One Model, Any CSP: Graph Neural Networks as Fast Global Search Heuristics for Constraint Satisfaction

Jan Tönshoff[*], **Berke Kisin**, **Jakob Lindner** and **Martin Grohe**
RWTH Aachen University
{toenshoff, grohe}@informatik.rwth-aachen.de
{berke.kisin, jakob.lindner}@rwth-aachen.de

Abstract

We propose a universal Graph Neural Network architecture which can be trained as an end-2-end search heuristic for any Constraint Satisfaction Problem (CSP). Our architecture can be trained unsupervised with policy gradient descent to generate problem specific heuristics for any CSP in a purely data driven manner. The approach is based on a novel graph representation for CSPs that is both generic and compact and enables us to process every possible CSP instance with one GNN, regardless of constraint arity, relations or domain size. Unlike previous RL-based methods, we operate on a global search action space and allow our GNN to modify any number of variables in every step of the stochastic search. This enables our method to properly leverage the inherent parallelism of GNNs. We perform a thorough empirical evaluation where we learn heuristics for well known and important CSPs, both decision and optimisation problems, from random data, including graph coloring, MAXCUT, and MAX-k-SAT, and the general RB model. Our approach significantly outperforms prior end-2-end approaches for neural combinatorial optimization. It can compete with conventional heuristics and solvers on test instances that are several orders of magnitude larger and structurally more complex than those seen during training.

1 Introduction

Constraint Satisfaction Problems (CSP) are a ubiquitous framework for specifying combinatorial search and optimization problems. They include many of the best-known NP-hard problems such as Boolean satisfiability (SAT), graph coloring (COL) and maximum cut (MAXCUT) and can flexibly adapted to model specific application dependent problems. CSP solution strategies range from general solvers based on methods such as constraint propagation or local search (see [Russell *et al.*, 2020], Chapter 6) to specialized solvers for individual problems like SAT (see [Biere *et al.*, 2021]). In recent years, there is a growing interest in applying deep learning methods to combinatorial problems including many CSPs (e.g. [Khalil *et al.*, 2017], [Selsam *et al.*, 2018], [Tönshoff *et al.*, 2021]). The main motivation for these approaches is to learn novel heuristics from data rather than crafting them by hand.

Graph Neural Networks (see Gilmer *et al.* 2017) have emerged as an effective tool for learning powerful, permutation invariant functions on graphs using deep neural networks, and they have become the primary architecture for neural combinatorial optimization. Problem instances are modelled as graphs and then mapped to approximate solutions with GNNs. However, most methods use graph reductions and GNN architectures that are problem specific. Transferring them across combinatoral tasks requires considerable engineering, limiting their use cases. *Designing a generic neural network architecture and training procedure for the general CSP formalism offers a powerful alternative.* Then learning heuristics for any specific CSP becomes a purely data driven process requiring no specialized graph reduction or architecture search.

We propose a novel GNN based reinforcement learning approach to general constraint satisfaction. The main contributions of our method called ANYCSP[1] can be summarized as follows: We define a new graph representation for general CSP instances which is generic and well suited as an input for recurrent GNNs. It allows us to directly process all CSPs with one unified architecture and no prior reduction to more specific CSPs, such as SAT. In particular, one ANYCSP model can take every CSP instance as input, even those with domain sizes, constraint arities, or relations not seen during training. Training is unsupervised using policy gradient ascent with a carefully tailored reward scheme that encourages exploration and prevents the search to get stuck in local maxima. During inference, a trained ANYCSP model iteratively searches the space of assignments to the variables of the CSP instance for an optimal solution satisfying the maximum number of constraints. Crucially, the search is *global*; it allows transitions between any two assignments in a single step. To enable this global search we use *policy gradient* methods to handle the exponential action spaces efficiently. This design choice speeds up the search substantially, especially on large instances. We thereby overcome a primary bottleneck of previous neural ap-

[*]This work was supported by the German Research Foundation (DFG) under grants GR 1492/16-1 and KI 2348/1-1 "Quantitative Reasoning About Database Queries".

[1]**A**re **N**eural Networks great heuristics? **Y**es, for **CSP**s.

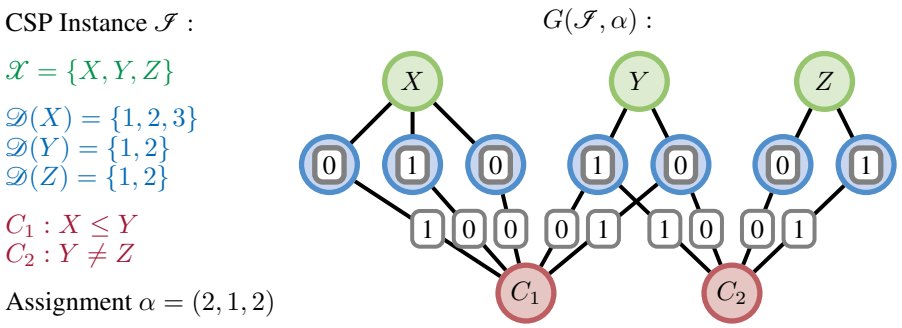

Figure 1: Example of the **constraint value graph** $G(\mathscr{I}, \alpha)$ for a given CSP instance $\mathscr{I}$ and an assignment α. The graph contains vertices for the variables, values and constraints of $\mathscr{I}$. Each value is connected to its variable and labeled with the assignment α. Each constraint is connected to the values of its variables. This edge set is labeled such that a label of 1 for edge (C, v) states that choosing value v will satisfy the constraint C if no other variables involved in C change their values.

proaches based on local search, which only flip the values of a single or a few variables in each step. GNN based local search tends to scale poorly to large instances as one GNN forward pass takes significantly more time than one step of classical local search heuristics. ANYCSP addresses this by exploiting the GNNs inherent parallelism to refine solutions globally.

