

# Learning a Large Neighborhood Search Algorithm for Mixed Integer Programs

Nicolas Sonnerat<sup>\*1</sup> Pengming Wang<sup>\*1</sup> Ira Ktena<sup>1</sup> Sergey Bartunov<sup>23</sup> Vinod Nair<sup>24</sup>

<sup>\*</sup>Equal contribution

## Abstract

Large Neighborhood Search (LNS) is a combinatorial optimization heuristic that starts with an assignment of values for the variables to be optimized, and iteratively improves it by searching a large neighborhood around the current assignment. In this paper we consider a learning-based LNS approach for mixed integer programs (MIPs). We train a *Neural Diving* model to generate an initial assignment. Formulating the subsequent search steps as a Markov Decision Process, we train a *Neural Neighborhood Selection* policy to select a search neighborhood at each step, which is searched using a MIP solver to find the next assignment. The policy network is trained using imitation learning. We propose a target policy for imitation that is designed to select the neighborhood containing the optimal next assignment amongst all possible choices for the neighborhood of a specified size. Our approach matches or outperforms all the baselines on five diverse real-world MIP datasets with large-scale instances, including two production applications at a large technology company. It achieves  $2\times$  to  $37.8\times$  better average primal gap than the best baseline on three datasets at large running times.

## 1. Introduction

Large Neighborhood Search (LNS) (Shaw, 1998; Pisinger & Ropke, 2010) is a powerful heuristic for hard combinatorial optimization problems such as Mixed Integer Programs (MIPs) (Danna et al., 2005; Rothberg, 2007; Berthold, 2007; Ghosh, 2007), Traveling Salesman Problem (TSP) (Smith & Imeson, 2017), Vehicle Routing Problem (VRP) (Shaw, 1998; Hojabri et al., 2018), and Constraint Programming

(CP) (Perron et al., 2004; Berthold et al., 2012). Given a problem instance and an initial feasible assignment (i.e., an assignment satisfying all constraints of the problem) of values to the variables of the problem, LNS searches for a better assignment within a neighborhood of the current one at each iteration. Iterations continue until the search budget (e.g., time) is exhausted. The neighborhood is “large” in the sense that it contains too many assignments to tractably search with naive enumeration. Large neighborhoods make the search less susceptible to getting stuck in poor local optima.

Two key choices that determine the effectiveness of LNS are 1) the initial assignment, and 2) the search neighborhood at each iteration. A good initial assignment makes good optima more likely to be reached. A good neighborhood selection policy allows faster convergence to good optima. Domain experts design sophisticated heuristics by exploiting problem structure to find an initial feasible assignment, e.g. for MIPs, (Fischetti et al., 2005; Berthold, 2007) and to define the neighborhood, e.g. Pisinger & Ropke (2010); Shaw (1998); Danna et al. (2005).

In this paper we use learned models to make both of these choices. We focus specifically on Mixed Integer Programs to demonstrate the approach, but it can be adapted to other combinatorial optimization problems also. Figure 1 gives a summary. To compute an initial feasible assignment of values for the variables, we use *Neural Diving* (section 2.2) proposed in Nair et al. (2020), which has been shown to produce high quality assignments quickly. The assignment is computed using a generative model that conditions on the input MIP and defines a distribution over assignments such that ones with better objective values are more probable. To define the search neighborhood at each LNS iteration, we use a *Neural Neighborhood Selection* policy (section 3) that, conditioned on the current assignment, selects a subset of the integer variables in the input MIP to unassign their values. The policy’s decisions can then be used to derive from the input MIP a smaller “sub-MIP” to optimize the unassigned variables. By setting the number of unassigned integer variables sufficiently small, the sub-MIP can be solved quickly using an off-the-shelf solver to compute the assignment for the next LNS step. The policy is trained by

<sup>1</sup>DeepMind, London, UK <sup>2</sup>Work done at DeepMind

<sup>3</sup>Charm Therapeutics, London, UK <sup>4</sup>Google Research, Bangalore, India. Correspondence to: Nicolas Sonnerat <sonnerat@deepmind.com>, Pengming Wang <pengming@deepmind.com>.

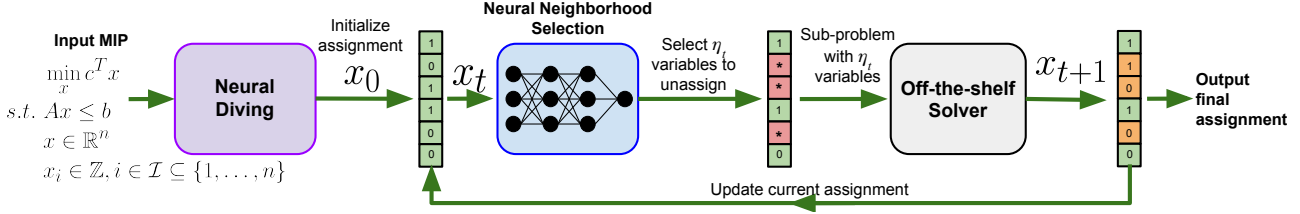


Figure 1: Overview of our approach at test time. The input is a mixed integer program (MIP). *Neural Diving* (Nair et al., 2020) combines a generative model with an off-the-shelf MIP solver to output an initial assignment  $x_0$  for the variables  $x$  to be optimized. At the  $t^{\text{th}}$  iteration of LNS the *Neural Neighborhood Selection* policy selects  $\eta_t$  variables to be unassigned (indicated by red, with  $\eta_t = 3$ ) from the current assignment  $x_t$ . A sub-MIP defined on those  $\eta_t$  variables is solved with a MIP solver to assign them new values (orange) to define the next assignment  $x_{t+1}$ . Iterations continue until the search budget is exhausted.

imitating an expert neighborhood selection policy (section 3.2). At each LNS iteration, the expert is designed to select the best assignment in a Hamming ball centered around the current assignment by solving a MIP optimally. The changes in the values of the integer variables between the current and new assignments give the expert’s unassignment decisions to use as targets for imitation learning. The expert itself is too computationally expensive for a practical LNS algorithm, but is still tractable for generating imitation training data offline. The neural network trained to approximate it can be orders of magnitude faster, making it practical to use at test time.

Previous works have combined learning with LNS. Hottung & Tierney (2019) use an approach complementary to ours for Capacitated VRPs by learning to search the neighborhood, instead of to select it. In our setting, since off-the-shelf MIP solvers can already search neighborhoods, we expect learning to be more useful for neighborhood selection. Song et al. (2020) learn a neighborhood selection policy using imitation learning and reinforcement learning (RL). Their method restricts the neighborhood selection policy to choose fixed, predefined variable subsets, instead of arbitrary subsets in our work. It uses a random neighborhood selection policy to generate training data for imitation learning. Ad-danki et al. (2020) use RL to learn a policy that unassigns one variable at a time, interleaved with solving a sub-MIP every  $\eta$  steps to compute a new assignment. For large MIPs, one neural network policy evaluation per variable can be prohibitively slow. Our approach is scalable – both selecting an initial assignment and a search neighbourhood at each LNS step are posed as modelling the joint distribution of a large number of simultaneous decisions. This allows us to exploit high-dimensional generative models for scalable training and inference. To demonstrate scalability, we evaluate on real world datasets with large-scale MIPs, unlike earlier works.

## Contributions:

1. We present a scalable learning-based LNS algorithm that combines learned models for computing the initial assignment and for selecting the search neighborhood at each LNS step.
2. We propose an imitation learning approach to train the neighborhood selection policy using as the imitation target an expert policy that, under certain assumptions, is guaranteed to select the neighborhood containing the optimal next assignment at a given LNS step.
3. We show results on five diverse large-scale real-world datasets, including two from the production systems of a large technology company. It matches or outperforms all baselines on all of them, with a  $2 - 37.8\times$  improvement over the best baseline with respect to the main performance metric, average primal gap, on three of them.

We have also published source code for the key components of our approach, covering data generation, training, and inference, at [https://github.com/deepmind/neural\\_lns](https://github.com/deepmind/neural_lns).

