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]

[1] Max-Planck-Institute for Intelligent Systems, Tübingen, Germany
[2] Masaryk University, Brno, Czechia
[3] Facebook AI Research, USA

anselm.paulus@tuebingen.mpg.de

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.

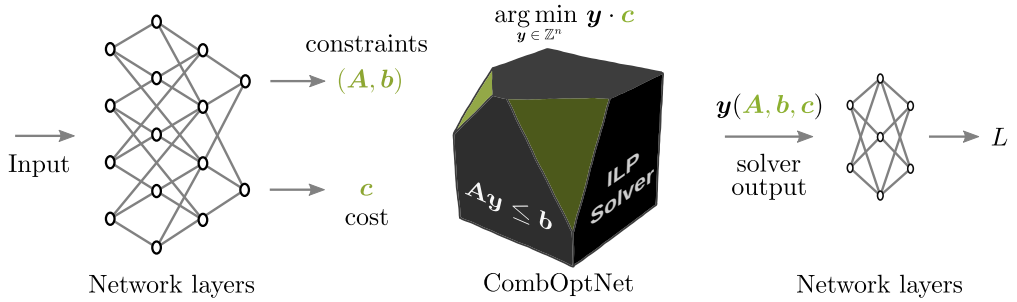


Figure 1: CombOptNet as a module in a deep architecture.

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 [44, 46]; b) adjust the topology of neural network architectures to express certain algorithmic behaviour [8, 23, 24]; c) provide an informative gradient approximation for the discrete algorithm [10, 41]. 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 Berthet et al. [10], Ferber et al. [20], Vlastelica et al. [41], 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 [9, 26, 33, 47], learning graph-based algorithms [39, 40, 45], learning to branch [6], solving SMT formulas [7] and TSP instances [28]. Nair et al. [32] have recently scaled up learned MIP solvers on non-trivial production datasets. In a more general computational paradigm, Graves et al. [23, 24] 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 [17, 22]. This approach has been recently studied in the convex setting in OptNet [4] for quadratic programs, and more general cone programs in Amos [3, Section 7.3] and Agrawal et al. [1, 2]. One use of this paradigm is to incorporate the knowledge of a downstream optimization-based task into a predictive model [18, 19]. Extending beyond the convex setting, optimization-based modeling and differentiable optimization are used for sparse structured inference [34], MAXSAT [44], submodular optimization [16] mixed integer programming [20], and discrete and combinational settings [10, 41]. Applications of optimization-based modeling include computer vision [36, 37], reinforcement learning [5, 14, 42], game theory [30], and inverse optimization [38], and meta-learning [11, 29].

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, $\boldsymbol{A}=[\boldsymbol{a}_1,\dots,\boldsymbol{a}_m]\in\mathbb{R}^{m\times n}$ 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 $\boldsymbol{c}=[c_1,\dots,c_n]\in\mathbb{R}^n$ are the prices of the items, $\boldsymbol{a}=[a_1,\dots,a_n]\in\mathbb{R}^n$ 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 [13, 25] 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. 19, 20, 41]. 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 A1 in the Appendix, 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 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

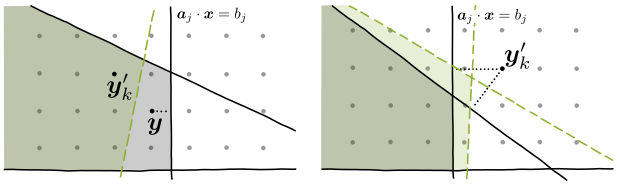
$$P_{\Delta_k}(\boldsymbol{A}, \boldsymbol{b}) = \begin{cases} \min_j \operatorname{dist}(\boldsymbol{a}_j, b_j; \boldsymbol{y}) \\ \qquad \text{if } \boldsymbol{y}'_k \text{ is feasible and } \boldsymbol{y}'_k \neq \boldsymbol{y} \\ \sum_j [\![\boldsymbol{a}_j \cdot \boldsymbol{y}'_k > b_j]\!] \operatorname{dist}(\boldsymbol{a}_j,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

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 Appendix.

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.



(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".

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 [41] or ε in [10]). 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 [19] 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 $k_1, \ldots, k_n$ 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).*

Module 1 CombOptNet

function FORWARDPASS($\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}$)
  $\boldsymbol{y} :=$ **Solver**($\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}$)
  save $\boldsymbol{y}$ and $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}$ for backward pass
  return $\boldsymbol{y}$
function BACKWARDPASS($\mathrm{d}\boldsymbol{y}$)
  load $\boldsymbol{y}$ and $\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}$ from forward pass
  Decompose $\mathrm{d}\boldsymbol{y} = \sum_k \lambda_k \Delta_k$
    // set Δ_k as in (9) and λ_k as in Proposition 1
  Calculate the gradients
    $\mathrm{d}\boldsymbol{A}^k := \frac{\partial P_{\Delta_k}}{\partial \boldsymbol{A}}, \mathrm{d}\boldsymbol{b}^k := \frac{\partial P_{\Delta_k}}{\partial \boldsymbol{b}}, \mathrm{d}\boldsymbol{c}^k := \frac{\partial P_{\Delta_k}}{\partial \boldsymbol{c}}$
    // P_{Δ_k} defined in (5) and (7)
  Compose $\mathrm{d}\boldsymbol{A}, \mathrm{d}\boldsymbol{b}, \mathrm{d}\boldsymbol{c} := \sum_k \lambda_k \left(\mathrm{d}\boldsymbol{A}^k, \mathrm{d}\boldsymbol{b}^k, \mathrm{d}\boldsymbol{c}^k\right)$
    // According to (6)
  return $\mathrm{d}\boldsymbol{A}, \mathrm{d}\boldsymbol{b}, \mathrm{d}\boldsymbol{c}$

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 Appendix.

