CombOptNet: Fit the Right NP-Hard Problem by Learning Integer Programming Constraints

Anselm Paulus [1] **Michal Rolínek** [1] **Vít Musil** [2] **Brandon Amos** [3] **Georg Martius** [1]

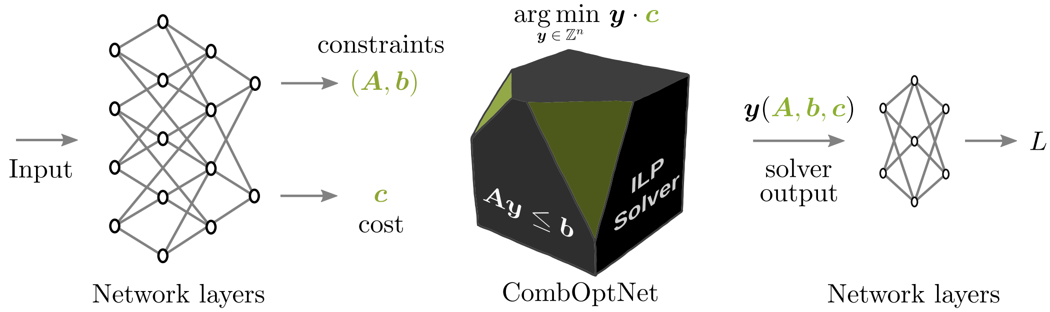


Figure 1: CombOptNet as a module in a deep architecture.

Abstract

Bridging logical and algorithmic reasoning with modern machine learning techniques is a fundamental challenge with potentially transformative impact. On the algorithmic side, many NP-HARD problems can be expressed as integer programs, in which the constraints play the role of their "combinatorial specification." In this work, we aim to integrate integer programming solvers into neural network architectures as layers capable of learning *both* the cost terms and the constraints. The resulting end-to-end trainable architectures jointly extract features from raw data and solve a suitable (learned) combinatorial problem with state-of-the-art integer programming solvers. We demonstrate the potential of such layers with an extensive performance analysis on synthetic data and with a demonstration on a competitive computer vision keypoint matching benchmark.

[1]Max-Planck-Institute for Intelligent Systems, Tübingen, Germany [2]Masaryk University, Brno, Czechia [3]Facebook AI Research, USA. Correspondence to: Anselm Paulus & Georg Martius <firstname.lastname@tuebingen.mpg.de>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

1. Introduction

It is becoming increasingly clear that to advance artificial intelligence, we need to dramatically enhance the reasoning, algorithmic, logical, and symbolic capabilities of data-driven models. Only then we can aspire to match humans in their astonishing ability to perform complicated abstract tasks such as playing chess only based on visual input. While there are decades worth of research directed at solving complicated abstract tasks *from their abstract formulation*, it seems very difficult to align these methods with deep learning architectures needed for processing raw inputs. Deep learning methods often struggle to implicitly acquire the abstract reasoning capabilities to solve and generalize to new tasks. Recent work has investigated more structured paradigms that have more explicit reasoning components, such as layers capable of convex optimization. In this paper, we focus on combinatorial optimization, which has been well-studied and captures nontrivial reasoning capabilities over discrete objects. Enabling its unrestrained usage in machine learning models should fundamentally enrich the set of available components.

On the technical level, the main challenge of incorporating combinatorial optimization into the model typically amounts to *non-differentiability* of methods that operate with discrete inputs or outputs. Three basic approaches to overcome this are to a) develop "soft" continuous versions of the discrete algorithms (Wang et al., 2019; Zanfir & Sminchisescu, 2018); b) adjust the topology of neural network architectures to express certain algorithmic behaviour (Graves et al., 2014; 2016; Battaglia et al., 2018); c) provide an informative gradient approximation for the discrete algorithm (Vlastelica et al., 2020a; Berthet et al., 2020). While the last strategy requires nontrivial theoretical considerations, it can resolve the non-differentiability in the strongest possible sense; without any compromise on the performance of the original discrete algorithm. We follow this approach.

The most succesful generic approach to combinatorial optimization is integer linear programming (ILP). Integrating ILPs as building blocks of differentiable models is challenging because of the nontrivial dependency of the solution on the cost terms and on the constraints. Learning parametrized cost terms has been addressed in Vlastelica et al. (2020a); Berthet et al. (2020); Ferber et al. (2020), the learnability of constraints is, however, unexplored. At the same time, the constraints of an ILP are of critical interest due to their **remarkable expressive power**. Only by modifying the constraints, one can formulate a number of diverse combinatorial problems (SHORTEST-PATH, MATCHING, MAX-CUT, KNAPSACK, TRAVELLING SALESMAN). In that sense, learning ILP constraints corresponds to **learning the combinatorial nature** of the problem at hand.

In this paper, we propose a backward pass (gradient computation) for ILPs covering their **full specification**, allowing to use blackbox ILPs as combinatorial layers at any point in the architecture. This layer can jointly learn the cost terms and the constraints of the integer program, and as such it aspires to achieve **universal combinatorial expressivity**. We demonstrate the potential of this method on multiple tasks. First, we extensively analyze the performance on synthetic data. This includes the inverse optimization task of recovering an unknown set of constraints, and a KNAPSACK problem specified in plain text descriptions. Finally, we demonstrate the applicability to real-world tasks on a competitive computer vision keypoint matching benchmark.

1.1. Related Work

Learning for combinatorial optimization. Learning methods can powerfully augment classical combinatorial optimization methods with data-driven knowledge. This includes work that learns how to solve combinatorial optimization problems to improve upon traditional solvers that are otherwise computationally expensive or intractable, e.g. by using reinforcement learning (Zhang & Dietterich, 2000; Bello et al., 2016; Khalil et al., 2017; Nazari et al., 2018), learning graph-based algorithms (Veličković et al., 2018; Veličković et al., 2020; Wilder et al., 2019), learning to branch (Balcan et al., 2018), solving SMT formulas (Balunovic et al., 2018) and TSP instances (Kool et al., 2018). Nair et al. (2020) have recently scaled up learned MIP solvers on non-trivial production datasets. In a more general computational paradigm, Graves et al. (2014; 2016) parameterize and learn Turing machines.