## 2. Background

### 2.1. Mixed Integer Programming

A Mixed Integer Program is defined as  $\min_x \{f(x) = c^T x \mid Ax \leq b, x_i \in \mathbb{Z}, i \in \mathcal{I}\}$ , where  $x \in \mathbb{R}^n$  are the variables to be optimized,  $A \in \mathbb{R}^{m \times n}$  and  $b \in \mathbb{R}^m$  specify  $m$  linear constraints,  $c \in \mathbb{R}^n$  specifies the linear objective function, and  $\mathcal{I} \subseteq \{1, \dots, n\}$  is the index set of integer variables. If  $\mathcal{I} = \emptyset$ , the resulting continuous optimization problem is called a *linear program*, which is solvable in polynomial time. A *feasible* assignment is a point  $x \in \mathbb{R}^n$  that satisfies all the constraints. A *complete solver* tries to produce a feasible assignment and a lower bound on the optimal objective value, and given sufficient compute resources will find the optimal assignment or prove that there exists no

feasible one. A *primal heuristic* (see, e.g., Berthold 2006) only attempts to find a feasible assignment. This work focuses on primal heuristics as production applications often only require finding a good feasible assignment quickly.

## 2.2. Neural Diving

Neural Diving (Nair et al., 2020) is a learning-based primal heuristic. It learns a probability distribution for assignments of integer variables of the input MIP  $M$  such that assignments with better objective values have higher probability. Assuming minimization, an *energy function* is defined over the integer variables of the problem  $x_{\mathcal{I}} = \{x_i | i \in \mathcal{I}\}$  as

$$E(x_{\mathcal{I}}; M) = \begin{cases} \hat{f}(x_{\mathcal{I}}) & \text{if } x_{\mathcal{I}} \text{ is feasible,} \\ \infty & \text{otherwise,} \end{cases} \quad (1)$$

where  $\hat{f}(x_{\mathcal{I}})$  is the objective value obtained by substituting  $x_{\mathcal{I}}$  for the integer variables in  $M$  and assigning the continuous variables to the solution of the resulting linear program. The target distribution is defined as  $p(x_{\mathcal{I}} | M) = \exp(-E(x_{\mathcal{I}}; M)) / Z(M)$  where  $Z(M) = \sum_{x_{\mathcal{I}}} \exp(-E(x_{\mathcal{I}}; M))$  is the *partition function*. The model is trained to minimize a weighted negative log likelihood loss on the training set  $\{(M^{(j)}, x_{\mathcal{I}}^{(j)})\}_{j=1}^N$  of  $N$  MIPs and corresponding assignments collected with an off-the-shelf solver. Assignments with better objective values are given larger weights so that the model better approximates the target distribution.

Nair et al. (2020) represent the MIP as a bipartite graph (see, e.g., Gasse et al. (2019)) and use a Graph Convolutional Network (Battaglia et al., 2018; Gori et al., 2005; Scarselli et al., 2008; Hamilton et al., 2017; Kipf & Welling, 2016) to parameterize a conditionally independent model of the joint distribution over integer variable assignments.

Given a MIP at test time, the model’s predicted distribution is used to generate multiple *partial* assignments for the integer variables. For each such partial assignment, substituting the values of the assigned variables in  $M$  defines a sub-MIP with only the unassigned variables, which is then solved by an off-the-shelf MIP solver to complete the assignment. Neural Diving outputs the best assignment among all such complete assignments. Since each partial assignment can be completed in parallel, Neural Diving is well-suited to exploit parallel computation for faster runtimes. Results in Nair et al. (2020) show that Neural Diving is effective in producing high quality assignments quickly on several datasets. See that paper for further details.

## 3. Neural Neighborhood Selection

### 3.1. MDP Formulation

We consider a contextual Markov Decision Process (Abbasi-Yadkori & Neu, 2014; Hallak et al., 2015)  $\mathcal{M}_z$  parameterized with respect to a *context*  $z$ , where the state space, action space, reward function, and the environment all depend on  $z$ . Here we define  $z$  to be the parameters of the input MIP, i.e.,  $z = M = \{A, b, c\}$ . The *state*  $s_t$  at the  $t^{\text{th}}$  step of an episode is the current assignment  $x_t$  of values for all the integer variables in  $M$ . The *action*  $a_t \in \{0, 1\}^{|\mathcal{I}|}$  at step  $t$  is the choice of the set of integer variables to be unassigned, specified by one indicator variable per integer variable in  $M$  where 1 means unassigned. All continuous variables are labelled as unassigned at every LNS step. For real-world applications the number of integer variables  $|\mathcal{I}|$  is typically large ( $10^3 - 10^6$ ), so the actions are high-dimensional binary vectors. The *policy*  $\pi_{\theta}(a_t | s_t, M)$  defines the distribution over actions, parameterized by  $\theta$ . We use a conditional generative model to represent this high-dimensional distribution over binary vectors (section 3.3).

Given  $s_t$  and  $a_t$ , the *environment* derives a sub-MIP  $M'_t = \{A'_t, b'_t, c'_t\}$  from  $M$  containing only the unassigned integer variables and all continuous variables, and optimizes it.  $M'_t$  is computed by substituting the values in  $x_t$  of the assigned variables into  $M$  to derive constraints and objective function with respect to the rest of the variables.  $M'_t$  is guaranteed to have a non-empty feasible set – the values in  $x_t$  of the unassigned variables itself is a feasible assignment for  $M'_t$ . The set of feasible assignments for  $M'_t$  is the search neighborhood for step  $t$ . The environment calls an off-the-shelf MIP solver, in our case the state-of-the-art non-commercial MIP solver SCIP 7.0.1 (Gamrath et al., 2020), to search this neighborhood. The output of the solve is then combined with the values of the already assigned variables to construct a new feasible assignment  $x_{t+1}$  for  $M$ . If the solver outputs an optimal assignment for the sub-MIP, then  $c^T x_{t+1} \leq c^T x_t$ . The per-step *reward* can be defined using a metric that measures progress towards an optimal assignment (Addanki et al., 2020), such as the negative of the *primal gap* (Berthold, 2006) (see equation 9) which is normalized to be numerically comparable across MIPs (unlike, e.g., the raw objective values).

An episode begins with an input MIP  $M$  and an initial feasible assignment  $x_0$ . It proceeds by running the above MDP to perform large neighborhood search until the search budget (e.g., time) is exhausted.

The size of the search neighborhood at the  $t^{\text{th}}$  step typically increases exponentially with the number of unassigned integer variables  $\eta_t$ . Larger neighborhoods can make LNS less susceptible to getting stuck at local optima, but it can also be computationally more expensive to search. We treat  $\eta_t$

as a hyperparameter to control this tradeoff.

### 3.2. Expert Neighborhood Selection Policy

We propose an expert policy that aims to compute the unassignment decisions  $a_t^*$  for finding the optimal next assignment  $x_{t+1}^*$  across all possible search neighborhoods around  $x_t$  given by unassigning any  $\eta_t$  integer variables. It uses *local branching* (Fischetti & Lodi, 2003) to compute the optimal next assignment  $x_{t+1}^*$  within a given Hamming ball of radius  $\eta_t$  centered around the current assignment  $x_t$ . The minimal set of unassignment decisions  $a_t^*$  is then derived by comparing the values of the integer variables between  $x_t$  and  $x_{t+1}^*$  and labelling only those with different values as unassigned. If the policy  $\pi_\theta(a_t|x_t, M)$  takes the action  $a_t^*$  and the corresponding sub-MIP  $M_t' = \{A_t', b_t', c_t'\}$  is solved optimally by the environment, then the next assignment will be  $x_{t+1}^*$ .

Local branching adds a constraint to the input MIP  $M$  such that only those assignments for  $x_{t+1}$  within a Hamming ball around  $x_t$  are feasible. If all integer variables in  $M$  are binary, the constraint is:

$$\sum_{i \in \mathcal{I}: x_{t,i}=0} x_{t+1,i} + \sum_{i \in \mathcal{I}: x_{t,i}=1} (1 - x_{t+1,i}) \leq \eta_t, \quad (2)$$

where  $x_{t,i}$  denotes the  $i^{\text{th}}$  dimension of  $x_t$  and  $\eta_t$  is the desired Hamming radius. The case of general integers can also be handled (see, e.g., slide 23 of Lodi (2003)). The optimal solution of the MIP with the extra constraint will differ from  $x_t$  only on at most  $\eta_t$  dimensions, so it is the best assignment across all search neighborhoods for the desired number of unassigned integer variables.

