OPT2023: 15th Annual Workshop on Optimization for Machine Learning

Reducing Predict and Optimize to Convex Feasibility

Saurabh Mishra SAURABH_MISHRA@SFU.CA
Sharan Vaswani VASWANI.SHARAN@GMAIL.COM
Simon Fraser University

Abstract

Numerous applications in operations research and computer science require a combination of prediction and optimization – use historical data to predict the parameters of an optimization problem and solve the optimization problem to output a decision. Addressing these two challenges independently results in the *predict-then-optimize* problem. This approach can result in discrepancies between the prediction error, minimized during training, and the ultimate objective of minimizing the decision error. Consequently, recent work has focused on the *predict and optimize* (PO) framework, which focuses on training an end-to-end model from the data to the decisions. We focus on linear programs (LPs) within the PO framework, where the main challenge is handling the non-differentiability of LPs. For a linear prediction model, we present a novel reduction from PO to a convex feasibility problem. This reduction enables us to use alternating projections onto convex sets for solving the PO problem, resulting in a computationally efficient and theoretically principled algorithm. Finally, we validate the effectiveness of our approach on synthetic shortest path and fractional knapsack problems, demonstrating improved performance compared to the prior work.

1. Introduction

Applications in optimal transport and vehicle routing problem [20], financial modeling [13], power systems [6] [21], healthcare [3], circuit design [11], robotics [25] require a combination of prediction and optimization. For example, scheduling in hospitals leverages historical data to predict wait-times [5] that are used by a scheduling algorithm that solves an optimization problem. The standard pipeline in such applications is *predict-then-optimize* where the coefficients or inputs of an optimization problem (wait-times in the above example) are *first* predicted using a machine learning model. This is followed by running an appropriate optimization algorithm. However, recent work [12, 18, 29] has argued that predict-then-optimize can be sub-optimal in settings where the prediction model is imperfect and prone to errors. This is because the model errors are not *aligned* to the decisions (output of the optimization algorithm). In the above example, given a model with finite capacity, we want to estimate better wait-times that have a larger impact on the scheduling decisions. Consequently, the prediction or learning part of the pipeline needs to be aware of the decisions of the optimization algorithm. This has given rise to the *predict and optimize* (PO) framework, where the learning is fused with optimization.

Consequently, in this work, we are interested in training an end-to-end model from the data to the decisions (a schedule in the above example) without access to any intermediate prediction target (wait-times in the above example). Throughout, we will assume that we have access to a dataset that comprises N tuples of the form $\{(z_i, x_i^*)\}_{i=1}^N$ where z is the input to the prediction model and x^* is the ground-truth output of the optimization problem. Since numerous combinatorial problems, including shortest path, max-flow, and depth-first-search, can be cast as linear programs

(LPs), we will exclusively focus on cases where the optimization problem is an LP. In this case, the key challenge of the (PO) framework lies in the non-differentiable nature of LPs. This limitation precludes the use of auto-differentiation techniques. Several approaches have been proposed to alleviate this problem. We review these below and contrast them with our proposed method.

Constructing Surrogate Losses: Methods such as SPO+ [18], [22] construct surrogate losses to incorporate the decision error while training the prediction model. However, creating these surrogate losses requires the knowledge of intermediate prediction targets. In contrast, our work does not assume access to such ground-truth prediction targets.

Differentiating through LPs: Some methods [30] estimate the gradient "through" the LP by calculating the change in the decision by perturbing the prediction. However, these methods introduce additional hyper-parameters that are non-trivial to tune. A common technique to differentiate through LPs is to use the straight-through-estimator referred to as the identity method in [26]. Given a set of predictions from the model, the identity method uses the LP to estimate the decisions but does not consider the LP (treats the corresponding Jacobian as an identity matrix) while back-propagating the gradient from the decisions to the model parameters. Though successful in practice, this method is not theoretically principled. It can be shown that both these approaches fail to model the gradient through the LP accurately. The method in [9] computes expected gradients by perturbing the prediction target in different directions. While this method accurately models the gradient, it is not practically feasible because of the computational cost of solving LPs multiple times for each update to the model. One advantage of these techniques is their "black-box" nature, meaning that they only rely on the outputs from an LP, thus allowing the use of faster problem-specific solvers. In contrast, our work uses the specific properties of an LP, and this additional structure enables us to develop a computationally efficient and principled technique.