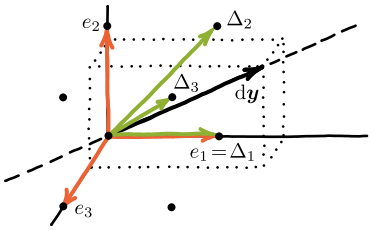


Figure 3: All basis vectors Δ_k (green) point more "towards the $\mathrm{d}\boldsymbol{y}$ direction" compared to the canonical ones (orange).

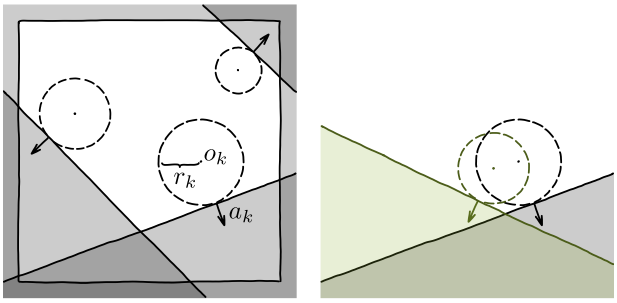
Constraint parametrization. For learning constraints, we have to specify their parametrization. The representation is of great importance, as it determines how the constraints respond to incoming gradients. Additionally, it affects the meaning of *constraint distance* by changing the parameter space.

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 overparametrized 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 Appendix.

4 Demonstration & Analysis

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



(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.

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.

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 [25] 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 Appendix. 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).

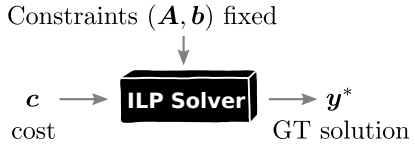


Figure 5: Dataset generation for the RC demonstration.

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).

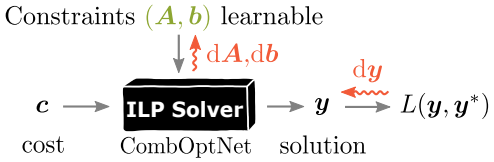
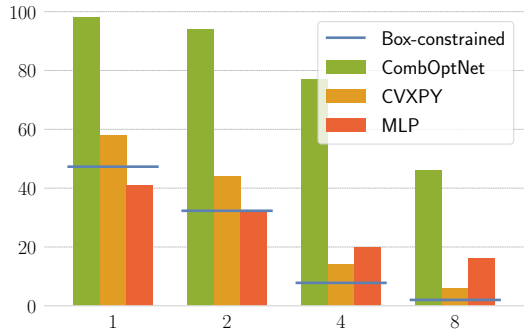


Figure 6: Architecture design for the RC demonstration.

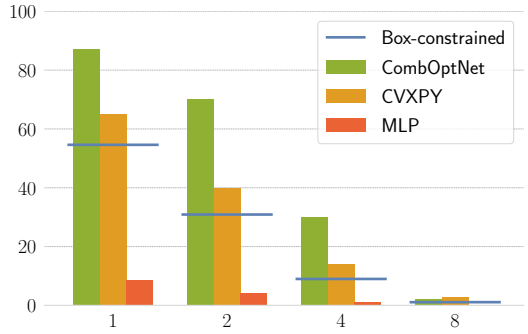
The number of learned constraints matches the number of constraints used for the dataset generation.

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 [15] of an LP solver that provides a backward pass proposed by Agrawal et al. [1]. 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.



(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.

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.

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 $U = \{1, \dots, m\}$, its covering $\mathcal{C} = \{S_1, \dots, S_n\}$ 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 $k = 1, \dots m$, 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 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" [35] 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.

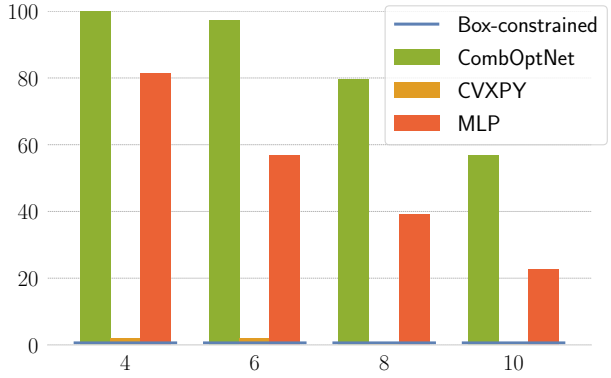


Figure 9: Results of the WSC demonstration. We report mean accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) over 10 datasets for universe sizes $m = 4, 6, 8, 10$ and $2m$ subsets.

Dataset. Similarly to the game, a KNAPSACK instance consists of 10 sentences, each describing one item. The sentences are preprocessed via the sentence embedding [12] 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. [43]). 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 $(\boldsymbol{x}, \boldsymbol{y}^*)$.