We evaluate ANYCSP by learning heuristics for a range of important CSPs: COL, SAT, MAXCUT and general CSP benchmark instances. We demonstrate that our method achieves a substantial increase in performance over prior GNN approaches and can compete with conventional algorithms. ANYCSP models trained on small random graph coloring problems are on par with state-of-the-art coloring heuristics on structured benchmark instances. On MAX-k-SAT, our method scales to test instances 100 times larger than the training data, where it outperforms state-of-the-art search heuristics despite performing 1000 times fewer search iterations.

2 Related Work

In this paper, we are primarily interested in end-2-end approaches which directly predict approximate solutions for combinatorial problems with neural networks. Another line of work integrates deep learning components into more traditional solvers (i.e. [Gasse *et al.*, 2019; Zhang *et al.*, 2020]). While also quite interesting, these are at most loosely related to our work, and we leave them out of the following discussion. For a comprehensive overview on neural combinatorial optimisation, we refer to [Cappart *et al.*, 2021].

Early work was done by [Bello *et al.*, 2016], who learned TSP heuristics with Pointer Networks [Vinyals *et al.*, 2015] and policy gradient descent. Several extensions of these ideas have since been proposed based on attention [Kool *et al.*, 2018] and GNNs [Joshi *et al.*, 2020]. [Khalil *et al.*, 2017] propose a general method for graph problems, such as MAXCUT or Minimum Vertex Cover. They model the expansion of partial solutions as a reinforcement learning task and train a GNN with Q-learning to iteratively construct approximate solutions.

A related group of approaches models local modifications to complete solutions as actions of a reinforcement learning problem. A GNN is then trained as a local search heuristic that iteratively improves candidate solutions through local changes. Methods following this concept are RLSAT [Yolcu and Póczos, 2019] for SAT, ECO-DQN [Barrett *et al.*, 2020] for MAXCUT, LS-DQN [Yao *et al.*, 2021] for graph partitioning problems and TSP as well as BiHyb [Wang *et al.*, 2021] for graph problems based on selecting and modifying edges. Like conventional search heuristics, these architectures can be applied for any number of search iterations to refine the solution. A shared drawback on large instances is the relatively high computational cost of GNNs, which slows down the search substantially when compared to classical algorithms. ECORD [Barrett *et al.*, 2022] addresses this issue for MAXCUT by applying a GNN once before the local search, which is carried out by a faster GRU-based architecture without message passes. We address the same problem, but not by iterating faster, but by allowing global modifications in each iteration.

A fundamentally different approach considers soft relaxations of the underlying problems which can optimized directly with SGD. Examples of this concept are PDP [Amizadeh *et al.*, 2019] for SAT and RUNCSP [Tönshoff *et al.*, 2021] for all binary CSPs with fixed constraint language. These architectures can predict completely new solutions in each iteration but the relaxed differentiable objectives used for training typically do not capture the full hardness of the discrete problem.

3 Preliminaries

A *CSP instance* is a triple $\mathscr{I} = (\mathscr{X}, \mathscr{D}, \mathscr{C})$, where $\mathscr{X}$ is a finite set of variables, $\mathscr{D}$ assigns to each variable $X \in \mathscr{X}$ a finite set $\mathscr{D}(X)$, the *domain* of X, and $\mathscr{C}$ is a set of constraints $C = (s^C, R^C)$, where for some $k \geq 1$, the *scope* $s^C = (X_1, \ldots, X_k) \in \mathscr{X}^k$ is a tuple of variables and $R^C \subseteq \mathscr{D}(X_1) \times \ldots \times \mathscr{D}(X_k)$ is a k-ary relation over the corresponding domains. We always assume that the variables in the scope s^C of a constraint C are mutually distinct; we can easily transform an instance not satisfying this condition into one that does by adapting the relation R^C accordingly.

Slightly abusing terminology, we call a pair (X, d) where $X \in \mathscr{X}$ and $d \in \mathscr{D}(X)$ a *value* for variable X. For all $X \in \mathscr{X}$ we let $\mathscr{V}_X = \{X\} \times \mathscr{D}(X)$ be the set of all values for X, and we let $\mathscr{V} = \bigcup_{X \in \mathscr{X}} \mathscr{V}_X$ be the set of all values. We usually denote values by v. Working with these values instead of domain elements is convenient because the sets $\mathscr{V}_X$,

for $X \in \mathscr{X}$, are mutually disjoint, whereas the domains $\mathscr{D}(X)$ are not necessarily.

An *assignment* for a CSP instance $\mathscr{I} = (\mathscr{X}, \mathscr{D}, \mathscr{C})$ is a mapping α that assigns a domain element $\alpha(X) \in \mathscr{D}(X)$ to each variable X. Alternatively, we may view an assignment as a subset $\alpha \subseteq \mathscr{V}$ that contains exactly one value from each $\mathscr{V}_X$. Depending on the context, we use either view, and we synonymously write $\alpha(X) = d$ or $(X, d) \in \alpha$. An assignment α *satisfies* a constraint $C = ((X_1, \dots, X_k), R)$ (we write $\alpha \models C$) if $(\alpha(X_1), \dots, \alpha(X_k)) \in R$, and α *satisfies* $\mathscr{I}$, or is a *solution* to $\mathscr{I}$, if it satisfies all constraints in $\mathscr{C}$. The objective of a CSP is to decide if a given instance has a satisfying assignment and to find one if it does. To distinguish this problem from the maximization version introduced below, we sometimes speak of the *decision version*. Specific CSPs such as Boolean satisfiability or graph coloring problems are obtained by restricting the instances considered.