Using KKT conditions: The methods in [1, 2] focuses on (strongly)-convex optimization problems. It calculates the gradient through such problems by differentiating through its optimality (KKT) conditions. However, since the solutions of LPs are located at the corners of the feasible polytope, this method will yield zero gradients for LPs. To address this, Wilder et al. [32] add a quadratic regularization to the LP, thus relaxing the problem to a non-linear strongly-convex quadratic program (QP) and then use the technique in Amos and Kolter [2]. Similarly, [12] relax Mixed Integer Programs (MIP) by using a log-barrier regularization followed by the use of [2]. These approaches suffer from two notable limitations: (i) they introduce additional hyper-parameters (the regularization strength), and (ii) they are only guaranteed to converge in the vicinity of the optimal solution (because of the bias introduced by the regularization). While our proposed framework also uses KKT conditions, it ensures convergence to the optimal solution without introducing additional hyper-parameters.

Using the reduced cost optimality condition: The method proposed in [28] uses the reduced cost optimality conditions [23] for LPs. The method constructs a surrogate loss function that encourages the prediction to satisfy the reduced cost optimality conditions. The resulting method has theoretical convergence guarantees, assuming that the LPs are non-degenerate and that the model is expressive enough to perfectly fit (interpolate) the training data. Both of these are strong assumptions and are not necessarily satisfied in practice. In contrast, our proposed framework provides theoretical guarantees without relying on these assumptions.

Contributions: In Section 3.1, we reduce the PO with a linear model to a convex feasibility problem. This reduction enables us to use standard algorithms from the projection-onto-convex-sets (POCS) framework. In Section 3.2, we use alternating projections onto convex sets and in-

stantiate Algorithm 1 for PO with a linear model. The reduction to POCS guarantees convergence for Algorithm 1 without additional assumptions such as degeneracy or interpolation. Finally, in Section 4, we validate the effectiveness of our approach with experiments on synthetic shortest path and fractional knapsack problems. Our empirical results demonstrate that Algorithm 1 results in improved performance compared to the prior work.

2. Problem Formulation

In this section, we outline the problem setup and specify the challenge associated with using auto-differentiation. In the PO framework, the training dataset is $D = \{z_i, x_i^*\}_{i=1}^N$ where $z_i \in \mathbb{R}^d$ is the input, $Z \in \mathbb{R}^{d \times N}$ is the corresponding feature matrix and $x_i^* \in \mathbb{R}^m$ is the corresponding optimal decision given out by an LP. Without loss of generality, we assume the standard form of LP and define $\hat{x}(c)$ as the solution to the LP with cost-vector $c \in \mathbb{R}^m$,

$$\hat{x}(c) := \arg\min_x \langle c, x \rangle \text{ s.t } Ax = b,\ x \geq 0\,, \tag{1}$$

where $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$ are parameters of the LP. When using the model f_θ with parameters θ to predict the cost-vector, we define $\hat{x}(\hat{c})$ as:

$$\hat{x}(\hat{c}) = \arg\min_x \langle \hat{c}, x \rangle \text{ s.t } Ax = b,\ x \geq 0,\ \hat{c} = f_\theta(z)\,. \tag{2}$$

Given the dataset D, the PO problem is to find the parameters θ such that $\hat{x}(f_\theta(z_i)) \approx x_i^*$ for all $i \in [N]$. To gain some intuition as to why auto-differentiation [24] will not work in this setting, consider the squared loss $\ell(\theta) := \frac{1}{2}\sum_{i=1}^N \|\hat{x}_i - x_i^*\|^2$ for a fixed θ, where $\hat{x}_i := \hat{x}(f_\theta(z_i))$. Using the chain rule to compute the gradient with respect to θ, we get that,

$$\frac{\partial l}{\partial \theta} = \frac{\partial l}{\partial x}\frac{\partial x}{\partial c}\frac{\partial c}{\partial \theta} \tag{3}$$

Here, the first and the last term can be easily calculated. However, since the decision x is piece-wise constant with respect to c, the corresponding gradient is either 0 or undefined (does not exist). Next, we will develop an algorithm to solve the PO problem without calculating the gradient $\frac{\partial x}{\partial c}$.