Optimization-based modeling for learning. In the other direction, optimization serves as a useful modeling paradigm to improve the applicability of machine learning models and to add domain-specific structures and priors. In the continuous setting, differentiating through optimization problems is a foundational topic as it enables optimization algorithms to be used as a layer in end-to-end trainable models (Domke, 2012; Gould et al., 2016). This approach has been recently studied in the convex setting in OptNet (Amos & Kolter, 2017) for quadratic programs, and more general cone programs in Amos (2019, Section 7.3) and Agrawal et al. (2019a;b). One use of this paradigm is to incorporate the knowledge of a downstream optimization-based task into a predictive model (Elmachtoub & Grigas, 2020; Donti et al., 2017). Extending beyond the convex setting, optimization-based modeling and differentiable optimization are used for sparse structured inference (Niculae et al., 2018), MAXSAT (Wang et al., 2019), submodular optimization (Djolonga & Krause, 2017) mixed integer programming (Ferber et al., 2020), and discrete and combinational settings (Vlastelica et al., 2020a; Berthet et al., 2020). Applications of optimization-based modeling include computer vision (Rolínek et al., 2020b;a), reinforcement learning (Dalal et al., 2018; Amos & Yarats, 2020; Vlastelica et al., 2020b), game theory (Ling et al., 2018), and inverse optimization (Tan et al., 2020), and meta-learning (Bertinetto et al., 2019; Lee et al., 2019).

2. Problem description

Our goal is to incorporate an ILP as a differentiable layer in neural networks that inputs both **constraints and objective** coefficients and outputs the corresponding ILP solution.

Furthermore, we aim to embed ILPs in a **blackbox manner**: On the forward pass, we run the unmodified optimized solver, making no compromise on its performance. The task is to propose an **informative gradient** for the solver as it is. We never modify, relax, or soften the solver.

We assume the following form of a bounded integer program:

$$\min_{\boldsymbol{y} \in Y} \boldsymbol{c} \cdot \boldsymbol{y} \qquad \text{subject to} \qquad \boldsymbol{A}\boldsymbol{y} \leq \boldsymbol{b}, \tag{1}$$

where Y is a bounded subset of $\mathbb{Z}^n$, $n \in \mathbb{N}$, $\boldsymbol{c} \in \mathbb{R}^n$ is the cost vector, $\boldsymbol{y}$ are the variables, $\boldsymbol{A} = [\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m] \in \mathbb{R}^{m \times n}$ is the matrix of constraint coefficients and $\boldsymbol{b} \in \mathbb{R}^m$ is the bias term. The point at which the minimum is attained is denoted by $\boldsymbol{y}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$.

The task at hand is to provide gradients for the mapping $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}) \to \boldsymbol{y}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$, in which the triple $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ is the specification of the ILP solver containing both the cost and the constraints, and $\boldsymbol{y}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}) \in Y$ is the optimal solution of the instance.

Example. The ILP formulation of the KNAPSACK problem can be written as

$$\max_{\boldsymbol{y} \in \{0,1\}^n} \boldsymbol{c} \cdot \boldsymbol{y} \qquad \text{subject to} \qquad \boldsymbol{a} \cdot \boldsymbol{y} \leq b, \tag{2}$$

where $\boldsymbol{c} = [c_1, \ldots, c_n] \in \mathbb{R}^n$ are the prices of the items, $\boldsymbol{a} = [a_1, \ldots, a_n] \in \mathbb{R}^n$ their weights and $b \in \mathbb{R}$ the knapsack capacity.

Similar encodings can be found for many more - often NP-HARD - combinatorial optimization problems including those mentioned in the introduction. Despite the apparent difficulty of solving ILPs, modern highly optimized solvers (Gurobi Optimization, 2019; Cplex, 2009) can routinely find optimal solutions to instances with thousands of variables.

2.1. The main difficulty.

Differentiability. Since there are finitely many available values of $\boldsymbol{y}$, the mapping $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}) \to \boldsymbol{y}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ is piecewise constant; and as such, its **true gradient is zero** almost everywhere. Indeed, a small perturbation of the constraints or of the cost does *typically* not cause a change in the optimal ILP solution. The zero gradient has to be suitably supplemented.

Gradient surrogates w.r.t. objective coefficients $\boldsymbol{c}$ have been studied intensively (see e.g. Elmachtoub & Grigas, 2020; Vlastelica et al., 2020a; Ferber et al., 2020). Here, we focus on the differentiation w.r.t. constraints coefficients $(\boldsymbol{A}, \boldsymbol{b})$ that has been unexplored by prior works.

LP vs. ILP: Active constraints. In the LP case, the integrality constraint on Y is removed. As a result, in the typical case, the optimal solution can be written as the unique solution to a linear system determined by the set of active constraints. This captures the relationship between the constraint matrix and the optimal solution. Of course, this relationship is differentiable.

However, in the case of an ILP the **concept of active constraints vanishes**. There can be optimal solutions for which no constraint is tight. Providing gradients for nonactive-but-relevant constraints is the principal difficulty. The complexity of the interaction between the constraint set and the optimal solution is reflecting the NP-HARD nature of ILPs and is the reason why relying on the LP case is of little help.

3. Method

First, we reformulate the gradient problem as a descend direction task. We have to resolve an issue that the suggested gradient update $\boldsymbol{y} - \mathrm{d}\boldsymbol{y}$ to the optimal solution $\boldsymbol{y}$ is typically unattainable, i.e. $\boldsymbol{y} - \mathrm{d}\boldsymbol{y}$ is not a feasible integer point. Next, we generalize the concept of active constraints. We substitute the binary information "active/nonactive" by a continuous proxy based on Euclidean distance.

Descent direction. On the backward pass, the gradient of the layers following the ILP solver is given. Our aim is to propose a direction of change to the constraints and to the cost such that the solution of the updated ILP moves towards the negated incoming gradient's direction (i.e. the descent direction).

Denoting a loss by L, let $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$ and the incoming gradient $\mathrm{d}\boldsymbol{y} = \partial L / \partial \boldsymbol{y}$ at the point $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ be given. We are asked to return a gradient corresponding to $\partial L / \partial \boldsymbol{A}$, $\partial L / \partial \boldsymbol{b}$ and $\partial L / \partial \boldsymbol{c}$. Our goal is to find directions $\mathrm{d}\boldsymbol{A}$, $\mathrm{d}\boldsymbol{b}$ and $\mathrm{d}\boldsymbol{c}$ for which the distance between the updated solution $\boldsymbol{y}(\boldsymbol{A} - \mathrm{d}\boldsymbol{A}, \boldsymbol{b} - \mathrm{d}\boldsymbol{b}, \boldsymbol{c} - \mathrm{d}\boldsymbol{c})$ and the target $\boldsymbol{y} - \mathrm{d}\boldsymbol{y}$ decreases the most.