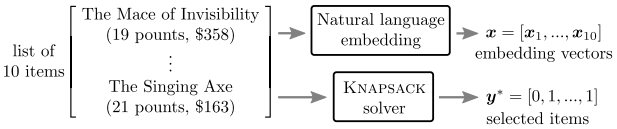


Figure 10: Dataset generation for the KNAPSACK problem.

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).

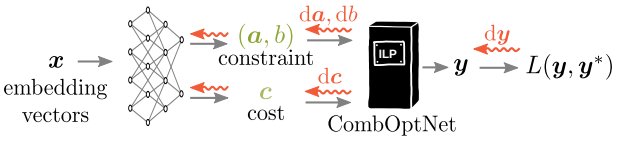
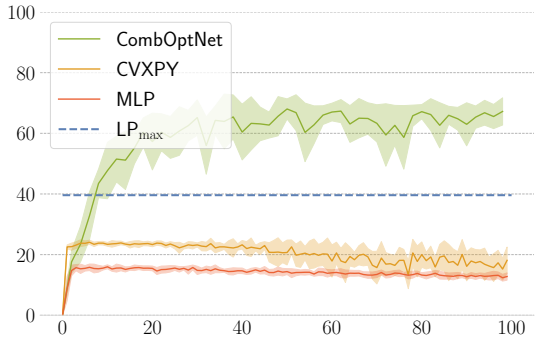


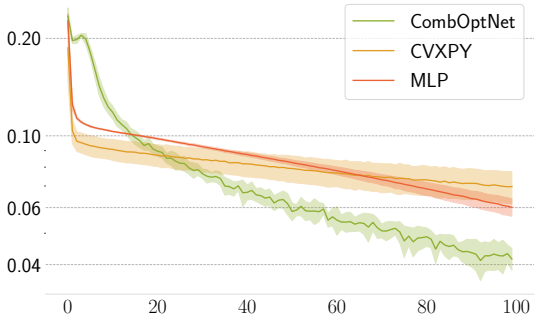
Figure 11: Architecture design for the KNAPSACK problem.

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).



(a) Evaluation accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) over training epochs. LP$_{\text{max}}$ 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.

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

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

and train the feature extractor for the cost vector from data end-to-end.

Dataset. We use the SPair-71k dataset [31] which was published in the context of dense image matching and was used as a benchmark for keypoint matching in recent literature [37]. 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 [37] that employs a dedicated solver for the quadratic assignment problem. The solver is made differentiable with blackbox backpropagation [41], which allows to differentiate through the solver with respect to the input cost vector.

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 amount 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 [21] by several percentage points on experiments of this difficulty. Example matchings are shown in Fig. 13.

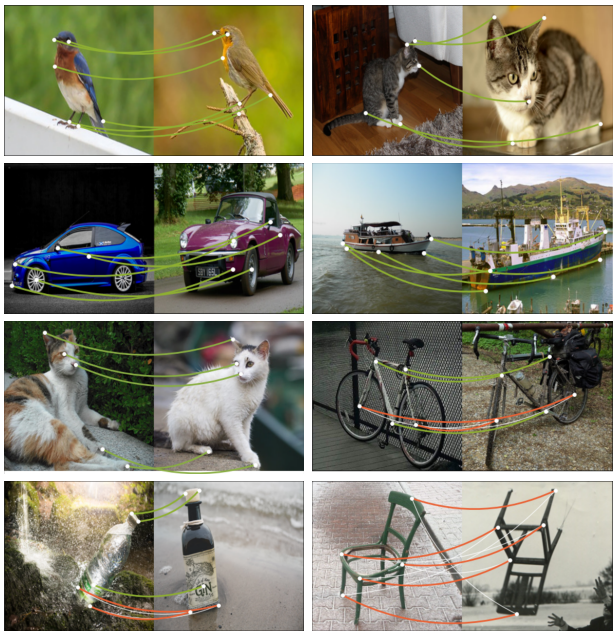


Figure 13: Example matchings predicted by CombOptNet.

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.

A Demonstrations

A.1 Implementation Details

When learning multiple constraints, we replace the minimum in definition (5) of mismatch function P_{Δ_k} with its softened version. Therefore, not only the single closest constraint will shift towards $\boldsymbol{y}'_k$, but also other constraints close to $\boldsymbol{y}'_k$ will do. For the softened minimum we use

$$\operatorname{softmin}(\boldsymbol{x}) = -\tau \cdot \log\left(\sum_k \exp\left(-\frac{x_k}{\tau}\right)\right), \qquad \text{(A1)}$$

which introduces the temperature τ, determining the softening strength.

In all experiments, we normalize the cost vector $\boldsymbol{c}$ before we forward it to the CombOptNet module. For the loss we use the mean squared error between the normalized predicted solution $\boldsymbol{y}$ and the normalized ground-truth solution $\boldsymbol{y}^*$. For normalization we apply the shift and scale that translates the underlying hypercube of possible solutions ($[0,1]^n$ in *binary* or $[-5,5]^n$ in *dense* case) to a normalized hypercube $[-0.5, 0.5]^n$.

The hyperparameters for all demonstrations are listed in Tab. A1. We use Adam [27] as the optimizer for all demonstrations.

