frontiers in Artificial Intelligence

ORIGINAL RESEARCH
published: 25 February 2021
doi: 10.3389/frai.2020.580607

Graph Neural Networks for Maximum Constraint Satisfaction

Jan Tönshoff, Martin Ritzert, Hinrikus Wolf and Martin Grohe*

Chair of Computer Science 7 (Logic and Theory of Discrete Systems), Department of Computer Science, RWTH Aachen University, Aachen, Germany

Many combinatorial optimization problems can be phrased in the language of constraint satisfaction problems. We introduce a graph neural network architecture for solving such optimization problems. The architecture is generic; it works for all binary constraint satisfaction problems. Training is unsupervised, and it is sufficient to train on relatively small instances; the resulting networks perform well on much larger instances (at least 10-times larger). We experimentally evaluate our approach for a variety of problems, including Maximum Cut and Maximum Independent Set. Despite being generic, we show that our approach matches or surpasses most greedy and semi-definite programming based algorithms and sometimes even outperforms state-of-the-art heuristics for the specific problems.

Keywords: graph neural networks, combinatorial optimization, unsupervised learning, constraint satisfaction problem, graph problems, constraint maximization

OPEN ACCESS

Edited by:
Sriraam Natarajan,
The University of Texas at Dallas,
United States

Reviewed by:
Mayukh Das,
Samsung (India), India
Ugur Kursuncu,
University of South Carolina,
United States

****Correspondence:***
Jan Tönshoff
toenshoff@
informatik.rwth-aachen.de

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: *06 July 2020*
Accepted: *29 October 2020*
Published: *25 February 2021*

Citation:
Tönshoff J, Ritzert M, Wolf H and Grohe M (2021) Graph Neural Networks for Maximum Constraint Satisfaction. Front. Artif. Intell. 3:580607. doi: 10.3389/frai.2020.580607

1 INTRODUCTION

Constraint satisfaction is a general framework for casting combinatorial search and optimization problems; many well-known NP-complete problems, for example, k-colorability, Boolean satisfiability and maximum cut can be modeled as constraint satisfaction problems (CSPs). Our focus is on the optimization version of constraint satisfaction, usually referred to as maximum constraint satisfaction (MAX-CSP), where the objective is to satisfy as many constraints of a given instance as possible. There is a long tradition of designing exact and heuristic algorithms for all kinds of CSPs. Our work should be seen in the context of a recently renewed interest in heuristics for NP-hard combinatorial problems based on neural networks, mostly GNNs (for example, Khalil et al., 2017; Selsam et al., 2019; Lemos et al., 2019; Prates et al., 2019).

We present a generic graph neural network (GNN) based architecture called RUN-CSP (Recurrent Unsupervised Neural Network for Constraint Satisfaction Problems) with the following key features:

Unsupervised: Training is unsupervised and just requires a set of instances of the problem.

Scalable: Networks trained on small instances achieve good results on much larger inputs.

Generic: The architecture is generic and can learn to find approximate solutions for any binary MAX-CSP.

We remark that in principle, every CSP can be transformed into an equivalent binary CSP (see **Section 2** for a discussion).

To solve MAX-CSPs, we train a GNN, which we view as a message passing protocol. The protocol is executed on a graph with nodes for all variables and edges for all constraints of the instance. After running the protocol for a fixed number of rounds, we extract probabilities for the possible values of each variable from its current state. All parameters determining the messages, the update of the internal states, and the readout function are learned. Since these parameters are shared over all

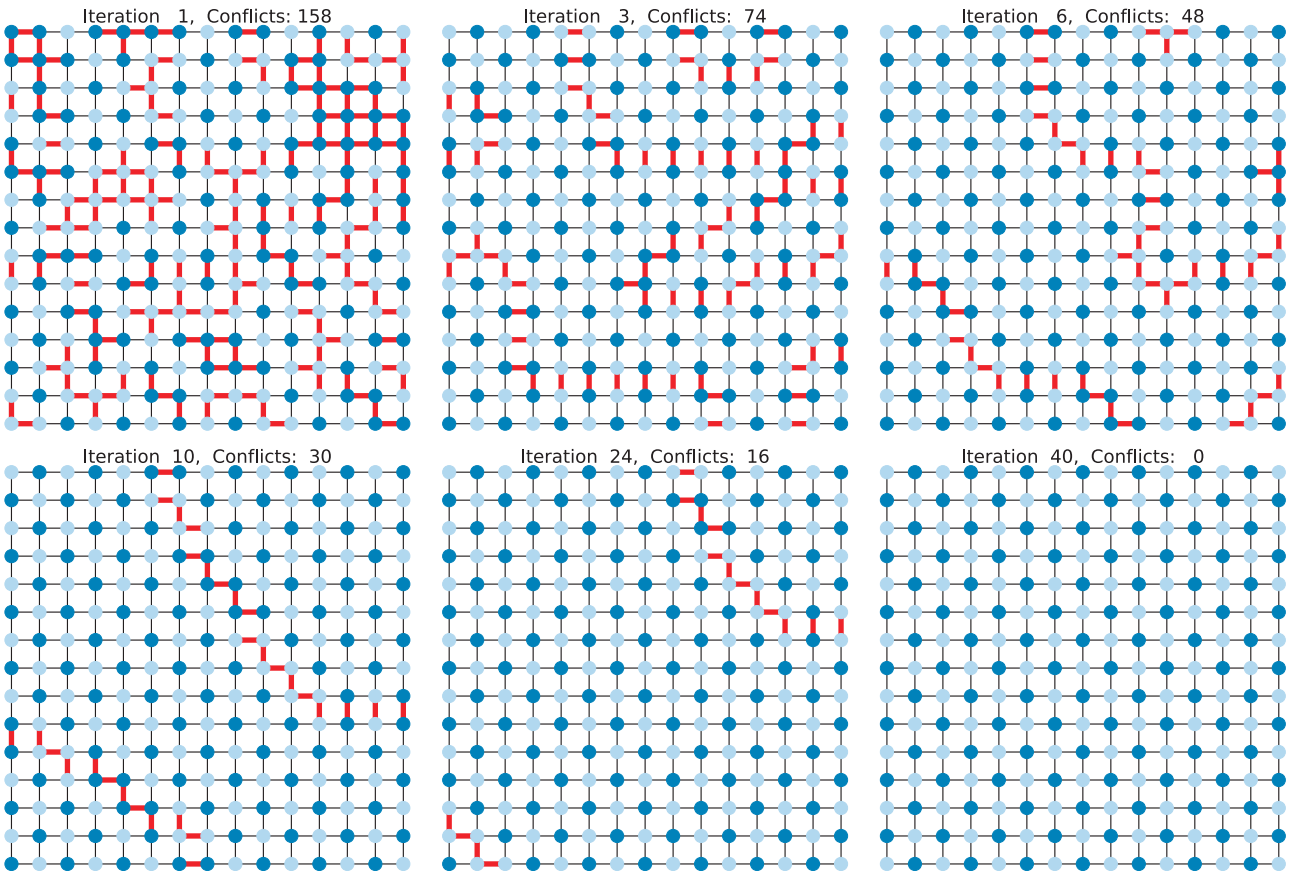

FIGURE 1 | A 2-coloring for a grid graph found by RUN-CSP in 40 iterations. Conflicting edges are shown in red.

variables, we can apply the model to instances of arbitrary size[1]. Our loss function rewards solutions with many satisfied constraints. Thus, our networks learn to satisfy the maximum number of constraints which naturally puts the focus on the optimization version MAX-CSP of the constraint satisfaction problem.

This focus on the optimization problem allows us to train unsupervised, which is a major point of distinction between our work and recent neural approaches to Boolean satisfiability (Selsam et al., 2019) and the coloring problem (Lemos et al., 2019). Both approaches require supervised training and output a prediction for satisfiability or coloring number. Furthermore, our approach not only returns a prediction whether the input instance is satisfiable, but it returns an (approximately optimal) variable assignment. The variable assignment is directly produced by a neural network, which distinguishes our end-to-end approach from methods that combine neural networks with conventional heuristics, such as Khalil et al., (2017) and Li et al., (2018).

We experimentally evaluate our approach on the following NP-hard problems: the maximum 2-satisfiability problem (MAX-2-SAT), which asks for an assignment maximizing the number of satisfied clauses for a given Boolean formula in 2-conjunctive normal form; the maximum cut problem (MAX-CUT), which asks for a partition of a graph in two parts such that the number of edges between the parts is maximal (see **Figure 1**); the 3-colorability problem (3-COL), which asks for a 3-coloring of the vertices of a given graph such that the two endvertices of each edge have distinct colors. We also consider the maximum independent set problem (MAX-IS), which asks for an independent set of maximum cardinality in a given graph. Strictly speaking, MAX-IS is not a maximum constraint satisfaction problem, because its objective is not to maximize the number of satisfied constraints, but to satisfy all constraints while maximizing the number of variables with a certain value. We include this problem to demonstrate that our approach can easily be adapted to such related problems.