We define the *quality* $Q_{\mathscr{I}}(\alpha)$ of an assignment α to be the fraction of constraints in $\mathscr{C}$ satisfied by α: $Q_{\mathscr{I}}(\alpha) = |\{C | C \in \mathscr{C}, \alpha \models C\}| / |\mathscr{C}|$. An assignment α is *optimal* if it maximizes $Q_{\mathscr{I}}(\alpha)$ for the instance $\mathscr{I}$. The goal of the maximisation problem MAXCSP is to find an optimal assignment for a given instance. A *soft assignment* for a CSP instance $\mathscr{I}$ is a mapping $\varphi : \mathscr{V} \to [0, 1]$ such that $\sum_{v \in \mathscr{V}_X} \varphi(v) = 1$ for all $X \in \mathscr{X}$. We interpret the numbers $\varphi(v)$ as probabilities and say that an assignment α is *sampled* from a soft assignment φ (we write $\alpha \sim \varphi$) if for each variable $X \in \mathscr{X}$ we independently draw a value $v \in \mathscr{V}_X$ with probability $\varphi(v)$.

4 Method

With every CSP instance $\mathscr{I} = (\mathscr{X}, \mathscr{D}, \mathscr{C})$ we associate a tripartite graph with vertex set $\mathscr{X} \cup \mathscr{V} \cup \mathscr{C}$, where $\mathscr{V}$ is the set of values defined in the previous section, and two kinds of edges: *variable edges* (X, v) for all $X \in \mathscr{X}$ and $v \in \mathscr{V}_X$, and *constraint edges* (C, v) for all $C \in \mathscr{C}$ and $v \in \mathscr{V}_X$ for some X in the scope of C. This graph representation is more or less standard; one slightly unusual feature is that we introduce edges from constraints directly to the values and not to the variables. This will be important in the next step, where information about the constraint relations R^C is compactly encoded through a binary labeling of the constraint edges. For each assignment α we introduce a vertex labeling L_V and an edge labeling L_E. The vertex labeling L_V is a binary encoding of α, that is, $L_V(v) = 1$ if $v \in \alpha$ and $L_V(v) = 0$ for each $v \in \mathscr{V} \setminus \alpha$. The edge labeling L_E encodes how changes to α affect each constraint. For every constraint $C \in \mathscr{C}$ and value $(X_i, d) \in \mathscr{V}_{X_i}$ of variable X_i in the scope $(X_1, \dots, X_k)$ of C we define the edge label to be $L_E(C, v) = 1$ if

$$(\alpha(X_1), \dots, \alpha(X_{i-1}), d, \alpha(X_{i+1}), \dots, \alpha(X_k)) \in R^C.$$

and $L_E(C, v) = 0$ otherwise. Intuitively, the edge labels encode for each constraint edge (C, v) whether or not choosing the value v for its variable would satisfy C under the condition that all other variables involved in C retain their current value in α. We call the labeled graph $G(\mathscr{I}, \alpha)$ obtained this way the *constraint value graph* of $\mathscr{I}$ at α. Figure 1 provides a visual example of our construction.

A complete encoding of each constraint relation would have exponential worst-case space requirements. Our method avoids this issue through a partial encoding using edge labels that dynamically adapts to the current assignment. In Appendix[2] A.5 we provide details on efficiently recomputing the edge labels for a new assignment α and a discussion on global constraints.

4.1 Architecture

We construct a recurrent GNN π_θ that maps constraint value graphs to soft assignments and serves as a trainable policy for our reinforcement-learning setup. Here, the real vector θ contains the trainable parameters of π_θ. The input of π_θ in iteration t is the current graph $G(\mathscr{I}, \alpha^{(t-1)})$ and recurrent vertex states $h^{(t-1)}$. The output is a new soft assignment $\varphi^{(t)}$ for $\mathscr{I}$ as well as updated recurrent states:

$$\varphi^{(t)}, h^{(t)} = \pi_\theta\big(G(\mathscr{I}, \alpha^{(t-1)}), h^{(t-1)}\big) \tag{1}$$

The next assignment $\alpha^{(t)}$ can then be sampled from $\varphi^{(t)}$ before the process is repeated. Here, we will provide an overview of the GNN architecture while we give a detailed formal description in Appendix A.

In a nutshell, our architecture is a recurrent heterogeneous GNN that uses distinct trainable functions for each of the three vertex types in the constraint value graph. The main hyperparameters of π_θ are the latent dimension $d \in \mathbb{N}$ and the aggregation function $\bigoplus$ which we either choose as an element-wise SUM, MEAN or MAX function. We usually found MAX to perform best on decision problems while MEAN worked better for maximization tasks. This coincides with observations of [Joshi *et al.*, 2020].

π_θ associates a recurrent state $h^{(t)}(v) \in \mathbb{R}^d$ with each value $v \in \mathscr{V}$ and uses a GRU cell to update these states after each round of message passing. Variables and constraints do not have recurrent states. We did consider versions with stateful constraints and variables, but these did not perform better while being slower. All remaining functions for message generation and combination are parameterized by standard MLPs with at most one hidden layer. In each iteration t, π_θ performs 4 directed message passes in the following order: (1) values to constraints, (2) constraints to values, (3) values to variables, (4) variables to values. The first two message passes incorporate the node and edge labels and enable the values to gather information about how changes to the current assignment effect each constraint. The final two message passes allow the values of each domain to negotiate the next variable assignment. Note that this procedure is carried out *once* in each search iteration t. As the recurrent states can carry aggregated information across search iterations we found a single round of message passes per iteration sufficient.

Finally, π_θ generates a new soft assignment $\varphi^{(t)}$. To this end, each value $v \in \mathscr{V}_X$ of each variable X predicts a scalar real number $o^{(t)}(v) = \mathbf{O}(h^{(t)}(v))$ from its updated latent state with a shared MLP $\mathbf{O} : \mathbb{R}^d \to \mathbb{R}$. We can then apply the softmax function *within each domain* to produce a soft value assignment:

$$\varphi^{(t)}(v) = \frac{\exp\big(o^{(t)}(v)\big)}{\sum_{v' \in \mathscr{V}_X} \exp\big(o^{(t)}(v')\big)} \tag{2}$$