Random Constraints. For selecting the set of constraints for data generation, we uniformly sample constraint origins $\boldsymbol{o}_k$ in the center subcube (halved edge length) of the underlying hypercube of possible solutions. The constraint normals $\boldsymbol{a}_k$ and the cost vectors $\boldsymbol{c}$ are randomly sampled normalized vectors and the bias terms are initially set to $b_k = 0.2$. The signs of the constraint normals $\boldsymbol{a}_k$ are flipped in case the origin is not feasible, ensuring that the problem has at least one feasible solution. We generate 10 such datasets for $m = 1, 2, 4, 8$ constraints in $n = 16$ dimensions. The size of each dataset is 1 600 train instances and 1 000 test instances.

For learning, the constraints are initialised in the same way except for the feasibility check, which is skipped since CombOptNet can deal with infeasible regions itself.

KNAPSACK from Sentence Description. Our method and CVXPY use a small neural network to extract weights and prices from the 4 096-dimensional embedding vectors. We use a two-layer MLP with a hidden dimension of 512, ReLU nonlinearity on the hidden nodes, and a sigmoid nonlinearity on the output. The output is scaled to the ground-truth price range $[10, 45]$ for the cost $\boldsymbol{c}$ and to the ground-truth weight range $[15, 35]$ for the constraint $\boldsymbol{a}$. The bias term is fixed to the ground-truth knapsack capacity $b = 100$. Item weights and prices as well as the knapsack capacity are finally multiplied by a factor of 0.01 to produce a reasonable scale for the constraint parameters and cost vector.

Table A1: Hyperparameters for all demonstrations.

	WSC & Random Constraints	Knapsack	Keypoint Matching
Learning rate	5×10^{-4}	5×10^{-4}	1×10^{-4}
Batch size	8	8	8
Train epochs	100	100	10
τ	0.5	0.5	0.5
Backbone lr	–	–	2.5×10^{-6}

The CVXPY baseline implements a greedy rounding procedure to ensure the feasibility of the predicted integer solution with respect to the learned constraints. Starting from the item with the largest predicted (noninteger) value, the procedure adds items to the predicted (integer) solution until no more items can be added without surpassing the knapsack capacity.

The MLP baseline employs an MLP consisting of three layers with dimensionality 100 and ReLU activation on the hidden nodes. Without using an output nonlinearity, the output is rounded to the nearest integer point to obtain the predicted solution.

Deep Keypoint Matching. We initialize a set of constraints exactly as in the *binary* case of the Random Constraints demonstration. We use the architecture described by Rolínek et al. [37], only replacing the dedicated solver module with CombOptNet.

We train models for varying numbers of keypoints $p = 4, 5, 6, 7$ in the source and target image, resulting in varying dimensionalities $n = p^2$ and number of constraints $m = 2p$. Consistent with Rolínek et al. [37], all models are trained for 10 epochs, each consisting of 400 iterations with randomly drawn samples from the training set. We discard samples with fewer keypoints than what is specified for the model through the dimensionality of the constraint set. If the sample has more keypoints, we chose a random subset of the correct size.

After each epoch, we evaluate the trained models on the validation set. Each model's highest-scoring validation stage is then evaluated on the test set for the final results.

A.2 Runtime analysis.

The runtimes of our demonstrations are reported in Tab. A2. Random Constrains demonstrations have the same runtimes as Weighted Set Covering since they share the architecture.

Unsurprisingly, CombOptNet has higher runtimes as it relies on ILP solvers which are generally slower than LP solvers. Also, the backward pass of CombOptNet has negligible runtime compared to the forward-pass runtime. In Random Constraints, Weighted Set Covering and KNAPSACK demonstration, the increased runtime is necessary, as the baselines simply do not solve a hard enough problem to succeed in the tasks.

Table A2: Average runtime for training and evaluating a model on a single Tesla-V100 GPU. For Keypoint Matching, the runtime for the largest model ($p = 7$) is shown.

	Weighted Set Covering	Knapsack	Keypoint Matching
CombOptNet	1h 30m	3h 50m	5h 30m
CVXPY	1h	2h 30m	–
MLP	10m	20m	–
BB-GM	–	–	55m

Table A3: Random Constraints demonstration with multiple learnable constraints. Using a dataset with m ground-truth constraints, we train a model with $k \times m$ learnable constraints. Reported is evaluation accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) for $m = 1, 2, 4, 8$. Statistics are over 20 restarts (2 for each of the 10 dataset seeds).

	m	1	2	4	8
binary	$1 \times m^\star$	**97.8 ± 0.7**	94.2 ± 10.1	77.4 ± 13.5	46.5 ± 12.4
	$2 \times m$	97.3 ± 0.9	95.1 ± 1.6	87.8 ± 5.2	63.1 ± 7.0
	$4 \times m$	96.9 ± 0.7	**95.1 ± 1.2**	**88.7 ± 2.3**	**77.7 ± 3.2**
dense	$1 \times m^\star$	87.3 ± 2.5	70.2 ± 11.6	29.6 ± 10.4	2.3 ± 1.2
	$2 \times m$	**87.8 ± 1.7**	**73.4 ± 2.4**	**32.7 ± 7.6**	2.4 ± 0.8
	$4 \times m$	85.0 ± 2.6	64.6 ± 3.9	28.3 ± 2.7	**2.9 ± 1.3**

In the Keypoint Matching demonstration, CombOptNet slightly drops behind BB-GM and requires higher runtime. Such drawback is outweighed by the benefit of employing a broad-expressive model that operates without embedded knowledge of the underlying combinatorial task.