Our experiments show that our approach works well for all four problems and matches competitive baselines. Since our approach is generic for all MAX-CSPs, those baselines include other general approaches such as greedy algorithms and semi-definite programming (SDP). The latter is particularly relevant, because it is known (under certain complexity theoretic

[1]Our Tensorflow implementation of RUN-CSP is available at https://github.com/toenshoff/RUN-CSP.

assumptions) that SDP achieves optimal approximation ratios for all Max-CSPs (Raghavendra, 2008). For Max-2-SAT, our approach even manages to surpass a state-of-the-art heuristic. In general, our method is not competitive with the highly specialized state-of-the-art heuristics. However, we demonstrate that our approach clearly improves on the state-of-the-art for neural methods on small and medium-sized binary CSP instances, while still being completely generic. We remark that our approach does not give any guarantees, as opposed to some traditional solvers which guarantee that no better solution exists.

Almost all models are trained on quite small training sets consisting of small random instances. We evaluate those models on unstructured random instances as well as more structured benchmark instances. Instance sizes vary from small instances with 100 variables and 200 constraints to medium sized instances with more than 1,000 variables and over 10,000 constraints. We observe that RUN-CSP is able to generalize well from small instances to instances both smaller and much larger. The largest (benchmark) instance we evaluate on has approximately 120,000 constraints, but that instance required the use of large training graphs. Computations with RUN-CSP are very fast in comparison to many heuristics and profit from modern hardware like GPUs. For medium-sized instances with 10,000 constraints inference takes less than 5 s.

1.1 Related Work

Traditional methods for solving CSPs include combinatorial constraint propagation algorithms, logic programming techniques and domain specific approaches, for an overview see Apt (2003), Dechter (2003). Our experimental baselines include a wide range of classical algorithms, mostly designed for specific problems. For Max-2-SAT, we compare the performance to that of *WalkSAT* (Selman et al., 1993; Kautz, 2019), which is a popular stochastic local search heuristic for Max-SAT. Furthermore, we use the state-of-the-art Max-SAT solver *Loandra* (Berg et al., 2019), which combines linear search and core-guided algorithms. On the Max-Cut problem, we compare our method to multiple implementations of a heuristic approach by Goemans and Williamson (1995). This method is based on semi-definite programming (SDP) and is particularly popular since it has a proven approximation ratio of $\alpha \simeq 0.878$. Other Max-Cut baselines utilize extremal optimization (Boettcher and Percus, 2001) and local search (Benlic and Hao, 2013). For Max-3-Col, we measure the results against *HybridEA* (Galinier and Hao, 1999; Lewis et al., 2012; Lewis, 2015), which is an evolutionary algorithm with state-of-the-art performance. Furthermore, a simple greedy coloring heuristic (Brélaz, 1979) is also used as a comparison. *ReduMIS* is a state-of-the-art Max-IS solver that combines kernelization techniques and evolutionary algorithms. We use it as a Max-IS baseline, together with a simple greedy algorithm.

Beyond these traditional approaches there have been several attempts to apply neural networks to NP-hard problems and more specifically CSPs. An early group of papers dates back to the 1980s and uses Hopfield Networks (Hopfield and Tank, 1985) to approximate TSP and other discrete problems using neural networks. Hopfield and Tank use a single-layer neural network with sigmoid activation and apply gradient descent to come up with an approximative solution. The loss function adopts soft assignments and uses the length of the TSP tour and a term penalizing incorrect tours as loss, hence being unsupervised. This approach has been extended to k-colorability (Dahl, 1987; Takefuji and Lee, 1991; Gassen and Carothers, 1993; Harmanani et al., 2010) and other CSPs (Adorf and Johnston, 1990). The loss functions used in some of these approaches are similar to ours.

Newer approaches involve modern machine learning techniques and are usually based on GNNs. NeuroSAT (Selsam et al., 2019), a learned message passing network for predicting satisfiability, reignited the interest in solving NP-complete problems with neural networks. Prates et al., (2019) use GNNs to learn TSP and trained on instances of the form $(G, \ell \pm \varepsilon)$ where ℓ is the length of an optimal tour on G. They achieved good results on graphs with up to 40 nodes. Using the same idea, Lemos et al., (2019) learned to predict k-colorability of graphs scaling to larger graphs and chromatic numbers than seen during training. Yao et al., (2019) evaluated the performance of unsupervised GNNs for the Max-Cut problem. They adapted a GNN architecture by Chen et al., (2019) to Max-Cut and trained two versions of their network, one through policy gradient descent and the other via a differentiable relaxation of the loss function which both achieved similar results. Amizadeh et al., (2019) proposed an unsupervised architecture for Circuit-SAT, which predicts satisfying variable assignments for a given formula. Khalil et al., (2017) proposed an approach for combinatorial graph problems that combines reinforcement learning and greedy search. They iteratively construct solutions by greedily adding nodes according to estimated scores. The scores are computed by a neural network, which is trained through Q-Learning. They test their method on the MVC, Max-Cut, and TSP problems, where they outperform traditional heuristics across several benchmark instances. For the #P-hard weighted model counting problem for DNF formulas, Abboud et al., (2019) applied a GNN-based message passing approach. Finally, Li et al., (2018) use a GNN to guide a tree search for Max-IS.

2 CONSTRAINT SATISFACTION PROBLEMS

Formally, a *CSP-instance* is a triple $I = (X, D, C)$, where X is a set of variables, D is a domain, and C is a set of constraints of the form $(x_1, \ldots, x_\ell, R)$ for some $R \subseteq D^\ell$. A *constraint language* is a finite set Γ of relations over some fixed domain D, and I is a Γ-instance if $R \in \Gamma$ for all constraints $(x_1, \ldots, x_\ell, R) \in C$. An *assignment* $\alpha : X \to D$ satisfies a constraint $(x_1, \ldots, x_\ell, R)$ if $(\alpha(x_1), \ldots, \alpha(x_\ell)) \in R$, and it satisfies the instance I if it satisfies all constraints in C. CSP(Γ) is the problem of deciding whether a given Γ-instance has a satisfying assignment and finding such an assignment if there is one. Max−CSP(Γ) is the problem of finding an assignment that satisfies the maximum number of constraints.

For example, an instance of 3-COL has a variable x_v for each vertex v of the input graph, domain $D = \{1, 2, 3\}$, and a constraint $(v, w, R^3_{\neq})$ for each edge vw of the graph. Here, $R^3_{\neq} = \{(1, 2), (2, 1), (1, 3), (3, 1), (2, 3), (3, 2)\}$ is the inequality relation on $\{1, 2, 3\}$. Thus 3-COL is $\mathrm{CSP}(\{R^3_{\neq}\})$.

In this paper, we only consider *binary CSPs*, that is, CSPs whose constraint language only contains unary and binary relations. From a theoretical perspective, this is no real restriction, because it is well known that every CSP can be transformed into an "equivalent" binary CSP (see Dechter, 2003). Let us review the construction. Suppose we have a constraint language Γ of maximum arity $k \geq 3$ over some domain D. We construct a binary constraint language $\hat{\Gamma}$ as follows. The domain $\hat{D}$ of $\hat{\Gamma}$ consists of all elements of D as well as all pairs $(\boldsymbol{a}, R)$ where $R \in \Gamma$ and $\boldsymbol{a}$ is a tuple occurring in R. For every $R \in \Gamma$, we add a unary relation Q_R consisting of all pairs $(\boldsymbol{a}, R) \in \hat{D}$ where $\boldsymbol{a} \in R$. Moreover, for $1 \leq i \leq k$ we add a binary "projection" relation P_i consisting of all pairs $((\boldsymbol{a}, R), a_i)$ for $R \in \Gamma$, say of arity $\ell \leq k$, and $\boldsymbol{a} = (a_1, \ldots, a_\ell) \in R$. Finally, for every instance $I = (X, D, C)$ of $\mathrm{CSP}(\Gamma)$ we construct an instance $\hat{I} = (\hat{X}, \hat{D}, \hat{C})$ of $\mathrm{CSP}(\hat{\Gamma})$, where $\hat{X}$ consists of all variables in X and a new variable y_c for every constraint $c = (x_1, \ldots, x_\ell, R) \in C$ and $\hat{C}$ consists of a *tuple constraint* (y_c, Q_R) and *projection constraints* (y_c, x_i, P_i) for all $1 \leq i \leq \ell \leq k$. Here, the tuple constraints select for every constraint $c = (\overline{x}, R) \in C$ a tuple $a \in R$ and the projection constraints ensure a consistent assignment to the original variables $X \subseteq \hat{X}$. Then the instances I and $\hat{I}$ are equivalent in the sense that I is satisfiable if and only if $\hat{I}$ is and there is a one-to-one correspondence between the satisfying assignments.

However, the construction is not approximation preserving. For example, it is not the case that an assignment satisfying 90% of the constraints of $\hat{I}$ yields an assignment satisfying 90% of the constraints of I. It is possible to fix this by adding weights to the constraints, making it more expensive to violate projection constraints. Moreover, and arguably more importantly in this context, it is not clear how well our method works on CSPs of higher arity when translated to binary CSPs using this construction. We leave a thorough experimental evaluation of CSPs with higher arities for future work.

