MIPaaL: Mixed Integer Program as a Layer

Aaron Ferber,[1] **Bryan Wilder,**[2] **Bistra Dilkina,**[1*] **Milind Tambe**[2]
[1]University of Southern California, [2]Harvard John A. Paulson School of Engineering and Applied Sciences
{aferber, dilkina}@usc.edu, bwilder@g.harvard.edu, milind_tambe@harvard.edu

Abstract

Machine learning components commonly appear in larger decision-making pipelines; however, the model training process typically focuses only on a loss that measures average accuracy between predicted values and ground truth values. Decision-focused learning explicitly integrates the downstream decision problem when training the predictive model, in order to optimize the quality of decisions induced by the predictions. It has been successfully applied to several limited combinatorial problem classes, such as those that can be expressed as linear programs (LP), and submodular optimization. However, these previous applications have uniformly focused on problems with simple constraints. Here, we enable decision-focused learning for the broad class of problems that can be encoded as a mixed integer linear program (MIP), hence supporting arbitrary linear constraints over discrete and continuous variables. We show how to differentiate through a MIP by employing a cutting planes solution approach, an algorithm that iteratively tightens the continuous relaxation by adding constraints removing fractional solutions. We evaluate our new end-to-end approach on several real world domains and show that it outperforms the standard two phase approaches that treat prediction and optimization separately, as well as a baseline approach of simply applying decision-focused learning to the LP relaxation of the MIP. Lastly, we demonstrate generalization performance in several transfer learning tasks.

Introduction

We propose a method of *training predictive models to directly optimize the quality of decisions that are made based on the model's predictions*. We are particularly interested in decision-making problems that take the form of *mixed integer programs (MIPs)* because they arise in settings as diverse as electrical grid load control (Mohsenian-Rad and Leon-Garcia 2010), RNA string prediction (Sato et al. 2011), and many other industrial applications (Nemhauser 2013). MIPs naturally arise in so many settings largely due to their flexibility, computational complexity (ability to capture NP-hard problems), and interpretability. In many practical situations it is often necessary to predict some component (e.g., the objective) of the MIP based on historical data, such as estimated demand (O'Mahony and Shmoys 2015), price forecasts (Demirovic et al. 2019), or patient readmission rate (Chan et al. 2012). Alternatively, practitioners may use a MIP to enforce that the outputs of the predictions meet semantically meaningful objectives such as ensuring predictions result in making fair decisions downstream (Benabbou et al. 2018; Trilling, Guinet, and Le Magny 2006; Warner 1976). In these settings with a *prediction* and *optimization* component, the predictive models are often trained without regard for the downstream optimization problem. For example, the mean squared error will consider errors for different examples to have the same importance. However, in practice it may be more important to distinguish between some values rather than others, meaning average error metrics may ignore impactful distinctions. Consider predicting item costs for a unit-weight knapsack setting, where we try to select 5 items minimizing total cost. Training a model to minimize mean squared error will concentrate model capacity on getting all values correct on average, and in fact multiplicative errors for large values will dominate the loss function. However, in this setting, we would rather dedicate capacity to correctly predicting cost for items which are potentially in the cheapest set of items over capturing the nuances between items that are unlikely to be in our set. As such, good average *predictions* may not lead to high quality *decisions*.

Decision-focused learning, introduced for a financial criterion in (Bengio 1997) and extended to the more general quadratic programs in (Amos and Kolter 2017), and linear programs in (Wilder, Dilkina, and Tambe 2019), explicitly and automatically integrates the downstream decision problem when training the predictive model, in order to optimize the quality of decisions induced by the predictive model. In the commonly used gradient-based predictive models, the central challenge is in passing gradients back to give the predictive model an indication of how it should shift its weights in order to improve decision quality of the resulting optimal solution. The *discrete* and *discontinuous* solution space that makes the MIP so widely applicable also prevents us from easily differentiating through it, as has been done for embedding continuous optimization problems in neural networks (Amos and Kolter 2017; Wilder, Dilkina, and Tambe 2019; de Avila Belbute-Peres et al. 2018). Our approach for computing gradients relies on the fact that we can *algorithmically generate a continuous surrogate* for the original discrete op-

*Corresponding author

timization problem. We employ previous work on cutting plane approaches, which can tighten a MIP relaxation by iteratively solving a continuous relaxation and cutting off the discovered integer infeasible solution until an integer feasible solution is found (Gomory 1960). The final continuous, and convex, optimization problem can then be used for backpropagation by differentiating the KKT conditions (Karush 1939) of the continuous surrogate, as has been suggested for convex problems (Amos and Kolter 2017). While pure cutting plane approaches are often slower than alternate branch-and-bound MIP solvers in practice (Dash et al. 2014), we note that our approach only needs the cutting plane methodology for backpropagation during training. Indeed, at test time, we can make predictions and find optimal decisions based on those predictions using any state-of-the-art MIP solver, ensuring the running time in deployment is exactly the same were any other training method used. Due to the computational complexity of backpropagating through large optimization problems, we analyze approaches that stop cut generation after a fixed number of cuts have been generated, trading off the tightness of the differentiable solver with improved training runtime. Finally, we compare against a decoupled learning and optimization approach, that simply relies on training using a relevant classification or regression loss function.

We demonstrate the effectiveness of our approach on three different real-world MIP problem domains concerning investment portfolio optimization, bipartite matching with diversity constraints, and energy-related knapsack, showing significant improvements in solution quality over the baseline. We then evaluate our method's ability to generalize to unseen tasks with differing problem sizes, and data distribution.