A.3 Additional Results

Random Constraints & Weighted Set Covering. We provide additional results regarding the increased amount of learned constraints in Tab. A3 and A4) and the choice of the loss function Tab. A5.

With a larger set of learnable constraints the model is able to construct a more complex feasible region. While in general this tends to increase performance and reduce variance by increasing robustness to bad initializations, it can also lead to overfitting similarly to a neural net with too many parameters.

In the *dense* case, we also compare different loss functions which is possible because CombOptNet can be used as an arbitrary layer. As shown in Tab. A5, this choice matters, with the MSE loss, the L1 loss and the Huber loss outperforming the L0 loss. This freedom of loss function choice can prove very helpful for training more complex architectures.

KNAPSACK from Sentence Description. As for the Random Constraints demonstration, we report the performance of CombOptNet on the KNAPSACK task for a higher number of learnable constraints. The results are listed in Tab. A6. Similar to the *binary* Random Constraints ablation with $m = 1$, increasing the number of learnable constraints does not result in strongly increased performance.

Additionally, we provide a qualitative analysis of the results on the KNAPSACK task. In Fig. A1 we compare the total ground-truth price of the predicted instances to the total price of the ground-truth solutions on a single evaluation of the trained models.

The plots show that CombOptNet is achieving much better results than CVXPY. The total prices of the predictions are very close to the optimal prices and only a few predictions are infeasible, while CVXPY tends to predict infeasible solutions and only a few predictions have objective values matching the optimum.

In Fig. A2 we compare relative errors on the individual item weights and prices on the same evaluation of the trained models as before. Since (I)LP costs are scale invariant, we normalize predicted price vector to match the size of the ground-truth price vector before the comparison.

CombOptNet shows relatively small normally distributed errors on both prices and weights, precisely as expected from the prediction of a standard network. CVXPY reports much larger relative errors on both prices and weights (note the different plot scale). The vertical lines correspond to the discrete steps of ground-truth item weights in the dataset. Unsurprisingly, the baseline usually tends to either

Table A4: Weighted set covering demonstration with multiple learnable constraints.

k	4	6	8	10
$1^\star$	100 ± 0.0	97.2 ± 6.4	79.7 ± 12.1	56.7 ± 14.8
2	100 ± 0.0	99.5 ± 1.9	**99.3 ± 0.8**	80.4 ± 13.0
4	100 ± 0.0	**99.9 ± 0.0**	97.9 ± 6.4	**85.2 ± 8.1**

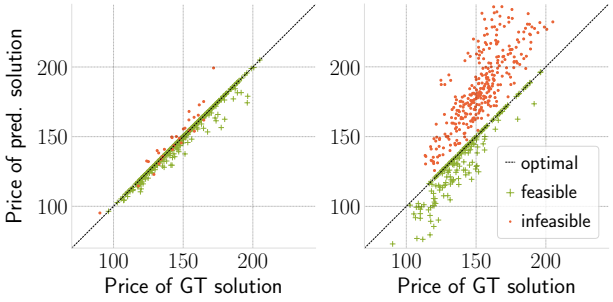
Table A5: Random Constraints *dense* demonstration with various loss functions. For the Huber loss we set $\beta = 0.3$. Statistics are over 20 restarts (2 for each of the 10 dataset seeds).

Loss	1	2	4	8
MSE*	87.3 ± 2.5	70.2 ± 11.6	29.6 ± 10.4	2.3 ± 1.2
Huber	88.3 ± 4.0	75.4 ± 9.3	25.0 ± 11.8	**2.6 ± 2.7**
L0	85.9 ± 3.4	65.8 ± 3.5	15.3 ± 4.3	1.1 ± 0.3
L1	**89.2 ± 1.6**	**75.8 ± 10.8**	**30.2 ± 16.5**	2.1 ± 1.2

Table A6: Knapsack demonstration with more learnable constraints. Reported is evaluation accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) for $m = 1, 2, 4, 8$ constraints. Statistics are over 10 restarts.

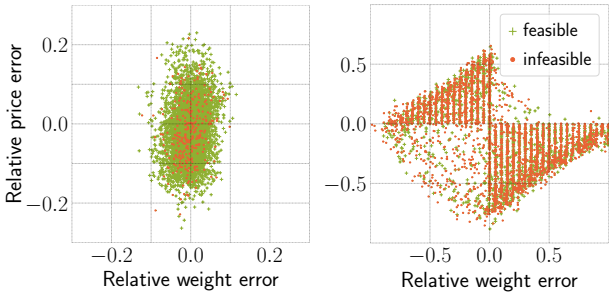
$1^\star$	2	4	8
64.7 ± 2.8	63.5 ± 3.7	**65.7 ± 3.1**	62.6 ± 4.4

*Used in the main demonstrations.



(a) CombOptNet (b) CVXPY

Figure A1: Prices analysis for the KNAPSACK demonstration. For each test set instance, we plot the total price of the predicted solution over the total price of the ground-truth solution. Predicted solutions which total weight exceeds the knapsack capacity are colored in red (cross).