3 METHOD

3.1 Architecture

We use a randomized recurrent GNN architecture to evaluate a given problem instance using message passing. For any binary constraint language Γ a RUN-CSP network can be trained to approximate $\textsc{Max}\text{-}\mathrm{CSP}(\Gamma)$. Intuitively, our network can be viewed as a trainable communication protocol through which the variables of a given instance can negotiate a value assignment. With every variable $x \in X$ we associate a short-term state $s_x^{(t)} \in \mathbb{R}^k$ and a hidden (long-term) state $h_x^{(t)} \in \mathbb{R}^k$ which change throughout the message passing iterations $t \in \{0, \ldots, t_{\max}\}$. The short-term state vector $s_x^{(0)}$ for every variable x is initialized by sampling each value independently from a normal distribution with zero mean and unit variance. All hidden states $h_x^{(0)}$ are initialized as zero vectors.

Every message passing step uses the same weights and thus we are free to choose the number $t_{\max} \in \mathbb{N}$ of iterations for which RUN-CSP runs on a given problem instance. This number may or may not be identical to the number of iterations used for training. The state size k and the number of iterations used for training $t^{\mathrm{tr}}_{\max}$ and evaluation $t^{\mathrm{ev}}_{\max}$ are the main hyperparameters of our network.

Variables x and y that co-occur in a constraint $c = (x, y, R)$ can exchange messages. Each message depends on the states $s_x^{(t)}, s_y^{(t)}$, the relation R, and the order of x and y in the constraint but not on the internal long-term states $h_x^{(t)}, h_y^{(t)}$. The dependence on R implies that we have independent message generation functions for every relation R in the constraint language Γ. The process of message passing and updating the internal states is repeated $t_{\max}$ times. We use linear functions to compute the messages as preliminary experiments showed that more complicated functions did not improve performance while being less stable and less efficient during training. Thus, the messaging function for every relation R is defined by a trainable weight matrix $M_R \in \mathbb{R}^{2k \times 2k}$ as

$$S_R\left(s_x^{(t)}, s_y^{(t)}\right) = M_R \begin{pmatrix} s_x^{(t)} \\ s_y^{(t)} \end{pmatrix}. \tag{1}$$

The output of S_R consists of two stacked k-dimensional vectors, which represent the messages to x and y, respectively. Note that the generated messages depend on the order of the variables in the constraint. This behavior is desirable for asymmetric relations. For symmetric relations we modify S_R to produce messages independently from the order of variables in c. In this case we use a smaller weight matrix $M_R \in \mathbb{R}^{k \times 2k}$ to generate both messages. Note that the two messages can still be different, but the content of each message depends only on the states of the endpoints.

The internal states h_x and s_x are updated by an LSTM cell based on the mean of the received messages. For a variable x which received the messages $m_1, \ldots, m_\ell$ the new states are thus computed by

$$h_x^{(t+1)}, s_x^{(t+1)} = \mathrm{LSTM}\left(h_x^{(t)}, s_x^{(t)}, \frac{1}{\ell}\sum_{i=1}^{\ell} m_i\right). \tag{2}$$

For every variable x and iteration $t \in \{1, \ldots, t_{\max}\}$, the network produces a soft assignment $\varphi^{(t)}(x)$ from the state $s_x^{(t)}$. In our architecture we use $\varphi^{(t)}(x) = \mathrm{softmax}\left(W s_x^{(t)}\right)$ with $W \in \mathbb{R}^{d \times k}$ trainable and $d = |D|$ (domain size of the CSP). In φ, the linear function reduces the dimensionality while the softmax function enforces stochasticity. The soft assignments $\varphi^{(t)}(x)$ can be interpreted as probabilities of a variable x receiving a certain value $v \in D$. If the domain D contains only two values, we compute a "probability" $p^{(t)}(x) = \sigma\left(W s_x^{(t)}\right)$ for each node with $W \in \mathbb{R}^{1 \times k}$. The soft assignment is then given by $\varphi^{(t)}(x) = \left(p^{(t)}(x), 1 - p^{(t)}(x)\right)$. To obtain a hard variable assignment $\alpha^{(t)} : X \to D$, we assign the value with the highest estimated probability in $\varphi^{(t)}(x)$ for each variable $x \in X$. From the hard assignments $\alpha^{(1)}, \ldots, \alpha^{(t^{\mathrm{ev}}_{\max})}$, we select the one with the most satisfied constraints as the final prediction of the network. This is not necessarily the last assignment $\alpha^{(t^{\mathrm{ev}}_{\max})}$.

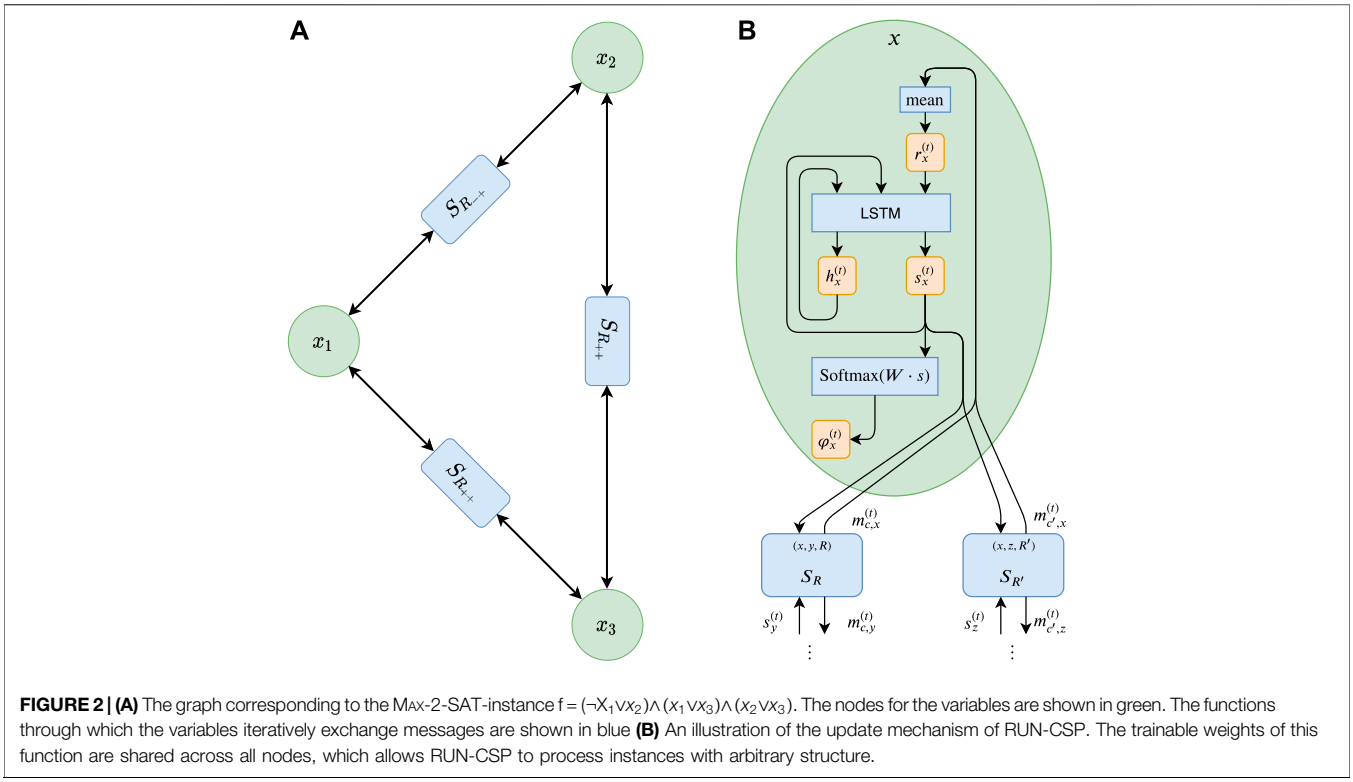

FIGURE 2 | (A) The graph corresponding to the MAX-2-SAT-instance $f = (\neg X_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)$. The nodes for the variables are shown in green. The functions through which the variables iteratively exchange messages are shown in blue **(B)** An illustration of the update mechanism of RUN-CSP. The trainable weights of this function are shared across all nodes, which allows RUN-CSP to process instances with arbitrary structure.