3. Methodology

We present the reduction of the PO problem to the convex set feasibility problem and instantiate the resulting algorithm.

3.1. Reduction to Convex Feasibility Problem

For an input $(z, x^*) \in D$, we aim to find a c such that $\hat{x}(c) = x^*$. However, due to the non-uniqueness of the mapping from x to c, there are potentially infinitely many c values that can yield x^*. We define C to represent the set encompassing all such values of c. The set C can be represented by exploiting the optimality conditions for the LP. KKT conditions [19] gives necessary and sufficient conditions for the optimality of the LP. If x^* is the solution to the standard LP form, then the KKT conditions can be written as follows:

$$\nu^{*T} A + \lambda^* - c = 0, x^* \cdot \lambda^* = 0, Ax^* = b, \lambda^* \geq 0 \tag{4}$$

where $\lambda^* \in \mathbb{R}^m, \nu^* \in \mathbb{R}^{n\times 1}$ are the optimal dual variables and $x_i^*\lambda_i^* = 0$ (for all i) represents the complementary slackness condition, the equation $Ax^* = b$ arises from the definition of LP. At optimality, the tuple (x^*, λ^*, ν^*) must satisfy the Equation 4.

Since the KKT optimality conditions are both necessary and sufficient, given an optimal solution x^*, we can identify the set of cost vectors c that satisfy these conditions. This enables us to express C as a convex cone succinctly as,

$$C = \{c, \lambda, \nu \,|\nu^T A + \lambda - c = 0, x^* \cdot \lambda = 0, \lambda \geq 0\} \tag{5}$$

Note that we omit the condition $Ax^* = b$ as it is inherently satisfied for a feasible solution x^*.

We define F as the set of cost vectors that are realizable by the linear model parameterized by $\theta \in \mathbb{R}^{d\times m}$. Formally, F can be written as:

$$F = \{c \,|\exists \theta \text{ s.t. } c = \theta^T z\}. \tag{6}$$

Our aim is to find a $c \in C$ which is also realizable by the model, i.e. it lies in set F. Hence, we aim to find a c that lies in the intersection $(F \cap C)$. For a linear model, F is a convex set in c; hence, PO is equivalent to a convex feasibility problem in this setting.

3.2. Algorithm

The commonly employed method for solving convex feasibility problems is the alternating projections (POCS) algorithm [8] [31]. The POCS algorithm alternatively projects the point from one set to the other. The algorithm is guaranteed to converge to a point in the intersection if the intersection is non-empty; otherwise, it converges to the closest point between the two sets [14] [7]. For the POCS algorithm, we require the projection of an arbitrary point q onto the set C. This corresponds to solving the quadratic program (QP) as follows:

$$\begin{aligned} \mathcal{P}_C(q), \nu^*, \lambda^* = \arg\min_{c,\nu,\lambda} &||c - q||_2^2 + 0\lambda + 0\nu \\ \text{subject to} \qquad & \nu^T A - c + \lambda = 0, \lambda \cdot x^* = 0, \lambda \geq 0 \end{aligned} \tag{7}$$

Equation (7) returns a point $\mathcal{P}_C(q)$, the Euclidean projection of q onto C. For the projection of a point q onto the set F, we require solving the following regression problem,

$$\hat{\theta} := \arg\min_{\theta} \frac{1}{2} ||q - \theta^T z||^2 \quad ; \quad \mathcal{P}_F(q) = \hat{\theta}^T z \tag{8}$$