If the mapping $\boldsymbol{y}$ is differentiable, it leads to the correct gradients $\partial L / \partial \boldsymbol{A} = \partial L / \partial \boldsymbol{y} \cdot \partial \boldsymbol{y} / \partial \boldsymbol{A}$ (analogously for $\boldsymbol{b}$ and $\boldsymbol{c}$). See Proposition S1 in the Supplementary material, for the precise formulation and for the proof. The main advantage of this formulation is that it is **meaningful even in the discrete case**.

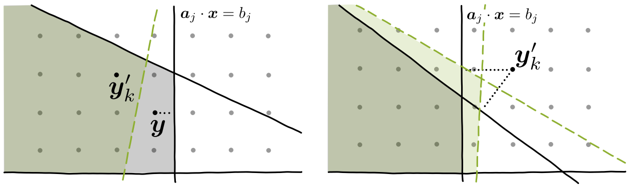
However, every ILP solution $\boldsymbol{y}(\boldsymbol{A} - \mathrm{d}\boldsymbol{A}, \boldsymbol{b} - \mathrm{d}\boldsymbol{b}, \boldsymbol{c} - \mathrm{d}\boldsymbol{c})$ is restricted to integer points and its ability to approach the point $\boldsymbol{y} - \mathrm{d}\boldsymbol{y}$ is limited unless $\mathrm{d}\boldsymbol{y}$ is also an integer point. To achieve this, let us decompose

$$\mathrm{d}\boldsymbol{y} = \sum_{k=1}^{n} \lambda_k \Delta_k, \tag{3}$$

where $\Delta_k \in \{-1, 0, 1\}^n$ are some integer points and $\lambda_k \geq 0$ are scalars. The choice of basis Δ_k is discussed in a separate paragraph, for now it suffices to know that every point $\boldsymbol{y}'_k = \boldsymbol{y} + \Delta_k$ is an integer point neighbour of $\boldsymbol{y}$ pointing in a "direction of $\mathrm{d}\boldsymbol{y}$". We then address separate problems with $\mathrm{d}\boldsymbol{y}$ replaced by the integer updates Δ_k.

In other words, our goal here is to find an update on $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{c}$ that eventually pushes the solution closer to $\boldsymbol{y} + \Delta_k$. Staying true to linearity of the standard gradient mapping, we then aim to compose the final gradient as a linear combination of the gradients coming from the subproblems.

Constraints update. To get a meaningful update for a realizable change Δ_k, we take a gradient of a piecewise affine local *mismatch* function P_{Δ_k}. The definition of P_{Δ_k} is based on a geometric understanding of the underlying structure. To that end, we rely on the Euclidean distance between a point and a hyperplane. Indeed, for any point $\boldsymbol{y}$ and a given hyperplane, parametrized by vector $\boldsymbol{a}$ and scalar



(a) $\boldsymbol{y}'_k$ is feasible but $\boldsymbol{y}'_k \neq \boldsymbol{y}$. (b) $\boldsymbol{y}'_k$ is infeasible.

Figure 2: Geometric interpretation of the suggested constraint update. (a) All the constraints are satisfied for $\boldsymbol{y}'_k$. The proxy minimizes the distance to the nearest ("most active") constraint to make $\boldsymbol{y}$ "less feasible". A possible updated feasible region is shown in green. (b) The suggested $\boldsymbol{y}'_k$ satisfies one of three constraints. The proxy minimizes the distance to violated constraints to make $\boldsymbol{y}'_k$ "more feasible".

Module 1 CombOptNet

```
function FORWARDPASS(A, b, c)
    y := Solver(A, b, c)
    save y and A, b, c for backward pass
    return y
function BACKWARDPASS(dy)
    load y and A, b, c from forward pass
    Decompose dy = Σ_k λ_k Δ_k
        // set Δ_k as in (9) and λ_k as in Proposition 1
    Calculate the gradients
        dA^k := ∂P_{Δ_k}/∂A, db^k := ∂P_{Δ_k}/∂b, dc^k := ∂P_{Δ_k}/∂c
        // P_{Δ_k} defined in (5) and (7)
    Compose dA, db, dc := Σ_k λ_k (dA^k, db^k, dc^k)
        // According to (6)
    return dA, db, dc
```

b as $\boldsymbol{x} \mapsto \boldsymbol{a} \cdot \boldsymbol{x} - b$, we have:

$$\operatorname{dist}(\boldsymbol{a}, b; \boldsymbol{y}) = |\boldsymbol{a} \cdot \boldsymbol{y} - b| / \|\boldsymbol{a}\|. \tag{4}$$

Now, we distinguish the cases based on whether $\boldsymbol{y}'_k$ is feasible, i.e. $\boldsymbol{A}\boldsymbol{y}'_k \leq \boldsymbol{b}$, or not. The infeasibility of $\boldsymbol{y}'_k$ can be caused by one or more constraints. We then define

$$P_{\Delta_k}(\boldsymbol{A}, \boldsymbol{b}) = \begin{cases} \min_j \operatorname{dist}(\boldsymbol{a}_j, b_j; \boldsymbol{y}) \\ \qquad \text{if } \boldsymbol{y}'_k \text{ is feasible and } \boldsymbol{y}'_k \neq \boldsymbol{y} \\ \sum_j [\![\boldsymbol{a}_j \cdot \boldsymbol{y}'_k > b_j]\!] \operatorname{dist}(\boldsymbol{a}_j, b_j; \boldsymbol{y}'_k) \\ \qquad \text{if } \boldsymbol{y}'_k \text{ is infeasible} \\ 0 \qquad \text{if } \boldsymbol{y}'_k = \boldsymbol{y} \text{ or } \boldsymbol{y}'_k \notin Y, \end{cases} \tag{5}$$

where $[\![\cdot]\!]$ is the Iverson bracket. The geometric intuition behind the suggested mismatch function is described in Fig. 2 and its caption. Note that tighter constraints contribute more to P_{Δ_k}. In this sense, the mismatch function **generalizes the concept of active constraints**. In practice, the minimum is softened to allow multiple constraints to be updated simultaneously. For details, see the Supplementary material.

Imposing linearity and using decomposition (3), we define the outcoming gradient $\mathrm{d}\boldsymbol{A}$ as

$$\mathrm{d}\boldsymbol{A} = \sum_{k=1}^{n} \lambda_k \frac{\partial P_{\Delta_k}}{\partial \boldsymbol{A}}(\boldsymbol{A}, \boldsymbol{b}). \tag{6}$$

and analogously for $\mathrm{d}\boldsymbol{b}$, by differentiating with respect to $\boldsymbol{b}$. The computation is summarized in Module 1.