The expert itself is too slow to be directly useful for solving MIPs, especially when the number of variables and constraints are large. Instead it is used to generate episode trajectories from a training set of MIPs for imitation learning. As a one-time offline computation, the compute budget for data generation can be much higher than that of solving a MIP, which enables the use of a slow expert.

### 3.3. Policy Network

**MIP representation:** Following Nair et al. (2020) and earlier works (e.g., Gasse et al. (2019)), we use a bipartite graph representation of a MIP for both Neural Diving and Neural Neighborhood Selection. Variables form one set of nodes in the graph and constraints form the other set. A variable appearing in a constraint is indicated by an edge between the two corresponding nodes. Coefficients in  $A$ ,  $b$ , and  $c$  are encoded as features of the corresponding edges, constraint nodes, and variable nodes, respectively. Additional features can be included (e.g., the linear relaxation solution as variable node features) – we use the feature set proposed in

Gasse et al. (2019). For the Neural Neighborhood Selection policy network, we additionally use a fixed-size window of past variable assignments as variable node features. The window size is set to 3 in our experiments.

**Network architecture:** We use a **Graph Convolutional Network** to represent the policy. Let the input to the GCN be a graph  $G = (\mathcal{V}, \mathcal{E}, \mathcal{A})$  defined by the set of nodes  $\mathcal{V}$ , the set of edges  $\mathcal{E}$ , and the graph adjacency matrix  $\mathcal{A}$ . In the case of MIP bipartite graphs,  $\mathcal{V}$  is the union of  $n$  variable nodes and  $m$  constraint nodes, of size  $K := |\mathcal{V}| = n + m$ .  $\mathcal{A}$  is an  $K \times K$  binary matrix with  $\mathcal{A}_{ij} = 1$  if nodes indexed by  $i$  and  $j$  are connected by an edge, 0 otherwise, and  $\mathcal{A}_{ii} = 1$  for all  $i$ . Let  $U \in \mathbb{R}^{K \times D}$  be the matrix containing  $D$ -dimensional feature vectors of all nodes as rows.

An  $L$ -layer GCN is defined as follows:

$$Z^{(0)} = U \quad (3)$$

$$Z^{(l+1)} = \text{Ag}_{\phi(l)}(Z^{(l)}), \quad l = 0, \dots, L-1, \quad (4)$$

where  $Z^{(l)} \in \mathbb{R}^{K \times H^{(l)}}$  is the matrix of  $H^{(l)}$ -dimensional *node embeddings* for the  $K$  nodes as rows, and  $\text{g}_{\phi(l)}()$  is a Multi-Layer Perceptron (MLP) (Goodfellow et al., 2016) with learnable parameters  $\phi(l) \in \theta$  for the  $l^{\text{th}}$  layer applied row-wise to  $Z^{(l)}$ . The  $L^{\text{th}}$  layer's node embeddings can be used as input to another MLP that computes the outputs for the final prediction task.

The policy is a conditionally independent model

$$\pi_\theta(a_t|x_t, M) = \prod_{i \in \mathcal{I}} p_\theta(a_{t,i}|x_t, M), \quad (5)$$

which predicts the probability of  $a_{t,i}$ , the  $i^{\text{th}}$  dimension of  $a_t$ , independently of its other dimensions conditioned on  $M$  and  $x_t$  using the Bernoulli distribution  $p_\theta(a_{t,i}|x_t, M)$ . Its success probability  $\mu_{t,i}$  is computed as

$$\lambda_{t,i} = \text{MLP}(v_{t,i}; \theta), \quad (6)$$

$$\mu_{t,i} = p_\theta(a_{t,i} = 1|x_t, M) = \frac{1}{1 + \exp(-\lambda_{t,i})}, \quad (7)$$

where  $v_{t,i} \in \mathbb{R}^{H^{(L)}}$  is the  $L^{\text{th}}$  layer embedding computed by a GCN for the node corresponding to  $x_{t,i}$ , and  $\lambda_{t,i} \in \mathbb{R}$ .

### 3.4. Training

Given a training set  $\mathcal{D}_{\text{train}} = \{(M^{(j)}, x_{1:T_j}^{(j)}, a_{1:T_j}^{(j)})\}_{j=1}^N$  of  $N$  MIPs and corresponding expert trajectories, the model parameters  $\theta$  are learned by minimizing the negative log likelihood of the expert unassignment decisions:

$$L(\theta) = - \sum_{j=1}^N \sum_{t=1}^{T_j} \log \pi_\theta(a_t^{(j)}|x_t^{(j)}, M^{(j)}), \quad (8)$$



where  $M^{(j)}$  is the  $j^{th}$  training MIP instance,  $\{x_t^{(j)}\}_{t=1}^{T_j}$  are the feasible assignments for the variables in  $M^{(j)}$ , and  $\{a_t^{(j)}\}_{t=1}^{T_j}$  are the corresponding unassignment decisions by the expert in a trajectory of  $T_j$  steps.

### 3.5. Using the Trained Model

Given an input MIP, first Neural Diving is applied to it to compute the initial feasible assignment. An episode then proceeds as described in section 3.1, with actions sampled from the trained model.

**Sampling actions:** Directly sampling unassignment decisions from the Bernoulli distributions output by the model often results in sets of unassigned variables that are much smaller than a desired neighborhood size. This is due to highly unbalanced data produced by the expert (typically most of the variables remain assigned), which causes the model to predict a low probability of unassigning each variable. Instead we construct the unassigned variable set  $U$  sequentially, starting with an empty set, and at each step adding to it an integer variable  $x_{t,i}$  with probability proportional to  $(p_\theta(a_{t,i} = 1|x_t, M) + \epsilon)^{\frac{1}{\tau}} \cdot \mathbb{I}[x_{t,i} \notin U]$  with  $\epsilon > 0$  to assign nonzero selection probability for all variables. Here,  $\tau$  is a temperature parameter. This ensures that  $U$  contains exactly the desired number of unassigned variables.

**Adaptive neighborhood size:** The number of variables unassigned at each step is chosen in an adaptive manner (Lee & Stuckey, 2021). The initial number is set as a fraction of the number of integer variables in the input MIP. At a given LNS step, if the sub-MIP solve outputs a provably optimal assignment, the fraction for the next step is increased by a factor  $\alpha > 1$ . If the sub-MIP solve times out without finding a provably optimal assignment, the fraction for the next step is divided by  $\alpha$ . This allows LNS to adapt the neighborhood size according to difficulty of the sub-MIP solves.

## 4. Evaluation Setup

### 4.1. Datasets

We evaluate our approach on five datasets: Neural Network Verification, Electric Grid Optimization, Production Packing, Production Planning, and MIPLIB. The first four are homogeneous datasets in which the instances are from a single application, while MIPLIB (Gleixner et al., 2019) is a heterogeneous public benchmark with instances from many, often unrelated, applications. They contain large-scale MIPs with thousands to millions of variables and constraints (figure 4 in Technical Appendix). In particular, Production Packing and Planning datasets are obtained from a large technology company’s production systems. See Technical Appendix section A.6 for details. All five datasets were split

into training, validation, and test sets, each consisting of 70%, 15%, and 15% of total instances, respectively. We train a separate model on each dataset, and evaluate it on the corresponding test set’s MIPs. We also report in section A.7 preliminary results for datasets from the NeurIPS’21 Machine Learning for Combinatorial Optimization competition.

### 4.2. Metrics

We follow the evaluation protocol of Nair et al. (2020) and report two metrics, the *primal gap* and the fraction of test instances with the primal gap below a threshold, both as a function of time. The *primal gap*  $\gamma(t)$  at time  $t$  is the normalized difference between the objective value achieved by an algorithm under evaluation at  $t$  and a precomputed best known objective value  $f(x^*)$  (Berthold, 2006):

$$\gamma(t) = \begin{cases} 1, & \text{if } f(x_t) \cdot f(x^*) < 0 \\ & \text{or no solution at time } t, \\ \frac{|f(x_t) - f(x^*)|}{\max\{|f(x_t)|, |f(x^*)|\}}, & \text{otherwise.} \end{cases} \quad (9)$$

We average primal gaps over all test instances at a given time and refer to this as *average primal gap*, and plot it as a function of running time.