Hence, POCS consists of alternatively solving the optimization problems in Equations (7) and (8). In order to extend the above idea to N training points, we will define sets $C_i := \{c_i | \nu_i^T A - c_i + \lambda_i = 0, \lambda_i \geq 0, \lambda_i \cdot x_i^* = 0\}$ for each $i \in [N]$, and define $\mathcal{C} = \{(c_1, c_2.., c_N) | \forall i \in [N], c_i \in C_i\} = C_1 \times C_2 \ldots \times C_N$, where $(c_1, c_2.., c_N) \in \mathbb{R}^{Nm}$ is a concatenated vector of the cost-vectors. Similarly, we define $\mathcal{F} := \{(c_1, c_2.., c_N) | \exists \theta \text{ s.t. } \forall i \in [N], c_i = \theta^T z_i\}$. Hence, the projection onto $\mathcal{C}$ corresponds to solving a QP for every point $i \in [N]$. Projecting an arbitrary point $\tilde{q} \in \mathbb{R}^{Nm}$ onto $\mathcal{F}$ involves solving the regression problem, $\hat{\theta} := \arg\min_\theta \frac{1}{2}||\tilde{q} - \text{vec}(\theta^T Z)||^2$, where $\text{vec}(B) \in \mathbb{R}^{Nm}$ denotes the row-wise rasterization of a matrix $B \in \mathbb{R}^{N\times m}$. We note that both $\mathcal{C}$ and $\mathcal{F}$ are convex; hence, our theoretical guarantees hold for the resulting algorithm. Finally, we note that our algorithmic framework can handle a generic model f_θ, though our theoretical results only hold for

Algorithm 1: End-to-end learning for LP problem

Input: A, b, Training dataset $D \equiv (z_i, x_i^*)_{i=1}^N$;
Initialize θ_1;
for $t = 1, 2, .., T$ **do**
  $\hat{c}_i = f_{\theta_t}(z_i), \forall i \in [N]$;
  for $i = 1, 2, .., N$ **do**
    $q_i = \mathcal{P}_{C_i}(\hat{c}_i)$ by solving the optimization problem in Equation (7)
  end
  $\theta_{t+1} = \arg\min_\theta \frac{1}{2} \sum_{i=1}^N ||q_i - \theta^T z_i||^2$
end
Output: θ_{T+1}

a linear model. The complete algorithm for a generic model f_θ is described in Algorithm 1 and is referred to as `revgrad`. In addition to having convergence guarantees, our algorithmic framework offers a notable computational advantage: it permits multiple updates to θ for each call to the QP solver in Equation (7). This feature is especially important when solving the QP is computationally expensive.

3.3. Practical Considerations

To enhance the practical performance of our algorithm, we redefine the set $C = \{\nu^T A - c + \lambda = 0, \lambda_i \mathcal{I}\{x_i^* \neq 0\} = 0, \lambda_i \mathcal{I}\{x_i^* = 0\} \geq \epsilon\}$ to include a margin ϵ. There are three key motivations for adding the margin in the set C. Firstly, without margin, the algorithm can converge to a trivial solution $\hat{c} = 0$. Secondly, the algorithm will converge to the boundary of C, and the margin forces the solution to be in the interior of C. Finally, adding a margin improves the generalization performance [17]. Recently, Sun et al. [28] have also noted the benefits of adding a margin to enhance performance. In our approach, we adopt a similar margin formulation as [28] and introduce corresponding modifications to the KKT conditions, resulting in the modified set C above. In Appendix A, we prove that our margin formulation generalizes that in Sun et al. [28] to handle degenerate LPs. Throughout our experiments, we set $\epsilon = 1$.

4. Experiments

We conduct numerical experiments for two LP problems – the shortest path (SP) problem and the fractional Knapsack problems in [28]. These synthetic datasets consist of (z_i, c_i^*, x_i^*) pairs where $x_i^* = \hat{x}(c_i^*)$. We compare our proposed method (`revgrad`) against several existing methods, including identity [26], redcost [28], blackbox [30], optnet [2], SPO+ [10] on both tasks. For all the methods, we implement the LPs and QPs using the CVXPY library [15]. For LPs, we use the ECOS solver [16], and for QPs, we use the OSQP solver [27]. For further details regarding the datasets and hyperparameter settings for each method, refer to Appendix B. We note that SPO+ requires access to ground-truth cost-vector c^*; while other methods, including `revgrad`, do not assume access to these vectors. For each method, we plot the *estimate-loss* on both the train and test set, Estimate-Loss$(\theta) = \sum_{i=1}^N \langle c_i^*, \hat{x}(f_\theta(z_i))\rangle - \langle c_i^*, x_i^*\rangle$. Figure 1 demonstrates that `revgrad`