Note that our mapping $\mathrm{d}\boldsymbol{y} \mapsto \mathrm{d}\boldsymbol{A}, \mathrm{d}\boldsymbol{b}$ is homogeneous. It is due to the fact that the whole situation is rescaled to one case (choice of basis) where the gradient is computed and then rescaled back (scalars λ_k). The most natural scale agrees with the situation when the "targets" $\boldsymbol{y}'_k$ are the closest integer neighbors. This ensures that the situation does not collapse to a trivial solution (zero gradient) and, simultaneously, that we do not interfere with very distant values of $\boldsymbol{y}$.

This basis selection plays a role of a "homogenizing hyperparamter" (λ in (Vlastelica et al., 2020a) or ε in (Berthet et al., 2020)). In our case, we explicitly construct a correct basis and do not need to optimize any additional hyperparameter.

Cost update. Putting aside distinguishing of feasible and infeasible $\boldsymbol{y}'_k$, the cost update problem has been addressed in multiple previous works. We choose to use the simplest approach of (Elmachtoub & Grigas, 2020) and set

$$P_{\Delta_k}(\boldsymbol{c}) = \begin{cases} \boldsymbol{c} \cdot \Delta_k & \text{if } \boldsymbol{y}'_k \text{ is feasible} \\ 0 & \text{if } \boldsymbol{y}'_k \text{ is infeasible or } \boldsymbol{y}'_k \notin Y. \end{cases} \tag{7}$$

The gradient $\mathrm{d}\boldsymbol{c}$ is then composed analogously as in (6).

The choice of the basis. Denote by $k_1, \dots, k_n$ the indices of the coordinates in the absolute values of $\mathrm{d}\boldsymbol{y}$ in decreasing order, i.e.

$$|\mathrm{d}\boldsymbol{y}_{k_1}| \geq |\mathrm{d}\boldsymbol{y}_{k_2}| \geq \cdots \geq |\mathrm{d}\boldsymbol{y}_{k_n}| \tag{8}$$

and set

$$\Delta_k = \sum_{j=1}^{k} \operatorname{sign}(\mathrm{d}\boldsymbol{y}_{k_j}) \boldsymbol{e}_{k_j}, \tag{9}$$

where $\boldsymbol{e}_k$ is the k-th canonical vector. In other words, Δ_k is the (signed) indicator vector of the *first k dominant directions*.

Denote by ℓ the largest index for which $|\mathrm{d}\boldsymbol{y}_\ell| > 0$. Then the first ℓ vectors Δ_k's are linearly independent and they form a basis of the corresponding subspace. Therefore, there exist scalars λ_k's satisfying (3).

Proposition 1. *If* $\lambda_j = |\mathrm{d}\boldsymbol{y}_{k_j}| - |\mathrm{d}\boldsymbol{y}_{k_{j+1}}|$ *for* $j = 1, \ldots, n-1$ *and* $\lambda_n = |\mathrm{d}\boldsymbol{y}_{k_n}|$*, then representation* (3) *holds with* Δ_k*'s as in* (9).

An example of a decomposition is shown in Fig. 3. Further discussion about the choice of basis and various comparisons can be found in the Supplementary material.

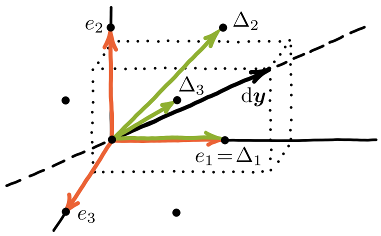


Figure 3: All basis vectors Δ_k (green) point more "towards the $\mathrm{d}\boldsymbol{y}$ direction" compared to the canonical ones (orange).

Constraint parametrization. For learning constraints, we have to specify their parametrization. The representation is of great importance, as it determines how the constraints respond to incoming gradients. Additionally, it affects the meaning of *constraint distance* by changing the parameter space.

We represent each constraint $(\boldsymbol{a}_k, b_k)$ as a hyperplane described by its *normal vector* $\boldsymbol{a}_k$, *distance from the origin* r_k and *offset* o_k of the origin in the global coordinate system as displayed in Fig. 4a. Consequently $b_k = r_k - \boldsymbol{a}_k \cdot o_k$.

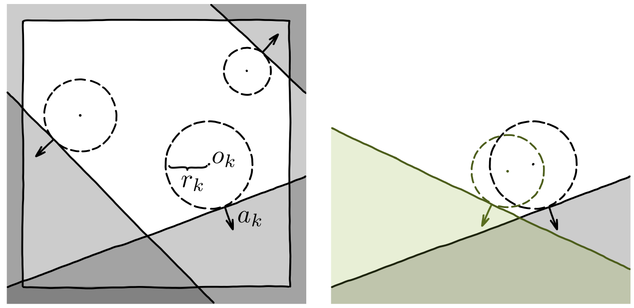
Compared to the plain parametrization which represents the constraints as a matrix $\boldsymbol{A}$ and a vector $\boldsymbol{b}$, our slightly over-parametrized choice allows the constraints to rotate without requiring to traverse large distance in parameter space (consider e.g. a $180°$ rotation). An illustration is displayed in Fig. 4b. Comparison of our choice of parametrization to other encodings and its effect on the performance can be found in the Supplementary material.

4. Demonstration & Analysis

We demonstrate the potential and flexibility of our method on four tasks.

Starting with an extensive performance analysis on synthetic data, we first demonstrate the ability to learn multiple constraints simultaneously. For this, we learn a *static set* of randomly initialized constraints from solved instances, while using access to the *ground-truth* cost vector $\boldsymbol{c}$.

Additionally, we show that the performance of our method on the synthetic datasets also translates to real classes of ILPs. For this we consider a similarly structured task as before, but use the NP-complete WSC problem to generate the dataset.



(a) Constraint representation (b) Possible constraint update

Figure 4: (a) Each constraint $(\boldsymbol{a}_k, b_k)$ is parametrized by its normal vector $\boldsymbol{a}_k$ and a distance r_k to its own origin o_k. (b) Such a representation allows for easy rotations around the learnable offset o_k instead of rotating around the static global origin.

Next, we showcase the ability to simultaneously learn the full ILP specification. For this, we learn a single *input-dependent* constraint and the cost vector *jointly* from the ground truth solutions of KNAPSACK instances. These instances are encoded as sentence embeddings of their description in natural language.