[2]Code and Appendix: https://github.com/toenshoff/ANYCSP

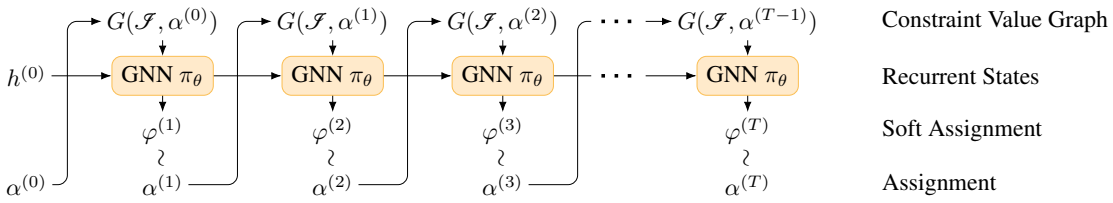

Figure 2: Illustration of a run of ANYCSP on a given CSP instance $\mathscr{I}$. We iteratively apply our policy GNN π_θ to the constraint value graph $G(\mathscr{I}, \alpha^{(t-1)})$ of $\mathscr{I}$ and the current assignment $\alpha^{(t-1)}$. From this we obtain a soft assignment $\varphi^{(t)}$ from which the next assignment $\alpha^{(t)}$ is sampled freely with no restrictions to locality.

This procedure leverages a major strength of our graph construction: By modeling values as vertices we can directly process arbitrary domains with one GNN. For larger domains, we simply add more value vertices to the graph.

4.2 Global Search as an RL Problem

We deploy the policy GNN π_θ as a trainable search heuristic. Note that a single GNN π_θ can search for solutions on any given CSP instance. ANYCSP takes a CSP instance $\mathscr{I}$ and a parameter $T \in \mathbb{N}$ as input and outputs a sequence $\boldsymbol{\alpha} = \alpha^{(0)}, \dots, \alpha^{(T)}$ of assignments for $\mathscr{I}$. The initial assignment $\alpha^{(0)}$ is simply drawn uniformly at random. In each iteration $1 \leq t \leq T$ the policy GNN π_θ is applied to the current constraint value graph $G(\mathscr{I}, \alpha^{(t-1)})$ to generate a new soft assignment $\varphi^{(t)}$. The next assignment $\alpha^{(t)} \sim \varphi^{(t)}$ is then sampled from the predicted soft assignment by drawing a new value $\alpha^{(t)}(X)$ for all variables X independently and in parallel without imposing any restrictions on locality. Any number of variables may change their value in each iteration which makes our method a *global* search heuristic. This allows ANYCSP to modify different parts of the solution simultaneously to speed up the search. Figure 2 provides a visual illustration of the overall process. Formally, our action space is the set of all assignments for the input instance, one of which must be chosen as the next assignment in each iteration t. This set is extremely large for many CSPs, with up to 10^{50} assignments to choose from for some of our training instances. Despite this, we found standard policy gradient descent algorithms to be effective and stable during training.

Rewarding Iterative Improvements. We devise a reward scheme that assigns a real-valued reward $r^{(t)}$ to each generated assignment $\alpha^{(t)}$. A simple approach would be to use the quality $Q_{\mathscr{I}}(\alpha^{(t)})$ as a reward. However, we found that models trained with this reward tend to get stuck in local maxima and have comparatively poor performance. Intuitively, this simple reward scheme immediately punishes the policy for stepping out of a local maximum causing stagnating behavior.

We, therefore, choose a more sophisticated reward system that avoids this issue. First, we define the auxiliary variable $q^{(t)} = \max_{t'<t} Q_{\mathscr{I}}(\alpha^{(t')})$, which tracks the highest quality achieved before iteration t. We then define the reward in iteration t as follows:

$$r^{(t)} = \begin{cases} 0 & \text{if } Q_{\mathscr{I}}(\alpha^{(t)}) \leq q^{(t)}, \\ Q_{\mathscr{I}}(\alpha^{(t)}) - q^{(t)} & \text{if } Q_{\mathscr{I}}(\alpha^{(t)}) > q^{(t)}. \end{cases} \tag{3}$$

The policy earns a positive reward in iteration t if the new assignment $\alpha^{(t)}$ satisfies more constraints than any assignment generated in the previous steps. In this case, the reward is the margin of improvement. Note that the reward is 0 in any step in which the new assignment is not an improvement over the previous best regardless of whether the quality of the solution is increasing or decreasing. This reward is designed to encourage π_θ to yield iteratively improving assignments while being agnostic towards how the assignments change between improvements. Our reward is conceptually similar to that of ECO-DQN [Barrett *et al.*, 2020]. The main difference is that we do not add intermediate rewards for reaching local maxima. Inductively, we observe that the total reward over all iterations is given by $\sum_{t=1}^{T} r^{(t)} = q^{(T+1)} - Q_{\mathscr{I}}(\alpha^{(0)})$. For any input instance $\mathscr{I}$ the total reward is maximal (relative to $Q_{\mathscr{I}}(\alpha^{(0)})$) if and only if the highest achieved quality $q^{(T+1)}$ is the optimal quality for $\mathscr{I}$. In Appendix C we provide an ablation study where we compare our reward scheme to the simpler choice of using $Q_{\mathscr{I}}(\alpha^{(t)})$ directly as a reward.

Markov Decision Process. For a given input $\mathscr{I}$ we model the procedure described so far as a Markov Decision Process $\mathscr{M}(\mathscr{I})$ which will allow us to deploy standard reinforcement learning methods for training: The state in iteration t is given by $s^{(t)} = (\alpha^{(t)}, q^{(t)})$ and contains the current assignment and highest quality achieved before step t. The initial assignment $\alpha^{(0)}$ is drawn uniformly at random and $q^{(0)} = 0$. The space of actions $\mathscr{A}$ is simply the set of all possible assignments for $\mathscr{I}$.[3] The soft assignments produced by the policy π_θ are distributions over this action space. After the next action is sampled from this distribution, the state transition of the MDP is deterministic and updates the state with the chosen assignment and its quality. The reward $r^{(t)}$ at time t is defined as in Equation 3.