Problem description

We consider problems that combine learning with an optimization problem that can be modeled as a mixed integer program (MIP). Specifically, each instance of the problem is a triple (ϕ, c, D), where ϕ is a feature vector, c is the objective coefficient vector of a MIP, and D represents additional known data that plays a role in the downstream optimization. In a MIP for a given instance, D will include the constraint coefficients, right-hand-side constants, and set of integral variables in each train instance $A, b, \mathcal{I}$, where the constraints encode data about individual instances and can vary from one problem instance to another. If c were known a priori, we could simply use branch and bound to solve the corresponding MIP: $\min_x\{c^T x | Ax \leq b,\ \forall i \in \mathcal{I}, x_i \in \mathbb{Z}\}$; however, we consider the setting where c is unknown and must be estimated from ϕ. We assume that we observe training instances $\{(\phi_1, c_1, D_1), \ldots, (\phi_m, c_m, D_m)\}$ drawn from some distribution , and we need to make decisions at test time using only features ϕ and known parameters D. We will use the training data to train a predictive model f_θ (where θ denotes the internal parameters of the model) which outputs an estimate $f_\theta(\phi) = \hat{c}$ on a test-time instance. Once we have estimates $\hat{c}$, an optimal solution to the estimated problem $x^*(\hat{c}; A, b, \mathcal{I})$ can be obtained. The quality of this solution with respect to the real coefficients c we observe after the fact is then $c^T x^*(\hat{c}; A, b, \mathcal{I})$. The standard two stage approach in this setting is to train the machine learning model f_θ that minimizes a loss that is generally an average distance between predicted values $\hat{c}$ and ground truth values c. However, our overall objective is to find model parameters θ which yield predictions $\hat{c}$ that directly maximize the quality of MIP solution for $\hat{c}$, *evaluated with respect to the (unknown) ground truth objective* c.

MIPaaL: Encoding MIP in a Neural Network

We formulate the MIP as a differentiable layer in a neural network which takes objective coefficients $\hat{c}$ as input and outputs the optimal MIP solution. Formally, we consider the optimal solution $x^*(\hat{c}; A, b, \mathcal{I})$ of the MIP as a function of the input coefficients $\hat{c}$ given linear constraints on the feasible region $Ax \leq b$ and the set of integral variables $\mathcal{I}$. We write a functional form of the layer as:

$$x^*(\hat{c}; A, b, \mathcal{I}) = \begin{array}{ll} \text{argmin}_x & \hat{c}^T x \\ \text{subject to} & Ax \leq b \\ & x_i \in \mathbb{Z}\, \forall i \in \mathcal{I} \end{array} \quad (1)$$

We can perform a forward pass given input objective coefficients, which are potentially outputs of a neural network, and feasibility parameters of the MIP using any solver.

Standard practice in this setting is to first train a model f_θ to predict the coefficients based on embeddings ϕ_i of the different predicted components such that on average the model predictions $\hat{c} = f_\theta(\phi)$ are not far away from the ground truth objective coefficients c. Then, decisions are made during deployment based on the predicted values by finding the optimal solution with respect to the predicted values $x^*(\hat{c}; A, b, \mathcal{I})$ based on the formulation in Equation 1.

We introduce an alternative approach that incorporates the above layer into the training pipeline, instead of only calling the forward pass at deployment. To do so, we need to provide method to compute the forward and backward passes. While forward propagation in this setting is straightforward using standard MIP solvers, the highly nonconvex and discrete structure of the MIP, which enable its flexibility, seem to render gradient computation untenable. Indeed, the optimization problem (1) as written is nondifferentiable since even infinitesimal changes to $\hat{c}$ can drastically change the optimal solution. Previous work has explored differentiation through optimization, but largely for problem classes which are significantly smoother. Amos and Kolter (2017) show how to differentiate through convex quadratic problems, which have only continuous variables and whose solutions are natively differentiable. Wilder et al. (2019) build on this work to tackle linear programs, which may have discontinuous solutions (but lack integer variables), via the addition of quadratic smoothing. However, both of these problem classes lack the fundamental representational power of MIPs, which are NP-hard to solve and hence often employed as a general-purpose engine for hard computational problems (while QPs and LPs are easily solved in polynomial time). This power comes at a price: MIPs are fundamentally connected to discrete, nonconvex structures via the integer variables $\mathcal{I}$, which do not easily admit a differentiable surrogate.

Our solution to this dilemma draws on algorithmic techniques for computing a new continuous relaxation *for each*

individual $\hat{c}$ which is much closer to the discrete problem than if we naively relaxed the integer variables. We then use the continuous-domain techniques developed previously to differentiate through this relaxation during training time. In particular, we use a pure cutting plane approach which tightens the LP relaxation (at the expense of potentially generating a large number of cutting planes along the way) (Bodur, Dash, and Günlük 2017; Wolsey and Nemhauser 2014).

A pure cutting plane approach iteratively solves the linear programming relaxation of the current problem. If the found solution is integral then the algorithm terminates since the found solution is both feasible to the original MIP and optimal for a relaxation of the original problem. Otherwise, a cut is generated which removes the discovered fractional solution and leaves all feasible integral solutions. Since the individual cuts do not remove any integral solutions, the final LP retains all integral solutions. Ideally, we would describe the convex hull of the integral solutions yielding an exponentially large linear program equivalent to the original MIP, thus ensuring that all potentially optimal integral solutions lie on extreme points of the feasible region. In practice, finding the convex hull of MIP solutions is not just NP-Hard but often numerically intractable. Instead, we can obtain cutting planes that tighten the feasible region via Gomory cuts or other globally valid cuts (Gomory 1960; Balas et al. 1996). We then consider the problem with generated cuts $Sx \leq t$, and write out the following linear program:

$$\begin{array}{ll} \text{minimize}_x & \hat{c}^T x \\ \text{subject to} & Ax \leq b \\ & Sx \leq t \end{array} \tag{2}$$

Given this continuous optimization problem, we can now find the gradient of the optimal solution with respect to the input parameters by differentiating through the KKT conditions, optimality conditions for Equation 2. Differentiation through the continuous LP is done via the quadratic smoothing approach proposed in (Wilder, Dilkina, and Tambe 2019) for linear programs based on differentiating the KKT conditions of a quadratic program as shown by Amos and Kolter (2017).