Input: Instance (X, C), $t_{\max} \in \mathbb{N}$.
Output: $(\varphi^{(1)}, \ldots, \varphi^{(t_{\max})})$, $\varphi^{(t)} : X \to [0,1]^d$.
for $x \in X$ **do**.
//random initialization

$$s_x^{(0)} \sim \mathcal{N}(0,1)^k$$
$$h_x^{(0)} := 0 \in \mathbb{R}^k$$

for $t \in \{1, \ldots, t_{\max}\}$ **do**.
for $c := (x, y, R) \in C$ **do**.
//generate messages

$$\left(m_{c,x}^{(t)}, m_{c,y}^{(t)}\right) := S_R\left(s_x^{(t-1)}, s_y^{(t-1)}\right)$$

for $x \in X$ **do**.
//combine messages and update

$$r_x^{(t)} := \frac{1}{\deg(x)} \sum_{c \in C, x \in c} m_{c,x}^{(t)}$$
$$\left(h_x^{(t)}, s_x^{(t)}\right) := \mathrm{LSTM}\left(h_x^{(t-1)}, s_x^{(t-1)}, r_x^{(t)}\right)$$
$$\varphi^{(t)}(x) := \mathrm{softmax}\left(W \cdot s_x^{(t)}\right)$$

Algorithm 1: Network Architecture.

Algorithm 1 specifies the architecture in pseudocode. **Figure 2** illustrates the message passing graph for a MAX-2-SAT instance and the internal update procedure of RUN-CSP. Note that the network's output depends on the random initialization of the short-term states $s_x^{(0)}$. Those states are the basis for all messages sent during inference and thus for the solution found by RUN-CSP. By applying the network multiple times to the same input and choosing the best solution, we can therefore boost the performance.

We did evaluate more complex variants of this architecture with multi-layered messaging functions and multiple stacked recurrent cells. No increase in performance was observed with these modifications, while the running time increased. Replacing the LSTM cells with GRU cells slightly decreased the performance. Therefore, we use the simple LSTM-based architecture presented here.

3.2 Loss Function

In the following we derive our loss function used for unsupervised training. Let $I = (X, D, C)$ be a CSP-instance. Assume without loss of generality that $D = \{1, \ldots, d\}$ for a positive integer d. Given I, in every iteration our network will produce a soft variable assignment $\varphi : X \to [0,1]^d$, where $\varphi(x)$ is stochastic for every $x \in X$. Instead of choosing the value with the maximum probability in $\varphi(x)$, we could obtain a hard assignment $\alpha : X \to D$ by independently sampling a value for each $x \in X$ from the distribution specified by $\varphi(x)$. In this case, the probability that any given constraint $(x, y, R) \in C$ is satisfied by α can be expressed by

$$\Pr_{\alpha \sim \varphi}\left[(\alpha(x), \alpha(y)) \in R\right] = \varphi(x)^T A_R \varphi(y) \tag{3}$$

where $A_R \in \{0,1\}^{d \times d}$ is the characteristic matrix of the relation R with $(A_R)_{i,j} = 1 \Leftrightarrow (i,j) \in R$. We then aim to minimize the combined negative log-likelihood over all constraints:

$$\mathcal{L}_{\mathrm{CSP}}(\varphi, I) := \frac{1}{|C|} \cdot \sum_{(x,y,R) \in C} -\log\left(\varphi(x)^T A_R \varphi(y)\right) \tag{4}$$

We combine the loss function $\mathcal{L}_{\text{CSP}}$ throughout all iterations with a discount factor $\lambda \in [0, 1]$ to get our training objective:

$$\mathcal{L}\left(\{\varphi_t\}_{t \le t^{\text{tr}}_{\text{max}}}, I\right) := \sum_{t=1}^{t^{\text{tr}}_{\text{max}}} \lambda^{t^{\text{tr}}_{\text{max}} - t} \cdot \mathcal{L}_{\text{CSP}}\left(\varphi^{(t)}, I\right) \tag{5}$$

This loss function allows us to train unsupervised since it does not depend on any ground truth assignments. Furthermore, it avoids reinforcement learning, which is computationally expensive. In general, computing optimal solutions for supervised training can easily turn out to be prohibitive; our approach completely avoids such computations.

We remark that it is also possible to extend the framework to weighted Max-CSPs where a real weight is associated with each constraint. To achieve this, we can replace the averages in the loss function and message collection steps by weighted averages. Negative constraint weights can be incorporated by swapping the relation with its complement. We demonstrate this in **Section 4.2** where we evaluate RUN-CSP on the weighted Max-Cut problem.

4 EXPERIMENTS

To validate our method empirically, we performed experiments for Max-2-SAT, Max-Cut, 3-COL and Max-IS. For all experiments, we used internal states of size $k = 128$; state sizes up to $k = 1024$ did not increase performance for the tested instances. We empirically chose to use $t^{\text{tr}}_{\text{max}} = 30$ iterations during training and, unless stated otherwise, $t^{\text{ev}}_{\text{max}} = 100$ for evaluation. Especially for larger instances it proved beneficial to use a relatively high $t^{\text{ev}}_{\text{max}}$. In contrast, choosing $t^{\text{tr}}_{\text{max}}$ too large during training ($t^{\text{tr}}_{\text{max}} > 50$) resulted in unstable training. During evaluation, we use 64 parallel runs for each instance and use the best result. Further increasing this number mainly increases the runtime but has no real effect on the quality of solutions. We trained most models with 4,000 instances split into in 400 batches. Training is performed for 25 epochs using the Adam optimizer with default parameters and gradient clipping at a norm of 1.0. The decay over time in our loss function was set to $\lambda = 0.95$. We provide a more detailed overview of our implementation and training configuration in the **Supplementary Material**.

We ran our experiments on machines with two Intel Xeon 8160 CPUs and one NVIDIA Tesla V100 GPU but got very similar runtime on consumer hardware. Evaluating 64 runs on an instance with 1,000 variables and 1,000 constraints takes about 1.5 s, 10,000 constraints about 5 s, and 20,000 constraints about 8 s. Training a model takes less than 30 min. Thus, the computational cost of RUN-CSP is relatively low.

4.1 Maximum 2-Satisfiability

We view Max-2-SAT as a binary CSP with domain $D = \{0, 1\}$ and a constraint language consisting of three relations R_{00} (for clauses with two negated literals), R_{01} (one negated literal) and R_{11} (no negated literals). For example, $R_{01} = \{(0, 0), (0, 1), (1, 1)\}$ is the set of satisfying assignments for a clause $(\neg x \vee y)$. For training a RUN-CSP model we used 4,000 random 2-CNF formulas with 100 variables each. The number of clauses was sampled uniformly between 100 and 600 for every formula and each clause was generated by sampling two distinct variables and then independently negating the literals with probability 0.5.

4.1.1 Random Instances

For the evaluation of RUN-CSP in Max-2-SAT we start with random instances and compare it to a number of problem-specific heuristics. All baselines can solve Max-SAT for arbitrary arities, not only Max-2-SAT, while RUN-CSP can solve a variety of binary Max-CSPs. The state-of-the-art Max-SAT Solver *Loandra* (Berg et al., 2019) won the unweighted track for incomplete solvers in the Max-SAT Evaluation 2019 (Bacchus et al., 2019). We ran Loandra in its default configuration with a timeout of 20 min on each formula. To put this into context, on the largest evaluation instance used here (9,600 constraints) RUN-CSP takes less than 7 min on a single CPU core and about 5 s using the GPU. *WalkSAT* (Selman et al., 1993; Kautz, 2019) is a stochastic local search algorithm for approximating Max-Sat. We allowed WalkSAT to perform 10 million flips on each formula using its "noise" strategy with parameters $n = 2$ and $m = 2000$. Its performance was boosted similarly to RUN-CSP by performing 64 runs and selecting the best result.

For evaluation we generated random formulas with 100, 400, 800, and 1,600 variables. The ratio between clauses and variables was varied in steps of 0.1 from 1 to 6. **Figure 3** shows the average percentage of satisfied clauses in the solutions found by each method over 100 formulas for each size and density. The methods yield virtually identical results for formulas with less than 2 clauses per variable. For denser instances, RUN-CSP yields slightly worse results than both baselines when only 100 variables are present. However, RUN-CSP matches the results of Loandra for formulas with 400 variables and outperforms it for instances with 800 and 1,600 variables. The performance of WalkSAT degrades on these formulas and is significantly worse than RUN-CSP.

4.1.2 Benchmark Instances

For more structured formulas, we use Max-2-SAT benchmark instances from the unweighted track of the Max-SAT Evaluation 2016 (Argelich, 2016) based on the Ising spin glass problem (De Simone et al., 1995; Heras et al., 2008). We used the same general setup as in the previous experiment but increased the timeout for Loandra to 60 min. In particular we use the same RUN-CSP model trained entirely on random formulas. **Table 1** contains the achieved numbers of unsatisfied constraints across the benchmark instances. All methods produced optimal results on the first and the third instance. RUN-CSP slightly deviates from the optimum on the second instance. For the fourth instance RUN-CSP found an optimal solution while both WalkSAT and Loandra did not. On the largest benchmark formula, RUN-CSP again produced the best result.

Thus, RUN-CSP is competitive for random as well as spin-glass-based structured Max-2-SAT instances. Especially on larger instances it also outperforms conventional methods. Furthermore, training on random instances generalized well to the structured spin-glass instances.