Applications typically specify a threshold on the gap between an assignment’s objective value and a lower bound, below which the assignment is deemed close enough to optimal to stop the solve. The dataset-specific gap thresholds are given in table 3. We apply these thresholds to the primal gap to decide when a MIP is considered solved. We plot the fraction of solved test instances as a function of running time, which we refer to as a *survival curve*.

As in Nair et al. (2020), we use *calibrated time* to measure running time. It reduces the variance of time measurements on a shared compute cluster needed for a large-scale evaluation. See the Technical Appendix for details.

### 4.3. Baselines

We compare our approach to three baselines:

1. Random Neighborhood Selection (RNS), where the integer variables to unassign are selected uniformly randomly (referred to as the *Random Destroy method* in Pisinger & Ropke (2010)), with an adaptive neighbourhood size as explained in section 3.5. We use Neural Diving to initialize the feasible assignment.
2. Neural Diving, as described in Nair et al. (2020) and section 2.2, which achieved the previous best primal gap results on the datasets in section 4.1.
3. SCIP 7.0.1 with its hyperparameters (“metaparameters” for the presolve, cuts, and heuristics components) tuned

for each dataset separately using grid search to achieve the best validation set average primal gap curves. SCIP is a complete solver that uses state-of-the-art primal heuristics. By tuning SCIP’s hyperparameters to minimize average primal gap quickly, we aim to make SCIP behave more like a primal heuristic.

**Use of parallel computation:** Neural Diving can naturally exploit parallel computation for faster performance. This advantage carries over to Neural Neighborhood Selection as well when combined with Neural Diving by using parallel LNS runs initialized with multiple feasible assignments. We evaluate Neural Diving and combinations of Neural Diving with Random or Neural Neighborhood Selection in the parallel setting. All of these primal heuristics are given the same amount of parallel compute resources in experiments. SCIP is evaluated only in the single core setting, as the main focus of this work is to evaluate the benefit of easily parallelizable primal heuristics.

## 5. Results

Figure 2b shows that on all five datasets, combining Neural Diving and Neural Neighbourhood Selection (ND + NNS) significantly outperforms SCIP on the test instances, in some cases substantially. On Production Packing, the final average primal gap is almost two orders of magnitude smaller, while on Neural Network Verification and Production Planning it is more than  $10\times$  smaller. On all datasets except MIPLIB, the advantage of ND + NNS over SCIP is substantial even at smaller running times.

ND + NNS outperforms Neural Diving alone on all datasets, with  $10 - 100\times$  smaller gap on Production Packing, Electric Grid Optimization, and Neural Network Verification. Neural Diving quickly reduces the average primal gap early on, but plateaus at larger running times. ND + NNS overcomes this limitation, reducing the average primal gap significantly with more running time. On MIPLIB, Neural Diving shows a better gap curve initially, before being overtaken by ND + NNS after about  $10^3$  seconds.

Combining Neural Diving with Random Neighbourhood Selection (ND + RNS) is a strong baseline on all datasets except Electric Grid Optimization. It is only slightly worse than ND + NNS on Neural Network Verification and MIPLIB. But on Production Planning, Production Packing, and Electric Grid Optimization, ND + NNS achieves a final average primal gap that is smaller by roughly  $2.0\times$ ,  $13.9\times$ , and  $37.8\times$ , respectively. Note that ND + RNS is not better than Neural Diving alone on all datasets, but ND + NNS is.

### 5.1. Survival Curves

Figure 2a shows the performance of ND + NNS using survival curves. Compared to SCIP, our method’s performance is considerably stronger on Production Packing, Electric Grid Optimization, and MIPLIB. On the first two, NNS solves almost all test instances to within the specified target gap, while SCIP only solves about 10% on Production Packing, and about 80% on Electric Grid Optimization. For Neural Network Verification, while SCIP eventually also solves all the instances, the survival curve for ND + NNS achieves the same fraction of solved instances faster. Even on MIPLIB, ND + NNS achieves a final solve fraction of roughly 80%, compared to SCIP’s 60%. Similarly, comparing ND + NNS to Neural Diving shows the former achieving higher final solve fractions on all datasets except Production Planning, where the two methods perform roughly the same.

ND + NNS outperforms ND + RNS on Electric Grid Optimization, Neural Network Verification, and MIPLIB, by either achieving a better final solve fraction or the same solve fraction in less time. Note that survival curves need not fully reflect the improvements in average primal gaps achieved by ND + NNS shown in figure 2b because improving the gap beyond the threshold does not improve the survival curve.

### 5.2. Ablation Study

We evaluate how the two main components of our approach contribute to its performance. We consider four variants in which the initial assignment is given by either SCIP or Neural Diving, and neighborhood search is done using either Random Neighborhood Selection or Neural Neighborhood Selection. Figure 2c shows that, on all datasets except Neural Network Verification and MIPLIB, the average primal gap becomes worse without Neural Diving. This is true regardless of whether we use NNS or RNS. For MIPLIB, ND + NNS finishes with the best average primal gap, but is worse at intermediate running times. For Neural Network Verification, SCIP turns out to be better than Neural Diving for providing the initial assignment. NNS is crucial, achieving roughly a  $100\times$  lower gap than SCIP + RNS. While the relative contribution of Neural Diving and Neural Neighborhood Selection to our approach’s performance depends on the dataset, it is clear across all datasets that learning is necessary to achieve the best performance.

### 5.3. Approximating the Expert

Figure 3 shows the average primal gap as a function of the number of LNS steps for the expert policy, Neural Neighborhood Selection, and Random Neighborhood Selection, all initialized with the same assignment computed using Neural Diving. This allows comparing the quality of the policies in-

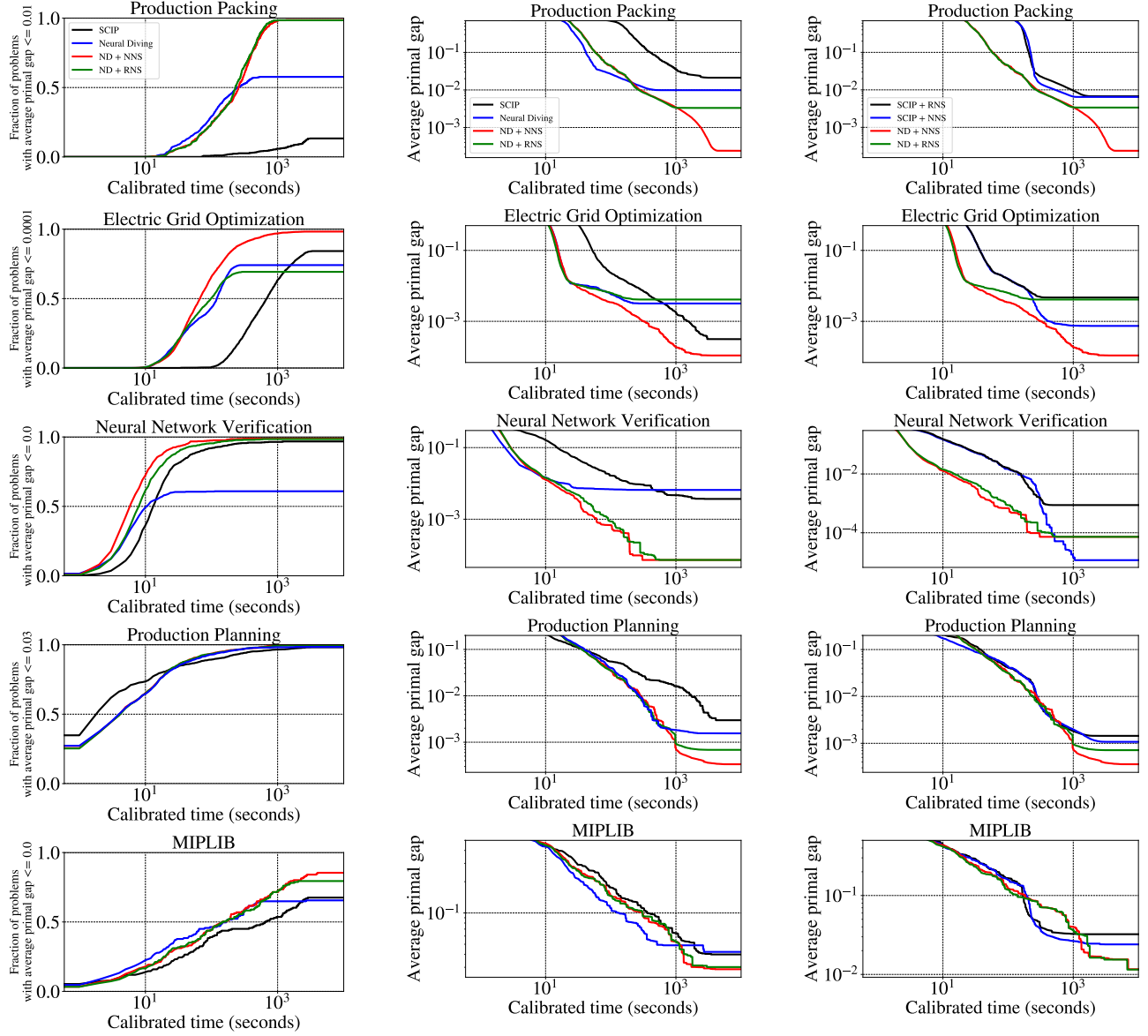


Figure 2: Experimental results.