In our setting we can compute $\frac{dx}{d\theta}$ using the gradient backpropagated to the MIP layer: $\frac{d\nabla_x f(x,\theta)}{d\theta}$, along with the optimal primal solution $\hat{x}$, and dual solution λ, μ corresponding to constraints $Ax \leq b$ and $Sx \leq t$ respectively. Using the quadratic smoothing term $\gamma\|x\|_2^2$ in the objective proposed by Wilder et al. (2019), we can find the gradient of the solution x with respect to the parameters θ by solving the following system of equations for $\frac{dx}{d\theta}$:

$$\begin{bmatrix} -2\gamma & A^T & S^T \\ D(\lambda)A & D(A\hat{x}-b) & 0 \\ D(\mu)S & 0 & D(S\hat{x}-t) \end{bmatrix} \begin{bmatrix} \frac{dx}{d\theta} \\ \frac{d\lambda}{d\theta} \\ \frac{d\mu}{d\theta} \end{bmatrix} = \begin{bmatrix} \frac{d\nabla_x f(x,\theta)}{d\theta} \\ 0 \\ 0 \end{bmatrix},$$

with $D(\cdot)$ being a shorthand for $\text{diag}(\cdot)$.

Decision-Focused Learning with MIPaaL

Having developed a differentiable layer to solve MIPs, we can incorporate this layer into the broader pipeline of a machine learning problem in order to train end-to-end for high test-time optimization performance. Specifically, we define the loss function to be the solution quality of the predicted optimal decision $x^*(\hat{c}, A, b, \mathcal{I})$ output by the model (which now includes the MIPaaL layer) with respect to the ground truth objective coefficients c. In other words, we can use the MIPaaL formulation to train the model to directly minimize the deployment objective. We instantiate this method using a neural network parameterized by θ where f_θ predicts the objective coefficients $\hat{c}$ based on embeddings ϕ of the decision variables. The foward pass, shown in Equation 3, calls f_θ to predict the objective coefficients, and then uses the MIPaaL cutting plane solver to generate the LP corresponding to the original MIP, along with the resulting decision x^*. We then set the loss to be the solution quality of the returned decision with respect to the ground truth coefficients c.

$$\hat{c} := f_\theta(\phi) \tag{3a}$$

$$\hat{x} := x^*(\hat{c}; A, b, \mathcal{I}) \qquad \text{via Equation 1} \tag{3b}$$

$$loss(\hat{c}, c) := c^T \hat{x} \tag{3c}$$

Since the dot product in Equation 3c and prediction of the neural network in Equation 3a are differentiable functions of their inputs, the parameters of the neural network predictor θ can be trained via backpropagation using KKT conditions of the computed surrogate LP found in Equation 2, relying on a small quadratic regularization term for the LP proposed in Wilder et al. (2019) to enforce strong convexity and perform backpropagation through Equation 3b.

MIPaaL Variants

MIPaaL - k cuts: Given that the cut generation process is time consuming and must be done for each forward pass, we examine the tradeoff of decision quality when stopping cut generation after the first k cuts have been generated. We experiment with two settings ($k = 100$ and $k = 1000$) to determine how the tightness of the cut generation process impacts the decision quality at test time. Note that in the case where $k = 0$ cuts are generated, this method is equivalent to just using the LP relaxation of the original MIP where all integrality constraints on decision variables are dropped.

MIPaaL-Warm: Warm starting the training process with a pretrained network may enable MIPaaL to take advantage of already learned representations. The predictive model feeding into MIPaaL can be initialized with any similarly parametrized model, as a result, we can initialize the predictive input of MIPaaL with a model trained to predict objective coefficients by minimizing a standard ML loss. The resulting model may yield better decision quality than the initialization and retain good predictive performance.

MIPaaL-Hybrid: We consider a hybrid loss balancing MIP decision quality and predictive performance. The hybrid approach is intended to simultaneously improve decision quality and minimize a standard ML loss, like cross-entropy or mean squared error, $\text{loss}_{ML}(\hat{c}, c)$ between neural network predictions $\tilde{c}$ and ground truth coefficients c. We define the hybrid loss as a weighted sum of MIP-based loss and ML loss $c^T x^*(\hat{c}, A, B, \mathcal{I}) + \alpha \text{loss}_{ML}(\hat{c}, c)$ with weight α.

Empirical Evaluation

We instantiate MIPaaL for a range of tasks which require prediction followed by optimization with the overall goal of improving the objective value upon deployment. Specifically, we run experiments on *combinatorial portfolio optimization*, *diverse bipartite matching* and *knapsack* instances. The portfolio optimization setting accounts for various combinatorial constraints enforcing small number of assets in the portfolio and limiting rebalancing transactions to maximize the overall predicted return of the portfolio. Diverse bipartite matching enforces diversity in the types of recommended matches to maximize an overall predicted utility of the suggested pairing. Knapsack instances select times to sell energy given a limit on when energy can be sold. In each setting, the predictive problem is nontrivial since the features do not contain much signal for predicting objective coefficients. However, we demonstrate that in these settings, we can train a predictive model whose outputted objective coefficients yield high-quality decisions after optimization. Experimental details and extended results are in the appendix[1].