Finally, we demonstrate that our method is also applicable to real-world problems. On the task of keypoint matching, we show that our method achieves results that are comparable to state-of-the-art architectures employing dedicated solvers. In this example, we *jointly* learn a *static set* of constraints and the cost vector from ground-truth matchings.

In all demonstrations, we use GUROBI (Gurobi Optimization, 2019) to solve the ILPs during training and evaluation. Implementation details, a runtime analysis and additional results, such as ablations, other loss functions and more metrics, are provided in the Supplementary material. Additionally, a qualitative analysis of the results for the Knapsack demonstration is included.

4.1. Random Constraints

Problem formulation. The task is to learn the constraints $(\boldsymbol{A}, \boldsymbol{b})$ corresponding to a fixed ILP. The network has only access to the cost vectors $\boldsymbol{c}$ and the ground-truth ILP solutions $\boldsymbol{y}^*$. Note that the set of constraints perfectly explaining the data does not need to be unique.

Dataset. We generate 10 datasets for each cardinality $m = 1, 2, 4, 8$ of the ground-truth constraint set while keeping the dimensionality of the ILP fixed to $n = 16$. Each dataset fixes a set of (randomly chosen) constraints $(\boldsymbol{A}, \boldsymbol{b})$ specifying the ground-truth feasible region of an ILP solver. For the constraints $(\boldsymbol{A}, \boldsymbol{b})$ we then randomly sample cost vectors $\boldsymbol{c}$ and compute the corresponding ILP solution $\boldsymbol{y}^*$ (Fig. 5).

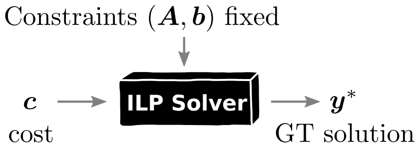


Figure 5: Dataset generation for the RC demonstration.

The dataset consists of 1 600 pairs $(\boldsymbol{c}, \boldsymbol{y}^*)$ for training and 1 000 for testing. The solution space Y is either constrained to $[-5, 5]^n$ (*dense*) or $[0, 1]^n$ (*binary*). During dataset generation, we performed a suitable rescaling to ensure a sufficiently large set of feasible solutions.

Architecture. The network learns the constraints $(\boldsymbol{A}, \boldsymbol{b})$ that specify the ILP solver from ground-truth pairs $(\boldsymbol{c}, \boldsymbol{y}^*)$. Given $\boldsymbol{c}$, predicted solution $\boldsymbol{y}$ is compared to $\boldsymbol{y}^*$ via the MSE loss and the gradient is backpropagated to the learnable constraints using CombOptNet (Fig. 6).

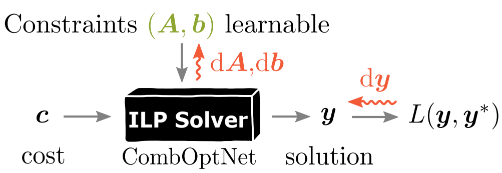
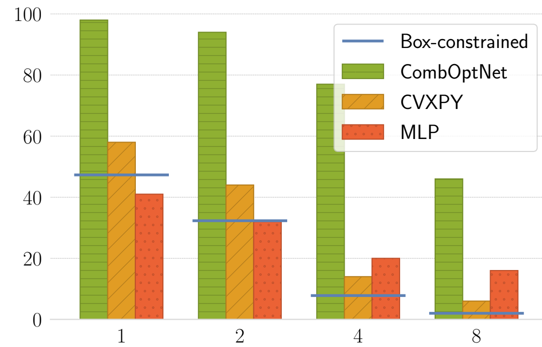


Figure 6: Architecture design for the RC demonstration.

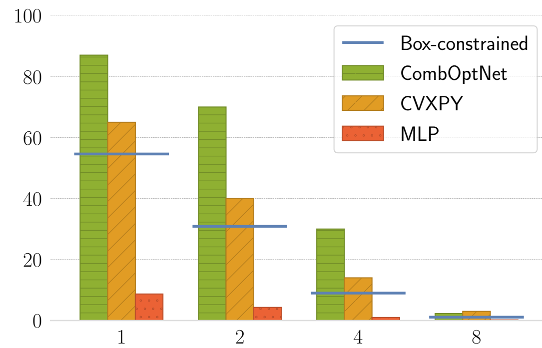
The number of learned constraints matches the number of constraints used for the dataset generation. Note that if the ILP has no feasible solution, the CombOptNet layer output is undefined and any loss or evaluation metric depending on the solution $\boldsymbol{y}$ is meaningless. In practise, updates (5) push the constraints outwards from the true solution $\boldsymbol{y}^*$ leading to a quick emergence of a feasible region.

Baselines. We compare CombOptNet to three baselines. Agnostic to any constraints, a simple MLP baseline directly predicts the solution from the input cost vector as the integer-rounded output of a neural network. The CVXPY baseline uses an architecture similar to ours, only the Module 1 of CombOptNet is replaced with the CVXPY implementation (Diamond & Boyd, 2016) of an LP solver that provides a backward pass proposed by Agrawal et al. (2019a). Similar to our method, it receives constraints and a cost vector and outputs the solution of the LP solver greedily rounded to a feasible integer solution. Finally, we report the performance of always producing the solution of the problem only constrained to the outer region $\boldsymbol{y} \in Y$. This baseline does not involve any training and is purely determined by the dataset.

Results. The results are reported in Fig. 7. In the *binary*, case we demonstrate a high accuracy of perfectly predicting the correct solution. The CVXPY baseline is not capable of matching this, as it is not able to find a set of constraints for the LP problem that mimics the effect of running an ILP solver. For most cost vectors, CVXPY often predicts the same solution as the unconstrained one and its ability to use constraints to improve is marginal. The reason is that the LP relaxation of the ground truth problem is far from tight and thus the LP solver proposes many fractional solutions, which are likely to be rounded incorrectly. This highlights the increased expressivity of the ILP formulation compared to the LP formulation.



(a) Results on the *binary* datasets.



(b) Results on the *dense* datasets.

Figure 7: Results for the Random Constraints demonstration. We report mean accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) over 10 datasets for 1, 2, 4 and 8 ground truth constraints in 16 dimensions. By Box-constrained we denote the performance of always producing the solution of the problem only constrained to the outer region $\boldsymbol{y} \in Y$, which does not involve any training and is purely determined by the dataset.