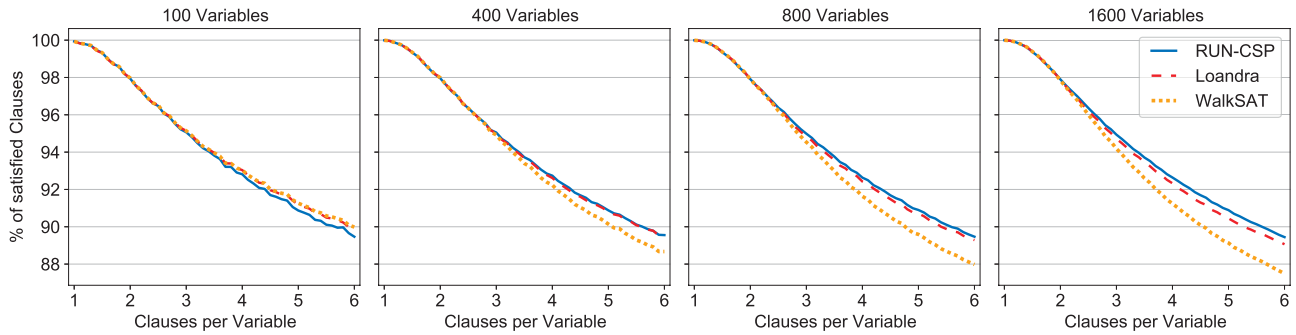

FIGURE 3 | Percentage of satisfied clauses of random 2-CNF formulas for RUN-CSP, Loandra and WalkSAT. Each data point is the average of 100 formulas; the ratio of clauses per variable increases in steps of 0.1.

TABLE 1 | Max-2-SAT: Number of unsatisfied constraints for Max-2-SAT benchmark instances derived from the Ising spin glass problem.

Instance	|V|	|C|	Opt	RUN-CSP	WalkSAT	Loandra
t3pm3	27	162	17	17	17	17
t4pm3	64	384	38	40	38	38
t5pm3	125	750	78	78	78	78
t6pm3	216	1,269	136	136	142	142
t7pm3	343	2,058	209	216	227	225

TABLE 2 | Max-Cut: *P*-values of graph cuts produced by RUN-CSP, Yao, SDP, and EO for regular graphs with 500 nodes and varying degrees. We report the mean across 1,000 random graphs for each degree.

d	RUN-CSP	Yao Rel	Yao Pol	SDP	EO
3	0.714	0.707	0.693	0.702	0.727
5	0.726	0.701	0.668	0.690	0.737
10	0.710	0.670	0.599	0.682	0.735
15	0.697	0.607	0.629	0.678	0.736
20	0.685	0.614	0.626	0.674	0.732

4.2 Max Cut

Max-Cut is a classical Max-CSP with domain $D = \{0, 1\}$ and only one relation $R_{\neq} = \{(0, 1), (1, 0)\}$ used in the constraints.

4.2.1 Regular Graphs

In this section we evaluate RUN-CSP's performance on this problem. Yao et al., (2019) proposed two unsupervised GNN architectures for Max-Cut. One was trained through policy gradient descent on a non-differentiable loss function while the other used a differentiable relaxation of this loss. They evaluated their architectures on random regular graphs, where the asymptotic Max-Cut optimum is known. We use their results as well as their baseline results for Extremal Optimization (EO) (Boettcher and Percus, 2001) and a classical approach based on semi-definite programming (SDP) (Goemans and Williamson, 1995) as baselines for RUN-CSP. To evaluate the sizes of graph cuts, Yao et al., (2019) introduced a relative performance measure called *P-value* given by $P(z) = \frac{z/n - d/4}{\sqrt{d/4}}$ where z is the predicted cut size for a d-regular graph with n nodes. Based on results of Dembo et al., (2017), they showed that the expected P-value of d-regular graphs approaches $P^* \approx 0.7632$ as $n \to \infty$. P-values close to P^* indicate a cut where the size is close to the expected optimum and larger values are better. While Yao et al. trained one instance of their GNN for each tested degree, we trained one network model on 4,000 Erdős–Rényi graphs and applied it to all graphs. For training, each graph had a node count of $n = 100$ and a uniformly sampled number of edges $m \sim U(100, 2000)$. Thus, the model was not trained specifically for regular graphs. **Table 2** reports the mean P-values across 1,000 random regular graphs with 500 nodes for different degrees. For every method other than RUN-CSP, we provide the values as reported by Yao et al. While RUN-CSP does not match the cut sizes produced by extremal optimization, it clearly outperforms both versions of the GNN as well as the classical SDP-based approach.

4.2.2 Benchmark Instances

We performed additional experiments on standard Max-Cut benchmark instances. The Gset dataset (Ye, 2003) is a set of 71 weighted and unweighted graphs that are commonly used for testing Max-Cut algorithms. The dataset contains three different types of random graphs. Those graphs are Erdős–Rényi graphs with uniform edge probability, graphs where the connectivity gradually decays from node 1 to n, and 4-regular toroidal graphs. Here, we use two unweighted graphs for each type from this dataset. We reused the RUN-CSP model from the previous experiment but increased the number of iterations for evaluation to $t^{\text{ev}}_{\text{max}} = 500$. Our first baseline by Choi and Ye (2000) uses an SDP solver based on dual scaling (DSDP) and a reduction based on the approach of Goemans and Williamson (1995). Our second baseline Breakout Local Search (BLS) is based on the combination of local search and adaptive perturbation (Benlic and Hao, 2013). Its results are among the best known solutions for the Gset dataset. For DSDP and BLS we report the values as provided in the literature. **Table 3** reports the achieved cut sizes for RUN-CSP, DSDP, and BLS. On G14 and G15, which are random graphs with decaying node degree, the graph cuts produced by RUN-CSP are similar in size to those reported for

TABLE 3 | Max-Cut: Achieved cut sizes on Gset instances for RUN-CSP, DSDP, and BLS.

Graph	|V|	|E|	RUN-CSP	DSDP	BLS
G14	800	4,694	2,943	2,922	3,064
G15	800	4,661	2,928	2,938	3,050
G22	2,000	19,990	13,028	12,960	13,359
G49	3,000	6,000	6,000	6,000	6,000
G50	3,000	6,000	5,880	5,880	5,880
G55	5,000	12,468	10,116	9,960	10,294

DSDP. For the Erdős–Rényi graphs G22 and G55 RUN-CSP performs better than DSDP but worse than BLS. Lastly, on the toroidal graphs G49 and G50 all three methods achieved the best known cut size. This reaffirms the observation that our architecture works particularly well for regular graphs. Although RUN-CSP did not outperform the state-of-the-art heuristic in this experiment it performed at least as well as the SDP based approach DSDP.

4.2.3 Weighted Maximum Cut Problem

Additionally, we evaluate RUN-CSP on the weighted Max-Cut problem, where every edge $e \in E$ has an associated weight $w_e \in \{1, -1\}$. The aim is to maximize the objective:

$$\Theta(S, T) = \sum_{e \in E \cap (S \times T)} w_e,$$

where the partition S, T of V defines a cut. We can apply RUN-CSP to this problem by training a model for the constraint language $\Gamma = \{R_=, R_\neq\}$ over the domain $D = \{0, 1\}$. Here, $R_=$ and $R_\neq$ are the equality and inequality relations, respectively. We model every positive edge as a constraint with $R_\neq$ and every negative edge with $R_=$. We trained a RUN-CSP network on 4,000 random Erdős–Rényi graphs with $n = 100$ nodes and $m \sim U(100, 300)$ edges. The weights $w_e \sim \{1, -1\}$ were drawn uniformly for each edge.

We evaluate this model on 10 benchmark instances obtained from the Optsicom Project[2], namely the 10 smallest graphs of set 2. These instances are based on the lsing spin glass problem and are commonly used to evaluate heuristics empirically. All 10 graphs have $n = 125$ nodes and $m = 375$ edges. Khalil et al., (2017) utilize reinforcement learning to guide greedy search heuristics for combinatorial problems including weighted Max-Cut. They evaluated their method on the same benchmark instances for weighted Max-Cut and compared the performance to a classical greedy heuristic (Kleinberg and Tardos, 2006) and an SDP-based method (Goemans and Williamson, 1995). Furthermore, they approximated the optimal values by running CPLEX for 1 h on every instance. We use their reported results and baselines for a comparison with RUN-CSP. Crucially, Khalil et al., (2017) trained their network on random variations of the benchmark instances, while RUN-CSP was trained on purely random data. **Table 4** provides the achieved cut sizes. On all but one benchmark instance RUN-CSP yields the largest cuts and on five out of 10 instances it even found the optimal cut value. The classical approaches based on Greedy Search and SDP performed substantially worse than both neural methods.

TABLE 4 | Max-Cut: Achieved cut sizes on Optsicom Benchmarks. The optimal values were estimated by Khalil et al., (2017) by running CPLEX for 1 h on each instance.