Benchmark problems

Portfolio Optimization has been heavily studied in financial settings. Given a set of assets to choose from and predicted price changes for the next period, the objective is to invest money among assets to maximize return, while limiting risk from uncertainty. Practitioners may also want to limit overexposure to any single financial sector or limit order quantity as in Bertsimas et al. (1999). In this setting, the prediction problem of forecasting the next period's returns is generally a difficult learning task. To tackle this, (Bengio 1997) proposes training a model to maximize the objective value obtained by a convex portfolio optimization problem given by (Markowitz 1952). We approach the problem considering a MIP criterion based on the formulation given by Bertsimas et al. (1999).

In the combined prediction and optimization problem, the next period's returns (i.e., the objective) are unknown and predicted from historical prices and features from Quandl's WIKI and FSE datasets (Quandl 2019). We evaluate on the SP500, a collection of the 505 largest companies representing the American market, and the DAX, a set of 30 large German companies. We split the data temporally, training, validating, and testing on data from periods Jan '05-Dec '10, Jan '11-Nov '13, and Dec '13-Nov '16 respectively. Details regarding data collection are in the appendix[1].

Diverse Bipartite Matching: Bipartite matching is used in many applications, where the success probability of a particular match is often estimated from data. Without additional constraints, bipartite matching can be formulated as an LP; a setting previously used for decision-focused learning (Wilder, Dilkina, and Tambe 2019) since the LP relaxation gives exact integral solutions. In practice though, matching problems are often subject to additional constraints. For instance, finding fair and high-quality housing allocations or kidney matching with additional contingency plans (Benabbou et al. 2018; Dickerson et al. 2016) require additional modeling which make the integer problem NP-Hard.

[1]https://tinyurl.com/aaai2020-mipaal

We use the problem of bipartite matching with added diversity constraints, which enforce a maximum and minimum bound on the percent of edges selected with a specified property. We use the experimental setup of (Wilder, Dilkina, and Tambe 2019), who did not include diversity constraints. Specifically, the matching instances are constructed from the CORA citation network (Sen et al. 2008) by partitioning the graph into 27 instances on disjoint sets of nodes (split into train, test and validation). Diversity constraints are added to enforce that at least some percent of edges selected are between papers in the same field and some percent are between papers in different fields.

Knapsack concerns choosing from a set of items each with a value and a weight. The objective is to maximize the total value of items selected, while not exceeding a budget on total weight. We consider a setting where the item values are not known at the time of decision-making, and must be predicted from data. Our knapsack dataset corresponds to unit commitment problems. In those problems we want to plan when to sell energy at half-hour intervals over a day, while limiting operating costs. We want to maximize the revenue we get but we need to predict half-hour prices. We use real-world energy-aware scheduling data from (Demirovic et al. 2019) which consist of historical energy data and prices from the ICON energy-aware scheduling competition. The time period weights are drawn uniformly between 0 and 1, and the budget is 10% of the total weight. In (Demirovic et al. 2019), the authors describe and analyze an algorithm for exhaustively searching linear models which yield good knapsack performance; however, the results are for linear predictive components feeding to knapsack instances, and do not directly extend to neural network models.

Methods

MIPaaL: Following our specified methodology, we use CPLEX's implementation of Gomory cuts (Gomory 1960) which we collect through the C API. Since we are only interested in the global cuts generated, we limit solving to the root node, disable heuristics, and all cut generation procedures other than Gomory cuts. Termination occurs when either an integral solution is found, in which case our resulting LP gives us the exact integral optimal solution, or when CPLEX stops generating viable cuts (since the cut generation based purely on Gomory cuts is not complete).

Two-Stage: We compare against the standard predict-then-optimize approach which treats prediction and optimization components separately. The predictive component is trained to minimize a standard loss between predicted objective coefficients and the ground truth (e.g., mean squared error or cross-entropy). Afterwards, we solve the MIP to optimality using the predicted coefficients.

RootLP: Next, we compare against a decision-focused learning approach that uses only the initial LP relaxation of the MIP (disregarding integrality constraints), and hence can be solved using the method proposed by (Wilder, Dilkina, and Tambe 2019) for linear programs. While the predictive model is trained using the LP relaxation, at test time we solve the true MIP using the predicted objective coefficients to obtain an integral decision. This tests the impact of our cutting plane

method, which allows us to fully account for combinatorial constraints in MIPaaL's gradients.

Setup: We average over 5 training and testing iterations per problem setting with different seeds to evaluate the given approaches. This results in 180 portfolio optimization, 55 diverse bipartite matching, and 95 weighted knapsack instances to compute testing metrics. The neural network architectures are fixed for a given problem distribution, and are selected from grid search using MIP decision quality on the validation set of the *TwoStage model*. The models are trained with Adam (Kingma and Ba 2014) with learning rate 0.01 and l2 normalization coefficient of 0.01. Details on model architectures and evaluation setup can be found in the appendix[1].

Experimental Results

We evaluate the realized decision quality and predictive performance on test instances. We provide 95% confidence halfwidths around the mean evaluation. Unless otherwise indicated, bolded entries are unbeaten by other methods using one-sided paired t-tests with significance level of 0.05 to indicate whether we can reject the hypothesis that observed improvement is due to random chance.

Decision quality: The decision quality of a given model's outputs is determined by the realized objective value of solutions obtained by using the model's predictions. A trained ML model predicts the objective coefficients of the particular decision problem, and we use an exact Branch-and-Bound solver to compute an optimal solution $\hat{x}$ with respect to the predicted objective coefficients $\hat{c}$. The decision quality is the the objective value of that solution under the ground truth objective coefficients, $c^T\hat{x}$. For portfolio optimization, the decision quality is the monthly rate of return so the 2.79% that MIPaaL achieves in Table 1 means that MIPaaL increases the portfolio value by 2.79% on average. In bipartite matching, our goal is to maximize the number of successful matches. In knapsack, we maximize the total value of selected items which represents gross income from an energy schedule.