Even though all methods decrease in performance in the *dense* case as the number of possible solutions is increased, the trend from the *binary* case continues. With the increased density of the solution space, the LP relaxation becomes more similar to the ground truth ILP and hence the gap between CombOptNet and the CVXPY baseline decreases.

We conclude that CombOptNet is especially useful, when the underlying problem is truly difficult (i.e. hard to approximate by an LP). This is not surprising, as CombOptNet introduces structural priors into the network that are designed for hard combinatorial problems.

4.2. Weighted Set Covering

We show that our performance on the synthetic datasets also translates to traditional classes of ILPs. Considering a similarly structured architecture as in the previous section, we generate the dataset by solving instances of the NP-complete WSC problem.

Problem formulation. A family $\mathcal{C}$ of subsets of a universe U is called a covering of U if $\bigcup \mathcal{C} = U$. Given $U = \{1, \ldots, m\}$, its covering $\mathcal{C} = \{S_1, \ldots, S_n\}$ and cost $\boldsymbol{c}\colon \mathcal{C} \to \mathbb{R}$, the task is to find the sub-covering $\mathcal{C}' \subset \mathcal{C}$ with the lowest total cost $\sum_{S \in \mathcal{C}'} \boldsymbol{c}(S)$.

The ILP formulation of this problem consists of m constraints in n dimensions. Namely, if $\boldsymbol{y} \in \{0,1\}^n$ denotes an indicator vector of the sets in $\mathcal{C}$, $a_{kj} = [\![k \in S_j]\!]$ and $b_k = 1$ for $k = 1, \ldots m$, then the specification reads as

$$\min_{\boldsymbol{y} \in Y} \sum_j \boldsymbol{c}(S_j)\boldsymbol{y}_j \qquad \text{subject to} \qquad \boldsymbol{A}\boldsymbol{y} \geq \boldsymbol{b}. \tag{10}$$

Dataset. We randomly draw n subsets from the m-element universe to form a covering $\mathcal{C}$. To increase the variance of solutions, we only allow subsets with no more than 3 elements. As for the Random Constraints demonstration, the dataset consists of 1 600 pairs $(\boldsymbol{c}, \boldsymbol{y}^*)$ for training and 1 000 for testing. Here, $\boldsymbol{c}$ is uniformly sampled positive cost vector and $\boldsymbol{y}^*$ denotes the corresponding optimal solution (Fig. 8). We generate 10 datasets for each universe size $m = 4, 5, 6, 7, 8$ with $n = 2m$ subsets.

Figure 8: Dataset generation for the WSC demonstration.

Architecture and Baselines. We use the same architecture and compare to the same baselines as in the Random Constraints demonstration (Sec. 4.1).

Results. The results are reported in Fig. 9. Our method is still able to predict the correct solution with high accuracy. Compared to the previous demonstration, the performance of the LP relaxation deteriorates. Contrary to the Random Constraints datasets, the solution to the Weighted Set Covering problem never matches the solution of the unconstrained problem, which takes no subset. This prevents the LP relaxation from exploiting these simple solutions and ultimately leads to a performance drop. On the other hand, the MLP

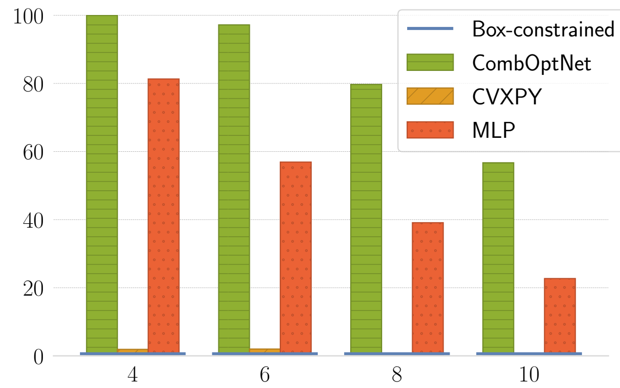


Figure 9: Results of the WSC demonstration. We report mean accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) over 10 datasets for universe sizes $m = 4, 6, 8, 10$ and $2m$ subsets.

baseline benefits from the enforced positivity of the cost vector, which leads to an overall reduced number of different solutions in the dataset.

4.3. KNAPSACK from Sentence Description

Problem formulation. The task is inspired by a vintage text-based PC game called "The Knapsack Problem" (Richardson, 2001) in which a collection of 10 items is presented to a player including their prices and weights. The player's goal is to maximize the total price of selected items without exceeding the fixed 100-pound capacity of their knapsack. The aim is to solve instances of the NP-Hard KNAPSACK problem (2), from their word descriptions. Here, the cost $\boldsymbol{c}$ and the constraint $(\boldsymbol{a}, b)$ are learned simultaneously.

Dataset. Similarly to the game, a KNAPSACK instance consists of 10 sentences, each describing one item. The sentences are preprocessed via the sentence embedding (Conneau et al., 2017) and the 10 resulting 4 096-dimensional vectors $\boldsymbol{x}$ constitute the input of the dataset. We rely on the ability of natural language embedding models to capture numerical values, as the other words in the sentence are uncorrelated with them (see an analysis of Wallace et al. (2019)). The indicator vector $\boldsymbol{y}^*$ of the optimal solution (i.e. item selection) to a knapsack instance is its corresponding label (Fig. 10). The dataset contains 4 500 training and 500 test pairs $(\boldsymbol{x}, \boldsymbol{y}^*)$.

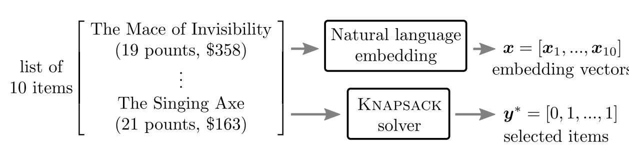


Figure 10: Dataset generation for the KNAPSACK problem.

Architecture. We simultaneously extract the learnable constraint coefficients $(\boldsymbol{a}, \boldsymbol{b})$ and the cost vector $\boldsymbol{c}$ via an MLP from the embedding vectors (Fig. 11).

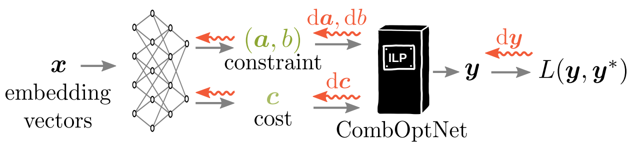


Figure 11: Architecture design for the KNAPSACK problem.