(a) CombOptNet (b) CVXPY

Figure A2: Qualitative analysis of the errors on weights and prices in the KNAPSACK demonstration. We plot the relative error between predicted and ground-truth item prices over the relative error between predicted and ground-truth item weights. Colors denote whether the predicted solution is feasible in terms of ground-truth weights.

overestimate the price and underestimate the item weight, or vice versa, due to similar effects of these errors on the predicted solution.

A.4 Ablations

We ablate the choices in our architecture and model design on the Random Constraints (RC) and Weighted Set Covering (WSC) tasks. In Tab. A7 and A8 we report constraint parametrization, choice of basis, and minima softening ablations.

The ablations show that our parametrization with learnable origins is consistently among the best ones. Without learnable origins, the performance is highly dependend on the origin of the coordinate system in which the directly learned parameters $(\boldsymbol{A}, \boldsymbol{b})$ are defined.

The choice of basis in the gradient decomposition shows a large impact on performance. Our basis Δ (9) is outperforming the canonical one in the *binary* RC and WSC demonstration, while showing performance similar to the canonical basis in the *dense* RC case. The canonical basis produces directions for the computation of $\boldsymbol{y}'_k$ that in many cases point in very different directions than the incoming descent direction. As a result, the gradient computation leads to updates that are very detached from the original incoming gradient.

Finally, the softened minimum leads to increased performance in all demonstrations. This effect is apparent particularly in the case of a binary solution space, as the constraints can have a relevant impact on the predicted solution $\boldsymbol{y}$ over large distances. Therefore, only updating the constraint which is closest to the predicted solution $\boldsymbol{y}$, as it is the case for a hard minimum, gives no gradient to constraints that may potentially have had a huge influence on $\boldsymbol{y}$.

B Method

To recover the situation from the method section, set $\boldsymbol{x}$ as one of the inputs $\boldsymbol{A}$, $\boldsymbol{b}$, or $\boldsymbol{c}$.

Proposition A1. *Let $\boldsymbol{y}\colon \mathbb{R}^\ell \to \mathbb{R}^n$ be differentiable at $\boldsymbol{x} \in \mathbb{R}^\ell$ and let $L\colon \mathbb{R}^n \to \mathbb{R}$ be differentiable at $\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}) \in \mathbb{R}^n$. Denote $\mathrm{d}\boldsymbol{y} = \partial L/\partial \boldsymbol{y}$ at $\boldsymbol{y}$. Then the distance between $\boldsymbol{y}(\boldsymbol{x})$ and $\boldsymbol{y} - \mathrm{d}\boldsymbol{y}$ is minimized along the direction $\partial L/\partial \boldsymbol{x}$, where $\partial L/\partial \boldsymbol{x}$ stands for the derivative of $L(\boldsymbol{y}(\boldsymbol{x}))$ at $\boldsymbol{x}$.*

Proof. For $\xi \in \mathbb{R}^\ell$, let $\varphi(\xi)$ denote the distance between $\boldsymbol{y}(\boldsymbol{x} - \xi)$ and the target $\boldsymbol{y}(\boldsymbol{x}) - \mathrm{d}\boldsymbol{y}$, i.e.

$$\varphi(\xi) = \big\|\boldsymbol{y}(\boldsymbol{x} - \xi) - \boldsymbol{y}(\boldsymbol{x}) + \mathrm{d}\boldsymbol{y}\big\|.$$

There is nothing to prove when $\mathrm{d}\boldsymbol{y} = 0$ as $\boldsymbol{y}(x) = \boldsymbol{y} - \mathrm{d}\boldsymbol{y}$ and there is no room for any improvement. Otherwise, φ is positive and differentiable in a neighborhood of zero. The Fréchet derivative of φ reads as

$$\varphi'(\xi) = \frac{-\big[\boldsymbol{y}(\boldsymbol{x} - \xi) - \boldsymbol{y}(\boldsymbol{x}) + \mathrm{d}\boldsymbol{y}\big] \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}(\boldsymbol{x} - \xi)}{\big\|\boldsymbol{y}(\boldsymbol{x} - \xi) - \boldsymbol{y}(\boldsymbol{x}) + \mathrm{d}\boldsymbol{y}\big\|},$$

whence

$$\varphi'(0) = -\frac{1}{\|\mathrm{d}\boldsymbol{y}\|}\frac{\partial L}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = -\frac{1}{\|\mathrm{d}\boldsymbol{y}\|}\frac{\partial L}{\partial \boldsymbol{x}}, \tag{A2}$$

where the last equality follows by the chain rule. Therefore, the direction of the steepest descent coincides with the direction of the derivative $\partial L/\partial \boldsymbol{x}$, as $\|\mathrm{d}\boldsymbol{y}\|$ is a scalar. □

proof of Proposition 1. We prove that

$$\sum_{j=\ell}^{n} u_j \boldsymbol{e}_{k_j} = \sum_{j=\ell}^{n} \lambda_j \Delta_j - |u_\ell| \sum_{j=1}^{\ell-1} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} \tag{A3}$$

for every $\ell = 1, \ldots, n$, where we abbreviate $u_j = \mathrm{d}\boldsymbol{y}_{k_j}$. The claimed equality (3) then follows from (A3) in the special case $\ell = 1$.