We compare MIPaaL against TwoStage and RootLP, as well as variants of MIPaaL discussed above. Our results in Table 1 show that MIPaaL significantly outperforms both baselines in decision quality across all four benchmarks. As shown in Table 1, the full MIPaaL results in *more than 2x the average return of the TwoStage or RootLP methods* for portfolio optimization and *40.3% more successful matches* for biparite matching, and *1% more gross revenue* for knapsack instances. We note that a randomly initialized network achieves around 501 objective value for knapsack. These results drive home the importance of integrating the full combinatorial problem into training, as enabled by MIPaaL.

We find that warm starting or using a hybrid loss of decision quality and ML loss degrade performance below RootLP. Limiting the number of cuts used in MIPaaL to 100 also results in degraded performance in three of the benchmarks, but with 1000 cuts, the model outperforms both baselines and is competitive with the full MIPaaL.

For the full MIPaaL due to the incompleteness of our cutting plane generator, solving the resulting cutting plane LP might not produce an integer optimal solution to the original MIP (as is also the case with the k cut limited versions). Hence, the LP solutions and the corresponding gradients we obtain during the backward pass are approximate some of the time. During the full MIPaaL training, cutting plane LPs of training instances produce integer optimal solutions in 83% of SP500, 94% of DAX, 65% of Matching, and 97% of Knapsack instances. On average, the cutting plane LP solutions during MIPaaL training matched the 0/1 value in the integer optimal solution for 89.20, 70.40, 88.35 and 97.13% of the binary variables for SP500, DAX, Matching and Knapsack respectively (See complete set of statistics in appendix[1]). However, the decision quality improvements of MIPaaL (using only the standard Gomory cut generation procedure) clearly show that the training is indeed effective and results in a final model that is better than both TwoStage as well as the k-cut limited forms of MIPaaL. The bad decision quality performance of MIPaaL-100 and RootLP highlights the fact that MIPaaL needs an effective cut generating procedure to succeed. For any given MIP domain, using a more comprehensive set of cutting plane generating procedures will improve the tightness of the resulting cutting-plane LP to the original MIP instance. In our results, we observed that using the full MIPaaL version over the version constrained to $k = 100, 1000$ always gave us an improvement in decision quality (Table 1), while not significantly increasing training time (See Appendix[1]) for a full set of statistics on LP integrality and timing results across different MIPaaL variants).

ML performance: We show the predictive performance in Table 2. For portfolio optimization and knapsack (which are regression problems) we report mean squared error (MSE) and correlation coefficient, while for bipartite matching (a classification problem) we report cross-entropy and AUC. SP500 Asset returns have mean value 0.055 and stdev 0.199, and in DAX have mean -0.9 and stdev 3.94, so the observed MSE are large compared to the values themselves. In knapsack instances, the values have mean 64 and stdev 35, with the MSE again being very large compared to these values.

Looking at the ML metrics (Table 2), we note the ML performance of the decision-focused methods varies widely. In particular, the testing MSE for portfolio optimization is quite high compared to the two stage approach. This mismatch between the MSE and decision quality exemplifies the need for training with the downstream optimization task in mind in that even though the MIPaaL model has worse MSE than TwoStage , it results in much higher-return decisions.

In terms of the matching problem, we see that even though TwoStage has better cross-entropy loss at test time, as it was trained with that specific classification loss in mind, it lacks in AUC which both corresponds to the findings in previous work (Wilder, Dilkina, and Tambe 2019), and indicates that the predictions learned by MIPaaL may sometimes also be accurate in a traditional sense.

In knapsack instances, MIPaaL gives improvement in MIP solution quality over other approaches. In addition, the MSE is best among optimization methods whereas the correlation is lower than all approaches other than MIPaaL-100. We note that the performance improvement isn't as drastic as for the other settings. It is possible that the constraints are not as combinatorial as in the other domains and thus even straightforward ML losses may still perform well in decision

Table 1: Decision quality. Comparison in terms of realized optimization objective: monthly percentage increase for portfolio optimization (SP500 and DAX), number of pairs successfully matched for Matching, and value of items for Knapsack. MIPaaL gives 2x monthly returns on SP500 and 8x on DAX, and improves the objective by 40.3% and 1.2% for Matching and Knapsack respectively. MIPaaL outperforms all other variants considered.

	SP500	DAX	Matching	Knapsack
MIPaaL	**2.79 ± 0.17**	**5.70 ± 0.68**	**4.80 ± 0.71**	**507.70 ± 0.471**
MIPaaL-Warm	1.09 ± 0.18	0.68 ± 1.01	2.14 ± 0.51	499.60 ± 0.566
MIPaaL-Hybrid	1.08 ± 0.15	0.74 ± 1.10	3.21 ± 0.73	503.36 ± 0.578
MIPaaL-1000	2.60 ± 0.16	4.39 ± 0.66	3.45 ± 0.71	506.34 ± 0.662
MIPaaL-100	1.25 ± 0.14	0.35 ± 0.63	2.57 ± 0.54	505.99 ± 0.621
RootLP (Wilder et al. 2019)	1.97 ± 0.17	-1.97 ± 0.69	3.17 ± 0.60	501.58 ± 0.662
TwoStage	1.19 ± 0.15	0.70 ± 1.46	3.42 ± 0.78	501.49 ± 0.523

Table 2: ML performance on test set. TwoStage wins on ML metrics used for training (MSE, CE), whereas MIPaaL has inferior ML metrics while improving decision quality. In all benchmarks, the predictive problem is hard as evidenced by the ML metrics of all methods. Bolded entries have 95% confidence intervals overlapping with the best entry.