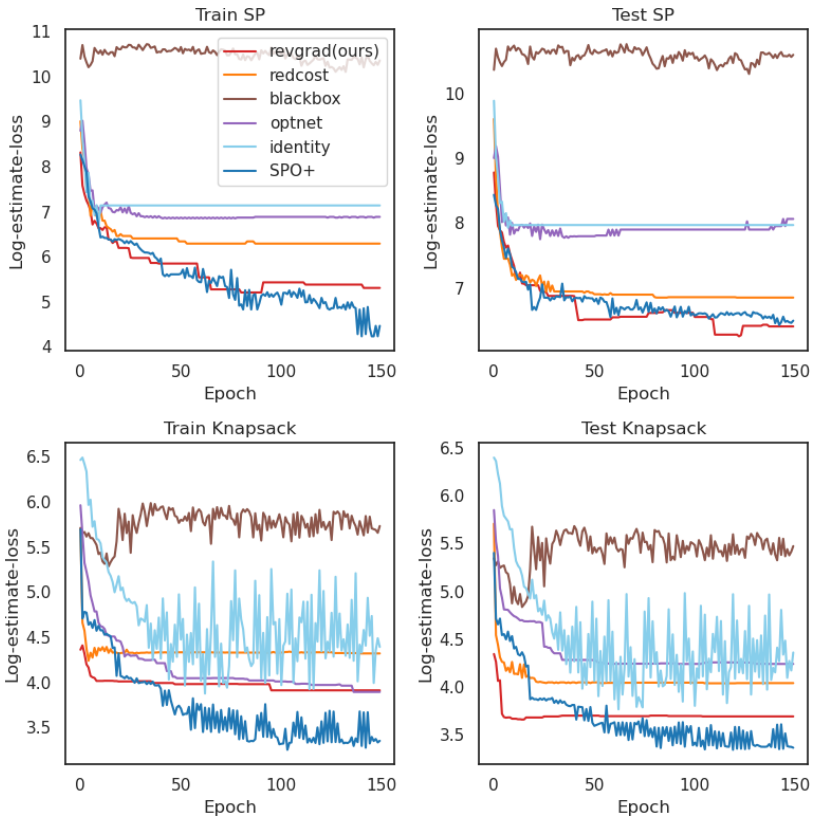

Figure 1: Training and Test plot for the Shortest Path and Fractional Knapsack tasks. For both problems, `revgrad` significantly outperforms the other methods (identity, optnet, blackbox, redcost) and is comparable to SPO+, that uses the knowledge of c^*.

significantly outperforms the other methods (identity, optnet, blackbox, redcost) and is comparable to SPO+ that uses the knowledge of c^*.

5. Discussion

We presented a reduction of the PO problem to convex feasibility, which guarantees theoretical convergence for linear models. We empirically compared the resulting algorithm on the Shortest Path and Fractional Knapsack tasks, demonstrating superior performance compared to the baselines. Future directions include (a) benchmarking our algorithm for non-convex real-world problems and (b) extending the algorithm to handle the stochastic setting.

References

[1] Brandon Amos. Differentiable optimization-based modeling for machine learning. *Ph. D. thesis*, 2019.

[2] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[3] Mallik Angalakudati, Siddharth Balwani, Jorge Calzada, Bikram Chatterjee, Georgia Perakis, Nicolas Raad, and Joline Uichanco. Business analytics for flexible resource allocation under random emergencies. *Management Science*, 60:1552–1573, 2014.

[4] Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1 – 3, 1966.

[5] Gah-Yi Ban and Cynthia Rudin. The big data newsvendor: Practical insights from machine learning. *Operations Research*, 67(1):90–108, 2019.

[6] RC Bansal. Optimization methods for electric power systems: An overview. *International Journal of Emerging Electric Power Systems*, 2, 2005.

[7] Heinz H Bauschke and Jonathan M Borwein. On the convergence of von neumann's alternating projection algorithm for two sets. *Set-Valued Analysis*, 1:185–212, 1993.

[8] Heinz H Bauschke and Jonathan M Borwein. On projection algorithms for solving convex feasibility problems. *SIAM review*, 38(3):367–426, 1996.

[9] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

[10] Dimitris Bertsimas and Nathan Kallus. From predictive to prescriptive analytics. *Management Science*, 66:1025–1044, 2020.