We proceed by induction. In the first step we show (A3) for $\ell = n$. Definition of Δ_n (9) yields

$$\lambda_n \Delta_n - |u_n| \sum_{j=1}^{n-1} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j}$$

Table A7: Ablations of CombOptNet on Random Constraints demonstration. Reported is evaluation accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) for $m = 1, 2, 4, 8$ ground-truth constraints. Statistics are over 20 restarts (2 for each of the 10 dataset seeds).

		Method	1	2	4	8
binary	param.	learnable origins*	**97.8 ± 0.7**	94.2 ± 10.1	**77.4 ± 13.5**	**46.5 ± 12.4**
		direct (origin at corner)	97.4 ± 1.0	**94.9 ± 7.0**	59.0 ± 26.8	26.9 ± 10.3
		direct (origin at center)	98.0 ± 0.5	97.1 ± 0.6	70.5 ± 19.1	44.6 ± 5.9
	basis	Δ basis*	**97.8 ± 0.7**	**94.2 ± 10.1**	**77.4 ± 13.5**	**46.5 ± 12.4**
		canonical	96.3 ± 1.9	70.8 ± 4.1	14.4 ± 3.2	2.7 ± 0.9
	min	hard		83.1 ± 13.2	55.4 ± 18.9	37.7 ± 8.7
		soft ($\tau = 0.5$)*	97.8 ± 0.7	94.2 ± 10.1	**77.4 ± 13.5**	**46.5 ± 12.4**
		soft ($\tau = 1.0$)		**95.7 ± 2.2**	70.2 ± 14.1	36.0 ± 9.7
dense	param.	learnable origins*	**87.3 ± 2.5**	70.2 ± 11.6	29.6 ± 10.4	2.3 ± 1.2
		direct (origin at corner)	86.7 ± 3.0	**74.6 ± 3.6**	**32.6 ± 13.7**	**2.8 ± 0.5**
		direct (origin at center)	83.0 ± 6.1	43.8 ± 13.2	11.6 ± 3.1	1.1 ± 0.5
	basis	Δ basis*	87.3 ± 2.5	70.2 ± 11.6	**29.6** ± 10.4	2.3 ± 1.2
		canonical	**88.6 ± 1.4**	**71.6 ± 1.6**	26.8 ± 4.1	**4.0 ± 0.7**
	min	hard		70.8 ± 15.1	21.4 ± 10.7	2.2 ± 2.1
		soft ($\tau = 0.5$)*	89.1 ± 2.8	70.2 ± 11.6	29.6 ± 10.4	**2.3 ± 1.2**
		soft ($\tau = 1.0$)		**73.0 ± 12.1**	**31.9 ± 11.7**	2.2 ± 1.5

Table A8: Ablations of CombOptNet on Weighted Set Covering. Reported is evaluation accuracy ($\boldsymbol{y} = \boldsymbol{y}^*$ in %) for $m = 4, 6, 8, 10$ ground-truth constraints.

	Method	4	6	8	10
param.	learnable origins*	**100 ± 0.0**	**97.2 ± 6.4**	**79.7 ± 12.1**	**56.7 ± 14.8**
	direct (origin at corner)	99.4 ± 2.9	94.1 ± 16.4	78.5 ± 15.7	47.7 ± 17.9
	direct (fixed origin at 0)	99.9 ± 0.6	87.6 ± 6.4	65.3 ± 11.9	46.7 ± 11.5
basis	Δ basis*	**100 ± 0.0**	**97.2 ± 6.4**	**79.7 ± 12.1**	**56.7 ± 14.8**
	canonical	8.4 ± 13.3	2.0 ± 2.6	0.2 ± 0.3	0.0 ± 0.1
min	hard	88.2 ± 13.4	64.3 ± 14.6	45.1 ± 14.1	32.3 ± 17.4
	soft ($\tau = 0.5$)*	**100 ± 0.0**	**97.2 ± 6.4**	**79.7 ± 12.1**	**56.7 ± 14.8**
	soft ($\tau = 1.0$)	99.9 ± 0.4	95.6 ± 9.6	70.3 ± 15.5	51.2 ± 16.4
	soft ($\tau = 2.0$)	98.8 ± 3.1	90.6 ± 14.3	66.4 ± 12.5	51.2 ± 9.5
	soft ($\tau = 5.0$)	97.5 ± 11.1	90.2 ± 9.1	64.2 ± 11.8	49.7 ± 10.4

$$
\begin{aligned}
&= |u_n| \sum_{j=1}^{n} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} - |u_n| \sum_{j=1}^{n-1} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} \\
&= u_n \boldsymbol{e}_{k_n}.
\end{aligned}
$$