	SP500		DAX		Matching		Knapsack	
	MSE	Corr	MSE	Corr	CE	AUC	MSE	Corr
MIPaaL	0.22 ± 0.043	**0.15 ± 0.015**	0.13 ± 0.017	**0.25 ± 0.032**	0.66 ± 0.009	**0.535 ± 0.004**	2774 ± 97.664	0.567 ± 0.002
MIPaaL-Warm	**0.11 ± 0.010**	-0.01 ± 0.010	**0.09 ± 0.067**	0.07 ± 0.030	0.52 ± 0.003	0.509 ± 0.003	4660 ± 72.008	0.593 ± 0.003
MIPaaL-Hybrid	**0.09 ± 0.030**	**0.13 ± 0.013**	0.13 ± 0.099	**0.26 ± 0.026**	0.55 ± 0.002	0.502 ± 0.004	3824 ± 82.828	0.608 ± 0.006
MIPaaL-1000	**0.12 ± 0.020**	**0.13 ± 0.013**	0.35 ± 0.010	**0.27 ± 0.035**	0.61 ± 0.010	0.506 ± 0.007	5821 ± 154.793	0.590 ± 0.005
MIPaaL-100	0.98 ± 0.089	0.12 ± 0.013	0.99 ± 0.060	**0.26 ± 0.037**	0.54 ± 0.013	0.503 ± 0.004	5801 ± 145.331	0.553 ± 0.007
RootLP (Wilder et al. 2019)	0.71 ± 0.178	**0.15 ± 0.013**	1.06 ± 0.137	**0.28 ± 0.032**	0.49 ± 0.007	0.513 ± 0.001	6267 ± 212.063	0.574 ± 0.002
TwoStage	**0.09 ± 0.017**	0.06 ± 0.011	**0.02 ± 0.066**	0.13 ± 0.032	**0.39 ± 0.004**	0.514 ± 0.005	**684 ± 15.568**	**0.649 ± 0.002**

quality. On the other combinatorial problems, our decision quality improves by anywhere from 40% to 8x compared to the baselines, emphasizing the benefit of using MIPaaL for problems with combinatorial structure.

Runtime: Table 3 summarizes our benchmarks as well as running time statistics for components of MIPaaL. It shows that the four benchmarks correspond to MIP instances of various structure, number of variables, and number of constraints. We report the average number of added cuts per MIP instance generated during training, the average time per epoch, and the percentage of that time dedicated to the forward and backward pass through the MIP layer in particular. The number of added cuts in the forward pass is on the order of a few thousands for all four problem types. SP500 and Matching take longer per epoch than DAX and Knapsack. The table shows that for both of these methods a big percentage of the train time is dedicated to the backward pass through the MIP layer rather than the forward Gomory-based solver. This is explained by the large size of the corresponding cutting plane LPs for which the backward pass needs to solve through the KKT conditions. Furthermore, recent GPU-accelerated QP solvers introduced in (Amos and Kolter 2017) would accelerate the backward pass. On average, the forward pass through MIPaaL takes 0.26, 0.10, 0.80, and 0.16 seconds for SP500, DAX, Matching, and Knapsack instances respectively, and 1.72, 0.02, 12.42, and 0.14 seconds for the backward passes respectively. Further timing results are in the appendix[1]).

Transfer learning: To test generalization performance, we evaluate MIPaaL, RootLP, and TwoStage on transfer learning tasks for portfolio optimization. In this transfer learning setting, models are trained on 30 assets randomly drawn from SP500 (SP-30^a), with data from Jan 2005 - Dec 2010. These learned models are then evaluated on data from Dec 2013 - Nov 2016 to test various generalization aspects. To test generalization across the data distribution we evaluate on 1) SP-30^b, a set of 30 randomly drawn assets from the SP500, disjoint from SP-30^a, and 2) the DAX, a separate index comprising 30 companies from a different country. Similarly, we evaluate on instances with a varying number of assets in SP-50, SP-100, SP-200 and SP-500 which contain 50, 100, and 200 each with unique assets disjoint from SP-30^a and SP-30^b, as well as on all 505 assets we have data for in SP-500.

Varying data distribution: The transfer learning results (Table 4) demonstrate that MIPaaL generalizes to unseen assets and countries. On SP-30^b, MIPaaL doubles the average rate of return over the standard TwoStage approach, and gives a 59% improvement over RootLP. Furthermore, while the transferred model applied to DAX doesn't beat MIPaaL trained on the same task (Table 1), it improves performance over the RootLP and TwoStage trained on DAX, showing that MIPaaL learns a useful model for portfolio optimization as a whole. Lastly, MIPaaL's performance improvement over TwoStage occurs despite a much higher MSE (Table 4).

Varying problem size: MIPaaL efficiently predicts for the general task of portfolio optimization, regardless of the set or number of assets (Table 4). Similarly to DAX, for SP500 the transferred MIPaaL model trained on SP-30^a outperforms the TwoStage method trained on SP500 data (Table 1).

These results indicate that MIPaaL can train predictive

Table 3: Problem statistics and timing results. Timing results are number of epochs until validation MIP performance convergence, average time per epoch, and average % time taken per epoch to compute Forward and Backward passes through the MIP Layer.

	Num Instances			Problem Sizes			Solve Statistics				
	Train	Val	Test	Bin Vars	Cont Vars	Cons	Avg Cuts	Epoch (s)	Forward	Backward	# Epochs
SP500	72	35	36	1000	3011	5026	2690	486	3.88%	25.46%	28
DAX	72	35	36	60	185	314	1387	47	15.63%	3.34%	20
Matching	16	11	11	2500	0	102	4984	604	2.13%	32.90%	35
Knapsack	56	19	19	48	0	1	1261	208	4.31%	0.36%	31

Table 4: Transfer learning metrics for models trained on SP-30^a and tested varying data distribution and problem size.