As only a single learnable constraint is used, which by definition defines a KNAPSACK problem, the interpretation of this demonstration is a bit different from the other demonstrations. Instead of learning the type of combinatorial problem, we learn which exact KNAPSACK problem in terms of item-weights and knapsack capacity needs to be solved.

Baselines. We compare to the same baselines as in the Random Constraints demonstration (Sec. 4.1).

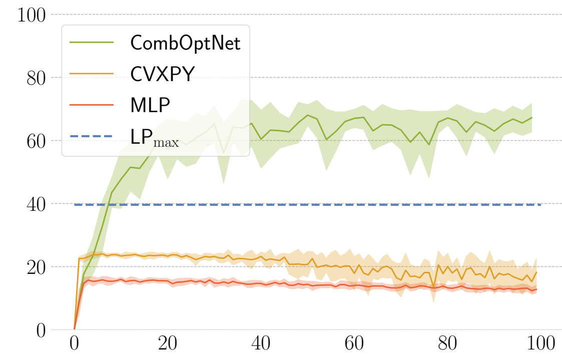
Results. The results are presented in Fig. 12. While CombOptNet is able to predict the correct items for the KNAPSACK with good accuracy, the baselines are unable to match this. Additionally, we evaluate the LP relaxation on the ground truth weights and prices, providing an upper bound for results achievable by any method relying on an LP relaxation. The weak performance of this evaluation underlines the NP-Hardness of KNAPSACK. The ability to embed and differentiate through a dedicated ILP solver leads to surpassing this threshold even when learning from imperfect raw inputs.

4.4. Deep Keypoint Matching

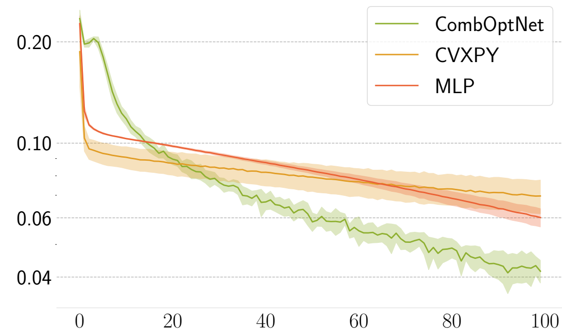
Problem formulation. Given are a source and target image showing an object of the same class (e.g. *airplane*), each labeled with a set of annotated keypoints (e.g. *left wing*). The task is to find the correct matching between the sets of keypoints from visual information without access to the keypoint annotation. As not every keypoint has to be visible in both images, some keypoints can also remain unmatched.

As in this task the combinatorial problem is known a priori, state-of-the-art methods are able to exploit this knowledge by using dedicated solvers. However, in our demonstration we make the problem harder by omitting this knowledge. Instead, we *simultaneously* infer the problem specification *and* train the feature extractor for the cost vector from data end-to-end.

Dataset. We use the SPair-71k dataset (Min et al., 2019) which was published in the context of dense image matching and was used as a benchmark for keypoint matching in



(a) Evaluation accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) over training epochs. $\mathrm{LP}_{\mathrm{max}}$ is the maximum achievable LP relaxation accuracy.



(b) Training MSE loss over epochs.

Figure 12: Results or KNAPSACK demonstration. Reported error bars are over 10 restarts.

recent literature (Rolínek et al., 2020b). It includes 70 958 image pairs prepared from Pascal VOC 2012 and Pascal 3D+ with rich pair-level keypoint annotations. The dataset is split into 53 340 training pairs, 5 384 validation pairs and 12 234 pairs for testing.

State-of-the-art. We compare to a state-of-the-art architecture BB-GM (Rolínek et al., 2020b) that employs a dedicated solver for the quadratic assignment problem. The solver is made differentiable with blackbox backpropagation (Vlastelica et al., 2020a), which allows to differentiate through the solver with respect to the input cost vector.

Table 1: Results for the keypoint matching demonstration. Reported is the standard per-variable accuracy $(\%)$ metric over 5 restarts. Column $p \times p$ corresponds to matching p source keypoints to p target keypoints.

Method	4×4	5×5	6×6	7×7
CombOptNet	83.1	80.7	78.6	76.1
BB-GM	84.3	82.9	80.5	79.8

Architecture. We modify the BB-GM architecture by replacing the blackbox-differentiation module employing the dedicated solver with CombOptNet.

The drop-in replacement comes with a few important considerations. Note that our method relies on a fixed dimensionality of the problem for learning a static (i.e. not input-dependent) constraint set. Thus, we can not learn an algorithm that is able to match any number of keypoints to any other number of keypoints, as the dedicated solver in the baseline does.

Due to this, we train four versions of our architecture, setting the number of keypoints in both source and target images to $p = 4, 5, 6, 7$. In each version, the dimensionality is fixed to the number of edges in the bipartite graph. We use the same number of learnable constrains as the number of ground-truth constraints that would realize the ILP representation of the proposed mathching problem, i.e. the combined number of keypoints in both images $(m = 2p)$.

The randomly initialized constraint set and the backbone architecture that produces the cost vectors $\boldsymbol{c}$ are learned simultaneously from pairs of predicted solutions $\boldsymbol{y}$ and ground-truth matchings $\boldsymbol{y}^*$ using CombOptNet.

Results. The results are presented in Tab. 1. Even though CombOptNet is uninformed about which combinatorial problem it should be solving, its performance is close to the privileged state-of-the-art method BB-GM. These results are especially satisfactory, considering the fact that BB-GM outperforms the previous state-of-the-art architecture (Fey et al., 2020) by several percentage points on experiments of this difficulty. Example matchings are shown in Fig. 13.

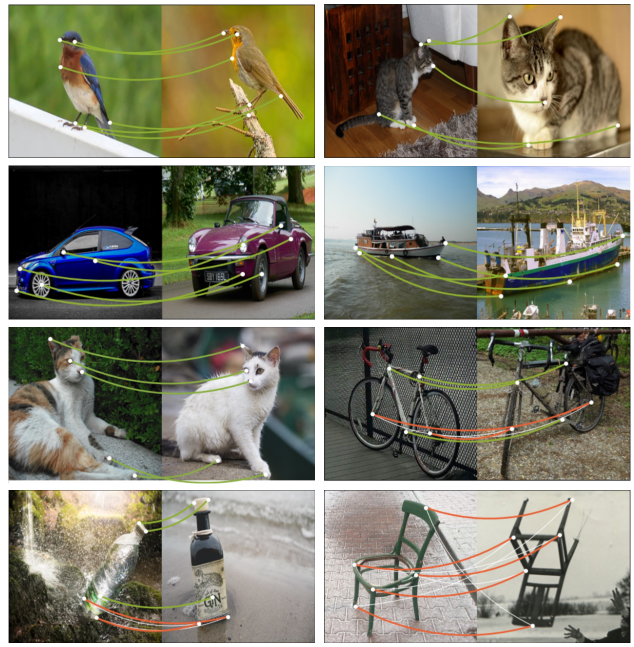


Figure 13: Example matchings predicted by CombOptNet.