Now, assume that (A3) holds for $\ell + 1 \geq 2$. We show that (A3) holds for ℓ as well. Indeed,

$$
\begin{aligned}
&\sum_{j=\ell}^{n} \lambda_j \Delta_j - |u_\ell| \sum_{j=1}^{\ell-1} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} \\
&\quad = \sum_{j=\ell+1}^{n} \lambda_j \Delta_j - |u_{\ell+1}| \sum_{j=1}^{\ell} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} + \lambda_\ell \Delta_\ell \\
&\qquad + |u_{\ell+1}| \sum_{j=1}^{\ell} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} - |u_\ell| \sum_{j=1}^{\ell-1} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} \\
&\quad = \sum_{j=\ell+1}^{n} u_j \boldsymbol{e}_{k_j} + \big(|u_\ell| - |u_{\ell+1}|\big) \sum_{j=1}^{\ell} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} \\
&\qquad + |u_{\ell+1}| \sum_{j=1}^{\ell} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} - |u_\ell| \sum_{j=1}^{\ell-1} \operatorname{sign}(u_j) \boldsymbol{e}_{k_j} \\
&\quad = \sum_{j=\ell+1}^{n} u_j \boldsymbol{e}_{k_j} + \operatorname{sign}(u_\ell) |u_\ell| \boldsymbol{e}_{k_\ell} = \sum_{j=\ell}^{n} u_j \boldsymbol{e}_{k_j},
\end{aligned}
$$

where we used the definitions of Δ_ℓ and λ_ℓ. □

References

[1] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, pages 9562–9574, 2019.

[2] Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M. Moursi. Differentiating through a cone program. *J. Appl. Numer. Optim*, 1(2):107–115, 2019.

[3] Brandon Amos. *Differentiable optimization-based modeling for machine learning*. PhD thesis, PhD thesis. Carnegie Mellon University, 2019.

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

[5] Brandon Amos and Denis Yarats. The differentiable cross-entropy method. In *International Conference on Machine Learning*, pages 291–302, 2020.

[6] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353, 2018.

[7] Mislav Balunovic, Pavol Bielik, and Martin Vechev. Learning to solve SMT formulas. In *Advances in Neural Information Processing Systems*, pages 10317–10328, 2018.

[8] Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261*, 2018.

[9] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv:1611.09940*, 2016.

[10] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable perturbed optimizers. In *Advances in Neural Information Processing Systems*, pages 9508–9519, 2020.

[11] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*, 2019.

[12] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Conference on Empirical Methods in Natural Language Processing*, pages 670–680, Copenhagen, Denmark, 2017. Association for Computational Linguistics.

[13] IBM ILOG Cplex. V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.

[14] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv:1801.08757*, 2018.

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

[16] Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems*, pages 1013–1023, 2017.

[17] Justin Domke. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326, 2012.

[18] Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 5484–5494, 2017.

[19] Adam N. Elmachtoub and Paul Grigas. Smart "predict, then optimize". *arXiv:1710.08005*, 2020.

[20] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. MIPaaL: Mixed integer program as a layer. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 1504–1511, 2020.

[21] Matthias Fey, Jan E Lenssen, Christopher Morris, Jonathan Masci, and Nils M Kriege. Deep graph matching consensus. In *International Conference on Learning Representations*, 2020.

[22] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv:1607.05447*, 2016.

[23] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv:1410.5401*, 2014.

[24] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, Adrià Puigdomènech Badia, Karl Moritz Hermann, Yori Zwols, Georg Ostrovski, Adam Cain, Helen King, Christopher Summerfield, Phil Blunsom, Koray Kavukcuoglu, and Demis Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, October 2016.

[25] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2019. URL http://www.gurobi.com.

[26] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2014.

[28] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

[29] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[30] Chun Kai Ling, Fei Fang, and J Zico Kolter. What game are we playing? End-to-end learning in normal and extensive form games. In *International Joint Conference on Artificial Intelligence*, 2018.

[31] Juhong Min, Jongmin Lee, Jean Ponce, and Minsu Cho. SPair-71k: A Large-scale Benchmark for Semantic Correspondence. *arXiv:1908.10543*, 2019.

[32] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *arXiv:2012.13349*, 2020.

[33] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems*, pages 9839–9849, 2018.

[34] Vlad Niculae, Andre Martins, Mathieu Blondel, and Claire Cardie. Sparsemap: Differentiable sparse structured inference. In *International Conference on Machine Learning*, pages 3799–3808, 2018.

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

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

[37] Michal Rolínek, Paul Swoboda, Dominik Zietlow, Anselm Paulus, Vít Musil, and Georg Martius. Deep graph matching via blackbox differentiation of combinatorial solvers. In *European Conference on Computer Vision*, pages 407–424, 2020.

[38] Yingcong Tan, Daria Terekhov, and Andrew Delong. Learning linear programs from optimal decisions. In *Advances in Neural Information Processing Systems*, pages 19738–19749, 2020.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[40] Petar Veličković, Rex Ying, Matilde Padovano, Raia Hadsell, and Charles Blundell. Neural execution of graph algorithms. In *International Conference on Learning Representations*, 2020.

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

[42] Marin Vlastelica, Michal Rolinek, and Georg Martius. Discrete planning with end-to-end trained neuro-algorithmic policies. *ICML 2020, Graph Representation Learning Workshop*, 2020.

[43] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 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*, pages 5310–5318, 2019.

[44] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *International Conference on Machine Learning*, pages 6545–6554, 2019.

[45] Bryan Wilder, Eric Ewing, Bistra Dilkina, and Milind Tambe. End to end learning and optimization on graphs. In *Advances in Neural Information Processing Systems*, pages 4672–4683, 2019.

[46] Andrei Zanfir and Cristian Sminchisescu. Deep learning of graph matching. In *Conference on Computer Vision and Pattern Recognition*, pages 2684–2693, 2018.

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