Training. During training, we assume that some data generating distribution Ω of CSP instances is given. We aim to find a policy that performs well on this distribution of inputs. Ideally, we need to find the set of parameters θ^* which maximizes the expected total reward if we first draw an instance from Ω and then apply the model to it for $T_{\text{train}} \in \mathbb{N}$ steps:

$$\theta^* = \arg\max_{\theta} \mathop{\mathbf{E}}_{\substack{\mathscr{I}\sim\Omega \\ \boldsymbol{\alpha}\sim\pi_\theta(\mathscr{I})}} \Big[\sum_{t=1}^{T_{\text{train}}} \lambda^{t-1} r^{(t)} \Big] \tag{4}$$

[3] Formally, the state and action space also contain the recurrent states $h^{(t)}$ which we omit for clarity.

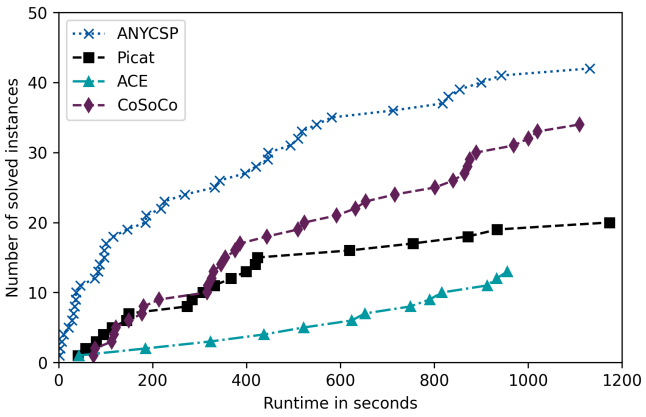

Figure 3: Survival plot for RB50. The x-axis gives the wall clock runtime in seconds. The y-axis counts the cumulative number of instances solved within a given time.

The discount factor $\lambda \in (0, 1]$ and the number of search iterations during training T_{train} are both hyperparameters. Starting with randomly initialized parameters θ, we utilize REINFORCE [Williams, 1992] to train π_θ with stochastic policy gradient ascent. REINFORCE is a natural choice for training ANYCSP since its complexity does not depend on the size of the action space $\mathscr{A}$. Soft assignments allow us to efficiently sample the next assignment $\alpha \sim \varphi$ and recover its probability $\mathbf{P}(\alpha|\varphi) = \prod_X \varphi(\alpha(X))$. These are the only operations on the action space required for REINFORCE. Note that we use vanilla REINFORCE without a baseline or critic network and we sample a single trajectory for every training instance. We found this simple version of the algorithm to be surprisingly robust and effective in our setting. Details on how the policy gradients are computed are provided in Appendix A.

4.3 Implementation and Hyperparameters

We implement ANYCSP in PyTorch. The code for relabeling $G(\mathscr{I}, \alpha^{(t)})$ in each iteration t is also fully based on PyTorch and is GPU-compatible. We implement generalized sparse matrix multiplication in the COO format in CUDA. This helps to increase the memory efficiency and speed of the message passes between values and constraints.

We choose a hidden dimension of $d = 128$ for all experiments. We train with the Adam optimizer for 500K training steps with a batch size of 25. Training a model takes between 24 and 48 hours, depending on the data. During training, we set the upper number of iterations to $T_{\text{train}} = 40$. During testing, we usually run ANYCSP with a timeout rather than a fixed upper number of iterations T. All hyperparameters are provided in Appendix A.

For each training distribution Ω we implement data loaders that sample new instances on-the-fly in each training step. With our hyperparameters we therefore train each model on 12.5 Million sampled training instances. We use fixed subsets of 200 instances sampled from each distribution before training as validation data. The exact generation procedures for each training distribution are provided in Appendix B.

METHOD	$\text{COL}_{<10}$	$\text{COL}_{\geq 10}$
RUNCSP	33	-
CoSoCo	49	33
PICAT	49	38
GREEDY	16	15
DSATUR	38	28
HYBRIDEA	**50**	**40**
ANYCSP	**50**	**40**

Table 1: Results on structured Graph Coloring instances. We provide the number of instances solved with a 20 Minute timeout for both splits, each containing 50 instances with chromatic number less than 10 and at least 10, respectively.

5 Experiments

We evaluate ANYCSP on a wide range of well-known CSPs: Boolean satisfiability (3-SAT) and its maximisation version (MAX-k-SAT for $k = 3, 4, 5$), graph colorability (k-COL), maximum cut (MAXCUT) as well as random CSPs (generated by the so-called MODEL RB). These problems are of high theoretical and practical importance and are commonly used to benchmark CSP heuristics. We train one ANYCSP model for each of these problems using randomly generated instances. Recall that the process of learning problem-specific heuristics with ANYCSP is purely data-driven as our architecture is generic and can take any CSP instance as input. We cross-compare all models on each others CSPs in Appendix B.6.

Here, we will compare the performance of ANYCSP to classical solvers and heuristics as well as previous neural approaches. When applicable, we also tune the configuration of the classical algorithms on our validation data to ensure a fair comparison. All neural approaches run with one NVIDIA Quadro RTX A6000 GPU with 48GB of memory. All classical approaches run on an Intel Xeon Platinum 8160 CPU (2.1 GHz) and 64GB of RAM.

MODEL RB. First, we evaluate ANYCSP on general CSP benchmark instances generated by the MODEL RB [Xu and Li, 2003]. Our training distribution Ω_{RB} consists of randomly generated MODEL RB instances with 30 variables and arity 2. The test dataset RB50 contains 50 satisfiable instances obtained from the XCSP project [Audemard *et al.*, 2020]. These instances each contain 50 variables, domains with 23 values and roughly 500 constraints. They are commonly used as part of the XCSP Competition to evaluate state-of-the-art CSP solvers. Note that the hardness of MODEL RB problems comes from the dense, random constraint relations chosen at the threshold of satisfiability and even instances with 50 variables are very challenging. We will compare ANYCSP to three state-of-the-art CSP solvers: Picat [Zhou, 2022], ACE [Lecoutre, 2022] and CoSoCo [Audemard, 2018]. Picat is a SAT-based solver while ACE and CoSoCo are based on constraint propagation. Picat in particular is the winner of the most recent XCSP Competition [Audemard *et al.*, 2022]. No prior neural baseline exists for this problem.