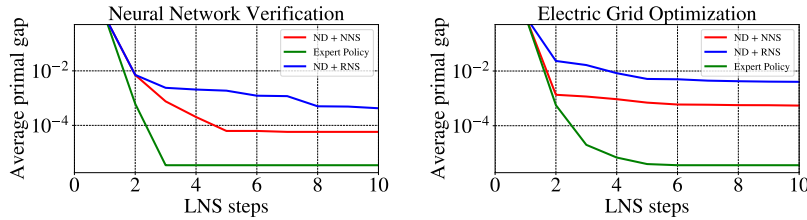


Figure 3: Comparison of expert policy used as the target for imitation learning to random (ND + RNS) and learned (ND + NNS) policies for selecting a search neighborhood at each step of large neighborhood search, with the initial assignment computed using Neural Diving for all three cases.

independently of their speed. For brevity we include only two representative datasets. The average primal gap is computed on a subset of the validation set. On both datasets the expert reduces the gap in the fewest steps, confirming its effectiveness as an imitation target. NNS learns to approximate it well enough to outperform RNS, which shows that imitation learning is a viable approach for the neighborhood selection task. Figure 3 also shows that there is more room for the learned policy to approximate the expert better. Improving the learned policy’s approximation while maintaining its speed can lead to even better average primal gap.

## 6. Discussion

Neural Diving and ND + NNS have complementary strengths. As shown in section 5, Neural Diving is often quicker than SCIP at achieving low average primal gaps in short running times, but its average primal gap tends to plateau at higher running times. One reason is that a single run of Neural Diving applies the learned model on the input MIP only once at the beginning, which has a fixed time cost. Any additional running time available is not used to apply the learned model again; instead it is allocated to the sub-MIP solve with SCIP. ND + NNS on the other hand can use the available time to repeatedly apply the NNS policy with more LNS steps. So higher running time limits can exploit the learned model better. As a result, ND + NNS is able to improve on Neural Diving at higher running time limits.

**Limitations:** 1) Our approach does not currently train models in an end-to-end fashion to directly optimize a final performance metric such as the average primal gap. Approaches based on Reinforcement Learning (RL) and off-line RL may address this limitation. 2) The conditionally-independent model (section 3.3) does not approximate the expert policy perfectly (figure 3). While this architecture supports efficient inference, the restrictive conditional independence assumption also limits its capacity. More powerful architectures, e.g., autoregressive models combined with GCNs, may give better performance by approximating the expert better.

## 7. Related Work

**Learning and MIP Primal Heuristics:** Ding et al. (2020) trains a neural network to predict values for a subset of a MIP’s binary variables, which are then used to significantly reduce the search space by defining an additional constraint that any assignment be within a pre-specified Hamming distance from the predicted values. Similarly, Xavier et al. (2020) learns to warm start a MIP solver specifically for the electric grid *unit commitment* problem using a  $k$ -Nearest Neighbor binary classifier to predict values for a subset of

binary variables in the MIP. Both works are similar in spirit to Neural Diving, and shares its weakness that the model is applied in a one-shot manner at test time without the ability to iteratively improve the assignment. ND + NNS addresses this weakness.

Learning has also been used to choose among an ensemble of *existing* primal heuristics in a complete solver. Khalil et al. (2017b) learn a binary classifier to predict whether applying a primal heuristic at a search tree node will improve the current best assignment. (Hendel, 2018) formulate a multi-armed bandit approach to learn a switching policy online. This is complementary to our approach of constructing neural primal heuristics for a given application and can be combined by adding the neural heuristics to the ensemble.

**Other Learning Approaches for MIPs:** Several works have used learning to improve tree search in a complete solver (He et al., 2014; Khalil et al., 2016; Alvarez et al., 2017; Gasse et al., 2019; Zarpellon et al., 2020; Gupta et al., 2020). They are most relevant when both an assignment and its optimality gap are required. Here we consider the setting where only the former is required. Learning has been used to predict hyperparameters for MIP solvers, either for *Algorithm Configuration* (Ansótegui et al., 2009; Hutter et al., 2009; 2011; Ansótegui et al., 2015) or *Algorithm Selection* (Kotthoff, 2016; Hutter et al., 2014). Such approaches can be combined with ours.

**Learning for Combinatorial Optimization:** Our work is an instance of learning for combinatorial optimization problems. Some of the earliest works in this area are (Zhang & Dietterich, 1995; Moll et al., 1999; Boyan & Moore, 1997). More recently, deep learning has been applied to the Travelling Salesman Problem (Vinyals et al., 2015; Bello et al., 2016), Vehicle Routing (Kool et al., 2019; Nazari et al., 2018), Boolean Satisfiability (Selsam et al., 2019; Amizadeh et al., 2019; Yolcu & Póczos, 2019), and general graph-structured combinatorial optimization problems (Khalil et al., 2017a; Li et al., 2018). A survey of the topic is available by Bengio et al. (2018).

## 8. Summary & Conclusions

We have proposed a learning-based LNS approach for MIPs. It trains a Neural Diving model to generate an initial assignment, and a Neural Neighborhood Selection policy to select a search neighborhood at each LNS step. The resulting neighborhood can be searched by solving with SCIP a smaller sub-MIP derived from the input MIP. Our approach matches or significantly outperforms all baselines with respect to average primal gap and survival curves on five datasets containing diverse, large-scale, real-world MIPs. It addresses a key limitation of Neural Diving as a standalone primal heuristic by improving the average primal gap at



larger running times. Even larger performance gains can potentially be achieved with end-to-end training of both models, and using more powerful network architectures.

