The loss of the SGD method benchmarked to the loss of the LP method for varying sample size (n) and feature size (p). The solid lines denote the mean and the shading denotes a 95% prediction interval. The SGD method has much higher loss for low sample sizes, but the discrepancy decreases with increasing sample size and converges at around 8-10% for all values of p. Note that for p=20, the data points for sample size of 100 were omitted due to a couple outliers that skewed the scale of the plot.

However, these outliers were drowned out from the plot when the means were computed across the 30 instances of each problem size.

Figure 3 compares performance of the two methods across each instance by showing the relative performance of the SGD method benchmarked to the LP method. For small sample sizes, the SGD models performed much worse than the LP models, but as sample size increases the performance of the SGD models converges to being only about 8-10% worse than the LP models for all feature sizes.

5.3 Discussion

For small problem instances, or when computation time is not a constraint, then the direct LP approach is preferable because it always minimizes SPO+ loss and usually outperforms SGD on SPO loss. However, for larger problems the direct LP approach has a significantly longer runtime than the SGD approach, and the SGD's performance on SPO loss is only about 8-10% worse. This tradeoff between runtime and performance should be considered in future work when determining what algorithm to use to fit linear SPO models. Another factor that could impact the choice between the LP and SGD approaches is the tuning of hyperparameters. The LP approach has no hyperparameters, while the SGD approach has a few. Our experiments used the hyperparameters recommended by Elmachtoub and Grigas [1], but there is no guarantee that these are optimal.

In this paper we evaluated the performance of the two algorithms on in-sample loss to evaluate their performance on fitting to data. Generalizing performance to out-of-sample data requires further and more complicated analysis. This is because out-of-sample performance is probably even more dependent on parameters we did not vary in our experiments such as feature variance, polynomial degree, and cost vector noise.

6 Conclusion and Future Works

We found that SGD is more computationally efficient than the direct LP approach when minimizing SPO+ loss for a square, grid shaped shortest path problem except for when the sample size and feature size are both very small. However, the direct LP approach always converges to more accurate solutions in terms of SPO+ loss and usually outperforms SGD in terms of SPO loss (8-10% better when sample size is large). This suggests that for very large problems SGD would be the preferred optimization method given the tradeoff of slightly worse SPO loss versus the immense computation time savings. For medium sized problems, a decision would have to be made based on the tradeoff of computation time and accuracy.

Future work would aim to provide more generalizable guidelines on the computational efficiency of training SPO models with the SGD and LP optimization methods using a variety of different problem formulations and datasets. This paper only analyzed the impact of sample size and feature size on performance because of computational limits. Future analysis should also analyze the impact of increasing the downstream optimization problem size. Additionally, the shortest path problem data could be extended to include randomly generated graph structures of non-grid shapes. Other problems such as portfolio optimization and the knapsack problem could also be tested with new datasets. This work could further be extended by utilizing real datasets rather than synthetic datasets, which would also allow model performance to be compared on out-of-sample (test) data to provide a more accurate representation of performance differences between optimization methods.

References

[1] A.N. Elmachtoub and P. Grigas. "Smart, Predict then Optimize" in *Management Science.* March 2021. [Online]. Available: https://doi.org/10.1287/mnsc.2020.3922

[2] Y. Kao, B. Roy and X. Yan. 2009, "Directed regression" in *NIPS,* 2009, pp. 889-887 [Online]. Available: https://papers.nips.cc/paper/2009/hash/0c74b7f78409a4022a2c4c5a5ca3ee19-Abstract.html

[3] A.N. Elmachtoub, J.C.N Liang, and R. McNellis, "Decision Trees for Decision-Making under the Predict-then-Optimize Framework", in *Proceedings of 37th International Conference on Machine Learning*, July 13-18 2020, PMLR 119: pp 2858-2867

[4] J. Mandi, T. Guns, E. Demirovic, and P.J. Stuckey. "Smart Predict-and-Optimize for Hard Combinatorial Optimization Problems", in *AAAI 2020: The Thirty-Fourth AAAI Conference on Artificial Intelligence, 2020,* pp. 1603-1610.

Contribution of Team Members

Contributions of team members are summarized below.

TABLE 3
Code Repository Contributions

Member	Code Contributions
Nada Al Aker	*get_item* and *get_y* functions, collaborated with Jamal on *ResultsAnalysis.ipynb*
Julian Ashworth	*generate_data* function, *problem_size_experiment* function
Jamal Chu	*DirectSolution* function, *ShortestPathSolver* class, collaborated with Nada on *ResultsAnalysis.ipynb*
Michael Jia	*GradientDescentSolution* function. Code clean up, docstring and comments

TABLE 4
Report Contributions

Member	Report Contributions
Nada Al Aker	Abstract (second half), Sections 5.1, 5.2, 5.3
Julian	Sections 1, 2, 4, References, Abstract (first

Ashworth	half), Document Formatting
Jamal Chu	Section 5.0 Edited Sections 1, 2, 3, 3.1, 3.2, 3.3, 3.4, 4, 5.1, 5.2, 5.3, 6
Michael Jia	Section 3, 3.1, 3.2, 3.4 and Section 6.0. Edited Sections 1, 2.