Supplementary Material for 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]

A. Demonstrations

The code for all demonstrations is available at

github.com/martius-lab/CombOptNet.

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

$$\operatorname{softmin}(\boldsymbol{x}) = -\tau \cdot \log\left(\sum_k \exp\left(-\frac{x_k}{\tau}\right)\right), \qquad \text{(S1)}$$

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. S1. We use Adam (Kingma & Ba, 2014) as the optimizer for all demonstrations. Gurobi parameters for all experiments are kept to default.

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.

Table S1: 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}

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.

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 Con-

Table S2: 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

straints demonstration. We use the architecture described by Rolínek et al. (2020), 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. (2020), 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. S2. 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.

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. S3 and S4) and the choice of the loss function Tab. S5.

Table S3: 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**

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.

Table S4: 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**

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. S5, 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. S6. Similar to the *binary* Random Constraints ablation with

*Used in the main demonstrations.

Table S5: 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 S6: 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*	2	4	8
64.7 ± 2.8	63.5 ± 3.7	**65.7** ± **3.1**	62.6 ± 4.4

$m = 1$, 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. S1 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. S2 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 overestimate the price and underestimate the item weight, or vice versa, due to similar effects of these errors on the predicted solution.

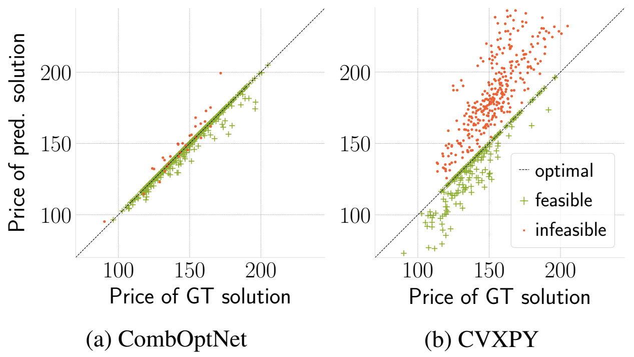

Figure S1: Prices analysis for the KNAPSACK demonstration. For each test set instance, we plot the total price of the predicted solution over the total price of the ground-truth solution. Predicted solutions which total weight exceeds the knapsack capacity are colored in red (cross).

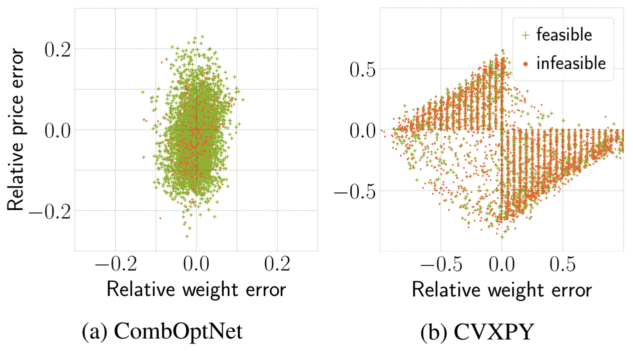

Figure S2: Qualitative analysis of the errors on weights and prices in the KNAPSACK demonstration. We plot the relative error between predicted and ground-truth item prices over the relative error between predicted and ground-truth item weights. Colors denote whether the predicted solution is feasible in terms of ground-truth weights.

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. S7 and S8 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 $\boldsymbol{y}$.

Table S7: 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 S8: 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. Statistics are over 20 restarts (2 for each of the 10 dataset seeds).

	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

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 S1. *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

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

hence

$$\varphi'(0) = -\frac{1}{\|\mathrm{d}\boldsymbol{y}\|}\frac{\partial L}{\partial \boldsymbol{y}} \cdot \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} = -\frac{1}{\|\mathrm{d}\boldsymbol{y}\|}\frac{\partial L}{\partial \boldsymbol{x}}, \tag{S2}$$

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

$$\sum_{j=\ell}^{n} u_j \boldsymbol{e}_{k_j} = \sum_{j=\ell}^{n} \lambda_j \Delta_j - |u_\ell| \sum_{j=1}^{\ell-1} \operatorname{sign}(u_j)\boldsymbol{e}_{k_j} \tag{S3}$$

for every $\ell = 1, \ldots, n$, where we abbreviate $u_j = \mathrm{d}\boldsymbol{y}_{k_j}$. The claimed equality (3) then follows from (S3) in the special case $\ell = 1$.

We proceed by induction. In the first step we show (S3) for $\ell = n$. Definition of Δ_n (9) yields

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

Now, assume that (S3) holds for $\ell + 1 \geq 2$. We show that (S3) 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. □