## References

- Abbasi-Yadkori, Y. and Neu, G. Online learning in mdps with side information. *arXiv preprint arXiv:1406.6812*, 2014.
- Achterberg, T., Koch, T., and Martin, A. Miplib 2003. *Operations Research Letters*, 34(4):361–372, 2006.
- Addanki, R., Nair, V., and Alizadeh, M. Neural large neighborhood search. In *Learning Meets Combinatorial Algorithms NeurIPS Workshop*, 2020.
- Alvarez, A., Louveaux, Q., and Wehenkel, L. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29:185–195, 01 2017. doi: 10.1287/ijoc.2016.0723.
- Amizadeh, S., Matushevych, S., and Weimer, M. Learning to solve circuit-SAT: An unsupervised differentiable approach. In *ICLR*, 2019. URL <https://openreview.net/forum?id=BJxgz2R9t7>.
- Ansótegui, C., Sellmann, M., and Tierney, K. A gender-based genetic algorithm for the automatic configuration of algorithms. In Gent, I. P. (ed.), *Principles and Practice of Constraint Programming - CP 2009*, pp. 142–157, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- Ansótegui, C., Malitsky, Y., Samulowitz, H., Sellmann, M., and Tierney, K. Model-based genetic algorithms for algorithm configuration. In *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI’15*, pp. 733–739. AAAI Press, 2015. ISBN 9781577357384.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.
- Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d’horizon, 2018.
- Berthold, T. Primal heuristics for mixed integer programs. Master’s thesis, 2006. URL [https://opus4.kobv.de/opus4-zib/files/1029/Berthold\\_Primal\\_Heuristics\\_For\\_Mixed\\_Integer\\_Programs.pdf](https://opus4.kobv.de/opus4-zib/files/1029/Berthold_Primal_Heuristics_For_Mixed_Integer_Programs.pdf).
- Berthold, T. Rens - relaxation enforced neighborhood search. Technical Report 07-28, ZIB, Takustr. 7, 14195 Berlin, 2007.
- Berthold, T., Heinz, S., Pfetsch, M., and Vigerske, S. Large neighborhood search beyond mip. 2012.
- Boyan, J. A. and Moore, A. W. Using prediction to improve combinatorial optimization search. In *In Proc. of 6th Int’l Workshop on Artificial Intelligence and Statistics*, 1997.
- Cheng, C.-H., Nührenberg, G., and Ruess, H. Maximum resilience of artificial neural networks. In D’Souza, D. and Narayan Kumar, K. (eds.), *Automated Technology for Verification and Analysis*, pp. 251–268. Springer, 2017.
- Danna, E., Rothberg, E., and Pape, C. L. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, Jan 2005.
- Ding, J., Zhang, C., Shen, L., Li, S., Wang, B., Xu, Y., and Song, L. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *AAAI*, 2020. URL <https://www.aaai.org/Papers/AAAI/2020GB/AAAI-DingJ.6745.pdf>.
- Fischetti, M. and Lodi, A. Local branching. *Mathematical Programming*, 98:23–47, 09 2003. doi: 10.1007/s10107-003-0395-5.
- Fischetti, M., Glover, F., and Lodi, A. The feasibility pump. *Mathematical Programming*, 104:91–104, 09 2005. doi: 10.1007/s10107-004-0570-3.
- Gamrath, G., Anderson, D., Bestuzheva, K., Chen, W.-K., Eifler, L., Gasse, M., Gemander, P., Gleixner, A., Gottwald, L., Halbig, K., et al. The SCIP optimization suite 7.0. 2020.
- Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 15554–15566, 2019.
- Ghosh, S. Dins, a mip improvement heuristic. In *International Conference on Integer Programming and Combinatorial Optimization*, pp. 310–323. Springer, 2007.
- Gleixner, A., Hendel, G., Gamrath, G., Achterberg, T., Bastubbe, M., Berthold, T., Christophel, P. M., Jarck, K., Koch, T., Linderoth, J., Lübbecke, M., Mittelman, H. D., Ozyurt, D., Ralphs, T. K., Salvagnin, D., and Shinano, Y. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. Technical report, Optimization Online, July 2019. URL [http://www.optimization-online.org/DB\\_FILE/2019/07/7285.html](http://www.optimization-online.org/DB_FILE/2019/07/7285.html).

- Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- Gori, M., Monfardini, G., and Scarselli, F. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pp. 729–734. IEEE, 2005.
- Gupta, P., Gasse, M., Khalil, E., Mudigonda, P., Lodi, A., and Bengio, Y. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33, 2020.
- Hallak, A., Di Castro, D., and Mannor, S. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.
- Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pp. 1024–1034, 2017.
- He, H., Daume III, H., and Eisner, J. M. Learning to search in branch and bound algorithms. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3293–3301. Curran Associates, Inc., 2014.
- Hendel, G. Adaptive large neighborhood search for mixed integer programming. *Mathematical Programming Computation*, 2018. under review.
- Hojabri, H., Gendreau, M., Potvin, J.-Y., and Rousseau, L.-M. Large neighborhood search with constraint programming for a vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 92, 2018.
- Hottung, A. and Tierney, K. Neural large neighborhood search for the capacitated vehicle routing problem. *arXiv preprint arXiv:1911.09539*, 2019.
- Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. Paramils: An automatic algorithm configuration framework. *J. Artif. Int. Res.*, 36(1):267–306, September 2009. ISSN 1076-9757.
- Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization, LION’05*, pp. 507–523. Springer-Verlag, 2011. ISBN 9783642255656.
- Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79 – 111, 2014. ISSN 0004-3702.
- Khalil, E., Le Bodic, P., Song, L., Nemhauser, G., and Dilkina, B. Learning to branch in mixed integer programming. In Schuurmans, D. and Wellman, M. (eds.), *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 724–731, 2016. URL <http://www.aaai.org/Conferences/AAAI/aaai16.php>.
- 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, 2017a.
- Khalil, E. B., Dilkina, B., Nemhauser, G. L., Ahmed, S., and Shao, Y. Learning to run heuristics in tree search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017b. doi: 10.24963/ijcai.2017/92. URL <https://doi.org/10.24963/ijcai.2017/92>.
- Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.
- Knueven, B., Ostrowski, J., and Watson, J.-P. On mixed integer programming formulations for the unit commitment problem. *Optimization Online Repository*, 11 2018. URL [http://www.optimization-online.org/DB\\_FILE/2018/11/6930.pdf](http://www.optimization-online.org/DB_FILE/2018/11/6930.pdf).
- Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ByxBFRqYm>.
- Kotthoff, L. *Algorithm Selection for Combinatorial Search Problems: A Survey*, pp. 149–190. Springer International Publishing, Cham, 2016.
- Lee, J. and Stuckey, P. Course on solving algorithms for discrete optimization, lecture 3.4.7 large neighbourhood search, 2021. URL <https://www.coursera.org/lecture/solving-algorithms-discrete-optimization/3-4-7-large-neighbourhood-search-brB2N>.
- Li, Z., Chen, Q., and Koltun, V. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems*, pp. 539–548, 2018.
- Lodi, A. Local Branching: A Tutorial. In *MIC*, August 2003. URL [http://www.or.deis.unibo.it/research\\_pages/ORinstances/mic2003-lb.pdf](http://www.or.deis.unibo.it/research_pages/ORinstances/mic2003-lb.pdf).
- Moll, R., Barto, A. G., Perkins, T. J., and Sutton, R. S. Learning instance-independent value functions to enhance local search. In Kearns, M. J., Solla, S. A., and Cohn, D. A.

- (eds.), *Advances in Neural Information Processing Systems 11*, pp. 1017–1023. MIT Press, 1999.
- 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, 2020. URL <https://arxiv.org/abs/2012.13349>.
- Nazari, M., Oroojlooy, A., Snyder, L., and Takac, M. Reinforcement learning for solving the vehicle routing problem. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 31, pp. 9839–9849. Curran Associates, Inc., 2018. URL <https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf>.
- Perron, L., Shaw, P., and Furnon, V. Propagation guided large neighborhood search. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, CP’04, pp. 468–481, Berlin, Heidelberg, 2004. Springer-Verlag. ISBN 9783540232414. doi: 10.1007/978-3-540-30201-8\_35. URL [https://doi.org/10.1007/978-3-540-30201-8\\_35](https://doi.org/10.1007/978-3-540-30201-8_35).
- Pisinger, D. and Ropke, S. Large neighborhood search. In Gendreau, M. and Potvin, J.-Y. (eds.), *Handbook of Metaheuristics*, pp. 399–419, Boston, MA, 2010.
- Rothberg, E. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.
- Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.
- Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. URL [https://openreview.net/forum?id=HJMC\\_iA5tm](https://openreview.net/forum?id=HJMC_iA5tm).
- Shaw, P. Using constraint programming and local search methods to solve vehicle routing problems. In *International conference on principles and practice of constraint programming*, pp. 417–431. Springer, 1998.
- Smith, S. L. and Imeson, F. Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87, 2017.
- Song, J., Lanka, R., Yue, Y., and Dilkina, B. A general large neighborhood search framework for solving integer programs. *arXiv preprint arXiv:2004.00422*, 2020.
- Tjeng, V., Xiao, K. Y., and Tedrake, R. Evaluating robustness of neural networks with mixed integer programming. In *ICLR*, 2019. URL <https://openreview.net/forum?id=HyGIdiRqtm>.
- Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28, pp. 2692–2700, 2015. URL <https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf>.
- Xavier, A. S., Qiu, F., and Ahmed, S. Learning to solve large-scale security-constrained unit commitment problems. *INFORMS Journal on Computing*, 2020.
- Yolcu, E. and Póczos, B. Learning local search heuristics for boolean satisfiability. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL <https://proceedings.neurips.cc/paper/2019/file/12e59a33dealbf0630f46edfe13d6ea2-Paper.pdf>.
- Zarpellon, G., Jo, J., Lodi, A., and Bengio, Y. Parameterizing branch-and-bound search trees to learn branching policies. *arXiv preprint arXiv:2002.05120*, 2020.
- Zhang, W. and Dietterich, T. G. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI’95*, pp. 1114–1120, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

Dataset	$N$	$r$	$n_l$	lr	$m$
Prod. Planning	7189	0.05	5	$5 \times 10^{-4}$	500
Prod. Packing	2924	0.01	20	$10^{-3}$	500
MIPLIB	553	0.05	5	$10^{-3}$	500
NN Verification	2554	0.4	7	$10^{-3}$	100
Electric Grid Opt.	11444	0.4	10	$10^{-3}$	500

Table 1: Hyperparameters for data generation and training.