Graphs	Opt	RUN-CSP	Khalil et al	Greedy	SDP
G54100	110	110	108	80	54
G54200	112	112	108	90	58
G54300	106	106	104	86	60
G54400	114	112	108	96	56
G54500	112	112	112	94	56
G54600	110	110	110	88	66
G54700	112	110	108	88	60
G54800	108	106	108	76	54
G54900	110	108	108	88	68
G541000	112	110	108	80	54
Approx. Ratio	1.0	1.01	1.02	1.28	1.90

4.3 Coloring

Within coloring we focus on the case of three colors, i.e., we consider CSPs over the domain $\{1, 2, 3\}$ with the inequality relation $R_\neq$. In general, RUN-CSP aims to satisfy as many constraints as possible and therefore approximates Max-3-Col. Instead of evaluating on Max-3-Col, we evaluate on its practically more relevant decision variant 3-COL which asks whether a given graph is 3-colorable without conflicts. We turn RUN-CSP into a classifier by predicting that a given input graph is 3-colorable if and only if it is able to find a conflict-free vertex coloring.

4.3.1 Hard Instances

We evaluate RUN-CSP on so-called "hard" random instances, similar to those defined by Lemos et al., (2019). These instances are a special subclass of Erdős–Rényi graphs where an additional edge can make the graph no longer 3-colorable. We describe our exact generation procedure in the **Supplementary Material**. We trained five RUN-CSP models on 4,000 hard 3-colorable instances with 100 nodes each. In **Table 5** we present results for RUN-CSP, a greedy heuristic with DSatur strategy (Brélaz, 1979), and the state-of-the-art heuristic HybridEA (Galinier and Hao, 1999; Lewis et al., 2012; Lewis, 2015). HybridEA was allowed to make 500 million constraint checks on each graph. We observe that larger instances are harder for all tested methods and between the three algorithms there is a clear hierarchy. The state-of-the-art heuristic HybridEA clearly performs best and finds solutions even for some of the largest graphs. RUN-CSP finds optimal colorings for a large fraction of graphs with up to 100 nodes and even a few correct colorings for graphs of size 200. The weakest algorithm is DSatur which even fails on most of the small 50 node graphs and gets rapidly worse for larger instances.

Choosing larger or more training graphs for RUN-CSP did not significantly improve its performance on larger hard graphs. We assume that a combination of increasing the state size, complexity

[2]http://grafo.etsii.urjc.es/optsicom/maxcut/.

TABLE 5 | 3-COL: Percentages of hard 3-colorable instances for which optimal 3-colorings were found by RUN-CSP, Greedy, and HybridEA. We evaluate on 1,000 instances for each size. We provide mean and standard deviation across five different RUN-CSP models.

Nodes	RUN-CSP	Greedy	HybridEA
50	98.4 ± 0.3	34.0	100.0
100	62.5 ± 2.7	6.7	100.0
150	15.5 ± 2.3	1.5	98.7
200	2.6 ± 0.4	0.5	88.9
300	0.1 ± 0.0	0.0	39.9
400	0.0 ± 0.0	0.0	15.3

TABLE 6 | MAX-3-COL: Percentages of unsatisfied constraints for each graph class under the different RUN-CSP models. Values are averaged over 1,000 graphs and the standard deviation is computed with respect to the five RUN-CSP models.

Graphs	M_{ER} (%)	M_{Geo} (%)	M_{Pow} (%)	M_{Reg} (%)	M_{Mix} (%)
Erdos-Renyi	4.75 ± 0.01	4.73 ± 0.02	4.72 ± 0.02	6.69 ± 1.60	4.73 ± 0.01
Geometric	10.33 ± 0.07	10.16 ± 0.04	11.39 ± 0.66	18.99 ± 3.32	10.18 ± 0.03
Pow. Cluster	1.89 ± 0.00	1.96 ± 0.01	1.87 ± 0.00	2.44 ± 0.67	1.89 ± 0.00
Regular	2.33 ± 0.01	2.41 ± 0.03	2.33 ± 0.02	2.32 ± 0.00	2.33 ± 0.00
Mean	4.83 ± 0.02	4.82 ± 0.03	5.08 ± 0.18	7.61 ± 1.40	4.78 ± 0.01

of the message generation functions, and number and size of training instances is able to achieve better results, but on the cost of efficiency.

In **Table 5** we do not report results for GNN-GCP by Lemos et al., (2019) as the structure of the output is fundamentally different. While the three algorithms in **Table 5** output a coloring, GNN-GCP outputs a guess on the chromatic number without providing a proof that this is achievable. We trained instances of GNN-GCP on 32,000 pairs of hard graphs of size 40 to 60 (small) and 50 to 100 (medium). For testing, we restricted the model to only choose between the chromatic numbers 3 and 4, when allowing a wider range of possible values, the accuracy of GNN-GCP drops considerably. The network was able to achieve test accuracies of 75% (respectively 65% when trained and evaluated on medium instances). The model generalizes fairly well, with the small model achieving 64% on the medium test set and the large model achieving 74% on the small test set, almost matching the performance of the network trained on graphs of the respective size. On a set of test instances of hard graphs with 150 nodes, GNN-GCP achieved an accuracy of 52% (54% for the model trained on medium instances). Thus, the model performs significantly worse than RUN-CSP which achieves 81% (GNN-GCP 59%) accuracy on a test set of graphs of size 100, and 68% on graphs of size 150 where GNN-GCP achieves up to 54%. The numbers for RUN-CSP are larger than those reported in **Table 5** since in the table only 3-colorable instances were considered. Here, the accuracy is computed over 3-colorable instances as well as their non-3-colorable counter parts. By design, RUN-CSP achieves perfect classification on negative instances.

Overall, we see that despite being designed for maximization tasks, RUN-CSP outperforms greedy heuristics and neural baselines on the decision variant of 3-COL for hard random instances.

4.3.2 Structure Specific Performance

On the example of the coloring problem, we evaluate generalization to other graph classes. We expect a network trained on instances of a particular structure to adapt toward this class and outperform models trained on different graph classes. We briefly evaluate this hypothesis for four different classes of graphs.

Erdős–Rényi Graphs: Graphs are generated by uniformly sampling m distinct edges between n nodes.

Geometric Graphs: A graph is generated by first assigning random positions within a 1×1 square to n distinct nodes. Then an edge is added for every pair of points with a distance less than r.

Powerlaw-Cluster Graphs: This graph model was introduced by Holme and Kim (2002). Each graph is generated by iteratively adding n nodes and connected to m existing nodes. After each edge is added, a triangle is closed with probability p, i.e., an additional edge is added between the new node and a random neighbor of the other endpoint of the edge.

Regular Graphs: We consider random 5-regular graphs as an example for graphs with a very specific structure.

We trained five RUN-CSP models on 4,000 random instances of each type where each graph had between 50 and 100 nodes. We refer to these groups of models as M_{ER}, M_{Geo}, M_{Pow} and M_{Reg}. Five additional models M_{Mix} were trained on a mixed dataset with 1,000 random instances of each graph class. The exact parameters for generating the graphs can be found in the **Supplementary Material**. Note that the parameters for each class were purposefully chosen such that most graphs are not 3-colorable. This allows us to evaluate the relative performance on the maximization task. **Table 6** contains the percentage of unsatisfied constraints over the models on 1,000 fresh graphs of each class. We observe that all models perform well on the class of structures they were trained on and M_{Reg} yields the worst performance on all other classes. Both M_{Geo} and M_{Pow} outperform M_{ER} on Erdős–Rényi graphs while M_{ER} outperforms M_{Geo} on Powerlaw-Cluster and M_{Pow} on geometric graphs. When averaging over all four classes, M_{Mix} produces the best results, despite not achieving the best results for any particular class. Additionally, we observe a very low variance in performance between the different models trained on the same dataset. Only the models trained on relatively narrow graph classes, namely regular graphs and to some extent also Powerlaw-Cluster graphs, exhibit a higher variance.

Overall, this demonstrates that training on locally diverse graphs (e.g., geometric graphs or a mixture of graph classes) leads to good generalization toward other graph classes. While all tested networks achieved competitive results on the structure that they were trained on, they were not always the best for that particular structure. Therefore, our original hypothesis appears to be overly simplistic and restricting the training data to the structure of the evaluation instances is not necessarily optimal.