		SP-30^b	DAX	SP-50	SP-100	SP-200	SP500
Decision Quality	MIPaaL	$\mathbf{2.02 \pm 0.48}$	$\mathbf{2.77 \pm 0.40}$	$\mathbf{1.93 \pm 0.13}$	$\mathbf{2.27 \pm 0.11}$	$\mathbf{2.17 \pm 0.48}$	$\mathbf{2.26 \pm 0.37}$
	RootLP	1.81 ± 0.44	1.74 ± 0.43	1.50 ± 0.09	1.58 ± 0.08	1.82 ± 0.41	1.90 ± 0.29
	TwoStage	0.71 ± 0.04	0.82 ± 0.54	1.58 ± 0.13	1.22 ± 0.09	1.50 ± 0.58	1.11 ± 0.35
ML Loss	MIPaaL	4.81 ± 8.59	4.59 ± 8.80	5.42 ± 3.16	5.42 ± 2.37	5.25 ± 1.83	5.43 ± 1.67
	RootLP	5.14 ± 1.02	5.39 ± 1.04	4.73 ± 3.17	4.88 ± 2.58	4.81 ± 1.91	4.83 ± 1.56
	TwoStage	$\mathbf{0.08 \pm 0.05}$	$\mathbf{0.07 \pm 0.03}$	$\mathbf{0.08 \pm 0.02}$	$\mathbf{0.07 \pm 0.01}$	$\mathbf{0.08 \pm 0.01}$	$\mathbf{0.08 \pm 0.01}$

models which generalize across data distribution and MIP problem size. Comparing MIPaaL trained on SP-30^a, which gets 2.26 returns on the full SP500, to TwoStage trained on SP500, which gets 1.19 return in Table 1, we can see that MIPaaL outperforms TwoStage even when MIPaaL doesn't see most assets in the dataset. Additionally, MIPaaL trained on choosing from 30 assets has more than an order of magnitude decrease in training time compared to training on 500 assets. In effect, when training time is limited, we can still efficiently train a model on small MIP instances which performs well on larger MIP instances compared to TwoStage, while training faster than MIPaaL on fully-sized MIPs.

Related Work

The interaction of machine learning and optimization has provided a diverse set of tools for efficiently solving a wide variety of problems. Recent work has framed traditionally heuristic components of optimization algorithms as machine learning tasks such as learning in branch and bound for MIPs (Khalil et al. 2016; 2017b; He, Daume III, and Eisner 2014; Bengio, Lodi, and Prouvost 2018). These approaches work inside an exact solver when the MIP is completely specified. Deep reinforcement learning methods have also been used to generate high-quality data-driven solutions to hard problems such as graph optimization (Khalil et al. 2017a), vehicle routing (Kool, van Hoof, and Welling 2019; Nazari et al. 2018), and realtime patrol planning in security games (Yu et al. 2019). These problems require known features, like the true objective coefficients, and are used to quickly generate high-quality solutions with known objectives, which is computationally intractable for an exact solver.

Several approaches have been proposed which embed optimization components in neural networks. This includes specific problems such as Markowitz portfolio optimization (Bengio 1997) or finding physically feasible state transitions (de Avila Belbute-Peres et al. 2018), as well as larger classes which exhibit properties like convexity or submodularity. Problem classes used in end-to-end training include polynomial-time solvable frameworks like quadratic programs (Amos and Kolter 2017) and linear programs (Wilder, Dilkina, and Tambe 2019), as well as zero-sum games (Ling, Fang, and Kolter 2018; Perrault et al. 2019). In addition, a solution is proposed for problems encoded as submodular optimization problems in (Wilder, Dilkina, and Tambe 2019). In (Elmachtoub and Grigas 2017), the authors optimize a decision-focused regret bound. (Kao, Roy, and Yan 2009) instantiate end-to-end learning with linear models predicting components of a quadratic optimization function, and (Demirovic et al. 2019) exhaustively search linear models to perform well on predict + optimize for knapsack. Lastly, (Wang et al. 2019) incorporate a differentiable semidefinite program as a relaxation for MAXSAT. To our knowledge, MIPaaL is the first approach for imbuing neural networks with the highly flexible Mixed Integer Program, a widely-used class of potentially inapproximable NP-Hard optimization problems, while providing tight feedback on decision quality.

Conclusion

We propose MIPaaL, a principled method for incorporating mixed integer programs as a differentiable layer in neural networks. We approach the task of differentiating through this flexible, discrete, and potentially inapproximable problem by algorithmically generating a potentially large but equivalent continuous optimization problem via cutting planes. We instantiate our proposed approach for decision-focused learning wherein a predictive model is trained with a loss function that directly corresponds to the quality of the decisions made based on the predictions. MIPaaL is evaluated on two settings of portfolio optimization, one setting of diverse bipartite matching, and a knapsack setting, all of which contain modeling techniques widely used in combinatorial optimization that make the problem more complex but more

realistic. Furthermore, we evaluate MIPaaL in several transfer learning settings. We demonstrate empirically that MIPaaL is able to outperform the standard approach of decoupling the prediction and decision components, as well as an approach of using a continuous relaxation of the original combinatorial optimization problem. To better understand the impact of the cutting plane technique, we explore hybrid strategies that stop the cutting plane generation early, simultaneously optimize ML loss and MIP loss, and initialize the network feeding into MIPaaL with a model trained via the decoupled approach. Ultimately, we find empirically that our approach can give high-quality solutions in the investigated settings.

Acknowledgements

Ferber and Dilkina were partially supported by NSF award #1914522. Wilder was partially supported by a NSF Graduate Research Fellowship. Tambe and Wilder were supported by the Army Research Office (MURI W911NF1810208).

References

Amos, B., and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *ICML*.

Balas, E.; Ceria, S.; Cornuéjols, G.; and Natraj, N. 1996. Gomory cuts revisited. *Operations Research Letters* 19(1).