[11] Stephen P Boyd, Thomas H Lee, et al. Optimal design of a cmos op-amp via geometric programming. *IEEE Transactions on Computer-aided design of integrated circuits and systems*, 20:1–21, 2001.

[12] Chris Cameron, Jason Hartford, Taylor Lundy, and Kevin Leyton-Brown. The perils of learning before optimizing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3708–3715, 2022.

[13] Gerard Cornuejols and Reha Tütüncü. *Optimization methods in finance*, volume 5. Cambridge University Press, 2006.

[14] Frank Deutsch. Rate of convergence of the method of alternating projections. *Parametric optimization and approximation (Oberwolfach, 1983)*, 72:96–107, 1984.

[15] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17:2909–2913, 2016.

[16] Alexander Domahidi, Eric Chu, and Stephen Boyd. Ecos: An socp solver for embedded systems. In *2013 European control conference (ECC)*, pages 3071–3076. IEEE, 2013.

[17] Othman El Balghiti, Adam N Elmachtoub, Paul Grigas, and Ambuj Tewari. Generalization bounds in the predict-then-optimize framework. *Advances in neural information processing systems*, 32, 2019.

[18] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

[19] Harold W Kuhn and Albert W Tucker. Nonlinear programming, paper presented at proceedings of the second berkeley symposium on mathematical statistics and probability, 1951.

[20] Bingjie Li, Guohua Wu, Yongming He, Mingfeng Fan, and Witold Pedrycz. An overview and experimental study of learning-based optimization algorithms for the vehicle routing problem. *IEEE/CAA Journal of Automatica Sinica*, 9:1115–1138, 2022.

[21] Jia Li, Feng Liu, Zhaojian Wang, Steven H Low, and Shengwei Mei. Optimal power flow in stand-alone dc microgrids. *IEEE Transactions on Power Systems*, 33:5496–5506, 2018.

[22] Mo Liu, Paul Grigas, Heyuan Liu, and Zuo-Jun Max Shen. Active learning in the predict-then-optimize framework: A margin-based approach. *arXiv preprint arXiv:2305.06584*, 2023.

[23] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*, volume 2. Springer, 1984.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[25] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal path planning of mobile robots: A review. *International journal of physical sciences*, 7:1314–1320, 2012.

[26] Subham Sekhar Sahoo, Anselm Paulus, Marin Vlastelica, Vít Musil, Volodymyr Kuleshov, and Georg Martius. Backpropagation through combinatorial algorithms: Identity with projection works. *arXiv preprint arXiv:2205.15213*, 2022.

[27] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[28] Chunlin Sun, Shang Liu, and Xiaocheng Li. Maximum optimality margin: A unified approach for contextual linear programming and inverse linear programming. *arXiv preprint arXiv:2301.11260*, 2023.

[29] Toon Vanderschueren, Tim Verdonck, Bart Baesens, and Wouter Verbeke. Predict-then-optimize or predict-and-optimize? an empirical evaluation of cost-sensitive learning strategies. *Information Sciences*, 594:400–415, 2022.

[30] Marin Vlastelica, Anselm Paulus, Vít Musil, Georg Martius, and Michal Rolínek. Differentiation of blackbox combinatorial solvers. *arXiv preprint arXiv:1912.02175*, 2019.

[31] John Von Neumann. On rings of operators. reduction theory. *Annals of Mathematics*, pages 401–485, 1949.

[32] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1658–1665, 2019.

Supplementary Material

Organization of the Appendix

Appendix A. Equivalence between the KKT formulation and [28]

In this section, we derive the equivalent margin for the KKT formulation as in Sun et al. [28].

From KKT conditions, we have $\nu^T A - c + \lambda = 0$. Consider B as the set of indices in basis, and N represents otherwise. Now, separating the above equation in terms of these two sets, we get:

$$\nu^T A_B - c_B + \lambda_B = 0 \tag{9}$$
$$\nu^T A_N - c_N + \lambda_N = 0 \tag{10}$$

where A_B, A_N are the the corresponding columns of matrix A defined by set B and N respectively. Moreover, matrix A_B is an invertible square matrix.