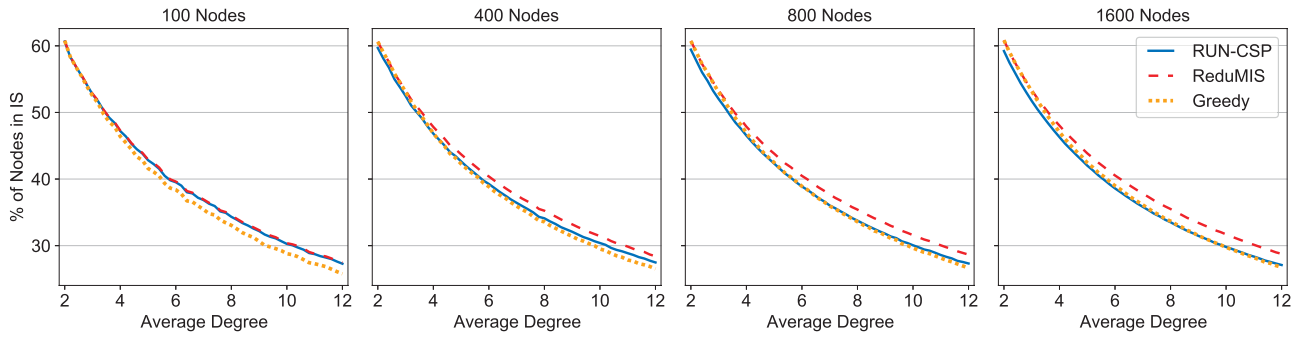

FIGURE 4 | Independent set sizes on random graphs produced by RUN-CSP, ReduMIS and a greedy heuristic. The sizes are given as the percentage of nodes contained in the independent set. Every data point is the average for 100 graphs; the degree increases in steps of 0.2.

4.4 Independent Set

Finally, we experimented with the maximum independent set problem MAX-IS. The independence condition can be modeled through a constraint language Γ_{IS} with one binary relation $R_{IS} = \{(0,0), (0,1), (1,0)\}$. Here, assigning the value 1 to a variable is interpreted as including the corresponding node in the independent set. MAX-IS is *not* simply MAX-CSP(Γ_{IS}), since the empty set will trivially satisfy all constraints. Instead, MAX-IS is the problem of finding an assignment which satisfies R_{IS} at all edges while maximizing an additional objective function that measures the size of the independent set. To model this in our framework, we extend the loss function to reward assignments with many variables set to 1. For a graph $G = (V, E)$ and a soft assignment $\varphi : V \to [0, 1]$, we define

$$\begin{aligned} \mathcal{L}_{\text{MIS}}(\varphi, G) &= \big(\kappa + \mathcal{L}_{\text{CSP}}(\varphi, G)\big) \cdot \big(1 + \mathcal{L}_{\text{size}}(\varphi, G)\big), \\ \mathcal{L}_{\text{size}}(\varphi, G) &= \frac{1}{|V|} \sum_{v \in V} \big(1 - \varphi(v)\big). \end{aligned} \tag{6}$$

Here, $\mathcal{L}_{\text{CSP}}$ is the standard RUN-CSP loss for Γ_{IS} and κ adjusts the relative importance of $\mathcal{L}_{\text{CSP}}$ and $\mathcal{L}_{\text{size}}$. Intuitively, smaller values for κ decrease the importance of $\mathcal{L}_{\text{size}}$ which favors larger independent sets. A naive weighted sum of both terms turned out to be unstable during training and yielded poor results, whereas the product in **Eq. 6** worked well. For training, $\mathcal{L}_{\text{MIS}}$ is combined across iterations with a discount factor λ as in the standard RUN-CSP architecture.

4.4.1 Random Instances

We start by evaluating the performance on random graphs. We trained a network on 4,000 random Erdős–Rényi graphs with 100 nodes and $m \sim U(100, 600)$ edges each and with $\kappa = 1$. For evaluation we use random graphs with 100, 400, 800 and 1,600 nodes and a varying number of edges. For roughly 6% of all predictions, the predicted set contained induced edges (just a single edge in most cases), meaning the predicted sets where not independent. We corrected these predictions by removing one of the endpoints of each induced edge from the set and only report results after this correction. We compare RUN-CSP against two baselines: ReduMIS, a state-of-the-art MAX-IS solver (Akiba and Iwata, 2016; Lamm et al., 2017) and a greedy heuristic, which we implemented ourselves. The greedy procedure iteratively adds the node with lowest degree to the set and removes the node and its neighbors from the graph until the graph is empty. **Figure 4** shows the achieved independent set sizes, each data point is the mean IS size across 100 random graphs. For graphs with 100 nodes, RUN-CSP achieves similar sizes as ReduMIS and clearly outperforms the greedy heuristic. On larger graphs our network produces smaller sets than ReduMIS. However, RUN-CSP's performance remains similar to the greedy baseline and, especially on denser graphs, outperforms it.

4.4.2 Benchmark Instances

For more structured instances, we use a set of benchmark graphs from a collection of hard instances for combinatorial problems (Xu, 2005). The instances are divided into five sets with five graphs each. These graphs were generated through the RB Model (Xu and Li, 2003; Xu et al., 2005), a model for generating hard CSP instances. A graph of the class frb$c$-k consists of c interconnected k-cliques and the MAX-IS has a forced size of c. The previous model trained on Erdős–Rényi graphs did not perform well on these instances and produced sets with many induced edges. Thus, we trained a new network on 2,000 instances we generated ourselves through the RB model. The exact generation procedure of this dataset is provided in the **Supplementary Material**. We set $\kappa = 0.1$ to increase the importance of the independence condition. The predictions of the new model contained no induced edges for all benchmark instances. **Table 7** contains the achieved IS sizes. We observe that RUN-CSP yields similar results to the greedy heuristic. While our network does not match the state-of-the-art heuristic, it beats the greedy approach on large instances with over 100,000 edges.

TABLE 7 | MAX-IS: Achieved IS sizes for the benchmark graphs. We report the mean and std. deviation for the five graphs in each group.

Graphs	|V|	|E|	RUN-CSP	Greedy	ReduMIS
frb30–15	450	18 k	25.8 ± 0.8	24.6 ± 0.5	30 ± 0.0
frb40–19	790	41 k	33.6 ± 0.5	33.0 ± 1.2	39.4 ± 0.5
frb50–23	1,150	80 k	42.2 ± 0.4	42.2 ± 0.8	48.8 ± 0.4
frb59–26	1,478	126 k	49.4 ± 0.5	48.0 ± 0.7	57.4 ± 0.9

5 CONCLUSIONS

We have presented a universal approach for approximating MAX-CSPs with recurrent neural networks. Its key feature is the ability to train without supervision on any available data. Our experiments on the optimization problems MAX-2-SAT, MAX-CUT, 3-COL and MAX-IS show that RUN-CSP produces high quality approximations for all four problems. Our network can compete with traditional approaches like greedy heuristics or semi-definite programming on random data as well as benchmark instances. For MAX-2-SAT, RUN-CSP was able to outperform a state-of-the-art MAX-SAT Solver. Our approach also achieved better results than neural baselines, where those were available. RUN-CSP networks trained on small random instances generalize well to other instances with larger size and different structure. Our approach is very efficient and inference takes only a few seconds, even for larger instances with over 10,000 constraints. The runtime scales linearly in the number of constraints and our approach can fully utilize modern hardware, like GPUs.

Overall, RUN-CSP seems like a promising approach for approximating Max-CSPs with neural networks. The strong results are somewhat surprising, considering that our networks consist of just one LSTM cell and a few linear functions. We believe that our observations point toward a great potential of machine learning in combinatorial optimization.

Future Work

We plan to extend RUN-CSP to CSPs of arbitrary arity and to weighted CSPs. It will be interesting to see, for example, how it performs on 3-SAT and its maximization variant. Another possible future extension could combine RUN-CSP with traditional local search methods, similar to the approach by Li et al., (2018) for MAX-IS. The soft assignments can be used to guide a tree search and the randomness can be exploited to generate a large pool of initial solutions for traditional refinement methods.

DATA AVAILABILITY STATEMENT

The code for RUN-CSP including the generated datasets and their generators can be found on github https://github.com/toenshoff/RUN-CSP. The additional datasets can be downloaded at their sources as specified in the following: Spinglass 2-CNF (Heras et al., 2008), http://maxsat.ia.udl.cat/benchmarks/ (Unweighted Crafted Benchmarks); Gset (Ye, 2003), https://www.cise.ufl.edu/research/sparse/matrices/Gset/; MAX-IS Graphs (Xu, 2005), http://sites.nlsde.buaa.edu.cn/kexu/benchmarks/graph-benchmarks.htm; Optsicom (Corberán et al., 2006), http://grafo.etsii.urjc.es/optsicom/maxcut/.

AUTHOR CONTRIBUTIONS

Starting from an initial idea by JT, all authors contributed to the presented design and the writing of this manuscript. Most of the implementation was done by JT, with help and feedback from MR and HW. The work was supervised by MG.

FUNDING

This work was supported by the German Research Foundation (DFG) under grants GR 1,492/16–1 *Quantitative Reasoning About Database Queries* and GRK 2236 (UnRAVeL).

ACKNOWLEDGMENTS

This work is part of Jan Tönshoff's Master's Thesis and already appeared as a preprint on arXiv (Toenshoff et al., 2019).

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frai.2020.580607/full#supplementary-material.

REFERENCES

Abboud, R., Ceylan, I. I., and Lukasiewicz, T. (2019). Learning to reason: leveraging neural networks for approximate DNF counting. Preprint repository name [Preprint]. Available at:arXiv preprint arXiv:1904.02688 (Accessed June 4, 2019).

Adorf, H.-M., and Johnston, M. D. (1990). "A discrete stochastic neural network algorithm for constraint satisfaction problems," in IJCNN international joint conference on neural networks, San Diego, CA, USA, June 17–21, 1990 (IEEE), 917–924.