Figure 3 provides a the results on the RB50 dataset. All algorithms run once on each instance with a 20 Minute timeout. ANYCSP solves the most instances by a substantial margin.

METHOD	$\|V\|$=800	$\|V\|$=1K	$\|V\|$=2K	$|V|\geq$3K
GREEDY	411.44	359.11	737.00	774.25
SDP	245.44	229.22	-	-
RUNCSP	185.89	156.56	357.33	401.00
ECO-DQN	65.11	54.67	157.00	428.25
ECORD	8.67	8.78	39.22	187.75
ANYCSP	**1.22**	**2.44**	**13.11**	**51.63**

Table 2: MAXCUT results on Gset graphs. The graphs are grouped by their vertex counts and we provide the mean deviation from the best known cut size.

METHOD	SL50	SL100	SL150	SL200	SL250
RLSAT	**100**	87	67	27	12
PDP	93	79	72	57	61
WALKSAT	**100**	**100**	97	93	87
PROBSAT	**100**	**100**	97	87	92
ANYCSP	**100**	**100**	**100**	**97**	**99**

Table 3: Number of solved 3-SAT benchmark instances from SATLIB. For each number of variables there are 100 satisfiable test instances.

The second strongest approach is the CoSoCo solver which solves 34 instances in total, 8 less than ANYCSP. Within the timeout of 20 Minutes, ANYCSP will perform 500K search iterations. Recall that we set $T_{\text{train}} = 40$. Therefore, the learned policy generalizes to searches that are over 10K times longer than those seen during training.

Graph Coloring. We consider the problem of finding a conflict-free vertex coloring given a graph G and number of colors k. The corresponding CSP instance has variables for each vertex, domains containing the k colors and one binary "$\neq$"-constraint for each edge. We train on a distribution Ω_{COL} of graph coloring instances for random graphs with 50 vertices. We mix Erdős-Rényi, Barabási-Albert and random geometric graphs in equal parts. The number of colors is chosen to be in $[3, 10]$. As test instances we use 100 structured benchmark graphs with known chromatic number $\mathcal{X}(G)$. The instances are obtained from a collection of hard coloring instances commonly used to benchmark heuristics[4]. They are highly structured and come from a wide range of synthetic and real problems. We divide the test graphs into two sets with 50 graphs each: $\text{COL}_{<10}$ contains graphs with $\mathcal{X}(G) < 10$ and $\text{COL}_{\geq 10}$ contains graphs with $\mathcal{X}(G) \geq 10$. The graphs in $\text{COL}_{\geq 10}$ have up to 1K vertices, 19K edges and a chromatic number of up to 73. This experiment tests generalization to larger domains and more complex structures.

We compare the performance to three problem specific heuristics: a simple greedy algorithm, the classic heuristic DSATUR [Brélaz, 1979] and the state-of-the-art heuristic HybridEA [Galinier and Hao, 1999], all implemented efficiently by [Lewis *et al.*, 2012; Lewis, 2015]. We also evaluate the best two CSP solvers from the MODEL RB experiment. The neural baseline RUNCSP is also tested on $\text{COL}_{<10}$. Unlike ANYCSP, RUNCSP requires us to fix a domain size before training. Therefore, we must train one RUNCSP model for each tested chromatic number $4 \leq \mathcal{X}(G) \leq 9$ and omit testing on $\text{COL}_{\geq 10}$. We use the same training data as [Tönshoff *et al.*, 2021] for their experiments on structured coloring benchmarks.

Table 1 provides the number of solved k-COL instances from both splits. ANYCSP is on par with HybridEA which solves the most instances of all baselines. RUNCSP solves significantly fewer instances than ANYCSP on $\text{COL}_{<10}$ and outperforms only the simple greedy approach. ANYCSP solves 40 out of the 50 instances in $\text{COL}_{\geq 10}$. The optimally colored graphs include the largest instance with 73 colors. Since ANYCSP trains with 3 to 10 colors the trained model is able to generalize to significantly larger domains.

MAXCUT. For MAXCUT we train on the distribution Ω_{MCUT} of random unweighted Erdős-Rényi graphs with 100 vertices and an edge probability $p \in [0.05, 0.3]$. Our test data is Gset [Ye, 2003], a collection of commonly used MAXCUT benchmarks of varying structure with 800 to 10K vertices. We evaluate three neural baselines: RUNCSP, ECO-DQN [Barrett *et al.*, 2020] and ECORD [Barrett *et al.*, 2022]. RUNCSP is also trained on Ω_{MCUT}. We train and validate ECO-DQN and ECORD models with the same data that [Barrett *et al.*, 2022] used for their Gset experiments. We omit S2V-DQN [Khalil *et al.*, 2017] since ECO-DQN and ECORD have been shown to yield substantially better cuts. We adopt the evaluation setup of ECORD and run the neural methods with 20 parallel runs and a timeout of 180s on all unweighted instances of Gset. The results of a standard greedy construction algorithm and the well-known SDP based approximation algorithm by [Goemans and Williamson, 1995] are also included as classical baselines. Both are implemented by [Mehta, 2019]. SDP runs with a 3 hour timeout for graphs with up to 1K vertices.

Table 2 provides results for Gset. We divide the test graphs into groups by the number of vertices (8-9 graphs per group) and report the mean deviation from the best-known cuts obtained by [Benlic and Hao, 2013] for each method. ANYCSP outperforms all baselines across all graph sizes by a large margin. Recall that RUNCSP trains on a soft relaxation of MAXCUT while ECO-DQN and ECORD are both neural local search approaches. Neither concept matches the results of our global search approach trained with policy gradients.