Dataset	$r_0$	$\tau$	$t$ (sec)
Prod. Planning	0.05	2.0	300
Prod. Packing	0.15	1.0	300
MIPLIB	0.25	2.0	300
NN Verification	0.4	1.5	60
Electric Grid Opt.	0.4	1.5	300

Table 2: Hyperparameters for evaluation.

## A. Technical Appendix

### A.1. Expert data generation

To generate trajectories of the expert policy, we computed 10 steps of the local branching procedure as described in Section 3.2 for each instance in the training datasets. The Hamming distance  $\eta$  of the local branching step was picked as a fixed fraction of the number of variables  $n$ , i.e.  $\eta = r \cdot n$ . For each step computation we used SCIP with default parameters, with a time limit of 3 hours. The parameter  $r$  was chosen using the validation set, and the results are shown in table 1.

### A.2. Training procedure

A model was trained for each dataset on a single NVIDIA V100 GPU for up to 400k steps, taking up to 48 hours. We used grid-search to tune hyperparameters (number of GCN layers  $n_l$ , learning rate (lr) and its decay, where we decayed the learning rate by 0.9 every  $m$  steps) on the validation set. The resulting choices, as well as the training set size  $N$  are shown in the table below. Node embeddings were of size 64 for each model, and were computed using a 2-layer MLP with 64 hidden units per layer.

### A.3. Evaluation procedure

During evaluation, we used the trained model as described in Section 3.5. For all datasets, we performed 50 steps of LNS, and the adaptation factor for the neighbourhood size was fixed to  $\alpha = 1.5$  and sampling probability bias to  $\epsilon = 10^{-3}$ . Hyperparameters (initial fraction of unassigned variables  $r_0$ , sampling temperature  $\tau$ , and search time limit per step  $t$ ) were optimized using the validation set to the values in table 2.

### A.4. Input representation

We use input features from (Gasse et al., 2019) along with the incumbent solution. The code for computing the features from (Gasse et al., 2019) is available at <https://github.com/ds4dm/learn2branch>.

### A.5. Calibrated time

The total evaluation workload across all datasets and comparisons requires a large amount of compute. To meet the compute requirements, we use a shared, heterogeneous compute cluster. Accurate running time measurement on such a cluster is difficult because the tasks may be scheduled on machines with different hardware, and interference from other unrelated tasks on the same machine increases the variance of solve times. To improve accuracy, for each solve task, we periodically solve a small *calibration MIP* on a different thread from the solve task on the same machine. We use an estimate of the number of calibration MIP solves during the solve task on the same machine to measure time, which is significantly less sensitive to hardware heterogeneity and interference. This quantity is then converted into a *calibrated time* value using the



Table 3: Optimality gap thresholds used for plotting survival curves for the datasets in our evaluation.

Dataset	Target Optimality Gap
Neural Network Verification	0.05
Production Packing	0.01
Production Planning	0.03
Electric Grid Optimization	0.0001
MIPLIB	0

calibration MIP’s solve time on a reference machine.

We define the speed of a machine on which a MIP solving job runs to be

$$\text{Speed} = \frac{1}{\text{Wall clock time to solve calibration MIP}}. \quad (10)$$

For each periodic measurement of calibrated time, we estimate the speed  $K$  times and use the average.  $K$  is set to be the number of samples needed to estimate mean speed with 95% confidence, with a minimum of 3 samples and a maximum of 30. The elapsed calibrated time  $\Delta t_{\text{calibrated}}$  since the last measurement is

$$\Delta t_{\text{calibrated}} = \text{Speed} \times \Delta t_{\text{wallclock}}, \quad (11)$$

where  $\Delta t_{\text{wallclock}}$  is the elapsed wallclock time since the last measurement. We use the MIP named *vpm2* from MIPLIB2003 (Achterberg et al., 2006) as the calibration MIP.

Note that the above definition of calibrated time does not have a time unit. Instead it is (in effect) a count of the calibrated MIP solves during the evaluation solve task. To give it a unit of seconds, one can choose a reference machine with respect to which evaluation solve times will be reported, accurately measure the calibration MIP’s solve time on it (without other tasks interfering), and multiply the calibrated time in equation 11 by the reference machine’s estimated calibration MIP solve time. The resulting quantity has a unit of seconds. It can be interpreted as the time the evaluation solve task would have taken if it ran on the reference machine. We select Intel Xeon 3.50GHz CPU with 32GB RAM as the reference machine. All the calibrated time results in the paper are expressed with respect to the reference machine in seconds.

#### A.6. Dataset Details

The target optimality gap used for each dataset in our evaluation as SCIP’s stopping criterion for a MIP solve is given in table 3. In the case of Neural Network Verification, the actual criterion used in the application is to stop when the objective value becomes negative, but this is not expressible as a constant target gap across all instances. In order to treat all datasets consistently, we have selected a gap of 0.05. Additional information about the applications that the datasets are extracted from is provided in table 4.

Figure 4 shows the MIP sizes for the datasets used in our evaluation after presolving using SCIP 7.0.1. Note that, among the application-specific datasets, the Production Planning dataset is the most heterogeneous (along with MIPLIB) in terms of instance sizes. Table 5 summarizes the characteristics of those datasets before and after presolving with the SCIP 7.0.1 solver. The number of constraints and variables ranges different orders of magnitude across the datasets. It is worth noting that during presolving some instances might be deemed infeasible and, hence, dropped from the dataset.

#### A.7. Results on NeurIPS’21 Competition Datasets

Figure 5 presents preliminary results based on limited hyperparameter tuning for Neural Diving and Neural Neighborhood Search on the three datasets (Item Placement, Anonymous, Load Balancing) from the NeurIPS’21 competition on [Machine Learning for Combinatorial Optimization](#). The Anonymous dataset is very small, with only 98 training cases. Not surprisingly, Neural Diving does not perform well on this dataset, and Neural Neighborhood Search does not provide any benefit over Random Neighborhood Search. Item Placement is a challenging dataset for SCIP. We have observed that even with several hours of running time for SCIP during data collection, SCIP still frequently reports very high optimality gaps (often  $> 100\%$ ). This affects both the quality of the data collected to train the ND and NNS models, as well as the test time

Table 4: Description of the five datasets we use in the paper. Please see (Nair et al., 2020) for more details.

Name	Description
Neural Network Verification	Verifying whether a neural network is robust to input perturbations can be posed as a MIP (Cheng et al., 2017; Tjeng et al., 2019). Each input on which to verify the network gives rise to a different MIP. In this dataset, a convolutional neural network is verified on each image in the MNIST dataset, giving rise to a corresponding dataset of MIPs.
Production Packing	A packing optimization problem solved in a large-scale production system.
Production Planning	A planning optimization problem solved in a large-scale production system.
Electric Grid Optimization	Electric grid operators optimize the choice of power generators to use at different times of the day to meet electricity demand by solving a MIP. This dataset is constructed for one of the largest grid operators in the US, PJM, using publicly available data about generators and demand, and the MIP formulation in (Knueven et al., 2018).
MIPLIB	Heterogeneous dataset containing ‘hard’ instances of MIPs across many different application areas that is used as a long-standing standard benchmark for MIP solvers (Gleixner et al., 2019). We use instances from both the 2010 and 2017 versions of MIPLIB.

 Table 5: Statistics (**median** in the first row and **maximum** in the second row) of constraints and variables (per type) for the different datasets before / after presolving using SCIP 7.0.1. The total number of nodes in the bipartite graph representation corresponds to the sum of the number of constraints and variables.