Now, from the complementary slackness conditions, we have $\lambda \cdot x^* = 0$. Considering non-degenerate case for equivalence with [28], we have $x_i > 0(\ \forall i \in B)$. This implies that $\lambda_B = 0$. Substituting this value in Equation (9), we get:

$$\nu^T = c_B(A_B)^{-1} \tag{11}$$
$$\lambda_N = c_N - \nu^T A_N \tag{12}$$
$$\lambda_N = c_N - c_B(A_B)^{-1} A_N \tag{13}$$

Furthermore, the condition $\lambda_N \geq 0$ implies that $c_N - c_B(A_B)^{-1}A_N \geq 0$. Thus, we retrieve the reduced cost optimality condition using the KKT conditions. Therefore, the equivalent of $c_N - c_B(A_B)^{-1}A_N \geq \epsilon$ to the KKT formulation would be to impose the same constraint on λ_N, i.e. $\lambda_N \geq \epsilon$.

To see the extension to the degenerate case, we can apply similar steps to derive the margin conditions. Consider a degenerate case in which $\exists i \in B$ such that $x_i = 0$. In this case, $\lambda_i(\forall i \in \{B\})$ is not necessarily 0. To solve this issue, consider a new set of basis B_d, N_d such that $x_i \mathcal{I}\{i \in B_d\} > 0, x_i \mathcal{I}\{i \in N_d\} = 0$. Substituting these values into Equation (11), we obtain:

$$\nu^T = c_{B_d}(A^{\dagger}_{B_d}) \tag{14}$$
$$\lambda_{N_d} = c_{N_d} - \nu^T A_{N_d} \tag{15}$$
$$\lambda_{N_d} = c_N - c_B(A^{\dagger}_{B_d}) A_{N_d} \tag{16}$$

Here, $A^{\dagger}_{B_d}$ is the pseudo-inverse of the matrix A_{B_d} as matrix A_{B_d} is not a invertible.

To retrieve margin, we can impose the same constraints $\lambda_{N_d} \geq \epsilon$. This approach also resolves the challenge of computing suitable B and N for the degenerate case, as there are exponentially many realizations of B and N, and the performance varies based on a particular choice.

We incorporate this modification in Equation (7) to introduce a margin in the set C. Notably, this formulation eliminates the requirement for special treatment for degenerate/non-degenerate cases.

Appendix B. Dataset and Experimental Setup Details

B.1. Dataset Details

We generated $N = 100$ samples for each task's training and test sets. We used the codebase provided by the [28] to generate the dataset.

The Shortest Path problem is defined on a 5×5 grid with $m = 40$ directed edges associated with the ground truth cost-vector $c^* \in \mathbb{R}^m$. Input $z \in \mathbb{R}^d$ with $d = 6$. Thus, $\theta \in \mathbb{R}^{d \times m}$. To make the problem harder, we use the $degree = 4$ in the data-generation process. We experimented with varying degrees and found the trend to be similar in all the cases.

The Fractional Knapsack problem is defined with input $z \in \mathbb{R}^d$ with $d = 5$. We have 10 items with associated cost-vectors, and slack variables are added to convert the problem to standard form, making the dimension $m = 21$. Thus, $\theta \in \mathbb{R}^{d \times m}$. To make the problem harder, we use the $degree = 2$ in the data-generation process with the attacking noise of $attack - power = 3.0$.

Refer to the Appendix in [28] for more details regarding the data-generation process.

B.2. Hyper-parameters for methods

In the `revgrad`, we solve the regression problem using closed-form solutions obtained through matrix inversion. For the redcost method, we employ the Armijo line search algorithm [4] with Gradient Descent. The identity method utilizes the Armijo line search for the Shortest Path task and a grid search for the Knapsack task. In the blackbox, optnet and SPO+ method, grid search is used to find the best learning rate for both tasks. For grid-search, we choose constant $lr \in \{10, 1, 0.1, 0.01, 0.001\}$, which gives the best performance.

Notably, we did not conduct a grid search for hyper-parameters, such as the regularizer in optnet and the perturbation weight λ in blackbox.

It's important to highlight that redcost for the Shortest Path task is trained with knowledge of c^* since it employs B^*, N^* derived from c^*. Note that the LP in the Shortest Path task is degenerate, and the optimal B^* is not unique. Utilizing other values of B^*, N^* led to suboptimal performance in our experiments.