3-SAT. For 3-SAT we choose the training distribution Ω_{3SAT} as uniform random 3-SAT instances with 100 variables. The ratio of clauses to variables is drawn uniformly from the interval $[4, 5]$. For 3-SAT we test on commonly used benchmark instances for uniform 3-SAT from SATLIB[5]. The test set SLN contains 100 instances with $N \in \{50, 100, 150, 200, 250\}$ variables each. The density of these formulas is at the threshold of satisfiability. We evaluate two neural baselines: RLSAT [Yolcu and Póczos, 2019] and PDP [Amizadeh *et al.*, 2019]. PDP is also trained on Ω_{3SAT}. We train RLSAT with the curriculum learning dataset for 3-SAT provided by its authors. We also adopt the experimental setup of RLSAT, which limits the evaluation run by the number of search steps instead of a

[4]https://sites.google.com/site/graphcoloring/vertex-coloring

[5]https://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

METHOD	3CNF	4CNF	5CNF
WALKSAT	2145.28	1556.68	1685.10
CCLS	1567.24	1323.14	1315.96
SATLIKE	1595.86	1188.56	1152.88
ANYCSP	**1537.46**	**1126.44**	**1105.62**

Table 4: Results on Max-k-SAT instances with 10K variables. For each $k \in \{3, 4, 5\}$ we provide the mean number of unsatisfied clauses over 50 random instances.

timeout. The provided code for both PDP and RLSAT is comparatively slow and a timeout would compare implementation details rather than the capability of the learned algorithms. We also evaluate two conventional local search heuristics: The classical WalkSAT algorithm [Selman *et al.*, 1993] based on random walks and a modern probabilistic approach called probSAT [Balint and Schöning, 2018]. Like [Yolcu and Póczos, 2019], we apply stochastic boosting and run each method 10 times for 10K steps on every instance. PDP is deterministic and only applied once to each formula. Note that our aim is to compare the performance of ANYCSP to prior end-2-end neural approaches and conventional stochastic search algorithms for SAT. We note that ANYCSP can not currently compete with CDCL solvers on structured decision SAT instances, like most known stochastic search heuristics.

Table 3 provides the number of solved instances for each tested size. All compared approaches do reasonably well on small instances with 50 variables. However, the performance of the two neural baselines drops significantly as the number of variables increases. ANYCSP does not suffer from this issue and even outperforms the classical local search algorithms on the three largest instance sizes considered here.

MAX-k-SAT. We train on the distribution Ω_{MSAT} of uniform random MAX-k-SAT instances with 100 variables and $k \in \{3, 4\}$. Here, the clause/variable ratio is chosen from $[5, 8]$ and $[10, 16]$ for $k = 3$ and $k = 4$, respectively. These formulas are denser than those of Ω_{3SAT} since we aim to train for the maximization task. Our test data for MAX-k-SAT consists of uniform random k-CNF formulas generated by us. For each $k \in \{3, 4, 5\}$ we generate 50 instances with 10K variables each. The number of clauses is chosen as 75K for $k = 3$, 150K for $k = 4$ and 300K for $k = 5$. These formulas are therefore 100 times larger than the training data and aim to test the generalization to significantly larger instances as well as unseen arities, since $k = 5$ is not used for training. Neural baselines for SAT focus primarily on decision problems. For MAX-k-SAT we therefore compare ANYCSP only to conventional search heuristics: the classical (Max-)WalkSAT [Selman *et al.*, 1993] and two state-of-the-art MAX-SAT local search heuristics CCLS [Luo *et al.*, 2015] and SATLike [Cai and Lei, 2020]. Table 4 provides a comparison. We provide the mean number of unsatisfied clauses after processing each instance with a 20 Minute timeout. Remarkably, ANYCSP outperforms all classical baselines by a significant margin. We note that for each tested arity ANYCSP finds the best solution on all 50 test instances.

We point out that the conventional search heuristics all perform over 100M search steps in the 20 Minute timeout. ANYCSP performs less than 100K steps on each instance in this experiment. The GNN cannot match the speed with which classical algorithms iterate, even though it is accelerated by a GPU. Despite this, ANYCSP consistently finds the best solutions. Figure 4 evaluates this surprising observation further. We plot the number of unsatisfied clauses in the best found solution against the search step in which the solution was found (Steps to Opt.) for all methods and all instances of our MAX-5-SAT test data. We also provide the results of a modified ANYCSP version (ANYCSP Local defined in Appendix C) that is only allowed to change one variable at a time and is therefore a local search heuristic. Note that the x-axis is logarithmic as there is a clear dichotomy separating neural and classical approaches: Compared to conventional heuristics ANYCSP performs roughly three orders of magnitude fewer search steps in the same amount of time. When restricted to local search, ANYCSP is unable to overcome this deficit and yields worse results than strong heuristics such as SATLike. However, when ANYCSP leverages global search to parallelize refinements across the whole instance it can find solutions in 100K steps that elude state-of-the-art local search heuristics after well over 100M iterations.

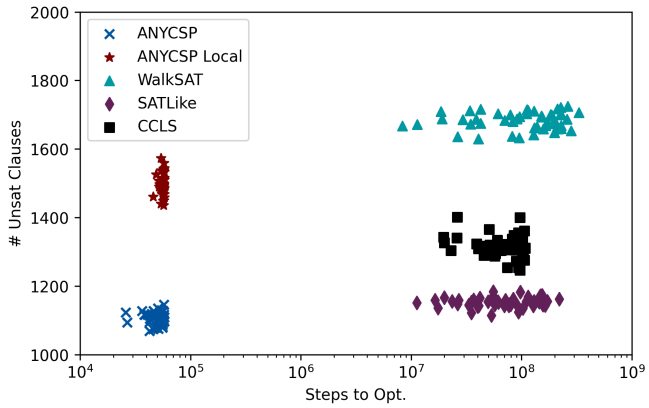

Figure 4: Detailed results for MAX-5-SAT. For each test instance and each method we plot the number of unsatisfied clauses in the best found solution against the search step in which it was found.