Dataset	Constraints	Variables	Binary	Integer (non-binary)	Continuous
<i>Neural Network Verification</i>	6531/1407	7142/1629	171/170	0/0	6972/1455
(max)	7123/2089	7535/2231	364/364	0/0	7171/1921
<i>Production Packing</i>	36905/24495	10046/8919	3773/3437	0/0	6231/5421
(max)	47414/33071	11233/10161	4120/3771	0/0	7113/6390
<i>Production Planning</i>	11910/478	9884/404	833/119	462/119	8337/136
(max)	1722678/338508	1548582/335783	117485/22626	58782/23628	1374182/335783
<i>Electric Grid Optimization</i>	61851/45834	60720/58389	42240/42147	0/0	18768/16623
(max)	67086/50908	117792/115098	98112/98021	0/0	21456/20184
<i>MIPLIB</i>	7706/4388	11090/9629	4450/3816	0/0	218/96
(max)	19912111/888363	38868107/6338552	20677405/6338552	549428/0	38335410/700426

performance of ND and NNS by using SCIP to solve the sub-MIPs, which explains the poor performance of all approaches on this dataset. On Load Balancing, ND significantly outperforms SCIP, which shows that learning is useful for this dataset. However, NNS and RNS both fail to provide any improvements over ND. We believe further tuning the hyperparameters can improve the results on Load Balancing.

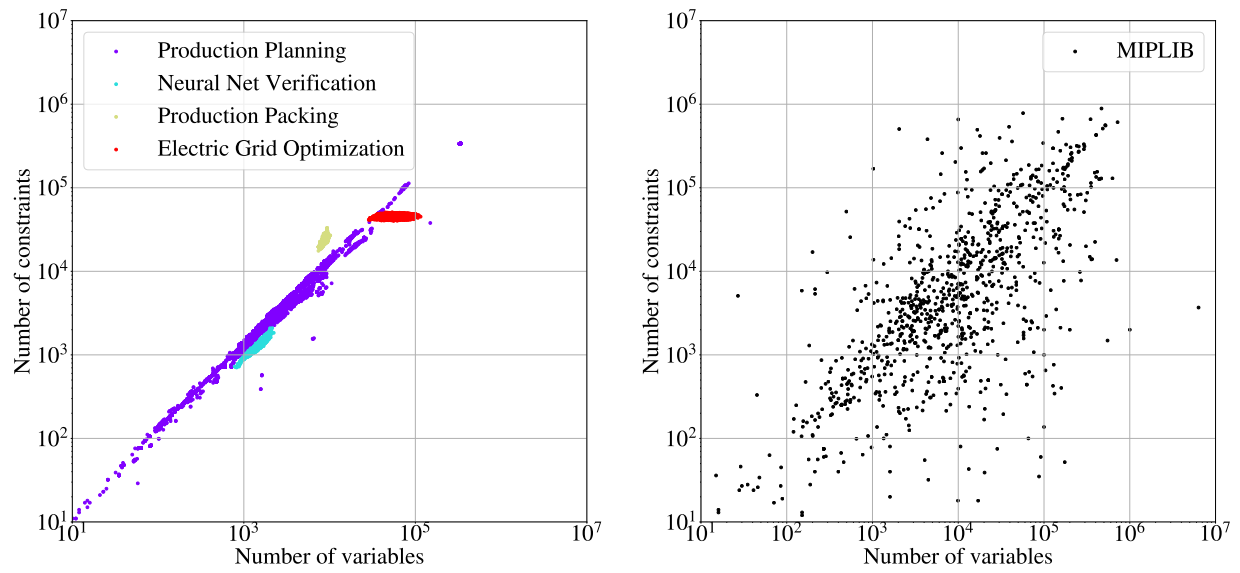
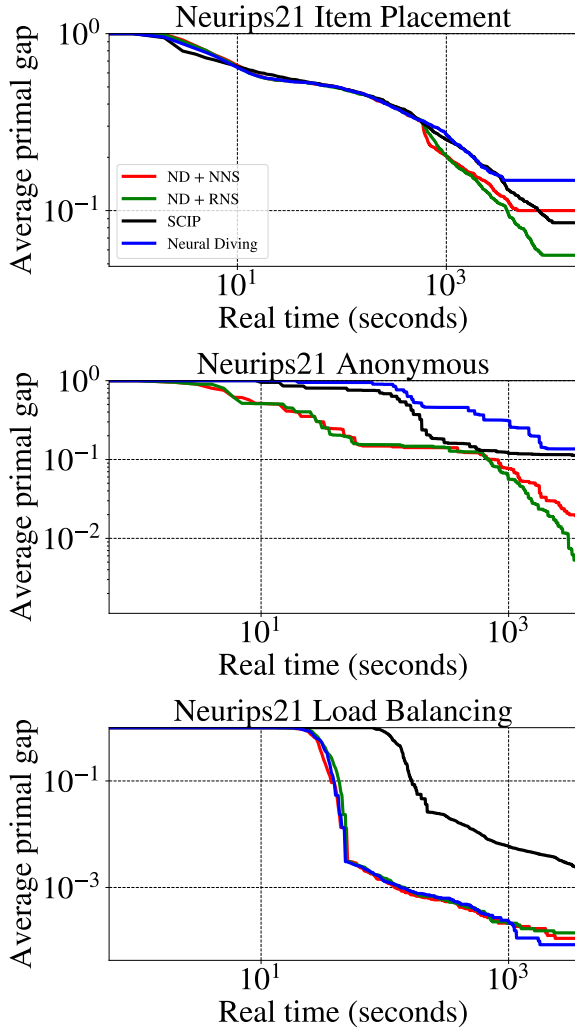
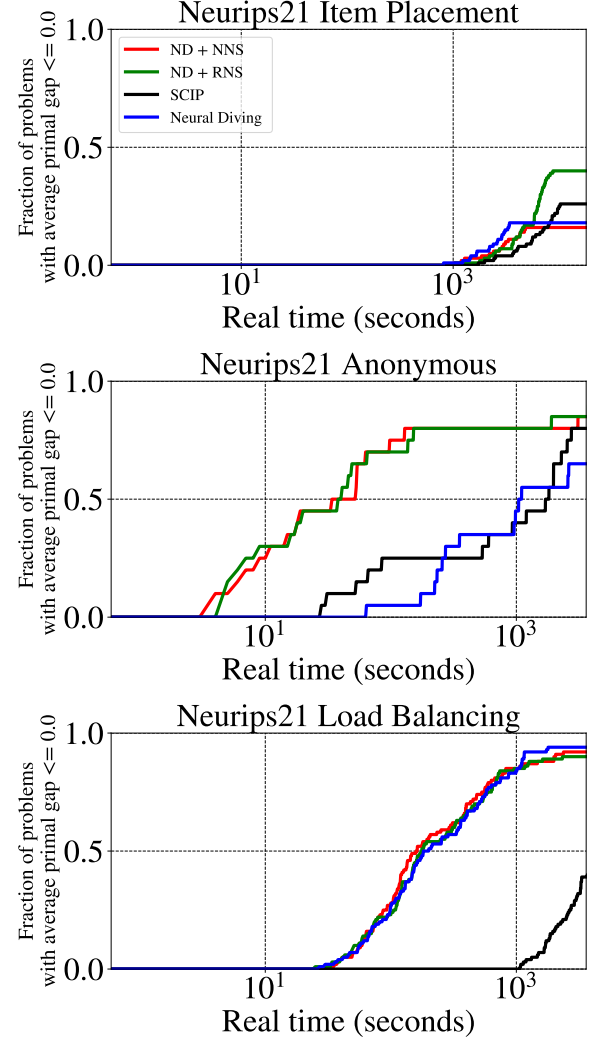


Figure 4: Number of variables versus number of constraints *after presolving* using SCIP 7.0.1 for the application-specific datasets (left) and MIPLIB (right) used in our evaluation. Presolving significantly reduces the problem size compared to that of the raw input MIP.



(a) Fraction of test set instances with primal gap below a dataset-specific threshold, as a function of running time for five datasets. (Note: For Production Planning, several curves closely overlap.)



(b) Test set average primal gap (see section 4.2, lower is better) as a function of running time for five datasets.

Figure 5: Results for the NeurIPS'21 Machine Learning for Combinatorial Optimization competition datasets.