Akiba, T., and Iwata, Y. (2016). Branch-and-reduce exponential/FPT algorithms in practice: a case study of vertex cover. *Theor. Comput. Sci.* 609, 211–225. doi:10.1016/j.tcs.2015.09.023

Amizadeh, S., Matusevych, S., and Weimer, M. (2019). "Learning to solve circuit-SAT: an unsupervised differentiable approach," in International conference on learning representations, New Orleans, Louisiana, USA, May 6–9, 2019 (Amherst).

Apt, K. (2003). *Principles of constraint programming*. Amsterdam, NL: Cambridge University Press, 1–17.

Argelich, J. (2016). [Dataset] Eleventh evaluation of max-SAT solvers (Max-SAT-2016).

F. Bacchus, M. Järvisalo, and R. Martins (Editors) (2019). *MaxSAT evaluation 2019: solver and benchmark descriptions*. Helsinki: University of Helsinki, 49.

Benlic, U., and Hao, J.-K. (2013). Breakout local search for the max-cutproblem. *Eng. Appl. Artif. Intell.* 26, 1162–1173. doi:10.1016/j.engappai.2012.09.001

Berg, J., Demirović, E., and Stuckey, P. J. (2019). "Core-boosted linear search for incomplete maxSAT," in International conference on integration of constraint programming, artificial intelligence, and operations research, Thessaloniki, Greece, June 4–7, 2019 (Cham, Switzerland: Springer), 39–56.

Boettcher, S., and Percus, A. G. (2001). Extremal optimization for graph partitioning. *Phys. Rev.* 64, 026114. doi:10.1103/physreve.64.026114

Brélaz, D. (1979). New methods to color the vertices of a graph. *Commun. ACM* 22, 251–256. doi:10.1145/359094.359101

Chen, Z., Li, L., and Bruna, J. (2019). "Supervised community detection with line graph neural networks," in International Conference on Learning Representations, New Orleans, Louisiana, USA, May 6–9, 2019 (Amherst).

Choi, C., and Ye, Y. (2000). *Solving sparse semidefinite programs using the dual scaling algorithm with an iterative solver*. Iowa City, IA: Department of Management Sciences, University of Iowa.

Corberán, Á., Peiró, J., Campos, V., Glover, F., and Martí, R. (2006). *Optsicom project*.

Dahl, E. (1987). "Neural network algorithms for an np-complete problem: map and graph coloring," in Proceedings First International Conference Neural Networks III, (San Diego, NY: IEEE), 113–120.

De Simone, C., Diehl, M., Jünger, M., Mutzel, P., Reinelt, G., and Rinaldi, G. (1995). Exact ground states of Ising spin glasses: new experimental results with a branch-and-cut algorithm. *J. Stat. Phys.* 80, 487–496. doi:10.1007/bf02178370

Dechter, R. (2003). *Constraint processing*. San Mateo, CA: University of Morgan Kaufmann, 344.

Dembo, A., Montanari, A., and Sen, S. (2017). Extremal cuts of sparse random graphs. *Ann. Probab.* 45, 1190–1217. doi:10.1214/15-aop1084

Galinier, P., and Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring. *J. Combin. Optim.* 3, 379–397. doi:10.1023/a:1009823419804

Gassen, D. W., and Carothers, J. D. (1993). "Graph color minimization using neural networks," in Proceedings of 1993 international conference on neural networks, October 25–29, 1993 (Nagoya, Japan: IEEE), 1541–1544.

Goemans, M. X., and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* 42, 1115–1145. doi:10.1145/227683.227684

Harmanani, H., Hannouche, J., and Khoury, N. (2010). A neural networks algorithm for the mi nimum coloring problem using FPGAs†. *Int. J. Model. Simulat.* 30, 506–513. doi:10.1080/02286203.2010.11442597

Heras, F., Larrosa, J., De Givry, S., and Schiex, T. (2008). 2006 and 2007 max-SAT evaluations: contributed instances. *Schweiz. Arch. Tierheilkd.* 4, 239–250. doi:10.3233/sat190046

Holme, P., and Kim, B. J. (2002). Growing scale-free networks with tunable clustering. *Phys. Rev.* 65, 026107. doi:10.1103/physreve.65.026107

Hopfield, J. J., and Tank, D. W. (1985). "Neural" computation of decisions in optimization problems. *Biol. Cybern.* 52, 141–5210. doi:10.1007/BF00339943 PubmedAbstract |

Kautz, S. (2019). [Dataset]. Walksat home page

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). "Learning combinatorial optimization algorithms over graphs," in Advances in neural information processing systems 30: annual conference on neural information processing systems 2017, Long Beach, CA, USA, December 4–9, 2017. 6348–6358. Red Hook (NY): Curran Associates.

Kleinberg, J., and Tardos, E. (2006). *Algorithm design*. Pearson Education India, 864 .

Lamm, S., Sanders, P., Schulz, C., Strash, D., and Werneck, R. F. (2017). Finding near-optimal independent sets at scale. *J. Heuristics* 23, 207–229. doi:10.1007/s10732-017-9337-x

Lemos, H., Prates, M., Avelar, P., and Lamb, L. (2019). *Graph coloring meets deep learning: effective graph neural network models for combinatorial problems*. Preprint repository name [Preprint]. Available at:arXiv preprint arXiv:1903.04598 (Accessed March 11, 2019).

Lewis, R. (2015). *A guide to graph coloring*. Basel: Springer, Vol. 7, 253.

Lewis, R., Thompson, J., Mumford, C., and Gillard, J. (2012). A wide-ranging computational comparison of high-performance graph coloring algorithms. *Comput. Oper. Res.* 39, 1933–1950. doi:10.1016/j.cor.2011.08.010

Li, Z., Chen, Q., and Koltun, V. (2018). "Combinatorial optimization with graph convolutional networks and guided tree search," in Advances in Neural Information Processing Systems, 539–548.

Prates, M., Avelar, P. H. C., Lemos, H., Lamb, L. C., and Vardi, M. Y. (2019). "Learning to solve NP-complete problems: a graph neural network for decision TSP, *Aaai*," in Proceedings of the AAAI conference on artificial intelligence, New York, USA, February 7–12, 2020 (Palo Alto, California USA: AAAI Press), 4731–4738.

Raghavendra, P. (2008). "Optimal algorithms and inapproximability results for every CSP?" in Proceedings of the 40th ACM symposium on theory of computing, New York, USA, May, 2018 (New York, NY: Association for Computing Machinery), 245–254.

Selman, B., Kautz, H. A., and Cohen, B. (1993). Local search strategies for satisfiability testing. *Cliques Coloring Satisfiab.* 26, 521–532. doi:10.1090/dimacs/026/25

Selsam, D., Lamm, M., Bünz, B., Liang, P., de Moura, L., and Dill, D. L. (2019). "Learning a SAT solver from single-bit supervision," in international conference on learning representations, New Orleans, LA, Apr 30, 2019(Amherst).

Takefuji, Y., and Lee, K. C. (1991). Artificial neural networks for four-coloring map problems and k-colorability problems. *IEEE Trans. Circ. Syst.* 38, 326–333. doi:10.1109/31.101328

Toenshoff, J., Ritzert, M., Wolf, H., and Grohe, M. (2019). *Graph neural networks for maximum constraint satisfaction*[Preprint]. arXiv:1909.08387.

Xu, K. (2005). [Dataset] BHOSLIB: benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring). Available at: http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm (Accessed April 20, 2014).

Xu, K., Boussemart, F., Hemery, F., and Lecoutre, C. (2005). "A simple model to generate hard satisfiable instances," in IJCAI-05, Proceedings of the nineteenth international joint Conference on artificial intelligence, Edinburgh, Scotland, UK, July 30–August 5, 2005. 337–342. Denver: Professional Book Center

Xu, K., and Li, W. (2003). Many hard examples in exact phase transitions with application to generating hard satisfiable instances. Preprint repository name [Preprint]. Available at:arXiv preprint cs/0302001 (Accessed November 11, 2003).

Yao, W., Bandeira, A. S., and Villar, S. (2019). *Experimental performance of graph neural networks on random instances of max-cut*. Preprint repository name [Preprint]. Available at:arXiv preprint arXiv:1908.05767 (Accessed August 15, 2019).

Ye, Y. (2003). [Dataset] *The Gset dataset*.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

ACRONYMS

RUN-CSP Recurrent Unsupervised Neural Network for Constraint Satisfaction Problems.
3-COL 3-Coloring Problem.
CSP Constrain Satisfaction Problem.
GNN Graph Neural Network.
MVC Maximum Vertex Cover Problem.
MAX-CUT Maximum Cut Problem.
MAX-2-SAT Maximum Satisfiability Problem for Boolean formulas with two literals per clause.
MAX-3-COL Maximum 3-Coloring Problem.
MAX-IS Maximum Independent Set.
TSP Traveling Sales Person Problem.