5. Conclusion

We propose a method for integrating integer linear program solvers into neural network architectures as layers. This is enabled by providing gradients for both the *cost terms* and the *constraints* of an ILP. The resulting end-to-end trainable architectures are able to *simultaneously* extract features from raw data and learn a suitable set of constraints that specify the combinatorial problem. Thus, the architecture learns to fit the right NP-hard problem needed to solve the task. In that sense, it strives to achieve universal combinatorial expressivity in deep networks – opening many exciting perspectives.

In the experiments, we demonstrate the flexibility of our approach, using different input domains, natural language and images, and different combinatorial problems with the same CombOptNet module. In particular, for combinatorially hard problems we see a strong advantage of the new architecture.

The potential of our method is highlighted by the demonstration on the keypoint matching benchmark. Unaware of the underlying combinatorial problem, CombOptNet achieves a performance that is not far behind architectures employing dedicated state-of-the-art solvers.

In future work, we aim to make the number of constraints flexible and to explore more problems with hybrid combinatorial complexity and statistical learning aspects.

Acknowledgements

Georg Martius is a member of the Machine Learning Cluster of Excellence, funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. We acknowledge the support from the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B). This work was supported from Operational Programme Research, Development and Education – Project Postdoc2MUNI (No. CZ.02.2.69/0.0/0.0/18_053/0016952)

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pp. 9562–9574, 2019a.

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

Amos, B. *Differentiable optimization-based modeling for machine learning*. PhD thesis, PhD thesis. Carnegie Mellon University, 2019.

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

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

Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. Learning to branch. In *International conference on machine learning*, pp. 344–353, 2018.

Balunovic, M., Bielik, P., and Vechev, M. Learning to solve SMT formulas. In *Advances in Neural Information Processing Systems*, pp. 10317–10328, 2018.

Battaglia, P., Hamrick, J. B. C., Bapst, V., Sanchez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K., Nash, C., Langston, V. J., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv:1611.09940*, 2016.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable perturbed optimizers. In *Advances in Neural Information Processing Systems*, pp. 9508–9519, 2020.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. Supervised learning of universal sentence representations from natural language inference data. In *Conference on Empirical Methods in Natural Language Processing*, pp. 670–680, Copenhagen, Denmark, 2017. Association for Computational Linguistics.

Cplex, I. I. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. Safe exploration in continuous action spaces. *arXiv:1801.08757*, 2018.

Diamond, S. and Boyd, S. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Djolonga, J. and Krause, A. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pp. 1013–1023, 2017.

Domke, J. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

Donti, P., Amos, B., and Kolter, J. Z. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, pp. 5484–5494, 2017.

Elmachtoub, A. N. and Grigas, P. Smart "predict, then optimize". *arXiv:1710.08005*, 2020.

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

Fey, M., Lenssen, J. E., Morris, C., Masci, J., and Kriege, N. M. Deep graph matching consensus. In *International Conference on Learning Representations*, 2020.

Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., and Guo, E. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv:1607.05447*, 2016.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv:1410.5401*, 2014.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471–476, October 2016.

Gurobi Optimization, L. Gurobi optimizer reference manual, 2019. URL http://www.gurobi.com.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pp. 6348–6358, 2017.

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

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. Meta-learning with differentiable convex optimization. In *Conference on Computer Vision and Pattern Recognition*, pp. 10657–10665, 2019.

Ling, C. K., Fang, F., and Kolter, J. Z. What game are we playing? End-to-end learning in normal and extensive form games. In *International Joint Conference on Artificial Intelligence*, 2018.

Min, J., Lee, J., Ponce, J., and Cho, M. SPair-71k: A Large-scale Benchmark for Semantic Correspondence. *arXiv:1908.10543*, 2019.

Nair, V., Bartunov, S., Gimeno, F., von Glehn, I., Lichocki, P., Lobov, I., O'Donoghue, B., Sonnerat, N., Tjandraatmadja, C., Wang, P., Addanki, R., Hapuarachchi, T., Keck, T., Keeling, J., Kohli, P., Ktena, I., Li, Y., Vinyals, O., and Zwols, Y. Solving mixed integer programs using neural networks. *arXiv:2012.13349*, 2020.

Nazari, M., Oroojlooy, A., Snyder, L., and Takác, M. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pp. 9839–9849, 2018.

Niculae, V., Martins, A., Blondel, M., and Cardie, C. Sparsemap: Differentiable sparse structured inference. In *International Conference on Machine Learning*, pp. 3799–3808, 2018.

Richardson, L. The knapsack problem, the game of premature optimization, 2001. URL https://www.crummy.com/software/if/knapsack/.

Rolínek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., and Martius, G. Optimizing ranking-based metrics with blackbox differentiation. In *Conference on Computer Vision and Pattern Recognition*, 2020a.

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

Tan, Y., Terekhov, D., and Delong, A. Learning linear programs from optimal decisions. In *Advances in Neural Information Processing Systems*, pp. 19738–19749, 2020.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020.

Vlastelica, M., Paulus, A., Musil, V., Martius, G., and Rolínek, M. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020a.

Vlastelica, M., Rolinek, M., and Martius, G. Discrete planning with end-to-end trained neuro-algorithmic policies. *ICML 2020, Graph Representation Learning Workshop*, 2020b.

Wallace, E., Wang, Y., Li, S., Singh, S., and Gardner, M. Do NLP models know numbers? Probing numeracy in embeddings. In *Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 5310–5318, 2019.

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

Wilder, B., Ewing, E., Dilkina, B., and Tambe, M. End to end learning and optimization on graphs. In *Advances in Neural Information Processing Systems*, pp. 4672–4683, 2019.

Zanfir, A. and Sminchisescu, C. Deep learning of graph matching. In *Conference on Computer Vision and Pattern Recognition*, pp. 2684–2693, 2018.

Zhang, W. and Dietterich, T. G. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Reseach*, 1:1–38, 2000.