Benabbou, N.; Chakraborty, M.; Ho, X.-V.; Sliwinski, J.; and Zick, Y. 2018. Diversity constraints in public housing allocation. In *AAMAS*, 973–981.

Bengio, Y.; Lodi, A.; and Prouvost, A. 2018. Machine learning for combinatorial optimization: a methodological tour d'horizon. *arXiv preprint arXiv:1811.06128*.

Bengio, Y. 1997. Using a financial training criterion rather than a prediction criterion. *Intl Journal of Neural Systems* 8(04):433–443.

Bertsimas, D.; Darnell, C.; and Soucy, R. 1999. Portfolio construction through mixed-integer programming at grantham, mayo, van otterloo and company. *Interfaces* 29(1):49–66.

Bodur, M.; Dash, S.; and Günlük, O. 2017. Cutting planes from extended lp formulations. *Math. Programming* 161(1-2):159–192.

Chan, C. W.; Farias, V. F.; Bambos, N.; and Escobar, G. J. 2012. Optimizing intensive care unit discharge decisions with patient readmissions. *Operations research* 60(6):1323–1341.

Dash, S.; Dobbs, N. B.; Günlük, O.; Nowicki, T. J.; and Świrszcz, G. M. 2014. Lattice-free sets, multi-branch split disjunctions, and mixed-integer programming. *Math. Programming* 145(1):483–508.

de Avila Belbute-Peres, F.; Smith, K.; Allen, K.; Tenenbaum, J.; and Kolter, J. Z. 2018. End-to-end differentiable physics for learning and control. In *NeurIPS*, 7178–7189.

Demirovic, E.; Stuckey, P. J.; Bailey, J.; Chan, J.; Leckie, C.; Ramamohanarao, K.; and Guns, T. 2019. Predict+optimise with ranking objectives: Exhaustively learning linear functions. In *IJCAI*.

Dickerson, J. P.; Manlove, D. F.; Plaut, B.; Sandholm, T.; and Trimble, J. 2016. Position-indexed formulations for kidney exchange. In *ACM Conference on Economics and Computation*, 25–42.

Elmachtoub, A. N., and Grigas, P. 2017. Smart" predict, then optimize". *arXiv preprint arXiv:1710.08005*.

Gomory, R. 1960. An algorithm for the mixed integer problem. Technical report, RAND CORP.

He, H.; Daume III, H.; and Eisner, J. M. 2014. Learning to search in branch and bound algorithms. In *NeurIPS*.

Kao, Y.; Roy, B. V.; and Yan, X. 2009. Directed regression. In *NeurIPS*.

Karush, W. 1939. Minima of functions of several variables with inequalities as side constraints. *M. Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago*.

Khalil, E. B.; Le Bodic, P.; Song, L.; Nemhauser, G.; and Dilkina, B. 2016. Learning to branch in mixed integer programming. In *AAAI*.

Khalil, E.; Dai, H.; Zhang, Y.; Dilkina, B.; and Song, L. 2017a. Learning combinatorial optimization algorithms over graphs. In *NeurIPS*, 6348–6358.

Khalil, E. B.; Dilkina, B.; Nemhauser, G. L.; Ahmed, S.; and Shao, Y. 2017b. Learning to run heuristics in tree search. In *IJCAI*.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Attention, learn to solve routing problems! In *ICLR*.

Ling, C. K.; Fang, F.; and Kolter, J. Z. 2018. What game are we playing? end-to-end learning in normal and extensive form games. *IJCAI*.

Markowitz, H. 1952. Portfolio selection. *The journal of finance*.

Mohsenian-Rad, A.-H., and Leon-Garcia, A. 2010. Optimal residential load control with price prediction in real-time electricity pricing environments. *IEEE Trans. Smart Grid* 1(2):120–133.

Nazari, M.; Oroojlooy, A.; Snyder, L.; and Takác, M. 2018. Reinforcement learning for solving the vehicle routing problem. In *NeurIPS*, 9839–9849.

Nemhauser, G. L. 2013. Integer programming: The global impact. ISyE DOS Optimization Seminar presentation at Georgia Tech http://hdl.handle.net/1853/49829.

O'Mahony, E., and Shmoys, D. B. 2015. Data analysis and optimization for (citi) bike sharing. In *AAAI*.

Perrault, A.; Wilder, B.; Ewing, E.; Mate, A.; Dilkina, B.; and Tambe, M. 2019. Decision-focused learning of adversary behavior in security games. *arXiv preprint arXiv:1903.00958*.

Quandl. 2019. Various end-of-day data. https://www.quandl.com/.

Sato, K.; Kato, Y.; Hamada, M.; Akutsu, T.; and Asai, K. 2011. Ipknot: fast and accurate prediction of rna secondary structures with pseudoknots using integer programming. *Bioinformatics*.

Sen, P.; Namata, G.; Bilgic, M.; Getoor, L.; Galligher, B.; and Eliassi-Rad, T. 2008. Collective classification in network data. *AI magazine* 29(3):93–93.

Trilling, L.; Guinet, A.; and Le Magny, D. 2006. Nurse scheduling using integer linear programming and constraint programming. *IFAC Proceedings Volumes* 39(3):671–676.

Wang, P.-W.; Donti, P.; Wilder, B.; and Kolter, Z. 2019. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, 6545–6554.

Warner, D. M. 1976. Scheduling nursing personnel according to nursing preference: A mathematical programming approach. *Operations Research* 24(5):842–856.

Wilder, B.; Dilkina, B.; and Tambe, M. 2019. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *AAAI*.

Wolsey, L. A., and Nemhauser, G. L. 2014. *Integer and combinatorial optimization*. John Wiley & Sons.

Yu, L.; Wu, Y.; Singh, R.; Joppa, L.; and Fang, F. 2019. Deep reinforcement learning for green security game with online information. In *AAAI*.