6 Conclusion

We have introduced ANYCSP, a novel method for neural combinatorial optimization to learn heuristics for any CSP through a purely data-driven process. Our experiments demonstrate how the generic architecture of our method can learn effective search algorithms for a wide range of problems. We also observe that standard policy gradient descent methods like REINFORCE are capable of learning on an exponentially sized action space to obtain global search heuristics for NP-hard problems. This is a critical advantage when processing large problem instances.

Directions for future work include widening the scope of the architecture even further: Weighted and partial CSPs are a natural extension of the CSP formalism and could be incorporated through node features and adjustments to the reward.

References

[Amizadeh *et al.*, 2019] Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Pdp: A general neural framework for learning constraint satisfaction solvers. *arXiv preprint arXiv:1903.01969*, 2019.

[Audemard *et al.*, 2020] Gilles Audemard, Frédéric Boussemart, Christophe Lecoutre, Cédric Piette, and Olivier Roussel. Xcsp3 and its ecosystem. *Constraints*, 25(1):47–69, 2020.

[Audemard *et al.*, 2022] Gilles Audemard, Christophe Lecoutre, and Emmanuel Lonca, editors. *XCSP3 Competition 2022 Proceedings*, XCSP3 Competition, Artois, France, 2022.

[Audemard, 2018] Gilles Audemard. Cosoco 1.12. In *XCSP3 Competition 2018 Proceedings*, XCSP3 Competition, pages 78–79, 2018.

[Balint and Schöning, 2018] Adrian Balint and Uwe Schöning. probsat. In *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Department of Computer Science, University of Helsinki, 2018.

[Barrett *et al.*, 2020] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3243–3250, 2020.

[Barrett *et al.*, 2022] Thomas D Barrett, Christopher WF Parsonson, and Alexandre Laterre. Learning to solve combinatorial graph partitioning problems via efficient exploration. *arXiv preprint arXiv:2205.14105*, 2022.

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

[Benlic and Hao, 2013] Una Benlic and Jin-Kao Hao. Breakout local search for the max-cutproblem. *Engineering Applications of Artificial Intelligence*, 26(3):1162 – 1173, 2013.

[Biere *et al.*, 2021] A. Biere, M. Heule, H. Van Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2nd edition, 2021.

[Brélaz, 1979] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[Cai and Lei, 2020] Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.

[Cappart *et al.*, 2021] Quentin Cappart, Didier Chételat, Elias B. Khalil, Andrea Lodi, Christopher Morris, and Petar Veličković. Combinatorial optimization and reasoning with graph neural networks. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4348–4355. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Survey Track.

[Galinier and Hao, 1999] Philippe Galinier and Jin-Kao Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of combinatorial optimization*, 3(4):379–397, 1999.

[Gasse *et al.*, 2019] Maxime Gasse, Didier Chetelat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[Goemans and Williamson, 1995] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[Joshi *et al.*, 2020] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent, and Xavier Bresson. Learning TSP requires rethinking generalization. *CoRR*, abs/2006.07054, 2020.

[Khalil *et al.*, 2017] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

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

[Lecoutre, 2022] Christophe Lecoutre. Ace a generic constraint solver. In *XCSP3 Competition 2022 Proceedings*, XCSP3 Competition, pages 58–59, 2022.

[Lewis *et al.*, 2012] Rhyd Lewis, Jonathan Thompson, Christine Mumford, and Jonathan Gillard. A wide-ranging computational comparison of high-performance graph colouring algorithms. *Computers & Operations Research*, 39(9):1933–1950, 2012.

[Lewis, 2015] Rhyd Lewis. *A guide to graph colouring*, volume 7. Springer, 2015.

[Luo *et al.*, 2015] Chuan Luo, Shaowei Cai, Wei Wu, Zhong Jie, and Kaile Su. Ccls: An efficient local search algorithm for weighted maximum satisfiability. *IEEE Transactions on Computers*, 64(7):1830–1843, 2015.

[Mehta, 2019] Hermish Mehta. Cvx graph algorithms. https://github.com/hermish/cvx-graph-algorithms, 2019.

[Russell *et al.*, 2020] S. Russell, S.J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4th edition, 2020.

[Selman *et al.*, 1993] Bart Selman, Henry A Kautz, Bram Cohen, et al. Local search strategies for satisfiability testing. *Cliques, coloring, and satisfiability*, 26:521–532, 1993.

[Selsam *et al.*, 2018] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

[Tönshoff *et al.*, 2021] Jan Tönshoff, Martin Ritzert, Hinrikus Wolf, and Martin Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers in Artificial Intelligence*, 3, 2021.

[Vinyals *et al.*, 2015] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[Wang *et al.*, 2021] Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 21453–21466. Curran Associates, Inc., 2021.

[Williams, 1992] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

[Xu and Li, 2003] Ke Xu and Wei Li. Many hard examples in exact phase transitions. *Science Direct Working Paper No S1574-034X (04)*, pages 70228–8, 2003.

[Yao *et al.*, 2021] Fan Yao, Renqin Cai, and Hongning Wang. Reversible action design for combinatorial optimization with reinforcement learning. In *AAAI-22 Workshop on Machine Learning for Operations Research (ML4OR)*, 2021.

[Ye, 2003] Yinyu Ye. Gset. https://web.stanford.edu/~yyye/yyye/Gset/, 2003.

[Yolcu and Póczos, 2019] Emre Yolcu and Barnabás Póczos. Learning local search heuristics for boolean satisfiability. *Advances in Neural Information Processing Systems*, 32, 2019.

[Zhang *et al.*, 2020] Wenjie Zhang, Zeyu Sun, Qihao Zhu, Ge Li, Shaowei Cai, Yingfei Xiong, and Lu Zhang. Nlocalsat: Boosting local search with solution prediction. *arXiv preprint arXiv:2001.09398*, 2020.

[Zhou, 2022] Neng-Fa Zhou. An xcsp3 solver in picat. In *XCSP3 Competition 2022 Proceedings*, XCSP3 Competition, pages 79–81, 2022.